PicoにおけるTinyUSBの働き

関本 健太郎

RP2040の電源投入からブート時の初期化処理，TinyUSBのUSBデバイスおよびUSBホストの初期化とその後の処理の流れを解説します．マイコンごとのUSBに関する，割り込みなどの低レベル処理と，TinyUSBでサポートされているUSBクラスの基本的な処理は，TinyUSBでカバーされており，利用者はユースケースに応じたUSBクラスのコールバック関数を実装するだけです．

起動シーケンス

USBホスト処理が始まる前のラズベリー・パイPico（以降，Pico）の起動シーケンスについて解説します（**図1**）．電源が投入されると0x00000000番地に配置されているRP2040内蔵ROM上のブートローダが実行されます．

●［BOOTSEL］ボタンを押したとき

［BOOTSEL］ボタンが押されている場合には，BOOTSELピンが“L”となり，外部SPIフラッシュ・メモリがマイコンから切り離された状態になり，引き続きブートローダ中のプログラムが実行されます．PicoはUSBマス・ストレージ・クラスのデバイスとして認識され，uf2形式のアプリケーション・プログラムを書き込めるモードになります．

このときのUSBデバイスとしてのUSBデバイス・ディスクリプタは，マス・ストレージ・クラスとベンダ独自クラスの2つのインターフェースを持っていま

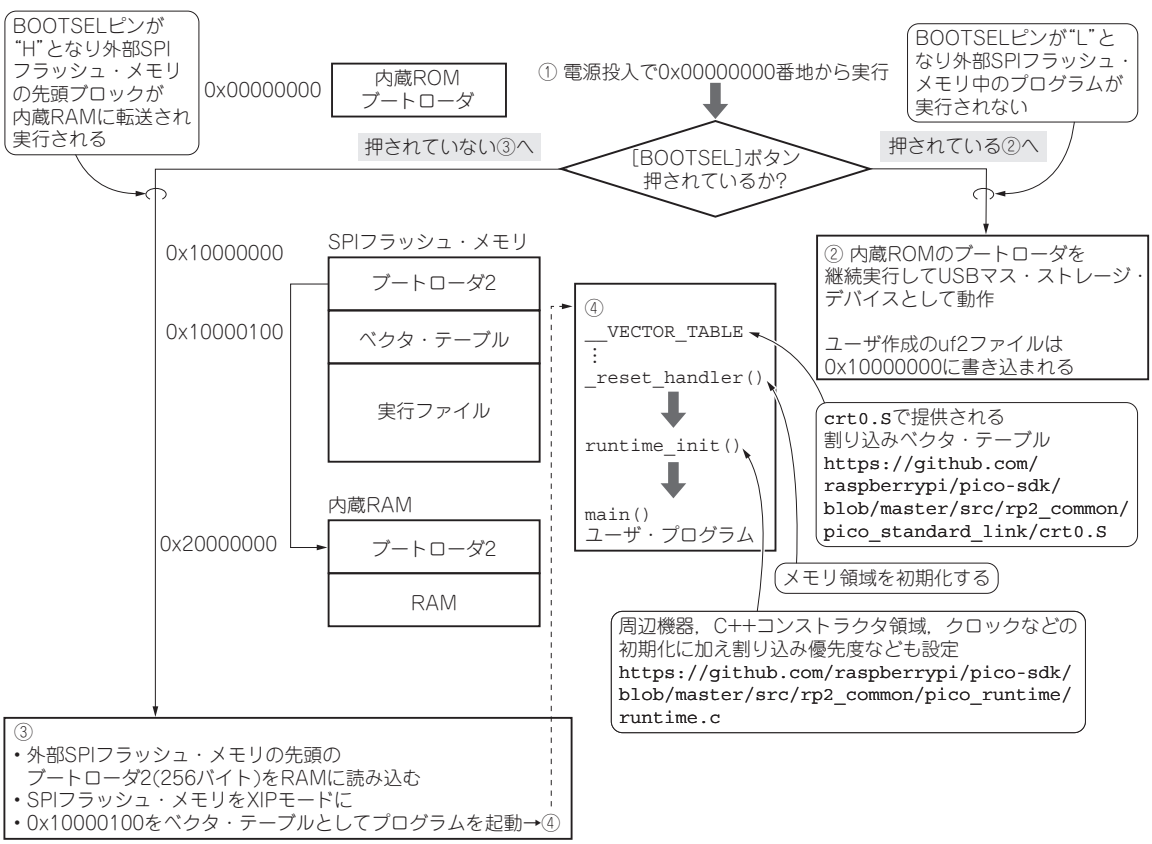

図1　Picoの起動シーケンス

注：TinyUSBは，工業製品で利用するレベルのソフトウェアの品質には及ばない点にご注意ください．

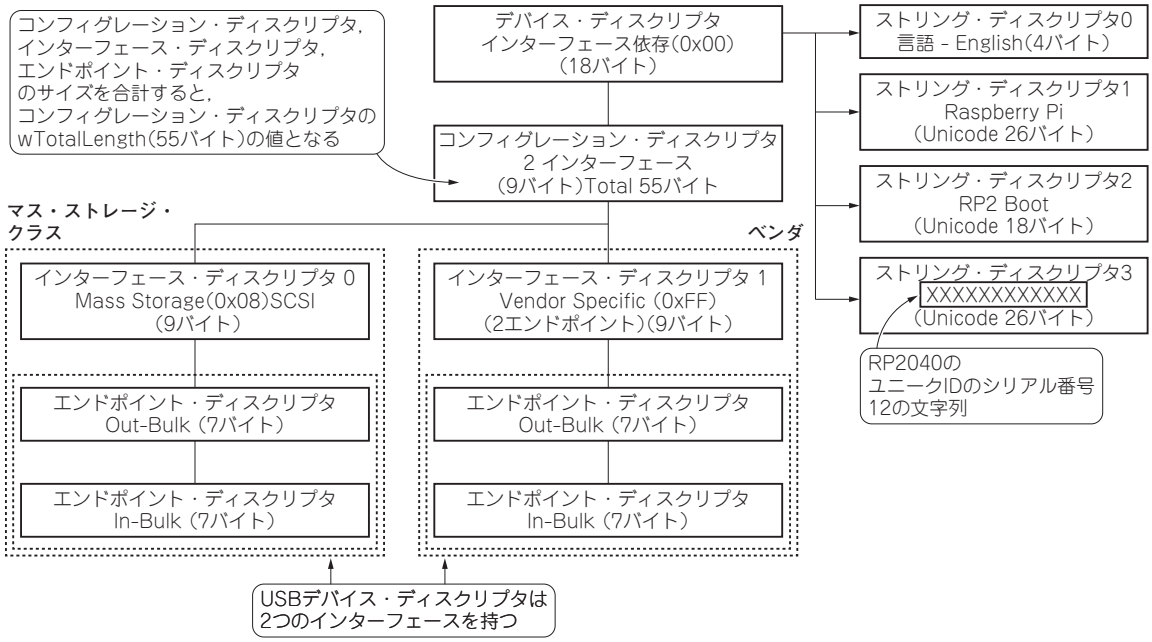

図2　［BOOTSEL］ボタン押下時のUSBディスクリプタ

す(**図2**)．シリアル番号はRP2040マイコンのユニークIDとなる12の文字列が設定されています．

●［BOOTSEL］ボタンを押さなかったとき

［BOOTSEL］ボタンが押されていない場合には，BOOTSELピンが“H”となり，0x10000000番地に配置されている外部SPIフラッシュ・メモリの先頭ブロック(ブートローダ2)が，マイコン内蔵RAM上0x20000000に転送され，実行されます．そして，外部SPIフラッシュ・メモリ中に保存されていたユーザ・アプリケーションが実行されます．

Picoの起動シーケンスを，**図1**に示します．_reset_handler(crt0.S)でメモリ領域が初期化され，runtime_init関数(runtime.c)，ユーザが作成したmain関数と呼び出されます．

runtime_init関数では，次の処理が順番に行われます．

- 周辺機能が初期化される(RESET_BASEレジスタ 0x4000c000)，
- C++のコンストラクタ領域が初期化される
- クロックが初期化される
- ミューテックスが初期化される
- spin_lockがリセットされる
- 割り込みの優先度が設定される
- C++のコンストラクタが呼び出される

その後，main関数へと続きます．

まとめると，PICO SDKで作成したプログラムは，0x10000000に配置されるようになっており，その先頭にはcrt0.Sで提供される割り込みベクタ・テーブルが置かれ，そのすぐ後にruntime.cで提供されるランタイム・ライブラリの初期化ルーチンのruntime_init()が呼ばれ，ユーザ・アプリケーションのmain()関数が呼び出されます．

USBデバイス処理

TinyUSBの処理を2つに分けて説明します．

1，USBデバイスに対する処理の流れ
2，USBホストに対する処理の流れ

処理の概要を説明することで，利用者はTinyUSBが用意している各種のコールバック関数を記載するだ

リスト1　TinyUSBデバイス処理のmain関数

```
int main(void)
{
  board_init();

  tud_init(BOARD_TUD_RHPORT); // TinyUSBのUSBデバイス
                                としての初期化関数

  while (1) // 無限ループ
  {
    tud_task(); // TinyUSBのデバイス・タスクの実行
:
    hid_task(); // HIDクラスのタスクの実行
  }

  return 0;
}
```

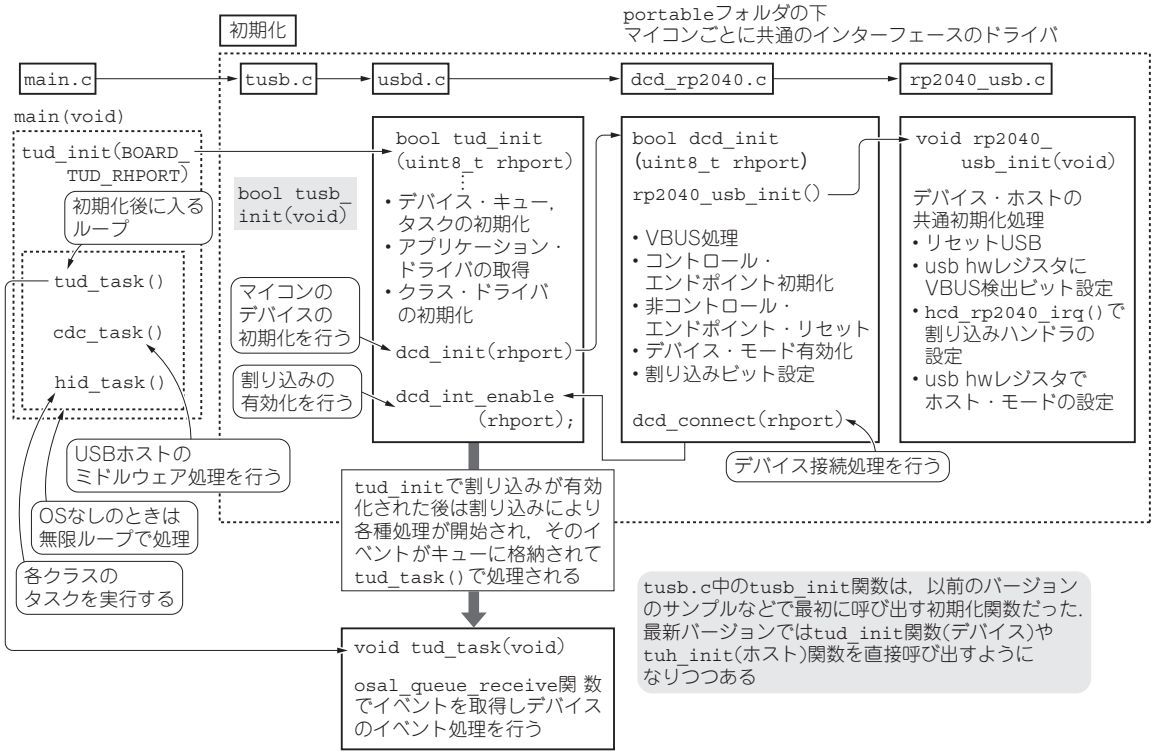

図3　TinyUSBデバイス処理初期化の流れ

けで，USBデバイスおよびUSBホストのクラス処理を容易に拡張できることを理解できると思います．

● メイン・ループ

リスト1のTinyUSBのサンプル・プログラムにあるmain関数の処理は，board_init関数，tusb_init関数，メイン・ループとなっています．

board_init関数は，マイコンごとに用意され，LEDピンの初期化後，UARTの初期化が行われ，標準出力のドライバ(RP2040の場合にはstdio_uart_init_full)が設定されます．

tud_init関数でTinyUSBの初期化が始まります．このときパラメータとして，ルート・ハブの番号を指定します．

tud_taskで割り込み処理で発生したイベントのキューからイベントを取り出し，USBデバイスに関わる低レベルの処理を行い，その後，hid_taskで各クラスのタスクが実行されます．

● 初期化処理

tud_init()のデバイスの初期化(**図3**)では，使用されるメモリ領域がクリアされ，イベントを格納するキューと排他制御のためのミューテックスが作成され，ターゲットのデバイスに対応するクラス・ドライバの初期化が行われます．その後，マイコンごとの共通のインターフェースを介して，dcd_init()によってマイコンのデバイスの初期化，dcd_int_enable()によって割り込みの有効化が行われます．

RP2040向けのdcd_init()では，RP2040のUSBのデバイスおよびホストの共通初期化処理であるrp2040_usb_init()が呼び出され，V_{BUS}処理，コントロール・エンドポイントの初期化，非コントロール・エンドポイントのリセット，デバイス・モードの有効化，必要な割り込みビットの設定へと続き，デバイス接続処理であるdcd_connect()が呼び出されます．tud_init()に戻り，dcd_init_enable()で割り込みが有効化されます．

● イベント処理

tud_init()で割り込みが有効化され，初期化処理が完了した後は，割り込みによってUSBデバイス各種処理が起動され，そのイベントがキューに格納され，メイン・ループの中のtud_task()[tud_task_ext()のマクロ定義になっている]で，非同期にイベントが処理されます．

tud_task_ext()では，switch文でイベント

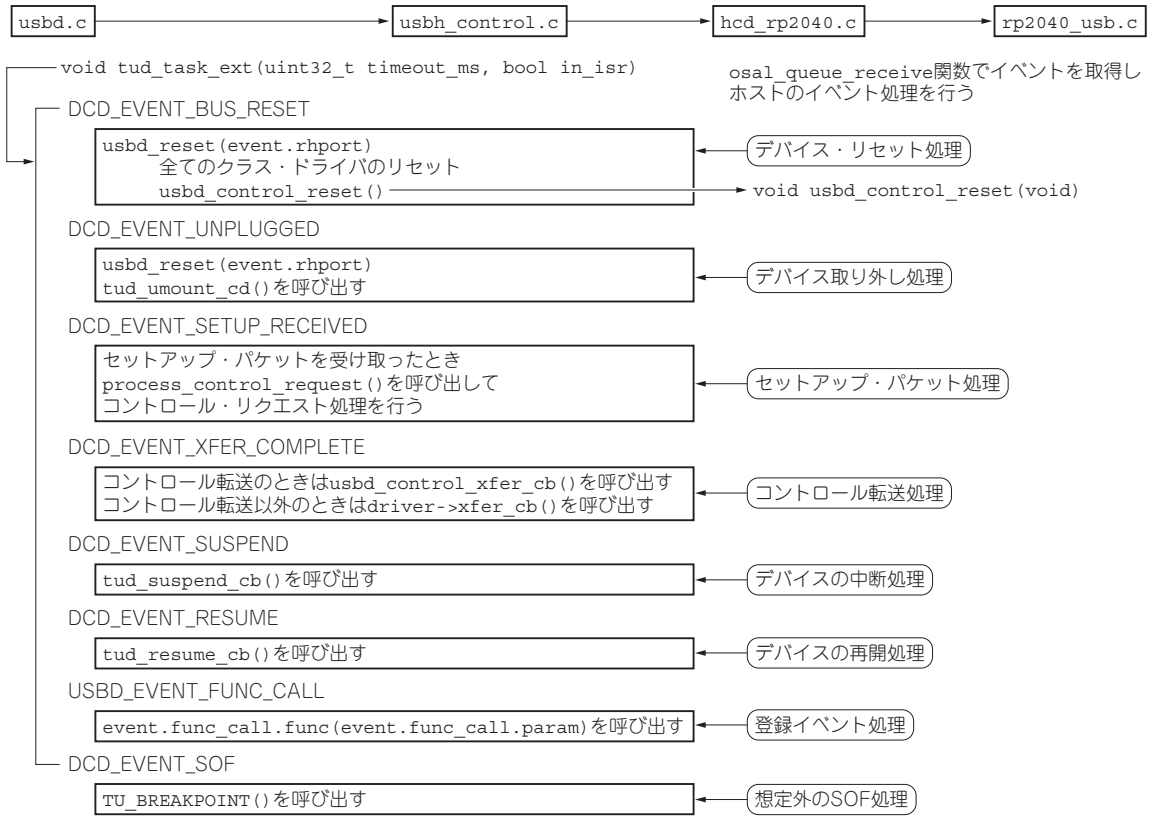

図4　TinyUSBデバイスのタスク処理の流れ

ごとの処理が呼び出される構造になっており，デバイス取り付け時のデバイス・リセット処理，デバイス取り外し処理，セットアップ・パケット処理，コントロール転送処理，デバイスの中断処理，デバイスの再開処理，登録イベント処理，および想定外のSOFの処理が行われます(**図4**)．

● エンドポイントの管理

RP2040マイコンのUSBエンドポイントの処理を見ていきます．hw_endpointは，USBエンドポイントの情報を管理するための構造体です(**リスト2**)．ホストでは追加要素がありますが，その他の要素はデバイス・モードでもホスト・モードでも共通に使われます．USBデバイスの処理では，この構造体はメモリ中に静的に確保されます．

USBデバイス・モードでは，hw_endpoint構造体は，エンドポイントごとにOUT方向とIN方向に2つ確保しています(**リスト3**)．

USBホスト・モードでは，hw_endpoint構造体は，デフォルトのエンドポイント1つと15の割り込み転送用のエンドポイント向けに確保しています(**リスト4**)．

リスト2　RP2040 USBエンドポイントの情報を管理するための構造体

```
typedef struct hw_endpoint
{
    bool configured;
    // この構造体が構成されているかのフラグ
    bool rx;
    // 転送の方向．ホストの時はIN，デバイスの時はOUT
    uint8_t ep_addr;  // エンドポイント・アドレス
    uint8_t next_pid;
    io_rw_32 *endpoint_control;
    // エンドポイント制御レジスタのアドレス
    io_rw_32 *buffer_control;
    // バッファ制御レジスタのアドレス
    uint8_t *hw_data_buf;
    // USB DPSRAM中のバッファのアドレス
    bool active;  // 現在の転送の情報
    uint16_t remaining_len;
    uint16_t xferred_len;
    uint8_t *user_buf;
    // メイン・メモリ中のユーザ・バッファのアドレス
    uint16_t wMaxPacketSize;
   // エンドポイント・ディスクリプタで指定される最大パケット・サイズ
    uint8_t transfer_type;
                // USB転送のクラス(インタラプト，バルク…)
#if TUSB_OPT_HOST_ENABLED
    // ホストの時のみ必要
    uint8_t dev_addr;  // デバイス・アドレス
    uint8_t interrupt_num;
   // インタラプト・エンドポイントであれば，その番号
#endif
} hw_endpoint_t;
```

リスト3　USBデバイス・モードでのhw_endpoint構造体の定義

```
// USB_MAX_ENDPOINTS Endpoints, direction TUSB_DIR_
                OUT for out and TUSB_DIR_IN for in.
static struct hw_endpoint hw_endpoints[
                                USB_MAX_ENDPOINTS][2];
```

リスト4　USBホスト・モードでのhw_endpoint構造体の定義

```
// Host mode uses one shared endpoint register for
                                non-interrupt endpoint
static struct hw_endpoint ep_pool[1 +
                PICO_USB_HOST_INTERRUPT_ENDPOINTS];
#define epx (ep_pool[0])
```

リスト5　TinyUSBホスト処理のmain関数

```
int main(void)
{
  board_init();
:
  tuh_init(BOARD_TUH_RHPORT);
  // TinyUSBのUSBホストとしての初期化関数
:
  while (1)
  {
    tuh_task();// TinyUSBのホスト・タスクの実行
:
#if CFG_TUH_CDC
    cdc_task();  // CDCクラスのタスクの実行
#endif

#if CFG_TUH_HID
    hid_app_task();  // HIDクラスのタスクの実行
#endif
  }
  return 0;
}
```

USBホスト処理

● メイン・ループ

リスト5のTinyUSBのサンプル・プログラムのmain関数の処理は，board_init関数，tuh_init関数，メイン・ループとなっています．

board_init関数では，LEDピンの初期化後，UARTの初期化が行われ，標準出力のドライバ(stdio_uart_init_full)が設定されます．

tuh_init関数でTinyUSBの初期化が始まります．このときパラメータとして，ルート・ハブの番号を指定します．

tuh_task()で割り込み処理で発生したイベントのキューからイベントを取り出し，USBホストに関わる低レベルの処理を行い，その後，cdc_task()とhid_app_task()で各クラスのタスクが実行されます．

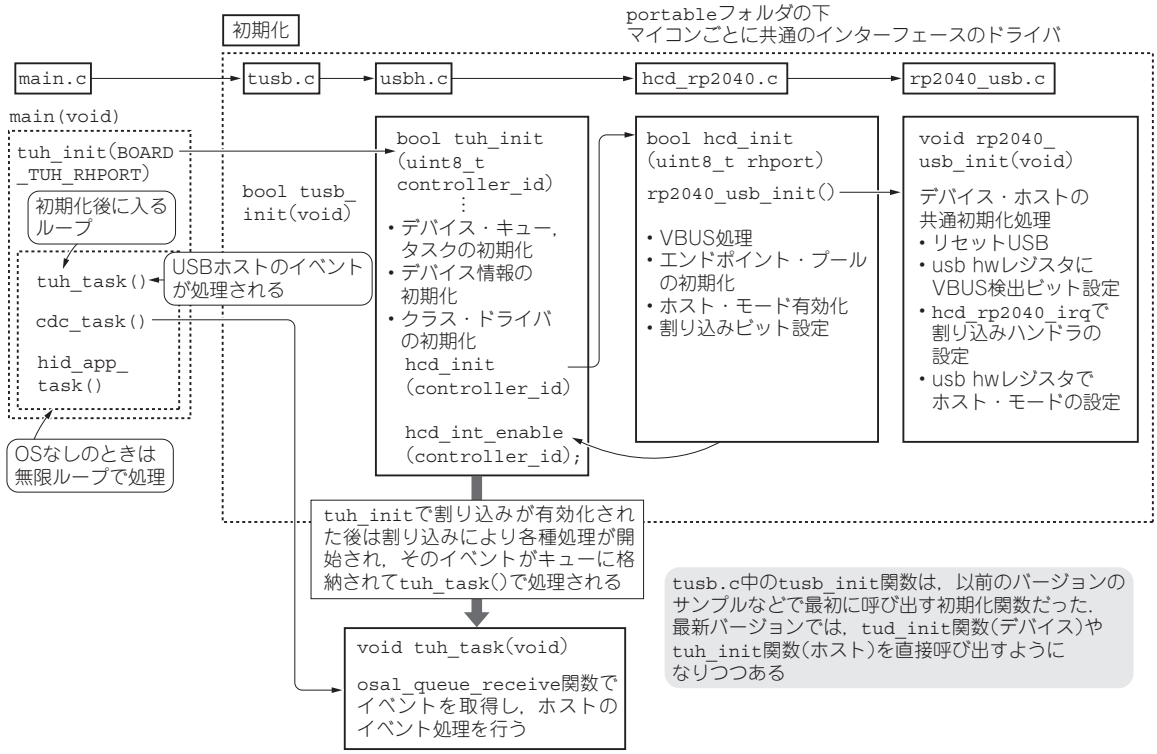

図5　TinyUSBホスト処理の初期化フロー

● 初期化処理

USBホスト・アプリケーションの場合には，tuh_init関数が呼び出され，USBホストの初期化が行われます(**図5**)．ホスト・ドライバで使用されるメモリ領域がクリアされ，イベントを格納するためのキューと排他制御用のミューテックスが作成されます．

初期化が終了すると，ホスト・ドライバのイベント処理tuh_task(void)と，USBホストのミドルウェア処理の無限ループのサイクル[cdc_task()やhid_app_task()]に入ります．

● イベント処理

tuh_task()では，USBデバイスのアタッチ，デタッチ，データの転送完了および割り込みで遅延スケジュールされたイベントなどといったUSBホストのイベントが処理されます(**図6**)．

TinyUSBの共通手順として，コントロール転送はusbh_edpt_control_open()でコントロール転送に必要な設定情報をUSBコントローラに設定し，tuh_control_xfer()でコントロール転送の詳細パラメータを指定し，転送を開始します(**図6**)．

▶usbh_edpt_control_open

usbh_edpt_control_open()は，hcd_rp2050.c中のhdc_end_edpt_open()を呼び出しています．詳細は以下の流れです．

hcd_edpt_open()では，指定したroothubポート番号，USBデバイス・アドレスおよびUSBエンドポイント・ディスクリプタ構造体を指定して，ディスクリプタ構造体に設定されているUSB転送情報をコントローラに設定します．

_hw_endpoint_allocate()では，エンドポイント構造体の領域を，エンドポイントのプール領域から新たに確保します．そして，インタラプト転送とそれ以外の場合で，それぞれ，エンドポイント・バッファ制御レジスタのアドレス，エンドポイント制御レジスタのアドレス，およびデータ・バッファのアドレスをエンドポイント構造体に設定します．

_hw_endpoint_init()では，エンドポイント

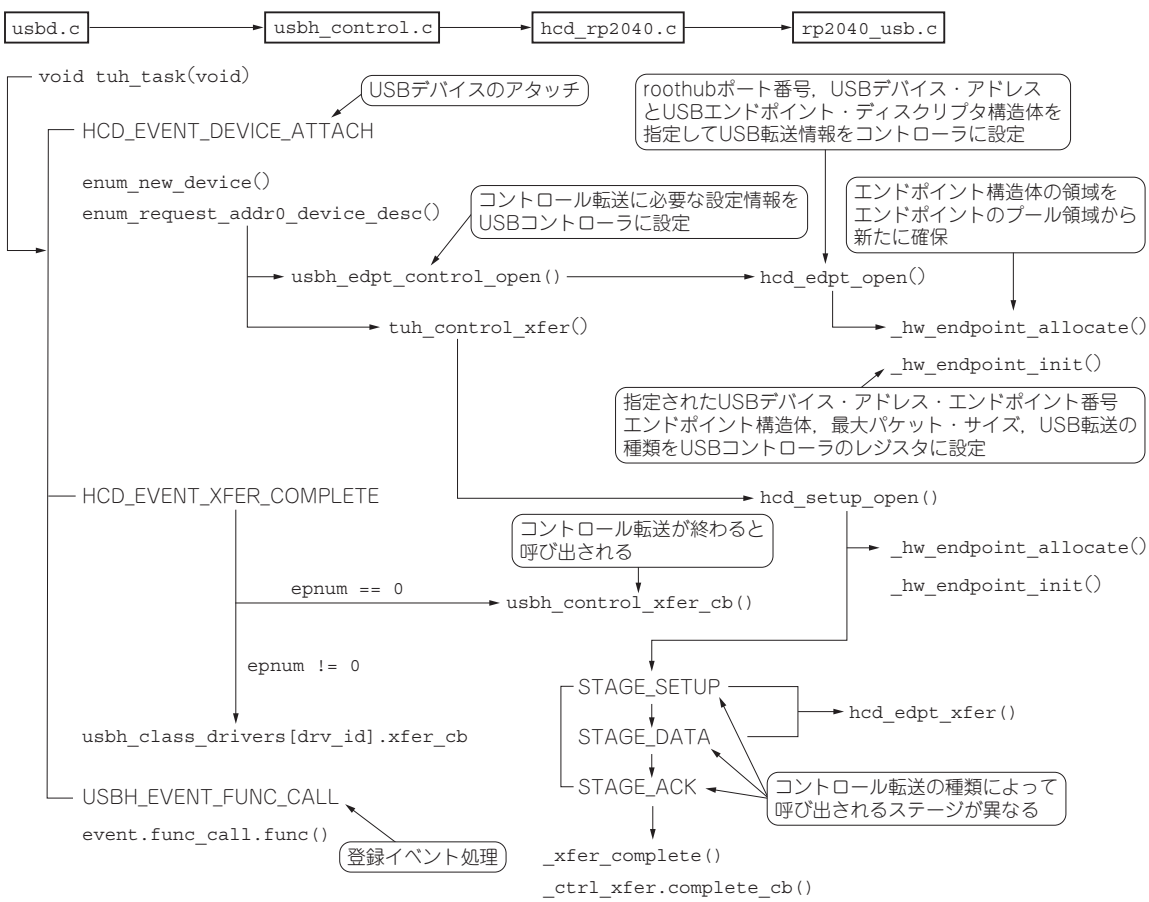

図6　tuh_task()ではUSBホストのイベントが処理される

コラム デバッグ・メッセージの有効化

関本 健太郎

● メッセージの数は3段階で設定できる

TinyUSB組み込みのデバッグ用のログ・メッセージの表示は，TU_LOGマクロで行います．tusb_debug.hファイルの中でprintfまたは独自のCFG_TUSB_DEBUG_PRINTFを呼び出すTU_LOG1，TU_LOG2，TU_LOG3の3段階のマクロが用意されており，CFG_TUSB_DEBUGの値により，3段階のマクロの一部が有効になります．例えば，CFG_TUSB_DEBUGの値が2であれば，TU_LOG1とTU_LOG2が有効になります．

RP2040の環境では，CFG_TUSB_DEBUGの値は，hw/bsp/rp2040/family.cmakeの中で，CMAKE_BUILD_TYPE="Debug"のときに強制的に1に設定されるようになっています（**リストA**）．従って，CFG_TUSB_DEBUGを2または3に設定する場合には，set(TINYUSB_DEBUG_LEVEL 1)の1を2または3に書き換えるとよいでしょう．ただし，CFG_TUSB_DEBUGを2以上に設定すると，表示されるデバッグ・メッセージの数が増えるので，メッセージの表示のための遅延がUSB通信を妨げ，通信エラーが発生することがあります．

● PicoprobeのUSB CDC出力

ラズベリー・パイPicoの代表的なデバッガであるPicoprobeでRP2040をデバッグする場合には，デフォルトでGP0，GP1ピンでuart0を有効にし，PicoprobeのUSB CDCでuart0のシリアル出力を観察することになると思います．その場合には，ソース・ファイルの先頭（例えばmain関数）で，stdio_init_all()を呼び出す（**リストB**）一方，CMakeLists.txtファイル中で標準出力をUSBからUARTに設定します（**リストC**）．

リストA CFG_TUSB_DEBUGの値はCMAKE_BUILD_TYPE="Debug"のとき強制的に1となる

```
:
set(TINYUSB_DEBUG_LEVEL 0)
if (CMAKE_BUILD_TYPE STREQUAL "Debug")
    message("Compiling TinyUSB with CFG_TUSB_DEBUG=1")
    set(TINYUSB_DEBUG_LEVEL 1)
endif()
:
```

リストB main.cでstdio_init_all()を呼び出す

```
int main(void)
{
:
  stdio_init_all();
:
}
```

リストC CMakeLists.txtで標準出力をUARTに設定する

```
:
pico_enable_stdio_uart(${PROJECT} 1)
pico_enable_stdio_usb(${PROJECT} 0)
:
```

構造体の各レジスタ・アドレス情報をもとに，指定されたUSBデバイス・アドレス，エンドポイント番号エンドポイント構造体，最大パケット・サイズおよびUSB転送の種類を，USBコントローラのレジスタに設定します．

▶tuh_control_xfer

tuh_control_xfer()は，hcd_rp2040.c中のhdc_set_up_send()を呼び出し，コントロール転送を開始しています．

hdc_set_up_send()は，セットアップ・パケットの領域をゼロ・クリアし，_hw_endpoint_allocate()でエンドポイント構造体を準備し，_hw_endpoint_init()でコントロール転送向けにUSBコントローラのレジスタを設定します．最後にUSB制御レジスタのSIE_CTRLレジスタにコントロール転送のセットアップ・ステージと転送開始のビットを立てて，転送を開始します．

▶usbh_control_xfer_cb

コントロール転送が完了すると，上位のイベント・ループで転送完了イベントが発生し，usbh_control_xfer_cb()が呼び出されます．その中でセットアップ・ステージ(STAGE_SETUP)，データ・ステージ(STAGE_DATA)，アクノリッジ・ステージ(STAGE_ACK)といったように，コントロール転送の各ステージの処理が実行されます（ホストのコントロール転送の種類によってどのステージ呼び出されるかは異なる）．

せきもと・けんたろう