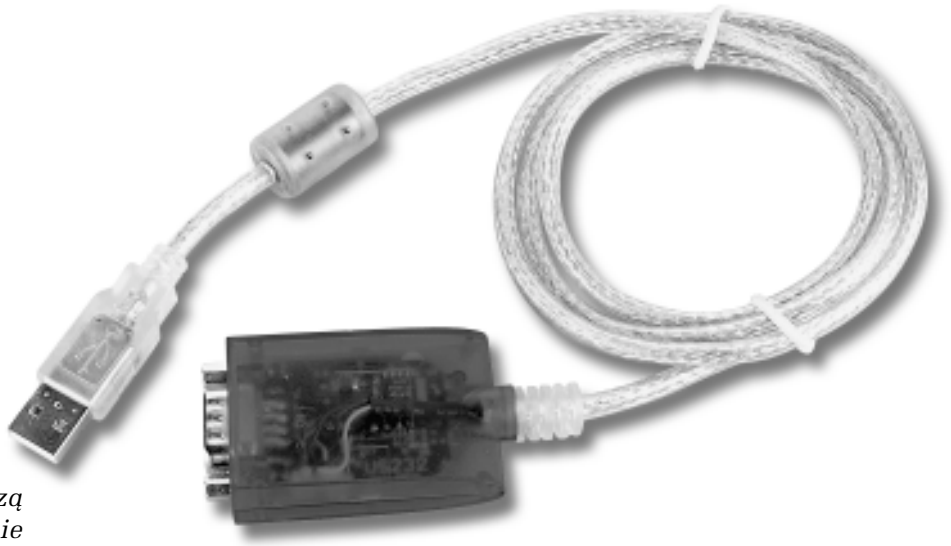
Konwerter USB<->RS232, część 1

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 szeregowe 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* w dziale *Download>Dokumentacje*). Zaczniemy od strony sygnałów przesyłanych kablem.

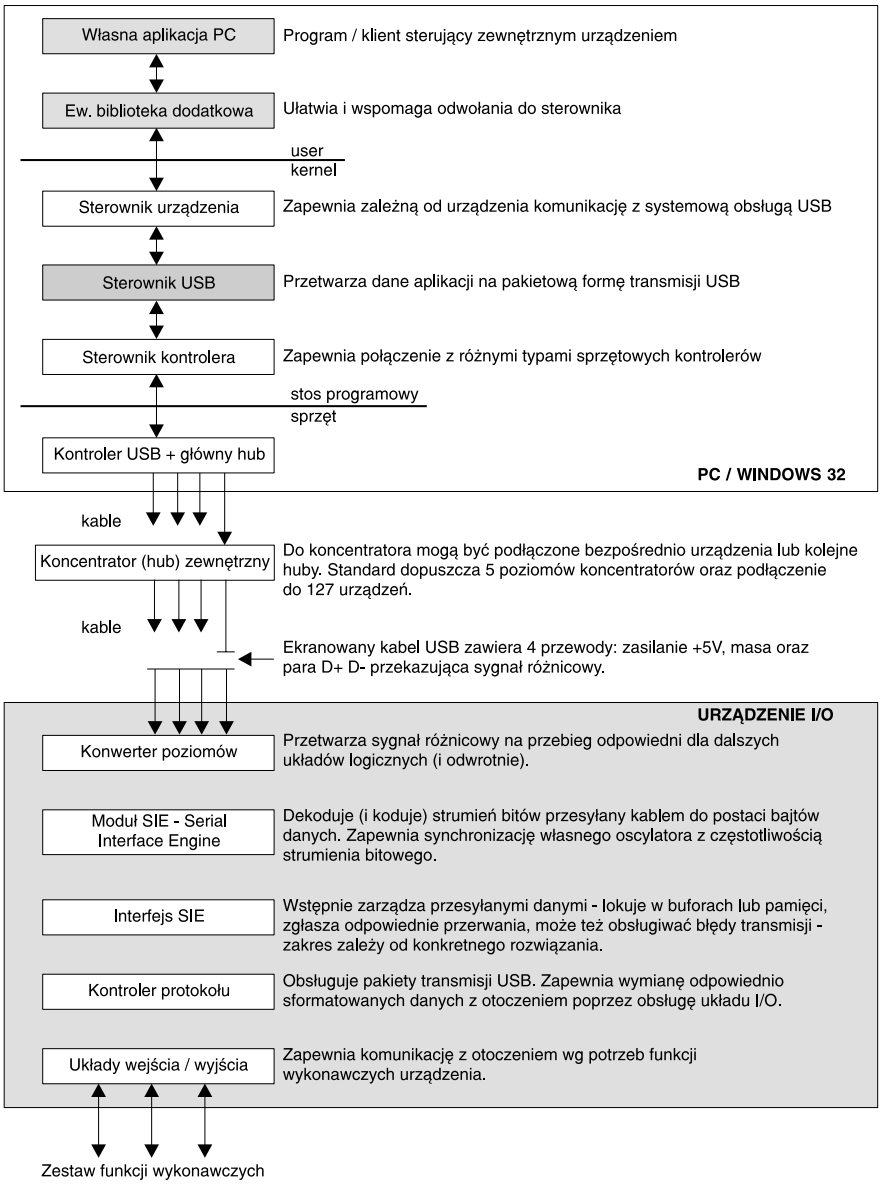
Standardowy kabel połączeniowy USB ma 4 żyły: dwie z nich przekazują zasilanie (masa oraz +5V), następne dwie to para przenosząca różnicowy sygnał danych. Sygnał ten pełni kilka jednoczesnych funkcji:

- 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 (nadpróbkowaniem) - dla szybkości przesyłu danych wynoszącej 12 Mbd konieczny będzie więc zegar o częstotliwości 48 MHz. Oscylator jest synchronizowany (pętlą 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 każdych 6 jedynkach) - 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 (wyślij pakiet danych przez EPx),
- pakiet DATA0 - z urządzenia (urządzenie wysyła 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 obsługiwania 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 przerwań 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

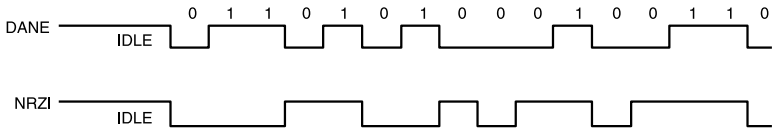
PID	Typ	Kategoria	Zawartość i funkcja
0101	SOF	token (kontrolny)	Start-of-frame: jest wysyłany co 1 ms (zapewnia to dodatkową synchronizację oraz pozwala na wykrywanie zawieszenia magistrali). Składa się z 11-bitowego kolejnego numeru ramki (frame) oraz 5-bitowego kontrolnego CRC.
1101	SETUP	token (kontrolny)	Priorytetowy pakiet kontrolny służący do podstawowego nadzoru i konfiguracji - składa się z 7-bitowego adresu urządzenia, 4-bitowego adresu EP0 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.

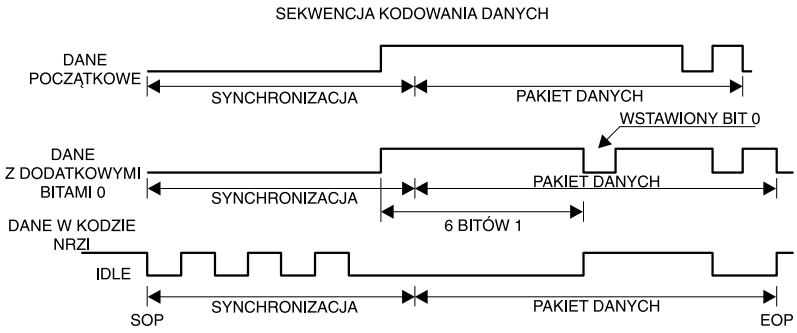
Aby dalej omawiać wykorzystanie tych transferów, należy się zapoznać z logiczną strukturą połączenia USB. Jest ona ogólnie przedstawiona na **rys. 5**. Podstawowym pojęciem jest tutaj *endpoint* (już wspomniany w opisach pakietów) - czyli punkt końcowy. *Endpoint* jest zakończeniem logicznego kanału 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*: EP0 (IN oraz OUT) o jednoznacznie określonej charakterystyce, który obsługuje podstawowe transfery kontrolne. Dalsze *endpointy* 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 sprawdzonych i często typowych elementów składowych). Zbiór interfejsów tworzy z kolei konfigurację, których też może być kilka (ale tylko jedna może być aktywna).

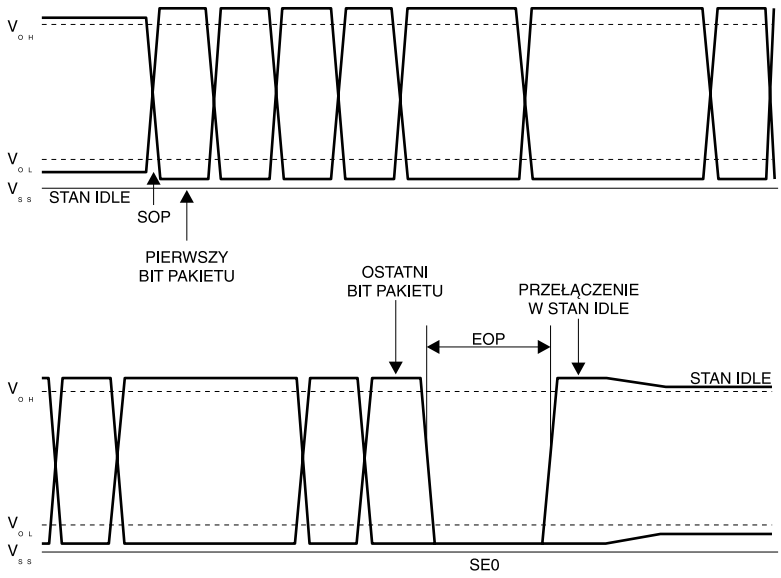
Jak widać na rys. 5, klient (czyli oprogramowanie korzystające z obsługi USB oferowanej przez system operacyjny), który dostaje do dyspozycji logiczne kanały komunikacji z interfejsem urządzenia USB, nie musi nic „wiedzieć" o wcześniej opisywanych mechanizmach wykonawczych niższego poziomu. W systemie 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)

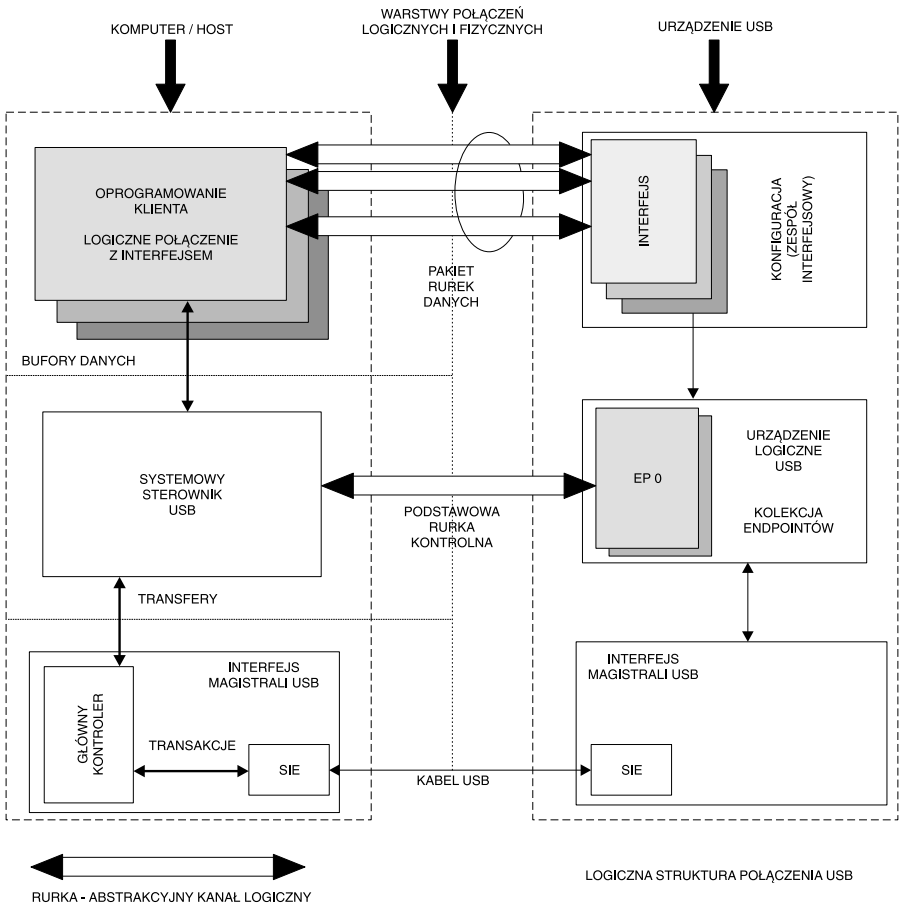
określonego mechanizmu pobrania odpowiednich informacji oraz dokonania wstępnej konfiguracji. Magistrala cały czas kontroluje dołączanie i odłączanie urządzeń - możemy to robić w dowolnym momencie.

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 dowiaduje 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 wykonania zerowania 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 (tylko 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 późniejszą komunikację z urządzeniem zgodnie z konkretnymi, specyficznymi potrzebami. System na podstawie danych w deskryptorze 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 znajdą 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 skrótowe - 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 PDIUSBD11 - interfejs SIE wyposażony w magistralę I^2C.

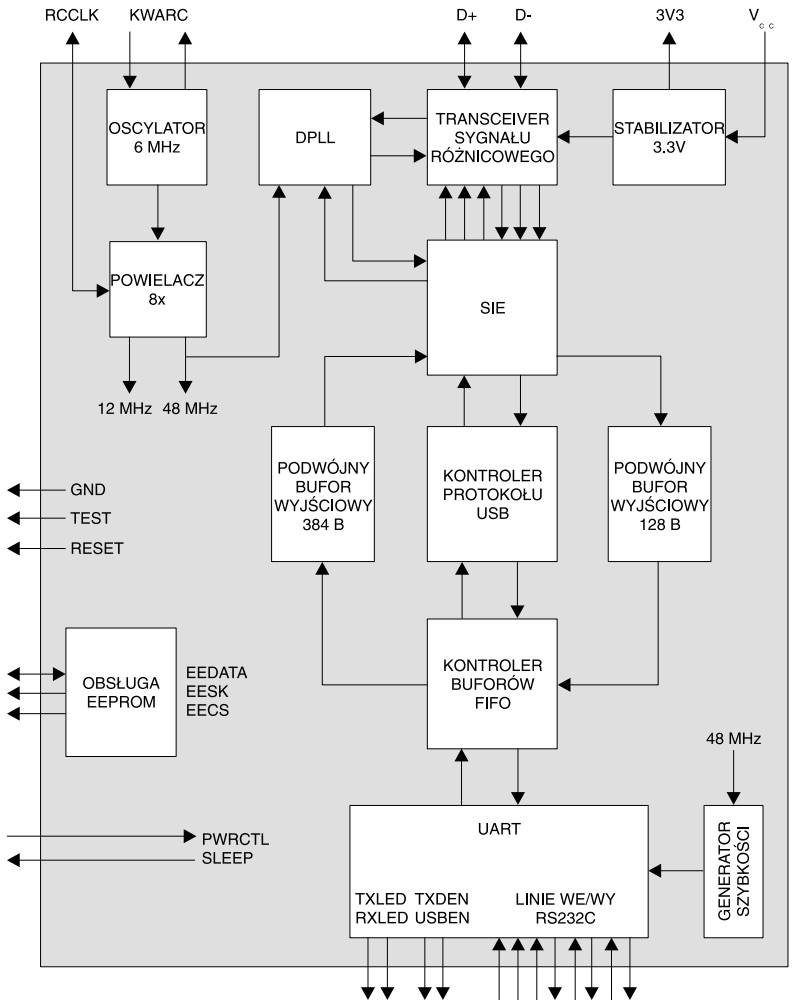
EZ-USB ma wbudowany *loader*, który po włączeniu urządzenia sam przeprowadza pierwszą enumerację, a następnie ładuje do obszaru pamięci kodu program pracy mikrokontrolera przesyłany przez współpracujący po stronie *hosta* sterownik. Po zakończeniu tego procesu magistrala jest zerowana i wykonywana jest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

PDIUSBD11 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^2C, 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 enumeracji itd.) muszą być zawarte w naszym programie. Nawet najprostszy demonstracyjny przyrządzik 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*) - peryferie 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, jednakowy w każdej kostce deskryptor urządzenia z danymi identyfikacyjnymi FTDI (VID, PID, nazwa). Jeśli chcemy użyć we własnym urządzeniu indywidualnych danych - 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).

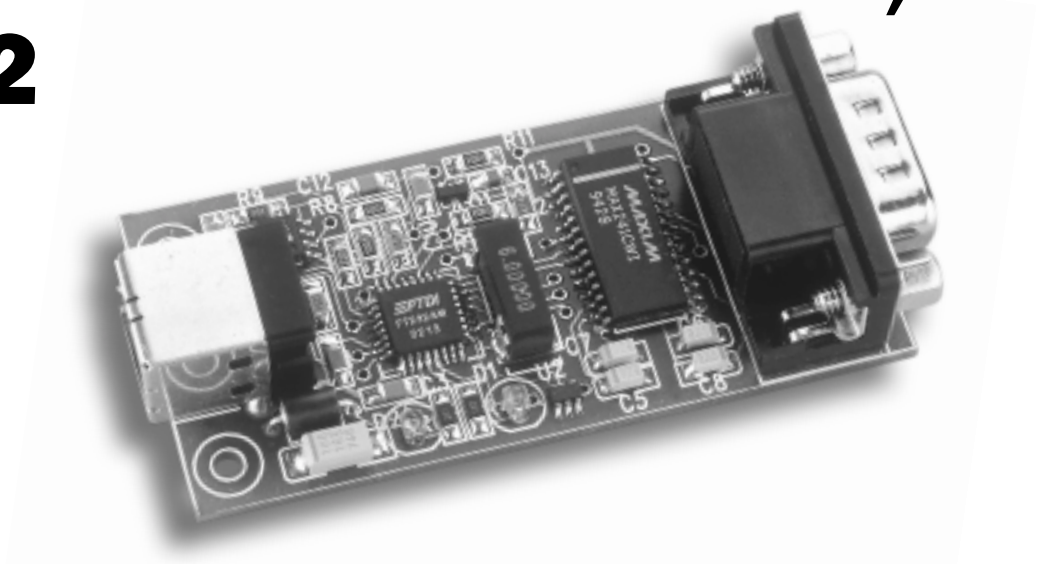
Natomiast rewelacyjną zmianą jest szeroki zakres wsparcia otrzymywanego od FTDI nie tylko przez producentów sprzętu, ale także przez indywidualnego użytkownika - amatora. Bezpłatne sterowniki (dla wszystkich popularnych systemów operacyjnych, w tym Linuxa), przykładowe projekty i schematy, wzorcowe programy dla popularnych środowisk programistycznych (np. Delphi), uniwersalne, łatwe w montażu moduły, dostępność kostek w sprzedaży detalicznej, obszerne 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****.*

Konwerter USB<->RS232, część 2

AVT-5080



Drugą część artykułu poświęcamy przedstawieniu budowy konwertera USB<->RS232. Zawarte w niej informacje wystarczą do samodzielnego wykonania i uruchomienia konwertera. Czytelników zainteresowanych tym tematem zachęcamy do sięgnięcia po wrześniowy numer EP.
Rekomendacje: *jest to wzorcowe opracowanie dla wszystkich konstruktorów zamierzających stosować we własnych konstrukcjach interfejs USB.*

W celu nabrania wprawy w korzystaniu z układów firmy FTDI oraz uniknięcia ewentualnych niespodzianek, najpierw wykonano podstawową, dokładnie opisaną aplikację konwertera USB - RS 232.

Schemat elektryczny konwertera pokazano na **rys. 7**. Jest on prawie identyczny ze schematem konwertera zalecanym przez producenta. Różni się od niego tylko typem interfejsu RS232, co wynikło z posiadanych akurat zapasów.

Sygnał różnicowy oraz zasilanie doprowadzone są przez gniazdo USB typu B. Zasilanie jest filtrowane za pomocą obwodu zbudowanego z elementów C1, L1, C9 oraz kilku rozmieszczonych na płytce kondensatorów 100 nF. Zasilanie części analogowej układu FT8U232 jest dodatkowo filtrowane przez filtr dolnoprzepustowy zbudowany z elementów R5, C10 (filtr służy do zasilenia analogowego układu powielacza częstotliwości 8x).

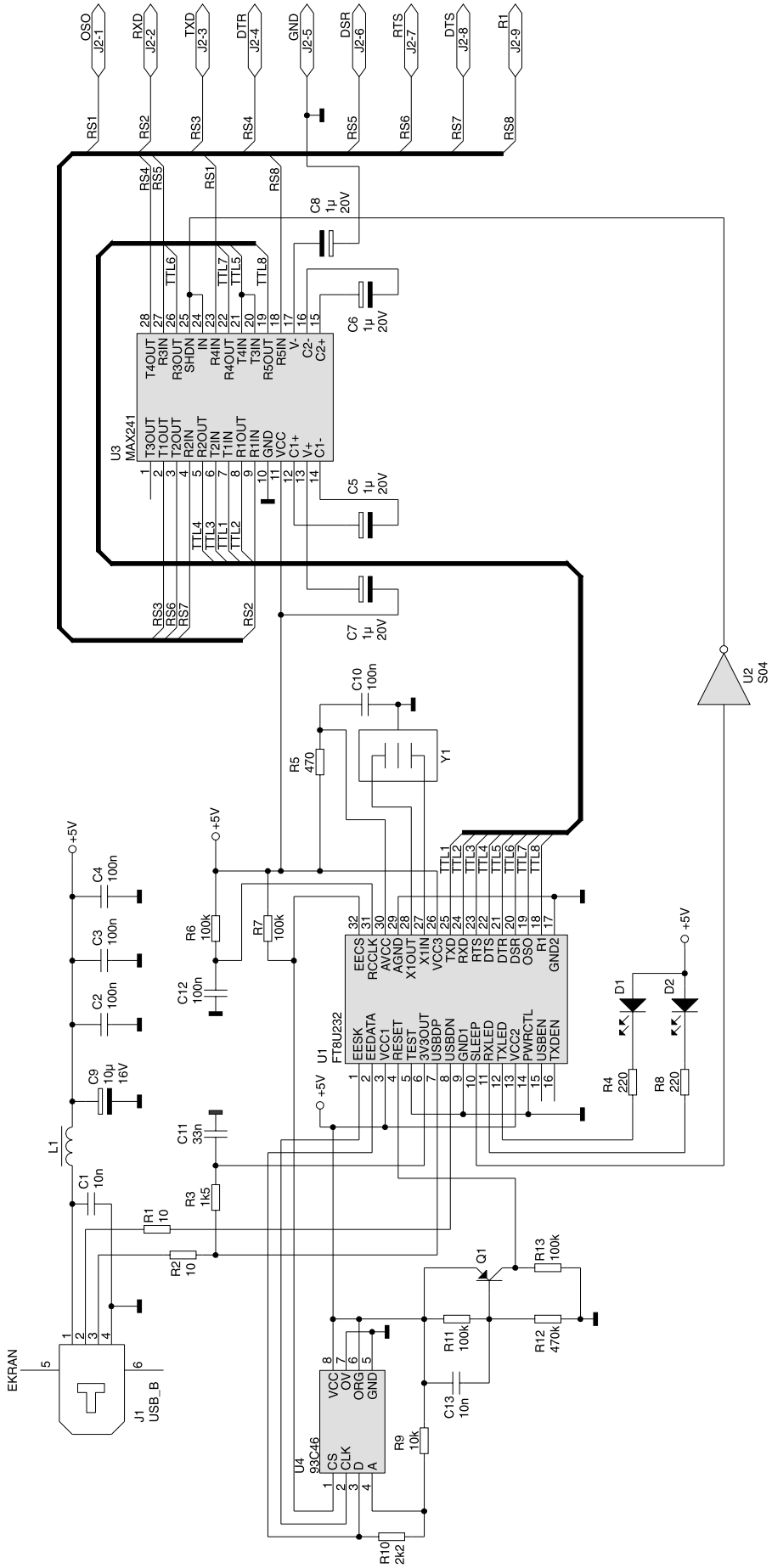
Rezystory R1, R2 zapewniają dopasowanie do impedancji wejściowej transceivera. Podanie napięcia 3,3 V (z wewnętrznego stabilizatora) poprzez rezystor R3 na linię D+ informuje hub o dołączeniu konwertera do magistrali.

Linie EECS, EESK, EEDATA służą do obsługi opcjonalnej pamięci EEPROM 93C46. Dodatkowo EECS, jako linia wejściowa, pozwala na wybór zastosowanego kwarcu. Podciągnięcie do zasilania (R7) odpowiada pracy z oscylatorem o częstotliwości 6 MHz i z 8-krotnym powielaniem. Dołączanie linii do masy przełącza oscylator na docelowe 48 MHz.

Na uwagę zasługuje dosyć rozbudowany układ zerowania. Po włączeniu zasilania na wyprowadzeniu RESET utrzymuje się niski poziom napięcia. Tranzystor Q1 jest włączany (dzięki dzielnikowi R11, R12) dopiero po osiągnięciu napięcia Vcc=3,6 V (U4). Kondensator C13 wprowadza niewielkie opóźnienie ok. 2 ms (zauważmy, że czas zerowania układu powinien być krótszy niż 10 ms - po tym czasie wystawiany jest sygnał zerowania magistrali przez hub - aby układ był od razu gotów do obsługi transferu kontrolnego).

Oddzielnego zerowania wymaga powielacz częstotliwości. Powinien on być zablokowany przez kilka milisekund zanim oscylator nie zacznie stabilnie pracować. Uzyskuje się to poprzez podanie poziomu niskiego na wyprowadzenie RCCLK. Jednocześnie układ wymaga, aby dla stabilnego uruchomienia powielacza jego odblokowanie następowało przy wysokim poziomie na wejściu RESET. W konwerterze osiąga się to przez dobór odpowiedniej stałej czasowej obwodu R6, C12.

Na wyjściach RXLED i TXLED występują przebiegi impulsowe podczas napływu danych do bufora odbiorczego oraz ich wysy-



Rys. 7. Schemat elektryczny konwertera USB<->RS 232C

łania z bufora nadawczego. Diody LED dołączone przez R4 i R6 pozwalają na optyczną kontrolę transmisji. Nie będziemy ich montować na płytce schowanej w obudowie, jednak na etapie testowania są bardzo przydatne.

Wyjście USBEN służy do sygnalizacji dołączenia do magistrali - może być wykorzystane np. do przełączenia kanału transmisyjnego w urządzeniach wyposażonych zarówno w USB, jak i zamienny RS232C.

Wyjście TXDEN jest aktywne przy wysyłaniu danych aż do chwili całkowitego opróżnienia bufora nadajnika. Służy do półdupleksowych połączeń RS485 i eliminuje programowe przełączanie kierunku transmisji, konieczne przy korzystaniu ze zwykłego portu.

Wyjście SLEEP informuje o trybie *suspend* (zawieszenia) magistrali. Tryb ten (o czym nie wspomniano wcześniej) dotyczy zachowania się konwertera podczas przełączenia hosta w uśpienie (*stand-by*). Wówczas host przestaje wysyłać pakiety SOF. Brak 3 pakietów (czyli przerwa 3 ms) jest wykrywany przez układ FT8U232, który także przechodzi w stan uśpienia z maksymalnie zmniejszonym poborem mocy. W trybie *suspend* hub może dostarczyć do konwertera co najwyżej 0,5 mA. Sam FT8U232 ma oczywiście wbudowane odpowiednie mechanizmy redukcji mocy, jednak pozostała część układu musi być sprzętowo wyłączona. Jest to zapewnione poprzez podłączenie wyjścia SLEEP do wejścia *shutdown* interfejsu RS232C. Oczywiście, najlepiej jest, jeśli aktywne poziomy napięć są całkowicie zgodne. Ponieważ jednak zamiast MAX213 zastosowano posiadany MAX241, konieczne było dopasowanie poziomów za pomocą jednobramkowego inwertera U2. Układ MAX241 ogranicza też szybkość transmisji do tradycyjnych dla portu szeregowego 115 kbd, podczas gdy FT8U232 oferuje transfer do 920 kbd - w razie potrzeby można wlutować nowocześniejszy interfejs - jest zachowana zgodność wyprowadzeń.

Ponieważ FT8U232 występuje tylko i wyłącznie w obudowie

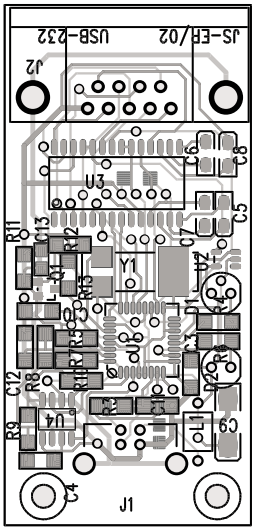
TQFP, w pozostałej części płytki także użyto głównie elementów SMD.

Nie ma raczej odwrotu od upowszechniania się tej technologii również w ręcznie lutowanych urządzeniach amatorskich. Po zdobyciu pewnej wprawy stwierdzimy, że montaż jest częstokroć szybszy i łatwiejszy niż przy sposobach tradycyjnych. Także kompletacja takich elementów sprawia coraz mniej kłopotu.

Elementy są wlutowane na płytkę drukowaną zgodnie z **rys. 8**. Kolejność lutowania ustalamy tak, aby zachować jak najlepszy dostęp do elementów (rezystory i kondensatory, półprzewodniki, kwarc, na końcu elementy przewlekane i gniazda). Kondensatory C1, C2, R1, R2 są montowane na spodzie płytki. Podczas początkowych testów można pominąć pamięć EEPROM.

Układ prototypowy został złożony bez pomocy specjalnych akcesoriów (jak np. grot *wave* i odpowiednie topniki) - wystarczył cienki grot i tinol 0,5 mm oraz taśma rozlutownicza *Wick*. Należy jednak zwracać uwagę na kolejność wyprowadzeń - wylutowanie np. obróconego o 180° układu będzie już wymagało bardziej wyrafinowanego sprzętu.

Z tego też względu warto sprawdzić przed montażem prawidłowość wykonania metalizacji na płytce - zajmie to chwilę, a może oszczędzić poważnych kłopotów. Po usunięciu resztek topnika preparatem Kontakt PCC i wysuszeniu płytka jest gotowa do uruchomienia.



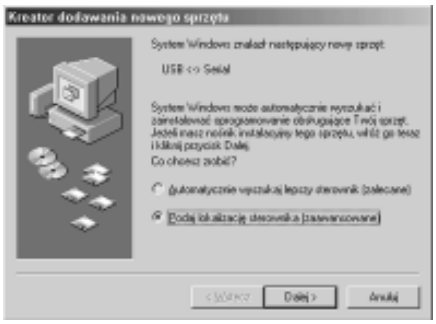
Rys. 8. Rozmieszczenie elementów na płytce konwertera

Przygotowanie sterowników

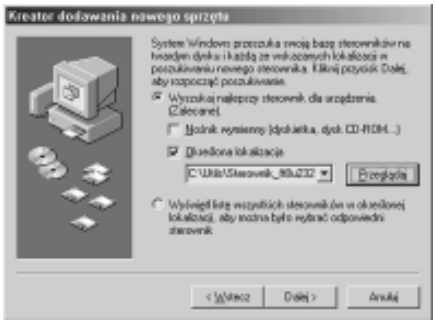
Przed pierwszym podłączeniem konwertera do komputera musimy się wyposażyć w odpowiedni sterownik. Na stronie producenta mamy do dyspozycji sterowniki dla różnych systemów operacyjnych (prototyp był sprawdzany pod Windows 98 oraz ME). Wybieramy odpowiedni i zapisujemy go na dysk. Należy też zwrócić uwagę, że FT8U232 może być obsługiwany (pod Windows) za pomocą kilku sterowników:

- podstawowy sterownik portu szeregowego, który przechwytuje wywołania funkcji API i przekierowuje je do stosu,
- USB (w komunikacji korzystamy wtedy z typowych funkcji Windows obsługi portu, w Delphi bez problemu można użyć dotychczasowe komponenty RS, np. *TComPort* lub *TRsPort* ze strony EP),
- rozszerzony sterownik portu - posiada możliwości jak wyżej, ale dodatkowo obsługuje mechanizm *Plug & Play* (wykrywanie przy starcie Windows nowych urządzeń - jeśli nie budujemy urządzenia, które zareaguje na wywołanie PnP, lepiej tego sterownika nie stosować, gdyż system będzie dosyć długo czekać na odpowiedź, co spowolni uruchamianie komputera),
- sterownik bezpośredni (*direct*), który wymaga oddzielnych funkcji do obsługi portu (funkcje te są udostępniane w dołączonej bibliotece *dll*) - nie można zatem korzystać z gotowych komponentów, ale za to mamy dostęp do rozszerzonego zestawu operacji (ustawianie nietypowych szybkości transmisji, dostęp do zawartości EEPROM).

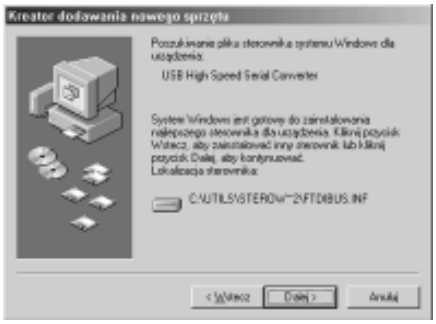
Dla potrzeb naszego projektu będą nam potrzebne dwa sterowniki: podstawowy oraz *direct*. Po ściągnięciu ze strony FTDI (*www.ftdichip.com*, publikujemy je także na naszej płycie CD-EP10/2002B) potrzebnych plików (dla Windows 98/ME są to - w chwili pisania artykułu - *N8002101.zip* oraz *D2XX10401.zip*) rozpakowujemy je do oddzielnych folderów (o dowolnych nazwach) utworzonych na twardym dysku.



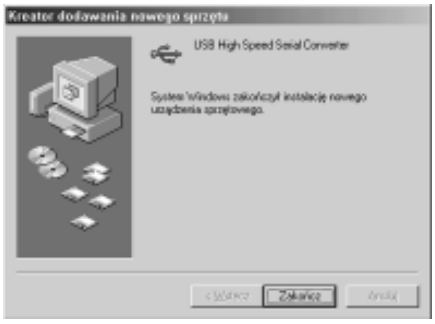
Rys. 9. Wykrycie przez Windows płytki konwertera



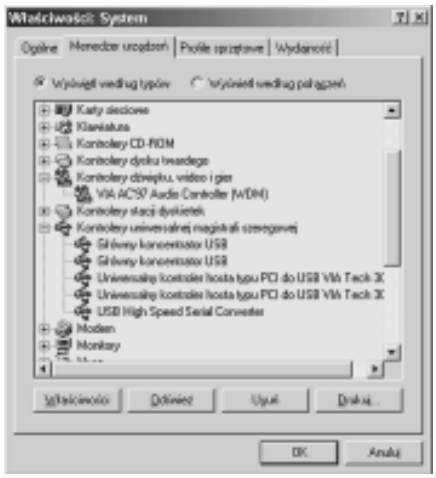
Rys. 10. Podanie lokalizacji potrzebnego sterownika



Rys. 11. Potwierdzenie odszukania sterownika



Rys. 12. Zakończenie instalacji konwertera jako wirtualnego portu COM



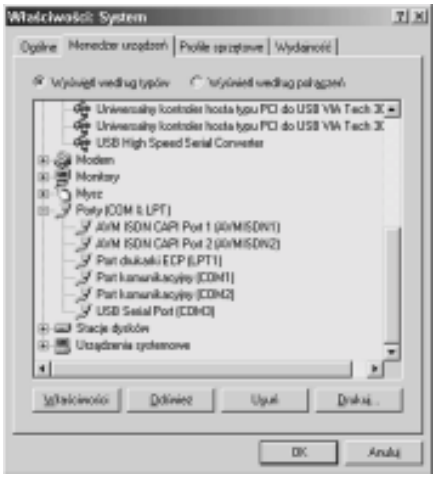
Rys. 13. Kontroler USB w oknie menedżera sprzętu Windows

Uruchomienie układu

Teraz nadchodzi chwila prawdy. Podłączamy kablem naszą płytkę do gniazda *downstream* komputera (lub huba). Poprawnie zmontowane urządzenie jest od razu wykryte przez system, który prosi o wskazanie lokalizacji sterownika (**rys. 9**) - zaznaczamy opcję *Podaj lokalizację sterownika*.

Po potwierdzeniu otwiera się okienko lokalizacji (**rys. 10**) - zaznaczamy opcję *Określona lokalizacja* i wybieramy folder, do którego rozpakowaliśmy pliki podstawowego drivera wirtualnego portu COM.

Windows potwierdza znalezienie prawidłowego sterownika (**rys. 11**), a następnie informuje o poprawnym zakończeniu instalacji nowego sprzętu (**rys. 12**).



Rys. 14. Wirtualny port COM w oknie menedżera sprzętu Windows

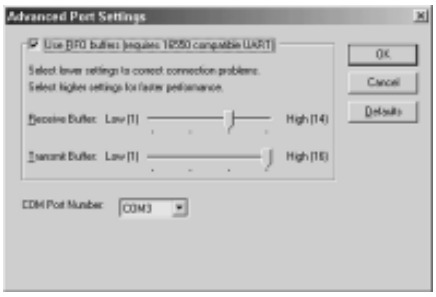
Po załadowaniu sterownika Windows umieszcza konwerter na swojej liście sprzętu i przydziela mu numer portu szeregowego - jest to widoczne w oknie menedżera urządzeń pod pozycjami *kontroler uniwersalnej magistrali szeregowej* oraz *porty* - **rys. 13** i **14**. Zwróćmy uwagę, że zapis na liście menedżera jest każdorazowo aktualizowany przy odłączaniu i dołączaniu płytki - możemy się naocznie przekonać, jak działa mechanizm wykrywania urządzeń przez magistralę USB. Numer portu jest wybrany jako pierwszy wolny, ale możemy go dowolnie zmienić, korzystając z okna dialogowego *Port>Właściwości>Zaawansowane* - **rys. 15**.

W tej chwili konwerter jest już praktycznie gotowy do działania. Jedyny problem to podłączanie dwóch płytek (bez ustawionych numerów seryjnych) do jednego komputera - wystąpią kłopoty z identyfikacją (mankament ten jest usunięty w wersji układu FT8U232BM).

Programowanie pamięci EEPROM

Dla zaprogramowania pamięci 93C46 na płytce służy bezpłatny program *ftd2xxst.exe* (pobrany ze strony FTDI wraz z opisem w formacie pdf). Wymaga on jednak zainstalowania wspomnianego wcześniej sterownika *direct*. Aby nie usuwać użytego do tej pory sterownika portu wirtualnego, wykorzystamy standardowy mechanizm aktualizacji sterowników w Windows. W oknie menedżera sprzętu wybieramy nasz kontroler USB (**rys. 16**) i otwieramy jego właściwości, w których uruchamiamy opcję *Aktualizuj sterownik* (**rys. 17**).

Przejdziemy wtedy przez szereg okien podobny do pierwszej instalacji - ale jako lokalizację podajemy folder z rozpakowanymi plikami sterownika *direct*. Po zakończeniu przeinstalowania w menedżerze sprzętu zmienia się opis konwertera a także znika wirtualny port szeregowy (**rys. 18**) - od tej chwili do obsługi UART-u musimy używać oddzielnych funkcji z dołączonej biblioteki *dll* (na stronie znajdziemy szereg przykładów dla popularnych środowisk

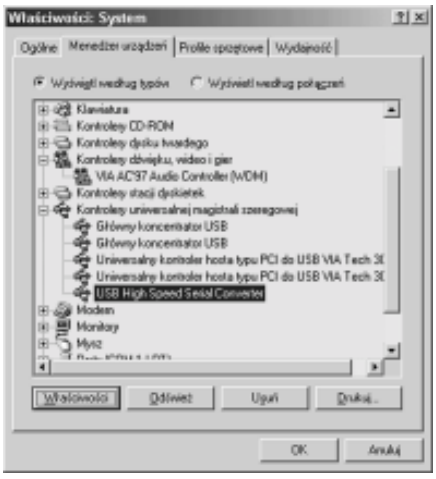


Rys. 15. Ustawienie numeru portu szeregowego

programistycznych, oczywiście także dla Delphi).

Teraz możemy uruchomić program serwisowy. Po jego uruchomieniu (**rys. 19**) należy wypełnić wszystkie pola edycyjne podstawowego opisu urządzenia (dalsze opcje uaktywniają się dopiero po ponownym przejściu do pola *Manufacturer* klawiszem *Tab*). Tutaj jedna bardzo istotna uwaga - dla własnych warsztatowych (a nie produkcyjno-komercyjnych) potrzeb nie warto zmieniać żadnych identyfikatorów poza opisem. Dotyczy to zwłaszcza numerów VID i PID, na podstawie których Windows identyfikuje potrzebny sterownik. Przy zmianie VID/PID należy wyszukać i zmienić również wpisy w odpowiednich plikach **.inf* - w przeciwnym razie system nie odnajdzie sterownika i konwerter pozostanie nieznanym urządzeniem.

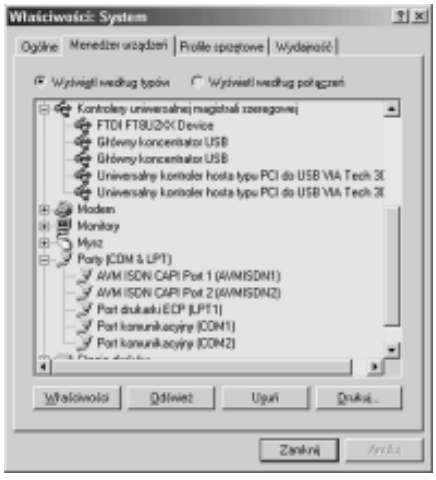
Po zapamiętaniu nastaw włączamy uaktywnione wtedy *ustawienia zaawansowane* (*advanced setup*) - **rys. 20**. Umożliwiają one:
- Wybór opcji *Plug and Play* - pozostawiamy niezaznaczoną,
- Wybór ręcznego lub automatycz-



Rys. 16. Rozpoczęcie wymiany sterownika

Rys. 17. Okno umożliwiające aktualizację (wymianę sterownika)

nego przyznania numeru seryjnego (automatyczny jest wygodny, gdyż zapobiega - przy programowaniu konwerterów na danym komputerze - powtórzeniu się numeru. Dla potrzeb naszego projektu obejmującego tylko kilka płytek został jednak wybrany tryb ręczny),
- Wybór sposobu zasilania: z magistrali albo samodzielne - pozostawiamy zasilanie z magistrali (nie jest to zbyt istotne, gdyż nadrzędny dla określenia sposobu zasilania jest poziom na wejściu PWRCTL kostki, co pozwala na sprzętową kontrolę aktualnie używanego źródła zasilania w bardziej rozbudowanych układach),
- Udostępnienie funkcji zdalnego budzenia *hosta* z trybu *standby*. Układ wykonuje to ustawiając parę różnicową w stan K (odwrócona polaryzacja) - dalszym przekazaniem tego sygnału zajmuje się już hub. W naszym przykładzie opcja pozostaje nieczynna (MAX241 nie posiada funkcji zachowania aktywności wybranych linii wejściowych w trybie *shutdown*, a tylko to pozwala fizycznie dostarczyć do konwertera sygnał budzący),
- Określenie maksymalnego poboru prądu: konwerter (jak wynika z not katalogowych użytych układów) pobiera zawsze poniżej 100 mA. Umożliwia to bezproblemowe zasilenie z magistrali USB, gdyż każdy hub gwarantuje 100 mA w portach *downstream* (nawet w trakcie enumeracji). W polu edycyjnym została wpisana przykładowo wartość nieco mniejsza.



Rys. 18. Menedżer urządzeń po przeinstalowaniu sterownika

Po wprowadzeniu danych wykonujemy programowanie i kontrolny odczyt EEPROM - wynik tej operacji widzimy na **rys. 21**.

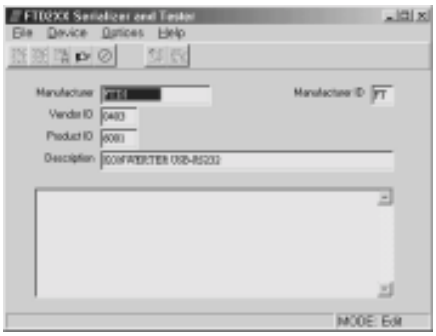
Sprzętowy test konwertera

Program serwisowy umożliwia też przeprowadzenie kompletnego testu sprzętowego. Wymagane jest jednak do tego posiadanie w komputerze wolnych portów szeregowych COM1 i COM2 oraz przygotowanie odpowiedniego kabla połączeniowego:

Pin 3 (TXD) - COM2 Pin 2 (RXD)
Pin 2 (RXD) - COM2 Pin 3 (TXD)
Pin 7 (RTS) - COM2 Pin 8 (CTS)
Pin 8 (CTS) - COM2 Pin 7 (RTS)
Pin 6 (DSR) - COM2 Pin 4 (DTR)
Pin 5 (GND) - COM2 Pin 5 (GND)
Pin 4 (DTR) - COM2 Pin 6 (DSR)
Pin 1 (CD) - COM1 Pin 4 (DTR)
Pin 9 (RI) - COM1 Pin 7 (RTS)
- dla złącz DB9.

Uwaga! Test uruchamiamy dopiero po wykonaniu wszystkich podłączeń - w przeciwnym przypadku program się zawiesza (w przypadku Windows 98 razem z systemem). Wynik poprawnie przeprowadzonego testu jest pokazany na **rys. 22**.

Teraz możemy - o ile zachodzi taka potrzeba - powrócić do sterownika wirtualnego portu COM. Ponieważ w systemie są już zainstalowane oba sterowniki, procedura znacznie się upraszcza: przy następnej aktualizacji wybierzemy w oknie lokalizacji opcję *Wyświetl listę wszystkich sterowników* i zaznaczymy potrzebny.



Rys. 19. Okno główne programu serwisowego

Zazwyczaj do wstępnego sprawdzenia najbardziej odpowiedni będzie pierwszy z wymienionych sterowników.

Teraz nadchodzi „chwila prawdy". Podłączamy kablem naszą płytkę do gniazda *downstream* komputera (lub huba). Poprawnie zmontowany konwerter jest od razu wykryty przez system, który prosi o wskazanie lokalizacji sterownika. Podajemy ścieżkę do folderu, w którym ulokowaliśmy odpowiedni sterownik FTDI. Po załadowaniu sterownika Windows umieszcza konwerter na swojej liście sprzętu i przydziela mu nu-

WYKAZ ELEMENTÓW

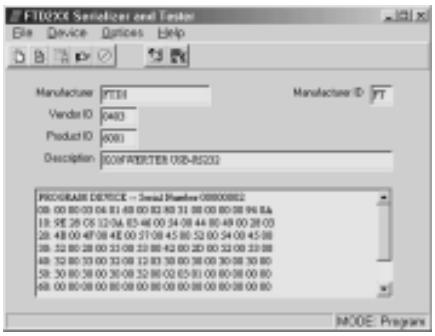
Rezystory
R1, R2: 10Ω 805
R3: 1,5kΩ 1206
R4, R6: 220Ω 1206
R5: 470Ω 1206
R7, R8, R11, R13: 100kΩ 1206
R9: 10kΩ 1206
R10: 2,2kΩ 1206
R12: 470kΩ 1206

Kondensatory
C1, C13: 10nF 805 ceramiczny
C2...C4: 100nF 1206 ceramiczny
C5...C8: 1µF/20V 3216 tantalowy
C9: 10µF/16V 6032 tantalowy
C10, C12: 100nF 1206 ceramiczny
C11: 33nF 1206 ceramiczny

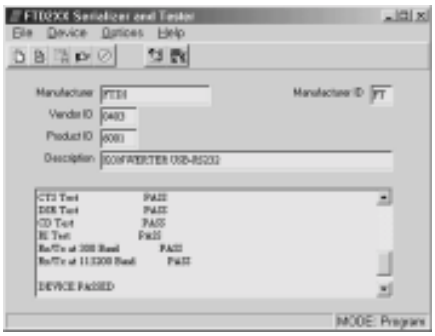
Półprzewodniki
D1, D2: LED 3mm
Q1: BC857 SOT23
U1: FT8U232AM
U2: NC7S04 SOT23-5
U3: MAX241 SO28
U4: 93C46 SO8

Różne
Y1: oscylator ceramiczny 6MHz
J1: gniazdo USB typu B
J2: gniazdo kątowe DB-9M
L1: koralik ferrytowy przewlekany

Rys. 20. Okno zaawansowanych ustawień układu FT8U2xxAM



Rys. 21. Odczyt kontrolny zaprogramowanej pamięci



Rys. 22. Wynik poprawnego testu sprzętowego konwertera

mer portu szeregowego (obejrzymy to w oknie menedżera urządzeń pod pozycjami „porty") oraz kontroler uniwersalnej magistrali szeregowej. Pierwszy szybki test możemy wykonać wysyłając cokolwiek z dowolnego programu terminala na port powiązany z konwerterem - dioda *TxLed* mruga potwierdzając wysyłanie, a na wyprowadzeniu 3 gniazda pojawia się przebieg impulsowy. Aby w pełni sprawdzić poprawność działania konwertera, musimy zestawić kompletny tor transmisyjny i skontrolować obustronne nadawanie/odbiór oraz ustawianie i odczyt linii kontrolnych. Można też przeprowadzić firmowy test, ale wymaga to posiadania wolnych portów COM1 i COM2, przygotowania kabla testowego oraz przeładowania sterownika (wymagany *direct*). Test jest wykonywany za pomocą programu dostępnego na stronie *www.ftdichip.com*. Znajduje się tam też PDF z dokładnym opisem obsługi, którego tutaj nie będziemy przytaczać. Ten sam program służy do zapisywania współpracującej pamięci EEPROM - jest to łatwe dzięki załączonym szczegółowym instrukcjom. Należy jedynie pamiętać o korekcie w plikach **.inf* sterowników w razie zmiany numerów VID lub PID (system przeszukuje pliki *inf* i szuka zgodnych numerów - jeśli nie znajdzie, to urządzenie pozostanie nieznane).

Co zrobić, gdy układ nie daje znaku życia? Cóż - nie ma zbyt wielkiego pola manewru. Sprawdźmy dokładnie płytkę: przejścia, zwarcia, kolejność wyprowadzeń, zimne luty, wartości pomocniczych elementów. Spróbujmy uruchomić na innym komputerze i kablu. O ile to wszystko nie pomoże - pozostaje niestety wymiana FT8U232 lub złożenie następnego egzemplarza konwertera.

Materiały pomocnicze FTDI zawierają wiele dodatkowych informacji na temat zastosowanego układu oraz zalecenia do programowania transmisji wynikające ze specyfiki transferów *bulk* używanych od strony magistrali USB. Warto przynajmniej częściowo zapoznać się z nimi przed pisaniem własnych procedur obsługi komunikacji z użyciem konwertera oraz budową innych urządzeń korzystających z FT8U232.

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/pazdziernik02.htm*** *oraz na płycie CD-EP10/2002B w katalogu* ***PCB****.*

W przygotowaniu artykułu wykorzystałem:

- specyfikację standardu USB rev. 1.0 (m.in. na stronie http://www.ep.com.pl),
- USB Design by Example, John Hyde, Wiley Computer Publishing 1999,
- materiały ze strony FTDI http://www.ftdichip.com/,
- materiały ze strony http://www.beyondlogic.org/,
- katalogi podzespołów Maxim oraz Elfa.