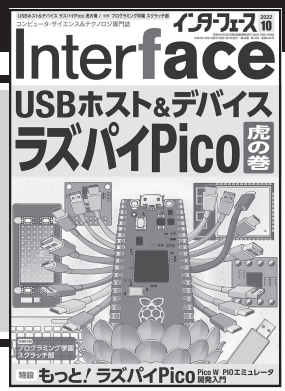

ラズパイPicoのUSB活用

第1回　USBベンダ・クラスを使ったUSB-I²Cブリッジ

関本 健太郎

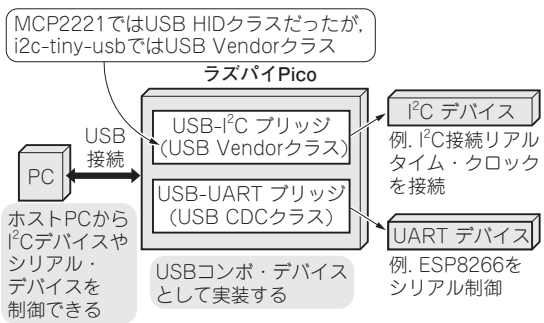

図1　i2c-tiny-usbベースのUSB-I²C/UARTブリッジ概要

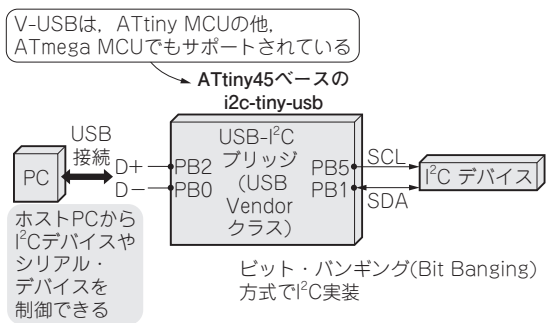

図2　ATtiny45ベースのi2c-tiny-usb

2022年10月号特集は「USBホスト&デバイス ラズパイPico虎の巻」でした．特集では，TinyUSBのサンプル・プログラムを詳しく解説したり，USBホスト，USBデバイスの製作事例を紹介したりしました．その中で，USB-I²Cブリッジについて，次のように解説しました．I²Cブリッジ製品には，

1. USB HIDクラスを利用したもの（10月号特集第3部第1章）
2. USBベンダ・クラスを利用したもの（今回）
3. USB CDCクラスを利用したもの（次回）

があります．10月号では，USB-I²Cブリッジの実現にUSB HIDクラスを用いましたが，USBベンダ・クラスを利用する方法もあります．USBベンダ・クラスを利用したものに，i2c-tiny-usbというプロジェクト（https://github.com/harbaum/i2c-tiny-usb）があります．

i2c-tiny-usbプロジェクトは，オープンソース/オープンハードウェア・プロジェクトです．一昔前にホストPCのプリンタ・ポートのピンを利用してディジタル入出力を実装していたユースケースの代替を目的に，USBポートを利用してI²Cデバイスを制御することを目的としています．

i2c-tiny-usbはLinuxの標準カーネル・モジュールに含まれています．本記事では，TinyUSBのベンダ・クラスの実装例を参考にi2c-tiny-usbをPicoで実装していきます（**図1**）．

ハードウェア

i2c-tiny-usbインターフェースのハードウェアは，安価で入手しやすいATtiny45（マイクロチップ・テクノロジー，20MHz，フラッシュ：4Kバイト，RAM：256バイト）で構成されています．i2c-tiny-usbインターフェースのUSBインターフェースは，ソフトウェアで実装されており，AVRの2つのピン（PB0とPB2）を使用します（**図2**，**図3**）．

このソフトウェア実装は，ロースピードUSBのみで動作します．I²Cインターフェースは，v-usbと呼ばれており，ビット・バンギング・アプローチ（マイコンの汎用I/Oポートでソフトウェアのみで実装することをこう呼ぶ）を使って実装されています．

ATtiny45のハードウェアでサポートされているI²C（twi）インターフェースは，USB動作に必要なチップのハードウェア・ピンにバインドされているため，I²Cには使っていません．代替用ビット・バンギングI²Cインターフェースは，完全にI²C互換ではない可能性があるため，全てのI²Cデバイスがこのバスで正しく機能することは保証していません．

i2c-tiny-usbは，ソフトウェアで調整可能なI²Cクロック遅延を提供し，I²Cクロックを構成できるようにします．デフォルトの遅延は10μsです．I²Cビット・バンギング・コードの追加の遅延により，これは

IF11_184-187_RP-USB(05)-1.indd 184　2022/09/12 22:28

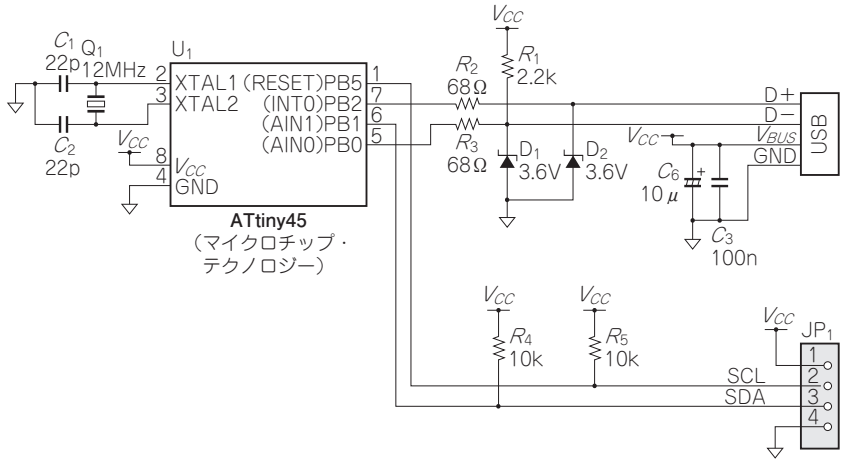

図3
ATtiny45を利用した
i2c-tiny-usbの回路図

約50kHzで動作します.

プログラム・フォルダの作成

2022年10月号特集第3部第1章のMCP2221向けのプログラムcdc_hidをベースにi2c-tiny-usbのブリッジのプログラムを作成します.

2022年10月号特集第3部第1章と同じようにprojectsフォルダ下にi2c-tiny-usbフォルダを新規作成し，cdc_hidフォルダ下のファイルを全てコピーし，そのフォルダ中のファイルを変更，削除して，i2c-tiny-usbブリッジ機能を実装します.

USBディスクリプタ

オリジナルのソースファイルを見ると，i2c-tiny-usbではデフォルトのエンドポイント0だけを利用するベンダ独自クラスのインターフェース・ディスクリプタを構成していました．TinyUSBのベンダ独自クラスのインターフェース・ディスクリプタ・マクロTUD_VENDOR_DESCRIPTORからエンドポイントの定義のパラメータを削除し，エンドポイントを0に固定したマクロTUD_VENDOR1_DESCRIPTORを作成し，desc_configuration配列に，コンフィギュレータ・ディスクリプタとインターフェース・ディスクリプタを定義しました(**リスト1**)．最終的なディスクリプタは，エンドポイント0を使っているため，エンドポイント・ディスクリプタを指定する必要はなく，非常にシンプルなものになりました(**図4**)．

リスト1　i2c-tiny-usbのコンフィグレーション・ディスクリプタの定義

```
#define TUD_VENDOR1_DESC_LEN  (9)
// インターフェース番号，ストリング・ディスクリプタ番号
// エンドポイント・アウト・イン・アドレス，エンドポイント・サイズ
#define TUD_VENDOR1_DESCRIPTOR(_itfnum, _stridx) \
  /* Interface */\
  9, TUSB_DESC_INTERFACE, _itfnum, 0, 0, \
  TUSB_CLASS_VENDOR_SPECIFIC, 0x00, 0x00, _stridx
#define CONFIG_TOTAL_LEN    (TUD_CONFIG_DESC_LEN \
                           + TUD_VENDOR1_DESC_LEN)

uint8_t const desc_configuration[] =
{
  // コンフィギュレーション番号，インターフェースの数
  // ストリング・ディスクリプタ番号，合計サイズなど
  TUD_CONFIG_DESCRIPTOR(1, ITF_NUM_TOTAL, 0,
                        CONFIG_TOTAL_LEN, 0x00, 100),
  // インターフェース番号，ストリング・ディスクリプタ番号
  // エンドポイント・アウト・イン・アドレス，エンドポイント・サイズ
  TUD_VENDOR1_DESCRIPTOR(ITF_NUM_VENDOR, 4)
};
```

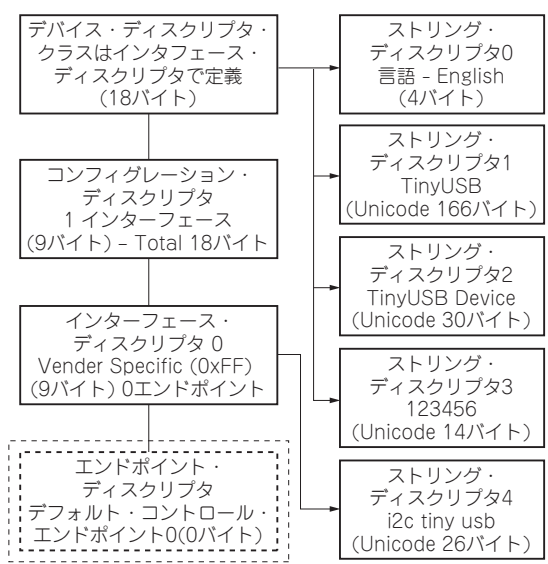

図4 i2c-tiny-usbのコンフィグレーション・ディスクリプタ

IF11_184-187_RP-USB(05)-1.indd 185　2022/09/12 22:28

リスト2　ベンダ独自クラスのUSBリクエストを切り分ける`tud_vendor_control_xfer_cb`コールバック関数

```
bool tud_vendor_control_xfer_cb(uint8_t rhport,
uint8_t stage, tusb_control_request_t const * request)
{
  bool ret = false;
  // DATAステージ，ACKステージのときは何もしない
  TU_LOG1("vendor_cb: st=%u bt=%x br=%x v=%x i=%x
                                        l=%x\r\n", stage,
          request->bmRequestType,
          request->bRequest,
          request->wValue,
          request->wIndex,
          request->wLength);
  // コントロール転送のセットアップ・ステージの
  // リクエストだけ処理を行う
  if (stage == CONTROL_STAGE_SETUP) {
    switch (request->bmRequestType_bit.type)
    {
      // リクエストはすべてusbFunctionSetup関数で
      // 処理する
      case TUSB_REQ_TYPE_VENDOR:
        TU_LOG1(" TyVendor:t=%x r=%x\r\n", request->
            bmRequestType_bit.type, request->bRequest);
        usbFunctionSetup((uint8_t *)request);
        ret = tud_control_status(rhport, request);
        break;
      case TUSB_REQ_TYPE_CLASS:
        TU_LOG1(" TyClass:t=%x r=%x\r\n", request->
            bmRequestType_bit.type, request->bRequest);
        usbFunctionSetup((uint8_t *)request);
        ret  = tud_control_xfer(rhport, request, (
                                    void*)usbMsgPtr, 4);
        break;
      default:
        TU_LOG1(" TyOther:t=%x r=%x\r\n", request->
            bmRequestType_bit.type, request->bRequest);
        break;
    }
  } else {
    ret = true;
  }
  if (!ret) {
    TU_LOG1("vendor_cb: ret=%s\r\n",
                              (ret == true)? "OK":"NG");
  }
  return ret;
}
```

ベンダ独自クラスのリクエストの処理の実装

ベンダ独自クラスのUSBリクエスト処理の部分をTinyUSBライブラリ向けに移植します．ベンダ独自クラスのUSBリクエストは，`tud_vendor_control_xfer_cb`コールバック関数で切り分け，`usbFunctionSetup`関数を呼び出します（**リスト2**）．

次に`usbFunctionSetup`関数の処理の概要です．i2c-tiny-usbのUSBベンダ・クラスでは，i2c-tiny-usb独自のコマンド，CMD_ECHO，CMD_GET_FUNC，CMD_SET_DELAY，CMD_I2C_IO，CMD_I2C_BEGIN，CMD_I2C_END，CMD_GET_STATUSが定義されています．これらのコマンドがUSBセットアップ・パケットの構造体の`bRequest`パラメータに指定されて送られてきます．コマンドごとに`switch`文で移植した関数に振り分けます．I²Cに関する処理は全て，`i2c_do`という関数で処理されます．`i2c_do`をPico用に置き換えていきます（**リスト3**）．

ATtiny45用のI²Cビット・バンギング処理を，Picoのgpio APIを利用して，Pico向けに置き換えます（**リスト4**）．

Linuxからの操作

● Linuxカーネル・モジュールの変更とビルド

i2c-tiny-usbのカーネル・モジュールは，Linuxカーネルのソースの`drivers/i2c/busses`フォルダ下に`i2c-tiny-usb.c`として含まれています．まず，Pico用のi2c-tiny-usbのVID/PIDをソース中に追加します（**リスト5**）．

カーネルの必要なツールをインストール後，使っているカーネルのソースコードを入手し，適当なフォルダに展開します．筆者はLinuxカーネル 5.4.195を使っています．

リスト3　コマンド処理を行うi2c-usb-tinyの`usbFunctionSetup`関数

```
uint8_t usbFunctionSetup(uint8_t *data) {
  static uint8_t replyBuf[4];
  usbMsgPtr = replyBuf;
省略
  switch(data[1]) {
  case CMD_ECHO:
    // エコー（転送の信頼性のテストのため）
省略
    break;
  case CMD_GET_FUNC:
    memcpy((void *)replyBuf, (const void *)&func,
                                        sizeof(func));
    return sizeof(func);
    break;
  case CMD_SET_DELAY:
省略
    break;
  case CMD_I2C_IO:
  case CMD_I2C_IO + CMD_I2C_BEGIN:
  case CMD_I2C_IO                    + CMD_I2C_END:
  case CMD_I2C_IO + CMD_I2C_BEGIN + CMD_I2C_END:
    return i2c_do((struct i2c_cmd*)data);
    break;
  case CMD_GET_STATUS:
省略
    break;
  default:
    break;
  }
  return 0;
}
```

サポートするI²C関連コマンド情報を返す

I²C関連操作は全てi2c_doで処理する

IF11_184-187_RP-USB(05)-1.indd 186　2022/09/12 22:28

リスト4　Pico向けの変更点
灰色の部分を置き換える

```
static void i2c_io_set_sda(uint8_t hi) {
  if (hi) {
    gpio_set_dir(sda_pin, GPIO_IN);      // input
    gpio_pull_up(sda_pin);               // pullup
  } else {
    gpio_set_dir(sda_pin, GPIO_OUT);     // output
    gpio_put(sda_pin, false);            // low
  }
}

static uint8_t i2c_io_get_sda(void) {
  return (uint8_t)gpio_get(sda_pin);
}

static void i2c_io_set_scl(uint8_t hi) {
#ifdef ENABLE_SCL_EXPAND
  _delay_loop_2(clock_delay2);
  if (hi) {
    gpio_set_dir(scl_pin, GPIO_IN);      // input
    gpio_pull_up(scl_pin);               // pullup
    while (!(gpio_get(scl_pin)))
      ;
  } else {
    gpio_set_dir(scl_pin, GPIO_OUT);     // output
    gpio_put(scl_pin, false);            // low
  }
  _delay_loop_2(clock_delay);
#else
  _delay_loop_2(clock_delay2);
  if (hi) {
    gpio_put(scl_pin, true);             // high
  } else {
    gpio_put(scl_pin, false);            // low
  }
  _delay_loop_2(clock_delay);
#endif
}

static void pico_i2c_init(void) {
  gpio_init(sda_pin);
  gpio_set_dir(sda_pin, GPIO_IN);      // input
  gpio_pull_up(sda_pin);               // pullup
  gpio_init(scl_pin);
#ifdef ENABLE_SCL_EXPAND
  gpio_set_dir(scl_pin, GPIO_IN);      // input
  gpio_pull_up(scl_pin);               // pullup
#else
  gpio_set_dir(scl_pin, GPIO_OUT);     // output
#endif

  /* no bytes to be expected */
  expected = 0;
}
```

カーネル，全てのカーネル・モジュールをビルドするには数時間かかりますので，ここではi2c/bussesフォルダ下のカーネル・モジュールのみをビルドします（**リスト6**）.

● Linuxカーネル・モジュールの有効化

/etc/udev/rules.dフォルダに99-i2c-tiny-usb.rulesファイル（**リスト7**）を作成します.

rulesファイルを有効化します.

```
sudo udevadm control --reload-rules
             && sudo udevadm trigger
```

i2c-tiny-usbのファームウェアを実装したPicoのUSBをホストPCに接続し，カーネル・モジュールをロードします.

```
cd ~/linux-5.4.195/drivers/i2c/
                                busses
sudo rmmod i2c-tiny-usb.ko
sudo insmod i2c-tiny-usb.ko
modprobe i2c-tiny-usb
```

これで，i2cdetectコマンドで認識できるようになります.

```
sudo i2cdetect -l
省略
i2c-8  i2c
i2c-tiny-usb at bus 001 device 102
I2C adapter
省略
```

リスト5　Pico用のUSB VID/PIDの追加
灰色の部分を置き換える

```
static const struct usb_device_id i2c_tiny_usb_
table[]
= {
    { USB_DEVICE(0xcafe, 0x4011) },
                              /* Raspi Pico Tinyusb */
    { USB_DEVICE(0x0403, 0xc631) }, /* FTDI */
    { USB_DEVICE(0x1c40, 0x0534) },
                                    /* EZPrototypes */
    { }                      /* Terminating entry */
};
```

リスト6　i2c/bussesフォルダ下のカーネル・モジュールのみのビルド手順

```
cd ~/linux-5.4.195/drivers/i2c/busses
sudo make -C /usr/src/linux-headers-5.4.195
                                  M=$('pwd') modules
```

リスト7　/etc/udev/rules.d/99-i2c-tiny-usb.rules
全て1行で書く

```
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}==
    "cafe", ATTRS{idProduct}=="4011",ATTRS{product}==
  "Raspi Pico i2c-tiny-usb", ATTR{bInterfaceNumber}==
           "08", RUN+="/sbin/modprobe i2c_tiny_usb",
     RUN+="/bin/sh -c 'echo cafe 4011 >/sys/bus/usb/
                      drivers/i2c-tiny-usb/new_id'"
```

せきもと・けんたろう

IF11_184-187_RP-USB(05)-1.indd 187　2022/09/12 22:28