



# **PL2561**

## **HID to UART/I2C/SPI/GPIO**

### **Linux SDK 使用手冊**

Document Revision: 0.3

Document Release: Mar 12, 2024

#### **Prolific Technology Inc.**

7F, No. 48, Sec. 3, Nan Kang Rd.

Nan Kang, Taipei 115, Taiwan, R.O.C.

Telephone: +886-2-2654-6363

Fax: +886-2-2654-6161

E-mail: [sales@prolific.com.tw](mailto:sales@prolific.com.tw)

Website: <http://www.prolific.com.tw>

## 目 錄

<b>Revision History .....</b>	<b>3</b>
<b>1. 簡介 .....</b>	<b>4</b>
<b>2. SDK 檔案 .....</b>	<b>5</b>
<b>3. SDK 函數 .....</b>	<b>5</b>
3.1 列舉裝置相關函數名稱及說明 .....	5
3.1.1 GetSDKVersion .....	5
3.1.2 InitDevice .....	5
3.1.3 ExitDevice .....	5
3.1.4 GetDeviceHandleListByVid .....	5
3.1.5 GetDeviceHandleListByVidPid .....	6
3.1.6 FreeDeviceHandleList .....	6
3.1.7 OpenDeviceHandle .....	6
3.1.8 CloseDeviceHandle .....	6
3.1.9 GetDeviceInfo .....	6
3.2 UART 相關函數名稱及說明 .....	7
3.2.1 GetUartConfig .....	7
3.2.2 SetUartConfig .....	7
3.2.3 UartRead .....	7
3.2.4 UartWrite .....	8
3.2.5 UartReset .....	8
3.3 I2C 相關函數名稱及說明 .....	9
3.3.1 SetI2CFrequency .....	9
3.3.2 I2CRead .....	9
3.3.3 I2CWrite .....	9
3.3.4 I2CWriteRead .....	10
3.3.5 I2CReset .....	10
3.4 SPI 相關函數名稱及說明 .....	11
3.4.1 SetSPIFrequency .....	11
3.4.2 SetSPIMode .....	11
3.4.3 SPIRead .....	11
3.4.4 SPIWrite .....	12
3.4.5 SPIWriteRead .....	12
3.4.6 SPIFullDuplexWriteRead .....	12
3.4.7 SPIFullDuplexWriteRead2 .....	13
3.4.8 SPISetCS .....	13
3.4.9 SPIReset .....	13
3.5 GPIO 相關函數名稱及說明 .....	14
3.5.1 SetGPIONDir .....	14
3.5.2 SetGPIO .....	14
3.5.3 GetGPIO .....	14
<b>4. UART API 呼叫流程 .....</b>	<b>15</b>
<b>5. I2C API 呼叫流程 .....</b>	<b>16</b>
<b>6. SPI API 呼叫流程 .....</b>	<b>17</b>

## Revision History

Revision	Description	Date
0.1	➤ Initial version	Jan 30, 2024
0.2	➤ Remove SPISetCSLowHigh function. ➤ Add SPISetCS function. ➤ Add SPIFullDuplexWriteRead function. ➤ Add SPIFullDuplexWriteRead2 function.	Feb 06, 2024
0.3	➤ Add UART and I2C function.	Mar 12, 2024

## 1. 簡介

此 HID to UART/I2C/SPI/GPIO SDK 提供相關應用程式介面用來與 PL2561 HID Mode 溝通。主要提供列舉 HID 裝置與讀寫 HID Report Data。若要用在多執行緒(Multi-Thread)，會同時呼叫相關函式，則請加上 Critical Section 來呼叫，用來實現執行緒安全(Thread Safety)。

## 2. SDK 檔案

PL2561 SDK 支援作業系統及檔案說明如下：

- 作業系統：Ubuntu 16.04
- 程式庫：libHidSdkApi.so
- 標題檔：HidSdkApi.h

## 3. SDK 函數

### 3.1 列舉裝置相關函數名稱及說明

函數名稱	說明
GetSDKVersion	取得 SDK 版本
InitDevice	初始化 SDK
ExitDevice	離開 SDK
GetDeviceHandleListByVid	取得符合VID的HID Device列表
GetDeviceHandleListByVidPid	取得符合VID與PID的HID Device列表
FreeDeviceHandleList	釋放取得的HID Device列表資源
OpenDeviceHandle	連接 HID 裝置
CloseDeviceHandle	關閉 HID 裝置
GetDeviceInfo	取得 HID Device 的信息

#### 3.1.1 GetSDKVersion

- 敘述：取得 SDK 版本，如版號為 v1.0.0.2，會回傳 0x01000002
- 語法：void GetSDKVersion(uint32\_t\* SdkVersion)
- 參數  
輸出：SdkVersion
- 回傳值：0 ➔ 成功

#### 3.1.2 InitDevice

- 敘述：初始化 SDK
- 語法：int32\_t InitDevice(void)
- 參數：無
- 回傳值：0 ➔ 成功

#### 3.1.3 ExitDevice

- 敘述：離開 SDK
- 語法：void ExitDevice(void)
- 參數：無
- 回傳值：無

#### 3.1.4 GetDeviceHandleListByVid

- 敘述：取得符合 VID 的 HID Device 列表
- 語法：int32\_t GetDeviceHandleListByVid(uint16\_t VID, HANDLE \*\*DeviceHandleList, uint8\_t \*DeviceCount);
- 參數：  
輸入：VID  
輸出：HID Device list  
輸出：HID Device count
- 回傳值：0 ➔ 成功

### 3.1.5 GetDeviceHandleListByVidPid

- 敘述: 取得符合 VID 和 PID 的 HID Device 列表
- 語法: `int32_t GetDeviceHandleListByVidPid(uint16_t VID, uint16_t PID, HANDLE **DeviceHandleList, uint8_t *DeviceCount)`
- 參數:
  - 輸入: VID
  - 輸入: PID
  - 輸出: HID Device list
  - 輸出: HID Device count
- 回傳值: 0 ➔ 成功

### 3.1.6 FreeDeviceHandleList

- 敘述: 釋放取得的 HID Device 列表資源
- 語法: `void FreeDeviceHandleList(HANDLE **DeviceHandleList)`
- 參數:
  - 輸入: HID Device list
- 回傳值: 無

### 3.1.7 OpenDeviceHandle

- 敘述: 連接 HID 裝置
- 語法: `int32_t OpenDeviceHandle(HANDLE *hDeviceHandle)`
- 參數:
  - 輸入: hDeviceHandle
- 回傳值: 0 ➔ 成功

### 3.1.8 CloseDeviceHandle

- 敘述: 關閉 HID 裝置
- 語法: `int32_t CloseDeviceHandle(HANDLE *hDeviceHandle)`
- 參數:
  - 輸入: hDeviceHandle
- 回傳值: 0 ➔ 成功

### 3.1.9 GetDeviceInfo

- 敘述: 取得 HID Device 的信息
- 語法: `int32_t GetDeviceInfo(HANDLE *hDeviceHandle, struct HIDDeviceInfo *DeviceInfo)`
- 參數:
  - 輸入: hDeviceHandle
  - 輸出: DeviceInfo
- 回傳值: 0 ➔ 成功

## 3.2 UART 相關函數名稱及說明

函數名稱	說明
GetUartConfig	取得 Uart 設定值
SetUartConfig	設定 Uart 設定值
UartRead	讀取 Uart data
UartWrite	寫入 Uart data
UartReset	重置 Uart Interface

### 3.2.1 GetUartConfig

- 敘述: 取得 Uart 設定值
- 語法: int32\_t GetUartConfig(HANDLE \*hDeviceHandle, uint32\_t \*BaudRate, enum UART\_STOP\_BIT \*StopBit, enum UART\_PARITY\_TYPE \*ParityType, enum UART\_DATA\_BIT \*Databit)
- 參數:
  - 輸入: hDeviceHandle
  - 輸出: BaudRate. Baud Rate, 單位為 bps
  - 輸出: StopBit, Stop Bit(1/1.5/2)
  - 輸出: ParityType, Parity Type(None/Odd/Even/Mark/Space)
  - 輸出: Databit, Data Bits(5/6/7/8)
- 回傳值: 0 ➔ 成功

### 3.2.2 SetUartConfig

- 敘述: 設定 Uart 設定值, 包含 BaudRate, StopBit, Parity, DataBits
- 語法: int32\_t SetUartConfig(HANDLE \*hDeviceHandle, uint32\_t BaudRate, enum UART\_STOP\_BIT StopBit, enum UART\_PARITY\_TYPE ParityType, enum UART\_DATA\_BIT Databit)
- 參數:
  - 輸入: hDeviceHandle
  - 輸入: BaudRate. Baud Rate, 單位為 bps, 最快 12,000,000bps
  - 輸入: StopBit, Stop Bit(1/1.5/2)
  - 輸入: ParityType, Parity Type(None/Odd/Even/Mark/Space)
  - 輸入: Databit, Data Bits(5/6/7/8)
- 回傳值: 0 ➔ 成功

### 3.2.3 UartRead

- 敘述: 讀取 Uart data
- 語法: int32\_t UartRead (HANDLE \*hDeviceHandle, uint8\_t \*Buffer, uint16\_t NumberOfBytesToRead, uint16\_t \*NumberOfBytesRead, uint32\_t TimeOutms)
- 參數:
  - 輸入: hDeviceHandle
  - 輸出: Buffer 讀取資料緩衝區
  - 輸入: NumberOfBytesToRead 讀取資料的位元數
  - 輸出: NumberOfBytesRead 實際讀取資料的位元數
  - 輸入: TimeOutms 逾時時間, 單位為 milliSecond
- 回傳值: 0 ➔ 成功

#### 3.2.4 UartWrite

- 敘述: 寫入 Uart data
- 語法: `int32_t UartWrite(HANDLE *hDeviceHandle, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `Buffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `NumberOfBytesWritten` 實際寫入資料的位元數
- 回傳值: 0 ➔ 成功

#### 3.2.5 UartReset

- 敘述: 重置 Uart Interface
- 語法: `int32_t UartReset(HANDLE *hDeviceHandle)`
- 參數:
  - 輸入: `hDeviceHandle`
- 回傳值: 0 ➔ 成功



### 3.3 I2C 相關函數名稱及說明

函數名稱	說明
SetI2CFrequency	設定 I2C 頻率
I2CRead	讀取 I2C data
I2CWrite	寫入 I2C data
I2CWriteRead	寫入並讀取 I2C data
I2CReset	重置 I2C Master Interface

#### 3.3.1 SetI2CFrequency

- 敘述: 設定 I2C 頻率
- 語法: `int32_t SetI2CFrequency(HANDLE *hDeviceHandle, uint8_t FreqDiv)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `FreqDiv` I2C 頻率除數
  - $I2C\ Frequency(KHz) = 96000 / ((FreqDiv * 2 + 1) * 2)$ , `FreqDiv` 需大於等於 24
- 回傳值: 0 ➔ 成功

#### 3.3.2 I2CRead

- 敘述: 讀取 I2C data
- 語法: `int32_t I2CRead (HANDLE *hDeviceHandle, uint8_t DeviceAddr, uint8_t *Buffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `DeviceAddr` I2C Device Address, 如輸入 0xA0
  - 輸出: `Buffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間, 單位為 `milliSecond`
- 回傳值: 0 ➔ 成功

#### 3.3.3 I2CWrite

- 敘述: 寫入 I2C data
- 語法: `int32_t I2CWrite (HANDLE *hDeviceHandle, uint8_t DeviceAddr, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `DeviceAddr` I2C Device Address
  - 輸入: `Buffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `NumberOfBytesWritten` 實際寫入資料的位元數
- 回傳值: 0 ➔ 成功

### 3.3.4 I2CWriteRead

- 敘述: 寫入並讀取 I2C data
- 語法: `int32_t I2CWriteRead(HANDLE *hDeviceHandle, uint8_t DeviceAddr, uint8_t *WriteBuffer, uint16_t NumberOfBytesToWrite, uint8_t *ReadBuffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `DeviceAddr` I2C Device Address
  - 輸入: `WriteBuffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `ReadBuffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間, 單位為 `milliSecond`
- 回傳值: 0 ➔ 成功

### 3.3.5 I2CReset

- 敘述: 重置 I2C Master Interface
- 語法: `int32_t I2CReset(HANDLE *hDeviceHandle)`
- 參數:
  - 輸入: `hDeviceHandle`
- 回傳值: 0 ➔ 成功

### 3.4 SPI 相關函數名稱及說明

函數名稱	說明
SetSPIFrequency	設定 SPI 頻率
SetSPIMode	設定 SPI 模式
SPIRead	讀取 SPI data
SPIWrite	寫入 SPI data
SPIWriteRead	寫入並讀取 SPI data
SPIFullDuplexWriteRead	用全雙工的方式寫入並讀取 SPI data
SPIFullDuplexWriteRead2	用全雙工的方式寫入並讀取 SPI data，單筆資料大於 506 bytes 時使用
SPISetCS	設定 CS 的準位
SPIReset	重置 SPI Interface

#### 3.4.1 SetSPIFrequency

- 敘述: 設定 SPI 頻率
- 語法: `int32_t SetSPIFrequency(HANDLE *hDeviceHandle, uint8_t FreqDiv)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `FreqDiv` SPI 頻率除數
  - $\text{SPI Frequency(KHz)} = 64000 / (2 * \text{FreqDiv})$ , `FreqDiv` 需大於等於 1, 頻率最快為 32000 KHz
- 回傳值: 0 ➔ 成功

#### 3.4.2 SetSPIMode

- 敘述: 設定 SPI 模式
- 語法: `int32_t SetSPIMode(HANDLE *hDeviceHandle, enum SPI_MODE spiMode)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `spiMode` ➔ `SPI_MODE_0`, `SPI_MODE_1`, `SPI_MODE_2`, `SPI_MODE_3`
- 回傳值: 0 ➔ 成功

#### 3.4.3 SPIRead

- 敘述: 讀取 SPI data
- 語法: `int32_t SPIRead(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *Buffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` ➔ `CS0` or `CS1`
  - 輸出: `Buffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間, 單位為 `milliSecond`
- 回傳值: 0 ➔ 成功

#### 3.4.4 SPIWrite

- 敘述: 寫入 SPI data
- 語法: `int32_t SPIWrite(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` → CS0 or CS1
  - 輸入: `Buffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `NumberOfBytesWritten` 實際寫入資料的位元數
- 回傳值: 0 → 成功

#### 3.4.5 SPIWriteRead

- 敘述: 寫入並讀取 SPI data
- 語法: `int32_t SPIWriteRead(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *WriteBuffer, uint16_t NumberOfBytesToWrite, uint8_t *ReadBuffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` → CS0 or CS1
  - 輸入: `WriteBuffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `ReadBuffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間, 單位為 millisecond
- 回傳值: 0 → 成功

#### 3.4.6 SPIFullDuplexWriteRead

- 敘述: 用全雙工的方式寫入並讀取 SPI data
- 語法: `int32_t SPIFullDuplexWriteRead(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *WriteBuffer, uint16_t NumberOfBytesToWrite, uint8_t *ReadBuffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` → CS0 or CS1
  - 輸入: `WriteBuffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `ReadBuffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間, 單位為 millisecond
- 回傳值: 0 → 成功

### 3.4.7 SPIFullDuplexWriteRead2

- 敘述: 用全雙工的方式寫入並讀取 SPI data，單筆資料大於 506 bytes 時使用
- 語法: `int32_t SPIFullDuplexWriteRead2(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *WriteBuffer, uint32_t NumberOfBytesToWrite, uint8_t *ReadBuffer, uint32_t NumberOfBytesToRead, uint32_t *NumberOfBytesRead, uint32_t TimeOutms)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` → CS0 or CS1
  - 輸入: `WriteBuffer` 寫入的資料
  - 輸入: `NumberOfBytesToWrite` 寫入資料的位元數
  - 輸出: `ReadBuffer` 讀取資料緩衝區
  - 輸入: `NumberOfBytesToRead` 讀取資料的位元數
  - 輸出: `NumberOfBytesRead` 實際讀取資料的位元數
  - 輸入: `TimeOutms` 逾時時間，單位為 `millisecond`
- 回傳值: 0 → 成功

### 3.4.8 SPISetCS

- 敘述: 設定 CS 的準位
- 語法: `int32_t SPISetCS(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, enum SPI_CS_STATE CSSState)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `SelectSPI` → CS0 or CS1
  - 輸入: `CSSState` CS 的準位
- 回傳值: 0 → 成功

### 3.4.9 SPIReset

- 敘述: 重置 SPI Interface
- 語法: `int32_t SPIReset(HANDLE *hDeviceHandle)`
- 參數:
  - 輸入: `hDeviceHandle`
- 回傳值: 0 → 成功

### 3.5 GPIO 相關函數名稱及說明

函數名稱	說明
SetGPIONDir	設定 GPIO 為輸入或輸出
SetGPIO	寫 GPIO
GetGPIO	讀 GPIO

#### 3.5.1 SetGPIONDir

- 敘述: 設定 GPIO 為輸入或輸出
- 語法: `int32_t SetGPIONDir(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, enum GPIO_DIR GPIONDir)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `Channel`, device 的 channel
  - 輸入: `PortNumber`, channel 的 port
  - 輸入: `GPIONDir`, 輸入或輸出
- 回傳值: 0 ➔ 成功

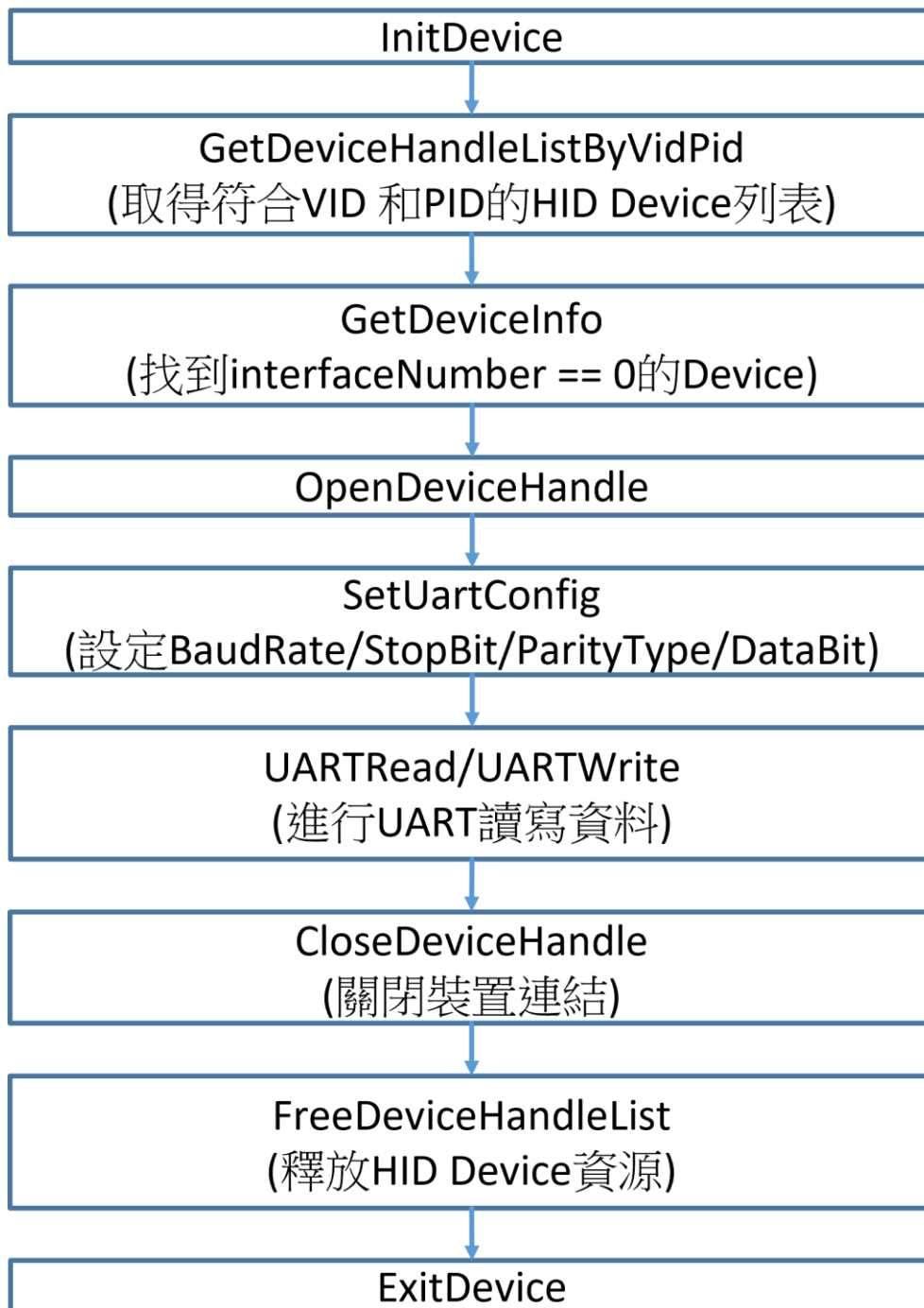
#### 3.5.2 SetGPIO

- 敘述: 寫 GPIO
- 語法: `int32_t SetGPIO(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, enum GPIO_VALUE Value)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `Channel`, device 的 channel
  - 輸入: `PortNumber`, channel 的 port
  - 輸入: `Value`, low 或 high
- 回傳值: 0 ➔ 成功

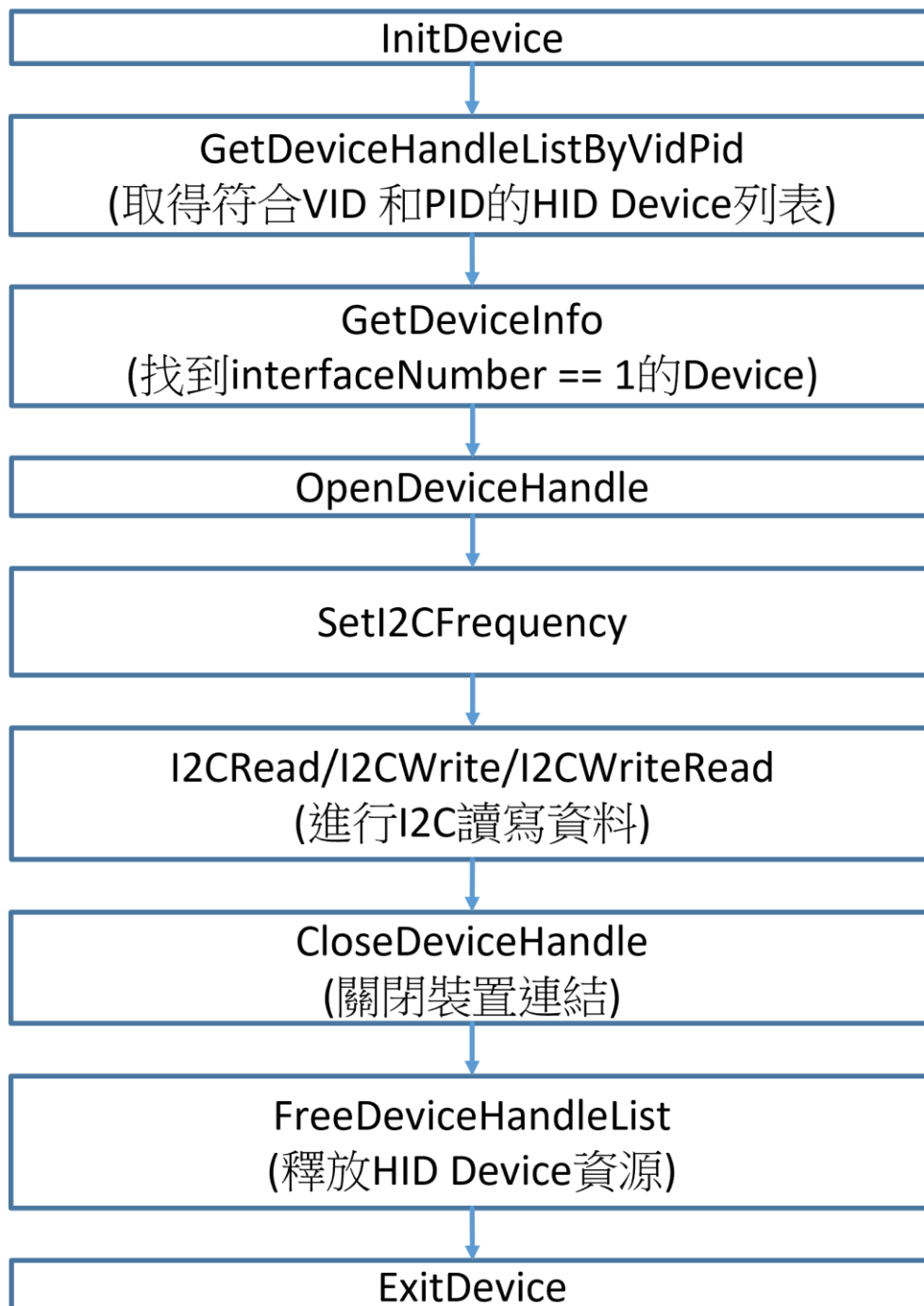
#### 3.5.3 GetGPIO

- 敘述: 讀 GPIO
- 語法: `int32_t GetGPIO(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, uint8_t *Value)`
- 參數:
  - 輸入: `hDeviceHandle`
  - 輸入: `Channel`, device 的 channel
  - 輸入: `PortNumber`, channel 的 port
  - 輸出: `Value`, 0 或 1
- 回傳值: 0 ➔ 成功

#### 4. UART API 呼叫流程

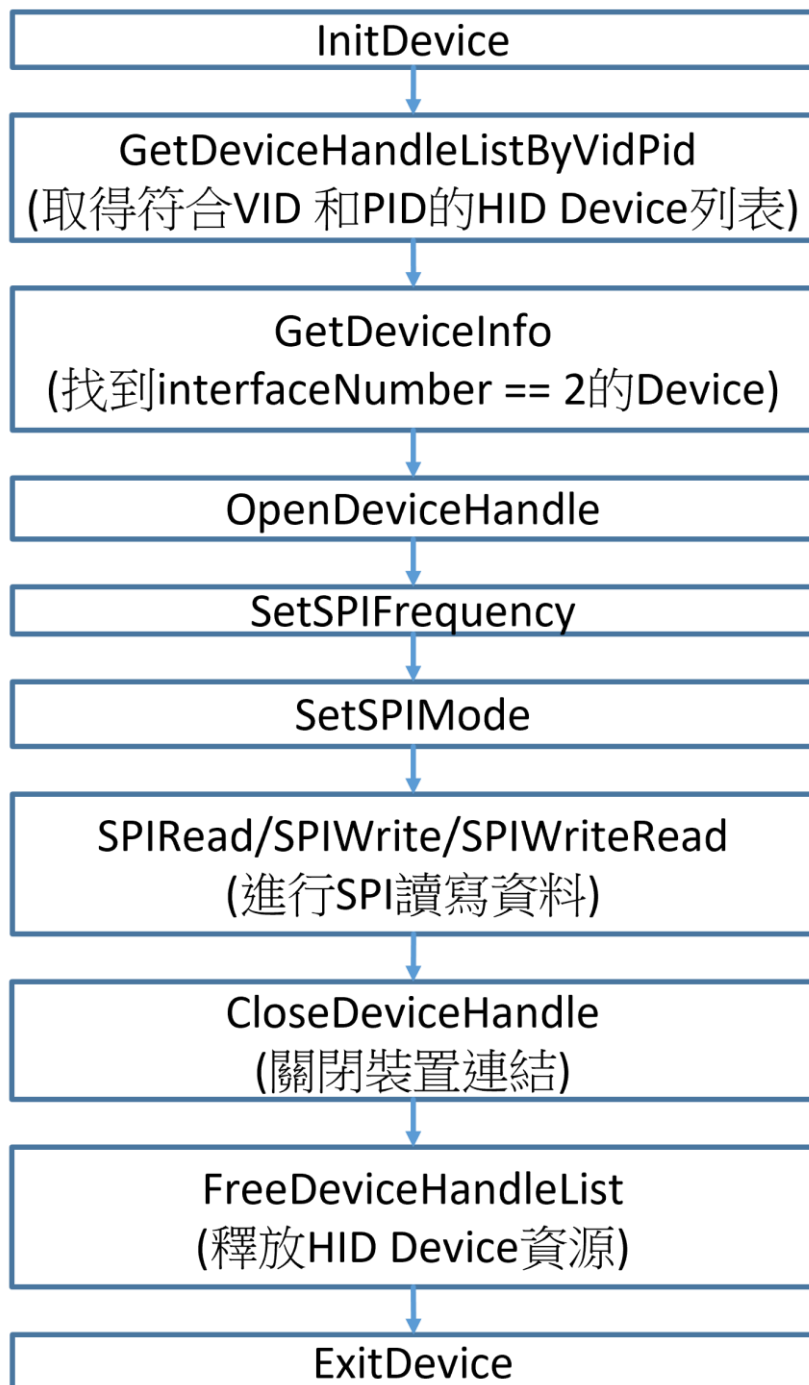


## 5. I2C API 呼叫流程





## 6. SPI API 呼叫流程



## **Disclaimer**

All the information in this document is subject to change without prior notice. Prolific Technology Inc. does not make any representations or any warranties (implied or otherwise) regarding the accuracy and completeness of this document and shall in no event be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages.

## **Trademarks**

The Prolific logo is a registered trademark of Prolific Technology Inc. All brand names and product names used in this document are trademarks or registered trademarks of their respective holders.

## **Copyrights**

**Copyright © 2016 Prolific Technology Inc. All rights reserved.**

No part of this document may be reproduced or transmitted in any form by any means without the express written permission of Prolific Technology Inc.