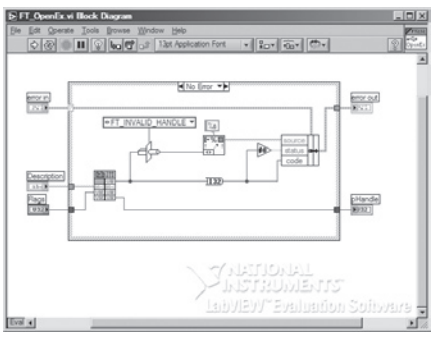
Użycie sterownika direct układów FTDI w środowisku LabVIEW, część 2

W pierwszej części artykułu omówiłem podstawy komunikacji LabView z układem FT232BM z wykorzystaniem biblioteki FTD2XX.dll. W tej części zaprezentuję przykład asynchronicznej transmisji danych do i z układu, z użyciem tzw. funkcji interfejsu klasycznego (Classic Interface Functions). Kolejnym krokiem będzie prosty program (vi) do zapisu i odczytu z pamięci EEPROM, z wykorzystaniem zestawu API (Application Programming Interface) EEPROM Programming Interface Functions. Na koniec zajmiemy się rozszerzeniem standardowego zestawu funkcji, umożliwiającego korzystanie z bardzo interesującego trybu pracy układu FT232BM – bitbang.

Wysyłamy łańcuch znaków…

Załóżmy że mamy urządzenie z interfejsem USB i programem sterującym (napisanym w LV oczywiście). Do komunikacji najwygodniej będzie opracować jakiś protokół składający się z zapytań i odpowiedzi. Dla czytelności protokołu, wszystkie jego komendy oraz parametry będą zapisane w formacie ASCII, mikrokontroler naszego urządzenia (sprzęgnięty z układem FT232BM za pomocą UART) będzie odbierał i analizował ciągi znaków nadawane z komputera. Jeśli w ciągu nadsyłanych znaków pojawi się znak nowej linii (LF – *line feed*, kod: 0x0A), będzie to informacja dla mikrokontrolera o tym, że komenda jest kompletna i może on przystąpić do jej analizy. Jeśli komenda jest prawidłowa, urządzenie podejmie określoną akcję, po czym wyśle do komputera odpowiedź (znów zakończoną znakiem LF). Tę prostą ideę komunikacji z użyciem USB zaimplementujemy teraz w LV.



Rys. 6. *Block Diagram,* czyli kod źródłowy *FT_OpenEx.vi*

Dla wygody i przejrzystości kodu, dla każdej użytej funkcji stworzymy osobny diagram *vi* (inna nazwa to *subvi*). Taki diagram stanowi w LV odpowiednik funkcji w klasycznych językach programowania. Występuje jednak kilka różnic. Najistotniejsza z nich jest taka, że *subvi* może zwracać, w odróżnieniu od funkcji w C, kilka parametrów. Zatem możliwe jest jednoczesne uzyskanie wartości odebranej z naszego urządzenia (w postaci *stringu*) oraz np. informacji o błędzie po pojedynczym wywołaniu *subvi*. W C musielibyśmy w tym celu przekazywać do funkcji adres bufora, w którym po wykonaniu się funkcji znaleźlibyśmy dane. Sama funkcja mogłaby, co najwyżej zwrócić strukturę zawierającą kod błędu i np. jego opis (a w praktyce ograniczylibyśmy się tylko do kodu błędu, czyli wartości liczbowej).

Skoro już o błędach mowa: w poprzedniej części analizowaliśmy wartość zwracaną przez funkcję *FT_OpenEx()*, która to wartość identyfikowała element typu *FT_STATUS*. Znów – dla wygody – stworzymy taki typ w LabVIEW. Z menu *File* wybieramy *New…*, następnie w drzewie po lewej stronie rozwijamy pole *Other Document Type* i wybieramy *Custom Control*. W nowo otwartym oknie w pasku narzędzi odnajdujemy pole *Control* i zmieniamy jego wartość, na Type *Def*. Klikając prawym przyciskiem myszy na pole naszej nowej kontrolki, umieszczamy obiekt *Enum* (typ wyliczeniowy) z palety *Ring & Enum*. Możemy teraz *zmienić* nazwę naszego nowego typu (np. na *FT_STATUS*) oraz zapisać całą kontrolkę (*File – > Save As…*). Klikając kursorem na obiekt *enum* możemy dodać pierwszą jego wartość. Wpisujemy, zatem *FT_OK*. Następnie klikamy prawym klawiszem myszy i wybieramy *Add Item After*, po czym wpisujemy kolejną możliwą wartość naszego nowego typu, czyli *FT_INVALID_HANDLE*. Powtarzamy tę czynność dla pozostałych wartości. Na sam koniec musimy dokonać wyboru reprezentacji liczbowej typu. W tym celu klikamy prawym przyciskiem i z menu *Representation* wybieramy U32 (*unsigned long*) i zapisujemy dokonane zmiany. W tym momencie mamy wszystko, co potrzebne do stworzenia przyzwoitego *subvi*, który będzie otwierał połączenie do FT232BM.

Na **rys. 6** widzimy kod *FT_OpenEx.vi*. Komponent *Call Library Function Node* skonfigurowany jest, tak jak poprzednio – przyjmuje jako parametry opis urządzenia oraz flagi otwarcia. Posiada również wejście *error in* , które podłączone jest do wejścia *select* struktury *case* (odpowiednik *case* w C), tworząc tym samym dwa możliwe przypadki wykonania kodu. Jeśli nasz *subvi* będzie wywołany z jakimś błędem na wejściu *error in*, wówczas wykonany zostanie przypadek *Error* i funkcja *OpenEx* nie zostanie w ogóle wywołana. W przypadku prawidłowego działania (bez błędu na wejściu), wykona się to, co widzimy na rys. 6. Standardowo błąd w LabVIEW jest definiowany jako tzw. klaster trzech wartości. Klaster jest odpowiednikiem struktury w C. Zawiera on w tym przypadku zmienną *status* typu BOOLEAN (1 – wystąpił błąd, 0 – nie było błędu), zmienną *code*

Tab. 1. Zestaw *subvi* do obsługi transmisji do i z FT232BM

Ikona	Nazwa	Funkcja
FT232 OpenEx	FT_OpenEx.vi	Omawiana poprzednio funkcja służąca do otwarcia połączenia do FT232BM
FT232 Reset	FT_Reset.vi	Zeruje układ
FT232 Set 8N1	FT_SetPar.vi	Ustala parametry transmisji, liczbę bitów, bit stopu, parzystość
FT232 Set RTS	FT_SetFlowCtrl.vi	Ustala kontrolę przepływu
FT232 BoudR	FT_SetBR.vi	Ustala prędkość transmisji
FT232 FlushB	FT_FlushBuff.vi	Opróżnia bufor nadawczy i/lub odbiorczy
FT232 Write	FT_Write.vi	Zapisuje łańcuch znaków (string) do układu (do bufora nadawczego)
FT232	FT_SeeQueue.vi	Zwraca liczbę bajtów oczekujących w buforze odbiorczym
FT232 Read	FT_Read.vi	Odczytuje zadana liczbę bajtów z bufora odbiorczego
FT232 Close	FT_Close.vi	Zamyka połączenie z układem

typu *integer* opisującą kod błędu (zerowy kod błędu oznacza powodzenie), oraz zmienną *source* typu *string* zawierającą opis błędu/źródło pochodzenia. Zatem wyjście *FT_Status* niosące informacje w postaci liczby o rezultacie operacji otwarcia urządzenia, posłużyło w tym przypadku wypełnienia wszystkich pól omawianej struktury. Pole *code* zostało wypełnione zwrócona liczbą przekonwertowaną do reprezentacji i32 I32 (32 bity ze znakiem). Pole *status* wypełniono za pomocą komponentu porównującego liczbę do zera ≠0. Jeśli liczba jest równa zeru, zwraca on wartość *FALSE*, natomiast w celu uzyskania opisu błędu posłużyłem się wcześniej zdefiniowanym typem *FT_STATUS* FT_INVALID_HANDLE, do którego zwrócona wartość została zrzutowana przy pomocy komponentu *Type Cast*, a następnie podana na wejście *Format Into String*, które jest odpowiednikiem funkcji *printf()* w C. Te trzy parametry zostały doprowadzone do (*Bundle By Name*), który modyfikuje zawartość *error in* i przekazuje je do wyjścia *error out*. W ten sposób został stworzony cały zestaw funkcji (*subvi*), który pozwoli nam na zapis i odczyt łańcucha znaków. Funkcje wraz z ich ikonami (stanowiącymi identyfikator w LV) przedstawiono w **tab. 1**.

Zestaw funkcji z tab. 1 stanowi niejako API FT232BM dla środowiska LabVIEW, przy użyciu którego został stworzony program obsługi transmisji pokazany na **rys. 7**.

Block Diagram natomiast przedstawia rys. 3. Sam program został napisany w konwencji maszyny stanów (odpowiedzialnej za operacje na układzie). Z kolei do obsługi interfejsu użytkownika użyto struktury *Event Structure*, której działanie oparte jest na zdarzeniach.

Po uruchomieniu, program stara się nawiązać połączenie z urządzeniem opisanym w polu *Description* (przypominam – opis ten powinien znajdować się w pamięci EEPROM dołączonej do układu FT232BM). Jeśli połączenie się powiodło, w polu *error out* zobaczymy *FT_OK*. Po utracie połączenia, (np. wskutek wyjęcia wtyczki z gniazda USB i ponownym włożeniu) możemy odnowić połączenie klikając przycisk *refresh*. Do sprawdzenia działania programu w praktyce, należy podłączyć interfejs UART układu FT232BM do wolnego portu COM komputera i uruchomić np. *Hyper Terminal* z żądanymi parametrami transmisji. Następnie w naszym programie ustalamy identyczne parametry, wypełniamy pole *String to write* łańcuchem znaków i klikamy przycisk *Write*. Zadany *string* powinien pojawić się w oknie *HT*. W drugą stronę – pisząc w aktywnym oknie *HT* – powinniśmy zobaczyć odbierane znaki. Program w każdej chwili, kiedy nie jest zajęty zapisem lub konfiguracją sprawdza bufor odbiorczy i jeśli są w nim nowe znaki – odczytuje je wyświetlając w polu *Read Buffer*. Menu rozwijane *View* służy do zmiany sposobu wyświetlania nadchodzących znaków. *Normal* – oznacza, że znaki

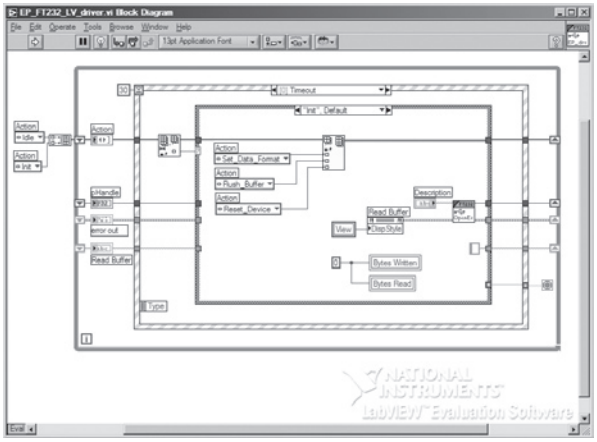
Rys. 7. *Front Panel* programu do zapisu i odczytu łańcucha znaków dla układu FT232BM

wyświetlane są jako ASCII z interpretacją znaków nowej linii, *codes* wyświetla ASCII oraz znaki nowej linii i powrotu karetki jako '*\n*' i '*\r*', HEX wyświela heksadecmalnie kody ASCII nadchodzących znaków. Nawigując po strukturze *case* (pole "Init", Default), można prześledzić, jakie funkcje są wywoływane do wykonania poszczególnych akcji programu.

API EEPROM

Zestaw funkcji do obsługi zewnętrznej pamięci EEPROM, (w której przechowywane są numery VID – *Vendor ID*, PID – *Product ID* oraz opis urządzenia) umożliwia zapis odczyt kasowanie oraz sprawdzanie rozmiaru pamięci. Został on podzielony na trzy grupy:

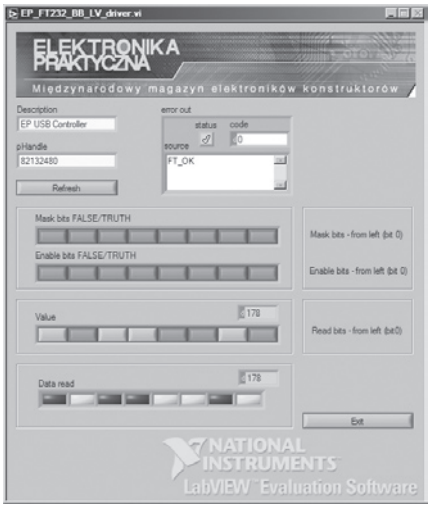
1. Funkcje ogólnego dostępu: *FT_WriteEE()*, *FT_ReadEE()*, *FT_EraseEE()* – oferują obsługę całego obszaru pamięci EEPROM.
2. Funkcje obsługujące dane używane przez FT232BM podczas enumeracji USB: *FT_EE_Read()*, *FT_EE_ReadEx()*, *FT_EE_Program()*, *FT_EE_ProgramEx()*. Za pomocą tych funkcji możemy szybko stworzyć aplikacje podobną do MProg. Funkcje dbają o umieszczanie właściwych danych pod właściwymi adresami w pamięci.
3. Funkcje tzw. obszaru użytkownika (*User Area*): *FT_EE_UARead()*, *FT_EE_UAWrite*, *FT_EE_UASize()*. Pozwalają na zagospodarowanie nieużywanego obszaru pamięci.



Rys. 8. *Block Diagram* aplikacji (kod programu)

Funkcja *FT_EE_UASize* zwraca ilość wolnego miejsca w pamięci do wykorzystania przez użytkownika. Może ono posłużyć np. do zapisywania ustawień urządzenia.

Do ostatniej grupy funkcji przygotowałem zestaw trzech *vi* (**tab. 2**), za pomocą których stworzona została aplikacja, której *Front Panel* jest widoczny na **rys. 9**.

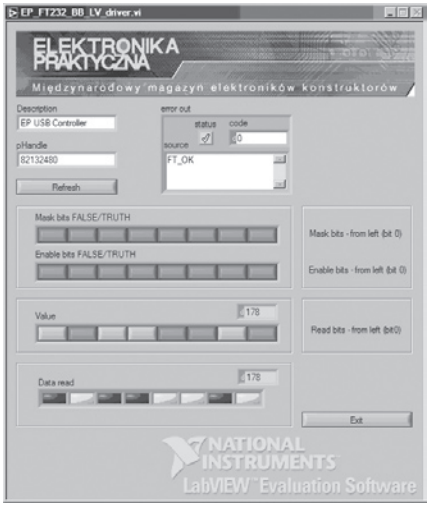


Rys. 9. Aplikacja umożliwiająca zapis i odczyt do i z obszaru użytkownika zewnętrznej pamięci EEPROM

BitBang

Na koniec pozostawiłem dwie najbardziej interesujące (według mnie) funkcje z całego zestawu API: *FT_SetBitMode()* i *FT_GetBitMode()*. Pozwalają one na zmianę funkcji linii obsługujących interfejs transmisyjny RS232 na 8–bitowy, dwukierunkowy port równoległy.

FT_SetBitMode() oprócz uchwytu do urządzenia, przyjmuje dwa dodatkowe argumenty w reprezentacji *uchar* (a wiec 8 bitów). Pierwszy – *ucMask* zawiera informację, która linia będzie zdefiniowana jako wejście, a która jako wyjście. Jeśli na pozycji bitu 0 ustawimy 1, wówczas linia *Data 0* będzie pracowała jako wyjście. Jeśli ustawimy ten bit na wartość 0, *Data 0* będzie pracować jako wejście. Drugi argument – również w reprezentacji *uchar* – niesie informacje o tym, które z wyjść mają zostać ustawione w tryb BitBang. Do odczytu stanu całego portu służy funkcja *FT_GetBitMode()*. Przyjmuje ona jako parametr wskaźnik do zmiennej typu *uchar*, w której przechowany zostanie właśnie odczytany stan portu. Jeśli linie skonfigurowane są jako wyjścia, funkcja zwróci ostatni ustawiony na nich stan. Przed zapisaniem wartości do takiego portu równoległego (za pomocyą *FT_Write.vi* – tego samego który służył do zapisu „szeregowego") należy ustawić prędkość transmisji (tak jak dla połączenia szeregowego). Maksymalna prędkość dla trybu BitBang wynosi 3 MB/s (zegar dla BitBang odpowiada szesnastokrotności prędkości bodowej). W celu praktycznego sprawdzenia



Rys. 10. Front Panel uruchomionej aplikacji wykorzystującej tryb BitBang

działania takiego portu USB możemy do wyjść układu podłączyć kilka diod LED i z pomocą aplikacji przedstawionej na **rys. 10** zmieniać ich stan (za pomocą przycisków oznaczonych jako *Value*).

Tryb ten w założeniu tworców FT232BM miał służyć do konfigurowania układów FPGA, jednak może znaleźć dużo więcej „banalnych", ale jakże potrzebnych zastosowań. Mam na myśli choćby prosty interfejs do akwizycji danych bądź sterownik kilku przekaźników. Po wyposażeniu układu w zewnętrzny rejestr przesuwny, uzyskujemy praktycznie nieograniczoną liczbę wyjść cyfrowych. Po dołączeniu przetwornika A/C z interfejsem SPI możemy rejestrować sygnały analogowe. Możliwa jest też realizacja różnego rodzaju magistral (choćby I²C). Jeśli dodać do tego łatwość przebudowy aplikacji sterującej, jaką daje LabVIEW, otrzymujemy w efekcie świetne narzędzie do szybkiego łączenia komputera ze światem zewnętrznym – inteligentnego (mamy przecież program) i nowoczesnego (USB 2.0) przesyłaia danych.

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

Tab. 2. Zestaw vi służący do obsługi obszaru użytkownika pamięci EEPROM

Ikona	Nazwa	Funkcja
FT232 EE UAwrite	FT_EE_UAWrite.vi	Zapis do EEPROM. Funkcja przyjmuje jako parametr tablice bajtów, które są z tej tablicy przepisywane pod kolejne komórki w pamięci począwszy od pierwszej przeznaczonej na obszar użytkownika
FT232 EE UAread	FT_EE_UARead.vi	Odczyt EEPROM. Funkcja przyjmuje jako parametr ilość bajtów do odczytu począwszy od pierwszego adresu UA
FT232 EE UAsize	FT_EE_UA_Size.vi	Funkcja zwraca rozmiar dostępnego obszaru użytkownika w bajtach. Wynik zależy od wielkości zastosowanej pamięci

Tab. 3. Vi do obsługi trybu BitBang

Ikona	Nazwa	Funkcja
FT232 BB S 0101	FT_BB_Set.vi	Ustala funkcje linii I/O
FT232 BB G 0101	FT_BB_Get.vi	Zwraca stan linii I/O