ISIXRTOS obsługa USB-Host na platformie STM32

na przykładzie klawiatury oraz joysticka

Interfejs USB jest jednym z powszechnie używanych interfejsów komunikacyjnych, stąd również pojawiła się potrzeba obsługi interfejsu USB w rozwiązaniach wykorzystujących mikrokontrolery. W większości przypadków mikrokontroler pełni rolę urządzenia USB, które jest dołączane do komputera hosta. Taki przykład zaprezentowano w jednym z poprzednich odcinków prezentując obsługę wirtualnego portu, gdzie do złącza USB komputera dołączono zaprogramowany mikrokontroler zgłaszający się jako port szeregowy. Było to zadanie stosunkowo skomplikowane, niemniej jednak dzięki dodatkowym bibliotekom i systemowi użycie wirtualnego portu sprowadzało się do napisania kilku linijek kodu.

Wyobraźmy sobie jednak sytuację odwrotną, gdy nasze urządzenie wyposażone w mikrokontroler będzie spełniać rolę kontrolera hosta USB. Takie rozwiązanie daje ogromne możliwości dostępu do bardzo tanich komputerowych urządzeń peryferyjnych, które dzięki masowej produkcji mogą być dużo tańsze od dedykowanych rozwiązań.

Napisanie kontrolera hosta USB jest zadaniem zdecydowanie bardziej skomplikowanym niż oprogramowanie urządzenia USB, oraz wymaga również mikrokontrolera, który jest wyposażony w kontroler USB host, lub USB OTG. W przypadku rodziny STM32 większe mikrokontrolery np. STM32F105/107, STM32F205, STM32F407 wyposażono w kontroler USB OTG, tak więc sprawa sprzętowa jest rozwiązana, pozostaje jeszcze kwestia oprogramowania.

Przeglądając rozwiązania programowe, które można było zastosować do obsługi USB pierwszym rozwiązaniem jakie przychodzi na myśl jest wykorzystanie bibliotek hosta dostarczonych przez producenta. Po dogłębnej analizie okazało się jednak, że jakość tych bibliotek pozostawia wiele do życzenia, w związku z czym rozwiązanie to zostało odrzucone. Do największych wad biblioteki dostarczonej od STM32 możemy zaliczyć wielki bałagan w kodzie źródłowym (np. zmienne globalne, nadużywanie słowa kluczowego extern), osobna biblioteka dla każdego kontrolera USB (np. dla rodziny F103 i F105) czy brak bezpośredniej przenośności kodu na inne mikrokontrolery.

Poszukując odpowiedniego rozwiązania natknąłem się na doskonały stos USB autorstwa Marcina Peczarskiego, stanowiący dodatek do książki jego autorstwa „USB dla niewtajemniczonych". Po analizie kodu źródłowego doszedłem do wniosku, że doskonale będzie nadawał się jako moduł bazowy dla systemu ISIXRTOS. Po nawiązaniu kontaktu z autorem oraz uzyskaniu zgody na wykorzystanie w systemie, przystąpiłem do pracy polegającej na dostosowaniu go do działania pod nadzorem systemu operacyjnego. W ten sposób powstało jądro USB systemu ISIX, które cechuje się następującymi parametrami:

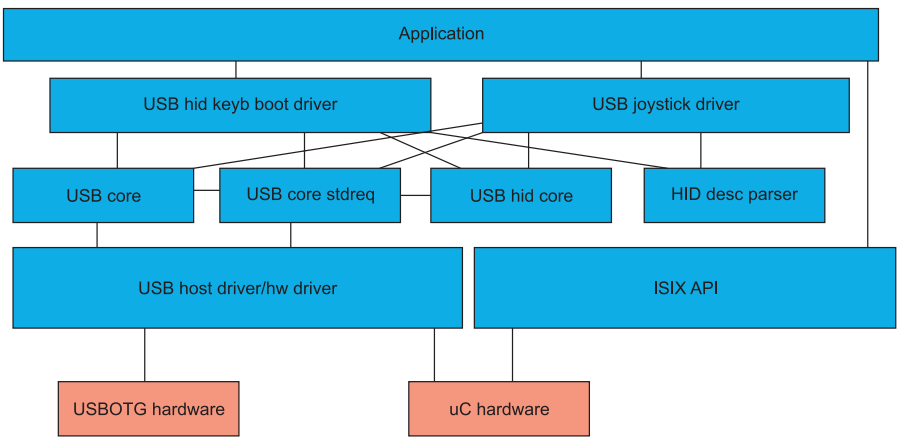
- Struktura warstwowa.
- Łatwa możliwość zamiany mikrokontrolera czy hosta USB poprzez przygotowanie odpowiednich sterowników.
- Warstwa abstrakcji uniezależniająca API sterowników od danego hosta i mikrokontrolera.
- Obsługa trybu USB Host oraz USB Device.

W aktualnej implementacji, nie ma możliwości obsługi protokołu służącego do obsługi HUBów, a więc ilość równocześnie obsługiwanych urządzeń ograniczona jest do jednego. Wykorzystanie stosu znacząco ułatwia rozwiązanie problemu, nie mniej pomimo tego nadal jest stosunkowo skomplikowane i wymaga znajomości podstaw protokołu USB. Z powyższego powodu dla rozpoczynających przygodę z USB przygotowano zestaw przykładowych sterowników HID umożliwiających obsługę klawiatury USB oraz JOYSTICK-a. Wzorując się na powyższym rozwiązaniu użytkownik, będzie w stanie stosunkowo niewielkim nakładem środków przygotować sterownik dla innego typu urządzenia.

Architektura stosu USB w systemie ISIXRTOS

Pierwotna wersja stosu została napisana w języku C. Aby nie zmieniać jego struktury, pozostałe sterowniki oraz dodatkową infrastrukturę warstwy sterowników również napisano w języku C, ale w taki sposób aby istniała możliwość, wykorzystania go z poziomu języka C++ Architektura stosu USB zorganizowana jest w sposób modułowy (**rysunek 1**).

Rysunek reprezentuje architekturę stosu USB w prezentowanym przykładzie obsługi klawiatury oraz Joysticka HID USB. Za fizyczną komunikację z urządzeniem USB odpowiedzialny jest sprzętowy kontroler USB *On The Go*. Wykorzystując wewnętrzne zasoby mikrokontrolera umożliwia komunikację w trybie *USB2.0 Full Speed*, charakteryzującą się prędkością transmisji 12 Mbps. Niektóre z układów rodziny STM32 umożliwiają również komunikację w trybie *High Speed* (480 Mbps), Rozwiązanie to jest jednak stosunkowo mało



Rysunek 1. Reprezentacja architektury stosu USB w prezentowanym przykładzie obsługi klawiatury oraz Joysticka HID USB

wygodne z uwagi na konieczność użycia zewnętrznego układu PHY i nie jest zbyt często stosowane. Biblioteka USB ma również zintegrowaną obsługę trybu *High Speed*. Najniższą warstwę obsługi USB stanowi warstwa sterownika kontrolera OTG, która zapewnia jednolity interfejs API uniezależniający kod stosu USB od zastosowanego mikrokontrolera. Sercem biblioteki jest moduł *USB Core*, który jest odpowiedzialny za obsługę rdzenia protokołu USB, czyli: wykrywanie urządzenia, enumerację, komunikację z urządzeniem. Udostępnia również wyższym warstwom sterownika interfejs niezależny od sprzętu. Moduł *HID Core* jest odpowiedzialny za realizację obsługi standardowych żądań protokołu USB HID zapewniając niezależne API dla sterowników HID, obsługujące standardowe i najczęściej używane żądania. Standard HID oprócz standardowych deskryptorów USB, jak np. deskryptor urządzenia czy interfejsu, definiuje również deskryptor HID zawierający opis funkcjonalności danego urządzenia HID. Aby odpowiednio obsłużyć konkretny typ urządzenia, konieczne jest odpowiednie parsowanie tego deskryptora celem określenia wszystkich jego właściwości. Specyfikacja HID definiuje wiele różnych urządzeń, takich jak: klawiatury, myszy, joysticki, pady, manipulatory, sygnalizatory itp. Mnogość obsługiwanych urządzeń oraz uniwersalność specyfikacji powoduje duże skomplikowanie struktury deskryptora.

Aby ułatwić parsowanie deskryptorów HID, zastosowano moduł HID DescParser, który udostępnia jednolity interfejs, i ułatwia analizę charakterystyki urządzenia. Kolejną warstwę abstrakcji stanowią już konkretne sterowniki urządzeń, udostępniające określoną funkcjonalność dla aplikacji użytkownika. Kod sterowników wykorzystuje API rdzenia protokołu USB oraz HID, i jest niezależne od warstwy sprzętowej. W systemie ISIX do hosta USB możemy przypisać wiele różnych sterowników urządzeń, ale w tym samym czasie może pracować tylko pojedyncze urządzenie, zatem aktywny jest tylko jeden ze sterowników. Wybór odpowiedniego sterownika odbywa się automatycznie. Na przykład, jeżeli kontroler hosta, posiada zarejestrowane sterowniki dla klawiatury USB i Joysticka, to podłączenie klawiatury do gniazda USB spowoduje wybranie odpowiedniego sterownika. Jeśli natomiast podłączymy urządzenie dla którego nie uda się znaleźć sterownika, wówczas kontroler hosta zwróci błąd informujący o braku sterownika dla danego urządzenia. Nie będziemy tutaj opisywać szczegółów implementacji, z uwagi na ograniczoną objętość czasopisma, zainteresowanych odsyłam do wspomnianej wcześniej książki.

USB host w ISIXRTOS jak najprościej (Klawiatura i Joystick)

Podłączenie urządzenia USB do hosta w mikrokontrolerze nie musi oznaczać konieczności poznania szczegółów implementacji USB. Mając do dyspozycji gotowe rozwiązanie zawarte w systemie ISIX, użycie urządzeń USB może być bardzo proste i sprowadza się do wywołaniu kilku funkcji. Wszystkie czynności związane z obsługa USB realizowane są przez stos systemowy. Aby zademonstrować działanie sterownika HID napisano prostą aplikację, która umożliwia podłączenie do gniazda USB, dowolnej klawiatury lub joysticka, zgodnego ze standardem HID. Przykład został przygotowany dla zestawu STM32Butterfly. Działanie przykładu polegać będzie na wyświetlaniu na standardowej konsoli szeregowej kodu wciśniętego klawisza lub statusu odchylenia danej osi joysticka, w zależności od dołączonego urządzenia. Aby zobaczyć efekt działania powyższego przykładu linię PD5 i PD6 należy dołączyć do linii RX i TX konwertera stanów logicznych na układzie MAX232 (**rysunek 2**).

Należy również dołączyć zasilanie zestawu, oraz programator JTAG, którym zaprogramujemy mikrokontroler. Skompilowany przykład wraz z kodami źródłowymi dostępny jest w następującej lokalizacji: http://goo.gl/8WiJyW. Aby zaprogramować mikrokontroler należy do złącza programującego zestawu dołączyć dowolny JTAG zgodny z OCDLINK, a następnie wydać polecenie *make program*. Po zaprogramowaniu mikrokontrolera należy uruchomić dowolny program terminalowy skonfigurowany według następujących parametrów transmisji: 115200, n, 8, 1. Po dołączeniu do złącza USB klawiatury, w oknie terminala powinna pojawić się informacja o wykryciu klawiatury, a następnie informacja o producencie oraz modelu. Po wykryciu urządzenia na dołączonej klawiaturze możemy zacząć wciskać dowolne klawisze, które w tym samym czasie powinny pojawić się na konsoli szeregowej. Odłączenie klawiatury powinno skutkować wykryciem stanu odłączenia klawiatury oraz stosowną informacją w konsoli (**rysunek 3**). Jeśli do dyspozycji mamy joystick możemy również dołączyć go do złącza USB. Zmiana odchylenia dowolnej dźwigni skutkować będzie cyklicznym wysyłaniem stanu wychylenia poszczególnych osi.

Interfejs API użytkownika został napisany w taki sposób, aby maksymalnie uprościć korzystanie z USB i został oparty jest o system zdarzeń. Wystąpienie zdarzenia (np. wciśnięcie klawisza, dołączenie urządzenia do gniazdka USB) powoduje wywołanie odpowiedniej funkcji zwrotnej (*callback*), która została zarejestrowana w sterowniku. Kod aplikacji demonstracyjnej znajduje się w pliku *usbhman.cpp*. Program rozpoczyna wykonanie od funkcji **main** (**listing 1**).

Na początku jest inicjalizowana konsola szeregowa oraz tworzony obiekt klasy *blinker*, którego zadaniem jest cykliczne sterowanie diodami LED zestawu w odstępach czasu co 1 sekundę. Zadaniem funkcji *usbh_controler_init* jest inicjalizacja rdzenia USB, jako pierwszy argument funkcja przyjmuje identyfikator kontrolera USB, natomiast jako drugi argument jest przyjmowana wartość priorytetu systemu

Listing 1. Funkcja *main*

```
int main()
{
   dblog_init(stm32::usartsimple_putc, NULL, stm32::usartsimple_init,
              USART2,115200, true, CONFIG_PCLK1_HZ, CONFIG_PCLK2_HZ );
  static app::ledblink blink;
  dbprintf(„USBHost app started OK”);
  // test
  usbh_controller_init(USB_PHY_A, 2); [1]
  usbh_controller_attach_driver( usbh_hid_keyboard_init(&kbd_fops) );
  usbh_controller_attach_driver( usbh_hid_joystick_init(&joy_ops) );
  dbprintf(„Joystick and KBD initialized and completed”);
  //Start the isix scheduler
  isix::isix_start_scheduler();
  dbprintf(„Scheduler exit”);
}
```

Listing 2. Struktura *usbh_hid_kbd_ops* zawierająca zestaw funkcji zwrotnych wywoływanych w reakcji na zdarzenia

```
namespace {
  void  kbd_connected( const usbh_keyb_hid_context_t* id ) {
  dbprintf(„Keyb ID %p connected”, id );
  }
  void  kbd_disconnected( const usbh_keyb_hid_context_t* id ) {
    dbprintf(„Keyb ID %p disconnected”, id );
  }
  void kbd_report( const usbh_keyb_hid_context_t* , const usbh_keyb_hid_
event_t* evt ) {
    if( evt->key ) {
      fnd::tiny_printf(„%c”, evt->key );
    } else {
    //dbprintf(„S %i %02x”, evt->scan_code, evt->scan_code );
    }
  }

  void kbd_desc( const usbh_keyb_hid_context_t* id, usbh_driver_desc_type
desc, const char *str ) {
    dbprintf(„ID %p Desc %i str: %s”, id, desc, str );
  }
  constexpr usbh_hid_kbd_ops kbd_fops = {
    kbd_connected,
    kbd_disconnected,
    kbd_report,
    kbd_desc
  };
}
```

ISIX, z którym będzie wykonywany wątek stosu. W kolejnej linii za pomocą funkcji *usbh_controller_atach_driver* do jądra USB jest dołączany sterownik klawiatury. Funkcja ta jako argument przyjmuję strukturę *usbh_driver*, która jest zwracana przez funkcję *usbh_hid_keyboard_init()* inicjalizującą sterownik klawiatury. Funkcja ta jako argument przyjmuje strukturę *usbh_hid_kbd_ops*, która zawiera zestaw funkcji zwrotnych wywoływanych w reakcji na zdarzenia. (**listing 2**).

Funkcja *kbd_connect*, wywoływana jest w kontekście wątku stosu USB w momencie wykrycia podłączenia klawiatury, jako argument zwracana jest struktura prywatna przechowująca status kontrolera klawiatury. Wskaźnik do tej struktury jednoznacznie identyfikuje dołączone urządzenie, i może być wykorzystany w przyszłości do jednoznacznej identyfikacji, gdy zostanie zaimplementowana obsługa HUBów lub wielu kontrolerów. Podobną rolę pełni funkcja *usb_disconnect*, która jest wywoływana w momencie odłączenia klawiatury USB od hosta. Funkcja *kbd_report* wywoływana jest w momencie zmiany stanu klawiatury, czyli w praktyce w reakcji na wciśnięcie lub zwolnienie klawisza. Istotny jest tutaj parametr *evt*, który zawiera strukturę zwracającą stan klawiatury:

```
struct usbh_keyb_hid_event {
  usbh_keyb_hid_context_t* kbd_id;
//! Keyboard initializer
  uint8_t key;
//! Translated key to UE char
  uint8_t scan_bits;
//! Special scan bits keys
  uint8_t scan_code;
//! Scan code extra key
};
```

Pole *key* zawiera kod wciśniętego klawisza, w postaci znaku ASCII, który reprezentuje wciśnięty klawisz. Jeśli kod ma wartość 0 znaczy to, że zwrócony klawisz jest klawiszem specjalnym, który nie ma reprezentacji w standardowym zestawie znaków. W takim przypadku stan klawiszy można sprawdzić badając pozostałe pola struktury. Pole *scan_bits* zawiera, stan bitów klawiszy specjalnych takich jak **CTRL, SHIFT, WIN** (lewy oraz prawy), natomiast pole *scan_code* zawiera kody klawiszy zwracane bezpośrednio przez klawiaturę bez dodatkowej translacji.

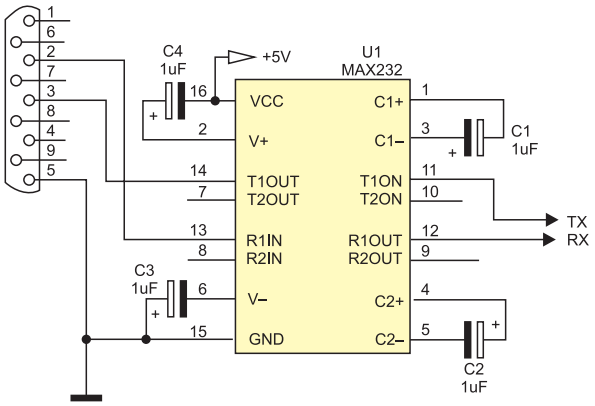
Funkcja *kbd_desc* jest wywoływana w momencie odczytania deskryptora tekstowego klawiatury i zawiera informacje zakodowane w deskryptorze USB przez producenta urządzenia. Parametr *str* zawiera wskaźnik do łańcucha tekstowego, natomiast parametr *desc* zawiera rodzaj odczytanego deskryptora:

```
enum usbh_driver_desc_type {
  usbh_driver_desc_product,
  usbh_driver_desc_manufacturer,
  usbh_driver_desc_serial
};
```

Listing 3. Struktura *joy_ops*

```
namespace {
  void joy_connected( const usbh_hid_joy_context_t* id ) {
    dbprintf(„Joy ID %p, connected, „ , id);
  }
  void joy_disconnected( const usbh_hid_joy_context_t* id ) {
    dbprintf(„Joy ID %p, disconnected, „ , id );
  }
  void joy_desc( const usbh_hid_joy_context_t* id, usbh_driver_desc_type desc,
const char *str ) {
    dbprintf(„Joy ID %p Desc %i str: %s”, id, desc, str );
  }
  void joy_report( const usbh_hid_joy_context_t *id, const usbh_joy_hid_
event_t* evt ) {
    dbprintf(„Joy ID %p events”, id );
    if( evt->has.X ) {
      dbprintf(„X_pos=%u”, evt->X );
    }
    if( evt->has.Y ) {
      dbprintf(„Y_pos=%u”, evt->Y );
    }
    if( evt->has.Z ) {
      dbprintf(„Z_pos=%u”, evt->Z );
    }
    if( evt->has.rX ) {
      dbprintf(„rX_pos=%u”, evt->rX );
    }
    if( evt->has.rY) {
      dbprintf(„rY_pos=%u”, evt->rY );
    }
    if( evt->has.rZ ) {
      dbprintf(„rZ_pos=%u”, evt->rZ );
    }
    if( evt->has.hat ) {
      dbprintf(„hat_pos=%u”, evt->hat );
    }
    if( evt->has.slider ) {
      dbprintf(„slider_pos=%u”, evt->slider );
    }
    if( evt->n_buttons > 0 ) {
      dbprintf(„got %u buttons val %x”, evt->n_buttons, evt->buttons );
    }
  }
  constexpr usbh_hid_joystick_ops joy_ops = {
    joy_connected,
    joy_disconnected,
    joy_report,
    joy_desc,
  };
}
```

Gdzie w zależności od przyjętej wartości oznacza odpowiednio według kolejności: deskryptor tekstowy opisujący produkt, deskryptor tekstowy opisujący producenta, deskryptor tekstowy opisujący numer seryjny urządzenia. W podobny sposób przebiega inicjalizacja sterownika joysticka, i jego rejestracja w jądrze stosu USB, co jest realizowane za pomocą wywołania *usbh_controller_attach_driver(usbh_hid_joystick_init(&joy_ops));*. Podobnie jak poprzednio zestaw funkcji zwrotnych, wywoływanych w reakcji na zdarzenia od joysticka rejestrowany jest za pomocą struktury *joy_ops* pokazanej na **listingu 3**.



Rysunek 2.

Rysunek 3.

Zakończenie

Zestaw funkcji zwrotnych jest w zasadzie identyczny jak w poprzednim przypadku, różna jest jedynie definicja funkcji *joy_report* wywoływanej w momencie zmiany statusu joysticka. W tym przypadku zwracana jest struktura *usbh_joy_hid_event_t*, zawierająca statusy odchylenia poszczególnych osi joysticka, oraz stan dodatkowych klawiszy.

Zagadnienia obsługi stosu USB są dość skomplikowane niemniej jeśli dysponujemy systemem ISIX, dołączenie klawiatury USB czy Joysticka jest równie proste jak dołączenie zewnętrznej klawiatury do portów GPIO. Napisanie dodatkowego sterownika dla innych typów urządzeń będzie zadaniem nieco bardziej skomplikowanym, jednak postaram się, aby w miarę potrzeb w systemie znalazły się inne dodatkowe sterowniki do najczęściej spotykanych urządzeń.

Lucjan Bryndza SQ5FGB, EP