Optoizolowany sprzęg USART–USB(VCP)

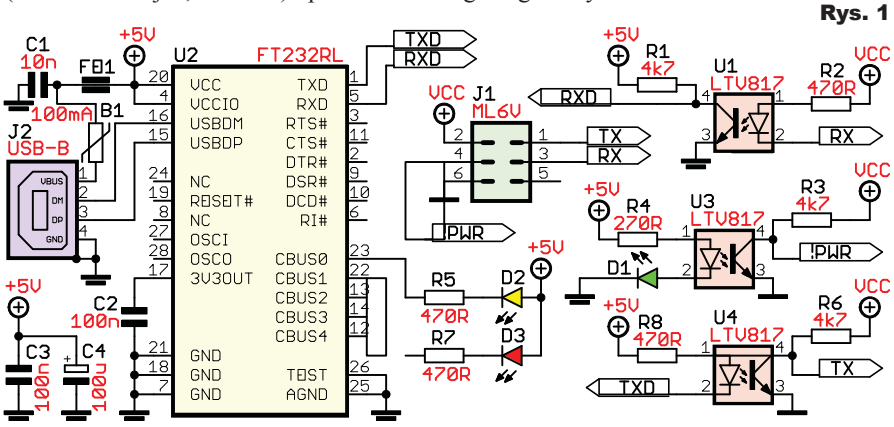
Układ realizuje optoizolowane łącze RS232 VCP. Realizacja i wykorzystanie tego pożytecznego układu okazują się proste. W artykule przedstawione są także dodatkowe informacje, przeznaczone dla bardziej dociekliwych Czytelników, którzy chcą się wprawiać w pisaniu własnych aplikacji korzystających z łącza szeregowego (sprzętowego, VCP). Te dodatkowe informacje nie są potrzebne do praktycznego wykorzystania prezentowanego układu.

Do czego to służy?

W komputerach przenośnych nastąpił niemal całkowity „zanik” sprzętowych portów szeregowych w standardzie RS232C. Również coraz mniej nowych płyt głównych komputerów stacjonarnych jest wyposażanych w ten interfejs. Często stosowanym obejściem problemu braku sprzętowego interfejsu RS232 jest użycie układu FTDI232, zainstalowanie sterowników VCP (Virtual Comm Port) i odwoływanie się z aplikacji (o czym w dalszej części artykułu) na komputerze PC do wirtualnego portu szeregowego tworzonego po dołączeniu układu FT232 do portu USB. Podstawowa aplikacja FT232RL nie zapewnia jednak izolacji galwanicznej, tj. masy komputera i podłączonego urządzenia są elektrycznie połączone.

Interfejs sprawdzi się wszędzie tam, gdzie występuje (może wystąpić) różnica potencjału mas komputera i podłączanego do niego urządzenia. Przykładem mogą być przyrządy pomiarowe współpracujące z komputerem np. przystawki oscyloskopowe. Bez zapewnienia izolacji galwanicznej należy bacznie uważać co i jak jest podłączane do takiej przystawki. Inną spotykaną sytuacją jest zasilanie komputera i urządzenia z dwóch obwodów sieci energetycznej. Nie zawsze mamy pewność, że styk ochronny gniazda, do którego podłączony jest komputer, uziemiony jest w tym samym punkcie co styk ochronny gniazda zasilającego urządzenie dołączone do interfejsu komputera. Może to powodować różnice potencjału między stykami ochronnymi obu gniazd. Najmniej przyjazna sytuacja może wystąpić w starszych instalacjach elektrycznych TN-C, gdzie styk ochrony jest łączony z przewodem neutralno-ochronnym (PEN). Wtedy potencjał styku zależy od prądu płynącego przez przewód PEN, jego rezystancji i rezystancji uziemienia. Różnica potencjałów wywoła przepływ prądów wyrównawczych w obwodzie masy interfejsu. Powstała w ten sposób pętla masy (masa interfejsu, ziemia) powodować będzie zakłócenia, mogące uniemożliwiać nawiązanie transmisji. Gdy ta różnica potencjałów jest duża (kilka–kilkadziesiąt V) możliwe jest uszkodzenie tak podłączonych urządzeń. Innym przykładem jest podłączenie komputera do mikrokontrolera (np. podczas „debuggerowania” oprogramowania w docelowym układzie) zasilanego za pomocą zasilacza beztransformatorowego nie zapewniającego izolacji galwanicznej. Zachodzi wtedy potrzeba zabezpieczenia komputera przed uszkodzeniem i, co bardziej istotne, zabezpieczenia programisty przed porażeniem. Dodanie do podstawowej aplikacji FT232 kilku fotoelementów zapewnia optoizolacje galwaniczną, która eliminuje opisany problem masy, kosztem niższej osiągalnej prędkości transmisji, która ograniczona jest pasmem przenoszenia transoptorów. Dodatkowo w artykule zawarto opis prostego modułu umożliwiającego łatwą realizację programowej obsługi łącza szeregowego w systemie Windows.

Rys. 1



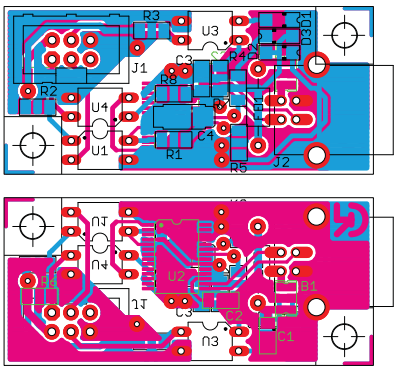
Jak to działa?

Na **rysunku 1** przedstawiono schemat ideowy. Układ U2 pracuje w typowej aplikacji, zasilanej z szyny VBUS komputera i ze względu na mnogość występujących w literaturze opisów, chyba nie wymaga szerszego komentarza. Transoptory dołączone do linii TXD, RXD U2 wraz z rezystorami im towarzyszącymi, realizują izolację galwaniczną interfejsu bez odwracania fazy przesyłanych przebiegów. Odpowiednio U1 do nadawania i U4 do odbioru danych przez mikrokontroler. Do połączenia z mikroprocesorem przewidziano złącze J1. Wymagane jest tu podłączenie napięcia zasilającego, tego samego, z którego zasilany jest mikrokontroler. Wyjście !PWR może być przydatne do uzyskania informacji o podłączeniu interfejsu do komputera PC, jeżeli mikrokontroler takiej informacji wymaga. Do wykrycia podłączenia wykorzystano napięcie VBUS (+5V) ze złącza J2, mimo że nóżka 12 (CBUS4) w U2 pełni podobną funkcję. Takie rozwiązanie zostało przyjęte ze względu na łatwiejsze prowadzenie ścieżek na PCB. Podłączenie komputera PC spowoduje przepływ prądu przez R4, LED w U3 i LED sygnalizacyjną D1. Oświetlony fototranzystor U3 „ściągnie" podciągnięte przez R3 do zasilania VCC wyjście !PWR do masy systemu mikroprocesorowego.

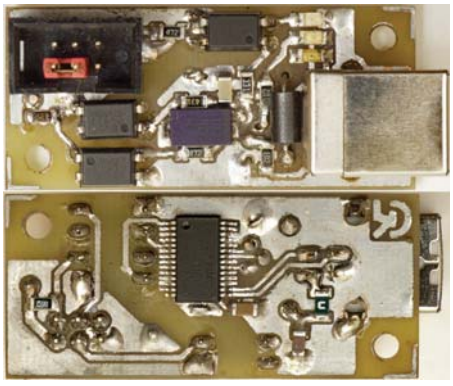
Do nawiązania komunikacji szeregowej między komputerem PC a systemem mikroprocesorowym można użyć programu emulującego terminal RS232. Niejednokrotnie jednak, ze względu na specyfikę przesyłanych danych, ich wizualnej prezentacji, możliwości programu terminalowego okażą się niewystarczające. W takich przypadkach potrzebny, dedykowany do „sterowania" konkretnym urządzeniem mikroprocesorowym program należy napisać samodzielnie. Transmisję szeregową w aplikacji dla systemu Windows, gdzie port szeregowy (czy to sprzętowy, czy to wirtualny) jest reprezentowany jako plik urządzenia, najłatwiej uzyskać, korzystając z gotowych bibliotek (modułów, nagłówków). Biblioteki obsługujące łącze szeregowe „opakowują" w sposób przyjazny dla programisty „mniej przyjazne" elementarne funkcje API realizujące dostęp do pliku, tj. odczyt, zapis czy modyfikację ustawień portu. Napisany przeze mnie i umieszczony w Elportalu, wśród materiałów dodatkowych do niniejszego artykułu, moduł ***RsPort.pas*** umożliwia szybkie i proste nawiązanie połączenia z portem szeregowym z projektowanej aplikacji dla Windows „pisanej" w języku Object Pascal. Jego zaletą jest prostota, umożliwiająca przez analizę kodu łatwe zrozumienie podstaw obsługi łącza szeregowego w Windows. Moduł zawiera definicję klasy *TRsPort*, która jest potomną klasy *TThread* (wątek). Umożliwia wielowątkową obsługę transmisji szeregowej, zarówno w aplikacjach dla 32 bitowej konsoli Windows, jak i aplikacji z graficznym interfejsem użytkownika (GUI). Wielowątkowość w tym przypadku oznacza tyle, że nadawanie i odbiór bajtów odbywa się w tle działającej aplikacji, nie wstrzymując jej głównego wątku. Za to, co wykonuje się w wątku, odpowiada metoda chroniona Execute. Zawiera ona pętlę wysyłającą dane do nadajnika portu, odczytującą dane z bufora odbiorczego, sprawdzająca zmianę stanu linii wejściowych. Warunkiem zakończenia pętli jest wywołanie destruktora klasy. Oprócz instrukcji odbierających, wysyłających, sprawdzających stan linii, wewnątrz pętli zastosowano procedurę opóźniającą Sleep(). Czas opóźnienia (próbkowania łącza) jest wliczany i wynosi tyle, co mniej więcej przesłanie jednego bajtu, zależnie od ustawionej prędkości bitowej portu. Ponieważ dane są buforowane, nie powoduje to problemu „gubienia znaków" i oszczędza w sposób znaczący zużycie czasu procesora (z 50% do poniżej 1%) w trakcie oczekiwania na dane. Ponieważ opóźnienie jest wywoływane w wątku próbkującym, więc nie ma wpływu na wątek główny aplikacji (jest to niezauważalne dla użytkownika). W porównaniu z użyciem skomplikowanych odwołań do struktury Overlapted w celu sterowania wątkiem próbkującym, jest to znacznie prostszym zabiegiem oszczędzającym użycie czasu procesora. Klasa umożliwia generowanie zdarzeń: gdy odbiornik odbierze bajt (ramkę danych), gdy na którejkolwiek kontrolnej linii wejściowej portu nastąpi zmiana stanu. Aby obsłużyć zdarzenia, należy zdefiniować metody ich obsługi w głównej klasie projektowanej aplikacji i przypisać je do pól obiektu, odpowiednio **Port.OnRxEvent** i **Port.OnCtrlInEvent**. Do odczytu danych z bufora służy metoda **Port.RxStr**, która zwraca wszystkie bajty zgromadzone w buforze i powinna być wywoływana w definicji metody obsługującej zdarzenie OnRxEvent. Interpretacja danych odebranych w dużej mierze zależy od tego, jakie dane są przesyłane do komputera PC. Jeżeli dane odbierane tworzą uporządkowane ciągi i wymagają parsowania, to odbierane dane do czasu wystąpienia znacznika końca danych (zwykle kody $0A$0D) należy przechowywać w zadeklarowanym polu typu łańcuchowego klasy głównej aplikacji i gdy łańcuch zwrócony przez metodę RxStr zawiera znacznik końca danych, podjąć odpowiednie działania i pod pole podstawić bajty występujące po wystąpieniu znacznika. Inną metodą jest zliczanie odebranych bajtów w zmiennej licznikowej „widzianej" w klasie głównej aplikacji i po osiągnięciu oczekiwanej liczby interpretacja danych zapamiętanych w polu łańcuchowym klasy głównej aplikacji, i wyzerowanie zmiennej licznikowej. **Listing 1** przedstawia przykład deklaracji i definicji metod obsługujących zdarzenia w klasie głównej aplikacji. Zgodnie z regułami programowania obiektowego przed korzystaniem z pól i metod klasy TRsPort powinno nastąpić wywołanie konstruktora **Create**, a przed zamknięciem aplikacji zwolnienie pamięci destruktorem **Free**. Przykładowo dla aplikacji konsolowej wywołanie konstruktora TRsPort może wystąpić w definicji konstruktora klasy głównej. Natomiast zwolnienie pamięci po obiekcie TRsPort może wystąpić w definicji destruktora klasy głównej. W aplikacji okienkowej (GUI) może to wyglądać tak: wywołanie konstruktora w metodzie obsługującej zdarzenie OnCreate, a destruktora w obsłudze zdarzenia OnDestroy. Otwarcie portu do transmisji metodą **Port.Open**

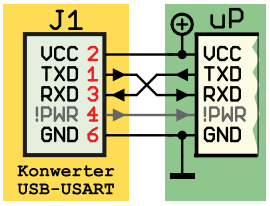
```
//-------------------------------------------
uses {Lista użytych modułów},RsPort;
type //Deklaracja klasy głównej aplikacji:
  TMain=class
  private
    Port: TRsPort;
    procedure OnRxData(Sender: TObject);
    procedure OnCtrlChange(Sender: TObject);
  public
    constructor Create;
    destructor Destroy;
  end;
implementation
//-------------------------------------------
constructor TMain.Create;
begin
  Port:=TRsPort.Create; //Utworzenie obiektu.
  //Przypisanie metod obsługi zdarzeń:
  Port.OnRxEvent:=OnRxData;
  Port.OnCtrlInEvent:=OnCtrlChange;
  //Wybór portu i parametrów transmisji:
  Port.Settings:='COM1:115200,N,8,1';
  Port.Open; //Otwarcie portu.
  //Ustawienie linii DTR w stan wysoki (On):
  Port.SetCtrlOut(Port.CtrlOut+[DTR]);
  //Przykładowe wysłanie znaków:
  Port.TxStr('Helo!'#13#10);
end;
//-------------------------------------------
destructor TMain.Destroy;
begin
  if Port.Connected then //Port otwarty?
    Port.Close; //Zamknięcie portu.
  Port.Free; //Zwolnienie pamięci po obekcie.
end;
//-------------------------------------------
//Definicja obsługi zdarzenia gdy w buforze
//odbiornika znajdują się nieodebrane bajty.
procedure TMain.OnRxData(Sender: TObject);
begin
  //Odczyt bajtów z bufora metodą: Port.RxStr
  //np. dla aplikacji konsolowej
  Write(Port.RxStr); //ich wydruk na ekranie.
end;
//-------------------------------------------
//Definicja obsługi zdarzenia zmiany stanu
//wejściowej linii kontrolnej.
procedure TMain.OnCtrlChange(Sender: TObject);
begin
  //Stosowne działanie w zależności na której
  //linii nastąpiła zmiana stanu
  //np. dla lini CTS w aplikacji konsolowej:
  if CTS in Port.CtrlIn then
    WriteLn('CTS On') else
  WriteLn('CTS Off');
end;
```

Listing 1



Rys. 2





Rys. 3

powinno być poprzedzone przypisaniem łańcucha znakowego z definicją parametrów transmisji do pola **Port.Settings** zgodnie ze specyfikacją funkcji API BuildComDCB(). Gdy otworzono port z dodatkowym parametrem **X**, włączony zostaje programowy protokół XOn/XOff znaki sterujące ($11, $13) nie są wstawiane ciągu bajtów odebranych. W innych przypadkach odbierane są wszystkie możliwe wartości numeryczne bajtów (0...255). W przypadku dodatkowego parametru **P** przy otwarciu portu włączane jest sprzętowe sterowanie przepływem, liniami kontrolnymi. Wysłanie ciągu bajtów (znaków) następuje po wywołaniu metody **Port.TxStr(aStr)**, która zapisuje bajty określone parametrem *aStr* do bufora nadawczego nadajnika (podwójne buforowanie). Mimo że *aStr* jest typu łańcuchowego, możliwe jest wysyłanie bajtów (znaków) niedrukowanych, sterujących, w tym znaku pustego o kodzie równym zero. Zastosowanie do nadawania funkcji API TransmitCommChar() jest może mało eleganckie, lecz gdy do sterowania przepływem zastosowano programowy protokół Xon/XOff, ma tę zaletę, że nie zawiesza aplikacji w przypadku odebrania znaku XOff (możliwy jest odbiór danych przez ReadFile(), gdy nie wszystkie dane z bufora nadawczego są wysłane). Pozwala to uniknąć niespodzianek np. niemożliwość odbioru danych w tym znaku XOn (!), jak ma to miejsce w przypadku bardziej wydajnej funkcji WriteFile(). Gdy nie jest stosowane sprzętowe sterowanie przepływem i istnieje potrzeba nietypowego wykorzystania kontrolnych linii wyjściowych możliwe jest zmienianie ich stanu metodą Port.SetCtrlOut(aValue), gdzie pod parametr aValue można przypisać kombinacje elementów zbioru TCtrlOutput ([DTR],[RTS]). Oprócz ww. metod klasa TRsPort posiada dwie metody „informacyjne" zwracające nazwy i klucze rejestru portów zainstalowanych w systemie, w tym nazwy portów wirtualnych (VCP). W Elportalu, wśród materiałów dodatkowych do niniejszego artykułu, zamieszczono listingi prostych aplikacji przykładowych (Borland Delphi), dla konsoli 32-bitowej Windows i aplikacji okienkowej (GUI). Czytelnicy posiadający podstawową znajomość języka Delphi nie powinni mieć problemów ze zrozumieniem ich działania.

Montaż i uruchomienie

Dwuwarstwowy obwód drukowany widoczny jest na **rysunku 2**. Montaż po sprawdzeniu PCB warto zacząć od przylutowania układu U2 i innych elementów SMD. Jako ostatnie należy montować elementy przewlekane.

W przypadku niewykorzystywania wyjścia informacyjnego o podłączeniu interfejsu do gniazda USB komputera PC transoptora U3 wraz z elementami towarzyszącymi nie należy montować. Uruchomienie układu sprowadza się do podłączenia zasilania do złącza J1 +5V (2) GND (6), zwarcia ze sobą linii TX, RX. Po podłączeniu do komputera PC powinna nastąpić zmiana stanu na linii !PWR (4 J1) z wysokiego na niski. Ze względu na ograniczone pasmo przenoszenia transoptorów, próba przesyłu danych wykaże niemożliwość uzyskania najwyższych typowych prędkości transmisji. Możliwe do osiągnięcia prędkości w większości przypadków będą jednak całkowicie wystarczające. Im większy prąd płynący przez LED transoptorów tym szersze pasmo przenoszenia. Dobór rezystorów ograniczających prąd LED transoptorów (oprócz czynników: Vf LED i napięcia zasilającego) ograniczony jest wydajnością prądową linii dołączonego mikrokontrolera i układu U2. Test prędkości można przeprowadzić programem Test232 zawartym w materiałach dodatkowych. Połączenie ze współpracującym mikrokontrolerem należy wykonać w sposób widoczny na **rysunku 3**. Przy montażu układu w docelowym urządzeniu szczególną uwagę należy zwrócić na odizolowanie złącza J2 od obudowy, gdy jest wykonana z materiału przewodzącego.

Wykaz elementów

R4	270Ω 1206
R2, R5, R7, R8	470Ω 1206
R1, R3, R6	4,7kΩ 1206
C1	10n 1206
C2, C3	100n 1206
C4	100u/6V 6032-28 Tantal
D1, D2, D3	LED 1206
U1, U3, U4	LTV817
U2	FTDI232RL
J1	ML06V
J2	USB-B
B1	100mA PTC
FE1	Ferryt przeciwzakłóceniowy

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-3093.

Cyprian Kamil Kowalski
c4v2@o2.pl

Literatura:

Andrzej Daniluk, „RS 232C – Praktyczne programowanie. Od Pascala i C++ do Delphi i Buildera", Helion, wyd. II.