Użycie sterownika *direct* układów FTDI w środowisku LabVIEW, część 1

Na łamach Elektroniki Praktycznej kilkakrotnie pojawiał się temat układów interfejsowych (USB<->RS232) produkowanych przez firmę FTDI. Parokrotnie też opisywano sposoby komunikacji z nimi, wybierając do projektów najczęściej metodę intuicyjną, a więc sterownik VCP (Virtual COM Port). Teraz zajmiemy się sposobem użycia sterownika direct i dostarczanej wraz z nim biblioteki dll. Natychmiast nasuwa się pytanie o środowisko programistyczne, jakie zostanie wykorzystane do wywoływania funkcji biblioteki. Również w tej materii podejście będzie niecodzienne, a na pewno nie oczywiste, ponieważ do obsługi układu (a raczej dll) użyjemy środowiska LabVIEW…

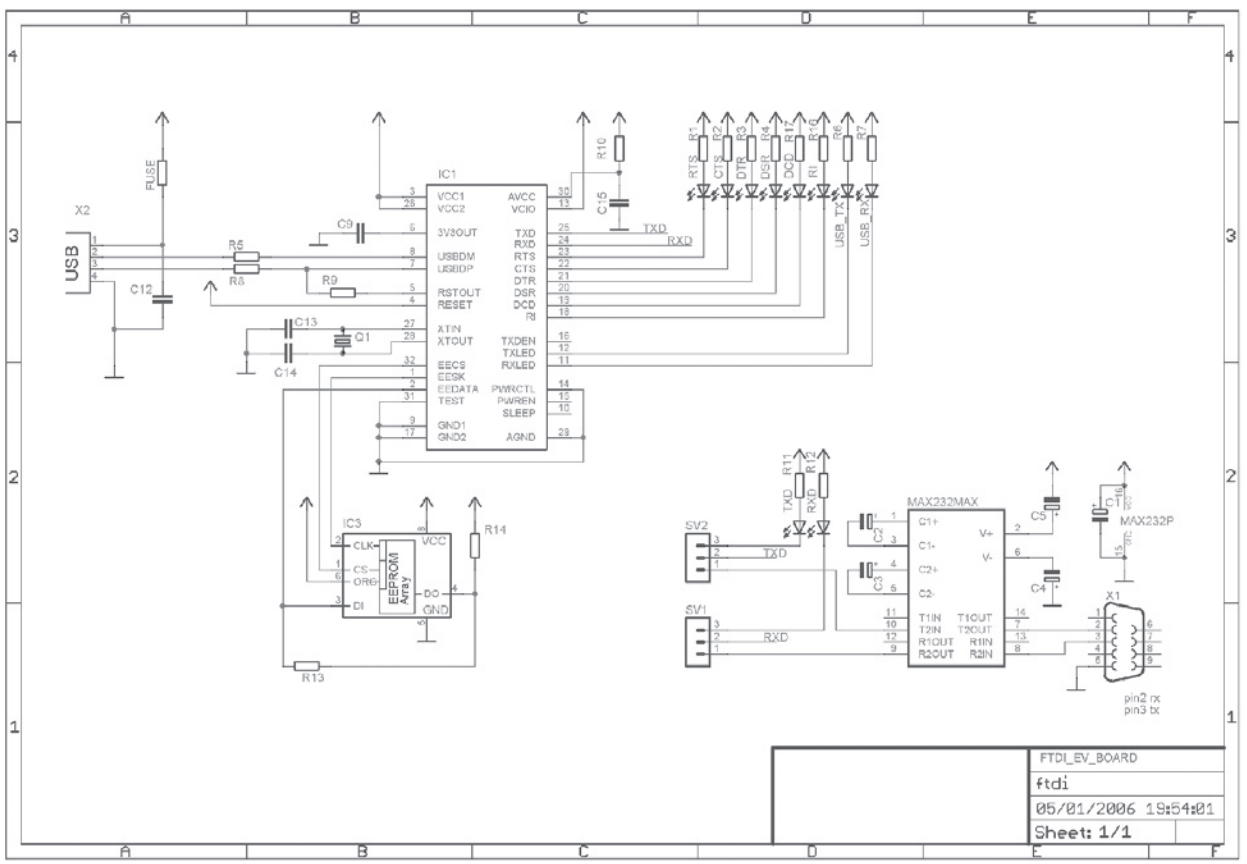
Zaczynamy

Na początku chcę zaznaczyć, że poniższy tekst nie jest kursem programowania. Zwięźle opiszę sposób obsługi pakietu LabVIEW w zakresie, jakiego wymagać będą prezentowane przykłady.

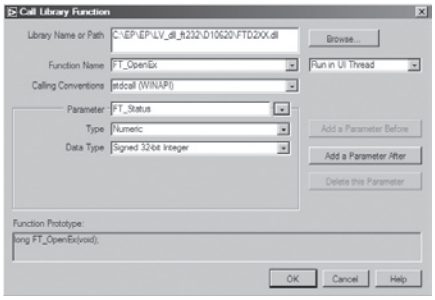
Na początek warto zaopatrzyć się w ewaluacyjną (darmową) wersję LabVIEW 7.0. Najnowsza wersja nosi numer 8.0, jednak z punktu widzenia artykułu i prezentowanych rozwiązań wersja 7.0 w zupełności wystarczy. Jest ona też wystarczająca do poznania specyfiki programowania w LabVIEW, a zdobyte umiejętności bez problemu będą mogły zostać wykorzystane w ewentualnej pracy z wersjami wyższymi.

LabVIEW 7.0 Evaluation jest dostępne do pobrania z serwera: *ftp://ftp.ni.com/evaluation/labview/pc/lv70_eval.exe*. Standardowa, intuicyjna instalacja nie powinna nikomu przysporzyć kłopotu.

Dobrze jest się także zaopatrzyć w moduł interfejsowy z układem FTDI. Przykładowe rozwiązanie przedstawiono na **rys. 1**. Jest to standardowa aplikacja FT232BM w wersji z zasilaniem z portu USB. Cześć „analogowa" układu również jest zasilona z 5 V (wyprowadzanie VCIO), co skutkuje tym, iż na wszystkich liniach I/O tego układu występują stany TTL (0…5 V).



Rys. 1. Schemat elektryczny modułu użytego w przykładach



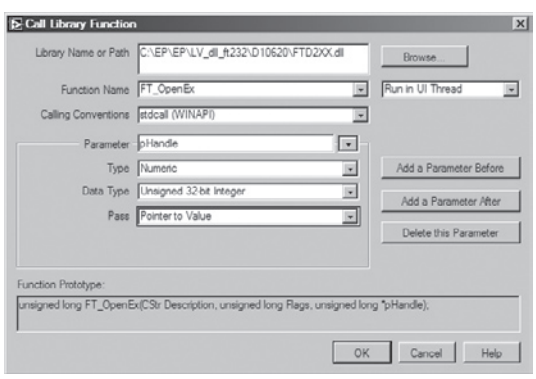
Rys. 2. Okno konfiguracji biblioteki FTD2XX.dll z prawidłowo skonfigurowanym typem zmiennej zwracanej przez funkcję FT_OpenEx

Przykładową aplikację zaopatrzono w 8 diod LED podłączonych do 8 wyprowadzeń FT232BM oraz w układ MAX 232, który będzie służył jako interfejs testowy podłączany do standardowego portu szeregowego komputera. Opcjonalnie można z niego zrezygnować i testować działanie programów np. z mikrokontrolerem. Pomimo, iż cały ten opis wygląda na nieco zawiły, sądzę, że wraz z kolejnymi krokami moje intencje staną się jaśniejsze.

Sterownik D2XX

Przed podłączeniem układu do komputera musimy zaopatrzyć się w odpowiednie sterowniki. Dostępne są one pod adresem *http://ftdichip.com/Drivers/D2XX.htm*. Należy pobrać odpowiednią dla posiadanego systemu wersję sterownika. Po rozpakowaniu, wewnątrz folderu znajdziemy oprócz plików instalacyjnych sprzętu *.inf* również plik *FTD2XX.dll*, czyli bibliotekę dołączaną dynamicznie, a funkcje, które zawiera będziemy wywoływać z poziomu LabVIEW.

Po podłączeniu układu do komputera, WinXP wykrywa kostkę FT232BM wyświetlając string zapisany w pamięci EEPROM. Przy pytaniu o lokalizację sterownika wybieramy plik *ftd2xx.inf*, po chwili w menadżerze urządzeń, w sekcji kontrolerów USB powinniśmy zobaczyć *FTU2XX Device*.



Rys. 3. Widok okna konfiguracji z kompletnym prototypem funkcji FT_OpenEx

LabVIEW

Idea programowania w języku G („potoczna" nazwa graficznego języka firmy National Instruments) opiera się na tworzeniu tzw. *virtual instruments*, a więc swego rodzaju przyrządów laboratoryjnych. Jest to konwencja przyjęta dawno temu i mająca wiele wspólnego z nazwą LabVIEW, sugerującą, iż środowisko przeznaczone jest do eksperymentów laboratoryjnych. Jednak obecnie, pomimo ciągłej zgodności z formą wirtualnego przyrządu, LabVIEW jest potężnym narzędziem developerskim używanym do tworzenia zarówno prawdziwych aplikacji uruchamianych na komputerach PC, jak również na dedykowanych platformach, przeznaczonych do wykonywania ściśle określonych zadań, np.: systemy czasu rzeczywistego. W naszym przypadku LabVIEW posłuży do interakcji z poziomu systemu operacyjnego ze światem zewnętrznym poprzez port USB. Sceptycy powiedzą, że to samo da się zrobić w każdym innym (konwencjonalnym środowisku, np. Delphi czy VISUAL C++) języku programowania, jednak postaram się udowodnić, że w LabVIEW, bez wielkiej wiedzy na jego temat, można zrobić to bardzo szybko. Jeśli ktoś chciałby w pierwszej kolejności poznać nieco samo LabVIEW i nabrać pewnej biegłości w programowaniu w tym środowisku, odsyłam do jednego z wielu tutoriali na jego temat znajdujących się w sieci (np. w portalu *www.labview.pl*).

Na początek postaramy się uzyskać tzw. uchwyt (*handle*) do układu FT232BM podłączonego do komputera. W tym celu na diagramie naszej aplikacji umieszczamy komponent „*Call library function node*" znajdujący się w palecie „*All Functions/Advanced*". W tym momencie strzałka jednokrotnego uruchomienia przybrała postać przełamanej, co wskazuje na to, iż umieszczony obiekt nie jest poprawianie skonfigurowany. Klikamy zatem na niego prawym przyciskiem myszy i wybieramy opcję *Configure*. W nowo otwartym oknie odnajdujemy pole *Library Name or Path*, które wypełniamy ścieżką dostępu do biblioteki *FTD2XX.dll*. Po kliknięciu w rozwijalną listę poniżej – *Function Name* - widzimy listę wszystkich dostępnych funkcji zawartych w bibliotece. Wybierzmy przykładowo funkcję FT_OpenEx (posłuży ona nam do uzyskania w/w uchwytu), w polu *Function prototype* (na dole okna) widzimy: *void FT_OpenEX(void);* Prototyp funkcji otwarcia oczywiście jest nieprawdziwy, ponieważ funkcja ta musi zwrócić status swojego wykonania (kod błędu), nie mówiąc już o tym, że funkcja może przyjmować jakieś parametry. Programista musi zatem sam zadbać o poprawność prototypu – jest to bardzo istotne, gdyż bez tego funkcja nie zostanie poprawnie wywołana, a element *Call Library Fuction Node* nie będzie posiadał prawidłowych wyjść i wejść (tzw. *vi terminals*) stanowiących odpowiedniki parametrów funkcji i wartości zwracanej w C. Deklaracje funkcji (prototypy) możemy znaleźć w pliku nagłówkowym *ftd2xx.h* dołączonym do sterownika *direct*. Po otwarciu pliku odnajdujemy:

```
FTD2XX_API
FT_STATUS WINAPI FT_OpenEx(
    PVOID pArg1,
    DWORD Flags,
    FT_HANDLE *pHandle
    );
```

Z punktu widzenia LabVIEW deklaracja zawiera wiele istotnych informacji, na podstawie których zbudujemy prototyp funkcji. Zacznijmy od wartości zwracanej ***FT_STATUS.*** Jest ona typu wyliczeniowego (*enum*), zdefiniowanego również w pliku FTD2XX.H:

```
typedef ULONG FT_STATUS;

enum {
   FT_OK,
   FT_INVALID_HANDLE,
   FT_DEVICE_NOT_FOUND,
   FT_DEVICE_NOT_OPENED,
   FT_IO_ERROR,
   FT_INSUFFICIENT_RESOURCES,
   FT_INVALID_PARAMETER,
   FT_INVALID_BAUD_RATE,

   FT_DEVICE_NOT_OPENED_FOR_ERASE,
   FT_DEVICE_NOT_OPENED_FOR_WRITE,
   FT_FAILED_TO_WRITE_DEVICE,
   FT_EEPROM_READ_FAILED,
   FT_EEPROM_WRITE_FAILED,
FT_EEPROM_ERASE_FAILED,
FT_EEPROM_NOT_PRESENT,
   FT_EEPROM_NOT_PROGRAMMED,
   FT_INVALID_ARGS,
   FT_NOT_SUPPORTED,
   FT_OTHER_ERROR
};
```

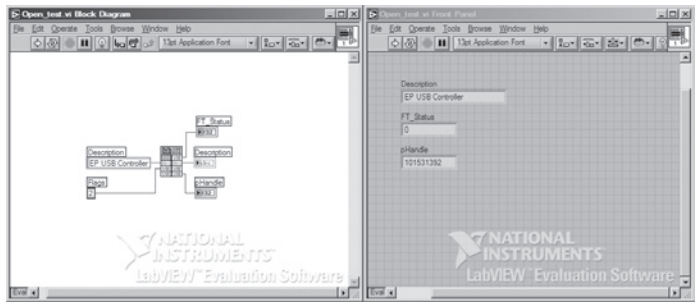
W ogólnym przypadku naszym pierwszym sukcesem będzie uzyskanie statusu *FT_OK*, a wiec wartości 0 zwróconej przez funkcję. LabVIEW oczywiście nie obsługuje domyślnie typu ***FT_STATUS***, zatem będziemy go musieli stworzyć samodzielnie. Ale o tym później. Na razie wystarczy nam informacja, że reprezentacją liczbową typu jest ULONG (*unsigned long*). Powracamy, zatem do okna konfiguracji komponentu. W pierwszej kolejności ustawiamy pole *Calling Conventions* na *stdcall(WINAPI)* – zgodnie z informacją zawartą w prototypie funkcji *FT_OpenEx*. Kolejnym krokiem jest pole *Parametr* z wartością *return type* (typ wartości zwracanej przez funkcję), klikając w pole możemy zmienić domyślną nazwę na *FT_Status*, natomiast w polu *Type* wybieramy *Numeric* (jako że wartość zwracana jest liczbą) o reprezentacji 32-bitowej (typ *long*) bez znaku (*unsigned*). Po tych zabiegach okno konfiguracyjne powinno wyglądać tak jak przedstawiono na **rys. 2**.

Następnie musimy zdefiniować parametry i ich typy przyjmowane prze funkcję. Jak widzimy w prototypie, pierwszym z nich jest ***PVOID pArg1***, którego interpretacja przez funkcję jest zależna od wartości drugiego z parametrów: ***DWORD Flags***, odpowiedzialnego za sposób lokalizacji układu (czy raczej za metodę wyszukania układu w systemie). Występują trzy wartości parametru *Flags* (typ *unsigned long*):

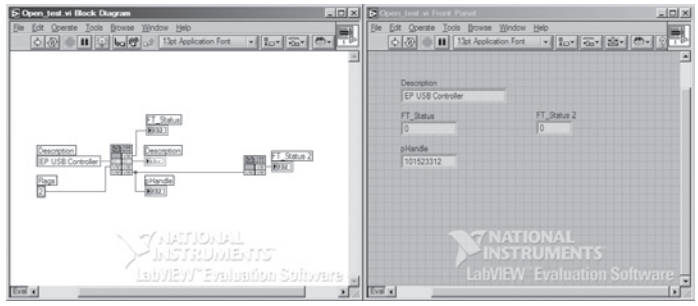
1. *FT_OPEN_BY_SERIAL_NUMBER* wartość 1 – parametr *pArg1* jest interpretowany jako wskaźnik na zmienną typu *string* reprezentującą numer seryjny układu FT232BM.
2. *FT_OPEN_BY_DESCRIPTION* wartość 2 - *pArg1* jest interpretowany jako wskaźnik na zmienną typu *string* reprezentującą nazwę układu. Nazwa ta jest zapisana w pamięci EEPROM dołączonej do układu. Tę właśnie flagę zastosujemy w dalszej części artykułu.
3. *FT_OPEN_BY_LOCATION* wartość 4 - *pArg1* jest interpretowany jako zmienna typu *long*, zawierająca lokalizację układu.

Trzecim parametrem, jaki należy zdefiniować jest wskaźnik ****pHandle*** na zmienną typu ***FT_HANDLE***. Krótko mówiąc, jest to miejsce w pamięci, do którego należy się odwołać chcąc wykonać jakiekolwiek operacje na układzie. Uchwyt ten jest wykorzystywany przez wszystkie kolejno używane przez nas funkcje. Jest on również typu *unsigned long*.

W celu dodania trzech w/w parametrów klikamy przycisk *Add a Parametr After* i konfigurujemy dla każdego z nich: nazwę, typ oraz reprezentację. W przypadku pierwszego parametru (nazwijmy go *Description*), po wybraniu typu *string* pojawi się pole *String Format* – wybieramy *C String Pointer*. W przypadku parametru *Flags* po wybraniu typu *Numeric* pojawi się pole *Pass* – stanowi ono wybór sposobu przekazania parametru przez wartość, bądź przez wskaźnik. Wybieramy wartość (*value*). Odwrotnie postępujemy w przypaku trzeciego parametru, wybierając w polu *Pass: Pointer to Value* (wskaźnik). Na **rys. 3** przedstawiono widok kompletnej konfiguracji funkcji wraz z poprawnym prototypem.



Rys. 4. Diagram oraz panel czołowy aplikacji



Rys. 5. Diagram oraz panel czołowy aplikacji, dodana funkcja zamykania połączenia z FT232BM

Po kliknięciu *OK* powracamy do diagramu naszej aplikacji. Wygląd komponentu nieco się zmienił, posiada on terminale o nazwach identycznych z tymi, jakie nadaliśmy wcześniej naszym parametrom. Przy terminalu *Description* tworzymy stałą (prawy klik – *Create Constant*) i wypełniamy ją nazwą taką, jaką nadaliśmy wcześniej układowi FT232BM podczas programowania EEPROM-u. Poniżej tworzymy stałą przy terminalu *Flags* i nadajemy jej wartość 2. Następnie tworzymy trzy wskaźniki (prawy klik – *Create Indicator*) z prawej strony komponentu. Najbardziej dla nas interesujące to *FT_Status* oraz *Handle*. Czynność ta spowoduje pojawienie się na panelu czołowym (*Front Panel*) tych właśnie pól. Widok diagramu i panelu przedstawiono na **rys. 4**. W tym momencie możemy zapisać stworzony przez nas *vi* pod dowolną nazwą oraz kliknąć przycisk jednokrotnego uruchomienia (strzałka w prawo). W rezultacie powinniśmy otrzymać wartość uchwytu do naszego urządzenia oraz kod błędu (*FT_Status*). Jeśli coś poszło nie tak, np. otrzymaliśmy kod błędu 6 (*FT_INVALID_PARAMETER*), oznacza to, że któryś z parametrów ma niewłaściwą wartość, np. występuje nieprawidłowa nazwa urządzenia. W przypadku, gdy otrzymaliśmy kod 0 wszystko wygląda dobrze i możemy przystąpić do dalszej zabawy.

Warto zauważyć, że jeśli po prawidłowym otwarciu urządzenia uruchomilibyśmy *vi* jeszcze raz, otrzymamy kod błędu 1 (*FT_INVALID_HANDLE*). Dzieje się tak dlatego, że nie zamknęliśmy połączenia do urządzenia za pomocą uzyskanego uchwytu. W tym celu należałoby umieścić na diagramie kolejny komponent *dll*, tym razem skonfigurowany do wywołania funkcji *FT_Close* służącej do zamknięcia połączenia. Dla ułatwienia na **rys. 5** przedstawiono diagram po wykonaniu tego zadania.

Marcin Chruściel, EP
marcin.chrusciel@ep.com.pl