



PL2561

HID to UART/I2C/SPI/GPIO

Mac SDK Programming Guide

Document Revision: 0.1
Document Release: July 28, 2025

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

Table of Contents

Revision History	3
1. Introduction.....	4
2. SDK Library File.....	5
3. SDK API Function	5
3.1 Enum Device API Function and Description	5
3.1.1 GetSDKVersion	5
3.1.2 GetDeviceHandleListByVid	5
3.1.3 GetDeviceHandleListByVidPid	5
3.1.4 FreeDeviceHandleList	6
3.1.5 OpenDeviceHandle	6
3.1.6 CloseDeviceHandle	6
3.1.7 GetDeviceInfo	6
3.2 UART API Function and Description	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 API Function and Description	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 API Function and Description.....	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 API Function and Description.....	14
3.5.1 SetGPIONDir.....	14
3.5.2 SetGPIO	14
3.5.3 GetGPIO	14
4. UART API Function Call Flow Diagram.....	15
5. I2C API Function Call Flow Diagram	16
6. SPI API Function Call Flow Diagram	17

Revision History

Revision	Description	Date
0.1	➤ Initial version.	July 28, 2025

1. Introduction

This HID to UART/I2C/SPI/GPIO SDK API Programming Guide provides simple API (application programming interface) for developers to communicate with the PL2561 HID to UART/I2C/SPI/GPIO device. It provides how to enumerate HID devices and read/write HID Report Data.

Note that when using the library functions simultaneously in multiple threads, the developer must implement thread safety and call the library functions from within a critical section such that only one single function is call at any given time.

2. SDK Library File

PL2561 SDK supported operating system and file description:

- Operating System : macOS 10.9
- Library : HidSdkApi.framework
- Include File : HidSdkApi.h

3. SDK API Function

3.1 Enum Device API Function and Description

Function	Description
GetSDKVersion	Get SDK Version
GetDeviceHandleListByVid	Get the list of HID Device matching the VID.
GetDeviceHandleListByVidPid	Get the list of HID Device matching the VID and PID.
FreeDeviceHandleList	Release the obtained HID Device list resources
OpenDeviceHandle	Open HID device
CloseDeviceHandle	Close HID device
GetDeviceInfo	Get the information of the HID device

3.1.1 GetSDKVersion

- Description: Get SDK Version, If Version is V1.0.0.2, it will return 0x01000002
- Syntax: void GetSDKVersion(uint32_t* SdkVersion)
- Parameters:
 - Output: SdkVersion
- Return Value: 0 ➔ successfully

3.1.2 GetDeviceHandleListByVid

- Description: Get the list of HID Device matching the VID
- Syntax: int32_t GetDeviceHandleListByVid(uint16_t VID, HANDLE **DeviceHandleList, uint8_t *DeviceCount)
- Parameters:
 - Input: VID
 - Output: HID Device list
 - Output: HID Device count
- Return Value: 0 ➔ successfully

3.1.3 GetDeviceHandleListByVidPid

- Description: Get the list of HID Device matching the VID and PID
- Syntax: int32_t GetDeviceHandleListByVidPid(uint16_t VID, uint16_t PID, HANDLE **DeviceHandleList, uint8_t *DeviceCount)
- Parameters:
 - Input: VID
 - Input: PID
 - output: HID Device list
 - output: HID Device count
- Return Value: 0 ➔ successfully

3.1.4 FreeDeviceHandleList

- Description: Release the obtained HID Device list resources
- Syntax: void FreeDeviceHandleList(HANDLE **DeviceHandleList)
- Parameters:
 - Input: HID Device list
- Return Value: none

3.1.5 OpenDeviceHandle

- Description: Open HID device
- Syntax: int32_t OpenDeviceHandle(HANDLE *hDeviceHandle)
- Parameters:
 - Input: hDeviceHandle
- Return Value: 0 → successfully

3.1.6 CloseDeviceHandle

- Description: Close HID device.
- Syntax: int32_t CloseDeviceHandle(HANDLE *hDeviceHandle)
- Parameters:
 - Input: hDeviceHandle
- Return Value: 0 → successfully

3.1.7 GetDeviceInfo

- Description: Get the information of the HID device
- Syntax: int32_t GetDeviceInfo(HANDLE *hDeviceHandle, struct HIDDeviceInfo *DeviceInfo)
- Parameters:
 - Input: hDeviceHandle
 - Output: DeviceInfo
- Return Value: 0 → successfully

3.2 UART API Function and Description

Function	Description
GetUartConfig	Get Uart setting value
SetUartConfig	Set Uart setting value
UartRead	Read Uart data
UartWrite	Write Uart data
UartReset	Reset Uart Interface

3.2.1 GetUartConfig

- Description: Get Uart setting value
- Syntax: int32_t GetUartConfig(HANDLE *hDeviceHandle, uint32_t *BaudRate, enum UART_STOP_BIT *StopBit, enum UART_PARITY_TYPE *ParityType, enum UART_DATA_BIT *Databit)
- Parameters:
 - Input: hDeviceHandle
 - Output: BaudRate, Baud Rate, unit in bps
 - Output: StopBit, Stop Bit (1/1.5/2)
 - Output: ParityType, Parity Type (None/Odd/Even/Mark/Space)
 - Output: Databit, Data Bits (5/6/7/8)
- Return Value: 0 ➔ successfully

3.2.2 SetUartConfig

- Description: Set Uart setting value, Including BaudRate, Stop Bit, Parity, DataBits
- Syntax: int32_t SetUartConfig(HANDLE *hDeviceHandle, uint32_t BaudRate, enum UART_STOP_BIT StopBit, enum UART_PARITY_TYPE ParityType, enum UART_DATA_BIT Databit)
- Parameters:
 - Input: hDeviceHandle
 - Input: BaudRate, Baud Rate, unit in bps, maximum 115,200bps
 - Input: StopBit, Stop Bit(1/1.5/2)
 - Input: ParityType, Parity Type(None/Odd/Even/Mark/Space)
 - Input: Databit, Data Bits(5/6/7/8)
- Return Value: 0 ➔ successfully

3.2.3 UartRead

- Description: Read Uart data
- Syntax: int32_t UartRead (HANDLE *hDeviceHandle, uint8_t *Buffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)
- Parameters:
 - Input: hDeviceHandle
 - Output: Buffer, read data buffer
 - Input: NumberOfBytesToRead, number of bytes read
 - Output: NumberOfBytesRead, actual number of bytes read
 - Input: TimeOutms, timeout, unit in millisecond
- Return Value: 0 ➔ successfully

3.2.4 **UartWrite**

- Description: Write Uart data
- Syntax: int32_t UartWrite(HANDLE *hDeviceHandle, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)
- Parameters:
 - Input: hDeviceHandle
 - Input: Buffer, write data buffer
 - Input: NumberOfBytesToWrite, number of bytes write
 - Output: NumberOfBytesWritten, actual number of bytes write
- Return Value: 0 → successfully

3.2.5 **UartReset**

- Description: Reset Uart Interface
- Syntax: int32_t UartReset(HANDLE *hDeviceHandle)
- Parameters:
 - Input: hDeviceHandle
- Return Value: 0 → successfully

3.3 I2C API Function and Description

Function	Description
SetI2CFrequency	Set I2C frequency
I2CRead	Read I2C data
I2CWrite	Write I2C data
I2CWriteRead	Write and read I2C data
I2CReset	Reset I2C Master Interface

3.3.1 SetI2CFrequency

- Description: Set I2C frequency
- Syntax: `int32_t SetI2CFrequency(HANDLE *hDeviceHandle, uint8_t FreqDiv)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `FreqDiv`, I2C frequency divisor
 - I2C Frequency (KHz) = $96000 / ((FreqDiv * 2 + 1) * 2)$, where `FreqDiv` must be greater than or equal to 24.
- Return Value: 0 → successfully

3.3.2 I2CRead

- Description: Read I2C data
- Syntax: `int32_t I2CRead (HANDLE *hDeviceHandle, uint8_t DeviceAddr, uint8_t *Buffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `DeviceAddress`, I2C Device Address, (ex: input 0xA0)
 - Output: `Buffer`, read data buffer
 - Input: `NumberOfBytesToRead`, number of bytes read
 - Output: `NumberOfBytesRead`, actual number of bytes read
 - Input: `TimeOutms`, timeout, unit in millisecond
- Return Value: 0 → successfully

3.3.3 I2CWrite

- Description: Write I2C data
- Syntax: `int32_t I2CWrite (HANDLE *hDeviceHandle, uint8_t DeviceAddr, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `DeviceAddress`, I2C Device Address
 - Input: `Buffer`, write data buffer
 - Input: `NumberOfBytesToWrite`, number of bytes write
 - Output: `NumberOfBytesWritten`, actual number of bytes write
- Return Value: 0 → successfully

3.3.4 I2CWriteRead

- Description: Write and read I2C data
- Syntax: 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)
- Parameters:
 - Input: hDeviceHandle
 - Input: DeviceAddress, I2C Device Address
 - Input: WriteBuffer, write data buffer
 - Input: NumberOfBytesToWrite, number of bytes write
 - Output: ReadBuffer, read data buffer
 - Input: NumberOfBytesToRead, number of bytes read
 - Output: NumberOfBytesRead, actual number of bytes read
 - Input: TimeOutms, timeout, unit in millisecond
- Return Value: 0 → successfully

3.3.5 I2CReset

- Description: Reset I2C Master Interface
- Syntax: int32_t I2CReset(HANDLE *hDeviceHandle)
- Parameters:
 - Input: hDeviceHandle
- Return Value: 0 → successfully

3.4 SPI API Function and Description

Function	Description
SetSPIFrequency	Set SPI frequency
SetSPIMode	Set SPI mode
SPIRead	Read SPI data
SPIWrite	Write SPI data
SPIWriteRead	Write and read SPI data
SPIFullDuplexWriteRead	Write and read SPI data in full-duplex mode
SPIFullDuplexWriteRead2	Write and read SPI data in full-duplex mode, used when a single data packet exceeds 506 bytes.
SPISetCS	Set CS signal level.
SPIReset	Reset SPI Interface

3.4.1 SetSPIFrequency

- Description: Set SPI frequency
- Syntax: `int32_t SetSPIFrequency(HANDLE *hDeviceHandle, uint8_t FreqDiv)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `FreqDiv`, SPI frequency divisor
 - SPI Frequency (KHz) = $64000/(2*FreqDiv)$, `FreqDiv` need greater than 1, maximum frequency is 32000KHz
 - Return Value: 0 ➔ successfully

3.4.2 SetSPIMode

- Description: Set SPI mode
- Syntax: `int32_t SetSPIMode(HANDLE *hDeviceHandle, enum SPI_MODE spiMode)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `spiMode` ➔ `SPI_MODE_0`, `SPI_MODE_1`, `SPI_MODE_2`, `SPI_MODE_3`
 - Return Value: 0 ➔ successfully

3.4.3 SPIRead

- Description: Read SPI data
- Syntax: `int32_t SPIRead(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *Buffer, uint16_t NumberOfBytesToRead, uint16_t *NumberOfBytesRead, uint32_t TimeOutms)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `SPISelect` ➔ `CS0` or `CS1`
 - Output: `Buffer`, read data buffer
 - Input: `NumberOfBytesToRead`, number of bytes read
 - Output: `NumberOfBytesRead`, actual number of bytes read
 - Input: `TimeOutms`, timeout, unit in millisecond
 - Return Value: 0 ➔ successfully

3.4.4 SPIWrite

- Description: Write SPI data
- Syntax: int32_t SPIWrite(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, uint8_t *Buffer, uint16_t NumberOfBytesToWrite, uint16_t *NumberOfBytesWritten)
- Parameters:
 - Input: hDeviceHandle
 - Input: SPISelect → CS0 or CS1
 - Input: Buffer, write data buffer
 - Input: NumberOfBytesToWrite, number of bytes write
 - Output: NumberOfBytesWritten, actual number of bytes write
- Return Value: 0 → successfully

3.4.5 SPIWriteRead

- Description: Write and read SPI data
- Syntax: 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)
- Parameters:
 - Input: hDeviceHandle
 - Input: SelectSPI → CS0 or CS1
 - Input: WriteBuffer, write data buffer
 - Input: NumberOfBytesToWrite, number of bytes write
 - Output: ReadBuffer, read data buffer
 - Input: NumberOfBytesToRead, number of bytes read
 - Output: NumberOfBytesRead, actual number of bytes read
 - Input: TimeOutms, timeout, unit in millisecond
- Return Value: 0 → successfully

3.4.6 SPIFullDuplexWriteRead

- Description: Write and read SPI data in full-duplex mode
- Syntax: 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)
- Parameters:
 - Input: hDeviceHandle
 - Input: SelectSPI → CS0 or CS1
 - Input: WriteBuffer, write data buffer
 - Input: NumberOfBytesToWrite, number of bytes write
 - Output: ReadBuffer, read the data buffer
 - Input: NumberOfBytesToRead, number of bytes read
 - Output: NumberOfBytesRead, actual number of bytes read
 - Input: TimeOutms, timeout, unit in millisecond.
- Return Value: 0 → successfully

3.4.7 SPIFullDuplexWriteRead2

- Description: Write and read SPI data in full-duplex mode, used when a single data packet exceeds 506 bytes.
- Syntax: `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)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `SelectSPI` → CS0 or CS1
 - Input: `WriteBuffer`, write data buffer
 - Input: `NumberOfBytesToWrite`, number of bytes write
 - Output: Read the data buffer by `ReadBuffer`
 - Input: `NumberOfBytesToRead`, number of bytes read
 - Output: `NumberOfBytesRead`, actual number of bytes read
 - Input: `TimeOutms`, timeout, unit in millisecond.
- Return Value: 0 → successfully

3.4.8 SPISetCS

- Description: Set CS signal level.
- Syntax: `int32_t SPISetCS(HANDLE *hDeviceHandle, enum SPI_SELECT SelectSPI, enum SPI_CS_STATE CSSState)`
- Parameters:
 - Input: `hDeviceHandle`
 - Input: `SelectSPI` → CS0 or CS1
 - Input: `CSSState`, CS signal level
- Return Value: 0 → successfully

3.4.9 SPIReset

- Description: Reset SPI Interface
- Syntax: `int32_t SPIReset(HANDLE *hDeviceHandle)`
- Parameters:
 - Input: `hDeviceHandle`
- Return Value: 0 → successfully

3.5 GPIO API Function and Description

Function	Description
SetGPIODir	Set GPIO as input or output
SetGPIO	Write GPIO
GetGPIO	Read GPIO

3.5.1 SetGPIODir

- Description: Set GPIO as input or output
- Syntax: int32_t SetGPIODir(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, enum GPIO_DIR GPIODir)
- Parameters:
 - Input: hDeviceHandle
 - Input: Channel, device channel
 - Input: PortNumber, channel port
 - Input: GPIODir, input or output
- Return: 0 ➔ successfully

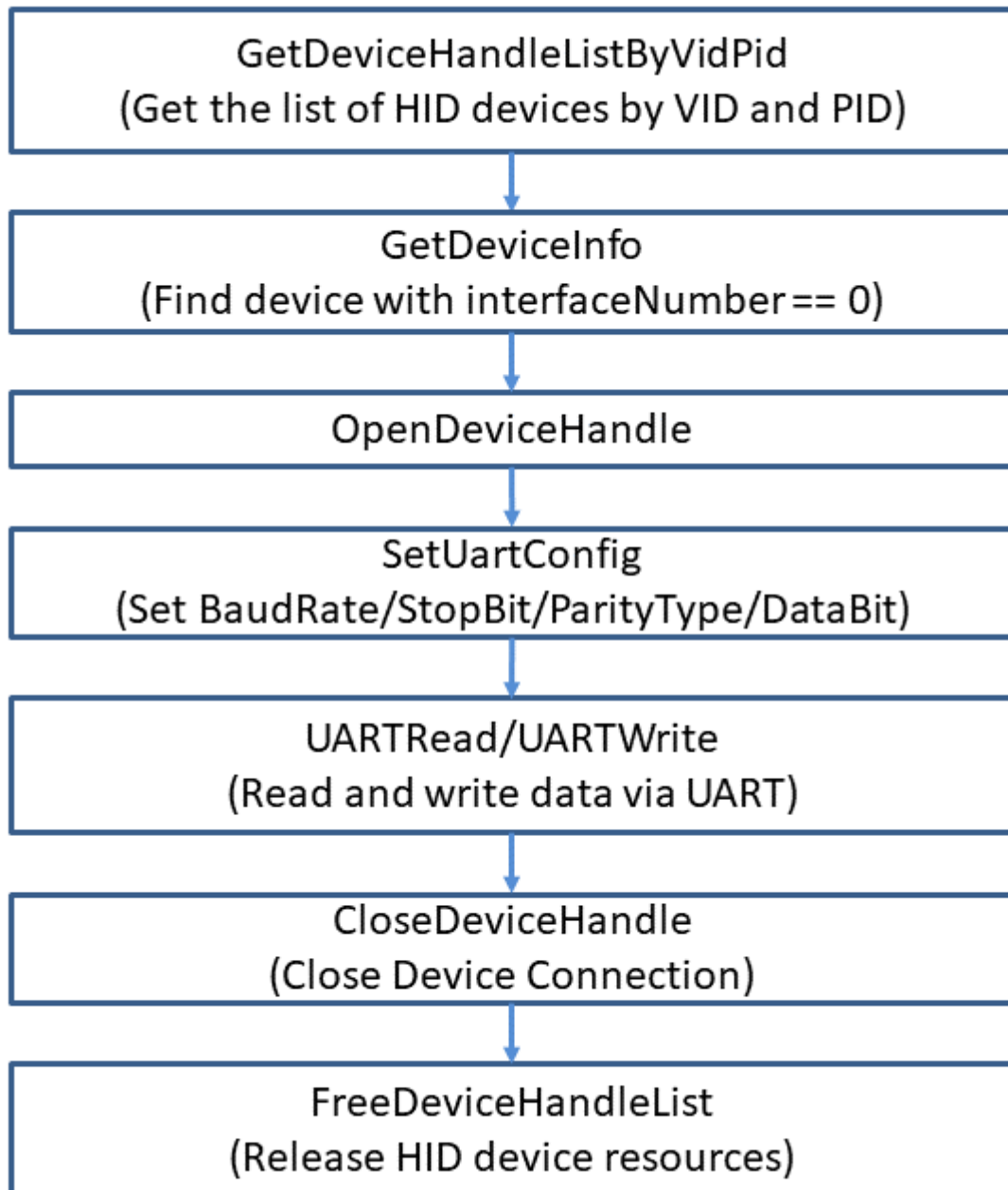
3.5.2 SetGPIO

- Description: Write GPIO
- Syntax: int32_t SetGPIO(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, enum GPIO_VALUE Value)
- Parameters:
 - Input: hDeviceHandle
 - Input: Channel, device channel
 - Input: PortNumber, channel port
 - Input: Value, low or high
- Return: 0 ➔ successfully

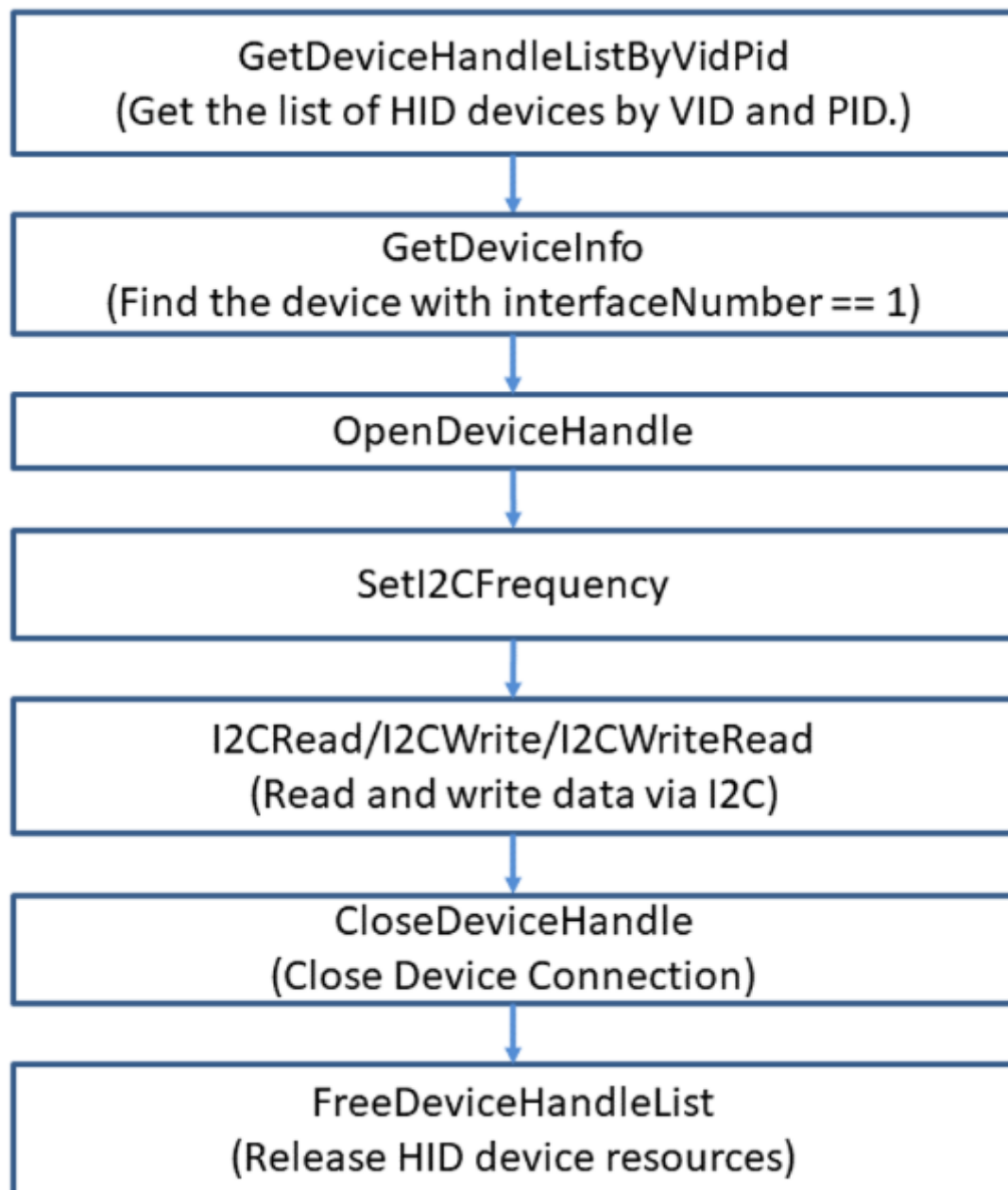
3.5.3 GetGPIO

- Description: Read GPIO
- Syntax: int32_t GetGPIO(HANDLE *hDeviceHandle, enum GPIO_CHANNEL_SELECT Channel, enum GPIO_PORT_NUMBER PortNumber, uint8_t *Value)
- Parameters:
 - Input: hDeviceHandle
 - Input: Channel, device channel
 - Input: PortNumber, channel port
 - Output: Value, 0 or 1
- Return: 0 ➔ successfully

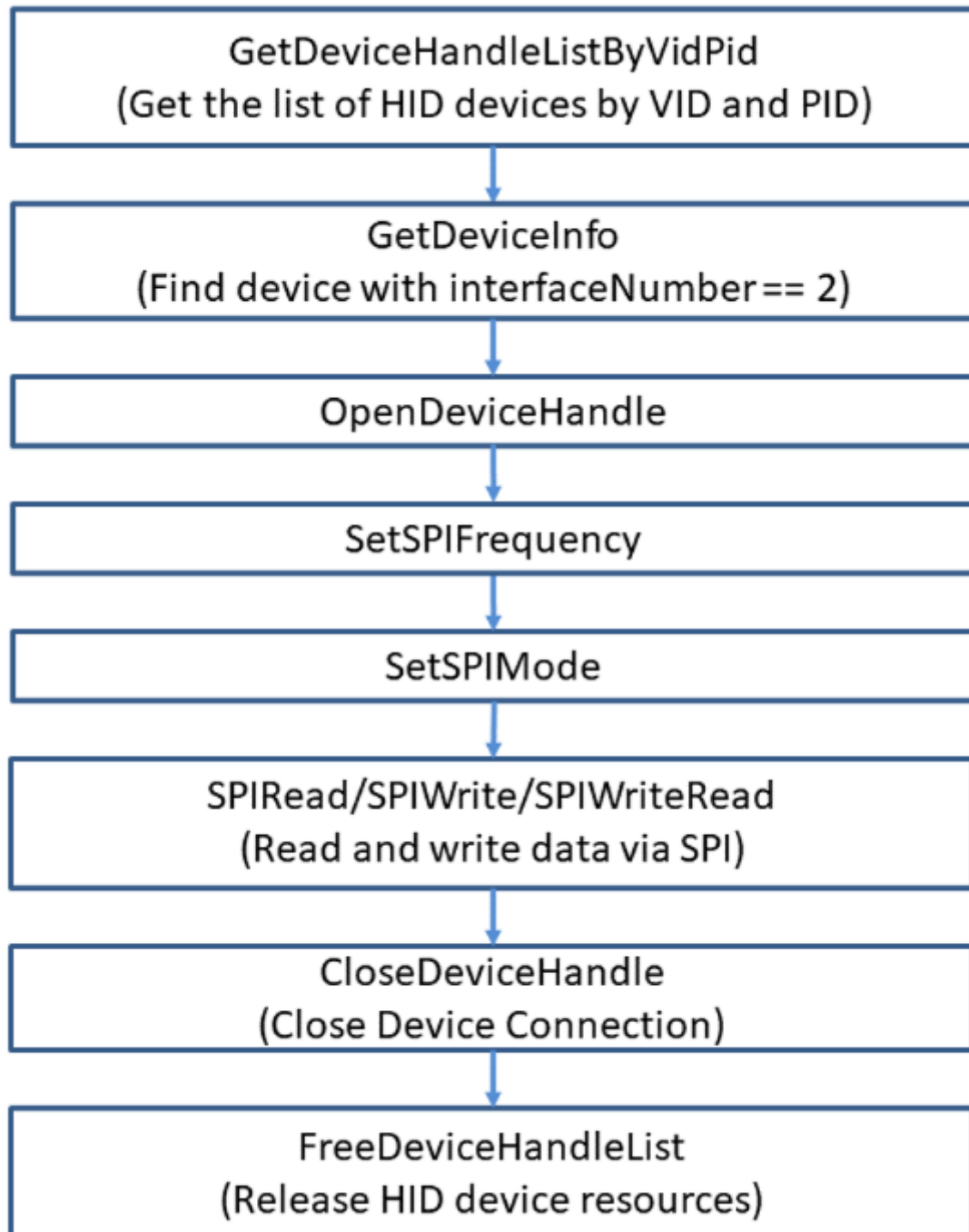
4. UART API Function Call Flow Diagram



5. I2C API Function Call Flow Diagram



6. SPI API Function Call Flow Diagram



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.