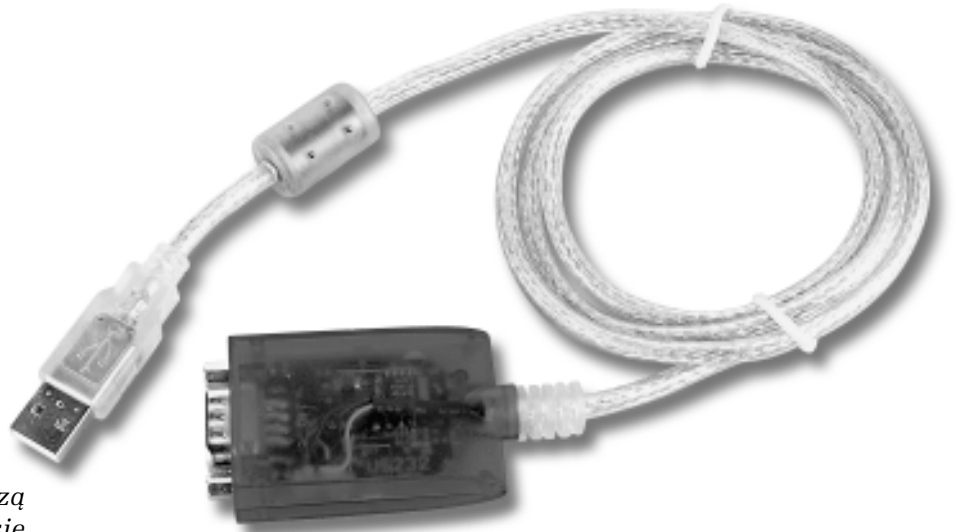


# Konwerter USB<->RS232, część 1

## AVT-5080



*Przedstawiamy pierwszą w polskiej prasie elektronicznej aplikację układu, który - jesteśmy tego pewni - zrewolucjonizuje współczesną elektronikę popularną. Jest to scalony, dwukierunkowy konwerter USB<->RS232, za pomocą którego można dołączyć do PC-ta lub McIntosha dowolne urządzenie zewnętrzne i to bez konieczności pisania własnych driverów.... Same zalety!*

Już od długiego czasu dobiegają zewsząd głosy, że najwyższa pora, aby w amatorskich projektach zagościły interfejsy USB. Tym bardziej, iż w świecie akcesoriów komputerowych stały się one już standardem. Producenci nie ustają w wysiłkach, żeby wdrażać nowe, coraz szybsze wersje tego interfejsu, natomiast dotychczasowe porty szeregowy i równoległe mają wkrótce zniknąć z „pokładu” PC, dzieląc los napędów 5 1/4” czy magistrali ISA.

Dotychczas jednak wysoka złożoność systemu USB w połączeniu z niewielką dostępnością potrzebnych podzespołów dla indywidualnego klienta oraz całkowitym brakiem polskojęzycznej literatury stawiała przed elektronikami-amatorami barierę trudną do pokonania.

Na rys. 1 pokazano strukturę połączenia USB. Pozwala ona ocenić, ile wyspecjalizowanych mechanizmów kryje się np. za najzwyklejszym zapaleniem LED-a na własnej prototypowej płytce za pomocą samodzielnie napisanej aplikacji dla PC. Rysunek ten jest tylko orientacyjny - na jego podstawie bardzo ogólnie opiszemy zasady działania USB (pełne specyfikacje to co najmniej kilkaset stron - są dostępne na stronie [www.ep.com.pl](http://www.ep.com.pl) w dziale *Download>Dokumentacje*). Zaczniemy od strony sygnałów przesyłanych kablem.

Standardowy kabel połączeniowy USB ma 4 żyły: dwie z nich

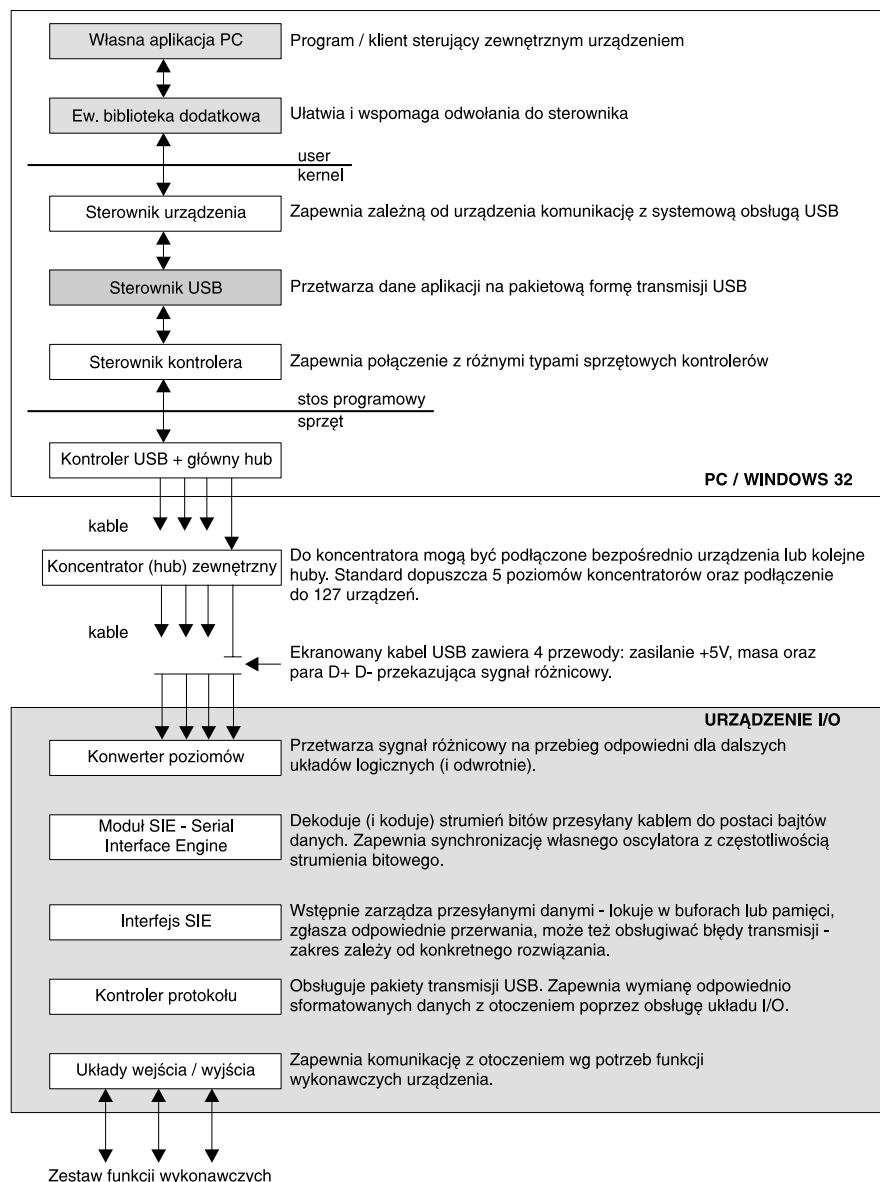
przekazują zasilanie (masa oraz +5V), następne dwie to para przenosząca różnicowy sygnał danych. Sygnał ten pełni kilka jednoczesnych funkcji:

- przesyła dane w postaci strumienia bitów,
- synchronizuje oscylatory podłączonych urządzeń z oscylatorem głównego kontrolera (PC),
- wyznacza początek i koniec pakietu,
- przekazuje żądanie programowego zerowania,
- detekcja dołączenia urządzenia do magistrali.

Strumień bitów jest kodowany w systemie NRZI: poszczególnym sekwencjom bitów odpowiada zmiana lub brak zmiany poziomu (czyli w parze różnicowej zmiana polaryzacji lub jej pozostawienie - **rys. 2**):

- ciąg zer przełącza poziom przy każdym bicie,
- ciąg jedynek pozostawia poziom bez zmian,
- para 1 0 przełącza poziom,
- para 0 1 pozostawia poziom bez zmiany.

Magistrala nie posiada oddzielnego sygnału zegarowego. Każde podłączone urządzenie dysponuje własnym oscylatorem do sprawdzania kolejnych bitów z 4-krotnym oversamplingiem (nadprób-kowaniem) - dla szybkości przesyłu danych wynoszącej 12 Mbd konieczny będzie więc zegar o częstotliwości 48 MHz. Oscylator jest synchronizowany (pętla PLL) z przebiegiem sygnału w ma-



Rys. 1. Budowa połączenia USB

gistrali. Dłuższy brak zmiany poziomu sygnału (występujący w kodzie NRZI przy ciągu kolejnych bitów „1”) może spowodować zerwanie tej synchronizacji. Dlatego wprowadzono rozdzielanie ciągu jedynek dodatkowym bitem „0” (po każdym 6 jedynek) - tzw. *bit stuffing* - **rys. 3**.

Wszelkie informacje USB są przesyłane w formie pakietów. Początek pakietu jest oznaczony jako pierwsza zmiana poziomu przy wyjściu ze stanu spoczynkowego (*idle*) magistrali. Potem następuje blok synchronizacji (ciąg bitów 0 zakończony 1), następnie blok danych. Koniec pakietu jest sygnalizowany dwubitowym stanem SE0 (skrót od *single-ended 0* - obie linie pary różnicowej na

poziomie niskim). Przebieg sygnałów dla pakietu jest pokazany na **rys. 4**.

Struktura pakietu zawsze zawiera blok synchronizacji oraz zaraz po nim bajt identyfikatora pakietu - PID (4 bity określają PID, następne 4 są negacją PID, co pozwala na kontrolę poprawności). Dalsza zawartość zależy od typu pakietu (może mieć długość od 0 do 1025 bajtów). Stosowane pakiety przedstawiono w **tab. 1**. Z wymienionych pakietów składane są wszystkie, występujące w USB, elementarne transakcje. Może to być np. transakcja wysłania danych z komputera:

- pakiet OUT - do urządzenia (przygotuj się do otrzymania

danych przeznaczonych dla EPx),

- pakiet DATA0 - do urządzenia (dane),

- pakiet ACK - z urządzenia (pakiet danych przyjęty prawidłowo),

albo odczytu danych przez komputer:

- pakiet IN - do urządzenia (wysłaj pakiet danych przez EPx),

- pakiet DATA0 - z urządzenia (urządzenie wysła dane),

- pakiet ACK - do urządzenia (pakiet danych dotarł prawidłowo).

Elementarna transakcja musi zawierać się w obrębie pojedynczej 1-ms ramki.

Z kolei odpowiednio zgrupowane zespoły transakcji tworzą cztery różne typy transferów. Transfery różnią się poza tym sposobem obsługiwanym przez komputer w zakresie kontroli błędów i przyznawania priorytetów w dostępie do magistrali:

1. **Control** - używany do rozpoznawania i konfigurowania urządzenia.

2. **Interrupt** - używany do sprawdzania statusu urządzenia i wykrywania jego ewentualnych żądań obsługi. Termin „przerwanie” w nazwie jest nieco mylący, zwłaszcza dla elektroników przyzwyczajonych do systemu przerwania w mikrokontrolerach. Urządzenie USB samo z siebie nie może rozpocząć żadnego transferu, może jedynie przesłać na życzenie *hosta* odpowiednią informację. Komputer sprawdza cyklicznie dołączone urządzenia (*polling*) i w razie potrzeby podejmuje odpowiednią akcję. Częstotliwość *pollingu* jest ustalana w trakcie początkowej konfiguracji, ale nie może przekraczać częstotliwości ramek (1 kHz). Wszelkie działania wymagające szybszej reakcji na sygnały zewnętrzne powinny być więc realizowane samodzielnie przez urządzenie.

3. **Bulk** - używany do przesyłania dużych bloków danych (ang. *bulk* - wielkość, objętość, masa) z dokładną kontrolą poprawności - stosowany np. w drukarkach czy skanerach.

4. **Isochronous** - stosowany do przesyłania ciągłego strumienia danych w przypadkach gdzie poprawność jest mniej istotna niż

Tab. 1. Zestawienie pakietów stosowanych w USB

PID	Typ	Kategoria	Zawartość i funkcja
0101	SOF	token (kontrolny)	Start-of-frame: jest wysyłany co 1 ms (zapewnia to dodatkową synchronizację oraz pozwala na wykrywanie zawieszenia magistrali). Składa się z 11-bitowego kolejnego numeru ramki (frame) oraz 5-bitowego kontrolnego CRC.
1101	SETUP	token (kontrolny)	Priorytetowy pakiet kontrolny służący do podstawowego nadzoru i konfiguracji - składa się z 7-bitowego adresu urządzenia, 4-bitowego adresu EPO i 5-bitowego CRC.
1001	IN	token (kontrolny)	Inicjalizuje przesył danych z urządzenia do PC - skład jw. z tym, że może być adresowany dowolny dostępny w urządzeniu endpoint (EP x).
0001	OUT	token (kontrolny)	Inicjalizuje przesył danych z PC do urządzenia. Skład jw.
0011	DATA0	dane	Pakiet składa się z ciągu 0 - 1023 bajtów danych oraz 16-bitowego kontrolnego CRC. Dwa typy pakietu są wprowadzone dla dodatkowej kontroli: nadajnik wysyła przemiennie typ 0 i 1, co umożliwia sprawdzanie, czy żaden pakiet nie zaginął.
1011	DATA1	dane	
0010	ACK	handshake	Potwierdzenie prawidłowego odbioru - pakiet zawiera tylko identyfikator.
1010	NAK	handshake	Zgłoszenie chwilowej zajętości urządzenia - transakcja jest ponawiana w następnej ramce. Urządzenie nie ma prawa odpowiedzieć NAK na pakiet SETUP - musi wtedy przerwać wszelkie inne operacje i obsłużyć transakcję kontrolną.
1110	STALL	handshake	Zgłoszenie braku możliwości obsługi konkretnej komendy (np. wpisu danych do nieistniejącego w urządzeniu endpointu).
1100	PRE	specjalny	Używany przy konfigurowaniu połączenia małej prędkości (1,5 Mb/s).
INNE	rezerwa	rezerwa	-

zachowanie stałego w czasie (*izochronicznego*) przepływu. W tym celu już na poziomie elementarnej transakcji pominięta jest kontrola błędów i pakiety potwierdzeń oraz używa się maksymalnego rozmiaru pakietów danych. Obszar wykorzystania to przede wszystkim urządzenia audio i wideo. Przykładem urządzenia korzystającego z izochronicznego trybu przesyłania danych jest karta dźwiękowa USB, którą opisaliśmy w EP3/99.

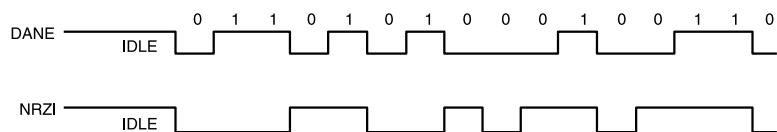
Aby dalej omawiać wykorzystanie tych transferów, należy się zapoznać z logiczną strukturą połączenia USB. Jest ona ogólnie przedstawiona na rys. 5. Podstawowym pojęciem jest tutaj *endpoint* (już wspomniany w opisach pakietów) - czyli punkt końcowy. *Endpoint* jest zakończeniem logicznego kanału przesyłania danych w stronę komputera - hosta (EP IN) lub w stronę urządzenia (EP OUT). Drugi koniec takiego kanału (nazywanego rurką - *pipe*) jest wejściem lub wyjściem danych dla oprogramowania uruchomionego na *hoście*. Każde - nawet najprostsze - urządzenie musi posiadać co najmniej jeden *endpoint*: EPO (IN oraz OUT) o jednoznacznie określonej charakterystyce, który obsługuje podsta-

wowe transfery kontrolne. Dalsze *endpoints* mogą już mieć charakterystyki dopasowane do potrzeb realizowanej funkcji (np. maksymalny rozmiar pakietu, typ transferu itd.). Zestaw *endpointów* tworzy interfejs.

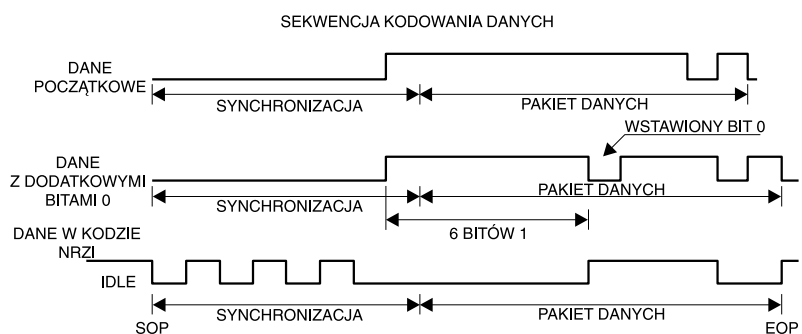
W bardziej złożonych urządzeniach interfejsów może być kilka (każdy odpowiada za określony wycinek funkcjonalności urządzenia - pozwala to na uzyskanie znacznego skomplikowania przy użyciu mniejszych, już sprawdzo-

nych i często typowych elementów składowych). Zbiór interfejsów tworzy z kolei konfigurację, których też może być kilka (ale tylko jedna może być aktywna).

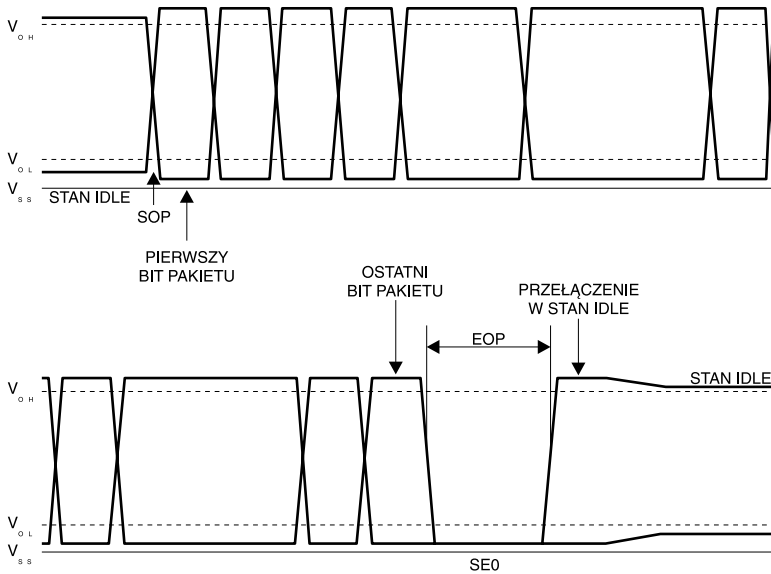
Jak widać na rys. 5, klient (czyli oprogramowanie korzystające z obsługi USB oferowanej przez system operacyjny), który dostaje do dyspozycji logiczne kanały komunikacji z interfejsem urządzenia USB, nie musi nic „wiedzieć” o wcześniej opisywanych mechanizmach wykonawczych niższego poziomu. W systemie Windows oprogramowanie klienta obejmuje sterownik (*driver*) konkretnego urządzenia (lub klasy urządzeń zbliżonych funkcjonalnie) oraz aplikację użytkownika. Sterownik - przypisany do interfejsu urządzenia USB - potrafi go skonfigurować według konkretnych możliwości i potrzeb oraz odpowiednio się z nim komunikować. Jednocześnie umożliwia stosowanie w aplikacji użytkownika typowych dla systemu funkcji wejścia/wyjścia danych. Oprogramowanie może też obejmować dodatkowe biblioteki udostępniające rozmaite komendy obsługi oraz konfiguracji urządzenia w postaci zestawu czytelnych i prostych w korzystaniu funkcji. Natomiast uniwersalny sterownik systemowy USB potrafi się skomunikować z *endpointem* 0 dowolnego urządzenia. Jest to niezbędne w chwili podłączenia do magistrali, kiedy system operacyjny nie ma jeszcze żadnych danych o dołączanym urządzeniu i potrzebuje wspólnego, dokładnie



Rys. 2. Kodowanie NRZI



Rys. 3. Bit stuffing



Rys. 4. Przebieg sygnału podczas przesyłu pakietu danych (SOP - start of packet, EOP - end of packet)

określonego mechanizmu pobrania odpowiednich informacji oraz dokonania wstępnej konfiguracji. Magistrala cały czas kontroluje dołączanie i odłączanie urządzeń - możemy to robić w dowolnym momencie.

W wolnym porcie *downstream* (dane płynące z *hosta* w kierunku urządzeń) huba obie linie różnicowe D+ i D- są połączone z masą rezystorami 15 kΩ i mają poziom niski (stan urządzenia: **nieprzyłączone**). Dołączany układ podaje na linię D+ napięcie zasilania przez rezystor 1,5 kΩ. W ten prosty sposób hub dowiadyuje się, że konkretny port ma już „klienta” i wpisuje to do swojego rejestru stanu (stan urządzenia: **przyłączone**). Hub dostarcza też dla urządzenia zasilanie do 100 mA (stan urządzenia: **zasilone**) - większy pobór prądu powoduje błąd i przerwanie konfigurowania.

Komputer - *host* kontroluje cyklicznie stan wszystkich podłączonych hubów. Po stwierdzeniu podłączenia urządzenia rozpoczyna proces tzw. enumeracji, czyli:

- Wydaje hubowi polecenie wyzerowania linii, polegającego na ustawieniu przez co najmniej 10 ms stanu SE0 w porcie obsługującym nowy przyrząd. Urządzenie w odpowiedzi na zerowanie przełącza się w stan **domyślny** (*default*) i przyjmuje adres 0, pod którym będzie widoczne dla hosta w początkowej fazie enumeracji (tyl-

ko jedno urządzenie może być w stanie domyślnym - każde następne musi poczekać, nie grozi więc przypadkowy konflikt adresów).

- *Host* wysyła pod adres 0 żądanie przesłania deskryptora urządzenia (bloku danych o ściśle określonym formacie opisujących konkretny przyrząd).
- Przyznaje kolejny wolny adres i wysyła do urządzenia - które go zapisuje i od tego momentu będzie odbierać tylko pakiety z właśnie takim polem adresu (stan urządzenia: **zaadresowane**).
- Ponawia żądanie przesłania deskryptora urządzenia - ale już z nowym adresem. Zwrócony deskryptor jest porównany z otrzymanym poprzednio, co pozwala skontrolować prawidłowość zaadresowania.
- Pobiera wszystkie dalsze deskryptory opisujące dostępne w urządzeniu konfiguracje, interfejsy i *endpointy*.

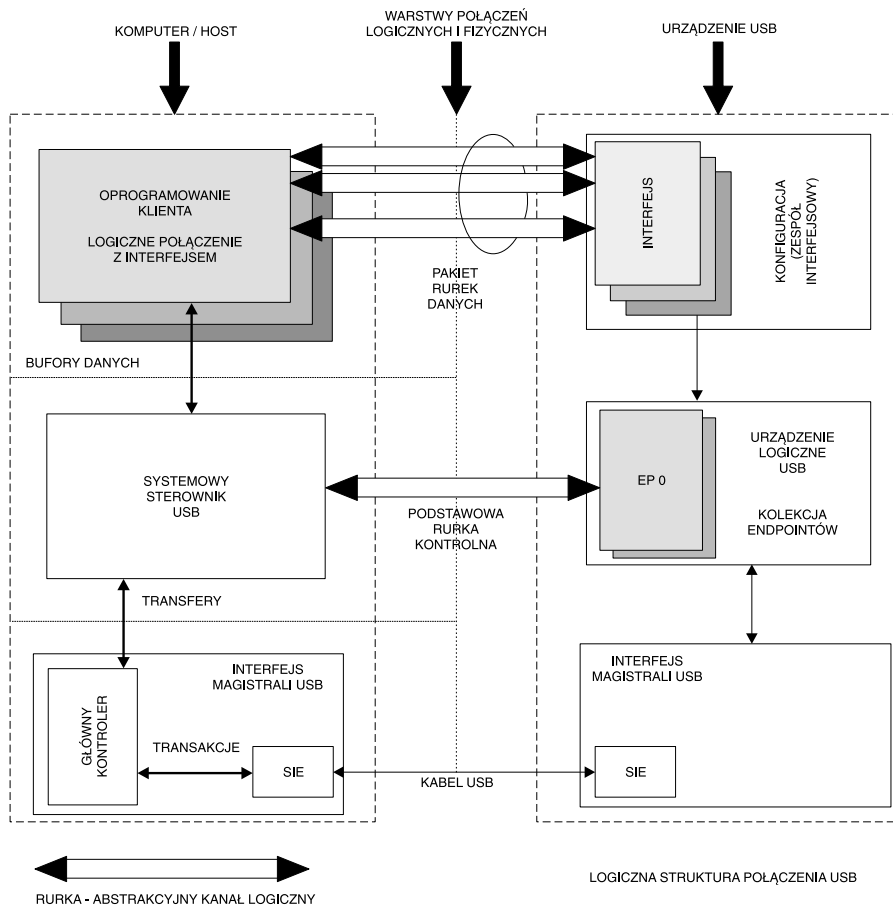
Do tej pory wykonywane były operacje wspólne dla wszystkich urządzeń USB. Dlatego mógł je zrealizować uniwersalny sterownik systemowy. Nie jest jednak możliwe, aby zakodować w nim pełną obsługę urozmaiconej gamy przyrządów. W systemie Windows rolę taką pełni sterownik urządzenia (*device driver*), napisany specjalnie dla potrzeb konkretnego rozwiązania technicznego. Przejmuje on dalszy proces konfiguracji oraz zapewnia później-

szą komunikację z urządzeniem zgodnie z konkretnymi, specyficznymi potrzebami. System na podstawie danych w deskrytorze urządzenia wybiera i podłącza potrzebny sterownik, ładując go w razie potrzeby do pamięci. Jeśli sterownik jeszcze nie jest zainstalowany w systemie - użytkownik jest proszony o jego dostarczenie.

- Następuje wysłanie serii ostatnich poleceń konfiguracyjnych - po ich przyjęciu urządzenie jest gotowe do normalnej pracy (stan: **skonfigurowane**).

Enumeracja opiera się na szeregu transferów typu *control* wymienianych zawsze z *endpointem* 0. Transfery te są stosunkowo skomplikowane. Składają się z trzech faz:

- Faza przygotowawcza (*setup*). Tworzy ją transakcja zbudowana z pakietu SETUP, 8-bajtowego pakietu danych DATA0 oraz potwierdzenia ACK (jak już wspomniano, urządzenie musi priorytetowo obsłużyć tę transakcję i nie może jej odkładać na później odsyłając brak gotowości NAK). W pakiecie danych zakodowany jest rodzaj operacji, jaką *host* chce przeprowadzić, m.in.:
  - kierunek przesyłu informacji,
  - rodzaj żądania (*standard* - standardowe; *class* - dotyczące klasy urządzeń, np. hubów,
  - *vendor* - związane z konkretnym urządzeniem),
  - temat żądania - precyzuje, o jaką informację i jaką reakcję chodzi (np. wśród standardowych wywołań znajdują się *Get\_Status* - pobranie stanu urządzenia czy *Get\_Descriptor* - pobranie odpowiedniego opisu),
  - przeznaczenie żądania (może dotyczyć urządzenia, interfejsu lub *endpointu*).
- Faza wymiany danych (opcjonalnie - o ile jest taka potrzeba): odpowiednia liczba transakcji wysłania lub pobrania danych określonych w fazie *setup*.
- Faza statusu - służy do potwierdzenia poprawnego zakończenia transferu. Tworzy ją transakcja wysłania lub odczytania danych z użyciem pakietu danych o zerowej długości.



Rys. 5. Logiczna struktura połączenia USB

### Opis budowy interfejsu-konwertera USB<->RS232

Powyższe - bardzo skrócone - omówienie zasady pracy magistrali USB pozwala ocenić nakład pracy wymagany przy przygotowaniu własnego układu. Dodatkowym utrudnieniem jest niewielka dostępność na rynku detalicznym odpowiednich elementów - dostawy są ukierunkowane na wielkoseryjną produkcję akcesoriów komputerowych. Do tej pory w opisywanych rozwiązaniach amatorskich najczęściej występował EZ-USB - mikrokontroler zgodny z rodziną '51, oraz PDIUSB11 - interfejs SIE wyposażony w magistralę I<sup>2</sup>C.

EZ-USB ma wbudowany *loader*, który po włączeniu urządzenia sam przeprowadza pierwszą enumerację, a następnie ładuje do obszaru pamięci kodu program pracy mikrokontrolera przesyłany przez współpracujący po stronie *hosta* sterownik. Po zakończeniu tego procesu magistrala jest zerowana i wykonywana jest ponowna enumeracja (zwana w związku z tym przez producenta *renume-*

*racją*) - teraz już konfigurująca układ jako docelowe, zgodne z napisanym przez nas programem urządzenie, które współpracuje z odpowiednim, dedykowanym sterownikiem. Takie rozwiązanie czyni z EZ-USB znakomite narzędzie do wszelkich eksperymentów i uruchomień oraz do tworzenia wielofunkcyjnych przyrządów.

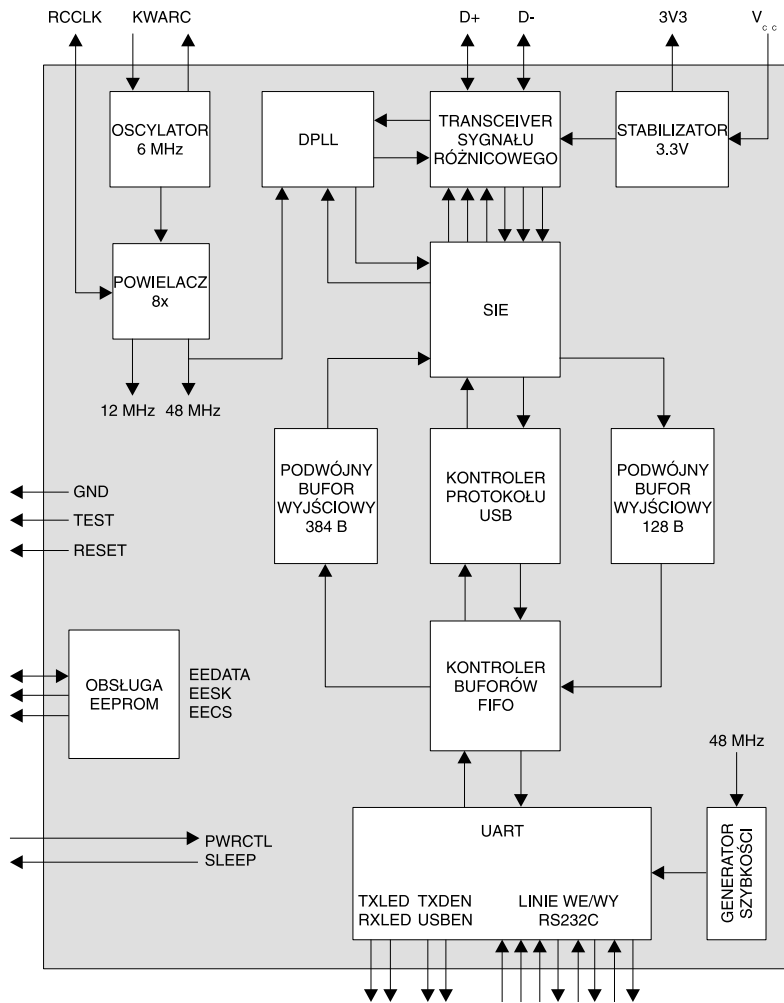
PDIUSB11 nie posiada możliwości samodzielnego działania - musi współpracować z zewnętrznym mikroprocesorem, którego program zapewnia obsługę poszczególnych transakcji. Komunikacja z kostką odbywa się za pośrednictwem interfejsu I<sup>2</sup>C, który ma wprawdzie wysoką maksymalną częstotliwość (do 1 MHz), ale i tak znacznie ogranicza przepustowość oferowaną przez magistralę USB.

Generalnie (oczywiście z wieloma różnicami co do szczegółów) oba urządzenia zapewniają obsługę USB do poziomu interfejsu SIE (rys. 1). Funkcje kontrolera protokołu (czyli obsługa transferów i transakcji, odpowiednia treść deskryptorów, przeprowadzenie enu-

meracji itd.) muszą być zawarte w naszym programie. Nawet najprostszy demonstracyjny przyrządek jest w związku z tym wyposażony w dosyć obszerny blok kodu realizującego komunikację USB, co bynajmniej nie ułatwia uruchomienia własnych projektów. Jest to kłopotem, zwłaszcza gdy nie zależy nam specjalnie na zgłębianiu tajników USB, a chcemy po prostu mieć port komunikacyjny do szybkiej wymiany danych z własną aplikacją uruchomioną na PC.

Drugą główną barierą w takiej sytuacji jest konieczność napisania własnego sterownika WDM - przedsięwzięcie znacznie odbiegające od zakresu działalności elektronika-amatora, a przy tym niemożliwe do wykonania za pomocą popularnych środowisk RAD, jak np. Delphi. Oczywiście na rynku obecne są odpowiednie narzędzia, ale nie jest to oferta dla hobbystów - wystarczy przejrzeć cenniki. Uproszczonym wyjściem jest ograniczenie się do jednej z klas urządzeń obsługiwanych samodzielnie przez system operacyjny, zazwyczaj jednak wyposażonych w dosyć skromne możliwości. Jedną z takich klas jest np. HID (*human interface devices*) - periferie komputera obsługiwane przez użytkownika (mysz, klawiatura), którym wystarczają krótkie transfery typu *interrupt* do przekazania od czasu do czasu niewielkich liczby danych.

Pojawienie się oferty firmy FTDI, a zwłaszcza układów FT8U232 i FT8U245 przedstawianych już na łamach EP, radykalnie zredukowało wymienione powyżej trudności. Dostajemy do dyspozycji zawarte w pojedynczych kostkach kompletne urządzenia USB i możemy w naszych projektach korzystać bezpośrednio z ich interfejsów *we/wy*, wcale nie zajmując się operacjami niższego poziomu. FT8U232 zapewnia interfejs zgodny z protokołem RS232, natomiast FT8U245 udostępnia bufor FIFO przeznaczony do szybkiego 8-bitowego równoległego zapisu i odczytu. Na rys. 6 pokazano schemat blokowy FT8U232 - łatwo możemy zidentyfikować zespoły ogólnie opisane na rys. 1, wiedząc teraz dokładnie do czego służą.



Rys. 6. Schemat blokowy układu FT8U232

Dodatkowego wyjaśnienia wymaga obecność kontrolera pamięci EEPROM. FT8U232 ma wpisany na stałe domyślny, jednakowo w każdej kostce deskryptor urządzenia z danymi identyfikacyjnymi FTDI (VID, PID, nazwa). Jeśli chcemy użyć we własnym urządzeniu indywidualnych da-

nych - możemy je załadować do zewnętrznej pamięci 93C46. Gdy układ wykryje obecność odpowiednio zapisanej pamięci, zastępuje jej treścią odpowiednie pozycje deskryptora.

Wymiana danych pomiędzy aplikacją PC a modułem UART odbywa się za pomocą transferów

typu *bulk*. Duża pojemność buforów pośredniczących zabezpiecza przed ewentualną utratą danych.

Moduł DPLL odpowiada za opisywaną wcześniej synchronizację własnego oscylatora z przebiegiem sygnału w parze różnicowej D+ D-.

Zauważmy, że - przynajmniej jeśli chodzi o FT8U232 - układy nie są wielką nowością. Podobne od dawna stosuje się w popularnych akcesoriach komputerowych (np. kostki firmy Prolific używane w konwerterach USB-RS232C).

Natomiast rewelacyjną zmianą jest szeroki zakres wsparcia otrzymywanego od FTDI nie tylko przez producentów sprzętu, ale także przez indywidualnego użytkownika - amatora. Bezpłatne sterowniki (dla wszystkich popularnych systemów operacyjnych, w tym Linuxa), przykładowe projekty i schematy, wzorcowe programy dla popularnych środowisk programistycznych (np. Delphi), uniwersalne, łatwe w montażu moduły, dostępność kostek w sprzedaży detalicznej, obszernie opisy dostępne bezpośrednio na stronie WWW producenta - to wszystko pozwala na szybkie i sprawne wyposażenie własnych urządzeń w port komunikacji USB.

**Jerzy Szczesiul, AVT**  
**jerzy.szczesiul@ep.com.pl**

*Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: <http://www.ep.com.pl/?pdf/wrzesien02.htm>.*