NuMicro® USB 2.0全速裝置之實作

Application Note for 32-bit NuMicro® Family

Document Information

摘要	本文介紹如何用NuMicro® USB 2.0全速裝置來實作各種USB類別的應用裝置。
適用範圍	NuMicro® 全系列USB 2.0全速裝置。

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

www.nuvoton.com

Table of Contents

1 概述

本文主要介紹使用新唐的NuMicro® 系列支持的USB2.0全速裝置控制器的韌體程式來開發USB裝置，其韌體程式就是一般的USB控制與狀態處理的部分，方便使用者直接套用，有助於不懂USB規格的使用者也能自行設計USB應用程式並減少USB程式設計的時間。

本文前半部分會簡單介紹USB的基本觀念，USB詳細規格說明請參閱USB Specification Revision 2.0；第二部分將以新唐的 Cortex® -M0 Nano100B 為例，介紹 NuMicro® USB2.0全速裝置控制器的功能與韌體程式的使用方式，最後則是以六個實際的類別應用範例做說明，應用範例有：

- HID Transfer
- HID Keyboard
- HID Mouse
- MSC
- VCOM
- Audio

2 USB簡介

通用序列匯流排(Universal Serial Bus，USB)用來連接電腦系統和外部裝置的一種串列埠匯流排標準，主要將連接至電腦的纜線減至最少，讓許多不同的週邊裝置都可以使用相同的連接線。而連接上電腦時，電腦會自動識別這些週邊裝置，並且搭配適當的驅動程式，無需使用者再另外重新設定。

USB匯流排其最大的特點是支援隨插即用的技術並且支援熱插拔(可以在不需要關閉電源的情況下插入或拔除)，當裝置插入時，主機列舉到此裝置並載入所需的驅動程式。USB的匯流排結構是採用階梯型星型拓撲(Tiered Star Network)結構，它由一個主機控制器和若干通過集線器裝置以樹狀連接的裝置組成。一個控制器下最多可以有5級Hub，包括Hub在內，最多可以連接128個裝置，因為在設計時是使用7位元定址欄位，二的七次方就等於128，若扣掉USB主機預設給第一次接上的週邊裝置使用，還剩127個位址可以使用。因此，一部電腦最多可以連接127個USB裝置。位址0是所有USB裝置的預設位址，當USB裝置接上主機時，其位址就是0，然後主機會盡快分配一個位址給它。USB裝置(USB Devices)指各種類型的USB週邊裝置，它具備某種Function(功能)的裝置；一種是「單一功能」的USB裝置，一個功能佔用一個USB纜線(Cable)和USB埠(Port)，例如滑鼠、鍵盤、RS-232或移動儲存裝置。另一種為「複合裝置」，是一個具有多個功能和配置多個介面的組合裝置，但是只有一個唯一的裝置(Device)地址，透過相對應的USB裝置驅動程式(Device Driver)來與主機通訊，可使用僅有的一條纜線和一個USB埠實現多個功能。

USB裝置定義許多不同的類別(Class)，常見的類別有：USB 通訊裝置類別 (CDC)、人性化介面裝置類別(HID)、大容量儲存裝置類別(MSC)，以及USB音訊裝置類別(UAC)。而依照目前USB產品的規格，可以將USB裝置分為以下三種不同的速度：

- 低速裝置(Low-Speed Devices)：傳輸速率最高為 1.5 Mbps
- 全速裝置(Full-Speed Devices)：傳輸速率最高為 12 Mbps
- 高速裝置(High-Speed Devices)：USB2.0 所提出的新規格，傳輸速率最高為 480 Mbps

每個週邊裝置都具有端點(Endpoint)，而主機與端點的通訊是經由虛擬管線(Virtual Pipe)所構成的。一旦虛擬管線建立好之後，每個端點就會傳回描述此裝置的相關資訊(也就是描述元Descriptor)給主機。USB的端點傳輸共有四種類型：控制(Control)、中斷(Interrupt)、巨量(Bulk)與等時(Isochronous)。

設計USB裝置的應用之前，必須了解USB介面的一些標準與規格，本章節會簡單描述USB的基本知識，詳細的USB規格可參考USB Specification Revision 2.0。

2.1 USB 介面

USB的連接線由+5伏特(VBus)與接地線(GND)加上兩條差動的資料線D+與D-所組成，兩條訊號線使用雙絞線(Twisted Pair)傳輸，以抵消長導線的電磁干擾。D+與D-它們各自使用半雙工的差動訊號，並利用NRZI(Non-Return-To-Zero Inverted)的編碼方式來傳送，來達成高速傳輸的目的。USB藉由兩條訊號線D+與D-的電位表現出不同的狀態，組合這些狀態來產生傳輸資料與多種匯流排狀態，包含閒置(Idle)、暫停(Suspend)、恢復(Resume)、封包開始(Start of Packet)、封

包結束(End of Packet)、切斷連接(Disconnect)、連接(Connect)、重置(Reset)，表 1顯示出不同的信號如何呈現出不同的匯流排狀態。

匯流排狀態	描述
Idle	(D+) - (D-) > 200 mv，此狀態在全速裝置中稱為J State。
Suspend	Idle狀態超過3ms。
Resume	(D-) - (D+) > 200 mv，此狀態在全速裝置中稱為K State。
Start-of-Packet (SOP)	從Idle轉換為Resume。
End-of-Packet (EOP)	D+與D-為低電位維持2bit時間，在第3個bit即轉為Idle。
Single-ended 0 (SE0)	D+與D- < $V_{SE\ (min)}$，$V_{SE\ (min)}$≒0.8v。
Disconnect	SE0超過2.5us。
Connect	D+或D-為高電位超過2.5us。
Reset	D+與D-為低電位超過2.5us。

表 1 USB 訊號分類

USB上的資料傳輸採用NRZI編碼方式來對每個位元個別編碼，其編碼規則為：當資料位元為1時不轉換，資料位元為0時再做轉換。圖 1顯示資料位元組的NRZI編碼範例，圖 1紅線表示資料轉換。

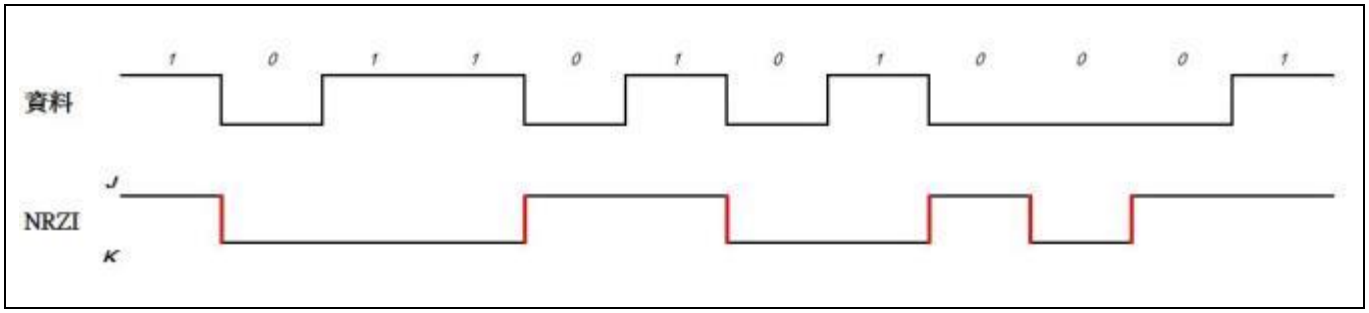


圖 1 NRZI 位元編碼範例

NRZI資料流中的位元轉換能使解碼器維持與進來的資料同步，為了避免重複相同1的信號一直進入時，導致讀取的時序會發生嚴重錯誤，最終將導致接收器失去同步狀態，所以採用位元填塞(Bit Stuffing)。其方法為：資料中含有六個連續的1，就需要在其後面填塞一個0，以確保接收端會定期同步化。圖 2呈現位元填塞的過程。

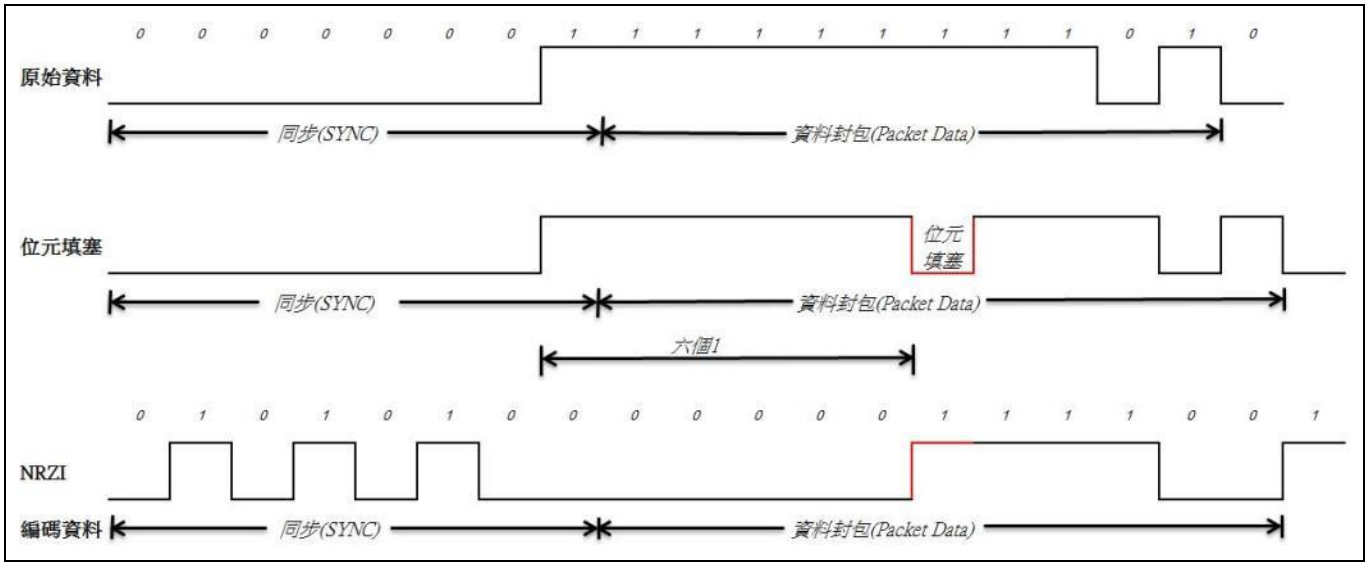


圖 2 NRZI 解碼的位元填塞過程

2.2 USB 通信協定

USB傳輸的控制權是由主機端掌握，不論是控制命令或是資料傳輸時，裝置都只是在回應主機的動作。主機端利用標準的USB通信協定來與USB週邊裝置溝通，USB傳輸(Transfer)由一或多個交易(Transaction)組成，而交易(Transaction)包含了二或三種封包(Packet)，封包(Packet)是組成USB傳輸的最小單位。圖 3說明USB Transfer-Transaction-Packet三者之間的關係。

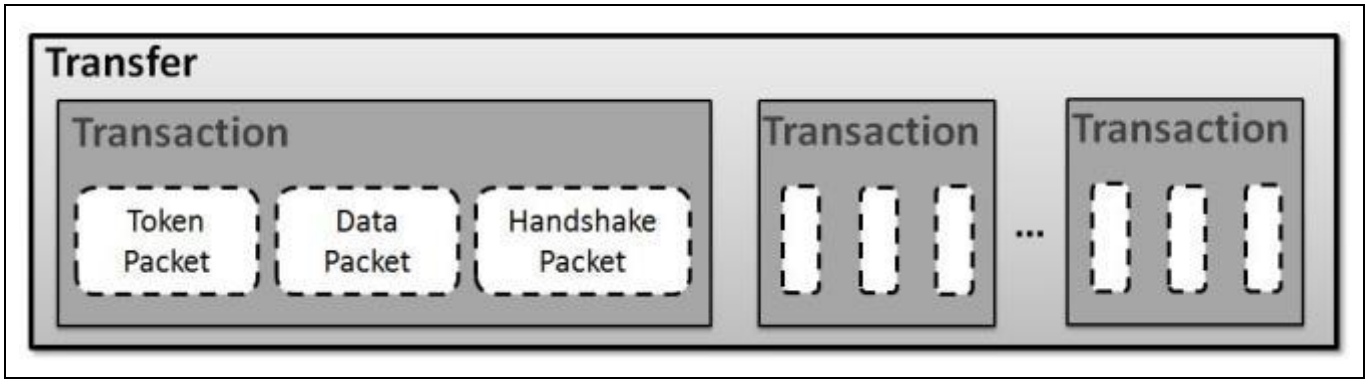


圖 3 USB Transfer-Transaction-Packet 關係圖

2.2.1 USB 封包格式

USB封包是執行所有處理動作的機制，封包格式依照需求會由七個欄位刪減而組合而成，這七個欄位包含：同步(Sync)、PID、位址(Address)、端點(Endpoint)、訊框數目(Frame Number)、資料(Data)、CRC。

封包緊跟在SYNC之後，高速裝置封包的SYNC長度為32bit，全速與低速裝置封包的SYNC長度為8Bit。PID用來定義封包的類型，分為Token、Data、Handshake及Special四種類別，共有16種類型的PID，具體定義請參考表 2。PID的資料長度為8個Bits，只用到前面四個Bit，後面4個Bit其值固定是前面4個Bit的補數，需注意的是PID的格式為LSB在前面MSB在後面。舉例來說，假

設PID的前面4個Bit=1001，那後面4個Bit就是0110，PID合起來就是10010110。其中Special是針對USB2.0特有的PID，包含了PING(檢查端點裝置是否能接收資料)、SPLIT(高頻寬的USB分割傳輸)、PRE(低頻寬的USB前導封包)、ERR(分割傳輸任務錯誤)。對於OUT和SETUP處理，位址與端點欄位用來選擇接收資料的端點；對於IN處理，位址與端點欄位用來選擇傳送資料的端點。最後，每個封包以一個CRC做結尾，它用來確認資料是否正確的傳輸。

所有處理動作都是由一個Token封包開始，主機先傳送Token封包來決定傳輸型態，再依照Token封包裡的資料需求傳送Data封包，此封包會決定想要收的資料類型與資料傳送方向；最後等待接收端回傳Handshake封包確認是否收到，注意Handshake封包與Data封包的傳輸方向相反。傳送方向分為：Host傳送資料至Device稱為OUT；Device送資料回Host稱為IN。

PID類型	PID名稱	PID <bit3:bit0>	描述
Token	OUT	0001B	裝置位址和Endpoint號碼，傳輸方向為主機到裝置。
	IN	1001B	裝置位址和Endpoint號碼，傳輸方向為裝置到主機。
	SOF	0101B	封包內包含一個11Bit的Frame number，用來辨識Frame的起始。
	SETUP	1101B	裝置位址和Endpoint號碼，用於控制型傳輸之Setup階段。
Data	DATA0	0011B	雙數的資料封包。
	DATA1	1011B	單數的資料封包。
	DATA2	0111B	使用在High speed裝置上。
	MDATA	1111B	使用在High speed裝置上。
Handshake	ACK	0010B	確認資料正確無誤的被接收。
	NAK	1010B	資料接收錯誤或忽略資料。
	STALL	1110B	無法完成傳輸，以及需要軟體介入才能使裝置從Stall狀態復原。
	NYET	0110B	使用在High speed裝置上。
Special	PRE	1100B	使用在Low speed裝置上。

	ERR	1100B	使用在High speed裝置上。
	SPLIT	1000B	使用在High speed裝置上。
	PING	0100B	使用在High speed裝置上。
	Reserved	0000B	保留。

表 2 PID 類型

2.2.1.1 Token封包

Token封包的欄位包含SYNC、PID、ADDR、ENDP、Frame Number與CRC5。PID用來識別OUT、IN、SOF與SETUP封包；ADDR欄位由7Bits所組成，可用來定址127個週邊裝置，每個裝置僅能對應一個唯一位址；處理OUT和SETUP封包時，位址與端點欄位用來選擇接收資料的端點；對於IN處理，位址與端點欄位用來選擇傳送資料的端點；Frame Number欄位用於SOF；CRC欄位用來做資料錯誤檢測，圖 4 及圖 5分別表示USB的四種Token封包格式。

- OUT 與 IN：用來傳送或接收資料。
- SOF：目標裝置利用 SOF 來辨識訊框(Frame)開始；主機每隔 1ms 就會發出一次 SOF 封包，用來定時與 Frame 計數；等時傳輸可使用 SOF 來啟動傳輸達到同步傳輸的作用。
- SETUP：只有在控制型傳輸的設定階段(Setup stage)使用。SETUP 封包長度固定為 8 Bytes，其格式描述請參考章節 2.4。

欄位	SYNC	PID	ADDR	ENDP	CRC5
位元數	8	8	7	4	5

圖 4 IN、OUT 與 SETUP Token 封包格式

欄位	SYNC	PID	Frame Number	CRC5
位元數	8	8	11	5

圖 5 SOF Token 封包格式

2.2.1.2 Data封包

Data封包位於Token封包之後，被使用在主機和裝置之間傳輸資料，在全速裝置中，Data封包最大的資料量為1024 Bytes。PID定義兩種Data封包型態：DATA0和DATA1，且DATA0及DATA1採取交互出現方式以達到同步與除錯的效果，CRC欄位用來做資料錯誤檢測。圖 6列出由4個欄位所組合而成的Data封包。

欄位	SYNC	PID	Data	CRC16
位元數	8	8	N*8(N=0~1024)	16

圖 6 Data 封包格式

2.2.1.3 Handshake封包

除了等時型傳輸(Isochronous)外，所有USB傳輸皆保證資料傳遞，Handshake封包回應資料是否正確的被傳送或接收，若執行處理動作中發生錯誤，處理動作將重新執行。

Handshake封包由一個PID所組成，如圖 7所示，用來表示資料傳輸的狀態。ACK代表確認資料正確無誤；NAK表示失敗的資料傳輸，要求重新傳輸；STALL表示裝置無法傳送或接收資料，需要主機介入來清除延遲狀況。

欄位	SYNC	PID
位元數	8	8

圖 7 Handshake 封包格式

2.2.2 USB 傳輸型態

USB裝置的每一個端點(Endpoint)都有其特徵，而這些特徵決定了此端點該如何被存取，傳輸的特徵與應用的需求有關。因應不同的週邊裝置的類型與應用，USB規範定義了下列四種端點的傳輸型態，分別為控制型(Control)、中斷型(Interrupt)、巨量型(Bulk)及等時型(Isochronous)傳輸。表 3將列出各個傳輸類型的格式並簡述其功能。

傳輸型態	階段(Stage)	封包	描述
控制型(Control)	Setup	Token	雙向傳輸，每一次的控制型傳輸分為兩個或三個階段(Stage)：Setup、Data(可能沒有)及Status，而每個階段都由一次或多次交易(Transcation)組合而成。所有的USB裝置都必須支援控制型傳輸，用來提供給介於主機與裝置之間的配置、請求或狀態的通訊協定，列舉就是通過控制端點拿到裝置的各種描述元。其三個傳送階段為： (1) Setup：主機對裝置提出要求，包含 8 Bytes 的設定封包。 (2) Data：主機傳送資料給 USB 裝置或是 USB 裝置回應主機的要求並回傳資料給主機。 (3) Status：對資料作確認的動作，來得知資料正
		Data	
		Handshake	
	Data(IN 或 OUT，對於無資料傳輸省略此階段)	Token	
		Data	
		Handshake	
	Status	Token	
		Data	

		Handshake	確的交易。
中斷型(Interrupt)	Data(IN 或 OUT)	Token	USB1.1及USB2.0定義為雙向傳輸，需要靠主機端以週期性地詢問(Polling)才會執行，並且用一個固定速率傳輸少量資料。若是因為錯誤而發生傳送失敗，會在下一個輪詢的期間重新再傳送一次。
		Data	
		Handshake	
巨量型(Bulk)	Data(IN 或 OUT)	Token	可為單向或雙向的傳輸，用於大量資料傳輸且需要確保資料無誤(例如隨身碟的資料讀寫)，若傳輸失敗就會重傳以確保傳輸正確。由於巨量型傳輸是針對未使用的USB頻寬來向主機提出傳輸要求，也就是說，若是目前可用的頻寬不多，巨量型傳輸就必須等待；反之，則可以快速傳輸。因此，巨量型傳輸不需要設定輪詢的時間間隔。
		Data	
		Handshake	
等時型(Isochronous)	Data(IN 或 OUT)	Token	可為單向或雙向的傳輸，等時型傳輸可容許錯誤，同樣用於大量的資料傳輸，以固定的傳輸速率連續不斷地在主機與USB裝置之間的傳輸資料，在傳送資料發生錯誤時，USB並不處理這些錯誤，而是繼續傳送新的資料。應用等時型傳輸的裝置有音訊或視訊裝置等，只要確保傳輸聲音或影像的速率是穩定的、視訊的影像不會變形，就算少幾筆資料也不會重傳。
		Data	

表 3 傳輸型態的格式

2.3 USB 描述元

每一個裝置都有相關描述元集合，描述元是一種描述裝置跟USB特性的資料結構，用來將裝置的特性呈現給主機，基本的描述元架構如圖 8所示，其中包含：裝置描述元(Device Descriptor)、組態描述元(Configuration Descriptor)、介面描述元(Interface Descriptor)、特定的類別描述元(Class-specific Descriptors)、端點描述元(Endpoint Descriptor)及字串描述元(String Descriptor)。連線到主機的一個裝置僅能有一個裝置描述元，裝置描述元中會交代組態描述元和字串描述元的個數，組態描述元至少要有一個，字串描述元是在電腦主機上查看USB裝置時，所看到的USB裝置描述，例如：此裝置的用途、公司名字等。一個USB裝置的功能被使用之前，主機會先向USB裝置取得組態資料，然後再由USB裝置將這些描述元的資料當作組態資料一起傳給主機，組態描述元除了本身組態描述元外，另外包含介面描述元、類別描述元和端點描述元，而HID類別則會有特定的類別描述元，此描述元會交代報告(Report)描述元和實體(Physical)描述元；一個HID介面至少要有一個報告描述元，描述所要交換的資料，稱之為報告(Report)，報告的格式能讓使用者自行修改來處理任何型態的資料，而實體描述元可有可無。

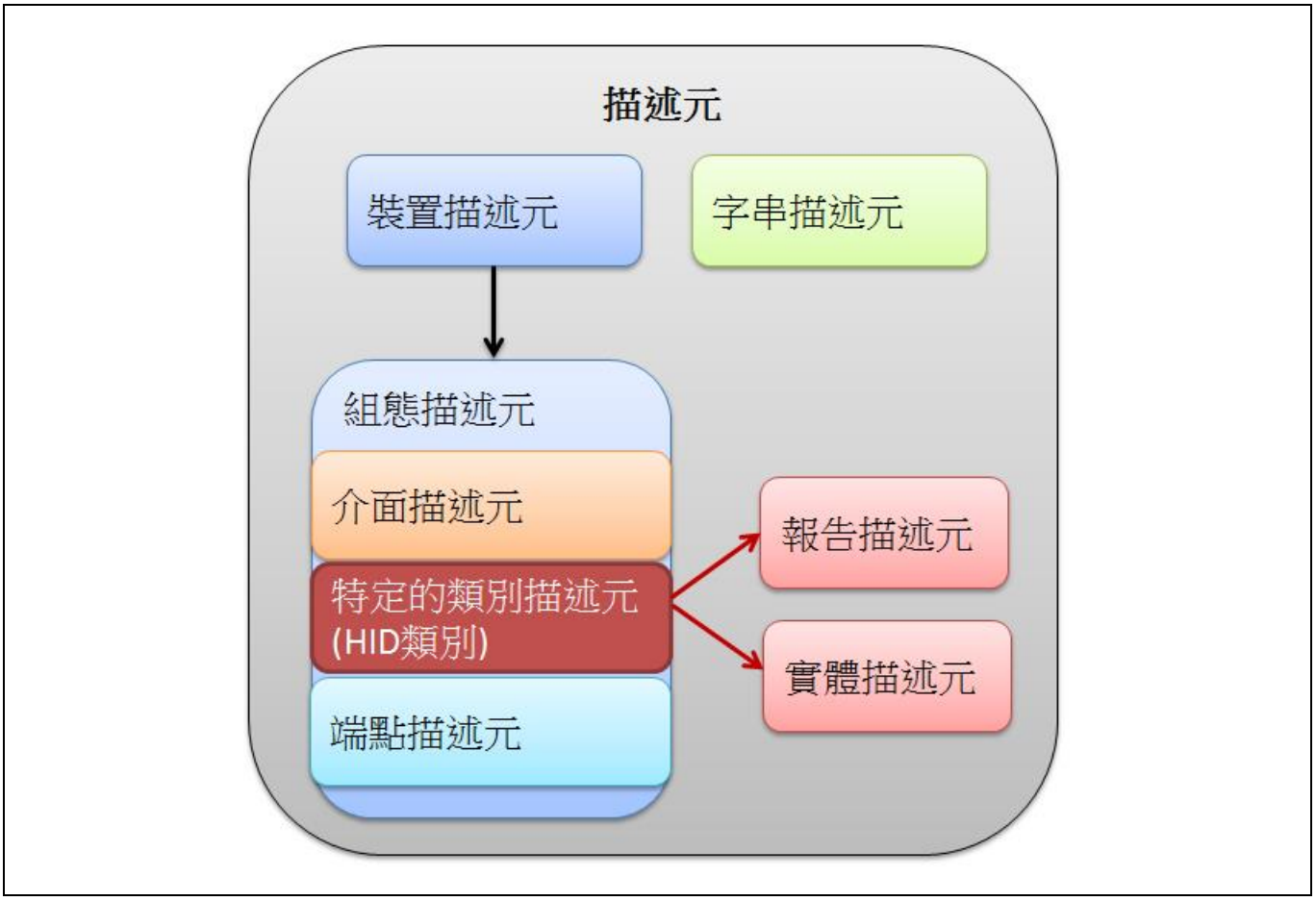


圖 8 描述元的架構

2.3.1 裝置描述元(Device Descriptor)

裝置描述元提供USB裝置之概略敘述，包含USB的版本、裝置類別(HID、MSC...等)、裝置協定、控制端點資料封包的最大長度等需要提供來配置USB裝置的資訊，其長度固定為18 Bytes。而表 4為裝置描述元的格式。

欄位名稱	位置	大　小 (Byte)	描述
bLength	0	1	描述元長度，此描述元的大小，以byte為單位
bDescriptorType	1	1	描述元種類(裝置描述元 = 1)
bcdUSB	2	2	USB版本，0x0110為USB1.1
bDeviceClass	4	1	裝置類別，依USB設計論壇(USB-IF)授權對此產品之分類編號，此欄位數值為0時，裝置分類碼由介面描述元決定

bDeviceSubClass	5	1	裝置次類別，依USB分類及子分類編碼而定，當裝置分類編號為0時，此數值亦為0
bDeviceProtocol	6	1	裝置協定碼，依USB設計論壇(USB-IF)授權編號而定，不使用時為0
bMaxPacketSize0	7	1	控制端點資料封包之最大資料長度
idVendor	8	2	製造商識別碼(VID)，由USB設計論壇(USB-IF)配給的廠商編號
idProduct	10	2	產品識別碼(PID)，廠商可自訂之產品編號
bcdDevice	12	2	裝置版本，由廠商自定之產品發行編號，為4位數之BCD碼
iManufacturer	14	1	製造商名稱索引，USB字串描述元中為廠商字串之偏移位址
iProduct	15	1	產品名稱索引，USB字串描述元中為產品字串之偏移位址
iSerialNumber	16	1	產品序號索引，USB字串描述元中為產品序號字串之偏移位址
bNumConfigurations	17	1	組態個數，此裝置的組態描述元個數

表 4 裝置描述元

2.3.2 組態描述元(Configuration Descriptor)

組態描述元除了本身組態描述元外，另外還包含介面(Interface)描述元、類別(Class)描述元和端點(Endpoint)描述元。多重介面功能的複合裝置介面描述元就必須使用介面關聯描述元(IAD)，以便將多重介面關聯在單一裝置功能中，可看作是多個獨立的USB裝置個別運作， 表 5為組態描述元。

欄位名稱	**位置**	**大　小 (Byte)**	**描述**
bLength	0	1	描述元長度，此描述元的大小，以byte為單位
bDescriptorType	1	1	描述元種類(組態描述元 = 2)
bTotalLength	2	2	組態描述元的總長度

<table>
<tr><td>bNumInterfaces</td><td>4</td><td>1</td><td>支援的介面描述元個數</td></tr>
<tr><td>bConfigurationValue</td><td>5</td><td>1</td><td>組態編號，Set_Configuration時所使用的參數</td></tr>
<tr><td>iConfiguration</td><td>6</td><td>1</td><td>組態名稱索引，用來宣告該名稱在字串描述元中的位置。</td></tr>
<tr><td>bmAttributes</td><td>7</td><td>1</td><td>組態屬性。
<table>
<tr><td>Bit7</td><td>Bit6</td><td>Bit5</td><td>Bit4~ Bit0</td></tr>
<tr><td>保留（設定為1）</td><td>自供電源</td><td>遠端喚醒</td><td>保留（設定為0）</td></tr>
</table></td></tr>
<tr><td>bMaxPower</td><td>8</td><td>1</td><td>最大消耗電流(單位2mA)，若數值=50時，表示100mA</td></tr>
</table>

表 5 組態描述元

介面描述元主要提供該介面使用的端點數量和所屬的裝置類型與規格，其長度固定為9 Bytes。表 6為介面描述元的格式。

欄位名稱	**位置**	**大　小 (Byte)**	**描述**
bLength	0	1	介面描述元長度為9
bDescriptorType	1	1	描述元種類(介面描述元 = 4)
bInterfaceNumber	2	1	介面編號，由0開始的介面描述元編號
bAlternateSetting	3	1	另一個設定的選項值
bNumEndpoints	4	1	此介面包含之端點個數
bInterfaceClass	5	1	介面類別，此值由USB設計論壇(USB-IF)訂立的分類編號
bInterfaceSubClass	6	1	介面次類別，此值由USB設計論壇(USB-IF)訂立的子分類編號
bInterfaceProtocol	7	1	介面協定碼，此值由USB設計論壇(USB-IF)訂立的協定編號
iInterface	8	1	介面名稱索引。USB字串描述元中，有關介面字串描述元之偏移位址

表 6 介面描述元

每個介面使用的端點都有自己的端點描述元，它紀錄端點地址、端點屬性、最大封包大小及輪詢間隔，此描述元位於配置描述元中，表 7說明端點描述元各個欄位意義。

欄位名稱	位置	大　小 (Byte)	描述
bLength	0	1	端點描述元之長度為7
bDescriptorType	1	1	描述元種類(端點描述元 = 5)
bEndpointAddress	2	1	端點地址 Bit[3:0] 端點位址 Bit[6:4] 未使用 Bit[7]=1時為輸入，Bit[7]=0為輸出(端點0可同時為輸出或輸入，不受此位元影響)
bmAttributes	3	1	端點使用的傳輸類型 Bit[1: 0]:00 為控制型傳輸 Bit[1: 0]:01 為等時型傳輸 Bit[1: 0]:10 為巨量型傳輸 Bit[1: 0]:11 為中斷型傳輸
wMaxPacketSize	4	2	最大封包長度
bInterval	6	1	輪詢間隔(單位ms)，介於1~255ms之傳輸週期。對於控制型和巨量型端點，此欄位是多餘的資訊；但是對於等時型端點，此欄的值必須為1；中斷型端點則此值可設為1 至255

表 7 端點描述元

字串描述元使用UNICODE編碼來提供使用者可閱讀的資訊，其描述元皆有索引指標藉由Get Descriptor Request至對應的字串描述元去讀取所需要的資訊，例如製造商、產品、序列編號等。表 8個別描述字串描述元的各個欄位。

欄位名稱	位置	大　小 (Byte)	描述
bLength	0	1	所宣告的製造商名稱之索引值
bDescriptorType	1	1	描述元種類(字串描述元 = 3)
wLANGID	2	2	語言識別碼陣列，此值為0x0409代表使用英文
bLength	4	1	製造商名稱長度

bDescriptorType	5	1	描述元型態
bString	6	8	製造商名稱
bLength	14	1	產品名稱長度
bDescriptorType	15	1	描述元型態
bString	16	32	產品名稱

表 8 字串描述元

2.3.3 HID 特定的類別描述元

HID類別會有特定的類別描述元，它是由描述HID介面的HID描述元及報告描述元所組成。HID描述元是放在組態描述元中，在列舉時用來辨識HID裝置，再經由報告描述元內定義的格式來傳送與接收資料，HID特定的類別描述元詳細資料可參考USB.org網站的Device Class Definition for HID 1.11。

2.4 USB 請求(Request)

主機與裝置之間就必須遵循特定的命令格式，來達到通訊的目的。而這個命令格式就是USB規格書中所訂定的請求(Request)。USB的請求包括標準請求、裝置請求及廠商請求。所有的請求都透過預設管道傳送並且按照控制型傳輸的三個階段(Setup階段、Data階段及Status階段)進行。首先，HOST通過一次控制型傳輸向裝置發送一個8 Bytes的Setup封包，這個封包說明請求的具體資訊，例如：請求類型、資料傳輸方向、接收目標(Device/Interface/Endpoint等)，表 9描述Setup封包格式，圖 9列出了各種的USB裝置標準請求。圖 10整理USB的十一項標準請求(完整的詳細資料可參考USB Specification Revision 2.0)。

以Setup封包”80 06 00 01 00 00 12 00”為例，第1個Byte為bmRequestType=0x80，表示資料傳輸方向是從裝置傳至主機並且為標準型態，而接收端為裝置。第2個Byte為bRequest=0x06，用來區分標準裝置請求，0x06代表Get_Descriptor取得描述元要求。第3及4 Byte為wValue欄位，此位元用來辨識裝置描述元、組態描述元或是字串描述元等標準裝置請求。此值為0x0001代表此要求是取得裝置描述元，而第7及8 Byte為wLength欄位，此值為0x0012代表Data階段的資料長度為18。

位移量	欄位值	大小(Byte)	描述		
0	bmRequestType	1	D7 資料傳輸方向	D[6:5] 型態	D[4:0] 接收端
			0: 主機到裝置	00 標準	0000 裝置
				01 群組	0001 介面

			1: 裝置到主機	10 販售商	0010 端點
				11 保留	0011 其他
1	bRequest	1	D[7:0] 特定請求(請參考圖 9)		
2	wValue	2	D[15:0]字長度欄位，視請求而定		
4	wIndex	2	D[15:0]字長度欄位，通常是傳遞索引或位移量		
6	wLength	2	D[15:0]。若目前的控制型傳輸需要Data階段的話，此欄位代表傳輸的資料長度。換言之，若無Data階段的控制型傳輸，就不需要此欄位。		

表 9 Setup 封包格式

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved for future use	2
SET_FEATURE	3
Reserved for future use	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

圖 9 USB 標準請求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

圖 10 控制傳輸 Setup 封包

2.5 驅動程式

當USB裝置接上電腦主機時，主機透過安裝驅動程式來辨識USB裝置，這些驅動程式能讓對應的裝置運作，並讓主機端的中央處理器能進行各項應用及操作，當裝置端與主機端的連線正確時，就可進行資料傳輸與通訊等功能。

大部分的電腦作業系統都支援這些裝置類別，為其提供驅動程式。常見的裝置如滑鼠、鍵盤、隨身碟等，都會採用電腦作業系統中的HID或MSC類別驅動程式。Windows載入INF檔案中的usbser.sys來當作CDC通訊裝置的驅動程式。INF檔案的檔案格式相當簡單，檔案內包含由USB設計論壇(USB-IF)取得授權的公司和應用產品獨有的VID/PID、裝置的相關資訊及作業系統在辨認或管理裝置時所使用的裝置名稱。Windows會根據VID/PID載入對應的驅動程式，不同類別的USB裝置範例要避免使用相同PID，否則會造成Windows辨識裝置錯誤。

VCOM範例使用由Nuvoton所提供的"NuvotonCDC.inf"，使用者可以自行修改Strings的內容，但是VID和PID需搭配其對應的USB類別裝置範例所提供的VID/PID。表 10列出本文所有應用範例裝置的VID/PID。

```
;
; Windows USB CDC Driver Setup File for Nuvoton-CDC (XP/2000)
;
;

[Version]
Signature    = "$Windows NT$"
Provider     = %COMPANY%
DriverVer    = 10/02/2014,1.0.0.1
CatalogFile  = NuvotonCDC.cat
Class        = Ports
ClassGuid    = {4D36E978-E325-11CE-BFC1-08002BE10318}

[Manufacturer]
%MFGNAME%    = Devices,NT,NTamd64

;-----------------------------------------------------------------------------
; Files
;-----------------------------------------------------------------------------
[DestinationDirs]
DefaultDestDir    = 12

;-----------------------------------------------------------------------------
; Device driver
;-----------------------------------------------------------------------------
[NuvotonCDC_DEV.NT]
Include     = mdmcpq.inf
CopyFiles   = FakeModemCopyFileSection
AddReg      = NuvotonCDC_DEV.NT.AddReg

[NuvotonCDC_DEV.NT.AddReg]
```

```
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,usbser.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"

[NuvotonCDC_DEV.NT.Services]
AddService=usbser, 0x00000002, DriverService

;-------------------------------------------------------------------------
; Services
;-------------------------------------------------------------------------
[DriverService]
DisplayName   = %SERVICE%
ServiceType   = 1
StartType     = 3
ErrorControl  = 1
ServiceBinary= %12%\usbser.sys

;-------------------------------------------------------------------------
; Devices
;-------------------------------------------------------------------------
[Devices.NT]
%DESCRIPTION%= NuvotonCDC_DEV, USB\VID_0416&PID_5011

[Devices.NTamd64]
%DESCRIPTION%= NuvotonCDC_DEV, USB\VID_0416&PID_5011

;-------------------------------------------------------------------------
; Strings
;-------------------------------------------------------------------------
[Strings]
COMPANY  = "Nuvoton Co., Ltd."
MFGNAME  = "www.nuvoton.com"
DESCRIPTION   = "Nuvoton Virtual Com Port"
SERVICE  = "USB RS-232 Emulation Driver"
```

BSP 範例名稱	VID	PID
USBD_HID_Transfer	0x0416	0x5020
USBD_HID_Mouse	0x0416	0x8240
USBD_HID_Keyboard	0x0416	0x0123
USBD_Mass_Storage_Flash	0x0416	0x501E
USBD_VCOM_SerialEmulator	0x0416	0x5011
USBD_Audio_Speaker	0x0416	0x1284

表 10 應用範例的 VID/PID

3 Nano100B USB 2.0 全速裝置控制器

本章節以Cortex® -M0 Nano100B 為例，介紹 NuMicro® USB2.0全速裝置控制器的控制方式。新唐的 Nano100B系列所支持的USB控制器是一組USB2.0全速裝置控制器和PHY收發器，它與USB2.0 Full Speed規範兼容，並且支持控制、中斷、巨量及等時傳輸類型。這個裝置控制器中有兩個主要介面，分別為APB Bus和來自USB PHY收發器的USB Bus。CPU可以透過控制暫存器來操作APB Bus，在此控制器中有個內建512Bytes SRAM作為資料緩衝區(Buffer)。對於IN或OUT傳輸，需要通過APB介面或SIE(Serial Interface Engine)將資料寫入SRAM或從SRAM讀取資料。使用者需要在USBD_BUFSEGx暫存器為每個端點的緩衝區設定SRAM的有效起始地址。

USB裝置控制器提供8個可配置的端點，每個端點可以配置為IN或OUT的資料傳遞方向。裝置的功能地址和每個端點的端點編號必須事先正確的配置好以便用於正確的接收和發送資料封包。每個端點發送或接收的長度定義在USB_MXPLDx暫存器中，只要設定好USB_MXPLDx暫存器就會觸發端點來傳送或接收資料，主機和裝置間Handshake也由端點來處理，且用來管理每個端點的資料同步、端點狀態、當下開始位址、當下Transcation狀態和資料Buffer狀態。

該控制器中有4種不同的中斷事件，分別是喚醒功能、裝置插拔事件、USB事件(如IN ACK、OUT ACK等)和BUS狀態(如Suspend和Resume等)。以上任何事件都將會引發一個中斷，使用者只需要USB_INTSTS暫存器中檢查相關事件的旗標便可獲知發生何種中斷；檢查USB_EPSTS暫存器來確認此端點上發生何種類型的事件。

此USB裝置控制器也支持軟體斷開連線的功能，用於模擬裝置從主機斷開連接的情況。如果使用者設定DRVSE0(USB_CTL[4])為1，USB裝置控制器會強制將D+和D-輸出低電位，等到DRVSE0(USB_CTL[4])設定為0，此時主機將重新列舉USB裝置。

3.1 特性

Nano100B USB2.0全速裝置控制器具有下列的特性：

- 與 USB 2.0 Full Speed 規範兼容
- 提供一個包含 4 個中斷事件 (WAKEUP、FLDET、USB 和 BUS)的中斷向量
- 支援控制、中斷、巨量、等時 4 種傳輸類型
- 支援偵測 USB 匯流排閒置達 3ms 切換至暫停(Suspend)
- 提供 8 個可配置為控制、中斷、巨量、等時傳輸類型的端點
- 512-Byte SRAM 資料緩衝區(Buffer)
- 提供遠程喚醒功能

3.2 功能描述

本章節將針對Nano100B USB2.0全速裝置控制器的方塊圖來介紹各個控制區塊的功能。圖 11為Nano100B USB裝置控制器的方塊圖。

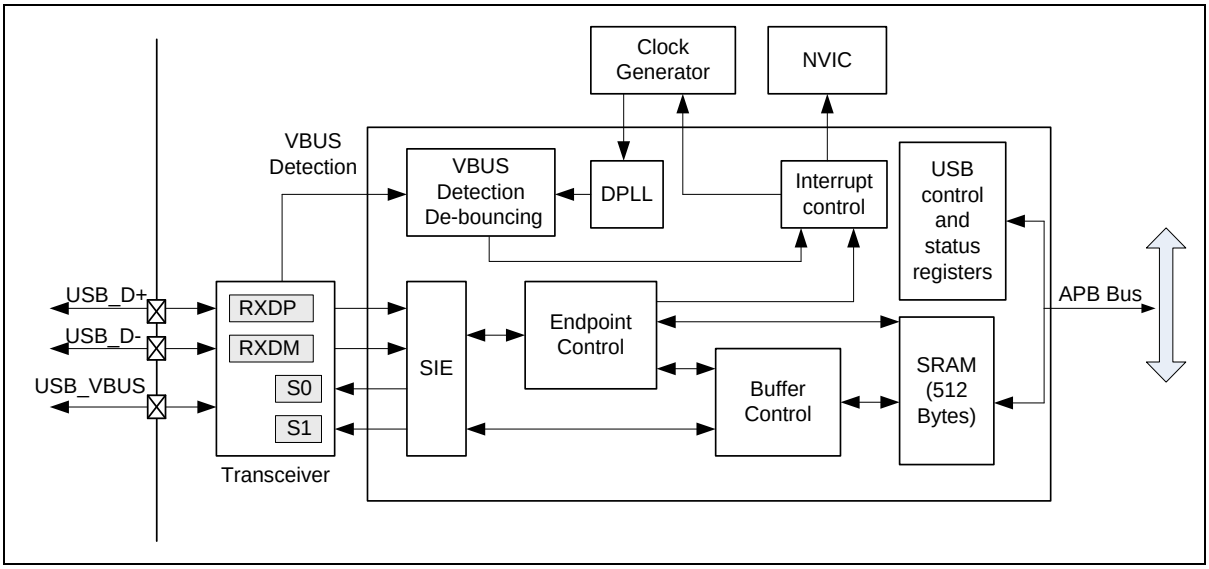


圖 11 Nano100B USB 裝置控制器方塊圖

3.2.1 SIE(Serial Interface Engine)

SIE是串列介面引擎(Serial Interface Engine，簡稱SIE)，SIE位於USB裝置控制器的前端部份，處理USB所定義的通信協定封包，最後再從USB週邊介面傳送與接收資料。SIE的功能包括：

- 封包辨識與處理
- 產生 SOP、EOP、RESET、RESUME 等信號與偵測
- Clock 與 Data 分離
- NRZI 編碼/解碼以及位元填塞
- CRC 的產生與檢查(僅對 Token 與 Data 封包)
- Packet ID (PID) 產生與檢查
- 串列－並列/並列－串列轉換

3.2.2 端點控制

Nano100B USB2.0全速裝置控制器共有8個可配置的端點。每個端點可以被配置成IN或OUT傳輸類型，但是不能設定為雙向來使用，也就是說，雙向的端點會占用2個端點。所有操作包括控制、中斷、巨量或等時型傳輸都在此區塊中執行。另外，端點控制區塊也用於管理每個端點的資料序列同步、端點狀態控制、當下端點起始位址、當下Transcation狀態和每個端點的資料Buffer狀態。

3.2.3 Buffer 控制

Nano100B USB2.0全速裝置控制器中有512 Bytes的內建SRAM作為USB Buffer，SETUP Token封包與8個端點共享此Buffer，用來和主機之間傳送或接收資料。傳送或接收資料功能使用前，使用者必須在USB_BUFSEGx暫存器中填入偏移位址，此值代表配置此端點的有效起始位址；端點發送與接收的長度定義在USB_MXPLDx暫存器。USB Buffer從位址0開始依序分配給SETUP Buffer、控制端點IN Buffer、控制端點OUT Buffer，接下來才分配給其它端點使用。

圖 12描述依據USB_BUFSEGx和USB_MXPLDx暫存器內容定義的每個端點起始位址及它在SRAM Buffer內所占的容量尺寸。若BUFSEG0被設定為0x08h，MXPLD0 設定為 0x40h，則端點0的Buffer使用位址是從USB_BASE+0x108h開始，結束位址為USB_BASE+0x147h；其它端點的起始位址依此類推。

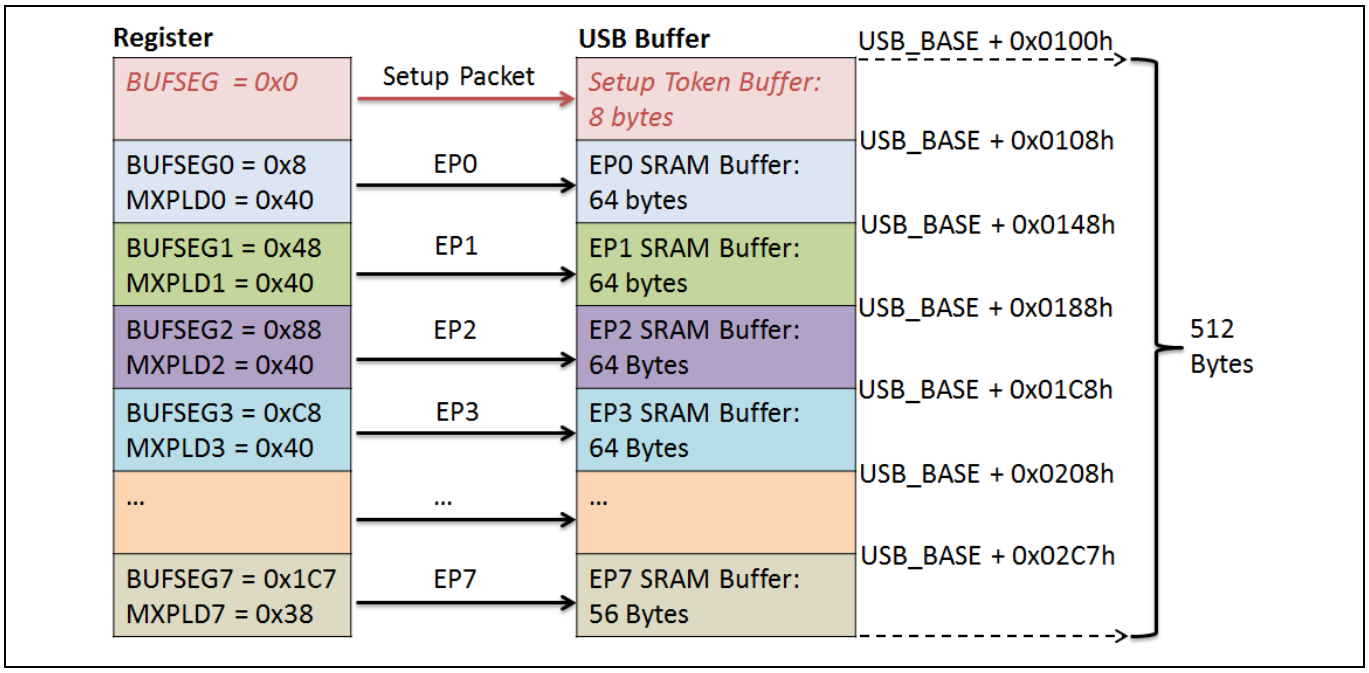


圖 12 各個端點的 Buffer 分配

3.2.4 DPLL

USB資料傳輸速率為12MHz。DPLL採用由Clock產生器產生的48MHz頻率，用來鎖住PHY收發器上RXDP和RXDM的輸入資料。另外，12MHz的USB資料傳輸速率也是由DPLL轉換而來。

3.2.5 VBUS 插拔去雜訊偵測

USB裝置支援熱插拔，為了監測USB裝置被拔掉時的狀態，此裝置控制器在VBUS檢測中斷內提供硬體去除雜訊抖動以防止在USB裝置插拔時所產生的抖動問題。VBUS檢測中斷大約會比USB裝置插入或拔除晚10ms，使用者可以透過讀取FLDET(USB_BUSSTS[4])來得知USB裝置的插拔狀態。

3.2.6 中斷和喚醒

此控制器提供喚醒功能和1個帶有4個中斷事件(WAKEUP、FLDET、USB 和BUS)的中斷向量。WAKEUP中斷事件用在睡眠模式下喚醒系統Clock(Power Down模式在CLK_PWRCON暫存器中定義)；FLDET中斷事件用於檢測USB裝置插入或拔除；USB中斷事件用來告知MCU產生一些USB請求(如IN ACK、OUT ACK等)；BUS中斷事件告知MCU產生USB BUS事件，例如暫停(Suspend)、恢復(Resume)等。需要作用這些中斷時，使用者必須在NVIC和USB_INTEN暫存器中設定相對應的位元以啟動USB中斷。

WAKEUP中斷只會出現在Nano100B進入Power Down模式且喚醒事件發生的時候。在Nano100B

進入Power Down模式後，在USB_VBUS、USB_D+和USB_D-上的任何電位變化都能喚醒Nano100B(假設USB喚醒功能被打開)。若USB喚醒系統超過20ms後且沒有其他USB中斷事件發生，將會發生Wake-up中斷。圖 13為WAKEUP中斷的控制流程。

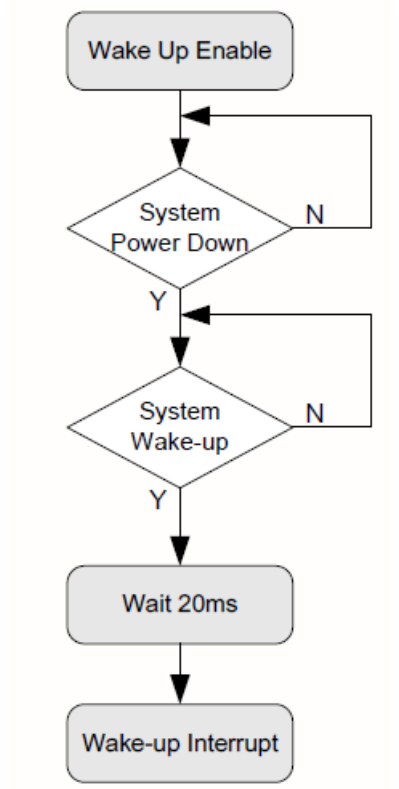


圖 13 USB Wake-up 中斷流程

3.2.7 省電

當Nano100B進入Power Down模式時，USB自動關閉PHY收發器以減少功耗。另外，在暫停(Suspend)情況下，使用者可以在PHY_EN(USB_CTL[1])寫入0也可以省電。

3.2.8 USB 通信控制

使用者可以透過中斷或使用輪詢方式一直去讀取USB_INTSTS暫存器來監測USB Transcation。當一個USB傳輸開始後，硬體會設置USB_INTSTS暫存器，並產生中斷請求給CPU(如果相關中斷有被打開)；若是中斷未打開，使用者可以使用輪詢方式一直去讀取USB_INTSTS暫存器來獲得這些事件。以下是打開中斷時的控制IN與OUT流程(分別如圖 14和圖 15所示)：

當主機向USB裝置控制器讀取資料請求時，使用者需要預先準備相關的資料到指定的端點緩衝區，當資料寫入緩衝區後，使用者需要把實際的資料長度寫入至指定的MAXPLDx暫存器，此暫存器被寫入資料後，內部訊號”In_Rdy”將會被觸發，一旦收到主機發送的IN Token封包後，緩衝區資料將立刻傳給主機。需要注意的是指定資料被傳送完成後，內部訊號”In_Rdy”會由硬體自動清除。如果使用者想要在交易(Transcation)開始之前取消此交易(Transcation)，使用者可以將CLRRDY(USB_CFGx[15])設定為1，此時內部訊號”In_Rdy”或是”Out_Rdy”就會被清除。

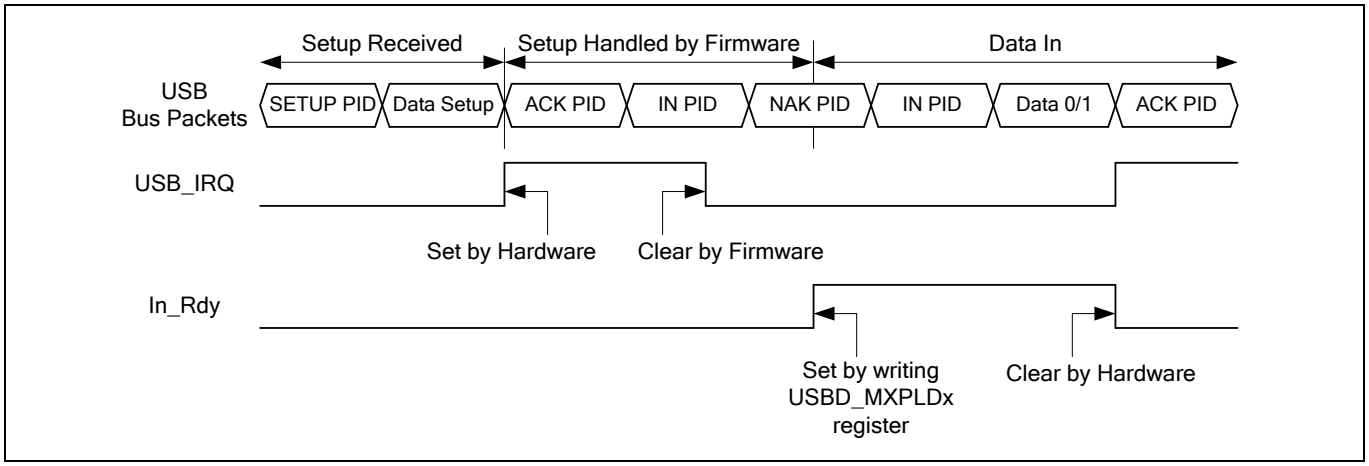


圖 14 Data In流程

相反地，若是主機要傳送資料到USB裝置控制器的OUT端點，硬體會將這些資料存在指定的端點緩衝區中，當這筆交易(Transcation)完成後，硬體會自動在對應端點的MAXPLDx暫存器記錄資料長度，並清除內部訊號”Out_Rdy”，此作法可以避免硬體在使用者尚未拿走當下資料而再次接收下一個交易(Transcation)。一旦使用者處理了這筆交易(Transcation)，對應端點的MAXPLDx暫存器需要由軟體寫入使之再產生一次”Out_Rdy”訊號來接收下一筆交易(Transcation)。

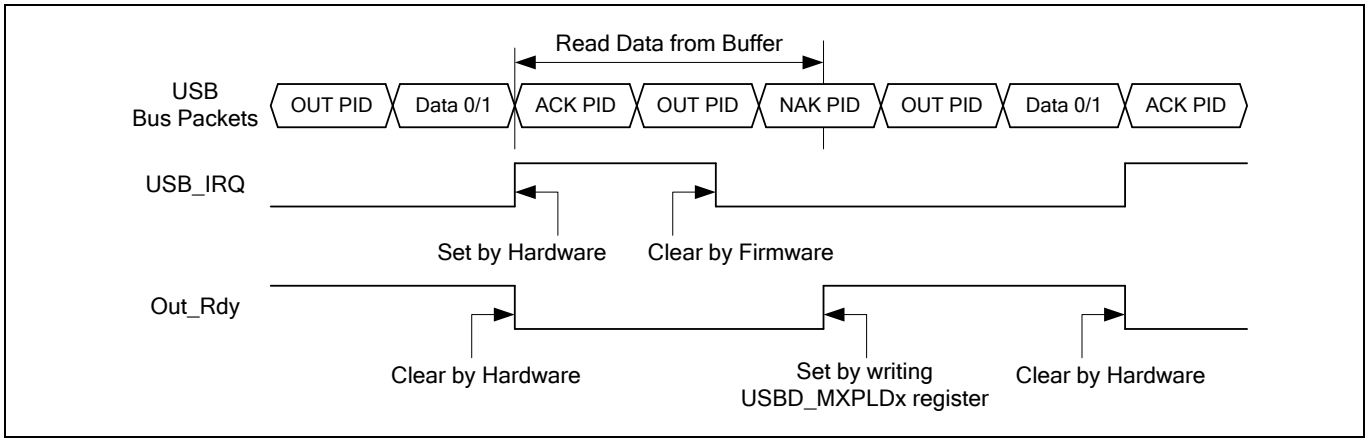


圖 15 Data Out 流程

3.3 暫存器介紹

Nano100B USB2.0全速裝置的所有暫存器如表 11和表 12，表 11暫存器分為兩部分，第一部分是系統設定，另一部分為每個端點設定。系統設定包括：啟動中斷、中斷狀態、端點狀態以及SETUP Token封包的Buffer地址；端點設定包括：每個端點的Buffer地址設定、端點地址配置和最大負載暫存器。接下來章節將個別描述這些暫存器的用法。

暫存器	偏移量	R/W	描述	預設值
USB_BA = 0x4006_0000				
USB_CTL	USB_BA+0x00	R/W	USB 控制暫存器	0x0000_0900

USB_BUSSTS	USB_BA+0x04	R	USB Bus狀態暫存器	0x0000_0000
USB_INTEN	USB_BA+0x08	R/W	中斷Enable暫存器	0x0000_0000
USB_INTSTS	USB_BA+0x0C	R/W	中斷事件狀態暫存器	0x0000_0000
USB_FADDR	USB_BA+0x10	R/W	裝置地址暫存器	0x0000_0000
USB_EPSTS	USB_BA+0x14	R	端點狀態暫存器	0x0000_0000
USB_BUFSEG	USB_BA+0x18	R/W	Setup Token封包Buffer地址偏移暫存器	0x0000_0000
USB_BUFSEGx	USB_BA+0x20	R/W	端點x Buffer地址偏移暫存器	0x0000_0000
USB_MXPLDx	USB_BA+0x24	R/W	端點x 最大負載暫存器	0x0000_0000
USB_CFGx	USB_BA+0x28	R/W	端點x 配置暫存器	0x0000_0000

表 11 USB 裝置暫存器

Nano100B USB2.0 全速裝置控制器中有 512 Bytes 的內建 SRAM(如表 12 所示)作為 USB Buffer，SETUP 封包與 8 個端點共享此塊 Buffer。

USB_BA = 0x4006_0000			
SRAM	USB_BA + 0100h ~ USB_BA + 02FFh	512-Bytes	USB 端點的SRAM Buffer

表 12 USB Buffer 暫存器

3.3.1 USB_CTL

USB_CTL為USB控制暫存器，暫存器裡面的位元作用如下：

1. USB_EN(USB_CTL[0])：Enable USB 功能。
2. PHY_EN(USB_CTL[1])：Enable PHY 收發器。
3. PWRDB(USB_CTL[2])： PHY 收發器的相關電路上電。
4. DPPU_EN(USB_CTL[3])：Enable USB_D+的 Pull-up 電阻。
5. DRVSE0(USB_CTL[4])：SE0 為 USB_D+和 USB_D-都被拉 Low。利用此 Bit 來模擬 USB 裝置插拔的動作。首先將 DRVSE0(USB_CTL[4])設定為 1，此時主機會認定 USB 裝置已經從主機上拔掉；接著把 DRVSE0(USB_CTL[4])設定為 0，表示將 USB 裝置插入主機，利用此方式來強迫主機重新列舉 USB 裝置。
6. RWAKEUP(USB_CTL[8])：USB 進入 K 狀態，用於遠程喚醒處於休眠狀態的主機。

7. WAKEUP_EN(USB_CTL[9])：Enable 喚醒功能。

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved						WAKEUP_EN	RWAKEUP
7	6	5	4	3	2	1	0
Reserved			DRVSE0	DPPU_EN	PWRDB	PHY_EN	USB_EN

3.3.2 USB_INTEN

各種USB的中斷Enable都由此暫存器來設定，包含：USB熱插拔中斷、USB Bus中斷、USB事件中斷與USB喚醒中斷，通常這些中斷都設定為Enable。

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved				WAKEUP_IE	FLDET_IE	USBEVT_IE	BUSEVT_IE

3.3.3 USB_INTSTS 與 USB_BUSSTS

USB_BUSSTS與USB_INTSTS暫存器用來查看所有中斷的狀態。當產生中斷時，都先檢查USB_INTSTS[0:3]藉以判別中斷來源並進行適當處置。

1. 當 BUS_STS(USB_INTSTS[0])被設定為 1 時，代表發生 USB Bus 中斷，然後再查看USB_BUSSTS[2:0]來確認發生哪種 USB Bus 事件。另外，TIMEOUT(USB_BUSSTS[3])是Timeout 事件，該事件只在 IN Token 才有用。當資料傳給主機之後，ACK 經過 18 個 12M Clock 都沒有收到，就會產生 Timeout 事件。
2. 當 USB_STS(USB_INTSTS[1])被設定為 1 時，代表發生 USB 事件中斷，需要再查看SETUP(USB_INTSTS[31])和 EPEVT7~EPEVT0(USB_INTSTS[23:16])來確認是發生 SETUP中斷還是哪個端點發生中斷；假設 SETUP(USB_INTSTS[31])為 1 時，代表發生 SETUP 事件，先讀取 SETUP Buffer 來查看主機發送哪種 USB 命令；如果是EPEVT7~EPEVT0(USB_INTSTS[23:16])的某一個位置被設定為 1，代表對應的端點發生中斷，此時需要再查看 USB_EPSTS 暫存器來確認發生什麼中斷。

3. 如果 FLD_STS(USB_INTSTS[2])被設定為 1 時，查看 FLDET(USB_BUSSTS[4])來得知 USB 裝置是插入還是移除。
4. WKEUP_STS(USB_INTSTS[3])被設定為 1 代表發生喚醒中斷。

Register	Offset	R/W	Description	Reset Value
USB_INTSTS	USB_BA+0x0C	R/W	Interrupt Event Status Register	0x0000_0000

31	30	29	28	27	26	25	24
SETUP	**Reserved**						
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
EPEVT7	**EPEVT6**	**EPEVT5**	**EPEVT4**	**EPEVT3**	**EPEVT2**	**EPEVT1**	**EPEVT0**
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Reserved							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved				**WKEUP_STS**	**FLD_STS**	**USB_STS**	**BUS_STS**

Register	Offset	R/W	Description	Reset Value
USB_BUSSTS	USB_BA+0x04	R	USB Bus Status Register	0x0000_0000

31	30	29	28	27	26	25	24
Reserved							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Reserved							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Reserved							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved			**FLDET**	**TIMEOUT**	**RESUME**	**SUSPEND**	**USBRST**

3.3.4 USB_FADDR

主機分配給該USB裝置的裝置地址要寫到此暫存器裡，用此位址來區分主機發送的封包是給哪個USB裝置使用。

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved	FADDR						

3.3.5 USB_EPSTS 及 USB_EPSTS2

USB_EPSTS和USB_EPSTS2用來表示8個端點的狀態，每個端點共有六種狀態，包括：IN ACK、IN NACK、OUT DATA0 ACK、OUT DATA1 ACK、SETUP ACK及Isochronous transfer end。OVERRUN(USB_EPSTS[7])則用來表示接收到資料是否大於MXPLDx暫存器中定義的最大負載或是Setup資料長度大於8 Bytes。

1. IN ACK 中斷是主機發出 IN Token 封包後，USB 裝置回資料給主機，然後主機回 ACK 給 USB 裝置，之後 USB 裝置才發出 IN ACK 中斷通知 CPU。
2. IN NACK 中斷是在主機發送 IN Token 封包後沒有設定 MXPLDx 暫存器，代表 USB 裝置沒有準備好資料就會回 NACK，之後會發生 IN NACK 中斷。
3. OUT DATA0 ACK 中斷表示主機發送 OUT Token 封包與 DATA0 Token 封包後，USB 裝置接收主機傳送過來的資料並且回 ACK 給主機，之後會發生 OUT DATA0 ACK 中斷。
4. OUT DATA1 ACK 中斷表示主機發送 OUT Token 封包與 DATA1 Token 封包後，USB 裝置接收主機傳送過來的資料並且回 ACK 給主機，之後會發生 OUT DATA1 ACK 中斷。
5. SETUP ACK 中斷是主機發送 8 Bytes 的 SETUP 封包給 USB 裝置，USB 裝置接收到資料之後回 ACK 給主機，接著 USB 裝置就會發出 SETUP ACK 中斷通知 CPU。
6. Isochronous transfer end 中斷表示傳輸一個等時傳輸封包完成。

Register	Offset	R/W	Description	Reset Value
USB_EPSTS	USB_BA+0x14	R	Endpoint Status Register	0x0000_0000

31	30	29	28	27	26	25	24
EPSTS5				EPSTS4			
23	22	21	20	19	18	17	16
EPSTS3				EPSTS2			
15	14	13	12	11	10	9	8
EPSTS1				EPSTS0			
7	6	5	4	3	2	1	0
OVERRUN	Reserved						

Register	Offset	R/W	Description	Reset Value
USB_EPSTS2	USB_BA+0x1C	R	Endpoint Status Register	0x0000_0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
EPSTS7				EPSTS6			

3.3.6 USB_BUFSEG 與 USB_BUFSEGx

USB_BUFSEG指定SETUP封包存放在SRAM的偏移地址；USB_BUFSEGx(x=0~7)暫存器則指定端點IN及端點OUT Buffer在SRAM的偏移地址，這兩個暫存器裡面只要填偏移地址就好。舉例來說，SETUP Buffer在SRAM的起始位置為0，那麼在USB_BUFSEG填0即可。

Register	Offset	R/W	Description	Reset Value
USB_BUFSEG	USB_BA+0x18	R/W	Setup Token Buffer Segmentation Register	0x0000_0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16

Reserved							
15	14	13	12	11	10	9	8
Reserved							BUFSEG
7	6	5	4	3	2	1	0
BUFSEG					Reserved		

Register	Offset	R/W	Description	Reset Value
USB_BUFSEG0	USB_BA+0x20	R/W	Endpoint 0 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG1	USB_BA+0x30	R/W	Endpoint 1 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG2	USB_BA+0x40	R/W	Endpoint 2 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG3	USB_BA+0x50	R/W	Endpoint 3 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG4	USB_BA+0x60	R/W	Endpoint 4 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG5	USB_BA+0x70	R/W	Endpoint 5 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG6	USB_BA+0x80	R/W	Endpoint 6 Buffer Segmentation Register	0x0000_0000
USB_BUFSEG7	USB_BA+0x90	R/W	Endpoint 7 Buffer Segmentation Register	0x0000_0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							BUFSEG
7	6	5	4	3	2	1	0
BUFSEG					Reserved		

3.3.7 USB_MXPLDx

USB_MXPLDx是觸發USB傳送與接收資料的暫存器，它用來定義傳送到主機(IN Token)的資料長度或是從主機接收到(OUT Token)的實際資料長度；也用來表示端點是否準備好發送(IN Token)或接收(OUT Token)資料。

IN Token時，若資料已準備好可以傳送，就在此暫存器中填入想要傳送至主機的資料長度；OUT Token時，表示USB裝置控制器準備接收主機的資料，MXPLD(USB_MXPLDx[7:0])填入的值表示接收主機傳送過來的資料長度，然而透過讀取這個暫存器可以得到實際從主機收到的封包長度。

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
							MXPLD
7	6	5	4	3	2	1	0
MXPLD							

3.3.8 USB_CFGx

USB_CFGx用來配置USB端點的暫存器，用法如下：

1. EP_NUM(USB_CFGx[3:0])用來記錄此端點的地址，此值需要跟端點描述元中的地址相同，這樣硬體在收到主機的訊息時才知道跟哪個端點通訊。
2. ISOCH(USB_CFGx[4])用來設置端點為等時傳輸的端點。
3. EPMODE(USB_CFGx[6:5])決定該端點是 IN 還是 OUT。
4. DSQ_SYNC(USB_CFGx[7])指明現在的傳輸是 DATA0 或 DATA1 PID；對於 IN Token，在主機應答 ACK 後，它將自動切換，然而其他 Token，使用者必須自行處理該位元，以保證 PID 的正確性。
5. CSTALL 和 SSTALL 只用於控制端點。設定 SSTALL(USB_CFGx[9])為 1 之後，USB 裝置會一直回 STALL 給主機，直到 SSTALL(USB_CFGx[9])清為 0。如果希望回傳一次 STALL 就自動將 SSTALL(USB_CFGx[9])清除，就需要將 CSTALL(USB_CFGx[8])設定為 1。

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved						SSTALL	CSTALL
7	6	5	4	3	2	1	0
DSQ_SYNC	EPMODE		ISOCH	EP_NUM			

4 Nano100B USB2.0全速裝置控制器的韌體實作

本章節以Cortex®-M0 Nano100B 為例，介紹 NuMicro® USB2.0全速裝置控制器的韌體實作。Nano100B系列支持USB2.0全速裝置控制器並請在BSP 中提供各種USB裝置類別的原始程式碼，例如：大容量儲存裝置(MSC)、HID裝置及USB轉UART的Virtual COM裝置(CDC)等應用範例，本章節將介紹Nano100B的USB韌體程式的用法，讓使用者比較容易理解如何套用所提供的API Function來實作USB各種類別裝置以節省開發的時間。

USB韌體程式可以分為核心層和應用層，核心層的USB協定提供一個中斷服務程式來處理所有的USB匯流排事件；應用層包括USB描述元資料、類別請求和端點資料處理。表 13和表 14將分別介紹各個類別應用裝置實作時所使用的檔案，這些檔案分成兩種類型，一個是USB Driver層的核心檔案，使用者通常不需要自行修改；另一個則是使用者開發不同USB類別裝置的應用程式編程介面(API)，對於大多數類別應用，需要修改類別請求、描述元資料以及處理不同應用的端點事件。因此，不同類別範例的類別應用層檔案內容也會有所不同。

檔案名稱	描述
usbd(.c,.h)	USB裝置控制器硬體初始化及USB Specification Revision所定義的USB協定，包括匯流排事件、列舉和標準請求。

表 13 核心層檔案

USB裝置範例	檔案名稱	描述
USBD_HID_Transfer	descriptors.c	定義此範例所使用到的裝置描述元、組態描述元、字串描述元、HID描述元和報告描述元。
	hid_transfer.c	HID類別請求和每個端點的事件處理。
	hid_transfer.h	定義HID類別請求及報告描述元類型。
	main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及自訂操作。
USBD_HID_Mouse	descriptors.c	定義此範例所使用到的裝置描述元、組態描述元、字串描述元、HID描述元和報告描述元。
	hid_mouse.c	HID類別請求和每個端點的事件處理。
	hid_mouse.h	定義HID類別請求及報告描述元類型。
	main.c	此範例的主要程式，裡面包含系統及USB裝置初始化，模擬滑鼠游標一直在螢幕上畫圓的滑鼠功

		能。
USBD_HID_Keyboard	descriptors.c	定義此範例所使用到的裝置描述元、組態描述元、字串描述元、HID描述元和報告描述元。
	hid_kb.c	HID類別請求和每個端點的事件處理。
	hid_kb.h	定義HID類別請求及報告描述元類型。
	main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及利用GPIO PB15接地來模擬在鍵盤上輸入按鍵'A'的鍵盤功能。
USBD_Mass_Storage_Flash	descriptors.c	定義此範例所使用到的裝置描述元、組態描述元、字串描述元。
	MassStorage.c	MSC類別請求和每個端點的事件處理。
	MassStorage.h	定義Mass Storage UFI 命令。
	main.c	實作USB MSC隨身碟的主要程式，其中隨身碟的儲存空間是使用Nano100B內的Data Flash，裡面包含系統、USB裝置各種的初始化。
USBD_VCOM_SerialEmulator	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元。
	vcom_serial.c	CDC類別請求和每個端點的事件處理。
	vcom_serial.h	定義CDC類別請求。
	main.c	此範例的主要程式，裡面包含系統及USB裝置初始化並且將裝置虛擬成一個COM Port來傳輸資料。
USBD_Audio_Speaker	descriptors.c	定義此範例所使用到的裝置描述元、組態描述元和字串描述元。
	usbd_audio.c	UAC類別請求和每個端點的事件處理。
	usbd_audio.h	定義UAC類別請求。
	main.c	USB Speaker與Recorder裝置的主要程式，裡面包含系統及USB裝置各種的初始化。

表 14 應用層檔案

4.1 USB 裝置核心層

核心層包含USB裝置控制器硬體初始化及所有的裝置請求。USB裝置控制器提供一個中斷服務程式來處理所有的匯流排事件、標準請求、列舉和USB控制型傳輸。表 15將介紹核心層API的功能，完整的API程式請參考usbd.c。

名稱	描述
USBD_Open	啟動USB裝置控制器功能、USB PHY收發器、上拉電阻接在USB_D+上及强制USB PHY收發器去驅動SE0。
USBD_Start	啟動所有USB相關的中斷(WAKEUP、FLDET、USB和BUS)並且在SysTick Timer延遲100ms後(確保USB裝置控制器已經初始化完成)，關閉軟體禁用功能。
USBD_GetSetupPacket	從使用者自訂的儲存位置取得SETUP封包。
USBD_ProcessSetupPacket	分析SETUP封包，並執行對應的操作。
USBD_GetDescriptor	分析GetDescriptor請求並執行對應的操作。
USBD_StandardRequest	分析標準請求並執行對應的操作。
USBD_PrepareCtrlIn	準備第一個Control IN傳輸。
USBD_CtrlIn	處理Control IN傳輸的剩餘資料。
USBD_PrepareCtrlOut	用於準備第一個Control OUT傳輸。
USBD_CtrlOut	處理連續的Control OUT傳輸。
USBD_SwReset	重置Protocol的所有變數，並將USB裝置地址重置為0。
USBD_SetVendorRequest	用於設定USB裝置的Vendor請求回調函數。
USBD_LockEpStall	用於鎖住相對應的端點來避免由SET FEATURE請求的Stall Clear。如果發生Stall而鎖住的端點，使用者需要重置USB裝置或重新配置裝置來解除鎖定。
USBD_SetStall	設置USB端點Stall狀態，端點將自動回應STALL Token封包。
USBD_ClearStall	清除USB端點Stall狀態，端點將回應ACK/NAK Token封包。

USBD_IRQHandler	中斷服務程式(ISR)，用來處理USB匯流排事件

表 15 核心層 API

列舉過程中的USB請求包括標準請求、類別請求以及廠商請求三類，所有的請求都透過控制型傳輸發送，並且按照控制型傳輸的三個階段進行。控制型傳輸的處理函數有USBD_ProcessSetupPacket()、USBD_PrepareCtrlIn()、USBD_PrepareCtrlOut()、USBD_CtrlIn()和USBD_CtrlOut()。USBD_ProcessSetupPacket用來分析命令，然後再透過USBD_PrepareCtrlIn和USBD_PrepareCtrlOut來傳送或接收資料。

```
void USBD_ProcessSetupPacket(void)
{
  /* 從SETUP Buffer中取得SETUP封包 */
  USBD_MemCopy(g_usbd_SetupPacket, (uint8_t *)USBD_BUF_BASE, 8);

  /* 檢查請求類型 */
  switch (g_usbd_SetupPacket[0] & 0x60) {
  /* 標準USB請求 */
  case REQ_STANDARD: { // Standard
    USBD_StandardRequest();
    break;
  }
  /* USB類別請求 */
  case REQ_CLASS: {
    if (g_usbd_pfnClassRequest != NULL) {
      g_usbd_pfnClassRequest();
    }
    break;
  }
  /* Vendor請求 */
  case REQ_VENDOR: {
    if (g_usbd_pfnVendorRequest != NULL) {
      g_usbd_pfnVendorRequest();
    }
    break;
  }
  default: { // reserved
    /* Setup error, stall the device */
    USBD_SET_EP_STALL(EP0);
    USBD_SET_EP_STALL(EP1);
    break;
  }
```

```
    }
}
```

標準USB命令的處理函數會根據不同的USB命令進行不同的處理。控制型傳輸Read/Write包含3個階段：Setup階段、Data階段、Status階段，圖 16為控制型傳輸Read/Write的時序圖，下列將分別說明控制型傳輸Read/Write的過程：

- 控制型傳輸 Write：

1. Setup 階段：主機發送 SETUP Token 封包，USB 裝置知道這是 USB 命令就會自動將後面 8 Bytes 儲存到 SETUP Buffer 中。
2. Data 階段：主機會發送 OUT Token 封包到端點地址 0，USB 裝置就可以得知這是控制端點的資料並且將資料放到控制端點的 OUT Buffer 中。
3. Status 階段：資料接收完畢後，USB 裝置需要回一個資料長度為 0 的封包給主機。

- 控制型傳輸 Read：

1. Setup 階段：主機發送 SETUP Token 封包，USB 裝置知道這是 USB 命令就會自動將後面 8 Bytes 儲存到 SETUP buffer 中。
2. Data 階段：主機會發送 IN Token 封包到端點地址 0，USB 裝置就可以得知這是主機要求資料，若此時控制端點的 IN Buffer 中有資料，硬體就會將資料回傳給主機；否則回 NACK 給主機。
3. Status 階段：資料接收完畢後，主機會回傳一個資料長度為 0 的封包給 USB 裝置。

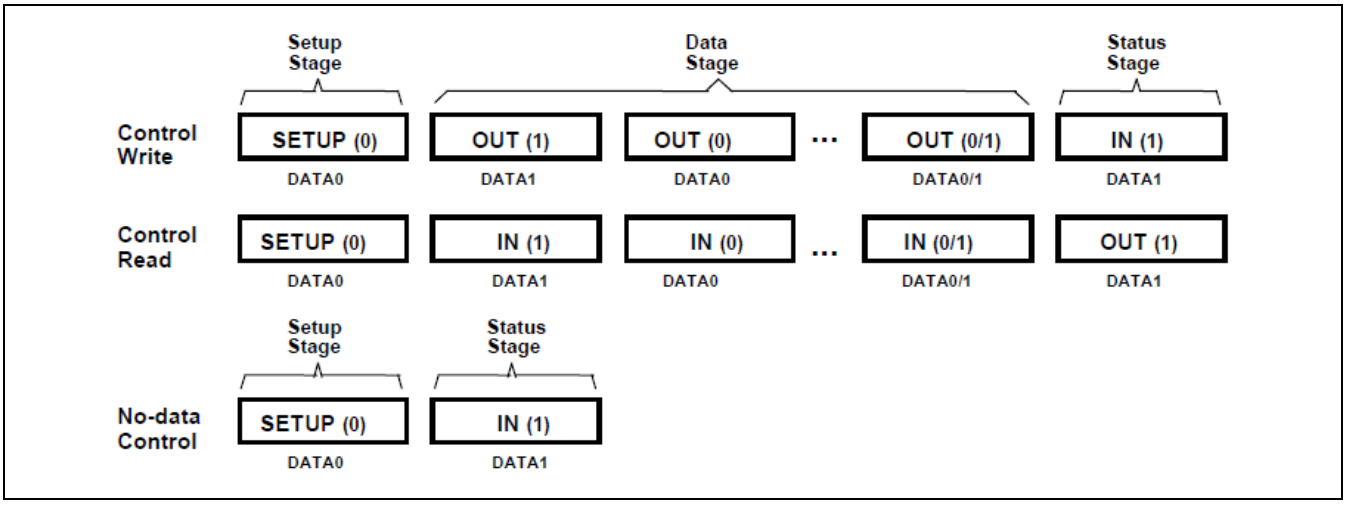


圖 16 控制型傳輸 Read 和 Write 的時序圖

下面的程式碼簡單說明幾個命令的處理方法，按照控制型傳輸的Setup階段、Data階段及Status階段準備所需要的封包，再搭配Firmware程式設定來回應標準請求。

```
void USBD_StandardRequest(void)
{
  /* 清除全域變數 */
  g_usbd_CtrlInPointer = 0;
```

```
g_usbd_CtrlInSize = 0;

if (g_usbd_SetupPacket[0] & 0x80) {
    /* 資料傳輸方向是Device - > Host */
    switch (g_usbd_SetupPacket[1]) {
    case GET_CONFIGURATION: {
        /* 回傳目前的配置設定，將回傳的資料寫入控制端點IN Buffer中 */
        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_UsbConfig;
        /* Data階段 */
        USBD_SET_DATA1(EP0); /* 使用DATA1 PID 發送資料 */
        USBD_SET_PAYLOAD_LEN(EP0, 1); /* 資料長度為1 */
        /* Status階段 */
        USBD_PrepareCtrlOut(0,0); /* 準備第一個Control OUT傳輸 */
        break;
    }
    case GET_DESCRIPTOR: {
        /* 取得描述元 */
        USBD_GetDescriptor();
        break;
    }
    …
} else {
    /* 資料傳輸方向是Host- > Device */
    switch (g_usbd_SetupPacket[1]) {
    case SET_ADDRESS: {
        /* SET_ADDRESS沒有Data階段 */
        g_usbd_UsbAddr = g_usbd_SetupPacket[2];
        /* Status階段 */
        USBD_SET_DATA1(EP0); /* 使用DATA1 PID 發送資料 */
        USBD_SET_PAYLOAD_LEN(EP0, 0); /* 資料長度為0 */
        break;
    }
    case SET_INTERFACE: {
        /* SET_INTERFACE沒有Data階段 */
        g_usbd_UsbAltInterface = g_usbd_SetupPacket[2];
        if (g_usbd_pfnSetInterface != NULL)
            g_usbd_pfnSetInterface(g_usbd_UsbAltInterface);
        /* Status階段 */
        USBD_SET_DATA1(EP0); /* 使用DATA1 PID 發送資料 */
        USBD_SET_PAYLOAD_LEN(EP0, 0); /* 資料長度為0 */
        break;
```

```
    }
    …
  }
}
```

若控制型傳輸要傳送给主機的資料長度大於本身控制型傳輸封包的最大長度，剩下的資料就在USBD_CtrlOut或USBD_CtrlIn函數裡面去傳輸。

```
void USBD_CtrlIn(void)
{
  if(g_usbd_CtrlInSize) { /*如果還有資料要傳送到主機 */
    /* 處理剩下的資料 */
    if(g_usbd_CtrlInSize > g_usbd_CtrlMaxPktSize) {
      /* 剩下的資料長度 > 封包的最大長度，複製MXPLD Bytes到控制端點IN Buffer中 */
      USBD_MemCopy((uint8_t *)USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0),
                   (uint8_t *)g_usbd_CtrlInPointer, g_usbd_CtrlMaxPktSize);
      /* 觸發MXPLD暫存器 */
      USBD_SET_PAYLOAD_LEN(EP0, g_usbd_CtrlMaxPktSize);
      g_usbd_CtrlInPointer += g_usbd_CtrlMaxPktSize;
      g_usbd_CtrlInSize -= g_usbd_CtrlMaxPktSize;
    } else {
      /* 剩下的資料長度 <= 封包的最大長度，將剩下的資料全部複製到控制端點IN Buffer中 */
      USBD_MemCopy((uint8_t *)USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0),
                   (uint8_t *)g_usbd_CtrlInPointer, g_usbd_CtrlInSize);
      /* 觸發MXPLD暫存器 */
      USBD_SET_PAYLOAD_LEN(EP0, g_usbd_CtrlInSize);
      g_usbd_CtrlInPointer = 0;
      g_usbd_CtrlInSize = 0;
    }
  } else {
    /* Set address Status階段结束 */
    if((g_usbd_SetupPacket[0] == REQ_STANDARD) && (g_usbd_SetupPacket[1] == SET_ADDRESS)) {
      if((USBD_GET_ADDR() != g_usbd_UsbAddr) && (USBD_GET_ADDR() == 0)) {
        USBD_SET_ADDR(g_usbd_UsbAddr); /* 將裝置地址寫到暫存器中 */
      }
    }
    /* 没有資料要送，準備Status階段 */
    USBD_SET_PAYLOAD_LEN(EP0, 0);
  }
}

void USBD_CtrlOut(void)
```

```
{
    uint32_t u32Size;
    /* 還沒接收完該收的資料 */
    if(g_usbd_CtrlOutSize < g_usbd_CtrlOutSizeLimit) {
        u32Size = USBD_GET_PAYLOAD_LEN(EP1);
    /* 將接收到的資料複製到Buffer中 */
        USBD_MemCopy((uint8_t *)g_usbd_CtrlOutPointer, (uint8_t *)USBD_BUF_BASE +
                                USBD_GET_EP_BUF_ADDR(EP1), u32Size);
        g_usbd_CtrlOutPointer += u32Size;
        g_usbd_CtrlOutSize += u32Size;
    }
    if(g_usbd_CtrlOutSize < g_usbd_CtrlOutSizeLimit)
        USBD_SET_PAYLOAD_LEN(EP1, g_usbd_CtrlMaxPktSize); /* 觸發下一次接收 */
}
```

4.2 USB 裝置應用層

USB裝置應用層用於裝置的配置、描述元資料、類別請求和端點資料處理。另外，USB類別的主要功能也在此層處理。使用者可以在應用層修改檔案，以適應各種不同類型的類別需求。不同類別範例的類別應用層檔案內容會有所不同。

Nano100B的BSP範例程式中提供不同類別USB裝置的應用範例，本章節將說明USB裝置的實作步驟，讓使用者在開發USB應用裝置時更容易上手。首先，初始化USB裝置控制器讓硬體可以正常動作，接下來根據不同的應用來配置所需使用的端點及其傳輸類型，最重要的是準備各種描述元讓列舉時來使用。最後，分別處理各種事件或請求來達成此裝置的功能。

4.2.1 初始化 USB 裝置

USB裝置的Clock來源是從PLL Clock除頻後產生48MHz。使用者必須在USB裝置控制器Enable之前，先對PLL進行相關設定。設置USBD_EN(APBCLK[27])來打開USB Clock並設置USB_N(CLKDIV[7:4])對USB Clock來源做除頻動作，以產生合適的USB Clock頻率。

```
void SYS_Init(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Enable external 12MHz HXT */
    CLK_EnableXtalRC(CLK_PWRCTL_HXT_EN_Msk);
    CLK_EnablePLL(CLK_PLLCTL_PLL_SRC_HXT, 96000000);
    /* Waiting for clock ready */
    CLK_WaitClockReady(CLK_CLKSTATUS_HXT_STB_Msk | CLK_CLKSTATUS_PLL_STB_Msk);

    CLK_SetHCLK(CLK_CLKSEL0_HCLK_S_PLL, CLK_HCLK_CLK_DIVIDER(3));
```

```
    /* Select IP clock source */
    CLK_SetModuleClock(USBD_MODULE, 0, CLK_USB_CLK_DIVIDER(2));
    /* Enable IP clock */
    CLK_EnableModuleClock(USBD_MODULE);

    /* Select IP clock source */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UART_S_HXT, CLK_UART_CLK_DIVIDER(1));
    /* Enable IP clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /*---------------------------------------------------------------------*/
    /* Init I/O Multi-function                                             */
    /*---------------------------------------------------------------------*/
    /* Set PA multi-function pins for UART0 RXD and TXD */
    SYS->PA_H_MFP &= ~( SYS_PA_H_MFP_PA15_MFP_Msk | SYS_PA_H_MFP_PA14_MFP_Msk);
    SYS->PA_H_MFP |= (SYS_PA_H_MFP_PA15_MFP_UART0_TX| SYS_PA_H_MFP_PA14_MFP_UART0_RX);

    /* Lock protected registers */
    SYS_LockReg();
}
```

g_usbd_pfnVendorRequest 、 g_usbd_pfnClassRequest 和 g_usbd_pfnSetInterface 結 構 為 USB 請 求 (Request)的回調函數，g_usbd_sInfo為應用層的各種描述元資料指標。使用者在初始化USB裝置控制器時，需要依照USB類別裝置應用來設定回調函數及設定描述元內容。

```
S_USBD_INFO_T *g_usbd_sInfo;
VENDOR_REQ g_usbd_pfnVendorRequest = NULL;
CLASS_REQ g_usbd_pfnClassRequest = NULL;
SET_INTERFACE_REQ g_usbd_pfnSetInterface = NULL;

void USBD_Open(S_USBD_INFO_T *param, CLASS_REQ pfnClassReq, SET_INTERFACE_REQ pfnSetInterface)
{
    g_usbd_sInfo = param;
    g_usbd_pfnClassRequest = pfnClassReq;
    g_usbd_pfnSetInterface = pfnSetInterface;

    /* get EP0 maximum packet size */
    g_usbd_CtrlMaxPktSize = g_usbd_sInfo->gu8DevDesc[7];

    /* Initial USB engine */
    USBD->CTL = 0x29f;
```

```
    USBD->PDMA |= USBD_PDMA_BYTEM_Msk;
    /* Force SE0, and then clear it to connect*/
    USBD_SET_SE0();
}
```

透過USB_CTL暫存器設定該控制器的屬性來啟動USB裝置控制器的硬體功能，例如選擇CPU以Byte或Word(4 Bytes)去存取SRAM、啟動USB裝置控制器功能、USB PHY收發器、上拉電阻接在USB_D+上及强制USB PHY收發器去驅動SE0用以模擬USB Device從主機上被移除。

```
void USBD_Open(S_USBD_INFO_T *param, CLASS_REQ pfnClassReq, SET_INTERFACE_REQ pfnSetInterface)
{
    g_usbd_sInfo = param;
    g_usbd_pfnClassRequest = pfnClassReq;
    g_usbd_pfnSetInterface = pfnSetInterface;

    /* get EP0 maximum packet size */
    g_usbd_CtrlMaxPktSize = g_usbd_sInfo->gu8DevDesc[7];

    /* Initial USB engine */
    USBD->CTL = 0x29f;
    USBD->PDMA |= USBD_PDMA_BYTEM_Msk;
    /* Force SE0, and then clear it to connect*/
    USBD_SET_SE0();
}
```

所有USB列舉的過程和管理裝置的狀態就是以中斷服務程式來處理所有的USB匯流排事件。USB的中斷處理函數USBD_IRQHandler，使用者可以透過它了解整個Nano100B的USB處理流程。使用中斷或輪詢檢測USB_INTSTS暫存器來監測USB匯流排上的所有事件，像是讀取暫存器中的USB_EPSTS和USB_INTSTS來得知是哪類請求、在哪個端點上，並採取必要的回應(EPx_Handler)，讀取USB_BUSSTS獲取USB重置、插拔、恢復或暫停，以下為USB的中斷處理流程：

```
void USBD_IRQHandler(void)
{
    uint32_t u32IntSts = USBD_GET_INT_FLAG();
    uint32_t u32State = USBD_GET_BUS_STATE();

    /* 熱插拔中斷 */
    if(u32IntSts & USBD_INTSTS_FLDET)
    {
        /* 清除熱插拔中斷狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_FLDET);

        if(USBD_IS_ATTACHED())
```

```
    {
      /* USB 插入，開啟USB和PHY */
      USBD_ENABLE_USB();
    }
    else
    {
      /* USB 拔除，關閉USB功能 */
      USBD_DISABLE_USB();
    }
  }

  /* USB匯流排中斷 */
  if(u32IntSts & USBD_INTSTS_BUS)
  {
    /* 清除USB匯流排狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_BUS);

    if(u32State & USBD_STATE_USBRST)
    {
      /* USB匯流排重置 */
      USBD_ENABLE_USB();
      USBD_SwReset();
    }
    if(u32State & USBD_STATE_SUSPEND)
    {
      /* 開啟USB,關閉PHY */
      USBD_DISABLE_PHY();
    }
    if(u32State & USBD_STATE_RESUME)
    {
      /* 開啟 USB 和 PHY */
      USBD_ENABLE_USB();
    }
  }

  /* USB事件中斷 */
  if(u32IntSts & USBD_INTSTS_USB)
  {
    /* SETUP事件 */
    if(u32IntSts & USBD_INTSTS_SETUP)
    {
```

```
    /* 清除SETUP事件狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_SETUP);

    /* 清除控制端點的Ready狀態 */
    USBD_STOP_TRANSACTION(EP0);
    USBD_STOP_TRANSACTION(EP1);

    /* 分析Setup Packet */
    USBD_ProcessSetupPacket();
}

/* 端點事件 */
if(u32IntSts & USBD_INTSTS_EP0) /* EP0被配置為控制端點IN */
{
    /* 清除EP0事件狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_EP0);
    /* 處理控制端點傳送資料給主機完完成中斷 */
    USBD_CtrlIn();
}

if(u32IntSts & USBD_INTSTS_EP1) /* EP1被配置為控制端點OUT */
{
    /* 清除EP1事件狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_EP1);

    /* 處理主機通過控制端點發送數據給Device */
    USBD_CtrlOut();
}

if(u32IntSts & USBD_INTSTS_EP2)
{
    /* 清除EP2事件狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_EP2);
    /* EP2中斷處理函數 */
    EP2_Handler();
}

if(u32IntSts & USBD_INTSTS_EP3)
{
    /* 清除EP3事件狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_EP3);
```

```
        /* EP3中斷處理函數 */
        EP3_Handler();
    }

    /* 其它端點也可以依照需求，設定端點處理函數 */
    if(u32IntSts & USBD_INTSTS_EP4)
    {
        /* 清除EP4事件狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_EP4);
    }

    if(u32IntSts & USBD_INTSTS_EP5)
    {
        /* 清除EP5事件狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_EP5);
    }

    if(u32IntSts & USBD_INTSTS_EP6)
    {
        /* 清除EP6事件狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_EP6);
    }

    if(u32IntSts & USBD_INTSTS_EP7)
    {
        /* 清除EP7事件狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_EP7);
    }
  }
}
```

4.2.2 端點配置

一個USB裝置可以有多個端點，每個端點的作用不同，所以需要端點地址來辨識端點。端點地址定義在USB_CFGx暫存器中的EP_NUM，它用來記錄端點的地址，此地址需要跟端點描述元中的地址相對應，這樣硬體才知道要跟哪個端點通訊，而每個端點發送或接收到的最大封包長度定義在USBD_MXPLD暫存器，此暫存器就會觸發USB裝置進行資料接收或傳送動作。以下的程式碼擷取自HID Transfer範例，用來說明端點地址與端點描述元中的地址關係。其中，HID_Init()用來配置所有被使用的端點，然而端點也會被記錄在組態描述元中的端點描述元裡面。因此，在端點描述元裡的端點地址必須根據在HID_Init()所定義的各端點地址來填寫。

```
void HID_Init(void)
{
```

```
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Interrupt OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}
```

4.2.3 描述元

主機靠各種描述元來辨識USB裝置，準備各種的描述元時，其描述內容要符合USB規範和其所要實現的USB類別的規定。各種描述元內容請參考各個類別範例的descriptors.c檔案。所有描述元定義在s_usbd_info的結構內，包含裝置描述元、組態描述元、HID類別描述元及字串描述元。使用者實作各種USB裝置時，只要針對其類別規定來填寫描述元的內容即可。

```
typedef struct s_usbd_info {
    uint8_t *gu8DevDesc;                /* 裝置描述元 */
    uint8_t *gu8ConfigDesc;             /* 組態描述元 */
    uint8_t **gu8StringDesc;            /* 字串描述元 */
    uint8_t **gu8HidReportDesc;         /* 報告描述元 */
    uint32_t *gu32HidReportSize;        /* 報告描述元的長度 */
    uint32_t *gu32ConfigHidDescIdx;     /* HID描述元在組態描述元內的位置 */
```

```
} S_USBD_INFO_T;

S_USBD_INFO_T gsInfo = {
    gu8DeviceDescriptor,
    gu8ConfigDescriptor,
    gpu8UsbString,
    gu8UsbHidReport,
    gu32UsbHidReportLen,
    gu32ConfigHidDescIdx,
};
```

接下來將示範HID Transfer裝置所需使用到的所有描述元。

```
/* 裝置描述元 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,                 /* bLength */
    DESC_DEVICE,                /* bDescriptorType */
    0x10, 0x01,                 /* bcdUSB */
    0x00,                       /* bDeviceClass */
    0x00,                       /* bDeviceSubClass */
    0x00,                       /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,           /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,                 /* bcdDevice */
    0x01,                       /* iManufacture */
    0x02,                       /* iProduct */
    0x00,                       /* iSerialNumber - no serial */
    0x01                        /* bNumConfigurations */
};

/* 組態描述元 */
uint8_t gu8ConfigDescriptor[] = {
    LEN_CONFIG,                 /* bLength */
    DESC_CONFIG,                /* bDescriptorType */
    /* wTotalLength */
    (LEN_CONFIG+LEN_INTERFACE+LEN_HID+LEN_ENDPOINT*2) & 0x00FF,
    ((LEN_CONFIG+LEN_INTERFACE+LEN_HID+LEN_ENDPOINT*2) & 0xFF00) >> 8,
```

```
0x01,                   /* bNumInterfaces */
0x01,                   /* bConfigurationValue */
0x00,                   /* iConfiguration */
0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),        /* bmAttributes */
USBD_MAX_POWER,     /* MaxPower */

/* 介面0 : HID Transfer介面描述元 */
LEN_INTERFACE,          /* bLength */
DESC_INTERFACE,         /* bDescriptorType */
0x00,                   /* bInterfaceNumber */
0x00,                   /* bAlternateSetting */
0x02,                   /* bNumEndpoints */
0x03,                   /* bInterfaceClass */
0x00,                   /* bInterfaceSubClass */
0x00,                   /* bInterfaceProtocol */
0x00,                   /* iInterface */

/* HID 的類別描述元 */
LEN_HID,                /* Size of this descriptor in UINT8s. */
DESC_HID,               /* HID descriptor type. */
0x10, 0x01,             /* HID Class Spec. release number. */
0x00,                   /* H/W target country. */
0x01,                   /* Number of HID class descriptors to follow. */
DESC_HID_RPT,           /* Descriptor type. */
/* Total length of report descriptor. */
sizeof(HID_DeviceReportDescriptor) & 0x00FF,
(sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點描述元: Interrupt in. */
LEN_ENDPOINT,           /* bLength */
DESC_ENDPOINT,          /* bDescriptorType */
(INT_IN_EP_NUM | EP_INPUT),         /* bEndpointAddress */
EP_INT,                 /* bmAttributes */
/* wMaxPacketSize */
EP2_MAX_PKT_SIZE & 0x00FF,
(EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
HID_DEFAULT_INT_IN_INTERVAL,    /* bInterval */
/* 端點描述元: Interrupt out. */
LEN_ENDPOINT,           /* bLength */
DESC_ENDPOINT,          /* bDescriptorType */
(INT_OUT_EP_NUM | EP_OUTPUT),       /* bEndpointAddress */
```

```
    EP_INT,                         /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL             /* bInterval */
};

/* 字串描述元 */
uint8_t *gpu8UsbString[4] = {
    gu8StringLang,
    gu8VendorStringDesc,
    gu8ProductStringDesc,
    NULL,
};

/* USB Language 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                              /* bLength */
    DESC_STRING,                    /* bDescriptorType */
    0x09, 0x04
};

/* USB Vendor 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* USB Product 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    26,                             /* bLength */
    DESC_STRING,                    /* bDescriptorType */
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'T', 0, 'r', 0, 'a', 0, 'n', 0, 's', 0, 'f', 0, 'e', 0, 'r', 0
};

uint8_t *gu8UsbHidReport[3] = {
    HID_DeviceReportDescriptor,         /* HID transfer報告描述元 */
    NULL,
    NULL,
};
```

```
uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_DeviceReportDescriptor),    /* HID transfer報告描述元的長度 */
    0,
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
    (LEN_CONFIG+LEN_INTERFACE),     /* HID transfer描述元在組態描述元內的位置*/
    0,
    0,
};

/* HID Transfer報告描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,          // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,                // Usage (Vendor Usage 1)
    0xA1, 0x01,                // Collection (Application)
    0x19, 0x01,                // Usage Minimum
    0x29, 0x40,                // Usage Maximum //64 input usages total (0x01 to 0x40)
    0x15, 0x00,                // Logical Minimum (data bytes in the report may have minimum value = 0x00)
    0x26, 0xFF, 0x00,          // Logical Maximum (data bytes in the report may have maximum value = 0x00FF
    0x75, 0x08,                // Report Size: 8-bit field size
    0x95, 0x40,                // Report Count: Make sixty-four 8-bit fields (the next time the parser hits  an "Input",
                               // "Output", or "Feature" item)
    0x81, 0x00,                // Input (Data, Array, Abs): Instantiates input packet fields based on the
                               // above report size, count, logical min/max, and usage.
    0x19, 0x01,                // Usage Minimum
    0x29, 0x40,                // Usage Maximum //64 output usages total (0x01 to 0x40)
    0x91, 0x00,                // Output (Data, Array, Abs): Instantiates output packet fields. Uses same
                               // report size and count as "Input" fields, since nothing new/different was
                               // specified to the parser since the "Input" item.
    0xC0                       // End Collection
};
```

5 USB類別範例

USB裝置定義許多不同的類別(Class)，常見的類別有：人性化介面裝置類別(HID)、大容量儲存裝置類別(MSC)、USB通訊裝置類別(CDC)，以及USB音訊裝置類別(UAC)，例如：鍵盤、滑鼠、USB隨身碟和USB轉RS232等。

本章討論如何以Nano100B系列開發實作各種類別應用裝置，表 16列出USB類別的示範程式。圖17為本章節所有USB類別範例的架構圖，紅色區塊代表使用者需要自行修改的項目。使用者自行修改的項目有：回應裝置的類別特定請求、Endpoint配置及定義各個描述元，各個範例章節中將針對這三項的修改做說明。測試環境使用NuMicro® Nano130 開發板或學習板。所有類別應用實例都可在Nano100B系列的BSP\SampleCode\StdDriver找到。

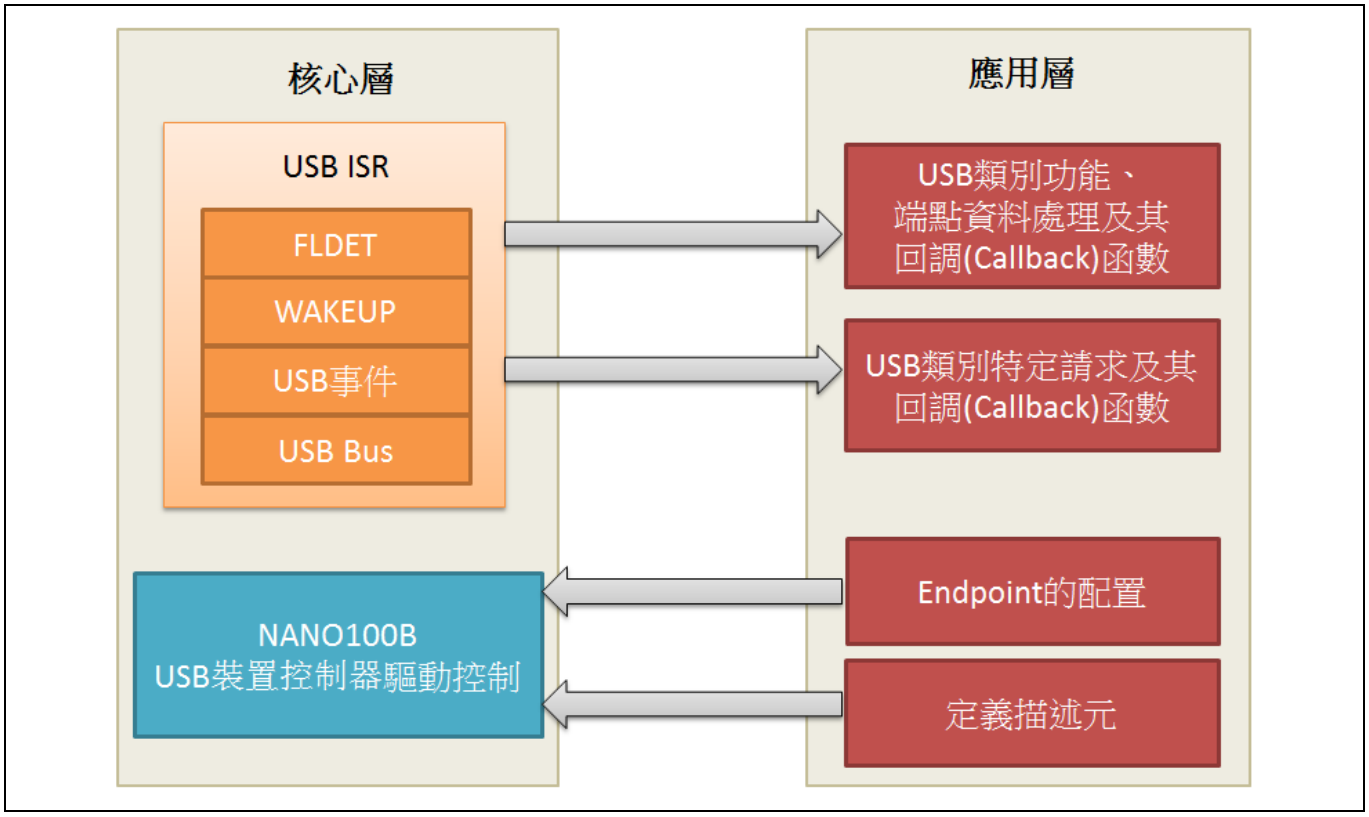


圖 17 USB 類別應用範例架構圖

USB類別	範例	說明
HID	USBD_HID_Transfer	透過HID介面讓USB裝置與電腦之間做資料傳輸，搭配主機端的應用程式來測試此功能。
	USBD_HID_Keyboard	利用GPIO PB15接地來模擬在鍵盤上輸入按鍵'A'。
	USBD_HID_Mouse	模擬滑鼠在電腦螢幕上畫圓移動顯示。
MSC	USBD_Mass_Storage_Flash	使用Nano100B 內的Data Flash當作儲存容量的儲存裝置。
CDC	USBD_VCOM_SerialEmulator	在電腦上虛擬出 COM Port，測試此功能使用終端機連線工具，需要將設定變更至對應的 COM Port才能成功連接使用。
UAC	USBD_Audio_Speaker	USB Audio音效裝置，透過NAU8822(音訊轉碼器)輸入或輸出音訊資料。

表 16 Nano100B 支援的 USB 類別示範程式

5.1 HID

USB裝置中最普遍使用HID(Human Interface Device)裝置，因為大部分作業系統會支援HID裝置的驅動程式，所以使用者就無須再重新撰寫其驅動程式。HID介面必須符合HID群組所定義的規格之要求，所有的HID傳輸使用預設的控制端點或是中斷端點，一個全速HID裝置的傳輸速率每秒64KBytes。如果裝置要送出資料(如滑鼠的移動或是鍵盤的按鍵敲擊)，HID能夠要求主機以週期性的方式輪詢裝置，以取得該HID裝置所送出的資料。

HID所定義的資料結構描述元中，報告描述元(Report Descriptor)是用來交換資料。此裝置的韌體必須包含用來描述所要交換資料的報告描述元，而此報告的格式能夠讓使用者修改以處理任何型態的資料。

5.1.1 HID Transfer

本章節以HID Transfer為例來說明如何實作該裝置並且如何通過中斷型端點向主機接收或傳送資料。搭配HID Transfer專用的Windows主機端應用程式，使用者可以利用此應用程式讓HID裝置與電腦之間做資料傳輸。

HID Transfer專用的主機端應用程式，利用Microsoft的Visual C++來開發。HID Transfer應用程式簡單示範如何透過Report來讀寫HID裝置，使用者可以利用此應用程式來驗證本文中的HID Transfer裝置是否可以正常操作；也可以參考此範例的程式碼來開發其他的HID應用。

HID Transfer專用主機端應用程式的執行檔放在WindowsTool\debug\HIDTransferTest.exe，執行此程式可以驗證HID Transfer裝置是否列舉成功並且可以正常操作使用，執行結果如圖 18。

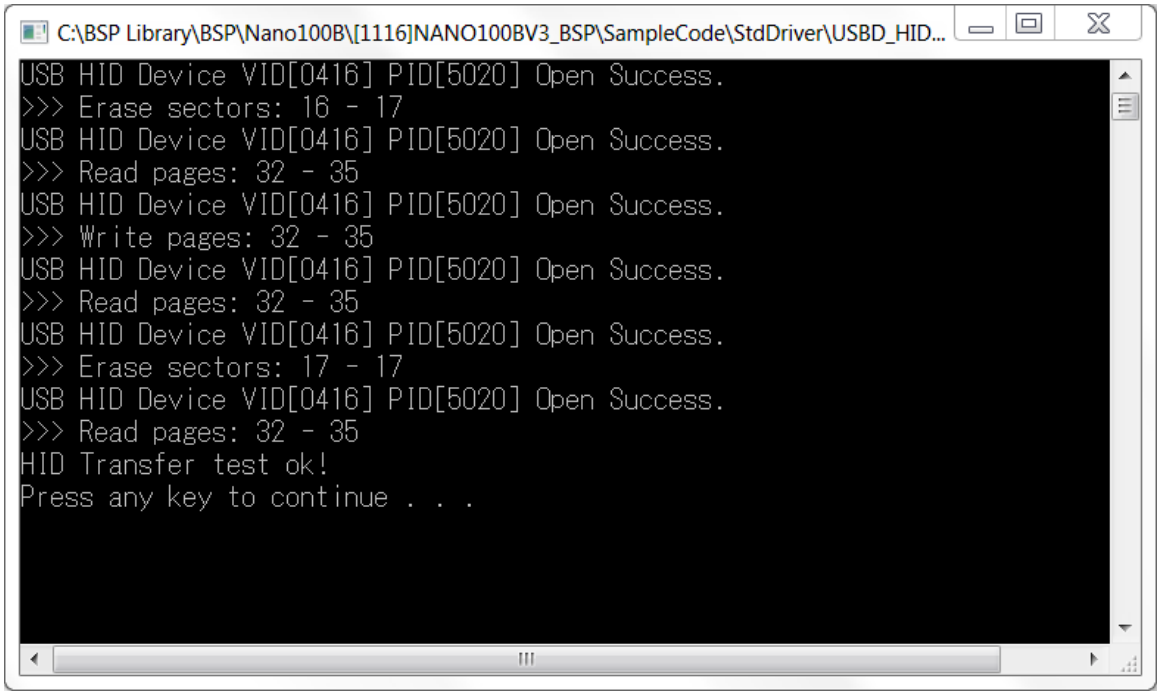


圖 18 HIDTransferTest.exe 執行結果

5.1.1.1 類別的特定請求

HID規格中所定義的標準裝置要求共有六個類別請求，依照請求進行不同的回應，但是並不是所有的命令都要處理回應。

```
void HID_ClassRequest(void)
{
   uint8_t buf[8];

   USBD_GetSetupPacket(buf);

   if (buf[0] & 0x80) { /* 資料傳輸方向是Device - > Host */
/* Device to host */
      switch (buf[1]) {
      case GET_REPORT:
      case GET_IDLE:
      case GET_PROTOCOL:
      default: {
         /* Setup error, stall the device */
         USBD_SetStall(0);
         break;
      }
      }
```

```
    } else {
/* Host to device */
        switch (buf[1]) {
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_PROTOCOL:
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}
```

5.1.1.2 Endpoint的配置

配置HID Transfer所要使用的端點，此範例共使用四個端點。端點0和端點1設定為控制型IN/OUT傳輸，用於USB標準及類別請求，使用此端點進行設定報告和獲取報告的資料傳輸；端點2設定為中斷型IN傳輸，使用EP2_Handler來處理HID輸入報告的資料傳輸；端點3設定為中斷型OUT傳輸，使用EP3_Handler來處理HID輸出報告的資料傳輸。

```
/* 端點的最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
```

```
#define SETUP_BUF_LEN    8
#define EP0_BUF_BASE     (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE     (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE     (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE     (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE

/* 端點地址 */
#define INT_IN_EP_NUM      0x01
#define INT_OUT_EP_NUM     0x02

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Interrupt OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
```

```
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}
```

EP2_Handler與EP3_Handler為端點2及端點3的中斷處理函數。EP2_Handler使用HID_SetInReport將報告傳回至主機，EP3_Handler透過HID_GetOutReport取出從主機所接收到的報告。

```
/* 端點2的中斷處理函數，每使用一次表示成功發送一個封包到主機 */
void EP2_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}
/* 端點3的中斷處理函數，每使用一次表示從主機收到一個封包 */
void EP3_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP3));
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}
```

5.1.1.3 定義各個描述元

HID類別會有特定的類別描述元，它是由描述HID介面的HID描述元及報告描述元所組成。HID描述元是放在組態描述元中，在列舉時用來辨識HID裝置，再經由報告描述元內定義的格式來傳送與接收資料。

```
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    /* 組態描述元 */
    LEN_CONFIG,                 /* bLength */
    DESC_CONFIG,                /* bDescriptorType */
    /* wTotalLength */
    (LEN_CONFIG+LEN_INTERFACE+LEN_HID+LEN_ENDPOINT*2) & 0x00FF,
    ((LEN_CONFIG+LEN_INTERFACE+LEN_HID+LEN_ENDPOINT*2) & 0xFF00) >> 8,
    0x01,                       /* bNumInterfaces */
    0x01,                       /* bConfigurationValue */
    0x00,                       /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/* bmAttributes */
    USBD_MAX_POWER,         /* MaxPower */

    /* 介面0 : HID Transfer介面描述元 */
    LEN_INTERFACE,              /* bLength */
    DESC_INTERFACE,         /* bDescriptorType */
```

```
    0x00,                       /* bInterfaceNumber */
    0x00,                       /* bAlternateSetting */
    0x02,                       /* bNumEndpoints */
    0x03,                       /* bInterfaceClass */
    0x00,                       /* bInterfaceSubClass */
    0x00,                       /* bInterfaceProtocol */
    0x00,                       /* iInterface */

    /* HID類別描述元 */
    LEN_HID,                    /* Size of this descriptor in UINT8s. */
    DESC_HID,                   /* HID descriptor type. */
    0x10, 0x01,                 /* HID Class Spec. release number. */
    0x00,                       /* H/W target country. */
    0x01,                       /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,               /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

    /* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,               /* bLength */
    DESC_ENDPOINT,              /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),             /* bEndpointAddress */
    EP_INT,                     /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */

    /* 端點3描述元: 中斷OUT，地址為INT_OUT_EP_NUM */
    LEN_ENDPOINT,               /* bLength */
    DESC_ENDPOINT,              /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),           /* bEndpointAddress */
    EP_INT,                     /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL         /* bInterval */
};

/* HID Transfer報告描述元 */
```

```
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,          // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,                // Usage (Vendor Usage 1)
    0xA1, 0x01,                // Collection (Application)
    0x19, 0x01,                // Usage Minimum
    0x29, 0x40,                // Usage Maximum //64 input usages total (0x01 to 0x40)
    0x15, 0x00,                // Logical Minimum (data bytes in the report may have minimum value = 0x00)
    0x26, 0xFF, 0x00,          // Logical Maximum (data bytes in the report may have maximum value = 0x00FF)
    0x75, 0x08,                // 報告大小(8) (bits)
    0x95, 0x40,                // 報告長度(64) (fields)
    0x81, 0x00,                // Input (Data, Array, Abs): Instantiates input packet fields based on the
                               // above report size, count, logical min/max, and usage.
    0x19, 0x01,                // Usage Minimum
    0x29, 0x40,                // Usage Maximum //64 output usages total (0x01 to 0x40)
    0x91, 0x00,                // Output (Data, Array, Abs): Instantiates output packet fields. Uses same
                               // report size and count as "Input" fields, since nothing new/different was
                               // specified to the parser since the "Input" item.
    0xC0                       // End Collection
};
```

5.1.2 HID Mouse

本章節介紹另一種HID裝置—滑鼠，它不需要搭配應用程式，可以直接從電腦上看到滑鼠游標移動。HID報告描述元的內容用來定義本範例的滑鼠功能，每個滑鼠裝置回報長度不一樣，此範例的報告總長度為4 Bytes，分別是按鍵、X及Y座標及滾輪，其中只控制滑鼠游標的XY座標，執行此範例時會在電腦上看見滑鼠的游標持續以繞圓方式移動。

5.1.2.1 類別的特定請求

此裝置的類別的特定請求與HID Transfer相同，請自行參考5.1.1.1類別的特定請求。

5.1.2.2 Endpoint的配置

配置HID Mouse所要使用的端點，此範例共使用三個端點。端點0和端點1設定為控制IN/OUT傳輸，用於USB標準及類別請求，使用此端點進行設定報告和獲取報告的資料傳輸；端點2設定為中斷IN傳輸，主機以週期性的輪詢方式，以取得報告內容。

```
/* 端點的最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    8

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
```

```
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN     EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN     EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN     EP2_MAX_PKT_SIZE

/* 端點地址 */
#define INT_IN_EP_NUM   0x01

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;
    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);
    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* Start to send IN data */
    g_u8EP2Ready = 1;
}
```

與HID Transfer的不同處為類別的功能，HID Mouse以送出X和Y座標資料來當作滑鼠的移動。HID Mouse類別功能函數HID_UpdateMouseData，此函數會以固定時間更新滑鼠的X和Y座標值，並且透過端點2的中斷型IN傳輸來傳送給主機。

```
/* 滑鼠移動方向 */
signed char mouse_table[] = {-16, -16, -16, 0, 16, 16, 16, 0};
```

```
uint8_t mouse_idx = 0;
uint8_t move_len, mouse_mode=1;

uint8_t volatile g_u8EP2Ready = 0;
void HID_UpdateMouseData(void)
{
   uint8_t *buf;

   if (g_u8EP2Ready) {
      buf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2));
      mouse_mode ^= 1;
      if (mouse_mode) {
         if (move_len > 14) {
            /* 更新滑鼠報告內容 */
            buf[0] = 0x00;
            buf[1] = mouse_table[mouse_idx & 0x07]; /* 滑鼠X座標 */
            buf[2] = mouse_table[(mouse_idx + 2) & 0x07]; /* 滑鼠Y座標 */
            buf[3] = 0x00;
            /* 執行下一個方向 */
            mouse_idx++;
            /* reset次數 */
            move_len = 0;
         }
      } else {
         /*不移動滑鼠 */
         buf[0] = buf[1] = buf[2] = buf[3] = 0;
      }
      /* 更新次數 */
      move_len++;
      g_u8EP2Ready = 0;
      /* 設定傳輸的資料長度並trigger IN transfer */
      USBD_SET_PAYLOAD_LEN(EP2, 4);
   }
}
```

5.1.2.3 定義各個描述元

在組態描述元裡面選擇HID協定InterfaceProtocol=HID_MOUSE(0x02)為滑鼠的設定，bInterfaceSubClass=0x0代表此類別不支援Boot介面，最後再設定HID報告描述元內的滑鼠的資料格式。

```
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
```

```
/* 組態描述元 */
    LEN_CONFIG,             /* bLength */
    DESC_CONFIG,            /* bDescriptorType */
    /* wTotalLength */
    LEN_CONFIG_AND_SUBORDINATE & 0x00FF,
    (LEN_CONFIG_AND_SUBORDINATE & 0xFF00) >> 8,
    0x01,                   /* bNumInterfaces */
    0x01,                   /* bConfigurationValue */
    0x00,                   /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5), /* bmAttributes */
    USBD_MAX_POWER,     /* MaxPower */

/* 介面0 : HID Mouse介面描述元*/
    LEN_INTERFACE,          /* bLength */
    DESC_INTERFACE,         /* bDescriptorType */
    0x00,                   /* bInterfaceNumber */
    0x00,                   /* bAlternateSetting */
    0x01,                   /* bNumEndpoints */
    0x03,                   /* bInterfaceClass */
    0x00,                   /* bInterfaceSubClass */
    HID_MOUSE,              /* bInterfaceProtocol */
    0x00,                   /* iInterface */

    /* HID描述元 */
    LEN_HID,                /* Size of this descriptor in UINT8s. */
    DESC_HID,               /* HID descriptor type. */
    0x10, 0x01,             /* HID Class Spec. release number. */
    0x00,                   /* H/W target country. */
    0x01,                   /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,           /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_MouseReportDescriptor) & 0x00FF,
    (sizeof(HID_MouseReportDescriptor) & 0xFF00) >> 8,

    /* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,           /* bLength */
    DESC_ENDPOINT,          /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),         /* bEndpointAddress */
    EP_INT,                 /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
```

```
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL       /* bInterval */
};

/* HID Mouse報告描述元 */
uint8_t HID_MouseReportDescriptor[] = {
    0x05, 0x01,                 /* Usage Page(Generic Desktop Controls) */
    0x09, 0x02,                 /* Usage(Mouse) */
    0xA1, 0x01,                  /* Collection(Application) */
    0x09, 0x01,                  /* Usage(Pointer) */
    0xA1, 0x00,                  /* Collection(Physical) */
    0x05, 0x09,                  /* Usage Page(Button) */
    0x19, 0x01,                  /* Usage Minimum(0x1) */
    0x29, 0x03,                  /* Usage Maximum(0x3) */
    0x15, 0x00,                  /* Logical Minimum(0x0) */
    0x25, 0x01,                  /* Logical Maximum(0x1) */
/* 按鍵輸入, 不使用 */
    0x75, 0x01,                 /* Report Size(0x1) */
    0x95, 0x03,                 /* Report Count(0x3) */
    0x81, 0x02,                 /* Input(3 button bit) */
    0x75, 0x05,                 /* Report Size(0x5) */
    0x95, 0x01,                 /* Report Count(0x1) */
    0x81, 0x01,                 /* Input(5 bit padding) */
    0x05, 0x01,                 /* Usage Page(Generic Desktop Controls) */
/* X,Y座標輸入 */
    0x09, 0x30,                 /* Usage(X) */
    0x09, 0x31,                 /* Usage(Y) */
    0x09, 0x38,                 /* Usage(Wheel) */
    0x15, 0x81,                 /* Logical Minimum(0x81)(-127) */
    0x25, 0x7F,                  /* Logical Maximum(0x7F)(127) */
/* 滾輪輸入 */
    0x75, 0x08,                 /* Report Size(0x8) */
    0x95, 0x03,                 /* Report Count(0x3) */
    0x81, 0x06,                 /* Input(1 byte wheel) */
    0xC0,                       /* End Collection */
    0xC0,                       /* End Collection */
};
```

5.1.3 HID Keyboard

HID Keyboard裝置會以一根GPIO Pin輸出為High或Low來示範兩種鍵盤行為，一種為完全不按任何鍵；另一種則是普通的按鍵行為，此範例示範一直按住A的按鍵。

5.1.3.1 類別的特定請求

此裝置為HID裝置的第三個範例，所有的HID裝置所要回應的類別請求皆相同，因此可自行參考5.1.1.1類別的特定請求。

5.1.3.2 Endpoint的配置

配置HID Keyboard所要使用的端點，此範例共使用三個端點。端點0和端點1設定為控制IN/OUT傳輸，用於USB標準及類別請求，使用此端點進行設定報告和獲取報告的資料傳輸；端點2設定為中斷IN傳輸，主機以週期性的輪詢方式以取得報告內容。 這裡的Endpoint的配置與5.1.2HID Mouse相同，因此不再次說明。

HID Keyboard需要送出 8 Bytes給主機端。HID_UpdateKbData透過端點2的中斷型IN來傳輸，模擬一直按住鍵盤上的按鍵A。

```
void HID_UpdateKbData(void)
{
    int32_t i;
    uint8_t *buf;
    uint32_t key = 0xF;
    static uint32_t preKey;

    if(g_u8EP2Ready){
        buf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2));

        /* If GPB15 = 0, just report it is key 'a' */
        key = (PB->PIN & (1 << 15)) ? 0 : 1;

        if(key == 0){
            for(i = 0; i < 8; i++){
                buf[i] = 0; /* 不按任何鍵 */
            }
            if(key != preKey){
                /* 設定傳輸的資料長度並trigger IN transfer */
                USBD_SET_PAYLOAD_LEN(EP2, 8);
            }
        }
        else{
            preKey = key;
            /* 按下按鍵A */
            buf[2] = 0x04; /* Key A */
            /* 設定傳輸的資料長度並trigger IN transfer */
            USBD_SET_PAYLOAD_LEN(EP2, 8);
        }
```

```
    }
}
```

5.1.3.3 定義各個描述元

在組態描述元裡面選擇HID協定InterfaceProtocol=HID_KEYBOARD(0x01)為鍵盤的設定，bInterfaceSubClass=0x0代表此類別不支援Boot介面，最後再設定HID報告描述元內的鍵盤的資料格式。

```
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    /* 組態描述元 */
    LEN_CONFIG,                 /* bLength */
    DESC_CONFIG,                /* bDescriptorType */
    /* wTotalLength */
    LEN_CONFIG_AND_SUBORDINATE & 0x00FF,
    (LEN_CONFIG_AND_SUBORDINATE & 0xFF00) >> 8,
    0x01,                       /* bNumInterfaces */
    0x01,                       /* bConfigurationValue */
    0x00,                       /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/* bmAttributes */
    USBD_MAX_POWER,             /* MaxPower */

    /* 介面0 : HID Keyboard介面描述元*/
    LEN_INTERFACE,              /* bLength */
    DESC_INTERFACE,             /* bDescriptorType */
    0x00,                       /* bInterfaceNumber */
    0x00,                       /* bAlternateSetting */
    0x01,                       /* bNumEndpoints */
    0x03,                       /* bInterfaceClass */
    0x00,                       /* bInterfaceSubClass */
    HID_KEYBOARD,               /* bInterfaceProtocol */
    0x00,                       /* iInterface */

    /* HID描述元 */
    LEN_HID,                    /* Size of this descriptor in UINT8s. */
    DESC_HID,                   /* HID descriptor type. */
    0x10, 0x01,                 /* HID Class Spec. release number. */
    0x00,                       /* H/W target country. */
    0x01,                       /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,               /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
```

```
    (sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,

    /* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,            /* bLength */
    DESC_ENDPOINT,           /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),    /* bEndpointAddress */
    EP_INT,                  /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL    /* bInterval */
};

/* HID Keyboard報告描述元 */
uint8_t HID_KeyboardReportDescriptor[] =
{
    0x05, 0x01,              /* Usage Page(Generic Desktop Controls) */
    0x09, 0x06,              /* Usage(Keyboard) */
    0xA1, 0x01,              /* Collection(Application) */
    0x05, 0x07,              /* Usage Page(Keyboard/Keypad) */
    0x19, 0xE0,              /* Usage Minimum(0xE0) */
    0x29, 0xE7,              /* Usage Maximum(0xE7) */
    0x15, 0x00,              /* Logical Minimum(0x0) */
    0x25, 0x01,              /* Logical Maximum(0x1) */
/* 鍵盤的輸入報告(8 Bytes) */
    0x75, 0x01,              /* Report Size(0x1) */
    0x95, 0x08,              /* Report Count(0x8) */
    0x81, 0x02,              /* Input (Data) => Modifier byte */
    0x95, 0x01,              /* Report Count(0x1) */
    0x75, 0x08,              /* Report Size(0x8) */
    0x81, 0x01,              /* Input (Constant) => Reserved byte */
    0x95, 0x05,              /* Report Count(0x5) */
    0x75, 0x01,              /* Report Size(0x1) */
    0x05, 0x08,              /* Usage Page(LEDs) */
    0x19, 0x01,              /* Usage Minimum(0x1) */
    0x29, 0x05,              /* Usage Maximum(0x5) */
    0x91, 0x02,              /* Output (Data) => LED report */
    0x95, 0x01,              /* Report Count(0x1) */
    0x75, 0x03,              /* Report Size(0x3) */
    0x91, 0x01,              /* Output (Constant) => LED report padding */
    0x95, 0x06,              /* Report Count(0x6) */
```

```
    0x75, 0x08,                 /* Report Size(0x8) */
    0x15, 0x00,                 /* Logical Minimum(0x0) */
    0x25, 0x65,                 /* Logical Maximum(0x65) */
    0x05, 0x07,                 /* Usage Page(Keyboard/Keypad) */
    0x19, 0x00,                 /* Usage Minimum(0x0) */
    0x29, 0x65,                 /* Usage Maximum(0x65) */
    0x81, 0x00,                 /* Input (Data) */
    0xC0,                       /* End Collection */
};
```

5.2 MSC

USB大容量儲存類別(Mass Storage Class)是一種電腦和行動裝置之間的傳輸協議，它允許透過USB裝置來連接主機，使兩者之間進行檔案資料傳輸。此範例使用Bulk Only傳輸規範，僅支持巨量型傳輸，Bulk Only傳輸分為三個階段，命令階段(CBW)、資料階段(IN或OUT)和狀態階段(CSW)，並且定義兩個專屬的類別請求：Bulk-Only Mass Storage Reset及Get Max LUN。另外，儲存裝置主機和裝置之間使用UFI命令，詳細內容可參考USB Mass Storage Class的規格。接下來將介紹USB MSC隨身碟的實作。

5.2.1 類別的特定請求

MSC_ClassRequest用來回應MSC的兩個類別請求，一個是Bulk-Only Mass Storage Reset是Reset介面，另一個是Get Max LUN用來確定邏輯單元的數量。MSC_ProcessCmd則用來處理UFI命令。

```
void MSC_ClassRequest(void)
{
   uint8_t buf[8];

   USBD_GetSetupPacket(buf);

   if (buf[0] & 0x80) { /* request data transfer direction */
      // Device to host
      switch (buf[1]) {
      case GET_MAX_LUN: {
         if ((((buf[3]<<8)+buf[2]) == 0) && (((buf[5]<<8)+buf[4]) == 0) &&
            (((buf[7]<<8)+buf[6]) == 1)) {
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 1);
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
         }
```

```
        else {/* Invalid Get MaxLun command */
            USBD_SetStall(0);
        }
        break;
    }
    default: {
        /* Setup error, stall the device */
        USBD_SetStall(0);
        break;
    }
    }
} else {
/* Host to device */
    switch (buf[1]) {
    case BULK_ONLY_MASS_STORAGE_RESET: {
        if ((((buf[3]<<8)+buf[2]) == 0) && (((buf[5]<<8)+buf[4]) == 0) &&
            (((buf[7]<<8)+buf[6]) == 0)) {
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            USBD_LockEpStall(0);

            /* Clear ready */
            USBD->EP[EP2].CFG |= USBD_CFG_CLRRDY_Msk;
            USBD->EP[EP3].CFG |= USBD_CFG_CLRRDY_Msk;

            /* Prepare to receive the CBW */
            g_u8EP3Ready = 0;
            g_u8BulkState = BULK_CBW;

            USBD_SET_DATA1(EP3);
            USBD_SET_EP_BUF_ADDR(EP3, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
        }
        else {/* Invalid Reset command */
            USBD_SetStall(0);
        }
        break;
    }
    default: {
        /* Setup error, stall the device */
        USBD_SetStall(0);
```

```
            break;
        }
        }
    }
}

void MSC_ProcessCmd(void)
{
    uint8_t u8Len;
    int32_t i;
    uint32_t Hcount, Dcount;

    if (g_u8EP3Ready) {
        g_u8EP3Ready = 0;
        if (g_u8BulkState == BULK_CBW) {
            u8Len = USBD_GET_PAYLOAD_LEN(EP3);

            /* Check Signature & length of CBW */
            /* Bulk Out buffer */
            if ((*(uint32_t *) ((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0) != CBW_SIGNATURE) ||
                (u8Len != 31))
            {
                /* Invalid CBW */
                g_u8Prevent = 1;
                USBD_SET_EP_STALL(EP2);
                USBD_SET_EP_STALL(EP3);
                USBD_LockEpStall((1 << EP2) | (1 << EP3));
                return;
            }

            /* Get the CBW */
            for (i = 0; i < u8Len; i++)
                *((uint8_t *) (&g_sCBW.dCBWSignature) + i) =
                *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0 + i);

            /* Prepare to echo the tag from CBW to CSW */
            g_sCSW.dCSWTag = g_sCBW.dCBWTag;
            Hcount = g_sCBW.dCBWDataTransferLength;

            /* Parse Op-Code of CBW */
            switch (g_sCBW.u8OPCode) {
```

```
        case UFI_PREVENT_ALLOW_MEDIUM_REMOVAL: {
            if (g_sCBW.au8Data[2] & 0x01) {
                g_au8SenseKey[0] = 0x05;  //INVALID COMMAND
                g_au8SenseKey[1] = 0x24;
                g_au8SenseKey[2] = 0;
                g_u8Prevent = 1;
            } else
                g_u8Prevent = 0;
            g_u8BulkState = BULK_IN;
            MSC_AckCmd();
            return;
        }
        case UFI_TEST_UNIT_READY: {
            if (Hcount != 0) {
                if (g_sCBW.bmCBWFlags == 0) {   /* Ho > Dn (Case 9) */
                    g_u8Prevent = 1;
                    USBD_SET_EP_STALL(EP3);
                    g_sCSW.bCSWStatus = 0x1;
                    g_sCSW.dCSWDataResidue = Hcount;
                }
            }
            else {  /* Hn == Dn (Case 1) */
                if (g_u8Remove) {
                    g_sCSW.dCSWDataResidue = 0;
                    g_sCSW.bCSWStatus = 1;
                    g_au8SenseKey[0] = 0x02;    /* Not ready */
                    g_au8SenseKey[1] = 0x3A;
                    g_au8SenseKey[2] = 0;
                    g_u8Prevent = 1;
                }
                else {
                    g_sCSW.dCSWDataResidue = 0;
                    g_sCSW.bCSWStatus = 0;
                }
            }
            g_u8BulkState = BULK_IN;
            MSC_AckCmd();
            return;
        }
        case UFI_START_STOP: {
            if ((g_sCBW.au8Data[2] & 0x03) == 0x2) {
```

```
            g_u8Remove = 1;
        }
    }
    case UFI_VERIFY_10: {
        g_u8BulkState = BULK_IN;
        MSC_AckCmd();
        return;
    }
    case UFI_REQUEST_SENSE: {
        if ((Hcount > 0) && (Hcount <= 18)){
            MSC_RequestSense();
            USBD_SET_PAYLOAD_LEN(EP2, Hcount);
            g_u8BulkState = BULK_IN;
            g_sCSW.bCSWStatus = 0;
            g_sCSW.dCSWDataResidue = 0;
            return;
        }
        else {
            USBD_SET_EP_STALL(EP2);
            g_u8Prevent = 1;
            g_sCSW.bCSWStatus = 0x01;
            g_sCSW.dCSWDataResidue = 0;
            g_u8BulkState = BULK_IN;
            MSC_AckCmd();
            return;
        }
    }
    case UFI_READ_FORMAT_CAPACITY: {
        if (g_u32Length == 0) {
            g_u32Length = g_sCBW.dCBWDataTransferLength;
            g_u32Address = MassCMD_BUF;
        }
        MSC_ReadFormatCapacity();
        g_u8BulkState = BULK_IN;
        if (g_u32Length > 0) {
            if (g_u32Length > EP2_MAX_PKT_SIZE)
                g_u8Size = EP2_MAX_PKT_SIZE;
            else
                g_u8Size = g_u32Length;

            /* Bulk IN buffer */
```

```
            USBD_MemCopy((uint8_t *)(USBD_BUF_BASE + g_u32BulkBuf1),
                                (uint8_t *)g_u32Address, g_u8Size);

            g_u32Address += g_u8Size;
            USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf0);
            MSC_Read();
        }
        return;
    }
    case UFI_READ_CAPACITY: {
        if (g_u32Length == 0) {
            g_u32Length = g_sCBW.dCBWDataTransferLength;
            g_u32Address = MassCMD_BUF;
        }

        MSC_ReadCapacity();
        g_u8BulkState = BULK_IN;
        if (g_u32Length > 0) {
            if (g_u32Length > EP2_MAX_PKT_SIZE)
                g_u8Size = EP2_MAX_PKT_SIZE;
            else
                g_u8Size = g_u32Length;

            /* Bulk IN buffer */
            USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                                (uint8_t *)g_u32Address, g_u8Size);

            g_u32Address += g_u8Size;
            USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf0);
            MSC_Read();
        }
        return;
    }
    case UFI_MODE_SELECT_6:
    case UFI_MODE_SELECT_10: {
        g_u32Length = g_sCBW.dCBWDataTransferLength;
        g_u32Address = MassCMD_BUF;

        if (g_u32Length > 0) {
            USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
            g_u8BulkState = BULK_OUT;
```

```
        }
        return;
    }
    case UFI_MODE_SENSE_6: {
        *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1+0) = 0x3;
        *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1+1) = 0x0;
        *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1+2) = 0x0;
        *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1+3) = 0x0;

        USBD_SET_PAYLOAD_LEN(EP2, 4);
        g_u8BulkState = BULK_IN;
        g_sCSW.bCSWStatus = 0;
        g_sCSW.dCSWDataResidue = Hcount - 4;;
        return;
    }
    case UFI_MODE_SENSE_10: {
        if (g_u32Length == 0) {
            g_u32Length = g_sCBW.dCBWDataTransferLength;
            g_u32Address = MassCMD_BUF;
        }
        MSC_ModeSense10();
        g_u8BulkState = BULK_IN;
        if (g_u32Length > 0) {
            if (g_u32Length > EP2_MAX_PKT_SIZE)
                g_u8Size = EP2_MAX_PKT_SIZE;
            else
                g_u8Size = g_u32Length;
            /* Bulk IN buffer */
            USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                    (uint8_t *)g_u32Address, g_u8Size);

            g_u32Address += g_u8Size;

            USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf0);
            MSC_Read();
        }
        return;
    }
    case UFI_INQUIRY: {
        if ((Hcount > 0) && (Hcount <= 36)){
            /* Bulk IN buffer */
```

```
            USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                    (uint8_t *)g_au8InquiryID, Hcount);
            USBD_SET_PAYLOAD_LEN(EP2, Hcount);
            g_u8BulkState = BULK_IN;
            g_sCSW.bCSWStatus = 0;
            g_sCSW.dCSWDataResidue = 0;
            return;
        }
        else {
            USBD_SET_EP_STALL(EP2);
            g_u8Prevent = 1;
            g_sCSW.bCSWStatus = 0x01;
            g_sCSW.dCSWDataResidue = 0;
            g_u8BulkState = BULK_IN;
            MSC_AckCmd();
            return;
        }
    }
    case UFI_READ_12:
    case UFI_READ_10: {
        /* Check if it is a new transfer */
        if(g_u32Length == 0) {

            Dcount = (get_be32(&g_sCBW.au8Data[4])>>8) * 512;
            if (g_sCBW.bmCBWFlags == 0x80) {    /* IN */
                if (Hcount == Dcount) { /* Hi == Di (Case 6)*/
                }
                /* Hn < Di (Case 2) || Hi < Di (Case 7) */
                else if (Hcount < Dcount) {
                    if (Hcount) {   /* Hi < Di (Case 7) */
                        g_u8Prevent = 1;
                        g_sCSW.bCSWStatus = 0x01;
                        g_sCSW.dCSWDataResidue = 0;
                    }
                    else {  /* Hn < Di (Case 2) */
                        g_u8Prevent = 1;
                        g_sCSW.bCSWStatus = 0x01;
                        g_sCSW.dCSWDataResidue = 0;
                        g_u8BulkState = BULK_IN;
                        MSC_AckCmd();
                        return;
```

```
            }
        }
        /* Hi > Dn (Case 4) || Hi > Di (Case 5) */
        else if (Hcount > Dcount) {
            g_u8Prevent = 1;
            g_sCSW.bCSWStatus = 0x01;
            g_sCSW.dCSWDataResidue = 0;
        }
    }
    else {  /* Ho <> Di (Case 10) */
        g_u8Prevent = 1;
        USBD_SET_EP_STALL(EP3);
        g_sCSW.bCSWStatus = 0x01;
        g_sCSW.dCSWDataResidue = Hcount;
        g_u8BulkState = BULK_IN;
        MSC_AckCmd();
        return;
    }
}

/* Get LBA address */
g_u32Address = get_be32(&g_sCBW.au8Data[0]);
g_u32LbaAddress = g_u32Address * UDC_SECTOR_SIZE;
g_u32Length = g_sCBW.dCBWDataTransferLength;
g_u32BytesInStorageBuf = g_u32Length;

i = g_u32Length;
if (i > STORAGE_BUFFER_SIZE)
    i = STORAGE_BUFFER_SIZE;

MSC_ReadMedia(g_u32Address * UDC_SECTOR_SIZE, i,
              (uint8_t *)STORAGE_DATA_BUF);
g_u32BytesInStorageBuf = i;
g_u32LbaAddress += i;

g_u32Address = STORAGE_DATA_BUF;

/* Indicate the next packet should be Bulk IN Data packet */
g_u8BulkState = BULK_IN;
if (g_u32BytesInStorageBuf > 0) {
    /* Set the packet size */
```

```
            if (g_u32BytesInStorageBuf > EP2_MAX_PKT_SIZE)
                g_u8Size = EP2_MAX_PKT_SIZE;
            else
                g_u8Size = g_u32BytesInStorageBuf;

            /* Prepare the first data packet (DATA1) */
            /* Bulk IN buffer */
            USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                    (uint8_t *)g_u32Address, g_u8Size);
            g_u32Address += g_u8Size;

            /* kick - start */
            USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf1);
            /* Trigger to send out the data packet */
            USBD_SET_PAYLOAD_LEN(EP2, g_u8Size);
            g_u32Length -= g_u8Size;
            g_u32BytesInStorageBuf -= g_u8Size;
        }
        return;
    }
    case UFI_WRITE_12:
    case UFI_WRITE_10: {
        if (g_u32Length == 0) {
            Dcount = (get_be32(&g_sCBW.au8Data[4])>>8) * 512;
            if (g_sCBW.bmCBWFlags == 0x00) {    /* OUT */
                if (Hcount == Dcount) { /* Ho == Do (Case 12)*/
                    g_sCSW.dCSWDataResidue = 0;
                    g_sCSW.bCSWStatus = 0;
                }
                /* Hn < Do (Case 3) || Ho < Do (Case 13) */
                else if (Hcount < Dcount) {
                    g_u8Prevent = 1;
                    g_sCSW.dCSWDataResidue = 0;
                    g_sCSW.bCSWStatus = 0x1;
                    if (Hcount == 0) {  /* Hn < Do (Case 3) */
                        g_u8BulkState = BULK_IN;
                        MSC_AckCmd();
                        return;
                    }
                }
                else if (Hcount > Dcount) { /* Ho > Do (Case 11) */
```

```
                g_u8Prevent = 1;
                g_sCSW.dCSWDataResidue = 0;
                g_sCSW.bCSWStatus = 0x1;
            }
            g_u32Length = g_sCBW.dCBWDataTransferLength;
            g_u32Address = STORAGE_DATA_BUF;
            g_u32DataFlashStartAddr = get_be32(&g_sCBW.au8Data[0]) *
                                      UDC_SECTOR_SIZE;
        }
        else {  /* Hi <> Do (Case 8) */
            g_u8Prevent = 1;
            g_sCSW.dCSWDataResidue = Hcount;
            g_sCSW.bCSWStatus = 0x1;
            USBD_SET_EP_STALL(EP2);
            g_u8BulkState = BULK_IN;
            MSC_AckCmd();
            return;
        }
    }

    if ((g_u32Length > 0)) {
        USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
        g_u8BulkState = BULK_OUT;
    }
    return;
}
case UFI_READ_16: {
    USBD_SET_EP_STALL(EP2);
    g_u8Prevent = 1;
    g_sCSW.bCSWStatus = 0x01;
    g_sCSW.dCSWDataResidue = 0;
    g_u8BulkState = BULK_IN;
    MSC_AckCmd();
    return;
}
default: {
    /* Unsupported command */
    g_au8SenseKey[0] = 0x05;
    g_au8SenseKey[1] = 0x20;
    g_au8SenseKey[2] = 0x00;
```

```
            /* If CBW request for data phase, just return zero packet to end data phase */
            if (g_sCBW.dCBWDataTransferLength > 0) {
               /* Data Phase, zero/short packet */
               if ((g_sCBW.bmCBWFlags & 0x80) != 0) {
                  /* Data-In */
                  g_u8BulkState = BULK_IN;
                  USBD_SET_PAYLOAD_LEN(EP2, 0);
               }
            } else {
               /* Status Phase */
               g_u8BulkState = BULK_IN;
               MSC_AckCmd();
            }
            return;
         }
         }
      } else if (g_u8BulkState == BULK_OUT) {
         switch (g_sCBW.u8OPCode) {
         case UFI_WRITE_12:
         case UFI_WRITE_10:
         case UFI_MODE_SELECT_6:
         case UFI_MODE_SELECT_10: {
            MSC_Write();
            return;
         }
         }
      }
   }
}
```

5.2.2 Endpoint 的配置

配置Mass Storage所要使用的端點，此範例共使用四個端點。端點0和端點1設定為控制型IN/OUT傳輸，用於USB標準及類別請求；端點2設定為巨量型IN傳輸，用於大容量儲存Bulk-Only傳輸協定的CSW傳輸；端點3設定為巨量型OUT傳輸，用於大容量儲存Bulk-Only傳輸協定的CBW傳輸。

```
/* 端點的最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
```

```
/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE     0
#define SETUP_BUF_LEN      8
#define EP0_BUF_BASE       (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN        EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE       (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN        EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE       (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN        EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE       (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN        EP3_MAX_PKT_SIZE

/* 端點地址 */
#define BULK_IN_EP_NUM     0x02
#define BULK_OUT_EP_NUM    0x03

void MSC_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 2 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 3 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
```

```
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);

    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /*****************************************************/
    g_u32BulkBuf0 = EP3_BUF_BASE;
    g_u32BulkBuf1 = EP2_BUF_BASE;

    g_sCSW.dCSWSignature = CSW_SIGNATURE;
    g_TotalSectors = DATA_FLASH_STORAGE_SIZE / UDC_SECTOR_SIZE;
}
```

5.2.3 定義各個描述元

裝置、組態、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入MSC類別的各種描述元。

```
/* USB 組態描述元，它包含介面及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    /* 組態描述元 */
    LEN_CONFIG,                         // bLength
    DESC_CONFIG,                        // bDescriptorType
    (LEN_CONFIG+LEN_INTERFACE+LEN_ENDPOINT*2), 0x00,        // wTotalLength
    0x01,                               // bNumInterfaces
    0x01,                               // bConfigurationValue
    0x00,                               // iConfiguration
    0xC0,                               // bmAttributes
    0x32,                               // MaxPower

    /* 介面0 : MSC 介面描述元 */
    LEN_INTERFACE,                      // bLength
    DESC_INTERFACE,                     // bDescriptorType
    0x01,                               // bInterfaceNumber
    0x00,                               // bAlternateSetting
    0x02,                               // bNumEndpoints
    0x08,                               // bInterfaceClass
    0x06,                               // bInterfaceSubClass
    0x50,                               // bInterfaceProtocol
    0x00,                               // iInterface

    /* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
```

```
    LEN_ENDPOINT,                       // bLength
    DESC_ENDPOINT,                      // bDescriptorType
    (EP_INPUT | BULK_IN_EP_NUM),    // bEndpointAddress
    EP_BULK,                            // bmAttributes
    EP2_MAX_PKT_SIZE, 0x00,             // wMaxPacketSize
    0x00,                               // bInterval

    /* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                       // bLength
    DESC_ENDPOINT,                      // bDescriptorType
    (EP_OUTPUT | BULK_OUT_EP_NUM),     //bEndpointAddress
    EP_BULK,                            // bmAttributes
    EP3_MAX_PKT_SIZE, 0x00,             // wMaxPacketSize
    0x00                                // bInterval
};
```

Mass Storage裝置使用Data flash來當作此裝置的儲存容量，並對Data Flash進行讀取或寫入資料。使用者可以設定g_TotalSectors來更改Mass Storage的容量，而讀寫資料分別可以透過MSC_Read函數裡面的MSC_ReadMedia函數以及MSC_Write函數裡面的DataFlashWrite函數，用以設計新的Mass Storage。

```
void MSC_Init(void)
{
    ...
    g_TotalSectors = DATA_FLASH_STORAGE_SIZE / UDC_SECTOR_SIZE;
}

void MSC_Read(void)
{
    uint32_t u32Len;

    if (USBD_GET_EP_BUF_ADDR(EP2) == g_u32BulkBuf1)
        USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf0);
    else
        USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf1);

    /* Trigger to send out the data packet */
    USBD_SET_PAYLOAD_LEN(EP2, g_u8Size);

    g_u32Length -= g_u8Size;
    g_u32BytesInStorageBuf -= g_u8Size;
```

```
    if (g_u32Length) {
        if (g_u32BytesInStorageBuf) {
            /* Prepare next data packet */
            g_u8Size = EP2_MAX_PKT_SIZE;
            if (g_u8Size > g_u32Length)
                g_u8Size = g_u32Length;

            if (USBD_GET_EP_BUF_ADDR(EP2) == g_u32BulkBuf1)
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0),
                (uint8_t *)g_u32Address, g_u8Size);
            else
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                (uint8_t *)g_u32Address, g_u8Size);
                g_u32Address += g_u8Size;
        } else {
            u32Len = g_u32Length;
            if (u32Len > STORAGE_BUFFER_SIZE)
                u32Len = STORAGE_BUFFER_SIZE;

            MSC_ReadMedia(g_u32LbaAddress, u32Len, (uint8_t *)STORAGE_DATA_BUF);
            g_u32BytesInStorageBuf = u32Len;
            g_u32LbaAddress += u32Len;
            g_u32Address = STORAGE_DATA_BUF;

            /* Prepare next data packet */
            g_u8Size = EP2_MAX_PKT_SIZE;
            if (g_u8Size > g_u32Length)
                g_u8Size = g_u32Length;

            if (USBD_GET_EP_BUF_ADDR(EP2) == g_u32BulkBuf1)
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0),
                (uint8_t *)g_u32Address, g_u8Size);
            else
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                (uint8_t *)g_u32Address, g_u8Size);
                g_u32Address += g_u8Size;
        }
    }
}

void MSC_Write(void)
```

```
{
    uint32_t lba, len;

    if (g_u32Length > EP3_MAX_PKT_SIZE) {
        if (USBD_GET_EP_BUF_ADDR(EP3) == g_u32BulkBuf0) {
            USBD_SET_EP_BUF_ADDR(EP3, g_u32BulkBuf1);
            USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
            USBD_MemCopy((uint8_t *)g_u32Address, (uint8_t *)((uint32_t)USBD_BUF_BASE +
            g_u32BulkBuf0), EP3_MAX_PKT_SIZE);
        } else {
            USBD_SET_EP_BUF_ADDR(EP3, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
            USBD_MemCopy((uint8_t *)g_u32Address, (uint8_t *)((uint32_t)USBD_BUF_BASE +
            g_u32BulkBuf1), EP3_MAX_PKT_SIZE);
        }

        g_u32Address += EP3_MAX_PKT_SIZE;
        g_u32Length -= EP3_MAX_PKT_SIZE;

        /* Buffer full. Writer it to storage first. */
        if (g_u32Address >= (STORAGE_DATA_BUF + STORAGE_BUFFER_SIZE)) {
            DataFlashWrite(g_u32DataFlashStartAddr, STORAGE_BUFFER_SIZE,
            (uint32_t)STORAGE_DATA_BUF);

            g_u32Address = STORAGE_DATA_BUF;
            g_u32DataFlashStartAddr += STORAGE_BUFFER_SIZE;
        }
    } else {
        if (USBD_GET_EP_BUF_ADDR(EP3) == g_u32BulkBuf0)
            USBD_MemCopy((uint8_t *)g_u32Address, (uint8_t *)((uint32_t)USBD_BUF_BASE +
            g_u32BulkBuf0), g_u32Length);
        else
            USBD_MemCopy((uint8_t *)g_u32Address, (uint8_t *)((uint32_t)USBD_BUF_BASE +
            g_u32BulkBuf1), g_u32Length);
            g_u32Address += g_u32Length;
            g_u32Length = 0;

        if ((g_sCBW.u8OPCode == UFI_WRITE_10) || (g_sCBW.u8OPCode == UFI_WRITE_12)) {
            lba = get_be32(&g_sCBW.au8Data[0]);
            len = g_sCBW.dCBWDataTransferLength;
```

```
            len = lba * UDC_SECTOR_SIZE + g_sCBW.dCBWDataTransferLength
                - g_u32DataFlashStartAddr;
            if (len)
                DataFlashWrite(g_u32DataFlashStartAddr, len, (uint32_t)STORAGE_DATA_BUF);
        }

        g_u8BulkState = BULK_IN;
        MSC_AckCmd();
    }
}
```

5.3 CDC

USB通信裝置類(CDC)提供USB轉UART功能，需要搭配驅動程式來模擬串列通訊埠。大部分的作業系統都帶有支持CDC類別的裝置驅動軟體，主機透過安裝驅動程式來辨識USB裝置，驅動程式能讓對應的裝置運作，並讓主控端的中央處理器能進行各項應用及操作，當裝置端與主控端的連線正確時，就可進行資料傳輸與通訊等功能。

Windows載入INF檔案中的usbser.sys來當作CDC通訊裝置的驅動程式，INF檔案中必須包含USB裝置的PID及VID，PID及VID是由USB設計論壇(USB-IF)授權。當驅動程式辨識USB裝置後，一個新的串列裝置就會出現在裝置管理員中，系統會自動為它指定一個可用的埠號。

第一次執行此範例與電腦連接時需要安裝驅動程式(圖 19)，透過此範例搭配的INF檔案來安裝CDC驅動程式(圖 20)，成功安裝驅動程式後，硬體的管理員中就會出現一個新的Nuvoton虛擬COM Port，如圖 21使用COM 59。圖 22使用者可以利用網路上免費的終端機連線工具來與COM Port做資料傳輸。執行此範例使用者可以在終端機連線工具上看到自己從鍵盤輸入的特定字元，圖 23為執行VOCM範例的執行結果。本章節將介紹使用Nano100B來實現USB Virtual COM Port(VCOM)。

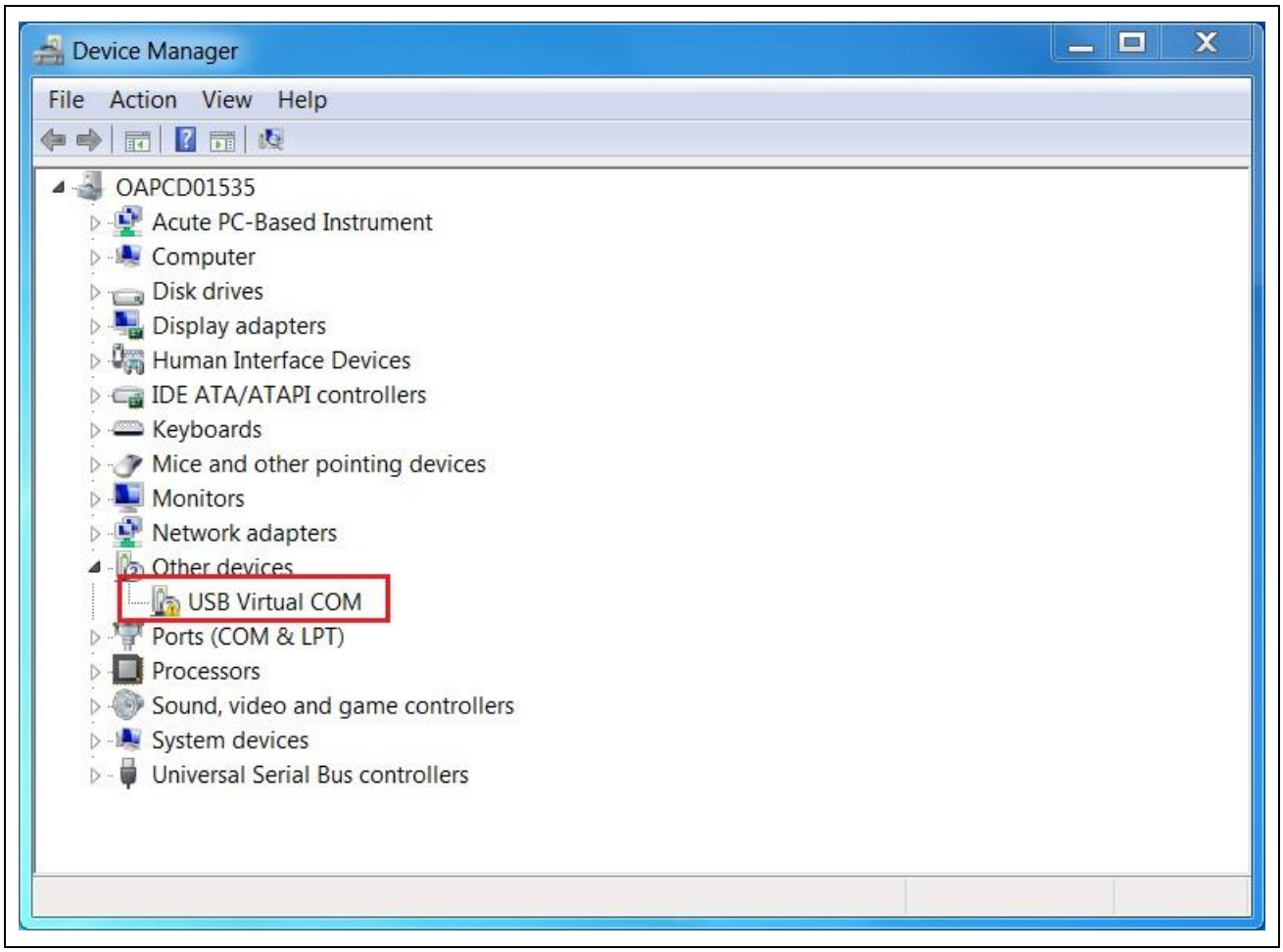
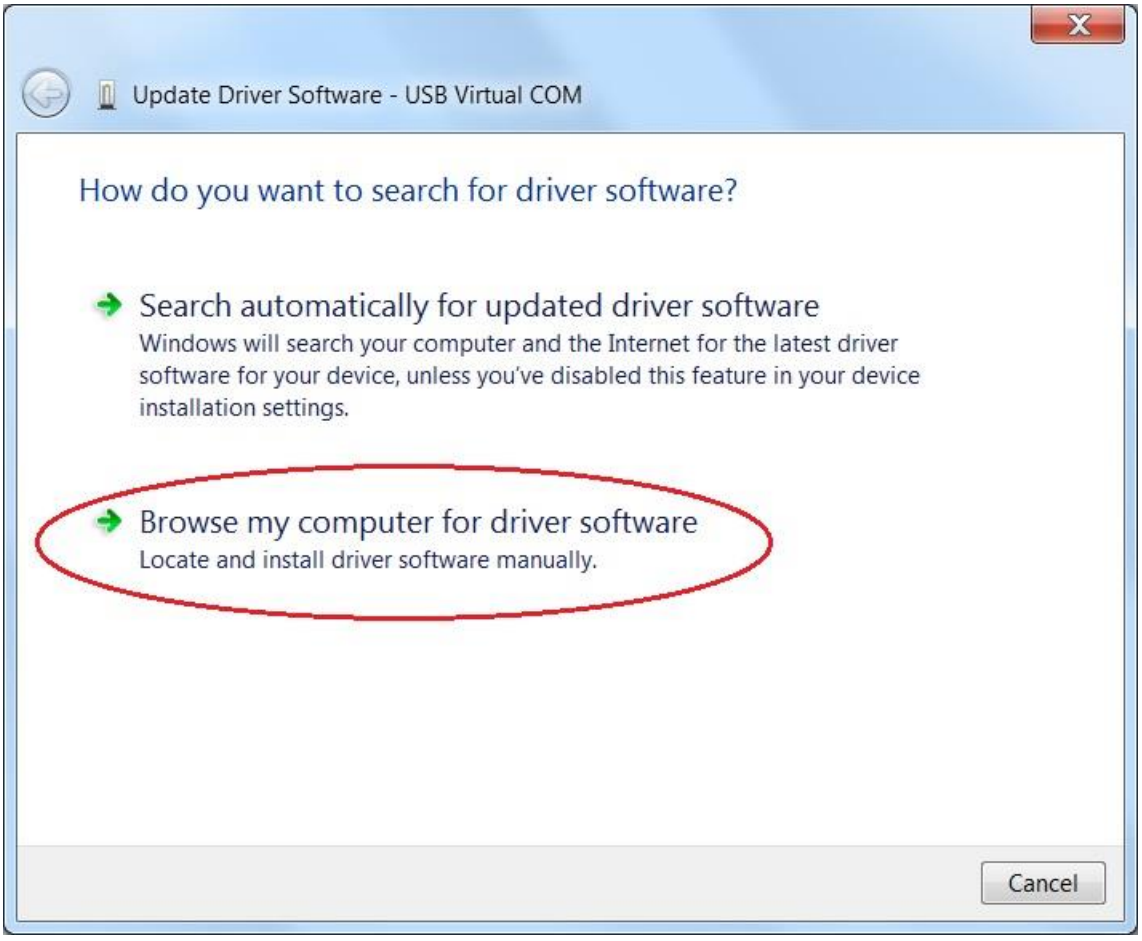


圖 19 未安裝驅動程式(VCOM 裝置)


Update Driver Software - USB Virtual COM
How do you want to search for driver software?
Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.
Browse my computer for driver software
Locate and install driver software manually.
Cancel

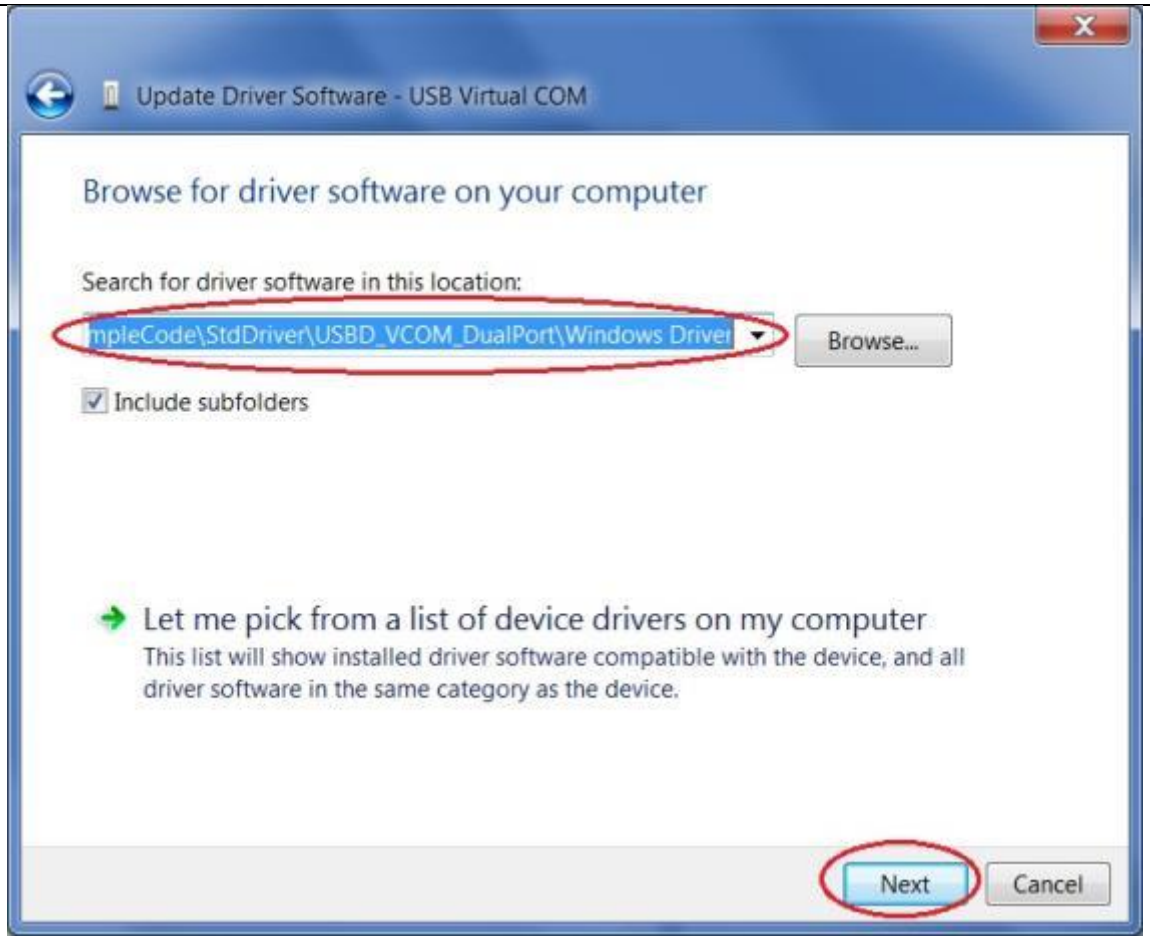

Update Driver Software - USB Virtual COM
Browse for driver software on your computer
Search for driver software in this location:
mpleCode\StdDriver\USBD_VCOM_DualPort\Windows Driver
Browse...
Include subfolders
Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.
Next
Cancel

圖 20 安裝 INF

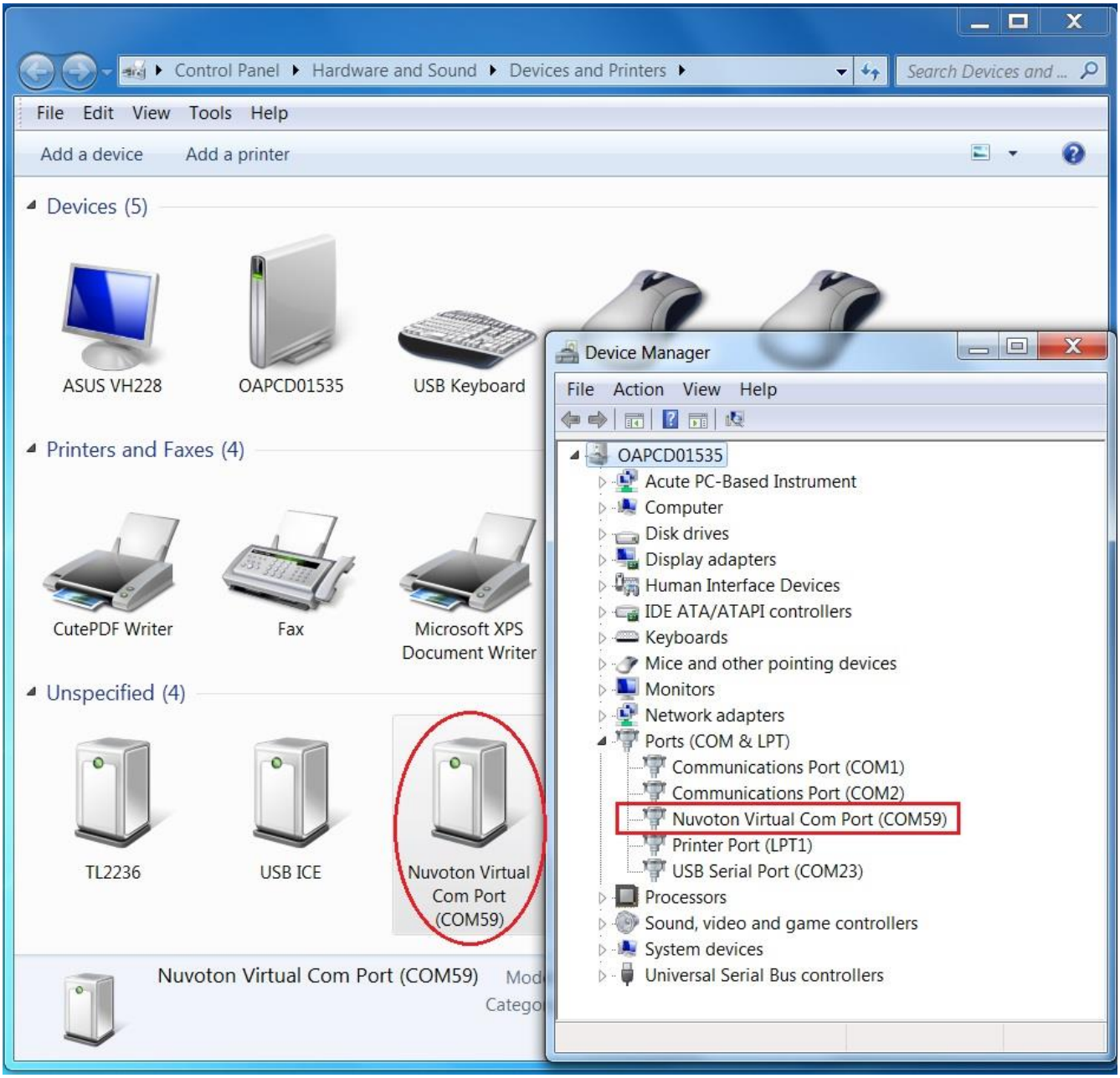

Control Panel ▸ Hardware and Sound ▸ Devices and Printers ▸
Search Devices and ...
File Edit View Tools Help
Add a device
Add a printer
Devices (5)
ASUS VH228
OAPCD01535
USB Keyboard
Printers and Faxes (4)
CutePDF Writer
Fax
Microsoft XPS Document Writer
Unspecified (4)
TL2236
USB ICE
Nuvoton Virtual Com Port (COM59)
Nuvoton Virtual Com Port (COM59)
Device Manager
File Action View Help
OAPCD01535
Acute PC-Based Instrument
Computer
Disk drives
Display adapters
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Nuvoton Virtual Com Port (COM59)
Printer Port (LPT1)
USB Serial Port (COM23)
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers

圖 21 VCOM 裝置

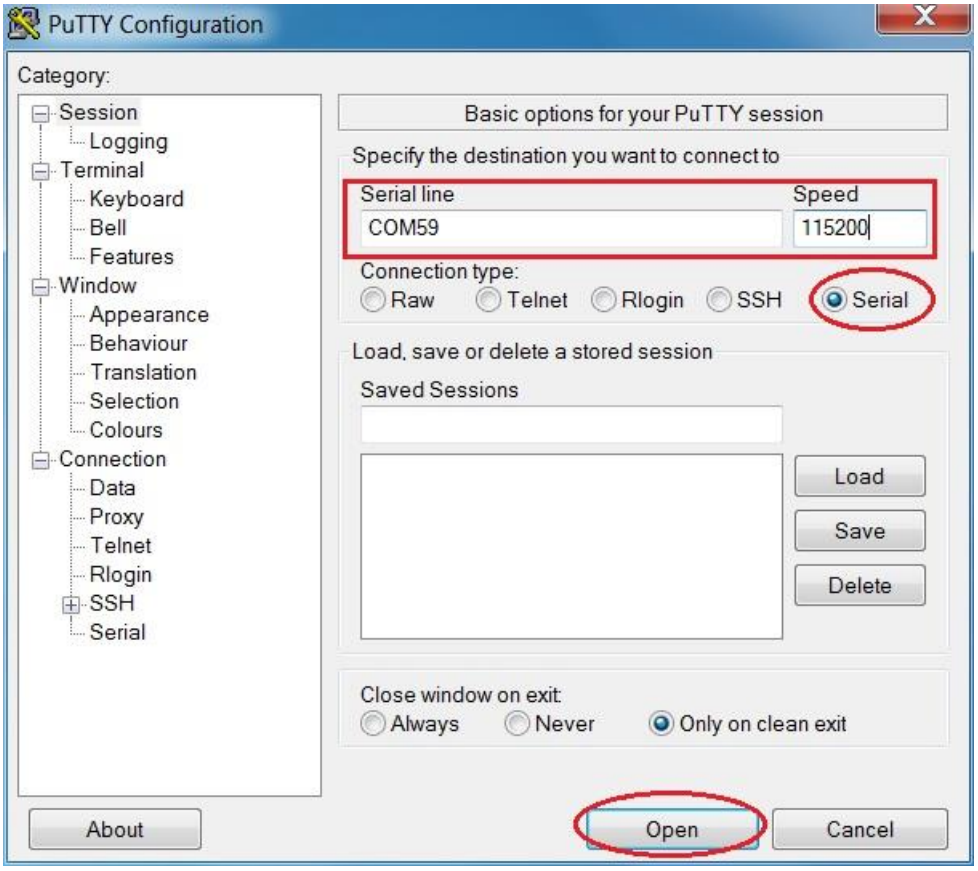

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM59
115200
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel

圖 22 VCOM 設定

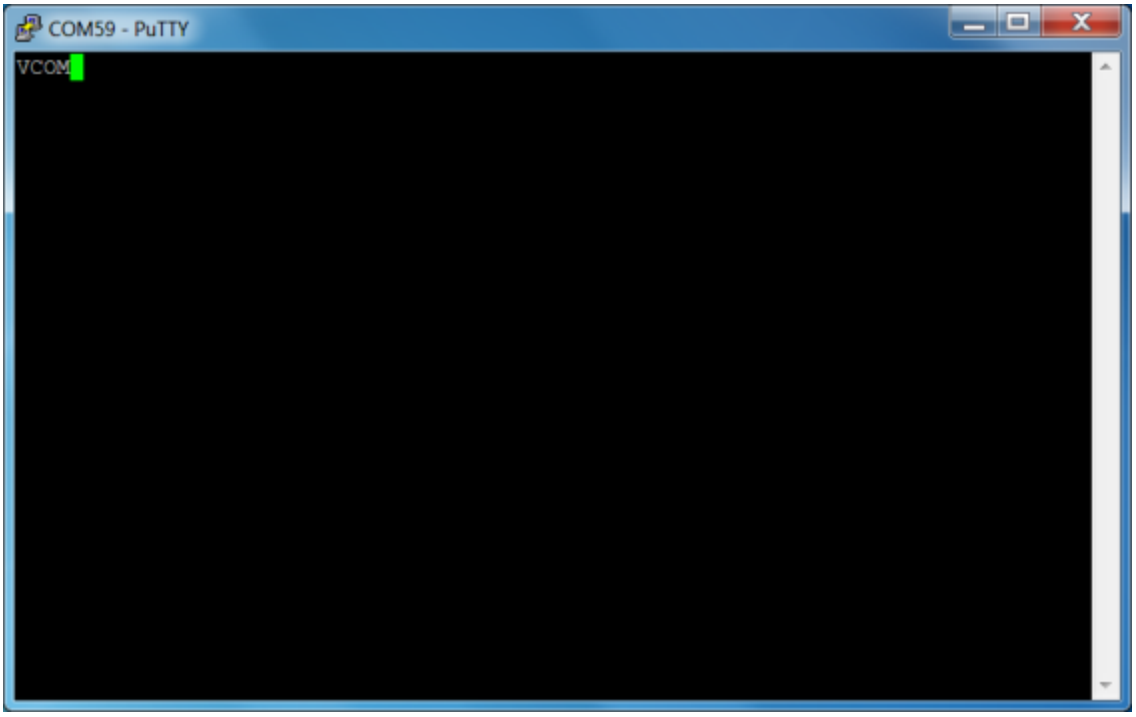


圖 23 VCOM 範例

5.3.1 CDC 類別的特定請求

在USB CDC類別中定義虛擬COM Port裝置必要的三個類別請求，這些請求在"vcom_serial.c"中執行。VCOM_ClassRequest用來回應CDC的三個類別請求：SET_LINE_CODING、GET_LINE_CODING及SET_CONTROL_LINE_STATE。SET_LINE_CODING、GET_LINE_CODING分別用來設定或取得COM Port的Baudrate、停止位元的數目、奇偶校驗和資料位元數；SET_CONTROL_LINE_STATE是設定RS-232 RTS和DTR控制信號。

```
void VCOM_ClassRequest(void)
{
  uint8_t buf[8];

  USBD_GetSetupPacket(buf);

  if (buf[0] & 0x80) { /* request data transfer direction */
    // Device to host
    switch (buf[1]) {
    case GET_LINE_CODE: {
      if (buf[4] == 0) { /* VCOM-1 */
        USBD_MemCopy((uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)),
                     (uint8_t *)&gLineCoding, 7);
      }
      /* Data stage */
```

```
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 7);
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
        }
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    } else {
        // Host to device
        switch (buf[1]) {
        case SET_CONTROL_LINE_STATE: {
            if (buf[4] == 0) { /* VCOM-1 */
                gCtrlSignal = buf[3];
                gCtrlSignal = (gCtrlSignal << 8) | buf[2];
            }

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_LINE_CODE: {
            if (buf[4] == 0) /* VCOM-1 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding, 7);

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);

            break;
        }
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
```

```
      }
     }
   }
}
```

5.3.2 Endpoint 的配置

配置VCOM所要使用的端點，此範例共使用五個端點。端點0和端點1設定為控制型IN/OUT傳輸，用於USB標準及類別請求；端點2及端點3分別設定為巨量型IN傳輸和巨量型OUT傳輸，用來傳輸主機與COM Port之間的資料。端點4設定為中斷型IN傳輸，在此範例中並無傳輸任何資料。

```
/* 端點的最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE     0
#define SETUP_BUF_LEN       8
#define EP0_BUF_BASE        (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN          EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE        (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN          EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE        (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN          EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE        (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN          EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE        (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN          EP4_MAX_PKT_SIZE

/* 端點地址 */
#define BULK_IN_EP_NUM       0x01
#define BULK_OUT_EP_NUM     0x02
#define INT_IN_EP_NUM         0x03

void VCOM_Init(void)
{
   /* Init setup packet buffer */
   /* Buffer for setup packet -> [0 ~ 0x7] */
   USBD->BUFSEG = SETUP_BUF_BASE;
```

```
    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);
}
```

VCOM實作的主要部分為USB與UART之間的資料傳輸處理。當資料待處理時，UART會產生中斷，UART0_IRQHandler會把資料放到UART的緩衝區中。VCOM_TransferData使用USB傳送UART緩衝區內的資料。

```
void UART0_IRQHandler(void)
{
    uint8_t bInChar;
    int32_t size;
    uint32_t u32IntStatus;

    u32IntStatus = UART0->ISR;
```

```
if((u32IntStatus & UART_ISR_RDA_IS_Msk) || (u32IntStatus & UART_ISR_RTO_IS_Msk)) {
    /* Receiver FIFO threshold level is reached or Rx time out */

    /* Get all the input characters */
    while (!(UART0->FSR & UART_FSR_RX_EMPTY_F_Msk)) {
        /* Get the character from UART Buffer */
        bInChar = UART0->RBR;

        /* Check if buffer full */
        if(comRbytes < RXBUFSIZE) {
            /* Enqueue the character */
            comRbuf[comRtail++] = bInChar;
            if(comRtail >= RXBUFSIZE)
                comRtail = 0;
            comRbytes++;
        } else {
            /* FIFO over run */
        }
    }
}

if(u32IntStatus & UART_ISR_THRE_IS_Msk) {

    if(comTbytes) {
        /* Fill the Tx FIFO */
        size = comTbytes;
        if(size >= TX_FIFO_SIZE) {
            size = TX_FIFO_SIZE;
        }

        while(size) {
            bInChar = comTbuf[comThead++];
            UART0->THR = bInChar;
            if(comThead >= TXBUFSIZE)
                comThead = 0;
            comTbytes--;
            size--;
        }
    } else {
        /* No more data, just stop Tx (Stop work) */
```

```
            UART0->IER &= ~UART_IER_THRE_IE_Msk;
        }
    }
}

void VCOM_TransferData(void)
{
    int32_t i, i32Len;

    /* Check whether USB is ready for next packet or not*/
    if(gu32TxSize == 0)
    {
        /* Check whether we have new COM Rx data to send to USB or not */
        if(comRbytes)
        {
            i32Len = comRbytes;
            if(i32Len > EP2_MAX_PKT_SIZE)
                i32Len = EP2_MAX_PKT_SIZE;

            for(i = 0; i < i32Len; i++)
            {
                gRxBuf[i] = comRbuf[comRhead++];
                if(comRhead >= RXBUFSIZE)
                    comRhead = 0;
            }

            __set_PRIMASK(1);
            comRbytes -= i32Len;
            __set_PRIMASK(0);

            gu32TxSize = i32Len;
            USBD_MemCopy((uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2)),
                                (uint8_t *)gRxBuf, i32Len);
            USBD_SET_PAYLOAD_LEN(EP2, i32Len);
        }
        else
        {
          /* Prepare a zero packet if previous packet size is EP2_MAX_PKT_SIZE and
            no more data to send at this moment to note Host the transfer has been done */
            i32Len = USBD_GET_PAYLOAD_LEN(EP2);
            if(i32Len == EP2_MAX_PKT_SIZE)
```

```
            USBD_SET_PAYLOAD_LEN(EP2, 0);
        }
    }

    /* Process the Bulk out data when bulk out data is ready. */
    if(gi8BulkOutReady && (gu32RxSize <= TXBUFSIZE - comTbytes))
    {
        for(i = 0; i < gu32RxSize; i++)
        {
            comTbuf[comTtail++] = gpu8RxBuf[i];
            if(comTtail >= TXBUFSIZE)
                comTtail = 0;
        }

        __set_PRIMASK(1);
        comTbytes += gu32RxSize;
        __set_PRIMASK(0);

        gu32RxSize = 0;
        gi8BulkOutReady = 0; /* Clear bulk out ready flag */

        /* Ready to get next BULK out */
        USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
    }

    /* Process the software Tx FIFO */
    if(comTbytes)
    {
        /* Check if Tx is working */
        if((UART0->IER & UART_IER_THRE_IE_Msk) == 0)
        {
            /* Send one bytes out */
            UART0->THR = comTbuf[comThead++];
            if(comThead >= TXBUFSIZE)
                comThead = 0;

            __set_PRIMASK(1);
            comTbytes--;
            __set_PRIMASK(0);

            /* Enable Tx Empty Interrupt. (Trigger first one) */
```

```
            UART0->IER |= UART_IER_THRE_IE_Msk;
        }
    }
}
```

5.3.3 定義各個描述元

裝置、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入CDC類別的各種描述元。

```
/* USB 組態描述元，它包含介面及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    LEN_CONFIG,             /* bLength */
    DESC_CONFIG,            /* bDescriptorType */
    0x43, 0x00,             /* wTotalLength */
    0x02,                   /* bNumInterfaces */
    0x01,                   /* bConfigurationValue */
    0x00,                   /* iConfiguration */
    0xC0,                   /* bmAttributes */
    0x32,                   /* MaxPower */

    /* 介面0 : VCOM Control介面描述元 */
    LEN_INTERFACE,          /* bLength */
    DESC_INTERFACE,         /* bDescriptorType */
    0x00,                   /* bInterfaceNumber     */
    0x00,                   /* bAlternateSetting    */
    0x01,                   /* bNumEndpoints        */
    0x02,                   /* bInterfaceClass      */
    0x02,                   /* bInterfaceSubClass   */
    0x01,                   /* bInterfaceProtocol   */
    0x00,                   /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,                   /* Size of the descriptor, in bytes */
    0x24,                   /* CS_INTERFACE descriptor type */
    0x00,                   /* Header functional descriptor subtype */
    0x10, 0x01,             /* Communication device compliant to the communication spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,                   /* Size of the descriptor, in bytes */
    0x24,                   /* CS_INTERFACE descriptor type */
    0x01,                   /* Call management functional descriptor */
    0x00,                   /* BIT0: Whether device handle call management itself. */
```

```
/* BIT1: Whether device can send/receive call management information over a Data Class Interface 0 */
0x01,                   /* Interface number of data class interface optionally used for call management */

/* Communication Class Specified INTERFACE descriptor */
0x04,                   /* Size of the descriptor, in bytes */
0x24,                   /* CS_INTERFACE descriptor type */
0x02,                   /* Abstract control management funcational descriptor subtype */
0x00,                   /* bmCapabilities       */

/* Communication Class Specified INTERFACE descriptor */
0x05,                   /* bLength */
0x24,                   /* bDescriptorType: CS_INTERFACE descriptor type */
0x06,                   /* bDescriptorSubType */
0x00,                   /* bMasterInterface */
0x01,                   /* bSlaveInterface0 */

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
 LEN_ENDPOINT,       /* bLength */
DESC_ENDPOINT,      /* bDescriptorType */
(EP_INPUT | INT_IN_EP_NUM),    /* bEndpointAddress */
EP_INT,                 /* bmAttributes     */
EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
0x01,                   /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
/* INTERFACE descriptor */
LEN_INTERFACE,       /* bLength */
DESC_INTERFACE,     /* bDescriptorType */
0x01,                   /* bInterfaceNumber */
0x00,                   /* bAlternateSetting */
0x02,                   /* bNumEndpoints */
0x0A,                   /* bInterfaceClass */
0x00,                   /* bInterfaceSubClass */
0x00,                   /* bInterfaceProtocol */
0x00,                   /* iInterface */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
LEN_ENDPOINT,         /* bLength */
DESC_ENDPOINT,        /* bDescriptorType */
(EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
EP_BULK,                /* bmAttributes */
```

```
    EP2_MAX_PKT_SIZE, 0x00,          /* wMaxPacketSize */
    0x00,                            /* bInterval */

    /* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                    /* bLength */
    DESC_ENDPOINT,                   /* bDescriptorType */
    (EP_OUTPUT | BULK_OUT_EP_NUM) ,  /* bEndpointAddress */
    EP_BULK,                         /* bmAttributes */
    EP3_MAX_PKT_SIZE, 0x00,          /* wMaxPacketSize */
    0x00,                            /* bInterval */
};
```

5.4 UAC

USB類別中定義音訊裝置，該規範指定每個USB音訊功能中的標準描述元和類別特定描述元，詳細的USB UAC規格可以參考USB開發者論壇(www.usb.org)。

本章節將介紹如何設計USB Speaker與Recorder裝置，NuMicro® Nano130學習板上提供耳機孔、Microphone與音源線輸入孔，本範例將示範從音源線輸入錄製語音資料再透過耳機來播放。USB用來傳送或接收語音資料，所有的語音資料都是經由音頻編碼器(NAU8822)來進行編解碼，而NAU8822是通過I^2S方式來接收語音資料，使用I^2C介面進行設置。這個音訊裝置的範例特性如下：

- 脈衝編碼調製(PCM)格式
- 採樣率：16KHz。
- 位分辨率：16
- 通道數量：2
- 靜音/取消靜音功能

使用者可以利用Windows作業系統內建的播放軟體(Windows Media Player)或錄音軟體(圖 24)來執行範例中的Speaker與Recorder功能。成功執行後，在電腦的硬體裝置管理員中會看到USB Audio裝置，並且會把它設為電腦端預設的音訊裝置，如圖 25所示。靜音與取消靜音功能可以從電腦Windows作業系統的音量控制視窗來設定(圖 26)，一開始會把它們都設為靜音。使用者可以先使用Windows作業系統內建的錄音軟體來取得語音資料，接著再利用Windows Media Player來播放語音資料，執行成功時可以從接在板子上的耳機中聽到錄製的聲音。

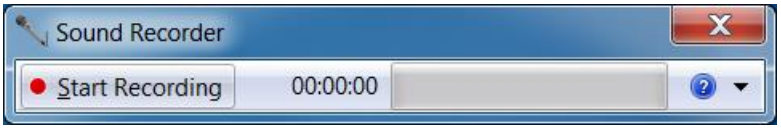


圖 24 Windows 內建錄音軟體

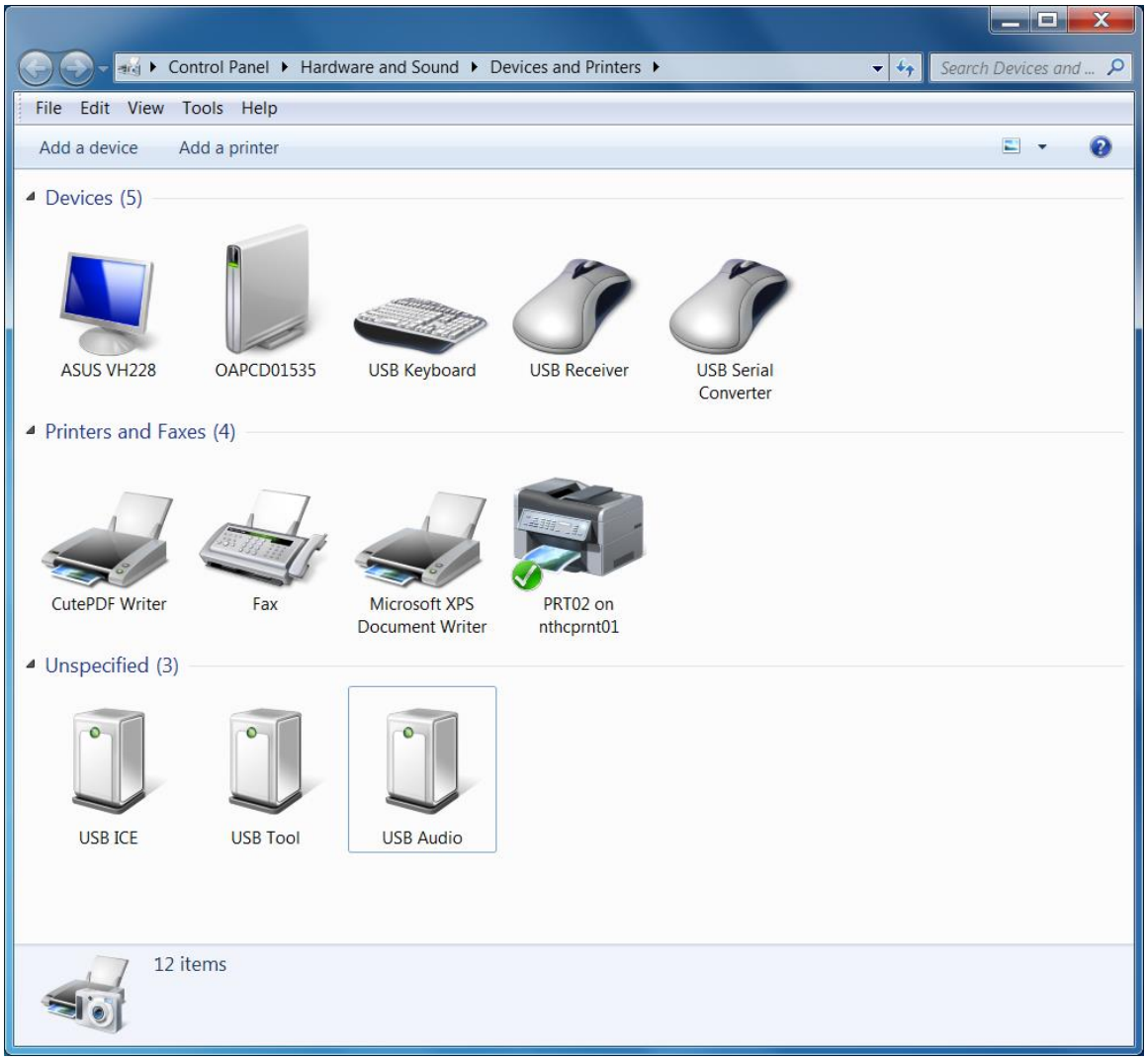


圖 25 Audio 裝置

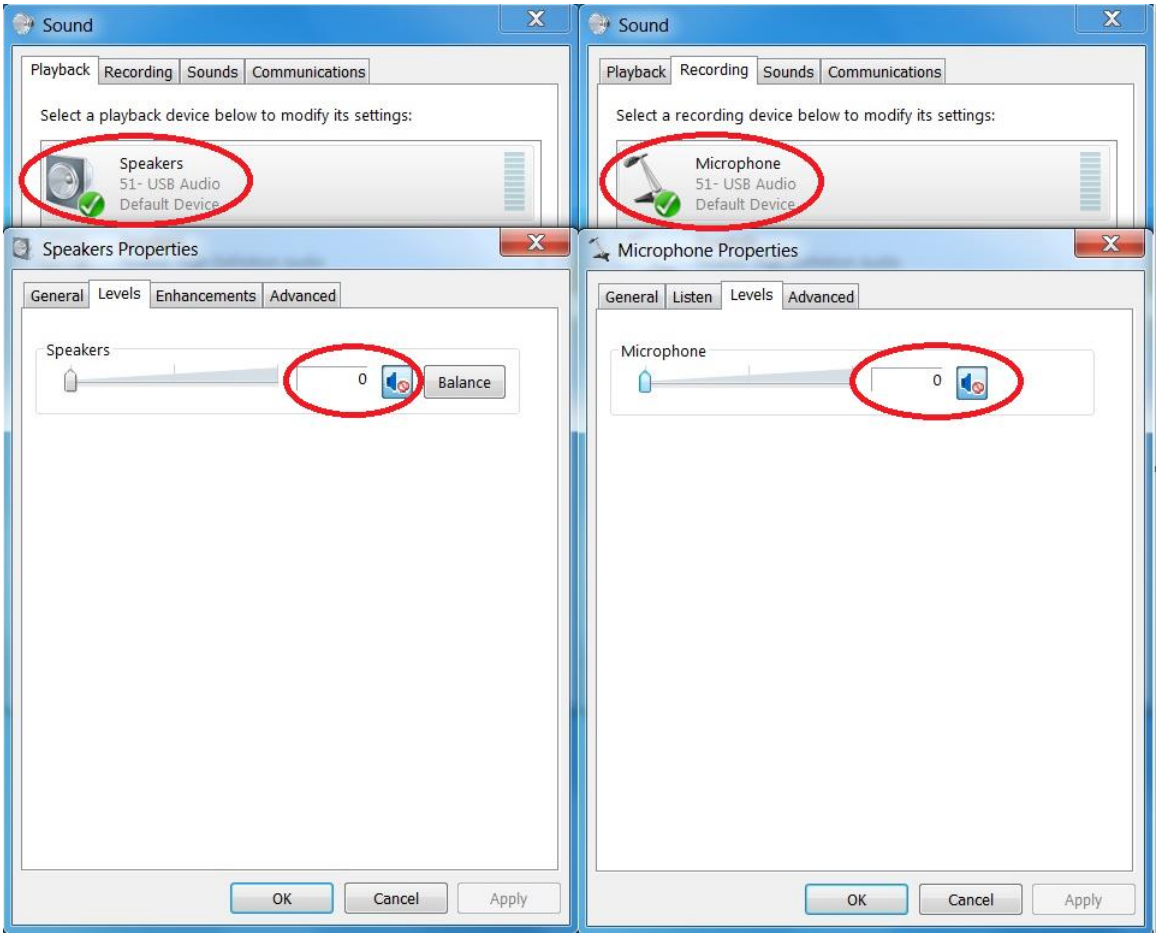


圖 26 靜音控制介面

5.4.1 類別的特定請求

UAC_ClassRequest用來回應USB Audio類別規範定義三個類別請求，分別為Class Specific Request、Audio Control Request及Audio Stream Request。

```
void UAC_ClassRequest(void)
{
   uint8_t buf[8];

   USBD_GetSetupPacket(buf);

   if (buf[0] & 0x80) { /* request data transfer direction */
      // Device to host
      switch (buf[1]) {
      case UAC_GET_CUR: {
         switch (buf[3]) {
         case MUTE_CONTROL: {
```

```
    if (REC_FEATURE_UNITID == buf[5])
        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_RecMute;
    else if (PLAY_FEATURE_UNITID == buf[5])
        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_PlayMute;

    /* Data stage */
    USBD_SET_DATA1(EP0);
    USBD_SET_PAYLOAD_LEN(EP0, 1);
    break;
}
case VOLUME_CONTROL: {
    if (REC_FEATURE_UNITID == buf[5]) {
        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_RecVolume;
        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
        g_usbd_RecVolume >> 8;
    } else if (PLAY_FEATURE_UNITID == buf[5]) {
        if(buf[2] == 1) {
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
            g_usbd_PlayVolumeL;
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
            g_usbd_PlayVolumeL >> 8;
        } else {
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
            g_usbd_PlayVolumeR;
            M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
            g_usbd_PlayVolumeR >> 8;
        }
    }
    /* Data stage */
    USBD_SET_DATA1(EP0);
    USBD_SET_PAYLOAD_LEN(EP0, 2);
    break;
}
default: {
    /* Setup error, stall the device */
    USBD_SetStall(0);
}
}
// Trigger next Control Out DATA1 Transaction.
/* Status stage */
USBD_PrepareCtrlOut(0,0);
```

```
        break;
    }

    case UAC_GET_MIN: {
        switch (buf[3]) {
        case VOLUME_CONTROL: {
            if (REC_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_RecMinVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_RecMinVolume >> 8;
            } else if (PLAY_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_PlayMinVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_PlayMinVolume >> 8;
            }
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 2);
            break;
        }
        default:
            /* STALL control pipe */
            USBD_SetStall(0);
        }
        // Trigger next Control Out DATA1 Transaction.
        /* Status stage */
        USBD_PrepareCtrlOut(0,0);
        break;
    }

    case UAC_GET_MAX: {
        switch (buf[3]) {
        case VOLUME_CONTROL: {
            if (REC_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_RecMaxVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_RecMaxVolume >> 8;
            } else if (PLAY_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_PlayMaxVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_PlayMaxVolume >> 8;
```

```
            }
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 2);
            break;
        }
        default:
            /* STALL control pipe */
            USBD_SetStall(0);
        }
        // Trigger next Control Out DATA1 Transaction.
        /* Status stage */
        USBD_PrepareCtrlOut(0,0);
        break;
    }

    case UAC_GET_RES: {
        switch (buf[3]) {
        case VOLUME_CONTROL: {
            if (REC_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_RecResVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_RecResVolume >> 8;
            } else if (PLAY_FEATURE_UNITID == buf[5]) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = g_usbd_PlayResVolume;
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
                g_usbd_PlayResVolume >> 8;
            }
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 2);
            break;
        }
        default:
            /* STALL control pipe */
            USBD_SetStall(0);
        }
        // Trigger next Control Out DATA1 Transaction.
        /* Status stage */
        USBD_PrepareCtrlOut(0,0);
        break;
```

```
    }

    default: {
        /* Setup error, stall the device */
        USBD_SetStall(0);
    }
    }
} else {
    // Host to device
    switch (buf[1]) {
    case UAC_SET_CUR: {
        switch (buf[3]) {
        case MUTE_CONTROL:
            if (REC_FEATURE_UNITID == buf[5])
                USBD_PrepareCtrlOut((uint8_t *)&g_usbd_RecMute, buf[6]);
            else if (PLAY_FEATURE_UNITID == buf[5]) {
                USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayMute, buf[6]);
            }
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;

        case VOLUME_CONTROL:
            if (REC_FEATURE_UNITID == buf[5])
                USBD_PrepareCtrlOut((uint8_t *)&g_usbd_RecVolume, buf[6]);
            else if (PLAY_FEATURE_UNITID == buf[5]) {
                if (buf[2] == 1) {
                    USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayVolumeL, buf[6]);
                    if(g_usbd_PlayVolumeL & 0x8000)
                        g_usbd_PlayVolumeL = (g_usbd_PlayVolumeL & 0x7FFF) >> 8;
                    else
                        g_usbd_PlayVolumeL = (g_usbd_PlayVolumeL >> 7);
                        bPlayVolumeLAdjust = TRUE;/* Set left DAC volume */
                } else {
                    USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayVolumeR, buf[6]);
                    if(g_usbd_PlayVolumeR & 0x8000)
                        g_usbd_PlayVolumeR = (g_usbd_PlayVolumeR & 0x7FFF) >> 8;
                    else
                        g_usbd_PlayVolumeR = (g_usbd_PlayVolumeR >> 7);
                        bPlayVolumeRAdjust = TRUE; /* Set right DAC volume */
```

```
            }
        }
        /* Status stage */
        USBD_SET_DATA1(EP0);
        USBD_SET_PAYLOAD_LEN(EP0, 0);
        break;

      default:
        /* STALL control pipe */
        USBD_SetStall(0);
        break;
      }
      break;
    }

    default: {
      /* Setup error, stall the device */
      USBD_SetStall(0);
      break;
    }
    }
  }
}
```

5.4.2 Endpoint 的配置

配置音訊裝置所要使用的端點，此範例共使用四個端點。端點0和端點1設定為控制型IN/OUT傳輸，用於USB標準及類別請求，或是使用此端點進行音量調整；端點2設定為等時型IN傳輸，由裝置傳送語音資料給主機。端點3設定為等時型Out是用來將語音資料經由USB Port傳給裝置。

```
/* 端點的最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    REC_RATE*REC_CHANNELS*2/1000
#define EP3_MAX_PKT_SIZE    PLAY_RATE*PLAY_CHANNELS*2/1000

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE      0
#define SETUP_BUF_LEN       8
#define EP0_BUF_BASE        (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN         EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE        (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN         EP1_MAX_PKT_SIZE
```

```
#define EP2_BUF_BASE        (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN         EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE        (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN         EP3_MAX_PKT_SIZE

/* 端點地址 */
#define ISO_IN_EP_NUM       0x01
#define ISO_OUT_EP_NUM      0x02

void UAC_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Isochronous IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | ISO_IN_EP_NUM | USBD_CFG_TYPE_ISO);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /*****************************************************/
    /* EP3 ==> Isochronous OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | ISO_OUT_EP_NUM | USBD_CFG_TYPE_ISO);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}
```

5.4.3 定義各個描述元

每個音訊裝置至少要有一個Audio Control介面加上零到多個Audio Streaming介面組成。Audio Control介面用於音量控制；而Audio Streaming介面用來接收或傳送語音資料(Audio stream)。

為了對聲音的各種物理特性進行控制，需要將聲音裝置分割成不同可獨立控制的實體(Entity)，在USB音頻類中分別稱為Units和Terminals。一個Unit包括一個或多個擁有各自編號的輸入端及唯一輸出端，每個裝置的Unit都有它的Unit描述元，Unit描述元包括所有Unit需要的資訊。Terminal有輸入端與輸出端兩種。輸入端代表音頻裝置內一個聲道的開始；輸出端表示音頻裝置內一個聲道的結束，Terminal也有自己的Terminal描述元。

配置描述元包含3組介面，Interface 0作為音訊控制介面，Interface 1作為Microphone介面，Interface 2作為Speaker介面。在Interface 0的定義中沒有包含任何端點，所以與音訊相關的控制將通過預設的控制端點0來傳遞資訊。Interface 0中的Feature Unit分別定義音量調整方式或靜音控制，其中ID5用來調整Microphone的音量，ID6則用來控制Speaker的音量。每一個Audio Streaming介面都定義了2個Setting，Setting0是預設時作為USB裝置，不做任何資料傳送；若是利用該USB裝置來進行播放和錄製語音，則使用的設定為Setting1。本裝置的語音資料使用PCM格式。

```
uint8_t gu8ConfigDescriptor[] = {
    LEN_CONFIG,             /* bLength */
    DESC_CONFIG,            /* bDescriptorType */
    0xC0,0x00,              /* wTotalLength */
    0x03,                   /* bNumInterfaces */
    0x01,                   /* bConfigurationValue */
    0x00,                   /* iConfiguration */
    0x80,                   /* bmAttributes */
    0x20,                   /* Max power */

    /* Standard AC inteface */
    0x09,                   /* bLength */
    0x04,                   /* bDescriptorType */
    0x00,                   /* bInterfaceNumber */
    0x00,                   /* bAlternateSetting */
    0x00,                   /* bNumEndpoints */
    0x01,                   /* bInterfaceClass:AUDIO */
    0x01,                   /* bInterfaceSubClass:AUDIOCONTROL */
    0x00,                   /* bInterfaceProtocol */
    0x00,                   /* iInterface */

    /* Class-spec AC interface descriptor */
    0x0A,                   /* bLength */
    0x24,                   /* bDescriptorType:CS_INTERFACE */
```

```
0x01,                   /* bDescriptorSubType:HEADER */
0x00,0x01,              /* bcdADC:1.0 */
0x46,0x00,              /* wTotalLength */
0x02,                   /* bInCollection */
0x01,                   /* baInterfaceNr(1) */
0x02,                   /* baInterfaceNr(n) */

/* TID 1: Input for usb streaming */
0x0C,                    /* bLength */
0x24,                   /* bDescriptorType:CS_INTERFACE */
0x02,                   /* bDescriptorSubType:INPUT_TERMINAL */
0x01,                   /* bTerminalID */
0x01,0x01,              /* wTerminalType: 0x0101 usb streaming */
0x00,                   /* bAssocTerminal */
PLAY_CHANNELS,      /* bNrChannels */
PLAY_CH_CFG,0x00,   /* wChannelConfig */
0x00,                   /* iChannelNames */
0x00,                   /* iTerminal */

/* UNIT ID 5: Feature Unit */
0x08,                   /* bLength */
0x24,                   /* bDescriptorType */
0x06,                   /* bDescriptorSubType */
REC_FEATURE_UNITID,    /* bUnitID */
0x04,                   /* bSourceID */
0x01,                   /* bControlSize */
0x03,                   /* bmaControls(0) */
0x00,                   /* iFeature */

/* TID 2: Output Terminal for usb streaming */
0x09,                   /* bLength */
0x24,                   /* bDescriptorType:CS_INTERFACE */
0x03,                   /* bDescriptorSubType:OUTPUT_TERMINAL */
0x02,                   /* bTerminalID */
0x01,0x01,              /* wTerminalType: 0x0101 usb streaming */
0x00,                   /* bAssocTerminal */
REC_FEATURE_UNITID,    /* bSourceID */
0x00,                   /* iTerminal */

/* UNIT ID 6: Feature Unit */
0x0A,                    /* bLength */
```

```
0x24,                   /* bDescriptorType */
0x06,                   /* bDescriptorSubType */
PLAY_FEATURE_UNITID,    /* bUnitID */
0x01,                   /* bSourceID */
0x01,                   /* bControlSize */
0x01,                   /* bmaControls(0) */
0x02,                   /* bmaControls(0) */
0x02,                   /* bmaControls(0) */
0x00,                   /* iFeature */

/* TID 3: Output for speaker */
0x09,                   /* bLength*/
0x24,                   /* bDescriptorType:CS_INTERFACE*/
0x03,                   /* bDescriptorSubType:OUTPUT_TERMINAL*/
0x03,                   /* bTerminalID*/
0x01,0x03,              /* wTerminalType: 0x0301 speaker*/
0x00,                   /* bAssocTerminal*/
0x06,                   /* bSourceID*/
0x00,                   /* iTerminal*/

/* TID 4: Input Terminal for microphone */
0x0C,                    /* bLength */
0x24,                   /* bDescriptorType:CS_INTERFACE */
0x02,                   /* bDescriptorSubType:INPUT_TERMINAL*/
0x04,                   /* bTerminalID*/
0x01,0x02,              /* wTerminalType: 0x0201 microphone*/
0x00,                   /* bAssocTerminal*/
REC_CHANNELS,       /* bNrChannels*/
REC_CH_CFG, 0x00,   /* wChannelConfig*/
0x00,                   /* iChannelNames*/
0x00,                   /* iTerminal*/

/* Standard AS interface 1, alternate 0 */
0x09,                   /* bLength */
0x04,                   /* bDescriptorType */
0x01,                   /* bInterfaceNumber */
0x00,                   /* bAlternateSetting */
0x00,                   /* bNumEndpoints */
0x01,                   /* bInterfaceClass:AUDIO */
0x02,                   /* bInterfaceSubClass:AUDIOSTREAMING */
0x00,                   /* bInterfaceProtocol */
```

```
0x00,                   /* iInterface */

/* Standard AS interface 1, alternate 1 */
0x09,                   /* bLength */
0x04,                   /* bDescriptorType */
0x01,                   /* bInterfaceNumber */
0x01,                   /* bAlternateSetting */
0x01,                   /* bNumEndpoints */
0x01,                   /* bInterfaceClass:AUDIO */
0x02,                   /* bInterfaceSubClass:AUDIOSTREAMING */
0x00,                   /* bInterfaceProtocol */
0x00,                   /* iInterface */

/* Class-spec AS interface, this interface's endpoint connect to TID 0x02 */
0x07,                   /* bLength */
0x24,                   /* bDescriptorType:CS_INTERFACE */
0x01,                   /* bDescriptorSubType:AS_GENERAL */
0x02,                   /* bTernimalLink */
0x01,                   /* bDelay */
0x01,0x00,              /* wFormatTag:0x0001 PCM */

/* Type I format type Descriptor */
0x0B,                    /* bLength */
0x24,                   /* bDescriptorType:CS_INTERFACE */
0x02,                   /* bDescriptorSubType:FORMAT_TYPE */
0x01,                   /* bFormatType:FORMAT_TYPE_I */
REC_CHANNELS,  /* bNrChannels: NumberOfChanne = 2 */
0x02,                   /* bSubFrameSize: 2byte */
0x10,                   /* bBitResolution: 16bit */
0x01,                   /* bSamFreqType : 0 continuous; 1 discrete: One sampling frequency */
REC_RATE_LO,
REC_RATE_MD,
REC_RATE_HI,     /* Sample Frequency 16k*/

/* Standard AS ISO Audio Data Endpoint */
0x09,                   /* bLength */
0x05,                   /* bDescriptorType */
ISO_IN_EP_NUM | EP_INPUT,    /* bEndpointAddress */
0x0d,                   /* bmAttributes */
EP2_MAX_PKT_SIZE,0x00,        /* wMaxPacketSize*/
0x01,                   /* bInterval*/
```

```
0x00,               /* bRefresh*/
0x00,               /* bSynchAddress*/

/* Class-spec AS ISO Audio Data endpoint Descriptor */
0x07,               /* bLength */
0x25,               /* bDescriptorType:CS_ENDPOINT */
0x01,               /* bDescriptorSubType:EP_GENERAL */
0x00,               /* bmAttributes */
0x00,               /* bLockDelayUnits */
0x00,               0x00,    /* wLockDelay */

/* Standard AS interface 2, alternate 0 */
0x09,               /* bLength */
0x04,               /* bDescriptorType */
0x02,               /* bInterfaceNumber */
0x00,               /* bAlternateSetting */
0x00,               /* bNumEndpoints */
0x01,               /* bInterfaceClass:AUDIO */
0x02,               /* bInterfaceSubClass:AUDIOSTREAMING */
0x00,               /* bInterfaceProtocol */
0x00,               /* iInterface */

/* Standard AS interface 2, alternate 1 */
0x09,               /* bLength */
0x04,               /* bDescriptorType */
0x02,               /* bInterfaceNumber */
0x01,               /* bAlternateSetting */
0x01,               /* bNumEndpoints */
0x01,               /* bInterfaceClass:AUDIO */
0x02,               /* bInterfaceSubClass:AUDIOSTREAMING */
0x00,               /* bInterfaceProtocol */
0x00,               /* iInterface */

/* Class-spec AS inf this interface's endpoint connect to TID 0x01 */
0x07,               /* bLength */
0x24,               /* bDescriptorType:CS_INTERFACE */
0x01,               /* bDescriptorSubType:AS_GENERAL */
0x01,               /* bTernimalLink */
0x01,               /* bDelay */
0x01,0x00,          /* wFormatTag:0x0001 PCM */
```

```
    /* Type I format type Descriptor */
    0x0B,                   /* bLength */
    0x24,                   /* bDescriptorType:CS_INTERFACE */
    0x02,                   /* bDescriptorSubType:FORMAT_TYPE */
    0x01,                   /* bFormatType:FORMAT_TYPE_I */
    PLAY_CHANNELS,    /* bNrChannels */
    0x02,                   /* bSubFrameSize */
    0x10,                   /* bBitResolution */
    0x01,                   /* bSamFreqType : 0 continuous; 1 discrete */
    PLAY_RATE_LO,
    PLAY_RATE_MD,
    PLAY_RATE_HI,   /* Sample Frequency */

    /* Standard AS ISO Audio Data Endpoint, output, addtess 2, Max 0x40 */
    0x09,                   /* bLength */
    0x05,                   /* bDescriptorType */
    ISO_OUT_EP_NUM | EP_OUTPUT,    /* bEndpointAddress */
    0x0d,                   /* bmAttributes */
    EP3_MAX_PKT_SIZE, 0x00,             /* wMaxPacketSize */
    0x01,                   /* bInterval */
    0x00,                   /* bRefresh */
    0x00,                   /* bSynchAddress */

    /* Class-spec AS ISO Audio Data endpoint Descriptor */
    0x07,                   /* bLength */
    0x25,                   /* bDescriptorType:CS_ENDPOINT */
    0x01,                   /* bDescriptorSubType:EP_GENERAL */
    0x80,                   /* bmAttributes */
    0x00,                   /* bLockDelayUnits */
    0x00, 0x00,             /* wLockDelay */
};
```

使用 USB 傳輸音訊資料會面臨 USB 資料傳輸速率與音頻編碼器(NAU8822)的速度不同步問題。因此，本範例每 1ms 調整 NAU8822 的取樣頻率，來克服 USB 與 NAU8822 頻率不匹配問題。

```
void TMR0_IRQHandler(void)
{
    TIMER_ClearIntFlag(TIMER0);

    if(u8PlayEn) {
        if(u32PlayPos_In >= u32PlayPos_Out) {
            if((u32PlayPos_In-u32PlayPos_Out) > (EP2_MAX_PKT_SIZE+8)) {
```

```
            AdjustCodecPll(E_RS_UP);
        } else if((u32PlayPos_In-u32PlayPos_Out) < (EP2_MAX_PKT_SIZE-8)) {
            AdjustCodecPll(E_RS_DOWN);
        } else {
            AdjustCodecPll(E_RS_NONE);
        }
    } else {
        if((u32PlayPos_In+BUFF_LEN-u32PlayPos_Out) > (EP2_MAX_PKT_SIZE+8)) {
            AdjustCodecPll(E_RS_UP);
        } else if((u32PlayPos_In+BUFF_LEN-u32PlayPos_Out) < (EP2_MAX_PKT_SIZE-8)) {
            AdjustCodecPll(E_RS_DOWN);
        } else {
            AdjustCodecPll(E_RS_NONE);
        }
    }
  } else if(u8RecEn) {
  }
}
```

6 結論

本文以循序漸進的方式讓使用者先建立USB的基本觀念、了解新唐的NuMicro® USB2.0全速裝置控制器的功能與韌體程式的控制方式，最後，以 Nano100B 介紹如何實作四種類別的應用裝置範例。USB各種類別的應用範例實作方式大同小異，只要掌握韌體的重點實作步驟即可輕鬆完成USB應用範例。首先，初始化USB2.0全速裝置控制器來讓硬體可以正常動作；接下來為實作的重要步驟，需要根據不同的應用來配置所需使用的端點及其傳輸類型，並且準備各式的描述元讓USB裝置在列舉過程可以成功辨識其裝置。最後，再依照範例所對應的類別規格書分別處理各種事件、請求或端點的資料來達成此裝置的功能。USB複合裝置範例請參考AN_0006_USB_Composite_Device_on_Nano100B _CHT_Rev1.01。

由於USB的學問很廣泛，而且USB規格也持續更新，本文中所描述內容可以提供基本的USB觀念、USB控制及應用程式設計，搭配NuMicro® BSP中提供的各種類別裝置的原始碼程式來開發各種的USB類別裝置應用，使用者邊學邊實作可加快學習速度。若是使用者對USB產生興趣，也可以進一步對新唐其他系列的USB全速或高速裝置控制器自行了解。

Revision History

Date	Revision	Description
2017.06.16	1.00	1. 初次發布

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.