STM32CubeMX – tworzenie urządzeń USB HID

Środowisko CubeMX umożliwia tworzenie szkieletów programów dla mikrokontrolerów STM32. Dostępny jest w nim m.in. stos USB wraz z obsługą wybranych klas urządzeń takich jak CDC, MSC czy HID. W artykule przedstawiono sposób tworzenia modeli urządzeń HID (Human Interface Device): myszy i klawiatury USB przy użyciu CubeMX.

Urządzenia USB klasy HID służą głównie do interakcji z użytkownikiem. Ich istotną zaletą jest możliwość współpracy z oprogramowaniem komputera osobistego bez konieczności tworzenia i instalowania w systemie programów sterowników. Dla większości urządzeń HID oprogramowanie aplikacyjne może korzystać ze standardowych sterowników HID obecnych w systemie operacyjnym. Urządzenia HID mogą mieć funkcje standardowe, takie jak klawiatura lub mysz, albo niestandardowe. Urządzenia standardowe działają i są przez system operacyjny traktowane identycznie, jak typowa klawiatura lub mysz. Urządzenia niestandardowe są widoczne w systemie tylko dla aplikacji, które znają je i chcą z nich korzystać.

Interfejs logiczny urządzenia HID

W standardzie USB urządzenie jest hierarchią, której kolejne poziomy logiczne stanowią funkcje, interfejsy i punkty końcowe. Urządzenie może być wyposażone w jedną lub więcej funkcji. Funkcja – to pojedyncze urządzenie logiczne, np. drukarka, klawiatura, pamięć masowa lub wirtualny port szeregowy. Funkcja zawiera jeden lub więcej interfejsów, reprezentujących logiczne składniki urządzenia (np. interfejs sterowania i interfejs danych). Interfejs może korzystać z jednego lub dwóch punktów końcowych (endpoint). Interfejsy są w obrębie urządzenia identyfikowane przez numery. Numeracja interfejsów jest ciągła i rozpoczyna się od zera.

Punkt końcowy jest jednokierunkowym kanałem transmisyjnym. Do transmisji dwukierunkowej używa się pary punktów końcowych. Przy opisie urządzenia kierunek transmisji punktu końcowego określa się z perspektywy urządzenia nadrzędnego. Punkt końcowy wyjściowy (OUT endpoint) służy do przesyłania danych z komputera nadrzędnego do urządzenia, a punkt końcowy wejściowy (IN endpoint) – z urządzenia do komputera. Punkty końcowe są identyfikowane przez numery. Każde urządzenie ma interfejs sterowania, korzystający z pary punktów końcowych o numerach 0. Punkty o wyższych numerach służą do realizacji interfejsów. Numeracja punktów końcowych wejściowych i wyjściowych jest niezależna i nie musi być ciągła; urządzenie może np. korzystać z punktów końcowych wyjściowych o numerach 1 i 3 oraz punktów końcowych wejściowych o numerach 1 i 5.

Urządzenie HID, tak, jak wszystkie urządzenia USB, ma dwukierunkowy interfejs sterowania. Ponadto ma ono zwykle interfejs danych, złożony z jednego lub dwóch punktów końcowych. Przez wejściowy punkt końcowy notyfikacji (Interrupt In Endpoint) komputer nadrzędny cyklicznie pobiera od urządzenia raporty, zawierające informację o stanie urządzenia, np. o wciśniętych przyciskach lub klawiszach, pozycji lub przesunięciu manipulatora. Wyjściowy punkt końcowy notyfikacji (Interrupt Out Endpoint) może służyć do przesyłania przez komputer danych, które w przypadku klawiatury służą do sterowania świeceniem wskaźników. Niektóre urządzenia mogą nie używać tego punktu końcowego, a informacja sterująca urządzeniem może być przesyłana przez interfejs sterowania (punkt końcowy 0).

Deskryptory urządzenia HID

Każde urządzenie USB jest identyfikowane przez komputer nadrzędny na podstawie zawartości dwóch struktur danych – deskryptora urządzenia i deskryptora konfiguracji, których budowę opisuje standard USB. Deskryptor konfiguracji zawiera deskryptory interfejsów i punktów końcowych.

Deskryptor urządzenia (device descriptor) w przypadku urządzeń HID nie zawiera informacji o klasie i podklasie urządzenia – stosowne pola mają wartości 0. Informacja o klasie i podklasie jest zawarta w deskryptorach interfejsów.

Dla urządzeń HID zdefiniowana jest podklasa 1, która oznacza uproszczone wersje urządzeń obsługiwane przez standardowe oprogramowanie wbudowane komputerów osobistych, przed załadowaniem systemu operacyjnego. Wyróżnione zostały 2 takie urządzenia rozróżniane przez wartość pola Protocol deskryptora interfejsu: klawiatura (Protocol = 1) oraz mysz (Protocol = 2). Urządzenia te muszą obsługiwać określony w standardzie protokół, tj. przesyłać i odbierać raporty o sztywno określonym formacie, opcjonalnie rozszerzonym o dodatkowe dane. Urządzenia nieobsługujące uproszczonych protokołów mają pola podklasy i protokołu o wartościach 0.

Deskryptory raportów

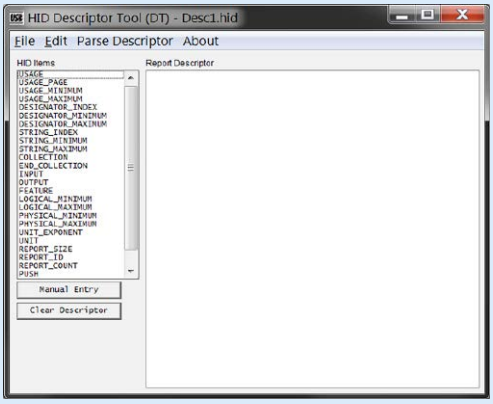
Ze względu na różnorodność urządzeń HID, w celu umożliwienia ich jednolitej obsługi przez oprogramowanie systemowe, wprowadzono deskryptory raportów – struktury danych przechowywane przez urządzenie i pobierane przez system operacyjny, opisujące format i znaczenie danych wymienianych przez urządzenie z komputerem. Deskryptor raportu jest charakterystycznym dla klasy HID, dodatkowym deskryptorem, uzupełniającym wymagane przez standard USB w urządzeniach wszystkich klas deskryptory urządzenia i konfiguracji. Stanowi on „wizytówkę", którą urządzenie HID przedstawia się systemowi operacyjnemu.

Pobranie i interpretacja deskryptora raportu umożliwia oprogramowaniu systemowemu poprawną współpracę z dowolnym urządzeniem klasy HID. Dzięki temu dane przesyłane przez urządzenie HID mogą mieć format wynikający z cech danego urządzenia, a nie sztywno wymuszony przez sam fakt przynależności urządzenia do określonej klasy i podklasy.

Program HID Descriptor Tool

Na stronie USB Implementers Forum pod adresem *https://goo.gl/8c7VqX* można znaleźć dokumentację klasy HID oraz program HID

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl



Rysunek 1. Widok okna programu HID Descriptor Tool

Descriptor Tool (**rysunek 1**), który umożliwia tworzenie deskryptorów raportów w środowisku graficznym przy pomocy zrozumiałych dla człowieka nazw, a następnie wygenerowanie na ich podstawie zawartości deskryptora raportu w postaci wektora wartości heksadecymalnych.

Po pobraniu programu należy go rozpakować, przejść do katalogu MSDEV\Projects\test i uruchomić program Dt.exe. W tym samym folderze znajdują się pliki o rozszerzeniach .hid, zawierające przykładowe deskryptory popularnych urządzeń takich jak mysz, klawiatura czy joystick.

W celu umieszczenia wpisu należy kliknąć dwukrotnie na jednym z elementów w panelu po lewej stronie. Jeśli dany element wymaga określenia wartości, pojawi się okno, gdzie można wybrać jeden z elementów z listy lub wpisać wartość – dopuszczalne wartości zależą od rodzaju wybranego elementu. Po dodaniu elementu w panelu po prawej stronie pojawi się jego nazwa, wartość oraz zapisane heksadecymalnie wartości bajtów, które odpowiadają temu wpisowi w deskryptorze raportu. Wartość elementu można zmienić klikając dwukrotnie na odpowiadający mu wpis w panelu po prawej. Po zmianie wartości element pozostaje zaznaczony. Jeśli w panelu po prawej zaznaczony jest element, to kolejne dodawane elementy zostają umieszczone powyżej niego. Aby powrócić do dodawania nowych elementów na końcu deskryptora należy wybrać Edit → Insert after last item.

Po utworzeniu deskryptora należy go zapisać. Należy przy tym zwrócić uwagę na wybranie z listy odpowiedniego typu pliku, ponieważ od tego zależy format wygenerowanych danych. Jeśli wybrany jest plik typu HID Descriptor File (*.hid), zapisany zostanie plik rozpoznawany przez program Dt.exe, natomiast wybranie Header File (.h) spowoduje wygenerowanie pliku nagłówkowego w języku C, z definicją tablicy bajtów, który można włączyć do tworzonego projektu lub skopiować jego zawartość i wkleić w odpowiednim miejscu w jednym z istniejących plików projektu. Rozszerzenie pliku wpisane w polu Nazwa pliku nie ma wpływu na format generowanych danych.

Format deskryptora raportu

W przeciwieństwie do innych deskryptorów USB deskryptor raportu nie ma ściśle określonego formatu ani długości. Jego rozmiar zależy od tego, ile różnych rodzajów raportów jest przesyłanych z urządzenia do komputera i z ilu pól się one składają – wszystkie pola muszą być odpowiednio opisane w deskryptorze.

Deskryptor raportu składa się z elementów (tzw. item). Najważniejsze są elementy Main (oprócz nich istnieją również elementy Global i Local) i tylko one zostaną tu opisane. Istnieje kilka rodzajów elementów Main:

- **INPUT** (wejście) – reprezentuje jedną lub kilka podobnych wartości sterujących, np. reprezentujących położenie na osiach x, y, z lub stan jednego lub kilku przycisków.
- **OUTPUT** (wyjście) – reprezentuje jedną lub kilka wartości sterujących stanem urządzenia, np. świeceniem LED.
- **FEATURE** (funkcja) – definiują opcje konfiguracyjne urządzenia, nieprzeznaczone bezpośrednio dla użytkownika (w przeciwieństwie do elementów Output).
- **COLLECTION** (kolekcja) – zgrupowanie elementów trzech powyższych typów mające określone znaczenie np. reprezentujące pewien punkt przestrzeni (kolekcja typu Physical) czy reprezentujące określony element sterujący np. wskaźnik, joystick, klawiaturę (kolekcja typu Application). Urządzenie łączące kilka funkcji np. klawiatura z wbudowanym touchpadem może dla każdej z funkcji składowych zdefiniować oddzielną kolekcję.
- **END COLLECTION** – znacznik końca kolekcji.

Deskryptor musi mieć przynajmniej jedną kolekcję typu Application, która obejmuje wszystkie pozostałe elementy deskryptora. Kolekcje mogą być w sobie zagnieżdżone. Wszystkie kolekcje oprócz wspomnianej kolekcji najwyższego poziomu są opcjonalne.

W deskryptorze raportu opisane są wszystkie pola danych przesyłane w raportach. Dla każdej grupy danych reprezentowanej przez pojedynczy element raportu typu Input, Output lub Feature należy określić:

- **USAGE** – określa znaczenie danej. Jest to 32 bitowe pole złożone z dwóch pól 16 -bitowych określających „stronę" Usage Page i „identyfikator" Usage ID (w HID Descriptor Tool oznaczone USAGE). W zależności od wybranej Usage Page zmienia się znaczenie Usage ID odpowiadających poszczególnym wzorcom bitowym. Po wybraniu określonej Usage Page można używać kolejnych Usage ID z tej strony bez ponawiania 16 bitów określających Usage Page.
- **LOGICAL MINIMUM, LOGICAL MAXIMUM** – określają przedział wartości liczbowych, jakie może przyjmować określone pole. Jeśli w raporcie pojawi się wartość spoza tego przedziału, to zostanie ona zignorowana.
- **REPORT SIZE** – rozmiar pojedynczego pola wyrażony w bitach.
- **REPORT COUNT** – liczba pól.
- **INPUT/OUTPUT/FEATURE** – znacznik powodujący wstawienie pola do raportu, jak również określający własności pola poprzez znaczniki bitowe, m.in.:
  - **DATA/CONSTANT** – określa, czy pole zawiera wartość zmienną czy stałą. Stałych używa się typowo do wyrównywania danych do granicy bajtu, słowa, itp.
  - **ARRAY/VARIABLE** – określa, czy jest to standardowa zmienna czy wartość tablicowa. W przypadku wartości tablicowych w polu podaje się indeks elementu określonej tablicy – np. numer przycisku na klawiaturze (numery przypisane poszczególnym przyciskom są zdefiniowane w odpowiednich tablicach, które można znaleźć w dokumentacji klasy HID).
  - **ABSOLUTE/RELATIVE** – określa, czy jest to wartość względna czy bezwzględna.

Jeśli poszczególne pola danych w obrębie jednego elementu Item mają różne wartości Usage, można określić je poprzez kolejne pola Usage: np.

USAGE(Button 1)
USAGE(Button 2)
USAGE(Button 3)

albo używając *Usage Minimum* i *Usage Maximum*, jeśli są to kolejne wartości Usage, a więc zamiast powyższej sekwencji można napisać:

USAGE_MINIMUM(Button 1)
USAGE_MAXIMUM(Button 3)

Urządzenie może wysyłać i odbierać więcej niż jeden rodzaj raportu. Poszczególne raporty mogą różnić się strukturą i są identyfikowane przez pole Report ID umieszczone na początku raportu. W przypadku, gdy w deskryptorze raportu nie ma żadnego pola Report ID, przyjmuje się, że występuje tylko jeden raport określonego

typu (wejściowy, wyjściowy). W takim wypadku odbierane i wysyłane raporty nie mają pola Report ID.

Dalej opisano przykładowe deskryptory raportów dla myszy i klawiatury obsługiwanych przez podsystem BIOS komputera PC.

Deskryptor raportu myszy

Deskryptor raportu standardowej myszy przedstawiono na **listingu 1**. Raport dla myszy składa się z trzech (REPORT_COUNT (3)) znaczników bitowych (REPORT_SIZE (1)) określających stan przycisków 1…3 (USAGE_MINIMUM (Button 1), USAGE_MAXIMUM (Button 3)), które mogą przyjmować wartość 0 lub 1 (LOGICAL_MINIMUM (0), LOGICAL_MAXIMUM (1)). Ponieważ zdefiniowane są one jako wartości bezwzględne, wartość 1 oznacza przycisk wciśnięty, a 0 zwolniony. W przypadku wartości względnych 1 oznaczałoby zmianę stanu, a 0 jej brak. Następnie występuje 5 (REPORT_COUNT (5)) stałych bitów (REPORT_SIZE (1)) zapewniających wyrównanie do granicy bajtu. Ważne jest, aby takie stałe definiować jako (Constant, Array, Absolute), co odpowiada wartości szesnastkowej 0x01. W przeciwnym wypadku urządzenie może nie być poprawnie rozpoznawane w systemie. Kolejne elementy raportu myszy – to 3 (REPORT_COUNT (3)) wartości przesunięć: wzdłuż osi x (USAGE (X)), wzdłuż osi y (USAGE (Y)) i przesunięcie koła myszy (USAGE (Wheel)), reprezentowane w postaci liczb 8-bitowych (REPORT_SIZE (8)), mogące przyjmować wartości od –127 (LOGICAL_MINIMUM (–127)) do +127 (LOGICAL_MAXIMUM (127)). Ponieważ są to zmiany położenia, więc reprezentowane są jako wartości względne.

Listing 1. Deskryptor raportu myszy

```
char ReportDescriptor[52] = {
    0x05, 0x01, //USAGE_PAGE (Generic Desktop)
    0x09, 0x02, //USAGE (Mouse)
    0xa1, 0x01, //COLLECTION (Application)
    0x09, 0x01, //   USAGE (Pointer)
    0xa1, 0x00, //   COLLECTION (Physical)
    0x05, 0x09, //     USAGE_PAGE (Button)
    0x19, 0x01, //     USAGE_MINIMUM (Button 1)
    0x29, 0x03, //     USAGE_MAXIMUM (Button 3)
    0x15, 0x00, //     LOGICAL_MINIMUM (0)
    0x25, 0x01, //     LOGICAL_MAXIMUM (1)
    0x95, 0x03, //     REPORT_COUNT (3)
    0x75, 0x01, //     REPORT_SIZE (1)
    0x81, 0x02, //     INPUT (Data,Var,Abs)
    0x95, 0x01, //     REPORT_COUNT (1)
    0x75, 0x05, //     REPORT_SIZE (5)
    0x81, 0x01, //     INPUT (Cnst,Ary,Abs)
    0x05, 0x01, //     USAGE_PAGE (Generic Desktop)
    0x09, 0x30, //     USAGE (X)
    0x09, 0x31, //     USAGE (Y)
    0x09, 0x38, //     USAGE (Wheel)
    0x15, 0x81, //     LOGICAL_MINIMUM (-127)
    0x25, 0x7f, //     LOGICAL_MAXIMUM (127)
    0x95, 0x03, //     REPORT_COUNT (3)
    0x75, 0x08, //     REPORT_SIZE (8)
    0x81, 0x06, //     INPUT (Data,Var,Rel)
    0xc0,       //   END_COLLECTION
    0xc0        //END_COLLECTION
};
```

Listing 2. Deskryptor raportu klawiatury

```
char ReportDescriptor[61] = {
    0x05, 0x01, //USAGE_PAGE (Generic Desktop)
    0x09, 0x06, //USAGE (Keyboard)
    0xa1, 0x01, //COLLECTION (Application)
    0x05, 0x07, //   USAGE_PAGE (Keyboard)
    0x19, 0xe0, //   USAGE_MINIMUM (Keyboard LeftControl)
    0x29, 0xe7, //   USAGE_MAXIMUM (Keyboard Right GUI)
    0x15, 0x00, //   LOGICAL_MINIMUM (0)
    0x25, 0x01, //   LOGICAL_MAXIMUM (1)
    0x95, 0x08, //   REPORT_COUNT (8)
    0x75, 0x01, //   REPORT_SIZE (1)
    0x81, 0x02, //   INPUT (Data,Var,Abs)
    0x95, 0x01, //   REPORT_COUNT (1)
    0x75, 0x08, //   REPORT_SIZE (8)
    0x81, 0x01, //   INPUT (Cnst,Ary,Abs)
    0x19, 0x00, //   USAGE_MINIMUM (Reserved (no event indi-
cated))
    0x29, 0x65, //   USAGE_MAXIMUM (Keyboard Application)
    0x15, 0x00, //   LOGICAL_MINIMUM (0)
    0x25, 0x65, //   LOGICAL_MAXIMUM (101)
    0x95, 0x06, //   REPORT_COUNT (6)
    0x75, 0x08, //   REPORT_SIZE (8)
    0x81, 0x00, //   INPUT (Data,Ary,Abs)
    0x05, 0x08, //   USAGE_PAGE (LEDs)
    0x19, 0x01, //   USAGE_MINIMUM (Num Lock)
    0x29, 0x05, //   USAGE_MAXIMUM (Kana)
    0x95, 0x05, //   REPORT_COUNT (5)
    0x75, 0x01, //   REPORT_SIZE (1)
    0x91, 0x02, //   OUTPUT (Data,Var,Abs)
    0x95, 0x01, //   REPORT_COUNT (1)
    0x75, 0x03, //   REPORT_SIZE (3)
    0x91, 0x01, //   OUTPUT (Cnst,Ary,Abs)
    0xc0        //END_COLLECTION
};
```

Deskryptor raportu klawiatury

Listing 2 zawiera deskryptor raportu standardowej klawiatury. Deskryptor opisuje dwa raporty – wejściowy, zawierający informację o stanie klawiatury, i wyjściowy, służący do sterowania wskaźnikami klawiatury przez komputer.

Dla typowej klawiatury komputera zawierającej co najmniej j 103 klawisze określa się USAGE(KEYBOARD), natomiast dla klawiatur nie spełniających tego wymogu – USAGE(KEYPAD).

Raport dla standardowej klawiatury obsługiwanej przez podsystem BIOS składa się z trzech części:

1. Bajtu stanu klawiszy modyfikatorów (Ctrl, Shift i podobnych).
2. Bajtu pustego (zarezerwowanego).
3. 6 bajtów zawierających numery aktualnie wciśniętych klawiszy.

Bajt *modifier* zawiera 8 (REPORT_COUNT (8)) znaczników bitowych (REPORT_SIZE (1)), z których każdy określa stan (LOGICAL_MINIMUM (0), LOGICAL_MAXIMUM (1)) jednego z przycisków specjalnych (Shift, Alt, Ctrl, GUI). Znaczenie poszczególnych bitów jest następujące:

- bit 0 – lewy Control,
- bit 1 – lewy Shift,
- bit 2 – lewy Alt,
- bit 3 – lewy GUI (Win/Apple/Meta key),
- bit 4 – prawy Control,
- bit 5 – prawy Shift,
- bit 6 – prawy Alt,
- bit 7 – prawy GUI.

Za polem modifier występuje jeden (REPORT_COUNT (1)) zarezerwowany bajt (REPORT_SIZE (8)).

Kolejne 6 (REPORT_COUNT (6)) bajtów (REPORT_SIZE (8)) może zawierać kod jednego z wciśniętych przycisków (USAGE_MAXIMUM (Keyboard Application)) lub wartość zarezerwowaną 0 (USAGE_MINIMUM (Reserved (no event indicated))) oznaczającą, że nie wciśnięto przycisku. Wartości przesyłane w tych polach (LOGICAL_MINIMUM (0), LOGICAL_MAXIMUM (101)) odpowiadają wartościom Usage, co można zauważyć analizując wartości szesnastkowe (0x65 = 101). Zalecane jest, aby również dla niestandardowych klawiatur zachować konwencję, że przesyłane wartości są zgodne z wartościami Usage dla odpowiednich przycisków. Numery przycisków używane w klawiaturach USB HID nie są ani kodami znaków ASCII, ani numerami stosowanymi w klawiaturach z interfejsem PS/2; muszą one być translowane przez oprogramowanie na kody znaków.

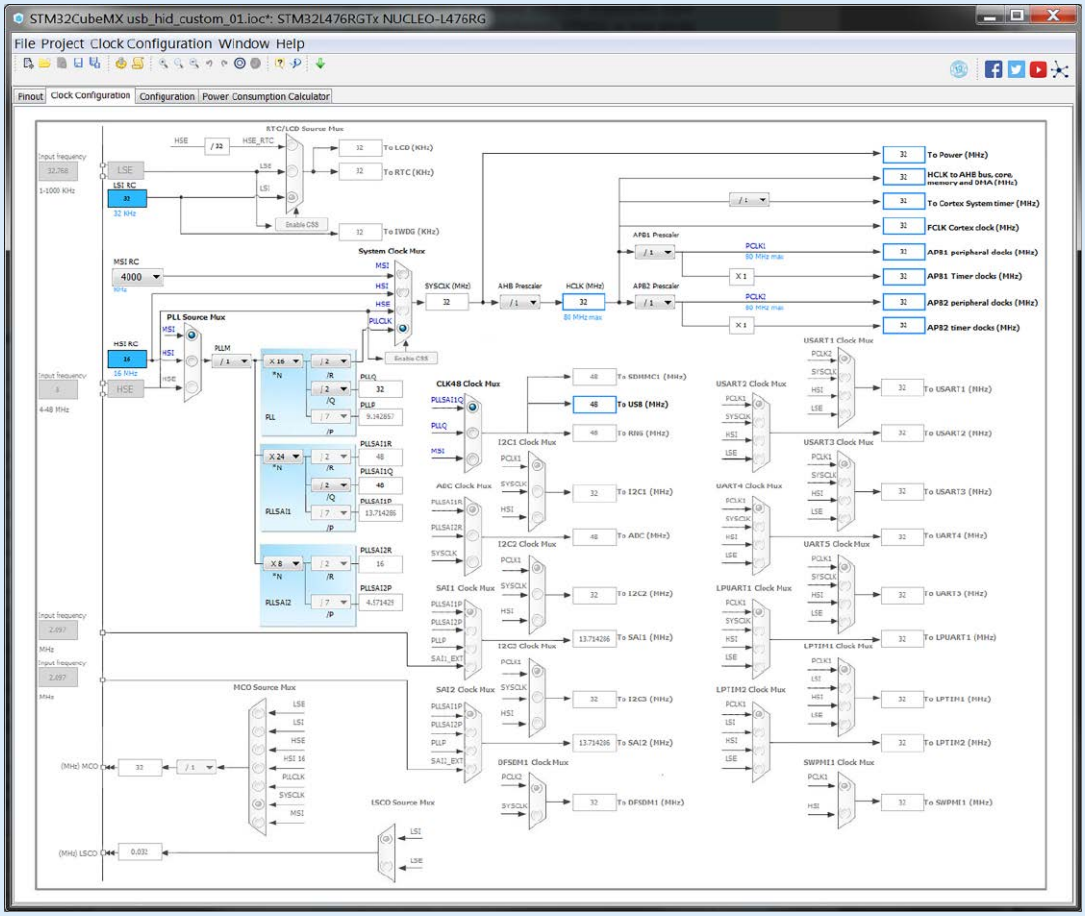
Dla standardowej klawiatury definiuje się również raport wyjściowy, który steruje stanem świecenia umieszczonych zwykle na klawiaturze diod LED sygnalizujących stan klawiatury (np. Caps Lock). Standardowo jest to pięć (REPORT_COUNT (5)) znaczników bitowych (REPORT_SIZE (1)) odpowiadających następującym funkcjom (USAGE_MINIMUM (Num Lock), USAGE_MAXIMUM (Kana)):

- bit 0 – Num Lock,
- bit 1 – Caps Lock,
- bit 2 – Scroll Lock,
- bit 3 – Compose,
- bit 4 – Kana.

Raport wyjściowy jest wyrównany do pełnego bajtu przez umieszczenie trzech (REPORT_COUNT (1)) dodatkowych bitów, których wartości są nieistotne.

Urządzenie HID w CubeMX

Poniżej przedstawimy dwa przykładowe projekty urządzeń HID: myszy i klawiatury. Generowanie projektu dla obu urządzeń w programie CubeMX przebiega podobnie i różni się wyborem jednej opcji,



Rysunek 2. Zakładka konfiguracji taktowania

co zaznaczono w dalszym opisie. Przystępując do tworzenia urządzenia USB HID, należy po uruchomieniu programu CubeMX, wybrać na ekranie głównym lub z menu File opcję New Project. W oknie, które się pojawi należy wybrać używany mikrokontroler lub płytkę – w przykładzie będzie to płytka Nucleo64 z mikrokontrolerem STM32L476. Na płytce Nucleo64 nie ma złącza interfejsu USB mikrokontrolera L476. Podłączenie złącza USB wymaga połączenia linii D+ i D- odpowiednio z wyprowadzeniami PA12 i PA11 mikrokontrolera dostępnymi na złączu ST Morpho płytki Nucleo. Można w tym celu użyć np. modułu z gniazdem USB przeznaczonego dla płytek stykowych. Zamiast płytki serii Nucleo-64 można użyć jednej z płytek uruchomieniowych wyposażonych w złącze USB, np. serii Discovery lub Nucleo-144.

Po odnalezieniu na liście prezentowanej przez CubeMX odpowiedniej płytki lub mikrokontrolera, należy dwukrotnie kliknąć na wybranym elemencie, co spowoduje utworzenie projektu i przełączenie do okna konfiguracji projektu.

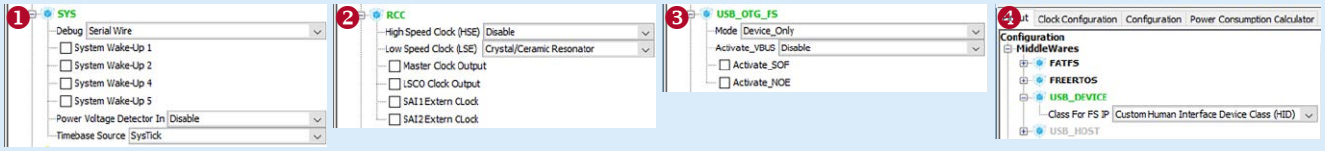
Taktowanie mikrokontrolera

Interfejs USB wymaga taktowania stałą częstotliwością, przy zachowaniu dokładności nie gorszej niż 0.5%. Istotna jest więc poprawna konfiguracja taktowania mikrokontrolera. Peryferiale USB stosowane w mikrokontrolerach STM32 wymagają taktowania częstotliwością 48 MHz, która może być różna od częstotliwości taktowania rdzenia mikrokontrolera. Poszczególne serie rodziny STM32 różnią się istotnie możliwościami generowania przebiegów zegarowych, w tym możliwymi sposobami uzyskania przebiegu zegarowego dla peryferiala USB. W większości płytek serii Discovery i Nucleo do mikrokontrolera jest doprowadzony przebieg zegarowy o częstotliwości 8 MHz, generowany przez interfejs ST-Link. Nie dotyczy to jednak płytki NUCLEO-L476RG. Najprostszym sposobem na uzyskanie precyzyjnego taktowania mikrokontrolera na tej płytce jest użycie wewnętrznego generatora MSI o niskiej precyzji, synchronizowanego generatorem LSE, korzystającym z oscylatora „zegarkowego" 32768 Hz. Poniżej zostanie opisany sposób uzyskania takiej właśnie konfiguracji taktowania.

Wybór modułów – zakładka Pinout

Po wybraniu w CubeMX typu mikrokontrolera lub płytki, zostaje uaktywniona zakładka Pinout. W widocznym po lewej stronie panelu konfiguracji peryferiów i oprogramowania należy wybrać następujące opcje (**rysunek 2**):

- Moduł SYS – Debug: Serial-Wire (w celu zapewnienia możliwości debugowania oprogramowania i ponownego programowania mikrokontrolera) ❶.
- Moduł RCC – LSE: wybrać tryb Crystal/Ceramic Resonator ❷.
- Moduł USB_OTG_FS – Mode: Device Only, Activate_VBUS: Disable ❸.
- Oprogramowanie (MiddleWares) – USB_DEVICE – dla myszy należy wybrać Class For FS IP: Human Interface Device Class (HID) (gotowy przykład z myszą, bez wyjściowego endpointu notyfikacji), natomiast dla klawiatury – Class For FS IP Custom Human Interface Device Class (HID) (szkielet z pustym deskryptorem raportu, z wyjściowym endpointem notyfikacji) ❹.



Konfigurowanie taktowania – zakładka Clock Configuration

Następnie należy przejść do zakładki Clock Configuration. Pojawi się okno z pytaniem, czy CubeMX powinien automatycznie przygotować konfigurację taktowania mikrokontrolera. Po potwierdzeniu, moduły związane z generowaniem przebiegów zegarowych powinny zostać skonfigurowane (rysunek 3). Funkcja ta działa poprawnie w wersji 4.23 CubeMX; wcześniejsze wersje mogą generować błędny kod konfiguracji taktowania.

W przypadku użycia mikrokontrolera z serii o uboższych możliwościach konfiguracji taktowania, należy ustawić częstotliwość taktowania modułu USB na 48 MHz i dobrać do niej częstotliwość taktowania rdzenia. W starszych modelach mikrokontrolerów (np. w serii STM32F4) w celu zapewnienia stabilnego przebiegu zegarowego dla modułu USB konieczne jest włączenie zewnętrznego oscylatora HSE.

Konfigurowanie modułów – zakładka Configuration

Kolejnym etapem jest edycja ustawień w zakładce Configuration:

W celu zapewnienia synchronizacji generatora MSI na podstawie generatora LSE, należy wejść w ustawienia modułu RCC i włączyć opcję MSI Auto Calibration.

W panelu Connectivity, w ustawieniach modułu USB_FS ustawić parametr VBUS sensing na Disabled. Domyślne ustawienie tej opcji w dotychczasowych wersjach CubeMX powoduje niemożność uruchomienia urządzenia USB bez połączenia linii VBUS.

Generowanie kodu

Następnie możemy przejść do generowania kodu projektu. W tym celu należy wybrać z menu opcję Project, a następnie Generate Code. W oknie, które się pojawi, w zakładce Project należy uzupełnić pole Project Name wpisując nazwę projektu, wybrać lokalizację plików projektu oraz środowisko programowania – w przykładzie jest to Keil MDK-ARM V5. Warto również zwiększyć rozmiar sterty i stosu – w przypadku programu przykładowego działa on prawidłowo z domyślnymi wartościami, jednak mając na uwadze potencjalną rozbudowę projektu oraz fakt, że zbyt mały rozmiar stosu jest częstą przyczyną niedziałania urządzania, warto od razu zmienić te wartości. Zaznaczenie opcji Copy only necessary library files w zakładce Code Generator spowoduje, że do katalogu projektu skopiowane zostaną tylko pliki bibliotek używanych w projekcie, co znacząco ograniczy zajętość pamięci masowej przez pliki projektu, natomiast opcja Set all free pins as analog umożliwia ograniczenie poboru mocy i emisji zakłóceń przez mikrokontroler poprzez wyłączenie cyfrowych bloków wejścia-wyjścia na nieużywanych wyprowadzeniach układu. Po wygenerowaniu pierwszej wersji projektu, opisane wyżej okno ustawień dostępne jest przez wybranie Project → Settings.

Po kliknięciu OK, o ile wcześniej nie został zainstalowany w CubeMX pakiet obsługi wybranego typu mikrokontrolera, nastąpi jego ściągnięcie i zainstalowanie. Po pomyślnym wygenerowaniu projektu CubeMX zaproponuje otwarcie go w wybranym środowisku programowania.

Uzupełnianie projektów wygenerowanych w CubeMX

Wygenerowany szkielet projektu jest niekompletny. W celu uzyskania założonej funkconalnści oprogramowania należy zmodyfikować niektóre pliki źródłowe. Pliki przeznaczone do modyfikacji przez użytkownika są umieszczone w wirtualnym folderze projektu noszącym w środowisku MDK-ARM nazwę Application/User. Modyfikacje zawartości plików źródłowych należy umieszczać między znacznikami /* USER CODE BEGIN … */ i /* USER CODE END … */, dzięki czemu w przypadku dokonania zmian w projekcie i ponownej generacji kodu w CubeMX nie zostaną one usunięte. Niestety w praktyce niezbędne staje się uzupełnianie plików, których modyfikacji przez użytkownika nie przewiduje CubeMX. Dotyczy to większości projektów urządzeń USB.

Ważną cechą oprogramowania USB generowanego przez CubeMX jest brak wielobieżności. Ponieważ oprogramowanie stosu USB działa w przerwaniu od interfejsu USB, oznacza to w praktyce, że procedury inicjujące np. transmisję danych do komputera muszą być wywoływane na poziomie priorytetowym równym poziomowi przerwania USB, a więc albo z samego przerwania USB, albo z przerwania o takim samym priorytecie.

Ponieważ z kolei sama obsługa żądań protokołu urządzenia USB, następująca w przerwaniach USB, jest czasochłonna – warto na etapie konfiguracji projektu w CubeMX obniżyć priorytet przerwania USB, tak, by było ono wywłaszczane przez inne, pilniejsze w obsłudze przerwania.

Projekt myszy

Projekt standardowego urządzenia HID, wygenerowany przez CubeMX, zawiera już zdefiniowany deskryptor raportu myszy. Zarówno deskryptor raportu jak i inne deskryptory oraz główne funkcje odpowiadające za obsługę urządzenia HID znajdują się w pliku usb_hid.c. Warto zwrócić uwagę na pola Class, Subclass oraz Protocol deskryptora interfejsu w tablicy USBD_HID_CfgDesc:

```
  0x03,          /*bInterfaceClass: HID*/
  0x01,          /*bInterfaceSubClass : 1=BOOT,
0=no boot*/
  0x02,          /*nInterfaceProtocol : 0=none,
1=keyboard, 2=mouse*/
```

Zgodnie z wartościami tych pól urządzenie to mysz obsługiwana przez podsystem BIOS.

W pliku usbd_hid.c znajduje się definicja funkcji *uint8_t USBD_HID_SendReport(USBD_HandleTypeDef *pdev, uint8_t *report, uint16_t len)* służącej do przesyłania raportu do komputera nadrzędnego. Wywołanie tej funkcji powoduje zainicjowanie transmisji raportu, o ile interfejs USB jest do tego gotowy; w przeciwnym razie funkcja zwraca kod błędu bez oczekiwania na gotowość.

W celu uzyskania minimalnej funkcjonalności, należy dodać do projektu generowanie treści raportu i wysyłanie go. W programie przykładowym ilustrującym działanie myszy HID, po kolejnych naciśnięciach przycisku umieszczonego na płytce Nucleo będą wysyłane następujące raporty (jeden raport przy jednym naciśnięciu przycisku):

- przesunięcie myszy w prawo – (0, 100, 0, 0),
- przesunięcie myszy w dół – (0, 0, 100, 0),
- przesunięcie myszy w lewo i w górę – (0, –100, –100, 0).

Po trzykrotnym naciśnięciu przycisku, kursor myszy powinien więc zakreślić na ekranie trójkąt prostokątny i powrócić do punktu wyjścia.

Z powodu opisanych powyżej cech stosu USB z CubeMX, wysyłanie danych przez interfejs USB powinno być wykonywane w przerwaniu o priorytecie równym priorytetowi przerwania USB. Na potrzeby prostego programu demonstracyjnego można do tego celu użyć przerwania SysTick, które jest konfigurowane przez CubeMX i domyślnie ma priorytet taki sam jak przerwanie USB. W praktyce, w przypadku większych projektów, gdzie dodatkowo obsługiwane są inne zdarzenia, przerwanie USB powinno mieć jeden z najniższych priorytetów, natomiast przerwanie SysTick powinno mieć priorytet nie niższy od przerwania USB.

Timer SysTick generuje przerwania z częstotliwością 1 kHz. Biblioteka HAL umożliwia wywołanie z przerwania SysTick procedury użytkownika o nazwie HAL_SYSTICK_Callback(). Procedurę tę możemy umieścić np. w pliku main.c. Procedura przy co dziesiątym wywołaniu będzie testowała stan przycisku, a przy wykryciu jego naciśnięcia – wysyłała raport myszy o następującej budowie:

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

Listing 3. Zmiany wprowadzone w pliku main.c w projekcie myszy

```
/* USER CODE BEGIN 0 */
#include "stm3214yy.h"
#include "stm32nucleo64.h"
#include "usbd_hid.h"

struct mouse_report {
    uint8_t buttons;
    int8_t x;
    int8_t y;
    int8_t wheel;
};

void HAL_SYSTICK_Callback(void)
{
static uint8_t tdiv;
    if (++tdiv == 10)
    {
        tdiv = 0;
        static const struct mouse_report mrep[3] = {
            {0, 100, 0, 0},
            {0, 0, 100, 0},
            {0, -100, -100, 0}
        };
        static uint8_t phase = 0;
        static uint8_t khist;
        if ((khist = (khist << 1 | BTN_DOWN) & 3) == 1)
        {
            USBD_HID_SendReport(&hUsbDeviceFS, (uint8_t*)&m-
rep[phase], sizeof(struct mouse_report));
            if (++phase == sizeof(mrep) / sizeof(mrep[0])) phase
= 0;
        }
    }
}
/* USER CODE END 0 */
```

```
struct mouse _ report {
    uint8 _ t buttons;
    int8 _ t x;
    int8 _ t y;
    int8 _ t wheel;
};
```

Zgodnie z opisanym wyżej deskryptorem raportu dla myszy, raport składa się z jednego bajtu zawierającego znaczniki bitowe odpowiadające poszczególnym przyciskom myszy zdefiniowanego jako liczba 8-bitowa bez znaku oraz trzech bajtów określających przemieszczenia wzdłuż poszczególnych osi i przesunięcie koła myszy zdefiniowanych jako 8-bitowe liczby ze znakiem (zgodnie z deskryptorem raportu mogą one przyjmować wartości od –127 do 127).

Rozwiązanie takie nie jest idealne, ale umożliwia ono poprawną prezentację działania urządzenia HID. W praktyce przygotowanie raportu powinno następować po przesłaniu poprzedniego, co jednak wymaga wprowadzenia większych modyfikacji w oprogramowaniu generowanym przez CubeMX.

Wszystkie wprowadzone modyfikacje przedstawiono na **listingu 3**. Wysyłane raporty zgrupowano w tablicy. Przy każdym wykryciu naciśnięcia przycisku na płytce Nucleo wysłany zostaje raport oraz jest inkrementowana (modulo 3) zmienna indeksująca tablicę raportów.

Po skompilowaniu programu i zaprogramowaniu mikrokontrolera (oraz zresetowaniu mikrokontrolera, jeśli w opcjach debugowania nie została zaznaczona opcja Reset and Run), urządzenie powinno zostać rozpoznane przez system operacyjny, a każde przyciśnięcie niebieskiego przycisku na płytce Nucleo powinno spowodować przesunięcie kursora.

Projekt klawiatury USB HID

Generowanie projektu klawiatury w CubeMX przebiega podobnie jak w przypadku myszy. Jedyna różnica polega na wybraniu innej klasy urządzenia – w oknie konfiguracji jako Class for FS IP wybieramy Custom Human Interface Device.

Następnie powtarzamy czynności związane z konfiguracją taktowania i interfejsu USB opisane przy projekcie myszy. Dodatkowo musimy w zakładce konfiguracji – Middlewares, USB-DEVICE, Parameter Settings ustawić wartość stałej USBD_CUSTOM_HID_REPORT_DESC_SIZE, określającej rozmiar deskryptora raportu, na 61.

Raport stanu klawiatury w naszym projekcie ma budowę pokazaną na **listingu 4**. Składa się on z ośmiu bajtów. Pierwszy bajt zawiera znaczniki stanu klawiszy typu Shift; każdy bit reprezentuje stan jednego klawisza. Drugi bajt jest pusty. Sześć kolejnych bajtów może zawierać kody (numery) naciśniętych klawiszy. Wartość 0 w tych bajtach oznacza brak naciśniętego klawisza. Jeśli wszystkie klawisze są zwolnione, przesyłanych jest 6 bajtów o wartości 0.

Deskryptor raportu oraz funkcje zapewniające interakcję programu z urządzeniem USB HID są zawarte w pliku usbd_custom_hid_if.c. Znajduje się w nim tablica CUSTOM_HID_ReportDesc_FS, do której należy skopiować zawartość pliku wygenerowanego przez program HID Descriptor Tool (wybierając w nim opcję zapisu w posatci pliku nagłówkowego języka C). Gotowy deskryptor raportu klawiatury przedstawiono na listingu 2.

Należy również zmodyfikować makro USBD_CUSTOM_HID_REPORT_DESC_SIZE nadając mu wartość 61.

W pliku usbd_custom_hid_if.c znajduje się również funkcja CUSTOM_HID_OutEvent_FS, wywoływana z procedury obsługi przerwania interfejsu USB po otrzymaniu raportu wyjściowego od hosta. Raport ten służy do sterowania wskaźnikami LED umieszczonymi na klawiaturze. W programie przykładowym zielona dioda dostępna na płytce Nucleo będzie sygnalizować stan klawisza Caps Lock. Domyślnie funkcja CUSTOM_HID_OutEvent_FS przyjmuje 2 argumenty typu uint8_t: event_idx oraz state. Po przeanalizowaniu funkcji obsługujących klasę HID, w szczególności funkcji USBD_CUSTOM_HID_DataOut zdefiniowanej w pliku usbd_customhid.c, widać, że argumenty te to dwa pierwsze bajty deskryptora raportu. W celu zapewnienia bardziej uniwersalnej obsługi raportów wyjściowych należy zmodyfikować funkcje CUSTOM_HID_OutEvent_FS oraz USBD_CUSTOM_HID_DataOut, tak aby druga z nich jako argumenty przyjmowała wskaźnik na bufor z raportem oraz jego długość. Na potrzeby programu demonstracyjnego nie jest to jednak konieczne – wystarczy prawidłowo zinterpretować przekazywane argumenty. Ponieważ w przypadku standardowej klawiatury raport ma tylko

Listing 4. Zmodyfikowany deskryptor urządzenia w pliku usbd_custom_hid_if.c

```
__ALIGN_BEGIN static uint8_t CUSTOM_HID_ReportDesc_FS[USBD_CUS-
TOM_HID_REPORT_DESC_SIZE] __ALIGN_END =
{
/* USER CODE BEGIN 0 */
    0x05, 0x01, //USAGE_PAGE (Generic Desktop)
    0x09, 0x06, //USAGE (Keyboard)
    0xa1, 0x01, //COLLECTION (Application)
    0x05, 0x07, //USAGE_PAGE (Keyboard)
    0x19, 0xe0, //USAGE_MINIMUM (Keyboard LeftControl)
    0x29, 0xe7, //USAGE_MAXIMUM (Keyboard Right GUI)
    0x15, 0x00, //LOGICAL_MINIMUM (0)
    0x25, 0x01, //LOGICAL_MAXIMUM (1)
    0x95, 0x08, //REPORT_COUNT (8)
    0x75, 0x01, //REPORT_SIZE (1)
    0x81, 0x02, //INPUT (Data,Var,Abs)
    0x95, 0x01, //REPORT_COUNT (1)
    0x75, 0x08, //REPORT_SIZE (8)
    0x81, 0x01, //INPUT (Cnst,Ary,Abs)
    0x19, 0x00, //USAGE_MINIMUM (Reserved (no event indicated))
    0x29, 0x65, //USAGE_MAXIMUM (Keyboard Application)
    0x15, 0x00, //LOGICAL_MINIMUM (0)
    0x25, 0x65, //LOGICAL_MAXIMUM (101)
    0x95, 0x06, //REPORT_COUNT (6)
    0x75, 0x08, //REPORT_SIZE (8)
    0x81, 0x00, //INPUT (Data,Ary,Abs)
    0x05, 0x08, //USAGE_PAGE (LEDs)
    0x19, 0x01, //USAGE_MINIMUM (Num Lock)
    0x29, 0x05, //USAGE_MAXIMUM (Kana)
    0x95, 0x05, //REPORT_COUNT (5)
    0x75, 0x01, //REPORT_SIZE (1)
    0x91, 0x02, //OUTPUT (Data,Var,Abs)
    0x95, 0x01, //REPORT_COUNT (1)
    0x75, 0x03, //REPORT_SIZE (3)
    0x91, 0x01, //OUTPUT (Cnst,Ary,Abs)
    //0xc0
    // END_COLLECTION
/* USER CODE END 0 */
    0xC0    /* END_COLLECTION */
};
```

Listing 5. Funkcja CUSTOM_HID_OutEvent_FS interpretująca raport wyjściowy

```
static int8_t CUSTOM_HID_OutEvent_FS  (uint8_t event_idx,
uint8_t state)
{
/* USER CODE BEGIN 6 */
    if (event_idx & 2) LED_PORT->BSRR = LED_MSK;
    else LED_PORT->BRR = LED_MSK;
    return 0;
/* USER CODE END 6 */
}
```

jeden bajt, zawierający stan wskaźników, drugi argument funkcji CUSTOM_HID_OutEvent_FS należy zignorować. Stan diody Caps Lock jest zapisany na bicie nr 1. Zmodyfikowana funkcja CUSTOM_HID_OutEvent_FS przedstawiona jest na **listingu 5**.

Listing 6. Zmiany wprowadzone w pliku usbd_custom_hid_if.c w projekcie klawiatury

```
/* USER CODE BEGIN 0 */
#include "stm3214yy.h"
#include "stm32nucleo64.h"
#define KEY_MOD_LCTRL           0x01
#define KEY_MOD_LSHIFT          0x02
#define KEY_MOD_LALT            0x04
#define KEY_MOD_LMETA           0x08
#define KEY_MOD_RCTRL           0x10
#define KEY_MOD_RSHIFT          0x20
#define KEY_MOD_RALT            0x40
#define KEY_MOD_RMETA           0x80
#define KEY_CAPSLOCK            0x39
#define KEY_LET(v)              (v - ‚A’ + 4) // A key - code 4

struct kbd_report {
    uint8_t modifier;
    uint8_t reserved;
    uint8_t key[6];
};
void HAL_SYSTICK_Callback(void)
{
    static const struct kbd_report kreps[5] = {
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, KEY_LET(‚A’), 0, 0, 0, 0, 0},
        {KEY_MOD_LSHIFT, 0, KEY_LET(‚B’), 0, 0, 0, 0, 0},
        {KEY_MOD_RALT, 0, KEY_LET(‚C’), 0, 0, 0, 0, 0},
        {0, 0, KEY_CAPSLOCK, 0, 0, 0, 0, 0},
    };
    static uint8_t phase = 5;
    static uint8_t khist;
    static uint8_t tdiv;
    ++ tdiv;
    if (tdiv % 10 == 0
        && phase == 5
        && (khist = (khist << 1 | BTN_DOWN) & 3) == 1))
        {
            phase = 0;
        }
        if (tdiv == 100)
        {
            tdiv = 0;
            USBD_CUSTOM_HID_SendReport_
FS((uint8_t*)&kreps[phase], sizeof(struct kbd_report));
            if (phase < 5) ++phase;
        }
    }
}
/* USER CODE END 0 */
```

Kolejna modyfikacja niezbędna do działania naszego urządzenia polega na odkomentowaniu definicji funkcji USBD_CUSTOM_HID_SendReport_FS. Następnie do pliku usbd_custom_hid_if.c należy dopisać procedurę obsługi przerwanie timera, podobną do tej z projektu myszy. Po wykryciu naciśnięcia przycisku będzie ona wysyłała kolejno sekwencję pięciu raportów w odstępach 100 ms. Raporty te mają następującą zawartość:

- wciśnięty przycisk A,
- wciśnięty przycisk B i lewy Shift,
- wciśnięty przycisk C i prawy Alt,
- wciśnięty przycisk CapsLock,
- brak wciśniętych przycisków.

W wyniku tego, jeżeli przy wybranym układzie klawiatury „Polski programisty” otworzony zostanie edytor tekstu, to naciśnięcie przycisku będzie powodowało wpisywanie na przemian tekstu „aBć” i „AbĆ” oraz zmianę stanu klawisza CapsLock sygnalizowaną zmianą stanu diody na płytce. Również wciśnięcie CapsLock na standardowej klawiaturze podłączonej do komputera PC spowoduje zmianę stanu diody w naszym urządzeniu (**listing 6**).

Jeśli klawiatura miałaby być obsługiwana przez podsystem BIOS, to konieczne jest również zmodyfikowanie pól podklasa I protokół w deskryptorze interfejsu – wymaga to zmiany wartości odpowiednich bajtów w tablicy USBD_CUSTOM_HID_CfgDesc w następujący sposób:

```
  0x03,          /*bInterfaceClass: HID*/
  0x01,          /*bInterfaceSubClass : 1=BOOT,
0=no boot*/
  0x01,          /*nInterfaceProtocol : 0=none,
1=keyboard, 2=mouse*/
```

Kompletne przykładowe projekty są zawarte w pliku stm32cubemx_hid.zip, dostępnym na serwerze EP.

Julia Kosowska
Grzegorz Mazur

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl