HIDMaker – USB dla każdego

Kiedy patrzy się na projekty urządzeń mikroprocesorowych, szczególnie tych mniej skomplikowanych, to do komunikacji z komputerem rzadko jest używany interfejs USB. Zazwyczaj konstruktor wykorzystuje dodatkowy układ – na przykład FT232 – realizujący funkcję wirtualnego interfejsu UART. Taki konwerter ma z jednej strony interfejs UART, z którym łączy się UART mikrokontrolera, a z drugiej interfejs USB. Specjalny sterownik uruchomiony na komputerze PC powoduje, że połączenie z FT232 jest widziane przez komputer jako dodatkowy, wirtualny port COM. Czy konstruktor „małych" urządzeń zawsze jest skazany na stosowanie konwerterów?

Podziękowanie
Autor artykułu dziękuje panu Robertowi Millerowi, właścicielowi firmy Trace System Inc. za dostarczenie do testów HIDmaker'a w wersji dla mikrokontrolerów PIC18F i PIC32MX

Obsługa połączenia PC z urządzeniem zewnętrznym poprzez port szeregowy COM jest bardzo dobrze opisana i niezbyt skomplikowana, więc trudno dziwić się, że takie rozwiązanie zdobyło dużą popularność. Jednak przy zastosowaniu mikrokontrolera z wbudowanym, sprzętowym interfejsem USB użycie dodatkowego układu emulującego interfejs UART jest bardzo kontrowersyjne, ponieważ niepotrzebnie podwyższa koszt opracowania i gotowego wyrobu. Nie bez znaczenia jest też fakt, że interfejs UART ma ograniczoną i niższą niż USB prędkość transmisji. Dla większej serii urządzeń zastąpienie układu typu FT232 wbudowanym interfejsem USB może przynieść wymierne korzyści. Jednak jest to łatwe, a podstawową trudność stanowi oprogramowanie stosu komunikacyjnego USB. Wynika to pośrednio z organizacji połączenia *host – device*. Całym przepływem danych przez USB kieruje *host*.

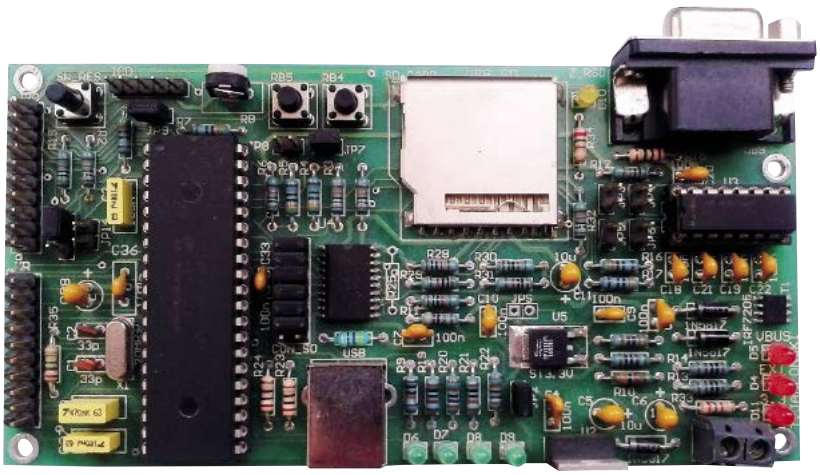
Host USB (najczęściej komputer PC) zawiera *USB Host Controller* i *Root Hub*. Oba komponenty współpracują, by umożliwić systemowi operacyjnemu komunikację z urządzeniami zewnętrznymi poprzez interfejs USB. *Host Controller* formatuje dane przed ich wysłaniem do urządzenia zewnętrznego oraz tłumaczy odebrane dane na postać zrozumiałą dla systemu operacyjnego. Ponadto, wykonuje szereg funkcji potrzebnych do zarządzania przepływem danych przez USB. *Root Hub* (również umieszczony w PC) przy współpracy z *Host Controller-em* wykrywa urządzenia zewnętrzne, wykonuje wszystkie polecenia *hosta* i przesyła dane pomiędzy urządzeniami i *Host Controller-em*. Urządzeniami zewnętrznymi są układy peryferyjne oraz rozgałęźniki (*hub*) dołączone do magistrali. Hub to urządzenie mające jeden lub więcej portów służących do dołączenia kolejnych urządzeń (układów peryferyjnych lub kolejnych hubów). Każde urządzenie musi zwierać sprzęt i oprogramowanie, które umożliwia komunikowanie się z hostem.

To szczególne uprzywilejowanie hosta wynikające przydzielonych mu zadań może się stać przyczyną problemów w czasie programowania i testowania własnych urządzeń dołączanych do USB. Host wysyła poprzez interfejs szereg komend i oczekuje na nie ściśle określonych odpowiedzi, w okre. Jeżeli urządzenie nie odpowie prawidłowo i to w określonych ramach czasowych, to inicjalizacja połączenia lub transfer danych zostaną przez hosta wstrzymane. Dodatkowo, nie będziemy mieli żadnych informacji ze strony hosta o przyczynie błędów. Dlatego programista musi znać dość szczegółowo „zasady ruchu" na magistrali, jeżeli chce samodzielnie napisać program do obsługi USB. Może też korzystać z gotowych przykładów i na ich podstawie wykonać transmisję danych. Trzecia możliwość to korzystanie z programów narzędziowych wykonujących za programistę większość „brudnej roboty".

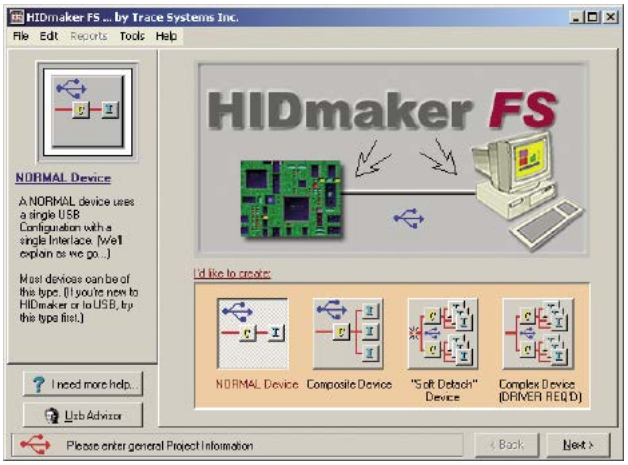
Jedną z firm szczególnie wspomagających konstruktorów w zastosowaniach interfejsu USB jest Microchip. Dobrze znane i szeroko wykorzystywane mikrokontrolery PIC18F4550 były jednymi z pierwszych, w których wbudowano bardzo dobry moduł peryferyjny USB. Obecnie jest dostępnych wiele mikrokontrolerów tej firmy z rodzin PIC16, PIC18, PIC24 i PIC32 z takim modułem. Microchip tradycyjnie wspomaga swoich klientów firmowymi programami demonstracyjnymi pomagającymi znacząco skrócić czas tworzenia aplikacji. Ja również korzystałem z tej pomocy i wyniki były bardziej niż zadowalające.

Tak było do czasu, kiedy natknąłem się w Internecie na informację o programie **HIDMaker** firmy **Trace Systems**, tej samej, która oferuje opisywany już przeze mnie program TCPMaker przeznaczony do tworzenia aplikacji internetowych. HIDMaker jest przykładem narzędzia zaprojektowanego ze zwróceniem szczególnej uwagi na szybkość tworzenia aplikacji i jej jakość. Obsługa programu jest tak łatwa i intuicyjna, jak tylko można to sobie tylko wyobrazić. Dodatkowo, ta łatwość idzie w parze olbrzymimi możliwościami.

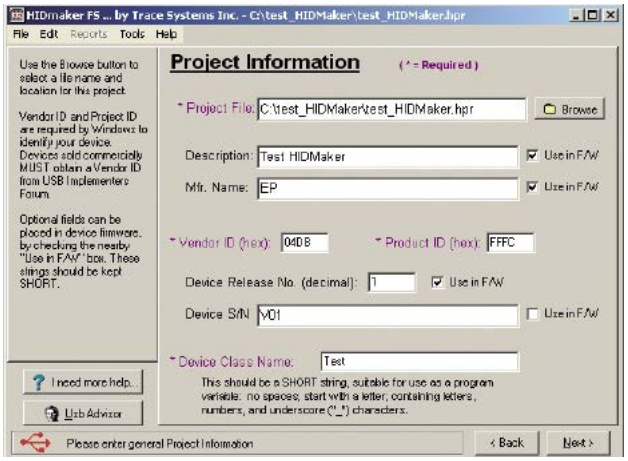
Zazwyczaj implementujemy USB w mikrokontrolerze po to, aby połączyć się z komputerem PC spełniającym rolę hosta. Jeżeli „program obsługi" interfejsu USB jest prawidłowo napisany, to host z urządzeniem na-



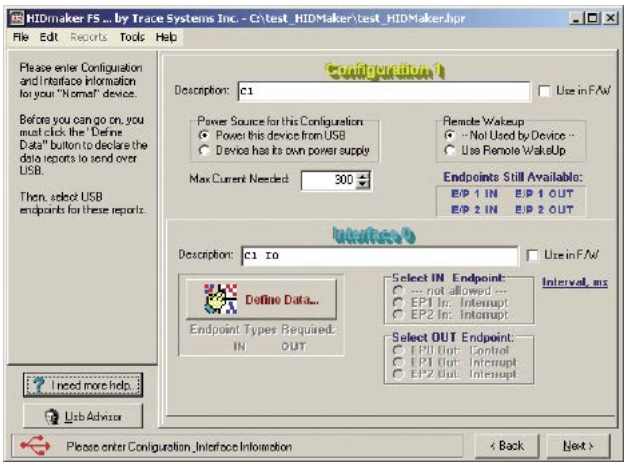
Fotografia 1. Moduł testowy z mikrokontrolerem PIC18F4450



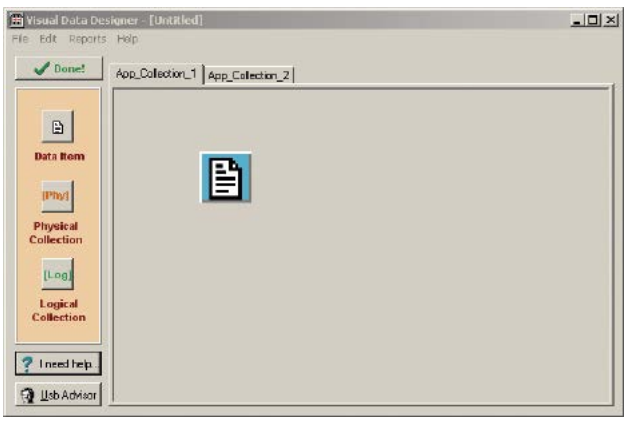
Rysunek 2. Okno tytułowe HIDmakera



Rysunek 3. Okno informacji o projekcie



Rysunek 4. Ekran ustawień Konfiguracji i Interfejsu



Rysunek 5. Okno Visual Data Designer

wiążą automatycznie łączność, ale trzeba jeszcze napisać aplikację, która potrafi wysyłać i odbierać dane. To dla wielu elektroników-programistów jest dużym wyzwaniem. HIDMaker załatwia ten problem kompleksowo: generuje kod zarówno dla mikrokontrolera, jak i dla komputera pracującego pod kontrolą systemu Windows. Aplikacje dla komputera można tworzyć w różnych środowiskach, z wykorzystaniem różnych kompilatorów języków wysokiego rzędu. Tu również HIDmaker staje na wysokości zadania oferując możliwość współpracy z kilkoma najbardziej znanymi kompilatorami. Kody generowane przez HIDMakera są czymś w rodzaju szkieletów, na których programista buduje działająca aplikację.

Jak to działa? Najlepiej jest to pokazać na konkretnym przykładzie. Potrzebny będzie do tego celu układ z mikrokontrolerem PIC wyposażonym w sprzętowy interfejs USB, kompilator języka C i jedno z obsługiwanych przez HIDMaker'a środowisk programistycznych do pisania aplikacji dla komputera PC pracującego pod kontrolą systemu operacyjnego Windows. Aplikacja przykładowa będzie napisana i uruchomiona pod tym właśnie systemem.

Moduł testowy

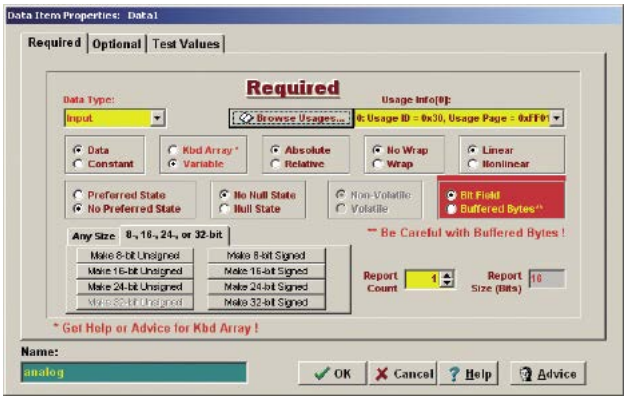
Do testów zastosujemy pokazany na **fotografii 1** moduł z mikrokontrolerem PIC18F4450, opisywany kiedyś na łamach Elektroniki Praktycznej. Moduł ma wszystko, co potrzeba do przeprowadzenia testów: potencjometr połączony z wejściem przetwornika A/C, dwa klawisze połączone z liniami portów RB4 i RB5 oraz diody LED sterowane za pomocą portów. Oczywiście jest wyposażony również w złącze USB.

Pakiet HIDMaker jest pakietem typu Visual Data Designer, a więc jest to środowisko graficzne i na etapie projektowania szkieletu nie ma potrzeby pisania jakichkolwiek procedur w języku programowania. Przed rozpoczęciem pracy z HIDmaker'em musimy sobie powiedzieć o zdefiniowanych konfiguracjach i interfejsach urządzenia USB.

Konfiguracja USB to rodzaj ustawień. Większość urządzeń USB ma jedną konfigurację, ale mogą być urządzenia, które mają ich więcej. W danym momencie urządzenie może pracować tylko z jedną konfiguracją. Przykład więcej niż jednej konfiguracji, to praca urządzenia w trybie „Low Power", kiedy USB nie może dostarczyć wytaczającej ilości energii do zasilania wszystkich funkcji urządzenia. Wtedy konfiguracja wyłącza mniej ważne funkcje. Kiedy ilość energii jest wystarczająca, to zostaje wybrana konfiguracja „High Power" i urządzenie może pracować z pełnymi możliwościami.

Innym przykładem mogą być różne konfiguracje dla różnych dostępnych prędkości transmisji. Interfejs USB, to wydzielona, niezależna funkcjonalność. W oprogramowaniu systemu Windows interfejs jest używany do „otwarcia" kanału komunikacyjnego. Jest to analogia do otwierania plików. Tak jak można otworzyć w systemie kilka plików i zapisywać oraz odczytywać z nich dane, tak samo można „otworzyć" kilka interfejsów w jednym czasie i komunikować się z nimi oddzielnie. Konfiguracja USB może zawierać wiele interfejsów USB. Przykładem urządzenia z wieloma interfejsami może być wielofunkcyjna drukarka biurowa. W jednej obudowie z jednym złączem USB są umieszczone oddzielnie: drukarka, skaner i faks.

Najprostsza z konfiguracji i jednocześnie jedna z najczęściej stosowanych, nosi nazwę *Normal Device*. Zawiera ona jedną konfigurację USB i jeden interfejs USB. Kolejna konfiguracja *Composite Device* ma jedną konfigurację i kilka interfejsów, tak jak wspomniana wyżej drukarka wielofunkcyjna. HIDMaker ogranicza liczbę interfejsów do trzech. Jeżeli jest potrzeba więcej niż jednej konfiguracji, a każda z konfiguracji ma więcej niż jeden interfejs, to jest stosowana konfiguracja *Complex*. W praktyce są z nią problemy. Pierwszy z nich, to konieczność napisania drivera, który miałby możliwość przełączania pomiędzy konfiguracjami. Kolejnym problemem jest niezbyt dobra współpraca systemu Windows z *Composite Device*. Twórca HIDmakera, chociaż umieścił tę konfigurację w swoim programie, to ze względu na wyżej wymienione problemy nie poleca jej stosowania. Zamiast niej może być użyta konfiguracja *Soft Detach* wykorzystująca właściwość *Soft Detach* systemu Windows. Polega ona na „oszukiwaniu" systemu USB, w taki sposób, że dostaje on informację,



Rysunek 6. Okno Data Item Properties

iż urządzenie USB zostało odłączone od magistrali, a następnie ponownie dołączone. Wymusza to uruchomienie procesu ponownej enumeracji. Kiedy to nastąpi, urządzenie może się zgłosić z inną konfiguracją. Można w ten sposób przełączać się pomiędzy konfiguracjami *Normal* i *Composite*. Dla urządzenia Soft Detach HIDMaker może definiować maksymalnie dwie konfiguracje, a każda nich z maksymalnie trzema interfejsami.

Po uruchomieniu HIDMaker'a pojawia się ekran tytułowy pokazany na **rysunku 2**. Dokonuje się tutaj wyboru poprzez kliknięcie na jedną z ikon w oknie *I'd like to create*. Na rys. 2 wybrano *Normal Device*, bo tej konfiguracji użyjemy w naszych testach. Z lewej strony okna jest umieszczona ikona wybranej konfiguracji i krótki opis w języku angielskim. Po wyborze przechodzimy do następnego ekranu po kliknięciu na belkę *Next*.

HIDMaker pracuje w oparciu o plik projektu. Zawiera on informację o wszystkich wybranych ustawieniach i dołączonych plikach. Zapisanie tworzonego projektu umożliwia ponowne jego otwarcie i kontynuowanie pracy od momentu jej przerwania. Ścieżkę dostępu oraz nazwę pliku projektu wybieramy w oknie *Project File* po kliknięciu na przycisk *Browse* (**rysunek 3**).

Kolejne dwa okna są opcjonalne przeznaczone do opisu projektu i nazwy wytwórcy. Okna *Vendor ID* i *Product ID* zawierają identyfikatory wymagane przez system Windows do identyfikacji urządzenia. Dla potrzeb naszego testu możemy nadać dowolne wartości, ale aby urządzenie mogło być sprzedawane, trzeba te wartości uzyskać z *USB Implementers Forum*. Ostatnią wymaganą informacją jest *Device Class Name*. Ponieważ ta nazwa będzie używana w nazwach zmiennych w wygenerowanych szkieletach programowych, to powinna być krótka i jednoznacznie identyfikować urządzenie.

Po zapisaniu wszystkich niezbędnych ustawień klikamy przycisk *Next* i przechodzimy do kolejnego ekranu (**rysunek 4**). Górna połowa okna jest przeznaczona do definiowania ustawień konfiguracji (*Configurations 1*), a dolna do definiowania ustawień interfejsu (*Interface 0*). Na rys. 4 pokazano możliwość ustawiania jednej z konfiguracji. Jest to wynik wybrania urządzenia typu *Normal* (rys. 3). Ustawiamy tu sposób zasilania urządzenia z magistrali USB (dozwolone lub nie) oraz zdalne wybudzanie (*remote wake-up*). W obszarze ustawień Interfejsu najpierw klikamy na belkę *Define data*. Pojawia się okno *Visual Data Designer*. W tym momencie możemy dodawać zmienne zawierające dane przesyłane w obu kierunkach przez USB. Żeby to zrobić, trzeba kliknąć na *Data Item* z przybornika umieszczonego z lewej strony okna, a potem kliknąć w polu okna *APP_Collection_1* i postawić ikonkę danej (**rysunek 5**). Po dwukrotnym kliknięciu na te ikonkę otwiera się okno *Data Item Properties*, w którym są definiowane właściwości danej. Okno ma trzy zakładki: *Required*, *Optional* i *Test Value*. Nas najbardziej będzie interesowała zakładka *Required* (wymagane). Najpierw w oknie *Name* wpisujemy nazwę zmiennej. Dobrze, jeśli ta nazwa w jakiś sposób koresponduje z przesyłanymi danymi. W naszym wypadku będzie to analog, bo będzie przeznaczona do przesyłania wartości odczytanej z przetwornika A/C. Ta nazwa będzie pojawiała się w definicjach zmiennych, zarówno po stronie programu mikrokontrolera, jak komputera PC.

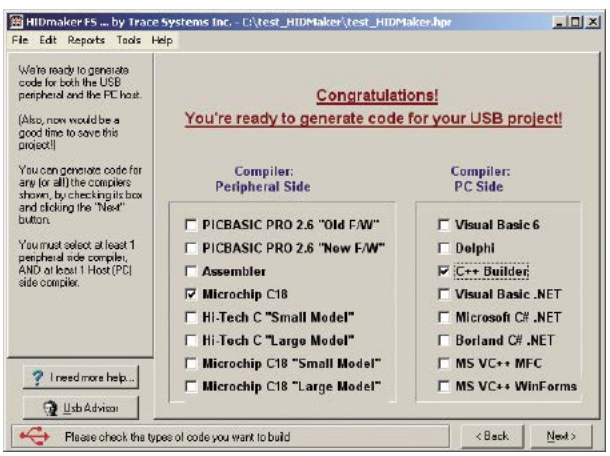
Kierunek przesyłania danych z punktu widzenia hosta wybieramy w okienku *Data type*. Nasza dana będzie przesyłana z urządzenia (mikrokontrolera) do hosta (PC) i dlatego musi być zdefiniowana, jako wejściowa (*Input*). Zaznaczamy też kolejno opcje: *Data*, *Variable*, *No Warp*, *Linear*, *No Preferred State* i *No Null State*. W lewym dolnym rogu okna jest wybierana długość zmiennej i to czy ma być ze znakiem (*signed*), czy bez znaku (*unsigned*). Na koniec, z rozwijanego menu *Browse Usages* wybieramy dowolne ID, tak aby nie pokrywały się z ID pozostałych zmiennych w projekcie. Na potrzeby projektu testowego zdefiniujemy jeszcze jedną daną o nazwie *control* typu *Output* o długości 8 bitów bez znaku. Będzie to dana przesyłana z hosta do urządzenia. Po zdefiniowaniu danych i kliknięciu na przycisk Done ponownie jest wyświetlany ekran z ustawieniami *Configuration 1* i *Interface 0*. Teraz można tu przypisać dane do endpointów i interwał przesyłania danych dla transferu *Interrupt*.

Jeżeli wszystko jest ustawione, to możemy przejść do następnego etapu projektowania, czyli generowania kodu wynikowego dla projektu USB. Jak już wspominałem kod wynikowy może być generowany dla różnych kompilatorów, zarówno po stronie mikrokontrolera, jak i po stronie komputera. Projekt po stronie mikrokontrolera może być wygenerowany w języku C dla kompilatorów firm Hi-Tech lub Microchip, w asemblerze lub w Basicu dla kompilatora PICBASIC PRO. Ponieważ my użyjemy modułu z PIC18F4550 i kompilatora MPLAB C-18 , to zaznaczamy w oknie *Compiler Peripherial Side* opcję *Microchip C18* (**rysunek 7**).

Po stronie komputera wybór dostępnych kompilatorów jest spory i większość programistów znajdzie coś odpowiedniego dla siebie. Ja zaznaczyłem opcję środowiska projektowego *C++ Builder*, niegdyś oferowanego przez firmę Borland, która została wchłonięta przez inną firmę, ale wielu programistów nadal używa C++ Builder w wersji 5, lub 6. Niefortunnie, bardzo popularna wersja darmowa tego pakietu projektowego nie nadaje się do kompilowania projektu wygenerowanego przez HIDmaker'a – niezbędne jest użycie wersji pełnej *Enterprise*. Po wybraniu kompilatorów klikamy na przycisk *Next*, a program – na podstawie opisanych ustawień – generuje pliki w katalogu projektu projektu: dla mikrokontrolera w podkatalogu *PIC*, a dla komputera PC w podkatalogu *Host*.

Testowanie połączenia USB zaczniemy od otwarcia projektu w MPLAB IDE 8x. Projekt składa się z 4 plików źródłowych, w tym trzech napisanych w asemblerze. Z punktu widzenia użytkownika najważniejszy jest plik w języku C. Można tam znaleźć szereg bardzo pomocnych komentarzy, które ułatwiają zrozumienie, jak program odbiera i wysyła dane.

HIDMakerdomyślnie dołącza do projektu plik nagłówkowy *p18f4550.h* prze-



Rysunek 7. Wybór kompilatorów dla plików wynikowych

Listing1. Pętla wykrywania czy została odebrana dana
```
do
{
    USBService();  //cykliczne wywyoływanie USBService.
    ReportInProgress = ReadAndUnpackOnePacket(1, (rom unsigned char const
*)&Ep1RcvVarTable);
    if ( (ReportInProgress == (unsigned char)0xFF) || (ReportInProgress ==
(unsigned char)0))
    {
        HandleEp1Rcv = TRUE;
    }
} while (ReportInProgress == 0xFF);
```

Listing 2. Wykrycie odebrania danej z Endpiont 1 OUT
```
if (HandleEp1Rcv)
{
    // procedury wykonywane po odebraniu danej
    PORTD=control; //sterownie diodami LED
    HandleEp1Rcv = 0; // sygnalizacja obsłużenia odebranej danej
}
```

Listing 3. Konfiguracja i włączenie modułu A/C
```
TRISA = 0xff;  // PORTA wszystkie linie wejściowe
ADCON1 = 0b00001110; // odblokowanie kanału AN0
ADCON2 = 0b10100101; // right  justified, 8TAD, FOSC/16
SelChanConvADC(ADC_CH0);//ustawienie kanał 0
ADCON0 = 1; //włączenie modułu przetwornika Uref=Uzasilania
```

Listing 4. Pomiar napięcia przez A/C i wysłanie wartości przez endpoint IN
```
ConvertADC();  //start konwersji
while(BusyADC());  //czekaj na zakończenie konwersji
analog=(ReadADC());  //zwróć wynik konwersji
EP1XmtDataReady = 1;


if (EP1XmtDataReady)
{
    do
    {
        USBService();   // musi być wywoływany sekwencyjnie
        ReportInProgress = PackAndSendOnePacket(1, (rom unsigned char const
*)&Ep1XmtVarTable);
    } while (ReportInProgress);
}
```

znaczony dla mikrokontrolera PIC18F4550. Użytkownik może wybrać konfigurację dla mikrokontrolera: PIC18F4455, PIC18F2550, PIC18F2455, PIC18F4450, PIC18F2455. Jak widać, nie ma tu najnowszych, bardzo ciekawych mikrokontrolerów Microchipa, z udoskonalonym interfejsem USB. Po dokładnej analizie danych katalogowych zaawansowany użytkownik będzie mógł samodzielnie skorygować procedurę *USBInit()*.

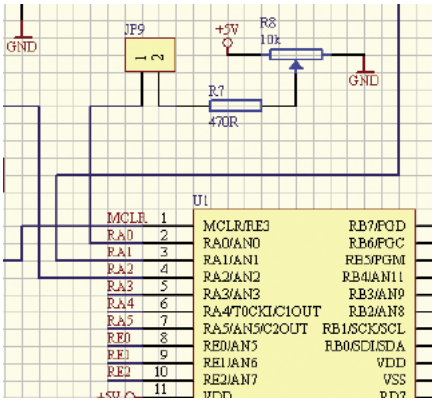
W funkcji *main()* jest umieszczona pętla nieskończona *MainLoop*, w której jest cyklicznie wywoływana procedura *USBService()* (napisana w asemblerze). Ta procedura obsługuje żądania (*requests*) wysyłane przez hosta do urządzenia. Kiedy USBService wykryje odebranie danej z hosta, ustawia znacznik *HandleEplRcv* (**listing 1**).

Jak pamiętamy za pomocą HIDmaker została zdefiniowana zmienna *control* typu *output*, czyli zmienna wyjściowa z punktu widzenia hosta. Ustawienie znacznika oznacza, że host wysłał zmienną, urządzenie, czyli nasz układ z PIC18F4550 odebrało ją i jej wartość z *Endpoint 1 OUT* została wpisana do zmiennej *control*. W szkielecie programu możemy testować warunek ustawienia znacznika i wykonywać jakieś działanie po odebraniu danej (**listing 2**).

Przy okazji omawiania programu uruchamianego od strony komputera PC zostanie pokazany sposób, w który można łatwo wykonać procedurę, na przykład, zaświecania lub gaszenia diody LED przez modyfikowanie zawartości zmiennej *control*. W module cztery diody świecące D6…D9 są połączone z liniami portu RD0…RD3.

Szkielet programu od strony urządzenia zostanie teraz uzupełniony o obsługę przetwornika A/C. W module jest zamontowany potencjometr R8 (10 k) zasilany napięciem +5 V. Jego suwak został połączony z wyprowadzeniem RA0/AN0 przez zworkę JP9. Moduł przetwornika A/C musi być odpowiednio skonfigurowany i włączony (**listing 3**). Po sprawdzeniu czy w *endponcie OUT* nie ma danej odebranej z hosta (sygnalizowanej ustawieniem *HandleEp1Rcv*), można przesłać dane do *endpointa IN*. W naszym wypadku będzie to 10-bitowa dana odczytana z przetwornika A/C. Na **listingu 4** pokazano fragment programu odpowiedzialny za: odczytanie danej z przetwornika A/C, ustawienie flagi *EP1XmtDataReady* sygnalizującej obecność danych do wysłania oraz cykliczne wywołanie *USBService* i wysłanie danych przez *endpoint IN*.

Fragmenty programu pokazane na listingach 2 i 4 pokazują, jak mało trzeba wysiłku programisty, aby dane mogły być przesyłane od/z hosta. Cała obsługa



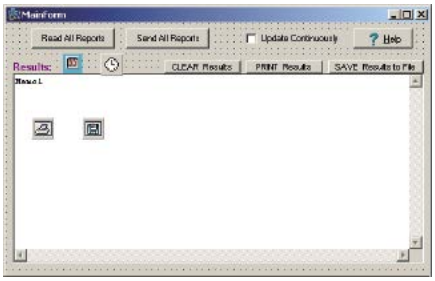
Rysunek 8. Połączenie R8 z RA0/AN0

transferu przez moduł sprzętowy USB jest umieszczona w szkielecie programu mikrokontrolera PIC18F4550.

Podobnie, jeżeli chodzi o wsparcie połączenia USB jest w przypadku szkieletu generowanego dla kompilatora Borland C++. Tutaj standardowo generowany jest program, który jest gotowym rozwiązaniem pozwalającym na natychmiastowe przetestowanie połączenia USB z prawidłowo zaprogramowanym mikrokontrolerem nawet wtedy, gdy w programie mikrokontrolera nie zostaną wykonane żadne modyfikacje.

W programie dla C++ Builder podstawowymi elementami szkieletu programu są komponenty: *memo*, dwa przyciski *Tbutton* nazwane *SendAllReports* i *ReadAllReports* oraz pole wyboru (check box) *Update Continously*. Do tego trzeba dodać bardzo ważny, niestandardowy komponent *HIDagent* utworzony i dostarczany przez twórcę programu HIDmaker. Aplikacja zawiera również komponent *Timer1* do odmierzania interwałów czasowych oraz mniej ważne (z punktu działania aplikacji podstawowej) komponenty: *Tbutton: Clear results*, *Print results*, *Save Results to File* i *Help*.

Okno formularza wygenerowanego przez projekt HIDmakera pokazano na **rysunku 9**. Użytkownik pakietu C++ Builder musi zainstalować kontrolkę *ActiveX HIDAgent*. Sposób konfigurowania Borlanda do poprawnej pracy z C++ Builder jest dokładnie opisany w pliku pomocy, w rozdziale zatytułowanym *Configuring Borland C++ Builder* i nie będziemy go



Rysunek 9. Okno formularza Builder C++ generowanego przez HIDMaker'a

Rysunek 10. Informacja o konieczności podłączenia zaprogramowanego modułu PIC

tutaj powtarza. W przypadku każdego z wybranych kompilatorów jest wymagana dodatkowa konfiguracja związana z komponentem *HIDAgent*.

Prawidłowo skonfigurowany Builder z zainstalowaną kontrolką ActiveX (*HIDagentXControl1 Libraray Cersion 1.2*) powinien skompilować projekt bez błędów i ostrzeżeń. Jeżeli zostanie uruchomiony bez dołączenia zaprogramowanego mikrokontrolera, to pojawi się informacja o braku możliwości połączenia i dalej program się nie uruchomi. Po dołączeniu modułu do portu USB i kliknięciu na *OK* w oknie *memo* pojawiają się wypisy o połączeniach interfejsów HID. System operacyjny Windows identyfikuje dołączone urządzenia USB klasy HID, tak jak je skonfigurowaliśmy w oknie *Browse Usages*. Te nazwy urządzeń nie maja znaczenia dla działanie naszej aplikacji.

Jeżeli zobaczymy takie komunikaty jak na **rysunku 11**, to oznacza to, że wszystko działa prawidłowo: moduł z mikrokontrolerem PIC18F4550 został poprawnie zaprogramowany a sprzęt po obu stronach połączenia USB jest sprawny.

Załóżmy, że wykonaliśmy modyfikację szkieletu programu dla mikrokontrolera, jak na list. 4. Wtedy jest odczytywana wartość napięcia z potencjometru, konwertowana na wartość liczbową, zapisywana do zmiennej analog i przesyłana do *endpointa IN*. Po kliknięciu na przycisk *ReadAllReports*, program komputera odczytuje poprzez USB wartość zmiennej analog i wypisuje ją w oknie komponentu *memo* (**rysunek 12**). Poprawność działania można sprawdzić zmieniając położenie suwaka potencjometru i klikając na *ReadAllReports*. Wartość zmiennej analog powinna również się zmieniać w zakresie od 0 do 1023.

Listing 5. Funkcja wywoływana po kliknięciu na ReadAllReports

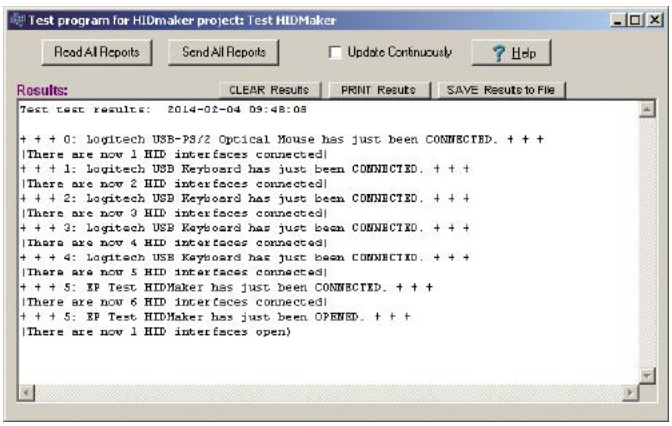
```
void __fastcall TMainForm::ReadRptsBtnClick(TObject *Sender)
{
AnsiString AStr;
int i, DevNum;

  ClearBtnClick(0);

  for (DevNum = 0; DevNum < MyDevList->Count; DevNum++)
    {
      AddToResults(0,"");
      AStr = "Device #" + IntToStr(DevNum) + ":";
      AddToResults(0,AStr);
      if (Test[DevNum]->ReadAllReports() )
        {
          AStr = "  analog = " + IntToStr(Test[DevNum]->analog-
>UnScaledValue);
          AddToResults(0,AStr);

        }
      else
        {
          Beep();
          AStr = "+ + + Unable to read from device"+ IntToStr(DevNum) + "! +
+ +";
          AddToResults(0,AStr);
        }
    }

}


void __fastcall TMainForm::SendRptsBtnClick(TObject *Sender)
{
//var
  AnsiString AStr;
  int i, DevNum;
  int MaxVal;
```

Listing 6 Funkcja wywoływana po kliknięciu na WriteAllReports

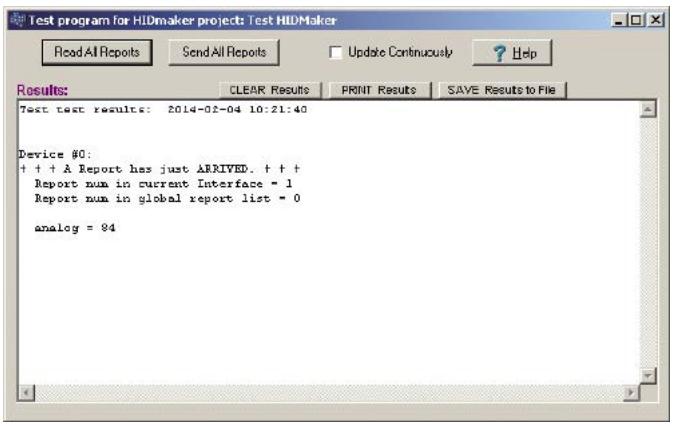
```
  for (DevNum = 0; DevNum < MyDevList->Count; DevNum++)
    {
      AddToResults(0,"");
      AStr = "Device #" + IntToStr(DevNum) + ":";
      AddToResults(0,AStr);

      Test[DevNum]->control->UnScaledValue = 0x00;

      AStr = Format("  Wrote to device: control = 0x%x",
                ARRAYOFCONST(( (int)Test[DevNum]->control->UnScaledValue )) );
      AddToResults(0,AStr);

      if ( Test[DevNum]->WriteAllReports() )
        {
          // Success
          AddToResults(0,"  Successfully wrote to device");
        }
      else
        {
          // Failure
          Beep();
          AStr = "+ + + Unable to send to device"+ IntToStr(DevNum) + "! + +
+";
          AddToResults(0,AStr);
        }
    }
}
```

Na **listingu 5** pokazano fragment programu obsługi przyciśnięcia przycisku *ReadAllReports*. Zapytanie o daną z *endpointa IN* jest realizowane przez funkcję *ReadAllReports()*. Jeżeli chcielibyśmy odczytywać pomiar „On Line" trzeba wywoływać tę funkcję cyklicznie.

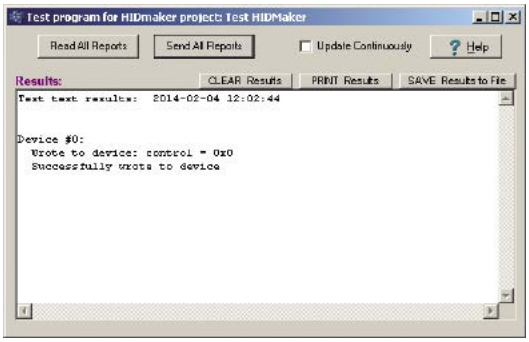
Dotarliśmy do momentu, w którym mamy możliwość przesyłania danych



Rysunek 11. Okno programu po podłączeniu modułu PIC18F4550



Rysunek 12. Wyświetlanie wartości zmiennej analog



Rysunek 13. Wysłanie control=0x00.

z układu mikroprocesorowego do komputera. Dalej już, zależnie od inwencji i możliwości programisty, można tworzyć bardziej lub mniej atrakcyjny interfejs wizualizacji tych danych. Zobaczmy teraz jak wygląda przesyłanie danych z komputera do mikrokontrolera. Na **listingu 6** pokazano funkcję wykonywaną po kliknięciu na przycisk *SendAllReports*. Pamiętamy, że w programie HIDMaker zdefiniowaliśmy zmienną *control* przeznaczoną do przesyłania w kierunku Host-urządzenie (*endpoint OUT*).

Wysyłanie danej jest realizowane przez funkcję *WriteAllReport()*. Do celów testowych najpierw wpiszemy *Test[DevNum]* → *control* → *UnScaledValue* = *0x00*. Po wysłaniu tej wartości i odebraniu przez mikrokontroler zostanie ona wpisana do rejestru portu PORTD (list. 2). Spowoduje to zgaszenie wszystkich diod LED. Po wpisaniu do *control* wartości 0x0f, wszystkie diody dołączone do linii RD0…RD3 zaświecą sie.

Jak widać, bardzo proste modyfikacje szkieletów obu programów umożliwiają dokładnie testowanie połączenia USB. Oczywiście, w aplikacji docelowej najprawdopodobniej większość komponentów szkieletu, tak jak komunikaty w oknie *memo*, mogą nie być potrzebne i będzie je można usunąć lub ukryć, jednak na początek tak pomyślany projekt wydaje się bardzo pomocny i przemyślany.

Podsumowanie

Z punktu widzenia programisty elektronika HIDmaker jest narzędziem właściwie idealnym. Daje bardzo duże możliwości przy minimum wysiłku. Dodatkową zaletą jest intuicyjny interfejs graficzny i bardzo dobre wsparcie w postaci plików pomocy i umieszczeniu w plikach źródłowych wielu przydatnych komentarzy. Przedstawiona tu wersja jest przeznaczona dla mikrokontrolerów PIC18F, ale testowałem również wersję dla mikrokontrolerów PIC32. I ta wersja również działa znakomicie.

Można sobie postawić pytanie: dla kogo jest ten program? Na pewno dla małych firm, które nie mogą sobie pozwolić na przeznaczenia wielu godzin pracy nad oprogramowaniem połączenia USB przez wykwalifikowanego programistę. Również amatorzy mogą korzystać z HIDmaker'a, tym bardziej, że jak wynika z powyższego opisu, jego używanie nie wymaga profesjonalnego przygotowania.

Są również pewne ograniczenia. Po pierwsze, HIDmaker jest przeznaczony tylko dla mikrokontrolerów firmy Microchip (rodziny PIC18, PIC24 i PIC32). Szeroka oferta układów Microchipa powoduje, że w zasadzie do każdej aplikacji można z niej coś dobrać. Jednak ktoś, kto ma w planach konstrukcję na przykład z mikrokontrolerem Cortex-M niewiele skorzysta z HIDmaker'a.

HIDmaker nie jest również programem bezpłatnym i jak polskie warunki – kosztuje niemało. Na stronie producenta *http://goo.gl/Op188u* podaje cenę ok 600 USD, czyli ok. 2000 złotych. To na pierwszy rzut oka sporo, ale jak się policzy, ile czasu można zaoszczędzić używając tego programu, to nie wydaje się to cena wygórowana, szczególnie dla tych, którzy muszą w swoich konstrukcjach stosować interfejs USB.

Tomasz Jabłoński, EP