INSTYTUT INFORMATYKI
Wydział IEiT AGH

Ćwiczenie laboratoryjne lub pokaz zdalny

Interfejs USB, pamięć masowa i system plików w urządzeniach mikroprocesorowych

Nazwa kodowa: **`usb-msd`**. Wersja **`230104.0`**

Ada Brzoza
ada.brzoza@agh.edu.pl

1 Wstęp

1.1 Cel ćwiczenia

Celem ćwiczenia jest:

- nabycie praktycznych umiejętności posługiwania interfejsem USB oraz pamięcią masową w nowoczesnych mikrokontrolerach,
- wykorzystanie systemu plików do przechowywania danych w urządzeniach mikroprocesorowych o ograniczonych zasobach sprzętowych,
- weryfikacja wiedzy i ilustracja zagadnień związanych z DMA i systemem przerwań,
- utrwalenie umiejętności związanych z korzystaniem z uprzednio opracowanych rozwiązań.

1.2 Wymagania wstępne

Wymagania wstępne do przeprowadzenia ćwiczenia są następujące:

- umiejętność posługiwania się językiem C,
- rozumienie pojęcia kompilacji skrośnej (*cross-compiling*) i umiejętność posługiwania się narzędziami do tego rodzaju kompilacji,
- umiejętność rozróżnienia typowych interfejsów zewnętrznych komputera PC,
- umiejętność rozróżniania rejestrów wewnętrznych mikroprocesora oraz rejestrów układów peryferyjnych mikrokontrolera lub systemu mikroprocesorowego,
- umiejętność posługiwania się narzędziami i elementami projektu poznanymi przy realizacji poprzednich ćwiczeń.

1.3 Stanowisko testowe i konfiguracja sprzętowa

Ćwiczenie przeprowadzone jest z wykorzystaniem platformy sprzętowej **STM32F429 Nucleo-144** z mikrokontrolerem z rdzeniem ARM Cortex-M4.

1.3.1 Narzędzia programowe

Od strony programowej użyjemy następujących narzędzi:

- zintegrowanego środowiska STM32CubeIDE **w wersji 1.7.0** (**uwaga: na starszej wersji może nie zadziałać**),
- programu terminalowego, np. **puTTY**.

1.3.2 Konfiguracja sprzętowa

Do realizacji zadania zostanie wykorzystana płytka testowa Nucleo-144 z mikrokontrolerem STM32F429ZI.

Płytka może lecz nie musi być wyposażona w przewód łączący PB11 z PG0, używany w ćwiczeniu

irq-dma (dotyczącym systemu przerwań i kanałów DMA).

Uwaga: zworka PB11-PG0 nie jest wykorzystywana w podstawowej wersji realizacji niniejszego ćwiczenia – jej obecność jest opcjonalna.

Jeśli ten przewód został dołączony, to może on się przydać przy własnych działaniach, np. do automatycznego podawania sygnałów mających generować przerwanie. Do podawania sygnału może wtedy zostać wykorzystany układ czasowo-licznikowy TIM2 skonfigurowany do generowania wolnozmiennego przebiegu PWM w kanale 4, na wyprowadzeniu zewnętrznym PB11 mikrokontrolera. Sygnał prostokątny PWM na PB11 zostaje wtedy skierowany do wyprowadzenia PG0 pracującego jako wejście zewnętrznego przerwania EXTI0.

PB11 (*TIM2_CH4*, w programie ***PWM_T2CH4_OUT***)

→

PG0 (*GPIO_EXTI0*, w programie ***EXT_INT***)

Do połączenia tych wyprowadzeń może służyć zworka z przewodu dołączonego do złącz na płytce Nucleo-144.

1.4 Materiały pomocnicze

Do wykonania ćwiczenia niezbędne lub przydatne są następujące materiały:

- projekt startowy ***nucleo-basic-v4***,
- **dokumentacja modułu obsługi systemu plików FatFs:**

  http://elm-chan.org/fsw/ff/00index_e.html ,
- dokumentacja systemu operacyjnego FreeRTOS: https://www.freertos.org/ ,
- nota katalogowa (*datasheet*) mikrokontrolera STM32F429ZI,
- instrukcja obsługi (*reference manual*) mikrokontrolera STM32F429ZI,
- instrukcja obsługi (user manual) zestawu STM32F429 Nucleo-144,
- zestaw przykładów dla STM32F429 Nucleo-144.

1.5 Co należy umieć i/lub co należy uwzględnić w sprawozdaniu?

W niniejszej instrukcji wyszczególniono zagadnienia, które należy opracować samodzielnie, a następnie do wskazanych fragmentów odnieść się zwięźle w sprawozdaniu i/lub utrwalić zdobytą wiedzę i umiejętności zależnie od ogłoszonej przez osobę prowadzącą formy weryfikacji efektów uczenia dla tego zadania.

2 Wykonanie ćwiczenia

2.1 Import i otwarcie projektu

Import i otwarcie projektu przeprowadzamy w sposób analogiczny jak przy poprzednim ćwiczeniu „Niskopoziomowa implementacja systemu czasu rzeczywistego – na przykładzie FreeRTOS”. Tym razem jednak pobieramy i importujemy projekt startowy ***nucleo-basic-v4***. Projekt jest bardzo podobny do tego, który był zestawiany w ćwiczeniu dotyczącym rozwoju niskopoziomowego oprogramowania (nazwa kodowa ***debug-devel***).

2.2 Sprawdzenie działania projektu startowego

Projekt startowy zawiera już fragmenty kodu do zapisu i odczytu danych z plików realizowane w ćwiczeniu o nazwie kodowej ***debug-devel***. Na początek należy uruchomić dostarczony kod aktualnego ćwiczenia i sprawdzić, czy działają odpowiednie moduły.

Obecność napięcia zasilającego urządzenie USB podłączone do płytki testowej Nucleo-144 sygnalizowana jest świeceniem zielonej diody LD8 znajdującej się w „dolnej” części płytki, blisko złącz Ethernet, USB-OTG i przycisków.

2.2.1 Komunikaty po zerowaniu

1. W zestawionym programie terminalowym powinny pojawić się komunikaty wypisywane przy pomocy funkcji *printf* i *xprintf*. Błąd dotyczący biblioteki LwIP możemy zignorować.

```
Nucleo-144 F429ZI nucleo-basic-v3 project
Funkcja xprintf dziala.
Funkcja printf tez dziala.
Assertion "netif is not up, old style port?" failed at line 727 in
../Middlewares/Third_Party/LwIP/src/core/ipv4/dhcp.c
```

2. Oczekiwanie na gotowość urządzenia magazynującego odbywa się w funkcji

```
void waitForUsbMsd(void)
{
  xprintf("waiting for USB device...\n");
      do{
            vTaskDelay(100);
            HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
      }while(Appli_state != APPLICATION_READY);
      xprintf("USB device ready!");
      HAL_GPIO_WritePin(LD2_GPIO_Port,LD2_Pin,GPIO_PIN_RESET);
}
```

   Powinna wtedy błyskać dioda LD2 (niebieska) i wyświetlany będzie komunikat

```
waiting for USB device...
```

3. Gdy urządzenie magazynujące będzie gotowe, zacznie błyskać LD1 (zielona). Jeśli wszystko jest prawidłowo zestawione (pamięć masowa podłączona do płytki), to błyskania LD2 możemy nawet nie zauważyć i od razu zacznie błyskać LD1. Urządzenie magazynujące powinno zgłosić się z odpowiednim komunikatem na terminalu. Przykładowy komunikat jest zależny od urządzenia magazynującego podłączonego do płytki, na której

realizowane jest ćwiczenie i może być następujący:

```
USB Device AttachedPID: bc06h
VID: 324h
Address (#1) assigned.
Manufacturer : OCZ
Product : ATV
Serial Number : AA04012700059443
Enumeration done.
This device has only 1 configuration.
Default configuration set.
Switching to Interface (#0)
Class    : 8h
SubClass : 6h
Protocol : 50h
MSC class started.
Number of supported LUN: 1
LUN #0:
Inquiry Vendor  : OCZ
Inquiry Product : ATV
Inquiry Version : 1100
MSC Device ready
MSC Device capacity : 3724541440 Bytes
Block number : 15663103
Block Size   : 512
USB device ready!
```

2.2.2 Dostępna funkcjonalność startowa

Wydawanie poleceń dla urządzenia z oprogramowaniem startowym odbywa się przez wysyłanie do terminala (na */dev/ttyACMx*) znaków ASCII z klawiatury:

- `'w'` spowoduje dodanie treści do pliku `test.txt` . W aktualnej wersji fragment wpisujący będzie prawdopodobnie zbliżony do poniższego:

```
case 'w':
{
  xprintf("write-append test\n");
  FRESULT res;
  UINT bw;
  FIL file;
  const int TEXTBUF_SIZE = 128;
  char text[TEXTBUF_SIZE];
  sprintf(text,"Linia tekstu. Czas systemowy to: %08d\n",
       (int)xTaskGetTickCount());
  xprintf("f_open... ");
  res = f_open(&file,"0:/test.txt",FA_WRITE|FA_OPEN_APPEND);
  xprintf("res=%d\n",res);
  if(res) break;
  xprintf("f_write... ");
  res = f_write(&file,text,strlen(text),&bw);
  xprintf("res=%d, bw=%d\n",res,bw);
  f_close(&file);
  break;
```

- Jeśli plik wcześniej nie istniał, zostanie utworzony na nośniku.
- `'h'` spowoduje odczyt pliku `test.txt` sformatowanego podobnie jak w programach do analizy danych w formacie heksadecymalnym. Limit dla jednego wywołania wynosi 4 KiB:

  ```
  int dispSizeLimit = 4*1024;
  ```

  Ta funkcjonalność może być przydatna w sytuacjach, gdyby w pliku nie znajdował się sam

czytelny tekst tylko dane binarne, w tym „trudniejsze” do wyświetlenia na terminalu znaki.

- `'r'` działa jak `'h'` ale próbuje wyświetlać tekst tylko jako znaki ASCII.

Pozostałe funkcjonalności mogą być przydatne w „sytuacjach awaryjnych”, gdyby potrzebne było zrestartowanie komunikacji USB lub sformatowanie nośnika pamięci masowej.

- `'0'` wyłączy zasilanie urządzenia magazynującego,

```
case '0':
	MX_DriverVbusFS(1);
	xprintf("Connected USB Device is OFF\n");
	break;
```

- `'1'` włączy zasilanie urządzenia magazynującego i wywoła funkcję oczekiwania na jego gotowość **`waitForUsbMsd`**,

```
case '1':
	MX_DriverVbusFS(0);
	xprintf("Connected USB Device is ON\n");
	waitForUsbMsd();
	break;
```

- `'i'` wywoła ponownie procedurę inicjalizacji modułu obsługi systemu plików:

```
case 'i':
	xprintf("Re-initializing FatFS...");
	MX_FATFS_Init();
	xprintf(" reinit done\n");
	break;
```

- `'!'` pozwala na wywołanie procedury formatowania urządzenia magazynującego USB (tutaj pendrive’a). W wyniku formatowania wszystkie dane na nośniku zostaną utracone, zostanie na nim utworzony jedynie „pusty” system plików oraz powinien on być po tej procedurze gotowy na utworzenie nowych plików. **Opcji formatowania należy używać jako „ostatecznej” i jedynie po konsultacji z osobą prowadzącą zajęcia, aby uzyskać informację, czy problem na pewno wymaga formatowania nośnika.** Aby, po wywołaniu polecenia `'!'` przejść do właściwej procedury formatowania, należy potwierdzić swój zamiar wysyłając znak `'Y'` (uwaga: *case sensitive*). Fragment kodu odpowiadający za formatowanie jest następujący:

```
case '!':
	xprintf("Format the USB drive? Send Y to confirm, any other to abort\n");
	do
	{
		key = inkey();
		if(key == 'Y')
		{
			xprintf("Format in progress...\n");
			FRESULT res =
			  f_mkfs("0:",FM_FAT|FM_FAT32,512,fs_buffer,512);
			xprintf("f_mkfs result code: %d\n",res);
		}
		else if(key != 0)
		{
			xprintf("abort\n");
		}
	}while(!key);
	key = 0;
	break;
```

2.2.3 Sprawozdanie

Do sprawozdania z niniejszej sekcji należy wkleić komunikaty, które pojawiły się od momentu wyzerowania płytki Nucleo-144:

a) po podłączeniu urządzenia pamięci masowej,

b) po wydaniu komunikatu aż do zasygnalizowania gotowości urządzenia USB,

c) po wydaniu poleceń **`0`**, **`i`**, trzykrotnie **`w`**, **`r`** i **`h`**.

2.3 Operacje na plikach

W module **`main.c`** należy napisać funkcje testowe pozwalające na wykonanie operacji na plikach. Funkcje powinny być wywoływane przez wysyłanie wybranych znaków z programu terminalowego. Do realizacji zadań używamy funkcji oferowanych przez bibliotekę FatFS.

Dla poszczególnych podpunktów niniejszej sekcji **w sprawozdaniu** (w formie tekstowej) umieszczamy fragmenty utworzonego kodu oraz screenshot ewentualnie uzupełniony krótkim nagraniem okna terminala wyświetlającego komunikaty potwierdzające działanie tych fragmentów.

W module **`main.c`** została utworzona statycznie alokowana tablica ogólnego przeznaczenia **`fs_buffer`** o rozmiarze wstępnie ustawionym na 8 KiB do wykorzystania do własnych działań. Rozmiar tej tablicy można zmieniać zależnie od potrzeb i dostępności pamięci operacyjnej. Należy pamiętać o mechanizmach synchronizacji, jeśli mamy zamiar wykorzystywać ją z więcej niż jednego kontekstu.

```
#define FS_BUFFER_SIZE  (8*1024)
static uint8_t fs_buffer[FS_BUFFER_SIZE];
```

2.3.1 Funkcja tworząca plik

Należy napisać funkcję tworzącą na nośniku plik o nazwie przekazywanej w jej argumencie. Jeśli plik istnieje, to powinien on zostać nadpisany nową zawartością. Do pliku wpisujemy fragment tekstu – może być podany jako stały ciąg znaków w programie. Sprawdzamy, czy zgadza się liczba wpisanych bajtów.

2.3.2 Dodanie danych do istniejącego pliku

Należy utworzyć funkcję dodającą do wskazanego pliku dalszą treść – analogicznie jak w przykładzie w funkcji main. Funkcję można przetestować na pliku utworzonym w punkcie 2.3.1 dopisując do niego nową treść bez kasowania poprzedniej zawartości.

2.3.3 Wyświetlenie zawartości pliku na terminalu

Tworzymy funkcję wyświetlającą zawartość pliku o nazwie przekazanej w argumencie. Ze względów praktycznych należy ograniczyć ilość wyświetlanych danych. Jeśli plik będzie krótszy niż np. 1 KB, jego zawartość powinna być wyświetlana w całości. Wyświetlanie dłuższych plików ograniczamy do 1 KB.

2.3.4 Wyświetlenie listy plików z katalogu głównego

Należy utworzyć funkcję wyświetlającą listę plików dostępnych w katalogach głównych nośników używanych w ćwiczeniu.

2.3.5 Sprawdzenie szybkości działania

W tej części tworzymy funkcję, która dla wybranego nośnika utworzy na nim plik, wpisze do niego arbitralne dane, a następnie odczyta te dane. Czasy zapisu i odczytu powinny zostać zmierzone. Na podstawie rozmiaru zapisywanych lub odczytywanych danych oraz czasu trwania tych operacji, należy obliczyć szybkość transferu dla zapisu i odczytu dla urządzenia magazynującego. Szybkość transferu należy wyświetlić w programie terminalowym i zamieścić w sprawozdaniu.

Uwagi:

- Zalecany orientacyjny rozmiar pliku testowego to 4 MiB. Można sprawdzić różne rozmiary plików testowych.
- Do pomiaru czasu można użyć wywołań funkcji **`xTaskGetTickCount`** nie pobierającej argumentów wejściowych i zwracającej aktualny czas systemowy wyrażony w tyknięciach systemowych trwających w domyślnej konfiguracji 1 ms.

2.4 Badanie i analiza poleceń dla urządzenia magazynującego wydawanych przez host USB

Host USB wysyła do urządzenia klasy magazynującej polecenia specyficzne dla urządzeń tej klasy, w tym odpowiada za komunikacje przez wirtualny interfejs SCSI.

2.4.1 Zadania do wykonania

Należy testowo wyświetlić i zaobserwować, które polecenia wysyłane są do dołączonego urządzenia magazynującego i w których chwilach takich jak: dołączenie pendrive'a, odczyt krótkiego pliku, zapis krótkiego pliku. Obsługę SCSI znajdziemy w pliku

`./Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Src/usbh_msc_scsi.c.`

Dla umożliwienia przeprowadzenia analizy możemy wyświetlić w sesji debugowania lub na terminalu parametry komend SCSI, np. numer bloku do zapisu/odczytu i kod komendy. Sam kod komendy można porównać z definicjami znajdującymi się w pliku nagłówkowym

`./Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Inc/usbh_msc_scsi.h`.

2.4.2 Sprawozdanie

W sprawozdaniu proszę uwzględnić następujące opisy, które także będą przydatne przy utrwalaniu wiedzy i umiejętności:

(1) Zaobserwowane działanie i wnioski.

(2) Po przeanalizowaniu m.in. treści modułu **`usbh_msc.c`** należy odpowiedzieć na pytanie, co to jest pojawiający się tam często **LUN** (*Logical Unit Number*)? W których sytuacjach może on przyjmować różne wartości i do czego będzie wtedy przydatny?

Po zakończeniu badania działania wirtualnego interfejsu SCSI warto usunąć funkcje wyświetlające komunikaty, aby niepotrzebnie nie opóźniały działania badanego dalej firmware'u – oczywiście, jeśli je wcześniej utworzyliśmy.

2.5 Badanie i analiza procesu enumeracji urządzenia USB

W oprogramowaniu badanym na laboratorium funkcją odpowiadającą za enumerację jest **`USBH_HandleEnum`**. Znajduje się ona w module

`./Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_core.c`

wraz ze stowarzyszonym plikiem nagłówkowym usbh_core.h. Do przebadania działania funkcji **`USBH_HandleEnum`** powinna wystarczyć analiza działania pary plików **`usbh_core.c`** i **`.h`**.

Proszę przeanalizować i **zamieścić w sprawozdaniu** wnioski opisujące własnymi słowami w kilku zdaniach informacje:

- co dzieję się podczas procesu enumeracji urządzenia USB?
- czemu może służyć ten proces: co ułatwia/umożliwia?
- których aspektów działania urządzenia USB dotyczą dane wymieniane w procesie enumeracji?

3 Zbiór zagadnień do uzupełnienia i utrwalenia zdobytej wiedzy i umiejętności

Zależnie od tego, na którym kursie i w jakiej formie odbywa się realizacja ćwiczenia laboratoryjnego *usb-msd*, obowiązujący może być pewien wybrany podzbiór zagadnień wskazany w trakcie zajęć przez osobę prowadzącą. Część zagadnień wymaga samodzielnego znalezienia informacji.

Dotyczy przedmiotu Sprzętowe aspekty cyberbezpieczeństwa: w sprawozdaniu proszę zawrzeć informacje z punktów: **2b), 2c), 3, 4 i 5**.

1. Zagadnienia zawarte w instrukcjach do ćwiczeń i badane w czasie realizacji ćwiczenia,
2. Umiejętność wskazania klasy USB, w której pracuje urządzenie badane na laboratorium *usb-msd* – to byłoby właściwie za proste ;) dlatego także:
   a) podstawowe informacje na temat innych popularnych klas urządzeń USB: CDC i HID – (przede wszystkim przykłady urządzeń),
   b) którego typu urządzenie USB badaliśmy na laboratorium: host, device, OTG? A może OTG działające w trybie host-only / device-only / dual-role?
   c) szybkość transmisji USB:
      1. z jaką szybkością pracuje badane na laboratorium urządzeni USB?
      2. jakie mogą być inne popularne szybkości działania urządzeń USB i jak są nazywane interfejsy USB o różnych szybkościach transferu danych?

d) właściwości i możliwości funkcjonalne urządzeń z interfejsem USB-OTG (*On The Go*),

3. Proces enumeracji urządzenia USB,
   a) co dzieję się podczas procesu enumeracji urządzenia USB?
   b) czemu może służyć ten proces: co ułatwia/umożliwia?
   c) których aspektów działania urządzenia USB dotyczą dane wymieniane w procesie enumeracji?
4. Ogólne informacje na temat urządzeń magazynujących USB MSD – BOT (*Mass Storage Device*, *Bulk-Only Transport*):
   a) typowy interfejs programowy i sposób wymiany danych dla urządzeń magazynujących o dostępie blokowym,
   b) adresowanie LBA,
   c) typowo stosowany rozmiar bloku,
   d) możliwość rozróżnienia wielu jednostek logicznych i fizycznych, numer LUN oraz przykłady jego zastosowania.
5. SCSI w urządzeniach USB MSD BOT (*Mass Storage Device Bulk-Only Transport*):
   a) sposób przesyłania danych przez SCSI przy wymianie danych z urządzeniami USB klasy MSD BOT
   b) podstawowe komendy SCSI,
   c) sposób przesyłania komend SCSI w urządzeniach pamięci masowej USB,
   d) dzięki której komendzie host może poznać nazwę i parametry urządzenia magazynującego?
6. Wiedza w zakresie dostępnych funkcji biblioteki FatFs wykorzystywanych w realizacji ćwiczenia *usb-msd*.
7. Właściwości, dostępne funkcje i potencjalne możliwości biblioteki obsługi systemu plików FatFs używanej w ćwiczeniu *usb-msd*,