STM32: Urządzenie USB-CDC z CubeMX – krok po kroku

Środowisko CubeMX, udostępniane za darmo przez ST Microelectronics, umożliwia łatwe konfigurowanie mikrokontrolerów rodziny STM32 i generowanie szkieletów programów. CubeMX zawiera wiele gotowych modułów programowych, które mogą być włączane do tworzonych programów. Znajdziemy wśród nich m.in. stos USB wraz z obsługą wybranych klas urządzeń. Jedną z obsługiwanych klas jest CDC, umożliwiająca implementację wirtualnego portu szeregowego. Korzystanie z CubeMX i praktyczna realizacja projektu wymaga jednak od programisty wiedzy, którą trudno odnaleźć w udostępnianej wraz z pakietem CubeMX dokumentacji.

W artykule zostanie przedstawiony proces tworzenia oprogramowania dla urządzenia CDC. Przebiega on niemal identycznie dla każdej docelowej platformy sprzętowej w obrębie rodziny STM32 – różnice w generowaniu projektu pomiędzy różnymi seriami i modelami mikrokontrolerów polegają jedynie na nieco odmiennej konfiguracji taktowania poszczególnych typów układu. W prezentowanym projekcie użyto prawdopodobnie najtańszej dostępnej obecnie platformy uruchomieniowej dla STM32 – wytwarzanej w Chinach płytki „STM32F103C8T6 ARM STM32 Minimum System Development Board". Do kompilacji programu użyto darmowej wersji środowiska Keil MDK-ARM 5.x.

Płytka STM32F103C8T6 ARM STM32 Minimum System Development Board

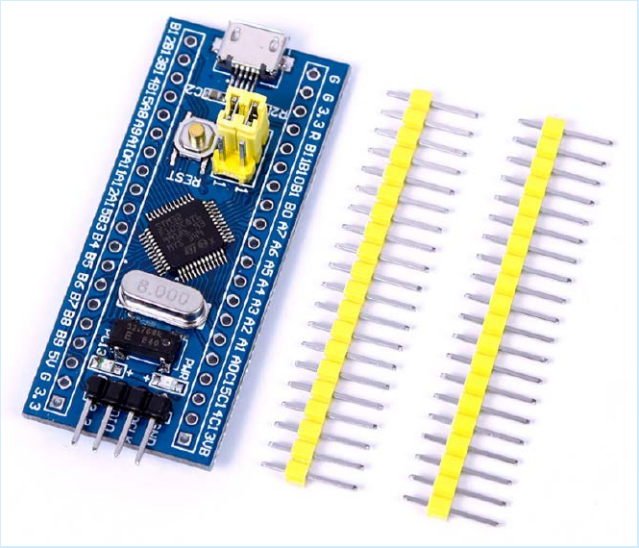
Płytka, którą dalej w skrócie nazywam F103MDB, została zaprojektowana przez chińską firmę występującą pod adresem internetowym *www.vcc-gnd.com*. Tam też można znaleźć dokumentację płytki – schemat i rysunek.

Płytka ma wymiary ok. 51 mm×23 mm (**fotografia 1**). Znajdują się na niej:

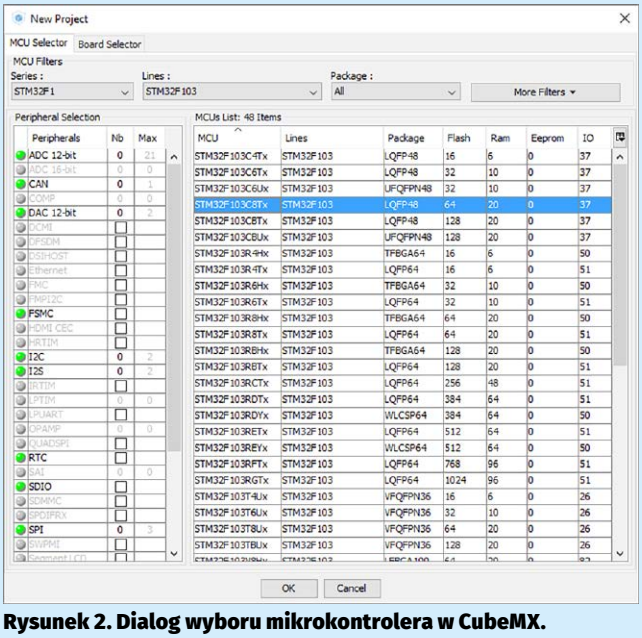
- Mikrokontroler STM32F103C8T6.
- Złącze USB micro AB dołączone do interfejsu USB mikrokontrolera i służące do zasilania modułu.
- Stabilizator napięcia zasilania 3,3 V.
- Przycisk RESET.
- Dwa rezonatory kwarcowe (8 MHz i 32768 Hz).
- Zwory umożliwiające ustalenie stanu wejść BOOT0 i BOOT1 mikrokontrolera w celu użycia wewnętrznego programu ładującego.
- Dwie diody LED – jedna do sygnalizacji zasilania, druga sterowana z wyjścia PC13 mikrokontrolera.
- Złącze interfejsu SWD do programowania i debugowania.

Na płytce znajdują się dwa rzędy otworów, w które można wlutować dostarczane wraz z płytką złącza typu header lub np. szpilki precyzyjne. W każdym rzędzie jest 20 otworów, a ich raster odpowiada obudowie DIL40, więc płytka F103MDB może być łatwo zintegrowana z dowolną płytką drukowaną z miejscem pod obudowę DIL40. Złącza te udostępniają niemal wszystkie wyprowadzenia mikrokontrolera oraz linie zasilania. Poszczególne wyprowadzenia są podpisane na płytce.

Płytka nie zawiera wbudowanego programatora/debuggera. Mikrokontroler można programować przez interfejs SWD, korzystając

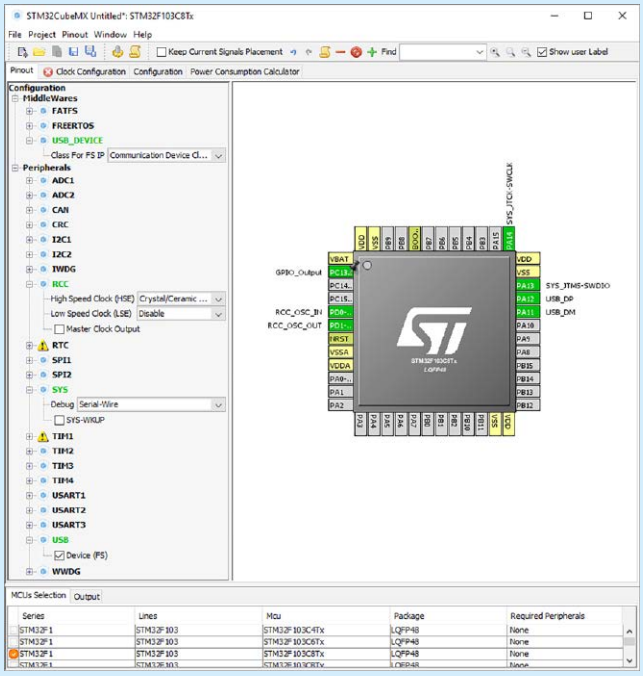


Fotografia 1. Płytka STM32F103C8T6 ARM STM32 Minimum System Development Board



Rysunek 2. Dialog wyboru mikrokontrolera w CubeMX.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **3372**, pass: **6nwd5fk4**



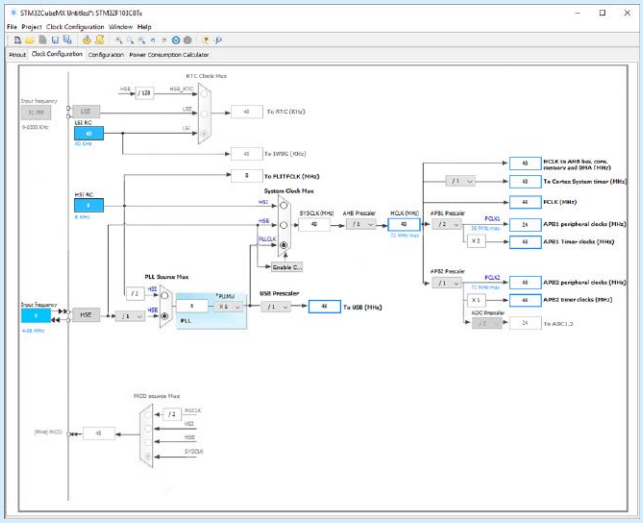
Rysunek 3. Zakładka konfiguracji peryferiów i oprogramowania

w tym celu z dowolnego programatora/debuggera zewnętrznego, np. ST-Link z płytki serii Discovery lub Nucleo. Jest również możliwe programowanie mikrokontrolera przy użyciu wbudowanego bootloadera, poprzez interfejs UART, przy użyciu zewnętrznego modułu interfejsu USB-UART.

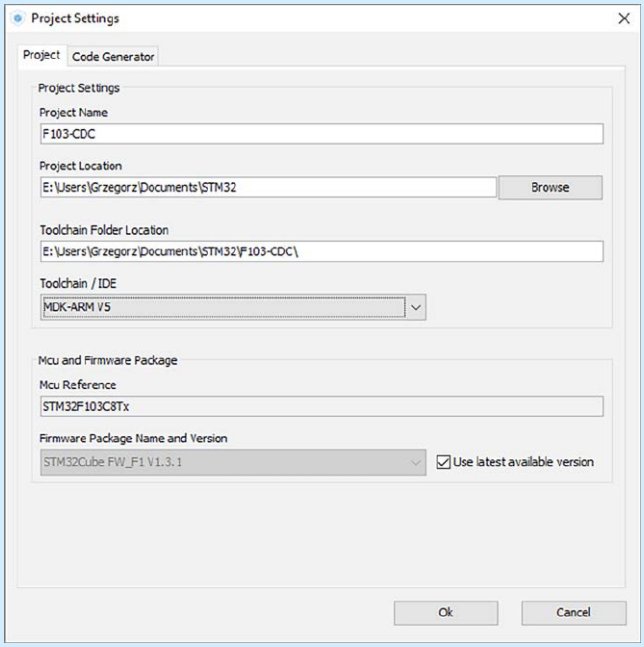
Niska cena modułu F103MDB sprawia, że jest on idealną platformą do prototypowania i projektów amatorskich, zwłaszcza, że wydajność obliczeniowa i własności funkcjonalne mikrokontrolera STM32F103 znacznie przewyższają odpowiednie parametry mikrokontrolerów 8-bitowych używanych w popularnych płytkach rodziny Arduino.

Tworzenie projektu oprogramowania w CubeMX

Tworzenie projektu rozpoczynamy od uruchomienia CubeMX i wybrania opcji *New Project* z ekranu głównego lub z menu. Następnie wybieramy typ mikrokontrolera (**rysunek 2**) – w naszym przypadku jest to STM32F103C8Tx, po czym naciskamy przycisk OK. Spowoduje to otwarcie w głównym oknie programu zakładki *Pinout* (**rysunek 3**) z widokiem wyprowadzeń mikrokontrolera i panelu konfiguracji peryferiów i oprogramowania.



Rysunek 4. Zakładka konfiguracji taktowania.



Rysunek 5. Dialog ustawień projektu, zakładka *Project*.

W zakładce tej konfigurujemy kolejno:

- Moduł SYS – *Debug: Serial-Wire* – w celu zapewnienia możliwości debugowania oprogramowania i ponownego programowania mikrokontrolera.
- Moduł RCC – *High Speed Clock (HSE): Crystal/Ceramic Resonator.*
- Moduł USB – *Device (FS).*
- Oprogramowanie (MiddleWares) – USB_DEVICE – *Class For FS IP: Communication Device Class.*
- W widoku układu klikamy prawym przyciskiem myszy na linii PC13 i wybieramy funkcję *GPIO_Output.*

Następnie przechodzimy do drugiej zakładki – *Clock Configuration* (**rysunek 4**), w której konfigurujemy taktowanie mikrokontrolera. Podczas uaktywniania tej zakładki pojawi się dialog z pytaniem o automatyczną konfigurację zegara. Jeżeli odpowiemy twierdząco, CubeMX zaproponuje konfigurację zegarów, którą powinniśmy zweryfikować i ewentualnie skorygować, zaczynając od częstotliwości oscylatora HSE (dla większości typowych płytek z mikrokontrolerami STM32, w tym płytki F103MDB, wynosi ona 8 MHz). Jeżeli zdecydujemy się na ręczną konfigurację taktowania, możemy zauważyć wyświetlone na czerwono znaczniki błędów konfiguracji. Po poprawnym skonfigurowaniu taktowania wszystkie znaczniki błędów powinny zniknąć. W celu skonfigurowania taktowania najpierw ustawiamy częstotliwość wejściową HSE: 8 MHz, a następnie wybieramy kolejno:

- Źródło taktowania PLL: HSE.
- Mnożnik PLL: 6.
- Preskaler USB: 1.
- Źródło zegara systemowego: PLLCLK.
- Preskaler AHB: 1.
- Preskaler APB1: 2.

W ten sposób uzyskamy częstotliwość taktowania rdzenia równą 48 MHz. Jeżeli chcemy użyć maksymalnej częstotliwości dostępnej dla STM32F103 (72 MHz), ustawiamy odpowiednio mnożnik PLL na 9 i preskaler USB na 1.5.

W przypadku przygotowywania projektu dla innego typu mikrokontrolera, musimy zawsze ustawić częstotliwość taktowania modułu USB na 48 MHz i dobrać do niej częstotliwość taktowania rdzenia (zwykle 32 MHz dla serii STM32L0 i nie mniej iż 48 MHz dla pozostałych serii rodziny STM32).

W ten sposób zakończyliśmy podstawową konfigurację zasobów mikrokontrolera i możemy przejść do generowania projektu oprogramowania. W tym celu z menu CubeMX wybieramy opcję *Project-Settings* (**rysunek 5**). Po otwarciu dialogu ustawień projektu, w zakładce *Project* wprowadzamy nazwę projektu, wybieramy folder projektu (jest to folder, w którym CubeMX utworzy plik projektu i folder z plikami źródłowymi oprogramowania) oraz wybieramy środowisko programowania – w naszym przypadku jest to Keil MDK-ARM V5.

W drugiej zakładce dialogu, *Code Generator* (**rysunek 6**), możemy ustawić dodatkowe opcje projektu. Jeśli nie planujemy dalszej rozbudowy projektu przy użyciu środowiska CubeMX, wybierając opcję *Copy only the necessary library files* możemy znacząco zmniejszyć zajętość pamięci masowej przez pliki projektu. 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 naciśnięciu przycisku OK, o ile wcześniej nie został zainstalowany w CubeMX pakiet obsługi wybranego typu mikrokontrolera, nastąpi jego ściągnięcie i zainstalowanie. Po uzupełnieniu konfiguracji CubeMX możemy przejść do generowania kodu. W tym celu naciskamy przycisk generowania kodu w pasku narzędzi lub wybieramy z menu opcję *Project-Generate Code*.

Po pomyślnym wygenerowaniu projektu CubeMX zaproponuje otwarcie go w wybranym środowisku programowania. Możemy od razu skompilować projekt, sprawdzając, że nie ma w nim błędów, jednak przed uruchomieniem oprogramowania musimy jeszcze nieco uzupełnić wygenerowany przez CubeMX kod źródłowy, w celu poprawienia kilku błędów i nadania programowi minimalnej funkcjonalności.

Niezbędne modyfikacje programu

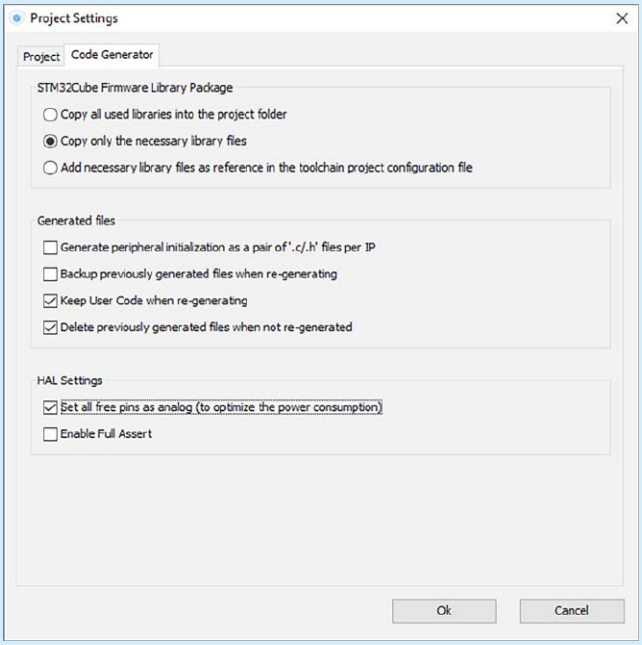
Interakcja programu z urządzeniem USB CDC jest zawarta w pliku usbd_cdc_if.c, umieszczonym w projekcie w folderze Application/User. Modyfikację kodu rozpoczniemy od niezbędnej poprawki – musimy zmienić definicje dwóch stałych, APP_RX_DATA_SIZE i APP_TX_DATA_SIZE, nadając im wartość 64 zamiast 4.

W plikach wygenerowanych przez CubeMX i przewidzianych do modyfikacji przez użytkownika umieszczone są pary komentarzy, rozpoczynających się od fraz /* USER CODE BEGIN i /* USER CODE END

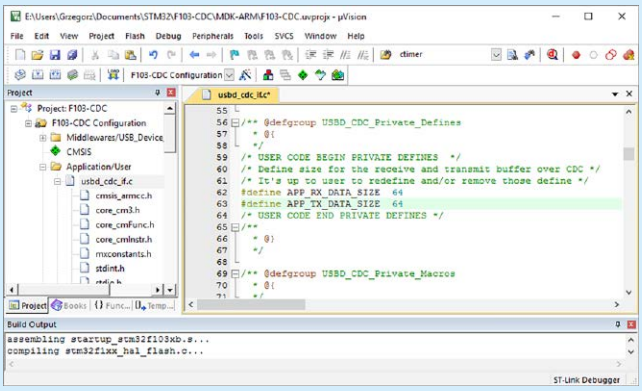
Kod użytkownika powinien być umieszczony pomiędzy tymi komentarzami. Gwarantuje to, że kod ten nie zostanie zmieniony w przypadku modyfikacji projektu przez CubeMX i powtórnym generowaniu kodu źródłowego. Niestety, mechanizm ten nie działa do końca poprawnie – po zmodyfikowaniu wymienionych wyżej definicji i powtórnym wygenerowaniu kodu w CubeMX, zostaną przywrócone błędne wartości obu stałych (**rysunek 7**).

Zanim zajmiemy się dopisaniem zaplanowanej funkcjonalności naszego programu, musimy poznać strukturę oprogramowania wygenerowanego przez CubeMX. Plik usbd_cdc_if.c zawiera cztery funkcje przeznaczone do rozbudowy przez programistę, których szkielety zostały wygenerowane przez CubeMX. Są one wywoływane przez procedurę obsługi przerwania modułu USB przy wystąpieniu zdarzeń wymagających reakcji programu użytkowego. Funkcje te, to:

- **CDC_Init_FS()** – wywoływana przy nawiązaniu połączenia z komputerem przez interfejs USB.
- **CDC_DeInit_FS()** – wywoływana przy odłączeniu od komputera.
- **CDC_Control_FS()** – wywoływana przy zmianie ustawień portu szeregowego urządzenia CDC (VCOM).
- **CDC_Receive_FS()** – wywoływana przy odebraniu danych z komputera.



Rysunek 6. Dialog ustawień projektu, zakładka *Code Generator*.



Rysunek 7. Poprawione definicje rozmiaru buforów w pliku usbd_cdc_if.c

Niestety, oprogramowanie USB CDC nie zawiera mechanizmu jawnego informowania o gotowości do wysyłania danych z urządzenia do komputera. W praktycznym zastosowaniu taki mechanizm jest potrzebny, a jego stworzenie wymaga zmodyfikowania plików *usbd_cdc_if.c*, *usbd_cdc_if.h*, *usbd_cdc.c* i *usbd_cdc.h*.

Po zainicjowaniu interfejs nie jest gotowy do odbioru – w celu włączenia odbioru danych należy w funkcji CDC_Init_FS() wywołać funkcję USBD_CDC_ReceivePacket().

W celu interpretacji danych przychodzących z komputera, musimy rozbudować funkcję CDC_Receive_FS(). Ma ona dwa argumenty: adres bufora, w którym są przechowywane dane odebrane przez interfejs USB oraz adres zmiennej zawierającej długość tych danych. Po przetworzeniu danych musimy zasygnalizować zwolnienie bufora i gotowość do przetworzenia następnej porcji danych. W tym celu przy końcu funkcji CDC_Receive_FS() musimy wywołać funkcję USBD_CDC_ReceivePacket(). Wbrew swej nazwie funkcja ta nie odbiera żadnych danych, a jedynie „uzbraja" interfejs USB CDC, zezwalając na odbiór danych.

Typowo, funkcja CDC_Receive_FS() powinna albo interpretować odebrane dane, albo kopiować je do bufora i informować inną część oprogramowania o ich nadejściu.

Do nadawania danych do komputera służy funkcja CDC_Transmit_FS(), zdefiniowana w pliku usbd_cdc_if.c. Ponieważ oprogramowanie USB CDC nie informuje jawnie o gotowości

Listing 1. Zmodyfikowana funkcja CDC_Init_FS() w pliku usbd_cdc_if.c

```
static int8_t CDC_Init_FS(void)
{
    hUsbDevice_0 = &hUsbDeviceFS;
    /* USER CODE BEGIN 3 */
    /* Set Application Buffers */
    USBD_CDC_SetTxBuffer(hUsbDevice_0, UserTxBufferFS, 0);
    USBD_CDC_SetRxBuffer(hUsbDevice_0, UserRxBufferFS);
    USBD_CDC_ReceivePacket(hUsbDevice_0);          // added
    return (USBD_OK);
    /* USER CODE END 3 */
}
```

Listing 2. Zmodyfikowana funkcja CDC_Receive_FS() w pliku usbd_cdc_if.c

```
static int8_t CDC_Receive_FS (uint8_t* Buf, uint32_t *Len)
{
    /* USER CODE BEGIN 6 */
    uint32_t i;
    for (i = 0; i < *Len; i ++)
    {
        if ((Buf[i] >= ,A' && Buf[i] <= ,Z')
            || (Buf[i] >= ,a' && Buf[i] <= ,z'))
        Buf[i] ^= ,A' ^ ,a';
    }
    CDC_Transmit_FS(Buf, *Len);
    USBD_CDC_ReceivePacket(hUsbDevice_0);
    GPIOC->ODR ^= 1 << 13;    // toggle LED
    return (USBD_OK);
    /* USER CODE END 6 */
}
```

do nadawania, a sama funkcja CDC_Transmit_FS() jedynie inicjuje transmisję, nie czekając na jej zakończenie, wywołanie tej funkcji może się nie powieść – w takim przypadku zwraca ona wartość USBD_BUSY.

W programie przykładowym chcemy zademonstrować minimalną funkcjonalność urządzenia CDC. Będzie on realizował funkcję echa, zamieniając małe litery na wielkie i odwrotnie. Ponadto po odebraniu każdego znaku nastąpi zmiana stanu diody LED. W tak prostym zastosowaniu nie ma potrzeby dodatkowego buforowania danych; ponadto możemy przyjąć, że szybkości odbioru i nadawania pakietów danych przez interfejs USB wraz z jego oprogramowaniem po stronie obu urządzeń są jednakowe, a więc odebrane dane mogą zostać zmodyfikowane i odesłane przed odebraniem następnego pakietu. Należy zauważyć, że projektując oprogramowanie o jakości produkcyjnej nie powinniśmy utrzymywać takich założeń.

Poza opisaną powyżej poprawką, musimy wprowadzić do pliku usbd_cdc_if.c dwie istotne modyfikacje:

- Dodać na końcu funkcji CDC_Init_FS(), bezpośrednio przed powrotem, wywołanie USBD_CDC_ReceivePacket(hUsbDevice_0).
- Rozbudować funkcję CDC_Receive_FS() o założoną funkcjonalność.

Zmodyfikowane wersje obu funkcji przedstawiono na **listingach 1** i **2**.

W zależności od wybranej serii mikrokontrolerów, oprogramowanie USB-CDC może zawierać jeszcze jeden poważny błąd uniemożliwiający jego poprawne działanie i wymagający korekty. Błąd ten nie występuje w bieżącej wersji oprogramowania dla STM32F1xx; polega on na użyciu dynamicznej alokacji pamięci przy niewystarczającym rozmiarze sterty. Jeżeli w plku usbd_conf nie występuje definicja *#define USBD_malloc (uint32_t *)USBD_static_malloc,* a symbol USBD_malloc jest zdefiniowany jako wywołanie standardowej funkcji malloc(), to niezbędne jest zwiększenie rozmiaru sterty w pliku startup_stm32xxx.s do np. 0x400, poprzez modyfikację definicji *Heap_Size EQU 0x00000400.*

Obsługa urządzenia CDC przez system operacyjny

ST Microelectronics dostarcza driver dla systemów Windows do urządzeń CDC stworzonych przy użyciu CubeMX. Systemy rodziny Linux zazwyczaj obsługują urządzenie USB CDC przy użyciu standardowego drivera systemowego, niewymagającego dodatkowej instalacji. Możliwość taką ma również Windows 10, pod warunkiem odpowiedniej zawartości deskryptora urządzenia (*Device Descriptor*). Windows 10 używa drivera wbudowanego, jeśli pola *DeviceClass* i *DeviceSubClass* mają wartość 2, co odpowiada klasie CDC i podklasie ACM w specyfikacji USB.

Listing 3. Zmodyfikowany deskryptor urządzenia w pliku usbd_desc.c

```
/* USB Standard Device Descriptor */
__ALIGN_BEGIN uint8_t USBD_FS_DeviceDesc[USB_LEN_DEV_DESC] __
ALIGN_END =
{
    0x12,                       /*bLength */
    USB_DESC_TYPE_DEVICE,       /*bDescriptorType*/
    0x00,                       /* bcdUSB */
    0x02,
    // changed 2 lines from 0 to 2
    0x02,                       /*bDeviceClass*/
    0x02,                       /*bDeviceSubClass*/
    0x00,                       /*bDeviceProtocol*/
    USB_MAX_EP0_SIZE,          /*bMaxPacketSize*/
    LOBYTE(USBD_VID),           /*idVendor*/
    HIBYTE(USBD_VID),           /*idVendor*/
    LOBYTE(USBD_PID_FS),           /*idVendor*/
    HIBYTE(USBD_PID_FS),           /*idVendor*/
    0x00,                       /*bcdDevice rel. 2.00*/
    0x02,
    USBD_IDX_MFC_STR,           /*Index of manufacturer
string*/
    USBD_IDX_PRODUCT_STR,       /*Index of product string*/
    USBD_IDX_SERIAL_STR,        /*Index of serial number
string*/
    USBD_MAX_NUM_CONFIGURATION  /*bNumConfigurations*/
  };
```

Aby wyeliminować potrzebę użycia dodatkowego drivera w Windows 10, należy zmodyfikować deskryptor urządzenia, zawarty w pliku *usbd_desc.c*, zgodnie z **listingiem 3**. Należy przy tym pamiętać, że przy ewentualnym powtórnym generowaniu projektu przez CubeMX, modyfikacja ta zostanie usunięta.

Po dokonaniu wszystkich modyfikacji można skompilować projekt, naciskając przycisk Build. Programowanie mikrokontrolera i debugowanie przy użyciu interfejsu ST-Link. Programowanie zewnętrznego mikrokontrolera przy użyciu interfejsu ST-Link z płytki serii Nucleo lub Discovery wymaga usunięcia dwóch zwor umieszczonych w pobliżu złącza interfejsu ST-Link.

Przed użyciem interfejsu ST-Link należy odpowiednio skonfigurować go w środowisku programowania. W tym celu po podłączeniu interfejsu ST-Link otwieramy dialog ustawień Options for Target..., wybieramy zakładkę Debug, wybieramy interfejs debugowania ST-Link (o ile nie jest już wybrany) i naciskamy przycisk Settings umieszczony przy nazwie interfejsu. Po otwarciu dialogu ustawień interfejsu debugowania sprawdzamy i ew. korygujemy następujące ustawienia:

- W zakładce *Debug*: *Port: SW, Max. Clock: 480 kHz, Reset: SYSRESETREQ.*
- W zakładce *Flash Download*, ramka *Download Functions* – wybieramy *Erase Sectors* oraz zaznaczamy wszystkie trzy opcje z prawej strony ramki, w tym opcję *Reset and Run.*

Po zamknięciu dialogów konfiguracji łączymy interfejs ST-Link z modułem F103MDB, wykonując połączenia linii GND, SWDIO i SWCLK. Podczas programowania zasilamy płytkę z jej własnego interfejsu USB – linia oznaczona Vdd_TARGET na złączu interfejsu ST-Link umieszczonego na płytkach Discovery i Nucleo nie jest wyjściem napięcia zasilającego!

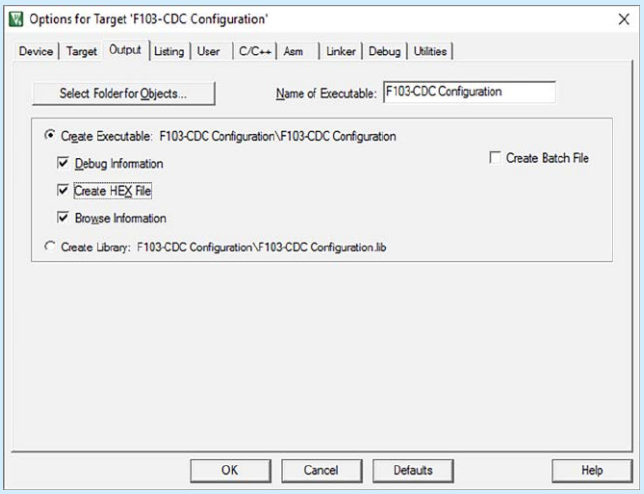
Po skonfigurowaniu oprogramowania i sprzętu możemy zaprogramować mikrokontroler, naciskając przycisk Load w pasku narzędzi środowiska Keil MDK-ARM.

Programowanie mikrokontrolera przy użyciu interfejsu USB-UART

Jeżeli nie dysponujemy interfejsem ST-Link, możemy zaprogramować mikrokontroler przy użyciu wbudowanego bootloadera, korzystając z interfejsu UART mikrokontrolera połączonego z dowolnym

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 33372, pass: 6nwd5fkt4

Tabela 1. Połączenia przejściówki UART z programowaną płytką F103MDB	
USB-UART	F103MDB
GND	GND
Vcc (3.3V)	3.3V
TXD	USART1_RX - PA10
RXD	USART1_TX - PA9



Rysunek 8. Włączanie generowania pliku .hex w Keil MDK-ARM.

modułem USB-UART, pracującym w standardzie logicznym 3.3V. Na czas programowania możemy zasilić mikrokontroler z modułu USB-UART. Wcześniej należy skonfigurować używany do programowania moduł do pracy w standardzie napięciowym 3.3 V.

W celu zaprogramowania mikrokontrolera STM32F103, należy ustawić na płytce F103MDB zworę BOOT0 w pozycję 0, pozostawiając zworę BOOT1 w pozycji 1. (Zwora BOOT1 znajduje się bezpośrednio przy przycisku RESET, BOOT1 – dalej od przycisku). Następnie wykonujemy połączenia opisane w **tabeli 1**.

Do programowania mikrokontrolera możemy użyć programu Flash Loader Demonstrator, udostępnianego przez ST Microelectronics na witrynie internetowej (plik zawierający program nosi nazwę STSW-MCU005.zip).

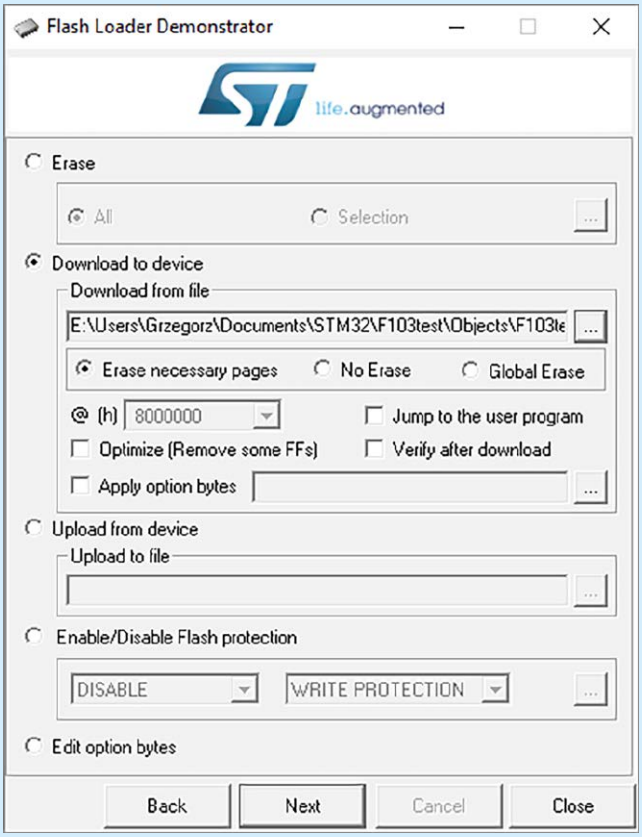
Program Flash Loader Demonstrator obsługuje pliki binarne oraz pliki w formacie Intel .hex i S-record. Należy więc skonfigurować środowisko programowania w taki sposób, aby wygenerowało ono odpowiedni plik – w Keil MDK-ARM należy w tym celu przed kompilacją zaznaczyć opcję *Create HEX file* w zakładce *Output* w dialogu ustawień projektu (**rysunek 8**).

Po zainstalowaniu i uruchomieniu programu Flash Loader Demonstrator, wybieramy właściwy interfejs szeregowy (numer portu COM przypisany do używanego interfejsu możemy sprawdzić w menedżerze urządzeń), pozostawiając wartości domyślne pozostałych ustawień. Po pomyślnym nawiązaniu komunikacji przechodzimy przez kilka kolejnych dialogów z informacjami o stanie układu, a następnie, w dialogu programowania (**rysunek 9**), wybieramy plik .hex zawierający postać ładowalną skompilowanego programu – jest on umieszczony w folderze MDK-ARM\F103-CDC Configuration. zawartym w głównym folderze projektu.

Po pomyślnym zaprogramowaniu mikrokontrolera odłączamy interfejs USB-UART i przywracamy wyjściową pozycję zwory BOOT0 – 0.

Sprawdzenie działania programu

Jeżeli używamy komputera z systemem Windows 10 lub Linux i zmodyfikowaliśmy deskryptor urządzenia USB CDC w sposób opisany powyżej, nasze urządzenie będzie obsługiwane



Rysunek 9. Dialog programowania mikrokontrolera w programie Flash Loader Demonstrator.

przez system operacyjny bez konieczności instalacji drivera. Jeśli używamy starszej wersji systemu Windows lub nie zmodyfikowaliśmy deskryptora urządzenia, musimy przed jego użyciem zainstalować driver wirtualnego portu szeregowego ST. W tym celu:

- Ściągamy z ***www.st.com*** plik STSW_STM32102.zip.
- Rozpakowujemy go i uruchamiając program VCP_Vx.x.x_Setup.exe instalujemy program do instalacji drivera. **Uwaga: nie powoduje to jeszcze zainstalowania drivera!**
- Używając eksploratora plików uruchamiamy odpowiednią dla naszego systemu operacyjnego wersję programu instalatora drivera, który standardowo znajdziemy w ścieżce *C:\Program Files (x86)\STMicroelectronics\Software\Virtual comport driver*.

Po przyłączeniu zaprogramowanej płytki F103MDB do komputera przewodem USB ze złączem microAB, komputer powinien wykryć i skonfigurować urządzenie jako wirtualny port szeregowy. Następnie uruchamiamy dowolny program terminala, np. TeraTerm i wybieramy nowo utworzony port szeregowy. Ustawione w programie terminala parametry transmisji nie mają wpływu na działanie urządzenia. Wprowadzone w terminalu znaki są przetwarzane i odsyłane zwrotnie – małe litery są zamieniane na wielkie i odwrotnie, pozostałe znaki nie są modyfikowane. Jednocześnie po wprowadzeniu każdego znaku zmienia się stan diody LED na płytce F103MDB.

Grzegorz Mazur

Bibliografia

1. RM0008 Reference manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM®-based 32-bit MCUs, ST Microelectronics 2015
2. STM32F103x8, STM32F103xB Datasheet, ST Microelectronics 2015