リストA usbd-serialクレートを使ったUSBシリアル・デバイス実装例

```
#[rp_pico::hal::entry]
fn main() -> ! {
    let pac = pac::Peripherals::take().unwrap();

    let mut resets = pac.RESETS;

    let mut watchdog = hal::Watchdog::new(pac.WATCHDOG);
    // クロックを初期化
    let clocks = hal::clocks::init_clocks_and_plls(
        rp_pico::XOSC_CRYSTAL_FREQ,
        pac.XOSC,
        pac.CLOCKS,
        pac.PLL_SYS,
        pac.PLL_USB,
        &mut resets,
        &mut watchdog,
    )
    .ok()
    .unwrap();
    // UsbBusを初期化
    let usb_bus = hal::usb::UsbBus::new(
        pac.USBCTRL_REGS,   // RP2040のUSBペリフェラルのレジスタ
        pac.USBCTRL_DPRAM,  // RP2040のUSBペリフェラルのDPRAM
        clocks.usb_clock,   // USBクロック
        true,               // Vbus検出ビットを強制的にセットする
        &mut resets,        // サブシステムのリセット・レジスタ
    );
    // UsbBusAllocatorを構築
    // ※UsbBusAllocatorは内部可変性を持つ型なのでmutでなくて良い
    let usb_bus_allocator = UsbBusAllocator::new(usb_bus);
    // usbd-serialクレートのSerialPortを構築
    let mut usb_serial = SerialPort::new(&usb_bus_allocator);
    // UsbDeviceを構築 VID=0x6666, PID=0x4444 (prototype product)
    let mut usb_device = UsbDeviceBuilder::new(&usb_bus_allocator, UsbVidPid(0x6666, 0x4444))
        .manufacturer("test manufacturer")  // Manufacturer  = "test manufacturer"
        .product("test product")            // Product       = "test product"
        .serial_number("serial number")     // Serial Number = "serial number"
        .composite_with_iads()              // IADを使った複合デバイスとする
        .max_packet_size_0(64)              // 最大パケットサイズ（64バイト）
        .build();                           // 上記の設定でUsbDeviceを構築

    // ループ・バック用のバッファ
    let mut buffer = [0u8; 64];
    // ループ・バック用バッファ内の送り返し待ちデータ
    // ※本来は型指定不要で後続の文から型が推論される
    let mut pending_bytes_to_write: Option<(usize, usize)> = None;
    loop {
        // USBシリアルのホストからの受信データを読み出して送り返す
        if pending_bytes_to_write.is_none() {   // 送り返し待ちデータなければ読む
            if let Ok(bytes_read) = usb_serial.read(&mut buffer) {
                pending_bytes_to_write = Some((0, bytes_read))  // 送り返し待ちデータ残量保存
            }
        }
        if let Some((bytes_written, bytes_to_write)) = pending_bytes_to_write { // 送り返し待ちデータあり？
            // バッファ内のデータを送信
            if let Ok(bytes_written_now) = usb_serial.write(&buffer[bytes_written..bytes_to_write])
            {
                // 送り返し待ちデータの送信完了位置を更新
                let bytes_written = bytes_written + bytes_written_now;
```

```
60                  if bytes_written == bytes_to_write {     // 全部送り終わったらNoneにしておく
61                      pending_bytes_to_write = None;
62                  } else {                                 // 残ってるので更新
63                      pending_bytes_to_write = Some((bytes_written, bytes_to_write));
64                  }
65              }
66          }
67          // USBデバイスのイベントなどを処理する
68          usb_device.poll(&mut [&mut usb_serial]);
69      }
70  }
```

図A　PCのUSBデバイス一覧に作成したUSBシリアル・デバイスが表示される

```
$ lsusb
...
Bus 002 Device 049: ID 6666:4444 Prototype product Vendor ID
...
$ lsusb -s 002:049 -v    # -sオプションにbus:deviceの形式で番号を指定

Bus 002 Device 049: ID 6666:4444 Prototype product Vendor ID
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.10
  bDeviceClass          239 Miscellaneous Device    // 0xef
  bDeviceSubClass         2
  bDeviceProtocol         1 Interface Association
  bMaxPacketSize0        64
  idVendor           0x6666 Prototype product Vendor ID
  idProduct          0x4444
  bcdDevice            0.10
  iManufacturer           1 test manufacturer   // 指定したmanufacturer
  iProduct                2 test product        // product
  iSerial                 3 serial number       // serial numberが確認できる
  bNumConfigurations      1
...
```

図B　dmesgでデバイス名を確認する

```
$ sudo dmesg # Ubuntu 22.04以降はsudoが必要
...
[229350.506602] cdc_acm 2-2.4:1.0: ttyACM0: USB ACM device
...
$ sudo apt install -y minicom # minicom入れてなければ入れる
$ minicom -D /dev/ttyACM0
minicom へようこそ 2.7.1

オプション: I18n
コンパイルされた日時は:  Dec 23 2019, 02:06:26.
ポート /dev/ttyACM0, 01:09:03
```

リストB　CMSIS-DAP向けのUsbClass実装型の定義

```
pub struct CmsisDapInterface<'a, B: UsbBus> {
    interface: InterfaceNumber,      // インターフェース番号
    serial_string: StringIndex,      // シリアル・ナンバ用ストリング・ディスクリプタ番号
    out_ep: EndpointOut<'a, B>,      // Bulk OUT用エンドポイント
    in_ep: EndpointIn<'a, B>,        // Bulk IN用エンドポイント
}
impl<B: UsbBus> CmsisDapInterface<'_, B> {
```

```
    /// CmsisDapInterfaceを初期化する
    /// alloc: UsbBusAllocatorへの参照
    /// max_packet_size: 最大パケット・サイズ
    pub fn new(alloc: &UsbBusAllocator<B>, max_packet_size: u16) -> CmsisDapInterface<'_, B> {
        CmsisDapInterface {
            interface: alloc.interface(),        // 新しいインターフェース番号を確保して保存
            serial_string: alloc.string(),       // シリアル番号のストリング・ディスクリプタ番号を確保して保存
            out_ep: alloc.bulk(max_packet_size),     // Bulk OUT用エンドポイントを確保
            in_ep: alloc.bulk(max_packet_size),      // Bulk IN用エンドポイントを確保
        }
    }
}
```

リストC CmsisDapInterfaceに対してUsbClassトレイトを実装する

```
use usb_device::device::DEFAULT_ALTERNATE_SETTING;
const USB_IF_CLASS_VENDOR: u8 = 0xff;
const USB_IF_SUBCLASS_VENDOR: u8 = 0x00;
const USB_IF_PROTOCOL_NONE: u8 = 0x00;
impl<B: UsbBus> UsbClass<B> for CmsisDapInterface<'_, B> {
    fn get_configuration_descriptors(&self, writer: &mut DescriptorWriter) -> Result<()> {
        writer.interface_alt(    // インターフェースディスクリプタを書き込み
            self.interface,      // インターフェース番号
            DEFAULT_ALTERNATE_SETTING,  // このコンフィグレーションのデフォルト・インターフェース
            USB_IF_CLASS_VENDOR,         // ベンダ固有クラス (0xff)
            USB_IF_SUBCLASS_VENDOR, // サブクラス (0x00)
            USB_IF_PROTOCOL_NONE,     // プロトコルなし (0x00)
            Some(self.serial_string),  // インターフェース文字列のインデックス
        )?;
        writer.endpoint(&self.out_ep)?; // Bulk OUT エンドポイントディスクリプタを書き込み
        writer.endpoint(&self.in_ep)?;  // Bulk IN エンドポイントディスクリプタを書き込み

        Ok(())
    }
    fn get_string(&self, index: StringIndex, lang_id: u16) -> Option<&str> {
        let _ = lang_id;
        if index == self.serial_string {     // インターフェース文字列に対する要求？
            Some("CMSIS-DAP interface")      // インターフェース文字列を返す
        } else {
            None
        }
    }
}
```

リストD get_bos_descriptorsの実装

```
const BOS_CAPABILITY_TYPE_PLATFORM: u8 = 0x05;
const MS_VENDOR_CODE: u8 = 0x01;
fn get_bos_descriptors(&self, writer: &mut BosWriter) -> Result<()> {
    writer.capability(
        BOS_CAPABILITY_TYPE_PLATFORM,
        &[
            0,  // Reserved
            0xdf, 0x60, 0xdd, 0xd8,  // MS_OS_20_Platform_Capability_ID
            0x89, 0x45, 0xc7, 0x4c,  // {D8DD60DF-4589-4CC7-9CD2-659D9E648A9F}
            0x9c, 0xd2, 0x65, 0x9d,  //
            0x9e, 0x64, 0x8a, 0x9f,  //
            0x00, 0x00, 0x03, 0x06,  // dwWindowsVersion - 0x06030000 (Win8.1 or later)
            174, 0,                  // wLength = MS OS 2.0 descriptor set
            MS_VENDOR_CODE,          // bMS_VendorCode
            0x00,                    // bAltEnumCmd - does not support alternate enum.
        ]
```

```
17      )?;
18      Ok(())
19  }
```

リストE control_in関数

```
fn control_in(&mut self, xfer: ControlIn<B>) {
    let request = xfer.request();
    if request.request_type == RequestType::Vendor
        && request.request == MS_VENDOR_CODE    // BOSで指定したリクエストID
        && request.value == 0                   // 0固定 (仕様で規定)
        && request.index == 7                   // 7固定 (仕様で規定)
    {
        let interface_number = self.interface;
        xfer.accept(|buffer| {
            // &mut [u8]型の変数bufferに対してMS OS 2.0 Descriptorの内容を書き込む処理
        }
    }
}
```

リストF DAP_Infoコマンドの実装

```
 1 pub struct CmsisDapInterface<'a, B: UsbBus> {
 2     // (省略)
 3     response_buffer: [u8; 64],
 4     pending_response_bytes: Option<usize>,
 5 }
 6
 7 impl<B: UsbBus> CmsisDapInterface<'_, B> {
 8     pub fn new(alloc: &UsbBusAllocator<B>, max_packet_size: u16) -> CmsisDapInterface<'_, B> {
 9         CmsisDapInterface {
10             // (省略)
11             response_buffer: [0u8; 64],          // レスポンス格納用バッファ
12             pending_response_bytes: None,        // 返信待ちレスポンスバイト数
13         }
14     }
15     pub fn poll(&mut self) -> Result<()> {
16         // 未送信レスポンスがあるか？
17         if let Some(pending_response_bytes) = self.pending_response_bytes.as_ref() {
18             self.in_ep.write(&self.response_buffer[..*pending_response_bytes])?;
19             // 送信成功したのでクリア
20             self.pending_response_bytes = None;
21         }
22         // コマンドを受信
23         let mut response_length = 0;
24         {
25             let mut response = &mut self.response_buffer[..];
26             // ホストからパケット受信
27             let mut request_buffer = [0u8; 64];
28             let request_length = self.out_ep.read(&mut request_buffer)?;
29             let mut request = &request_buffer[..request_length];
30             while request.len() > 0 {
31                 match request[0] {
32                     0x00 => {   // DAP_Infoコマンド
33                         if request.len() >= 2 {
34                             // ID
35                             let response_bytes = match request[1] {
36                                 0x01 => "vendor".as_bytes(),    // ベンダ名
37                                 0x02 => "product".as_bytes(),   // プロダクト名
38                                 0x03 => "serial".as_bytes(),    // シリアル番号
39                                 0x04 => "2.0.0".as_bytes(),     // CMSIS-DAPバージョン
40                                 0x09 => "1.0.0".as_bytes(),     // ファームウェアバージョン
```

```
                            0xf0 => &[0x01, 0x00],          // Capabilities = SWD
                            0xfe => &[0x01],                // 最大パケット数
                            0xff => &[64, 0],               // 最大パケットサイズ
                            _ => &[],                       // 未実装
                        };
                        // レスポンス・バッファに書き込み
                        response[response_length + 0] = 0;
                        response[response_length + 1] = response_bytes.len() as u8;
                        response[response_length + 2..response_length + 2 + response_bytes.len()]
                            .copy_from_slice(response_bytes);
                        let response_length_inc = 2 + response_bytes.len();
                        response_length += response_length_inc;
                        response = &mut response[response_length_inc..];
                        // リクエストの読み出し位置を更新
                        request = &request[2..];
                    }
                },
                _ => {
                    // 未実装コマンド．無視する
                    break;
                },
            }
        }
        if let Err(_) = self.in_ep.write(&self.response_buffer[..response_length]) {
            // 送信できなかったので送信待ち状態とする
            self.pending_response_bytes = Some(response_length);
        }
        Ok(())
    }
}
// （省略）
fn main() -> ! {
    // （省略）
    loop {
        // USBデバイスのイベントなどを処理する
        usb_device.poll(&mut [&mut cmsis_dap]);
        // CMSIS-DAPのコマンドを処理する
        cmsis_dap.poll().ok();
    }
}
```

リストG　SwdIoトレイトの定義

```
pub trait CmsisDapCommand { /* 省略 */ }
pub trait CmsisDapCommandInner { /* 省略 */ }
impl<Inner: CmsisDapCommandInner> CmsisDapCommand for Inner { /* 省略 */ }
impl<Io: SwdIo> CmsisDapCommandInner for Io { /* 省略 */ }

pub trait SwdIo {
    /// ターゲットに接続する
    fn connect(&mut self);
    /// ターゲットから切断する
    fn disconnect(&mut self);
    /// SWDのクロック周波数を設定する
    fn swj_clock(
        &mut self,
        config: &mut SwdIoConfig,
        frequency_hz: u32,
    ) -> core::result::Result<(), DapError>;
    /// SWJの書き込みシーケンスを実行する
    fn swj_sequence(&mut self, config: &SwdIoConfig, count: usize, data: &[u8]);
    /// SWDの読み出しシーケンスを実行する
```

```
    fn swd_read_sequence(&mut self, config: &SwdIoConfig, count: usize, data: &mut [u8]);
    /// SWDの書き込みシーケンスを実行する
    fn swd_write_sequence(&mut self, config: &SwdIoConfig, count: usize, data: &[u8]);
    /// SWDの転送処理を実行する
    fn swd_transfer(
        &mut self,
        config: &SwdIoConfig,
        request: SwdRequest,
        data: u32,
    ) -> core::result::Result<u32, DapError>;
    /// 出力を有効にする
    fn enable_output(&mut self);
    /// 出力を無効にする
    fn disable_output(&mut self);
}

pub trait PrimitiveSwdIo { /* 省略 */ }
impl<Io: PrimitiveSwdIo> SwdIo for Io { /* 省略 */ }
pub trait BitBangSwdIo { /* 省略 */ }
impl<Io: BitBangSwdIo> PrimitiveSwdIo for Io { /* 省略 */ }
```

リストH　BitBangSwdIoトレイトの定義

```
pub trait BitBangSwdIo {
    /// 半クロック・サイクル分待つためのCPUサイクル数を計算する
    fn calculate_half_clock_cycles(_frequency_hz: u32) -> Option<u32> {
        None
    }
    /// SWCLKを入力に設定する
    fn to_swclk_in(&mut self);
    /// SWCLKを出力に設定する
    fn to_swclk_out(&mut self, output: bool);
    /// SWDIOを入力に設定する
    fn to_swdio_in(&mut self);
    /// SWDIOを出力に設定する
    fn to_swdio_out(&mut self, output: bool);
    /// SWCLKの出力値を設定する
    fn set_swclk_output(&mut self, output: bool);
    /// SWDIOの出力値を設定する
    fn set_swdio_output(&mut self, output: bool);
    /// SWDIOの入力値を取得する
    fn get_swdio_input(&mut self) -> bool;
    /// 半クロック分待つ
    fn clock_wait(&self, config: &SwdIoConfig);
}
```

リストI　文字列表示部分

```
    rprintln!("LED Blink application start.");
    let mut led_pin = pins.led.into_push_pull_output();
    loop {
        led_pin.set_high().unwrap();
        delay.delay_ms(500);
        led_pin.set_low().unwrap();
        delay.delay_ms(500);
    }
```