第1章 Windowsの標準ドライバがそのまま使える！
市販ブリッジIC相当品を作る

HIDクラスを使った USB-I²Cブリッジ

関本 健太郎

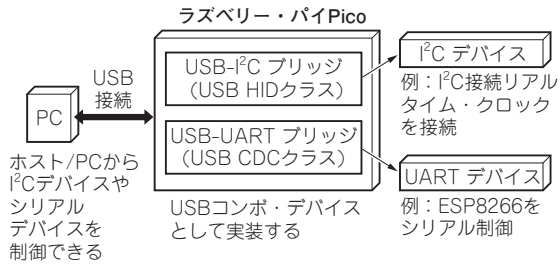


図1 USB-I²C/UARTブリッジの概要

表1 USB-I²Cブリッジ・チップ製品比較

項 目	MCP2221 MCP2221A	CP2112	FT260Q
メーカ名	マイクロチップ・テクノロジー	シリコン・ラボラトリーズ	FTDI
機能	I²C/SMBus, UART (USB CDC), GPIO, 10ビットA-Dコンバータ×3, 5ビットD-Aコンバータ×1, クロック出力	I²C/SMBus, GPIO×8	I²C/SMBus, UART (USB CDC), GPIO×13 (他機能と排他)
USBクラス	HID/CDC	HID	HID/CDC
USBスピード	フルスピード (12Mbps)	フルスピード (12Mbps)	フルスピード (12Mbps)
I/O電圧	3.0V～5.0V	1.8V～V_{DD}	1.8V～3.3V

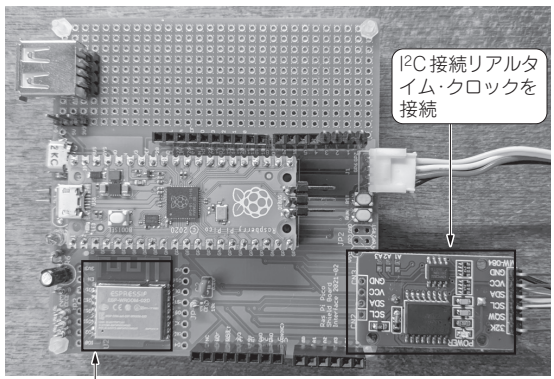


写真1 PicoによるUSB-I²C/UARTブリッジ

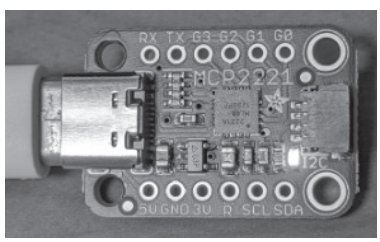


写真2 USB-I²Cブリッジ・チップMCP2221の利用製品例
(Adafruit MCP2221A Breakout)
本章ではMCP2221のUSB-I²Cブリッジ機能に絞った相当品を製作する(MCP2221を用意する必要はない)

本章では，ラズベリー・パイPico(以降，Pico)のUSBデバイス機能の活用例として，USBブリッジを取り上げます．USBブリッジと言えば，USB-シリアル変換の市販品が頭に浮かぶでしょう．次に思いつくものとしては，MCP2221(マイクロチップ・テクノロジー)やCP2112(シリコン・ラボラトリーズ)などのUSB-I²Cブリッジ製品があります(**表1**)．

USB-I²Cブリッジ製品には，

1，USB HIDクラスを利用したもの
2，USB Vendorクラスを利用したもの
3，USB CDCクラスを利用したもの

があります．ここでは例として，

- **1**としてMCP2221(これはデバイス)
- **2**としてI²C-TINY-USB(これはプロジェクト)
- **3**としてFirmata(これはプロトコル)

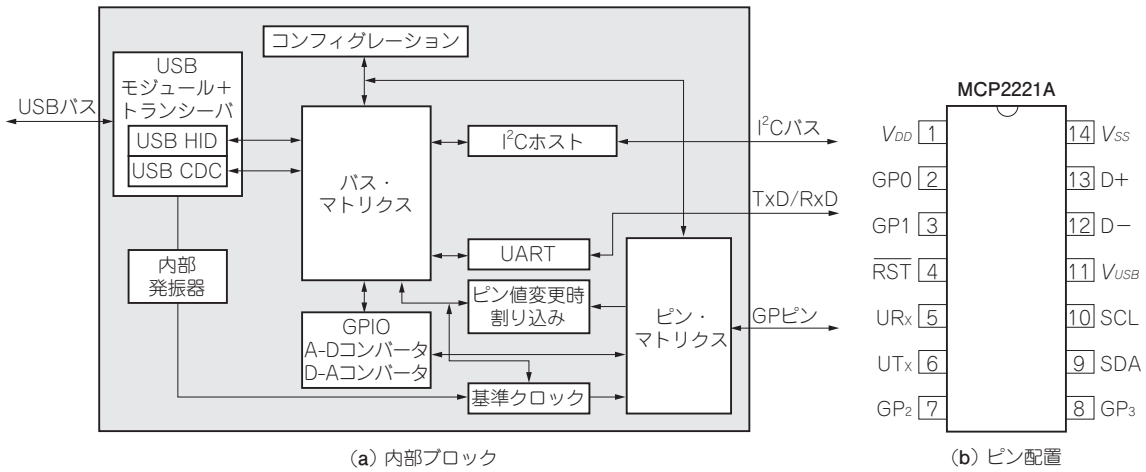
相当品の製作にチャレンジします．なお，誌面の都合で**2**，**3**は次号以降で紹介します．

概要

● Windowsの標準HIDドライバが使用できる

本章では，USB-I²C変換とUSB-UART変換の両方を実現するMCP2221(マイクロチップ・テクノロジー)相当のデバイスを作成します(**図1**)．USB-I²Cブリッジは多くの場合，USBのHIDクラスのデータ通信の仕組みを利用して，ホストPCなどから送信されたI²Cプロトコル処理をI²Cホストとして処理するデバイスです．I²Cチップ制御用のマイコンを使わずに，ホストPCなどからI²Cチップを手軽に制御できます．

◆参考・引用*文献◆
(1) MCP2221Aデータシート，マイクロチップ・テクノロジー．
https://www.microchip.com/downloads/aem



(a) 内部ブロック　　(b) ピン配置

図2(1)　**MCP2221のブロック・ダイヤグラムとパッケージ**

また，HID通信の場合には，Windowsの標準HIDドライバが使用できます．そのため，幾つものUSB-I²CブリッジICが販売されています(**表1**)．用途としては，USBドングル，医療メータ，ハンドヘルド型の制御機器，データ・ロガーなどです．USB-UARTブリッジは，USB-シリアル・コンバータと呼ばれるデバイスの機能です．

ここではUSB-I²Cブリッジ機能の接続例として，I²C接続の(EEPROM 24C32付き)DS1307リアルタイム・クロック・モジュールを紹介します．

USB-UART機能の接続例として，ESP8266のATコマンドの実行を取り上げます(**写真1**)．

● I²C機能を解説する…他にUARTやGPIO機能も備える

ここでは**表1**の製品のうち，MCP2221相当品を作ります(**写真2**)．MCP2221は，USBフルスピード対応です．USB-I²Cブリッジ機能の他に，USB-UARTブリッジ機能やGPIO機能，10ビットA-D変換機能，5ビットのD-A変換機能など(**図2**)も備えていますが，実装の対象はI²C機能に絞っています．言い換えるとMCP2221を模して作るのはI²C機能だけです．

なお，本章ではI²C機能以外にUART機能も実装します．

HIDディスクリプタ

● 構成

HIDディスクリプタは**表2**の形式をしています．

USB HIDデバイスのUSBデバイス・ディスクリプタの標準的な構成は**図3**の通りです．USB HID通信は，ホストとデバイスの間で，HIDディスクリプタを介してデータのやり取りを行います．HIDディスクリプタには，レポート・ディスクリプタ(Report descriptor)と物理ディスクリプタ・セット(Physical descriptor set)があります(**図4**)．

● レポート・ディスクリプタの構造

レポート・ディスクリプタには，Inputレポート，Outputレポート，Featureレポートの3種類があります．Inputレポートは，キーボードのキー情報，マウスの位置，ボタンの押下情報など，デバイスからホストへの情報を報告する目的で利用されます．Outputレポートは，キーボードのLEDの点灯制御など，ホ

表2　HIDディスクリプタ

オフセット	フィールド	サイズ	値の形式	説　明
0	bLength	1	Number	このディスクリプタのバイト・サイズ
1	bDescriptorType	1	Constant	デバイス・ディスクリプタのタイプ
2	bcdHID	2	BCD	HIDクラスの仕様のバージョン(BCD形式)
4	bCountryCode	1	Number	カントリ・コード
5	bNumDescriptors	1	Number	クラス・ディスクリプタの数
6	bDescriptorType	1	Constant	クラス・ディスクリプタのタイプ
7	wDescriptorLength	2	Number	レポート・ディスクリプタのトータル・サイズ
9	[bDescriptorType]…	1	Constant	オプションのディスクリプタ
10	[wDescriptorLength]…	2	Number	オプションのディスクリプタのトータル・サイズ

Documents/documents/APID/Product
Documents/DataSheets/MCP2221A-Data-Sheet
-20005565E.pdf

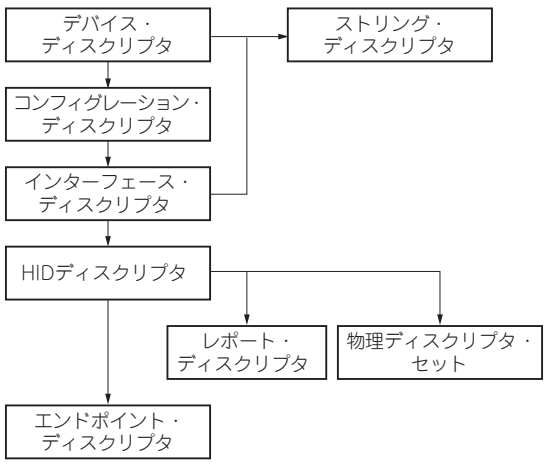


図3　USB HIDデバイスのUSBデバイス・ディスクリプタの標準的な構成

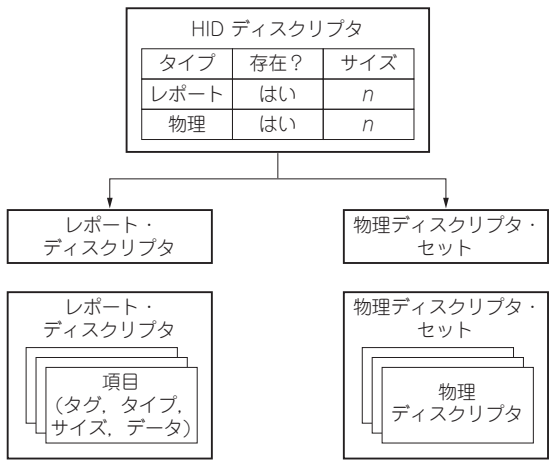


図4　HIDディスクリプタの構成

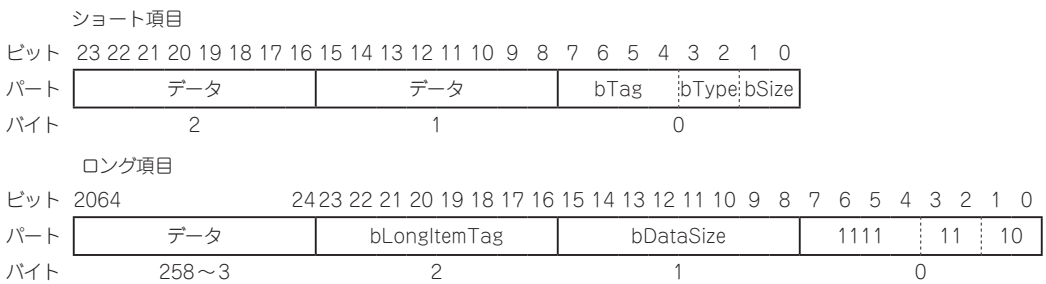


図5[(2)]　**レポート・ディスクリプタの項目の構造**

ストからデバイスへの操作リクエストを行う目的で利用されます．Featureレポートはデバイスのコンフィグレーション情報を双方向でやり取りするなどの目的で利用されます．

レポート・ディスクリプタは，項目（item）と呼ばれる情報片から構成されています．一片の項目は，1バイトのプレフィックス（タグ，タイプ，サイズ）と数バイトのデータという構造です．項目にはショートとロングの2種類があります（**図5**）．4ビットのタグの情報で，項目データの種類が分類されます．

● 物理ディスクリプタ・セット

物理ディスクリプタ・セットは，オプションのディスクリプタです．USB HIDクラスを利用したI²Cブリッジ製品では多くの場合，USB HID通信はHIDのInput/Outputのレポート機能を利用します．

● 作成ツール HID Descriptor Tool

レポート・ディスクリプタは，複雑な形式をしているため，手動で作成するには無理があり，作成ツールが提供されています．具体的には，USB Implementers Forumのウェブ・ページ（`https://www.usb.org/hid#HID`）に，HID Descriptor Toolという，レポート・ディスクリプタを作成，編集および有効性のチェックを行うツールが公開されています（**図6**）．このツールを利用して，独自のレポート・ディスクリプタを作成できます．

USBディスクリプタ

MCP2221には，次の機能が備わります．

1. USB-I²Cブリッジ
2. USB-シリアル変換
3. USB-GPIOブリッジ

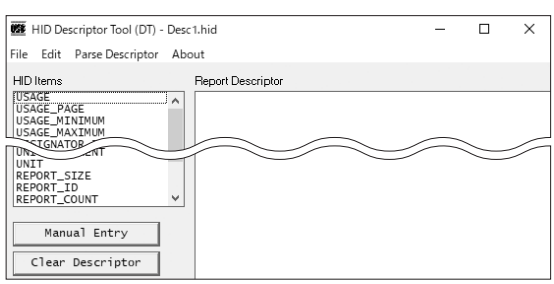


図6　HID Descriptor Tool

(2) Universal Serial Bus (USB) Device Class Definition for Human Interface Devices (HID), Firmware Specification—5/27/01, Version 1.11, USB Implementers' Forum.

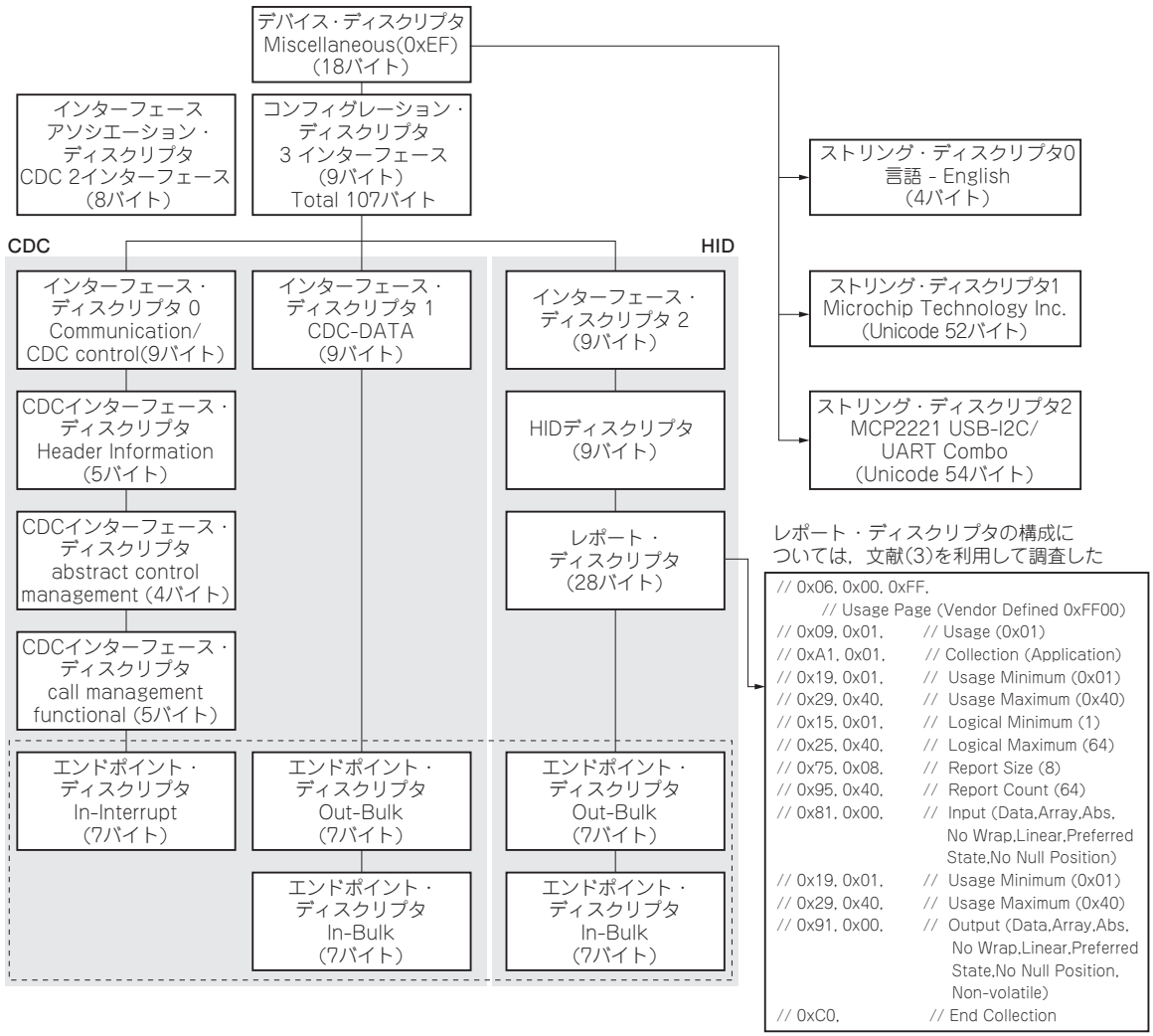


図7　MCP2221のUSBディスクリプタ

4，USB-A-D変換

MCP2221のように，同時に複数の機能を複数のUSBクラスで実装している場合は，USBコンボ・デバイスと呼ばれ，例えばHIDクラスとCDCクラスの2つのインターフェースを持つようにデバイス・ディスクリプタを構成しています(**図7**)．

レポート・ディスクリプタ部分については，MCP2221を模擬するために，現物のMCP2221製品から取得した情報を元に構成しました．HIDレポート・ディスクリプタは，Windowsで動作するUsbTree View(フリーのツール)では取得できず，Linux環境のusbhid-dumpユーティリティを使って取得しました(第1部 第2章コラム参照)．

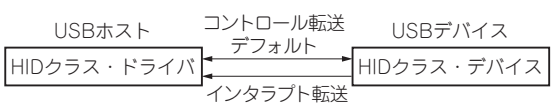


図8　USB HIDクラスの通信方法

● USB HIDクラスの通信方法

USB HIDクラスの通信方法は2つあります．コントロール・パイプ経由のコントロール転送と，インタラプト・パイプ経由のインタラプト転送です(**図8**)．コントロール転送は次の目的で使われます．

- USBコントロールおよびクラス・データのリクエストに対して応答する．
- Get_Reportリクエストを介してHIDクラス・ドライバによってポーリングされたときにデータを送信する
- ホストからデータを受信する

インタラプト転送は次の目的で使われます．

- デバイスから非同期(要求されていない)データを

(3) USB Descriptor and Request Parser.
https://eleccelerator.com/usbdescreqparser/
(4) USB デバイスを接続する，マイクロソフト.

表3(1)　MCP2221のコマンドおよびレスポンスの構成

バイト・インデックス	機能説明	値	効　果
0	−	0x10	ステータス/設定パラメータ-コマンド・コード
1	影響なし	どんな値でも	−
2	現在のI²C/SMBus転送のキャンセル（サブコマンド）	0x10	この値がフィールドに設定されると，デバイスは現在のI²C/SMBusの転送をキャンセルし，I²Cバスの解放を試みる． このコマンドは転送をキャンセルし，I²Cバスを解放するので，非常に便利である．例としては誤ったアドレスを指定しデバイスと通信した際である．結果として，タイムアウトを発生する．タイムアウトは，“ステータス/設定パラメータ”より取得でき，I²C/SMBusの転送キャンセルはこのサブコマンドで達成できる
		他の値	効果なし
3	I²C/SMBus通信スピード	0x20	この値がフィールドに設定されると，デバイスは次のフィールドをI²C/SMBus通信スピードとなるシステム・クロックの分割値として解釈する
		他の値	効果なし
4	I²C/SMBusのシステム・クロックの分割値	−	バイト・インデックス3に新しい通信スピードの値が設定されたときのみ，この値が考慮される．それ以外の場合はこの値は影響を及ぼさない
5～63	影響なし	どんな値でも	−

受信する
・低遅延データをデバイスに送信する

● MCP2221のHIDクラスの通信方法

前節の通りHIDクラスのデバイスは，コントロール転送またはインタラプト転送でHIDの構成およびデータを転送します．MCP2221の場合はコントロール転送でレポート・データを介して，I²Cブリッジ機能の構成およびデータの転送を行います．詳細はMCP2221のデータシート(1)に記載されていますが，その一部を**表3**，**表4**（次頁）に抜粋します．

コマンド構成は64バイト単位で，インデックス0にコマンド，インデックス2以降にコマンドごとに追加情報を指定します．ホストからコマンドを発行し，MCP2221からコマンドに対するレスポンスが返されますが，そのレスポンス構成も，64バイト単位です．インデックス0にコマンド，インデックス1にコマンドの実行結果（完了あるいはその他のステータス），場合によってインデックス4以降にMCP2221からのデータが格納されます．1回のレスポンスでは，MCP2221からデータは最大60バイトまでやり取りできます．60バイトを超えるデータをやり取りする場合には，別のコマンドを発行し，後続のデータのやり取りを行います．TinyUSBライブラリのHIDのコールバック関数を利用して，コマンドのやり取りを実装できます．

USB-I²C/UARTブリッジ機能の実装

● 作成手順

USB-I²C/UARTブリッジ機能の実装は，**図9**の手順で進めます．

▶プログラム・フォルダの作成

TinyUSBのdeviceサンプルのhid_generic_inoutプログラムとcdc_mscプログラムをベースに，USB-I²C/UARTブリッジのプログラムを作成します．projectsフォルダ下にhid_cdcフォルダを新規作成し，hid_generic_inoutフォルダ下のファイルを全てコピーします．

▶コンボ・デバイスのUSBディスクリプタの記述

MCP2221のデバイス・ディスクリプタで説明した構造のUSBディスクリプタをusb_descriptors.cファイル中に記述します．

▶デバイス・ディスクリプタ

USBデバイス・ディスクリプタは，CFG_TUD_HIDとCFG_TUD_CDCが同時に1に設定したときに，USB MISCクラスのコンボ・デバイスとして定義されます（**リスト1**）．

▶レポート・ディスクリプタのマクロ

レポート・ディスクリプタのマクロは，hid_generic_inoutプログラム・フォルダ中のusb_descriptors.cファイルを参照しながら，前述のコンボ・デバイスのディスクリプタ定義（**図7**）を満たすように変更して，TUD_HID_REPORT_DESC_GENERIC_INOUT1として定義しました（**リスト2**）．

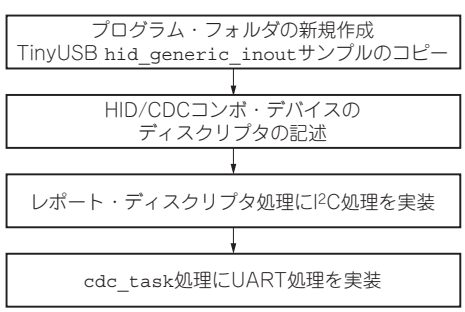


図9　USB-I²C/UARTブリッジ実装手順

(1) https://docs.microsoft.com/ja-jp/windows/wsl/connect-usb

表4　MCP2221の通信仕様

コマンド・コード	コマンド/レスポンス	サブコマンド	説　明	実装	補　足
0x10	Status/Set Parameters	省略	デバイスのステータス取得など	−	−
0xB0	Read Flash Data	0x00	チップのフラッシュ中の設定を取得する	−	−
		0x01	チップのフラッシュ中のGPIOの設定を取得する	−	−
		0x02	USB Manufacturer Descriptor Stringを取得する	○	USBディスクリプタに定義した文字列を返す
		0x03	USB Product Descriptor Stringを取得する	○	USBディスクリプタに定義した文字列を返す
		0x04	USB Serial Number Descriptor Stringを取得する	○	プログラム中に定義した文字列を返す
		0x05	Read Chip Factory Serial Number	○	プログラム中に定義した文字列を返す
0xB1	Write Flash Data	0x00	チップのフラッシュ中の設定に値を書き込む	−	−
		0x01	チップのフラッシュ中のGPIOの設定に値を書き込む	−	−
		0x02	USB Manufacturer Descriptor Stringを書き込む	−	−
		0x03	USB Product Descriptor Stringを書き込む	−	−
		0x04	USB Serial Number Descriptor Stringを書き込む	−	−
0xB2	Send Flash Access Password	−	フラッシュ・アクセス・パスワードを書き込む	−	−
0x90	I²C Write Data	−	スタート・コンディションを発行して，指定したアドレスのI²Cスレーブに指定した長さのデータを書き込む．指定したデータを全て書き込んだ場合のみストップ・コンディションを発行する	○	−
0x92	I²C Write Data Repeated-START	−	リピート・スタート・コンディションを発行して，指定したアドレスのI²Cスレーブに指定した長さのデータを書き込む．指定したデータを全て書き込んだ場合のみストップ・コンディションを発行する	○	−
0x94	I²C Write Data No STOP	−	スタート・コンディションを発行して，指定したアドレスのI²Cスレーブに指定した長さのデータを書き込む．指定したデータを全て書き込んだ場合でもストップ・コンディションを発行しない	○	−
0x91	I²C Read Data	−	スタート・コンディションを発行して，指定したアドレスのI²Cスレーブから指定した長さのデータを読み込む．指定したデータを全て読み込んだ場合のみストップ・コンディションを発行する	○	読み込んだデータはチップ内部に保存される．この時点ではホストにデータは送らない
0x93	I²C Read Data Repeated-START	−	リピート・スタート・コンディションを発行して，指定したアドレスのI²Cスレーブから指定した長さのデータを読み込む．指定したデータを全て読み込んだ場合のみストップ・コンディションを発行する	○	読み込んだデータはチップ内部に保存される この時点ではホストにデータは送らない
0x40	I²C Read Data − Get I²C Data	−	デバイスに一時的に保存されているデータをホストに読み出す	○	−
0x50	Set GPIO Output Values	省略	GP0～GP3のGPIOの制御をする	−	−
0x51	Get GPIO Values	省略	GP0～GP3のGPIOの値を取得する	−	−
0x60	Set SRAM settings	省略	一時的にSRAMの値を変更する	−	−
0x61	Get SRAM Settings	省略	SRAMの設定値を取得する	−	−
0x70	Reset Chip	省略	チップをリセットする	−	−

▶CDCディスクリプタのマクロ

CDCディスクリプタのマクロは，cdc_mscプログラム・フォルダ中のusb_descriptors.cファイルを参照しながら，前述のコンボ・デバイスのディスクリプタ定義(**図7**)を満たすように変更して，TUD_CDC_DESCRIPTOR1として定義しました(**リスト3**)．

▶コンフィグレーション・ディスクリプタの定義

コンフィグレーション・ディスクリプタ，HID，CDCディスクリプタの定義は，desc_configuration配列として，TUD_CONFIG_DESCRIPTOR1マクロ，前述のHIDディスクリプタのマクロ，前述のCDCディスクリプタのマクロを利用して行います(**リスト4**)．

▶hidレポート処理にI²C処理の実装

I²C処理の実装は，TinyUSBフレームワークで定義されているtud_hid_set_report_cbコールバック関数の中で行います(**リスト5**)．TinyUSBフ

リスト1　USB-I²Cブリッジ USBディスクリプタ定義

```
//--------------------------------------------------
// Device Descriptors
//--------------------------------------------------
tusb_desc_device_t const desc_device =
{
    .bLength            = sizeof(tusb_desc_device_t),
    .bDescriptorType    = TUSB_DESC_DEVICE,
    .bcdUSB             = 0x0200,

  // Use Interface Association Descriptor(IAD) for CDC
  // As required by USB Specs IAD's subclass must be
  // common class (2) and protocol must be IAD (1)
#if (CFG_TUD_CDC == 1) && (CFG_TUD_HID == 0)
    .bDeviceClass       = 0x00,
    .bDeviceSubClass    = 0x00,
    .bDeviceProtocol    = 0x00,
#elif (CFG_TUD_CDC == 0) && (CFG_TUD_HID == 1)
    .bDeviceClass       = 0x00,
    .bDeviceSubClass    = 0x00,
    .bDeviceProtocol    = 0x00,
#else
    .bDeviceClass       = TUSB_CLASS_MISC,
    .bDeviceSubClass    = MISC_SUBCLASS_COMMON,
    .bDeviceProtocol    = MISC_PROTOCOL_IAD,
#endif
    .bMaxPacketSize0    = CFG_TUD_ENDPOINT0_SIZE,
    .idVendor           = USB_VID,
    .idProduct          = USB_PID,
    .bcdDevice          = 0x0100,
    .iManufacturer      = 0x01,
    .iProduct           = 0x02,
    .iSerialNumber      = 0x00,
    .bNumConfigurations = 0x01
};
```

リスト2　USB-I²C/UARTブリッジ USB HIDレポート・ディスクリプタ定義

```
#if CFG_TUD_HID
省略
#define TUD_HID_REPORT_DESC_GENERIC_INOUT1(         \
                                  report_size, ...) \
    HID_USAGE_PAGE_N ( HID_USAGE_PAGE_VENDOR, 2   ),\
    HID_USAGE        ( 0x01                       ),\
    HID_COLLECTION   ( HID_COLLECTION_APPLICATION ),\
      /* Report ID if any */\
      __VA_ARGS__ \
      /* Input */ \
      HID_USAGE       ( 0x02                      ),\
      HID_LOGICAL_MIN ( 0x00                      ),\
      HID_LOGICAL_MAX_N ( 0xff, 2                 ),\
      HID_REPORT_SIZE ( 8                         ),\
      HID_REPORT_COUNT( report_size               ),\
      HID_INPUT       ( HID_DATA | HID_VARIABLE |   \
                                   HID_ABSOLUTE ),\
      /* Output */ \
      HID_USAGE       ( 0x03                      ),\
      HID_LOGICAL_MIN ( 0x00                      ),\
      HID_LOGICAL_MAX_N ( 0xff, 2                 ),\
      HID_REPORT_SIZE ( 8                         ),\
      HID_REPORT_COUNT( report_size               ),\
      HID_OUTPUT      ( HID_DATA | HID_VARIABLE |   \
                                   HID_ABSOLUTE ),\
    HID_COLLECTION_END \

省略
uint8_t const desc_hid_report[] =
{
  0x06, 0x00, 0xff, 0x09, 0x01, 0xa1, 0x01, 0x19,
  0x01, 0x29, 0x40, 0x15, 0x01, 0x25, 0x40, 0x75,
  0x08, 0x95, 0x40, 0x81, 0x00, 0x19, 0x01, 0x29,
  0x40, 0x91, 0x00, 0xc0
};
```

リスト3　USB-I²C/UART USB CDCディスクリプタ定義

```
#if CFG_TUD_CDC
// CDC Descriptor Template
// Interface number, string index, EP notification address and size, EP data address (out, in) and size.
#define TUD_CDC_DESCRIPTOR1(_itfnum, _stridx, _ep_notif, _ep_notif_size, _epout, _epin, _epsize) \
  /* Interface Associate */\
  8, TUSB_DESC_INTERFACE_ASSOCIATION, _itfnum, 2, TUSB_CLASS_CDC, CDC_COMM_SUBCLASS_ABSTRACT_CONTROL_MODEL,\
                                                          CDC_COMM_PROTOCOL_ATCOMMAND, 0,\
  /* CDC Control Interface */\
  9, TUSB_DESC_INTERFACE, _itfnum, 0, 1, TUSB_CLASS_CDC, CDC_COMM_SUBCLASS_ABSTRACT_CONTROL_MODEL,         \
                                                      CDC_COMM_PROTOCOL_ATCOMMAND, _stridx,\
  /* CDC Header */\
  5, TUSB_DESC_CS_INTERFACE, CDC_FUNC_DESC_HEADER, U16_TO_U8S_LE(0x0110),\
  /* CDC ACM: support line request */\
  4, TUSB_DESC_CS_INTERFACE, CDC_FUNC_DESC_ABSTRACT_CONTROL_MANAGEMENT, 2,\
  /* CDC Union */\
  5, TUSB_DESC_CS_INTERFACE, CDC_FUNC_DESC_UNION, _itfnum, (uint8_t)((_itfnum) + 1),\
  /* CDC Call */\
  5, TUSB_DESC_CS_INTERFACE, CDC_FUNC_DESC_CALL_MANAGEMENT, 0, (uint8_t)((_itfnum) + 1),\
  /* Endpoint Notification */\
  7, TUSB_DESC_ENDPOINT, _ep_notif, TUSB_XFER_INTERRUPT, U16_TO_U8S_LE(_ep_notif_size), 2,\
  /* CDC Data Interface */\
  9, TUSB_DESC_INTERFACE, (uint8_t)((_itfnum)+1), 0, 2, TUSB_CLASS_CDC_DATA, 0, 0, 0,\
  /* Endpoint Out */\
  7, TUSB_DESC_ENDPOINT, _epout, TUSB_XFER_BULK, U16_TO_U8S_LE(_epsize), 0,\
  /* Endpoint In */\
  7, TUSB_DESC_ENDPOINT, _epin, TUSB_XFER_BULK, U16_TO_U8S_LE(_epsize), 0
#endif
```

レームワークは，HIDデバイスからのUSBインタラプト転送として，SET_REPORTコントロール・リクエストを受け取ったときに呼び出されます．

tud_hid_set_report_cbコールバック関数では，前述のMCP2221のコマンド・データへのポインタが，buffer引数として受け渡され，レスポン

リスト4　USB-I²C/UART USBコンフィグレーション・ディスクリプタ定義

```
uint8_t const desc_configuration[] =
{
  // Config number, interface count, string index, total length, attribute, power in mA
  TUD_CONFIG_DESCRIPTOR(1, ITF_NUM_TOTAL, 0, CONFIG_TOTAL_LEN, 0x00, 100),
#if CFG_TUD_CDC
  // Interface number, string index, EP notification address and size, EP data address (out, in) and size.
  TUD_CDC_DESCRIPTOR1(ITF_NUM_CDC, 0, EPNUM_CDC_NOTIF, 8, EPNUM_CDC_OUT, EPNUM_CDC_IN, 16),
#endif
#if CFG_TUD_HID
  // Interface number, string index, EP Out & EP In address, EP size
  TUD_HID_INOUT_DESCRIPTOR1(ITF_NUM_HID, 0, HID_ITF_PROTOCOL_NONE, sizeof(desc_hid_report), EPNUM_HID,
                                                              0x80 | EPNUM_HID, CFG_TUD_HID_EP_BUFSIZE, HID_INTERVAL)
#endif
};
```

リスト5　I²C処理の実装tud_hid_set_report_cb関数

```
void tud_hid_set_report_cb(uint8_t itf, uint8_t
          report_id, hid_report_type_t report_type,
          uint8_t const* buffer, uint16_t bufsize)
{
省略
  memset((void *)m_response, 0,
         (size_t)MCP2221_RESPONSE_SIZE);
  switch (buffer[0]) {
  case MCP2221_RFD:
      handle_read_flash_data(buffer[1],
                             (uint8_t *)m_response);
      break;
  case MCP2221_I2C_WD:
  case MCP2221_I2C_WD_RS:
  case MCP2221_I2C_WD_NS:
  case MCP2221_I2C_RD:
  case MCP2221_I2C_RD_RS:
  case MCP2221_I2C_RD_GET_I2C_DATA:
      handle_i2c(buffer, (uint8_t *)m_response);
      break;
  default:
      handle_default(buffer, (uint8_t *)m_response);
      break;
  }
省略
  // echo back anything we received from host
  tud_hid_report(0, m_response, sizeof(m_response));
}
```

リスト6　I²Cの初期化処理pico_i2c_init関数

```
static void pico_i2c_init(void) {
    m_i2c_buf_write_index = 0;
    m_i2c_read_start = false;
    m_i2c_buf_stored_size = 0;
    m_i2c_buf_read_index = 0;
    //Initialize I2C port at 400 kHz
    i2c_init(i2c, 400 * 1000);
    // Initialize I2C pins
    gpio_set_function(I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL_PIN, GPIO_FUNC_I2C);
}
```

ス・データをtud_hid_report関数でUSBデータとしてホストに返信します．従ってbufferの先頭バイトをMCP2221コマンドとして処理します．

▶RP2040のI²C処理

RP2040のI²Cの初期化処理は，pico_i2c_init関数で行っています(**リスト6**)．pico-sdkのi2c_init関数で，I²Cのクロックを設定し，gpio_set_function関数でI²C機能を設定しています．

I²Cの読み書きの処理は，handle_i2c関数(**リスト7**)を定義し，pico-sdkのi2c_write_blocking関数とi2c_read_blocking関数を呼び出すことで実装しています．ブロッキング処理の関数を呼び出した場合には，I²Cデバイスを適切なピンに接続しなかったときにプログラムが無限ループでハングするので，タイムアウト機能付きの関数を呼び出すことも可能です．

● 実装上の工夫

MCP2221では，一昔前のRAMが少ないマイコン環境を想定したAPI処理が定義されているため，RP2040ような最近のマイコンでは想定できないI²C処理が必要になっていました．

1. pico-sdkで用意されている関数では，I²Cのスレーブ・アドレスだけ送信し，書き込みデータを0バイトに指定することができません．後述のMCP2221のWindows環境でのユーティリティを使って，I²Cデバイスのスキャンをする際，書き込みデータ・バイト数が0として指定される場合の対応を正確に実装できません．そこで，今回の実装では書き込みデータ数が0と指定された場合には，1バイトのダミー・データを書き込む処理を追加しました．
2. MCP2221のAPIの定義では，1つのコマンドで，I²C書き込みにおいて60バイトより多いデータ数を指定できません．そこで，複数のコマンドでI²C書き込みが行われる場合には，各コマンドで書き込みデータをバッファリングし，最後の書き込みコマンドでI²C処理を行うように実装しています．

● CDC処理の実装

USB CDC処理は，TinyUSBのcdc_task(**リスト8**)中に実装しました．CDCの読み込みバッファにデータが存在する場合には，データを読み出

リスト7　I²Cの読み書きの処理handle_i2c関数

```
static void handle_i2c(uint8_t const *buf,
                       uint8_t *res) {
省略
    switch (buf[0]) {
    case MCP2221_I2C_WD:
        // スタート・コンディションを送る場合
        if (size > MCP2221_DATA_MAX) {
            // APIの最大データ長より大きいなデータの
            // 場合にはバッファに書き込み，
            // ステータスをビジーにする
            len = MCP2221_DATA_MAX;
            memcpy((void *)&m_i2c_buf[
                  m_i2c_buf_write_index], (const void *)
                                  &buf[4], (size_t)len);
            res[1] = MCP2221_CMD_BUSY;
            m_i2c_buf_write_index += len;
        } else {
            // APIの最大データ長以下のデータの場合には，
            // バッファのデータに追加しI2C書き込みを行う
            memcpy((void *)&m_i2c_buf[
                  m_i2c_buf_write_index], (const void *)
                                  &buf[4], (size_t)len);
            m_i2c_buf_write_index += len;
            i2c->restart_on_next = false;
              // リピート・スタート・コンディションを
              // 無効=スタート・コンディションを有効
            if (len == 0) {
                // pico-sdkではi2cの書き込みデータ長を
                // 0に指定できないので、ダミー・データと
                // して1バイトのゼロ値を指定する
                uint8_t dummy = 0;
#if MCP2221_USE_TIMEOUT
                // タイムアウト値を指定する場合
                flag = i2c_write_timeout_us(i2c, addr,
                     (const uint8_t *)&dummy, (size_t)1,
                                false, MCP2221_TIMEOUT);
#else
                // タイムアウト値を指定しない場合
                flag = i2c_write_blocking(i2c, addr,
                     (const uint8_t *)&dummy, (size_t)1,
                                                false);
#endif
            } else {
                // 書き込みデータ長が0でないときは，
                // タイムアウト値を指定しないで
                // I2C書き込みを行う
                // ストップ・コンディションを送るので
                // 最後のパラメータをfalseとする
                flag = i2c_write_blocking(i2c, addr,
                          (const uint8_t *)&m_i2c_buf[0],
                     (size_t)m_i2c_buf_write_index, false);
            }
省略
        }
        break;
    case MCP2221_I2C_WD_RS:
        // リピート・スタート・コンディションを送る
        // 場合，上記のスタート・コンディションと
        // ほぼ同様の処理
省略
            memcpy((void *)&m_i2c_buf[
                  m_i2c_buf_write_index], (const void *)
                                  &buf[4], (size_t)len);
            m_i2c_buf_write_index += len;
            i2c->restart_on_next = true;
              // リピート・スタート・コンディション有効
            // ストップコンディションを送るので，
            // 最後のパラメータをfalseとする
            flag - i2c_write_blocking(i2c, addr,
                (const uint8_t *)&m_i2c_buf[0], (size_t)
                          m_i2c_buf_write_index, false);
省略
    case MCP2221_I2C_WD_NS:
        // ストップ・コンディションを送らない場合
省略
            memcpy((void *)&m_i2c_buf[
                  m_i2c_buf_write_index], (const void *)
                                  &buf[4], (size_t)len);
            m_i2c_buf_write_index += len;
            i2c->restart_on_next = false;
              // リピート・スタート・コンディションを
              // 無効=スタート・コンディションを有効
            // ストップ・コンディションを送らないので，
            // 最後のパラメータをtrueとする
            flag = i2c_write_blocking(i2c, addr,
                (const uint8_t *)&m_i2c_buf[0], (size_t)
                          m_i2c_buf_write_index, true);
省略
    }
}
```

し，Wi-FiモジュールESP8266が接続されているUARTポート（TX：GP8とRX：GP9）に書き込みます．UARTポートの読み込みバッファにデータが存在する場合には，そのデータを読み出し，CDCの書き込みバッファの空きエリアが読み出したデータ数になるまで待ち，その後CDCの書き込みバッファにデータを書き込み，バッファをフラッシュするという手順です．

リスト8　CDCの実装

```
省略
#define BAUD_RATE 115200
#define UART_TX_PIN 8
#define UART_RX_PIN 9
:
void cdc_task(void)
{
  if ( tud_cdc_connected() )
  {
    if ( tud_cdc_available() )
    {
      char rbuf[64];
      uint32_t rcount = tud_cdc_read(rbuf,
                                     sizeof(rbuf));
      uart_write_blocking(uart,
                  (const uint8_t *)&rbuf, rcount);
    }
    if (uart_is_readable(uart))
    {
      char wbuf[64];
      uint32_t wcount = 0;
      while (uart_is_readable(uart)) {
          uart_read_blocking(uart,
              (uint8_t *)&wbuf[wcount], (size_t)1);
          wcount ++;
          if (wcount == 64) {
              break;
          }
      }
      while (wcount > tud_cdc_write_available()) {
      }
      tud_cdc_write((const char *)&wbuf, wcount);
      tud_cdc_write_flush();
    }
  }
}
```

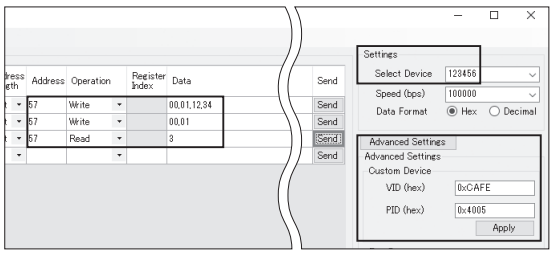


図10　MCP2221 I2C SMBus Terminalアプリケーション

ホストPCからの操作

● Windowsからの操作

MCP2221をWindowsホストに接続し制御する場合には，WindowsのHIDデバイス向けの標準APIを利用します．HIDデバイスの場合，INFファイルを用意することなく，APIを利用できます．

Windowsから制御の動作確認は，マイクロチップ・テクノロジーから提供されているMCP2221 I²C SMBus Terminalアプリケーションで行いました．I²Cスキャン，リアルタイム・クロック（DS3201）の操作およびEEPROM（24C64）の操作が正常に動作することを確認しました．

起動後，「Advanced Settings」の「Custom Device」でTinyUSBライブラリを利用したときのデフォルト値（VID：0xCAFE, PID：0x4005）を指定します．すると，「Settings」の「Select Device」のフィールドにUSBデバイスのシリアル番号が検出されます．**図10**の例では，EEPROMのI²Cスレーブ・アドレス（57）を指定して，

- 0x0001番地からデータ“12”,“34”を書き込み
- 読み出しアドレスとして0x0001番地を書き込み
- 読み出す数として3を指定

することでEEPROM中のデータを読み出しました．

● Linuxからの操作

▶ Ubuntu 22.04でUSB-I²Cブリッジを利用する

ホストPCのOSはUbuntu 22.40デスクトップです．Ubuntu 22.04のLinuxカーネルでは，カーネル・モジュール（hid-mcp2221.ko）が有効になっているようなので，カーネル自体パラメータを変更するビルドの必要はありませんでした．ただし，USBのVID/PIDが異なるので，VID/PIDを変更してカーネル・モジュールをビルドする必要があります．

マイクロチップ・テクノロジー社のウェブ・ページからMCP2221のカーネル・モジュールのドライバのソースファイルをダウンロードします．

```
wget http://ww1.microchip.com/
                    downloads/en/DeviceDoc/
                        mcp2221_0_1.tar.gz
tar zxvf mcp2221_0_1.tar.gz
cd mcp2221_0_1
```

i2c-mcp2221.cとdriver_load.shファイル中の，VIDとPIDをTinyUSBで利用されている値（VID：0xcafe, PID：0x4005）に変更します．

```
vi i2c-mcp2221.c
vi driver_load.sh
```

カーネル・モジュールをビルドし，インストールします．

```
sudo make modules
sudo make install
sudo ./driver_load.sh
```

Linuxのi2c-toolsをインストールします．

```
sudo apt install i2c-tools
```

PicoをホストPCにUSBで接続し，i2cdetectツールでPicoに接続されたI²Cデバイスをスキャンしてみます．I²Cデバイスとして，リアルタイム・クロックDS1307（およびEEPROM 24C64）を搭載したモジュールを使用しました．i2cdetect -lでチャネル8にUSB-I²Cブリッジが検出され，i2cdetect -y 8でアドレス0x57にEEPROM 24C64，0x68にDS1307が検出されました（**リスト9**）．

リスト9　i2cdetectツールでPicoに接続されたI²Cデバイスをスキャンした

```
sudo i2cdetect -l
省略
i2c-8   i2c  i2c-mcp2221 at bus 001 device 020
                                          I2C
adapter
省略
sudo i2cdetect -y 8
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

▶ Ubuntu 22.04でUSB-UARTブリッジを利用する

USB-UARTブリッジの機能は，カーネル・モジュールを作成することなく利用できました．minicomを利用して，USB-UARTブリッジの先に接続しているESP8266にATコマンドを実行してみます．minicomをインストールして，現在のユーザをdialoutグループに追加します．

```
sudo apt install minicom
sudo usermod -a -G dialout $USER
```

USB-UART機能は，ttyACM[x]（xは0，1，…，数字）デバイスとして認識されます．dmesgコマン

リスト10　minicomの画面

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyS3, 21:43:11

Press CTRL-A Z for help on special keys

AT+GMR
AT version:1.7.4.0(May 11 2020 19:13:04)
SDK version:3.0.4(9532ceb)
compile time:May 27 2020 10:12:22
Bin version(Wroom 02):1.7.4
OK
```

ドで認識されたデバイス名を確認します．

```
dmesg | grep ttyACM
[  111.997040] cdc_acm 1-1:1.0:
            ttyACM0: USB ACM device
```

minicomを起動します．

```
minicom /dev/ttyACM0 -b 115200
```

ESP8266のATモードでは，行末はCR/LFコードにする必要がありますので，［Enter］キーを押した後に，［Ctrl-J］を押します．AT+GMRコマンドでESP8266のファームウェアのバージョンを確認してみます（**リスト10**）．

● Windows 10 WSL2のUbuntu 20.04で利用する

Windowsのupbipd-winというコンポーネントを使用すると，Windowsホストに接続したUSBデバイスをWSL2（Windows Subsystem for Linux）に共有できます．usbipd-winは，Windowsに接続されたUSBデバイスからのUSBパケットをIPプロトコルを介してトンネリングしてHyper-VのゲストOSやWSL2に共有します（**図11**）．この機能を使って，Windowsホストに接続したUSB-I²CブリッジをWSL2のUbuntu 20.04から利用してみます．

2022年5月時点でWindows 10でサポートされているWSL2のUbuntu 20.04では，usbipd-winはサポートされていません．さらにUSB-I²Cブリッジを使用するのに必要なカーネル・モジュールにも対応していないので，usbipd-winをサポートするためにカーネルの再構築が必要です．

▶ WSL2 Ubuntu 20.04にWindowsに接続したUSB-I²Cブリッジを認識させる

マイクロソフトのUSBデバイスを接続するページ(4)を参考にして，Windows 10ホストに最新のusbipd-winツール（usbipd-win_2.3.0.msi）をインストールします．

WSL2のUbuntu 20.04のコンソールを起動して，linux-tools-5.4.0-77とhwdataパッケージをインストールします（**リスト11**）．

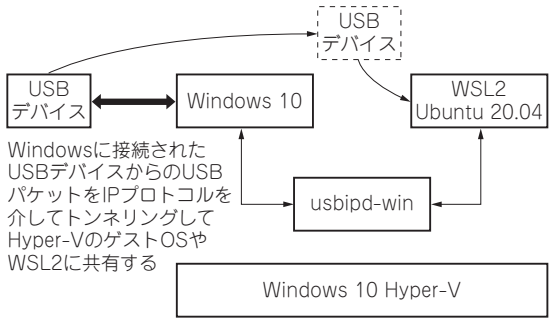


図11　usbpid-winの仕組み

テキスト・エディタを起動して，secure_pathに/usr/lib/linux-tools/5.4.0-77-genericを追加します（**リスト12**）．

管理者権限のPowershellのコンソールを起動して，Ubuntu-20.04をデフォルトに設定します．次にUSB-I²CブリッジをWindows 10ホストに接続してから，updipd bindコマンドを実行して，STATEが“attached”から“Shard（forced）”に変わり，デバイスが認識されることを確認します（**リスト13**）．

▶ Ubuntu 20.04のカーネル・ビルド

WSL2のUbuntu 20.04に，カーネルのビルドに必要なパッケージをインストールします（**リスト14**）．

WSL2のLinuxカーネルのリポジトリをクローンします．

```
$ git clone https://github.com/
    microsoft/WSL2-Linux-Kernel.git
$ cd WSL2-Linux-Kernel
```

menuconfigを起動して，HID_MCP2221（およびUSBシリアル）がカーネル・モジュールとして，有効になるように，カーネル・パラメータを変更します（**図12**）．

▶補足

menuconfigで「Microchip MCP2221 HID USB-to-I2C/SMBus host support」を表示するためには，USB_HID，I²C，GPIOLIBも有効化する必要があります（**リスト15**）．

リスト11　linux-tools-5.4.0-77とhwdataパッケージをインストール

```
sudo apt install linux-tools-5.4.0-77-generic hwdata
sudo update-alternatives --install /usr/local/bin/
  usbip usbip /usr/lib/linux-tools/5.4.0-77-generic/
                                          usbip 20
```

リスト12　secure_pathに追加

```
Defaults secure_path="/usr/lib/linux-tools/
              5.4.0-77-generic:/usr/local/sbin:..."
```

リスト13　usbipdコマンド実行結果

```
PS C:\WINDOWS\system32> wsl --set-default Ubuntu-20.04
PS C:\WINDOWS\system32> usbipd wsl list
BUSID  VID:PID    DEVICE                                                        STATE
3-11   cafe:4005  USB シリアル デバイス (COM6), USB 入力デバイス                Not attached
省略
PS C:\WINDOWS\system32> usbipd bind --force --busid 3-11
usbipd: info: Device with busid '3-11' was already shared.
PS C:\WINDOWS\system32> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
省略
3-11   cafe:4005  USB シリアル デバイス (COM6), USB 入力デバイス                Shared (forced)
省略
Persisted:
GUID                                  DEVICE
82203bed-8ff7-49d6-8d8d-6c35c21a6fd0  USB シリアル デバイス (COM20), TinyUSB i2c tiny usb
usbipd: warning: USB filter 'USBPcap' is known to be incompatible with this software; 'bind --force' will be
                                                                                                    required.
```

リスト14　パッケージをインストール

```
$ sudo apt install build-essential flex bison
   libssl-dev libelf-dev dwarves libncurses-dev git
```

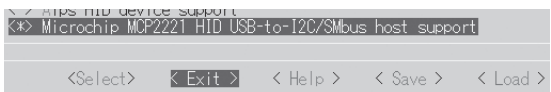


図12　menuconfigの設定画面

リスト15　HID_MCP2221有効化のための依存パラメータ情報

```
config HID_MCP2221
  tristate "Microchip MCP2221 HID USB-to-I2C/SMbus
                                        host support"
  depends on USB_HID && I2C
  depends on GPIOLIB
  help
  Provides I2C and SMBUS host adapter functionality
  over USB-HID through MCP2221 device.

  To compile this driver as a module, choose M here:
  the module will be called hid-mcp2221.ko.
```

WSL2のカーネルをビルドします.

```
make -j4 KCONFIG_CONFIG=Microsoft/
                              config-ws
sudo make modules
```

ビルドしたカーネルをWindowsのデフォルトのユーザ・フォルダにコピーします.

```
cp  arch/x86/boot/bzImage  /mnt/c/
          Users/ksgadget/wsl_kernel
```

.wslconfigファイル(**リスト16**)を作成し，wslカーネル・ファイルのパスを指定します.

```
nano /mnt/c/Users/[ユーザ名]/
                            .wslconfig
wsl -shutdown
```

▶ Ubuntu 20.04のUSB-I²Cカーネルモジュールの有効化

前節と同じようにMCP2221のカーネル・モジュールのソース・ファイルをダウンロードし，VID/PIDを変更し，カーネル・モジュールのビルドをします(**リスト17**)．インストールの際にエラーが発生した場合には，エラー・メッセージを参照し，手動で/lib/modules/5.10.102.1-microsoft-standard-WSL2+/kernel/drivers/i2c/busses，あるいはその他のフォルダにmcp2221.koファイルコピーをしてください.

リスト16　.wsl_configファイル

```
[wsl2]
kernel = C:\\Users\\[ユーザ名]\\wsl_kernel
```

リスト17　MCP2221カーネル・モジュールのビルド手順

```
$ wget http://ww1.microchip.com/downloads/en/
                          DeviceDoc/mcp2221_0_1.tar.gz
$ tar zxvf mcp2221_0_1.tar.gz
$ cd mcp2221_0_1
$ sudo make modules
$ sudo make install
```

前節と同じように，sudo ./driver_load.shでカーネル・モジュールを有効化すると，i2c-toolsのコマンドが使用できるようになります.

▶ Ubuntu 20.04のUSB-UARTカーネル・モジュールの有効化

本稿では解説しませんが，USB-UARTブリッジ機能も同じように有効化できます.

Picoで作ったUSB-I²Cデバイス向けにAdafruit Blinkaを利用する

● PCのPythonからデバイスのCircuitPythonを呼び出せる

Adafruit Blinkaは，デバイスで動作するCircuitPythonのライブラリを，ホストPCのPythonから利用するためのAPIを提供するPythonのモジュールです(**図13**)．MCP2221はCircuitPythonが動作するデ

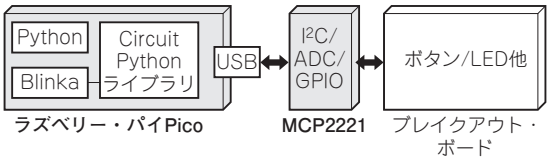


図13　Blinkaの仕組み

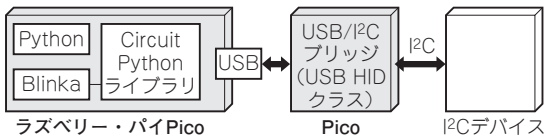


図14　Blinkaを利用したUSB-I²Cブリッジ

リスト18　Blinka用Python3ライブラリのインストール

```
sudo apt-get install python-dev libusb-1.0-0-dev
                                                libudev-dev
sudo apt install python3-pip
sudo pip3 install hidapi
```

リスト19　99-mcp2221.rules

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="cafe",
                ATTR{idProduct}=="4005", MODE="0666"
```

バイスではないのですが，CircuitPythonをサポートするデバイスと同じようにGPIO，I²C，SPIなどの制御がAdafruit Blinkaでサポートされています．Adafruit Blinkaを利用することで，CircuitPython向けに作成されたさまざまなプログラムをホストPCに接続されたデバイスで利用できます．ここでは，TinyUSBで作成したPico USB-I²Cデバイス向けにAdafruit Blinkaを利用する手順を解説します．

● LinuxでBlinkaを利用する

Ubuntu 20.04が動作するホストPCにBlinkaのライブラリをインストールし，USB-I²Cブリッジとして動作するPicoに接続したデバイスを制御してみます（**図14**）．Python3のライブラリをインストールします（**リスト18**）．

Ubuntu 20.04にデフォルトでインストールされているhid_mcp2221.koカーネル・モジュールが有効になっている場合には，無効化しておきます．

```
sudo rmmod hid_mcp2221
```

/etc/udev/rules.d/99-mcp2221.rules（**リスト19**）ファイルを作成し，USB-I²CブリッジのVID/PIDを追加します．

udevadmを実行して，追加ルールを反映します．その後，Adafruit Blinkaライブラリをインストールします．

```
sudo udevadm trigger
pip3 install adafruit-blinka
```

リスト20　Pythonライブラリの変更点

```
省略
class MCP2221:
省略
    # VID = 0x04D8
    # PID = 0x00DD
    VID = 0xcafe
    PID = 0x4005
省略
```

（a）mcp2221.py

```
省略
    for dev in hid.enumerate():
        # if dev["vendor_id"] == 0x04D8 and dev
        # ["product_id"] == 0x00DD:
        if dev["vendor_id"] == 0xcafe and dev
        ["product_id"] == 0x4005:
            self._chip_id = chips.MCP2221
            return self._chip_id
省略
```

（b）chip.py

リスト21　Pythonインタープリタから Blinkaライブラリの呼び出し

```
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license"
                                for more information.
>>> import hid
>>> import board
>>> import busio
>>> i2c = busio.I2C(board.SCL, board.SDA)
>>> i2c.try_lock()
True
>>> i2c.scan()
[57, 68]
```

インストールされたPythonライブラリのソースファイルを検索して，MCP2221のVID/PIDをUSB-I²CブリッジのVID/PIDに変更します．ライブラリはユーザのホーム・ディレクトリ下の以下の2ファイルを変更しました（**リスト20**）．

```
* ~/.local/lib/python3.8/
    site-packages/adafruit_blinka/
      microcontroller/mcp2221.py
* ~/.local/lib/python3.8/
          site-packages/adafruit_
           platformdetect/chip.py
```

環境変数BLINKA_MCP2221を設定します．

```
export BLINKA_MCP2221=1
```

Pythonインタープリタを起動し，Blinkaライブラリを呼び出し，USB-I²Cブリッジに接続されたI²Cデバイスをスキャンしてみます（**リスト21**）．

CircuitPythonでは，さまざまなI²Cデバイスがサポートされていますので，ホストPCからPythonプログラムを作成し，I²Cデバイスを容易に利用できるようになります．

せきもと・けんたろう