Multimedia remote, czyli nie bój się USB!

Artykuł opisuje układ i zasady realizacji urządzeń USB typu HID, których współpraca z komputerem nie wymaga instalacji żadnych sterowników.

Do czego to służy?

Czy nie zamarzyło ci się, aby usiąść w fotelu, obejrzeć film lub posłuchać muzyki, a przy tym nie wstawać, by zmienić głośność lub utwór? Z pewnością tak – wystarczy do tego pilot. Jeżeli jednak w roli odtwarzacza wykorzystujemy komputer, jest pewien problem z pilotem. Oprócz pilota, potrzebny jest jakiś odbiornik - przystawka do komputera.

Na rynku jest cała masa firmowych pilotów oraz przystawek do pilotów od sprzętu RTV, ale trudno znaleźć przystawkę, która realizowałaby przynajmniej podstawowe funkcje, jak zmiana głośności czy utworu.

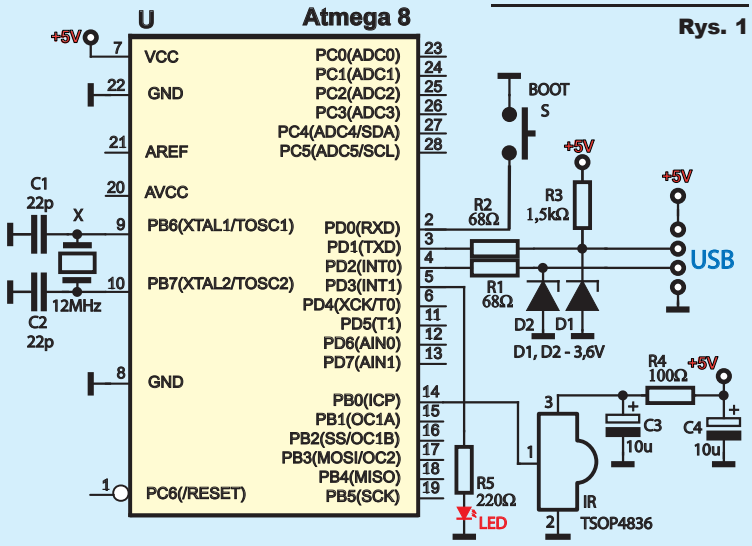
A może ją zrobić samemu? Na pierwszy rzut oka wygląda to na karkołomne przedsięwzięcie z uwagi na konieczność wykorzystała łącza USB, bowiem wtedy w komputerze potrzebne są odpowiednie sterowniki. Z pewnością zniechęca, licząca aż 650 stron, specyfikacja USB2.0, dostępna na *www.usb.org*. Jednak czy aby na pewno jest to niewykonalne dla zwykłego użytkownika lutownicy?

Nie taki diabeł straszny!

Po pierwsze, do wykonania odbiornika - przystawki USB można wykorzystać mikrokontroler AVR firmy ATMEL i bibliotekę V-USB:
www.obdev.at/products/vusb/index.html
A co ze sterownikami dla komputera?

Zapewne zauważyłeś, że przy podłączaniu pewnych urządzeń do portu USB, po chwili system informuje o prawidłowym ich funkcjonowaniu. Tak! Niektóre sterowniki są już zainstalowane w komputerze. W każdej wersji Windows (od wersji WIN98) oraz Linux czy Mac standardowo obsługiwane są urządzenia USB typu HID, czyli urządzenia obsługiwane przez człowieka: klawiatury, myszki, dżojstiki, itd.

Mamy więc pilot (pracujący w kodzie RC-5) i aby sterować komputerem, budujemy przystawkę USB z procesorem AVR. Nasza przystawka będzie się przedstawiać komputerowi jako urządzenie HID, jako klawiatura multimedialna, która przecież ma przyciski do sterowania multimediami. Odpadnie pisanie sterowników na dany komputer czy system operacyjny, a pisząc program dla procesora przystawki, wykorzystamy gotową bibliotekę V-USB oraz inne służące do dekodowania sygnałów pilota. Sprawa jest w zasadzie prosta, trzeba tylko napisać program w języku C. Język ten jest uznawany za trudny, jednak jeżeli ja, jako raczej początkujący w tym języku, podołałem zadaniu, to oznacza, że także wielu Czytelników sprosta temu wyzwaniu.



Rys. 1

Jak to działa?

Na **rysunku 1** widzimy schemat naszej przystawki do komputera. Sercem układu jest popularny mikrokontroler AVR ATmega8, taktowany rezonatorem X. Diody D1, D2 oraz rezystory R1, R2 dopasowują poziomy napięć do standardu portu USB, a R3 informuje kontroler, że ma do czynienia z urządzeniem *low speed*. C4 filtruje napięcie główne, natomiast R4 wraz z C3 napięcie zasilające odbiornik podczerwieni. Dioda LED informuje o poprawnym odebraniu kodu z pilota. Przycisk BOOT służy do uruchomienia bootloadera.

Kluczową sprawą jest napisanie programu dla procesora AVR, który zajmie się: komunikacją przez USB, odbiorem sygnałów w podczerwieni oraz symulowaniem urządzeń HID.

Aby cokolwiek wgrać do pamięci mikrokontrolera, należy napisać program i skompilować wsad. Można to zrobić w darmowym środowisku AVRStudio 4 oraz WinAVR. Skorzystałem odpowiednio z wersji AVRStudio 4.18 oraz WinAVR 20100110. Obie bezpłatnie i bezproblemowo można pobrać z Internetu. Instalacja jest typowa i ogranicza się głównie do klikania *next*. Aby nie było problemu, proponuję wybrać domyślne ścieżki zaproponowane przez instalatora. Po zainstalowaniu obu programów uruchamiamy plik *Multimedia_remote.aps* (powinien mieć ikonę z biedronką). Włączy się AVRStudio i najprawdopodobniej otworzy plik *main.c*. W tym momencie nie jest on do niczego potrzebny. Po lewej stronie w oknie AVRGCC jest folder *External Dependencies*. Rozwijamy go i wybieramy *keys.h*. W pliku tym przypisane są funkcje do poszczególnych przycisków pilota – od połowy w tablicy z klawiszami. Fragment znajduje się na **rysunku 2**, a całość w Elportalu. Widać, że nie ma tu nic skomplikowanego. W nawiasach klamrowych przecinkami oddzielonych jest pięć liczb lub ich zdefiniowane wcześniej wartości. Przy modyfikacjach należy pamiętać, aby nie zgubić przecinków, gdyż kompilator

zasygnalizuje problem. Dla ułatwienia, po lewej stronie są podane najczęściej wykorzystywane funkcje pilota RC5 do TV.

W górnej części pliku *keys.h* są definicje większości klawiszy. Gdyby nie było definicji któregoś z nich należy skorzystać z tabeli pobranej ze strony Microsoftu – *http://download.micro-soft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/translate* i dopisać. Przykładowo **KEY_J** to po prostu litera **J (**ale nie duża czy mała litera, bo to zależy od tego, czy naciśnięty jest klawisz SHIFT), **MOD_CONTROL_LEFT** to lewy klawisz **Control,** a **Volume_Up** to zwiększanie głośności. Analogicznie jest z pozostałymi definicjami klawiszy i ich rozszyfrowanie nie powinno stanowić problemu.

Gdy już zmodyfikujemy plik, najlepiej całość zapisać (CTRL+S) i skompilować, naciskając klawisz F7. Jeśli wszystko przebiegnie poprawnie, na dole w oknie otrzymamy komunikat **Build succeeded with 0 Warnings...**

Zmiany w pliku *keys.h* możemy także dokonać w dowolnym edytorze tekstowym np. Notepad++, pamiętając, aby przed kompilacją zapisać w nim zmiany.

Skompilowany plik *Multimedia_remote.hex,* znajdujący się w folderze *default*, wgrywamy za pomocą botloadera. Jeśli chcemy nanieść jakieś zmiany, to ponownie modyfikujemy plik *keys.h*, kompilujemy i po raz kolejny wgrywamy do mikrokontrolera.

Na początku zapewne wielokrotnie będziemy wprowadzać zmiany, gdyż uzyskanie prawidłowej konfiguracji może zająć trochę czasu, jednak późniejsze przyzwyczajenie się do pilota nie będzie tego wymagało i ostateczna konfiguracja pozostanie na długi czas.

Obłożenie klawiszy dostosowałem do swoich upodobań tak, jak się już wcześniej przyzwyczaiłem, korzystając z przystawki AVT3015, dzięki czemu po wgraniu skompilowanej przeze mnie wersji, bez modyfikacji, po naciśnięciu przycisków pilota, „coś zadziała" :)

Przystawka była testowana na systemach Win 98SE, Xp, Win7 oraz Ubuntu i na każdym z nich nie ma potrzeby instalowania sterowników USB.

W programie domyślnie jest wybrany pilot do TV, jednak gdyby ktoś miał pilot RC5 z innym adresem niż zero, np. od magnetowidu, to należy dokonać zmiany w pliku *main.c*. Nie powinno to stanowić problemu, gdyż zaznaczone to zostało odpowiednim komentarzem. Oprócz wspomnianej zmiany pilota RC5 od innego sprzętu niż TV, można też po zmianie biblioteki skorzystać z pilota nadającego w innym standardzie. Bardzo popularny jest standard Sony SIRC oraz inne. Również wykorzystanie pilotów radiowych nie powinno być trudne. Jeśli ktoś chciałby wykorzystać sterownik w komputerze typu Home Theater Personal Computer, może całości nie montować w obudowie, tylko wstawić do jego wnętrza oraz podłączyć się do portu USB typu goldpin. Inne bardziej ambitne możliwości to dodanie dodatkowych urządzeń HID, jednak to tylko dla doświadczonych. A teraz wróćmy do HID.

```
/*mouse*/
    0x05, 0x01, // USAGE_PAGE (Generic Desktop)
    0x09, 0x02, // USAGE (Mouse)
    0xa1, 0x01, // COLLECTION (Application)
    0x09, 0x01, //   USAGE (Pointer)
    0xA1, 0x00, //   COLLECTION (Physical)
    0x85, 0x01, //     REPORT_ID (1)
    0x05, 0x09, //     USAGE_PAGE (Button)
    0x19, 0x01, //     USAGE_MINIMUM 1
    0x29, 0x03, //     USAGE_MAXIMUM 3
    0x15, 0x00, //     LOGICAL_MINIMUM (0)
    0x25, 0x01, //     LOGICAL_MAXIMUM (1)
    0x95, 0x03, //     REPORT_COUNT (3)
    0x75, 0x01, //     REPORT_SIZE (1)
    0x81, 0x02, //     INPUT (Data,Var,Abs)
    0x95, 0x01, //     REPORT_COUNT (1)
```

Rys. 3

HID

Cała informacja, co potrafi dane urządzenie, zawarta jest w deskryptorze HID. Początkującemu łatwiej jest zmodyfikować już istniejący plik, niż tworzyć od zera. Ułatwia to program HID Descriptor Tool dostępny na *http://www.usb.org/developers/hidpage/*. Na podanej stronie znajduje się również specyfikacja HID, dostępna w formie pliku PDF.

Aby trochę przybliżyć, jak to wygląda, proszę spojrzeć na deskryptor HID. **Rysunek 3** pokazuje fragment – całość dostępna jest w Elportalu wśród materiałów dodatkowych do tego numeru EdW. Cały deskryptor zajmuje 159 bajtów i definiuje 4 rodzaje urządzeń – mysz i trzy rodzaje klawiatur. Deskryptor myszki został żywcem sczytany z oryginalnego urządzenia, dzięki czemu miałem mniej kombinowania i jednocześnie pewność, że będzie działał. Definiuje on trzy standardowe przyciski, osie x, y oraz kółko. Pozostałe trzy urządzenia to klawiatury. Pierwsza z nich należy do grupy *consumer devices*, czyli popularnych przycisków multimedialnych, takich jak Play, Stop, itd. Druga to typowa klawiatura 101 przycisków, a trzecia należy do grupy *system controls.* Odpowiada ona za wyłączanie czy uśpienie komputera – przyciski Power i Sleep.

Wszystkie informacje wysyłane do komputera są różnej długości, zgodnie ze zdefiniowaną formą w deskryptorze HID. Strukturę widać na **rysunku 4**.

Uwaga 1. Oprócz właściwych danych, trzeba również wysyłać tzw. *Repord ID*, który informuje komputer, z którym urządzeniem wymienia informacje. Ta liczba może być dowolna (niekoniecznie numerowana po kolei – to zawsze jest pierwszy bajt).

Pozostałe są zależne od wcześniej zdefiniowanych w deskryptorze i w dwóch przypadkach wykorzystana jest tylko część bitów. Przy zwykłej klawiaturze jest dodatkowy bajt informujący o wciśniętych klawiszach pomocniczych (Ctrl, logo Windows itd.), a przy myszce w sumie pięć bajtów – ID, przyciski i trzy osie.

Uwaga 2. Kody klawiatury przy USB są inne niż przy PS/2 i są zgodne z tabelą, do której link podałem wcześniej.

Aby lepiej zrozumieć działanie, podam przykład, jak wyłączyć komputer. W pierwszym bajcie trzeba wpisać 4 – ID *system control* (choć lepiej wpisać zdefiniowaną stałą **System)**, w drugim wybrać 1 (choć prawidłowo należy ustawić najmłodszy bit – wpisać **Power**), a w pozostałych… Ta klawiatura wykorzystuje tylko 2 bajty, a w tablicy z pliku *keys.h* jest ich aż pięć. Zastosowałem tu uproszczenie, przyjmując, że każdy element tablicy ma pięć bajtów, choć w większości przypadków dwa lub trzy są niewykorzystane – zastąpione zerami. Właściwie nie

```
/* command  0 - cyfra 0 */      {0, 0, 0, 0, 0},
/* command  1 - cyfra 1 */      {Consumer, Play_Pause, 0, 0, 0}, // play/paue
/* command  2 - cyfra 2 */      {Consumer, Stop, 0, 0, 0}, // stop
/* command  3 - cyfra 3 */      {0, 0, 0, 0, 0},
/* command  4 - cyfra 4 */      {Consumer, Scan_Previous_Track, 0, 0, 0}, // prev track
/* command  5 - cyfra 5 */      {Consumer, Scan_Next_Track, 0, 0, 0}, // next track
/* command  6 - cyfra 6 */      {Keyboard, MOD_CONTROL_LEFT | MOD_ALT_LEFT, UP_ARROW, 0, 0}, // vol up winamp
/* command  7 - cyfra 7 */      {Keyboard, MOD_CONTROL_LEFT | MOD_ALT_LEFT, LEFT_ARROW, 0, 0}, // rewind winamp
/* command  8 - cyfra 8 */      {Keyboard, MOD_CONTROL_LEFT | MOD_ALT_LEFT, RIGHT_ARROW, 0, 0}, // forward winamp
/* command  9 - cyfra 9 */      {Keyboard, MOD_CONTROL_LEFT | MOD_ALT_LEFT, DOWN_ARROW, 0, 0}, // vol down winamp
/* command 10 - -/--            */      {0, 0, 0, 0, 0},
/* command 11 - P/C             */      {0, 0, 0, 0, 0},
/* command 12 - standby */      {System, Power, 0, 0, 0}, // PC off
/* command 13 - mute            */      {Consumer, Mute, 0, 0, 0}, // mute
/* command 14 - normalizacja    */      {0, 0, 0, 0, 0},
/* command 15 - info            */      {0, 0, 0, 0, 0},
```

Rys. 2

HID descriptor dla 4 urządzeń

Mouse							
Report ID 1							
Pusty	Pusty	Pusty	Pusty	Pusty	Przycisk 3	Przycisk 2	Przycisk 1
Os X							
Oś Y							
Kółko							

Consumer Devices							
Report ID 2							
Pusty	Głośniej	Ciszej	Wycisz	Play/Pause	Stop	Następny utwór	Poprzedni utwór

Klawiatura							
Report ID 3							
Kod klawisza							
Lewy CTRL	Lewy SHIFT	Lewy ALT	Lewy WIN	Prawy CTRL	Prawy SHIFT	Prawy ALT	Prawy WIN

System Control							
Report ID 4							
Pusty	Pusty	Pusty	Pusty	Pusty	Wake Up	Sleep	Power Down

Rys. 4

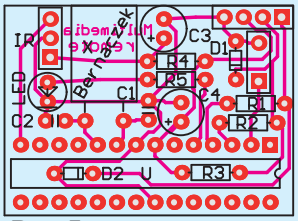
ma znaczenia, co będzie wpisane w „pustych" bajtach. Trzeba tylko pamiętać, że każdy element tablicy musi mieć pięć bajtów, inaczej kompilator zasygnalizuje nam problem. Analogicznie jest z innymi klawiszami.

Wyjaśnienia mogą wymagać zdefiniowane wartości przy osiach myszki. Podczas wolnego przesuwania, kursor płynnie przesuwa się po ekranie, jednak przy gwałtownym ruchu widać jego skoki. Jest to spowodowane przeliczaniem chwilowego przyśpieszenia na położenie kursora na ekranie. Typowa mysz ma częstotliwość odświeżania na poziomie 100Hz dla PS/2 i 125Hz dla USB. Przy transmisji RC5 jeden kod wysyła się stosunkowo szybko – kilkanaście ms – ale po nim jest długa przerwa – ponad 100ms – co oznacza, że przy ciągłym trzymaniu przycisku na pilocie, w ciągu sekundy wysyłanych jest mniej niż dziesięć kodów. Jeśliby zastosować płynne przesuwanie kursora po jednym pikselu, to przy rozdzielczości 1024x768, w poziomie od jednego do drugiego boku, przesuwanie trwałoby grubo ponad minutę, a to zdecydowanie za długo. Przyjąłem, że skok co osiem pikseli jest wystarczający, aby wybrać kursorem dowolny element i jednocześnie by przesuw był w miarę szybki. Dla stałej **Mouse_right** wynoszącej 8 wszystko jest zrozumiałe, ale dla **Mouse_left** wynoszącej 248 już może budzić wątpliwości. Przyjąłem, że brak ruchu to zero, a więc ruch w prawo to +8, a ruch w lewo to –8 (256 – 8). Analogicznie jest z ruchem w pionie. Kółko przyjmuje wartości 1 i –1 (1 i 255) gdyż okazały się wystarczająco szybkie. W ustawieniach myszki w systemie często jedno przesunięcie kółka powoduje ruch strony aż o 3 pozycje, np. linie tekstu.

Trochę czasu straciłem na poznanie, dlaczego po wybraniu jakiejś kombinacji symulującej naciśnięcie klawisza na klawiaturze jest powtarzana, mimo że nie trzymamy wciśniętego przycisku na pilocie. Co ciekawe, wybranie innego przycisku spowodowało powtarzanie innej funkcji. Działo się tak, gdyż komputer rejestruje tylko naciśnięcie oraz zwolnienie klawisza na klawiaturze, a samym powtarzaniem zajmuje się system (jego prędkość oraz opóźnienie początkowe możemy ustawić). Aby rozwiązać problem, po każdym wysłaniu kodu jest wysyłane zero, aby komputer sądził, że nie naciskamy już klawisza.

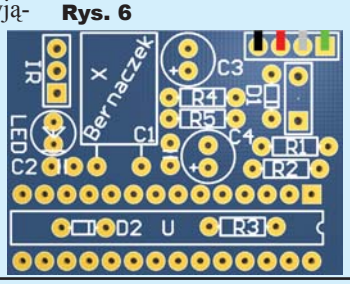
Montaż i uruchomienie

Montaż można wykonać na płytce z **rysunku 5**, która została przystosowana do transparentnej obudowy KM-1. Montaż jest typowy, zaczynamy od najmniejszych elementów, a kończymy na podstawce pod mikrokontroler. Należy uważać przy stosowaniu podstawki, aby umożliwiała montaż diody i rezystora znajdującego się pod nią. Jako odbiornik podczerwieni można zastosować dowolny o identycznym rozkładzie wyprowadzeń na 36kHz. Montaż przycisku jest trochę nietypowy, gdyż w uswitchu zostały wyprostowane nóżki, aby był kątowy. Pozostałe dwa zostały odcięte. W obudowie należy wykonać 2 otwory: na kabel USB i przycisk. Ostatnie lutowanie to kabel USB: kolejność przewodów jest nietypowa i zgodna z **rysunkiem 6**. Przed wlutowaniem warto sprawdzić, czy na przewodzie czerwonym faktycznie jest +5V, a na czarnym masa, gdyż zdarzył mi się taki kabel, co miał przewody podłączone na odwrót.



Rys. 5

Rys. 6



Warto zabezpieczyć kabel przed wyrwaniem wewnątrz obudowy np. przez zaciśnięcie na nim opaski elektrycznej. Nie polecam zalewania klejem na gorąco, gdyż ewentualne serwisowanie będzie bardzo trudne.

Następnie należy zaprogramować mikrokontroler plikiem bootloadera USB *main.hex* – dostępnym w Elportalu oraz ustawić fusebity zgodnie z **rysunkiem 7**. Po tym pozostaje tylko wgranie za pomocą bootloadera wygenerowanego pliku, z wcześniej ustaloną konfiguracją.

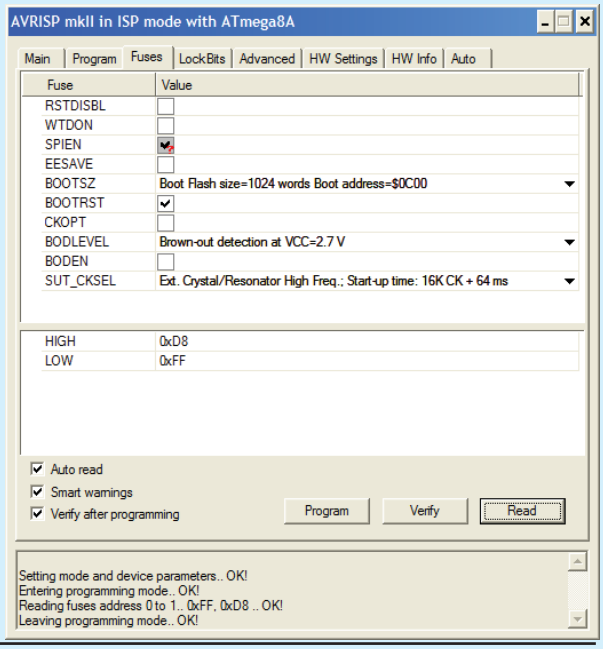
Bootloader został wykorzystany, aby ułatwić konfigurację urządzenia, gdyż wszelkie zmiany należy wprowadzić na etapie kompilowania programu, a każdorazowe używanie programatora byłoby bardzo uciążliwe.

BootloadHID

Do wgrania skompilowanego programu (bootloader) można wykorzystać łącze USB. Skorzystałem z rozwiązania dostępnego na stronie *www.obdev.at/products/vusb/bootloadhid.html.*

Bootloader nie wymaga żadnych sterowników. Aby wgrać skompilowany wsad, trzeba skorzystać z programu HIDBootFlash.exe – jest na podanej wyżej stronie, jak i w materiałach do tego numeru EdW. Należy ww. program uruchomić (nie wymaga instalowania), przytrzymać

Rys. 7



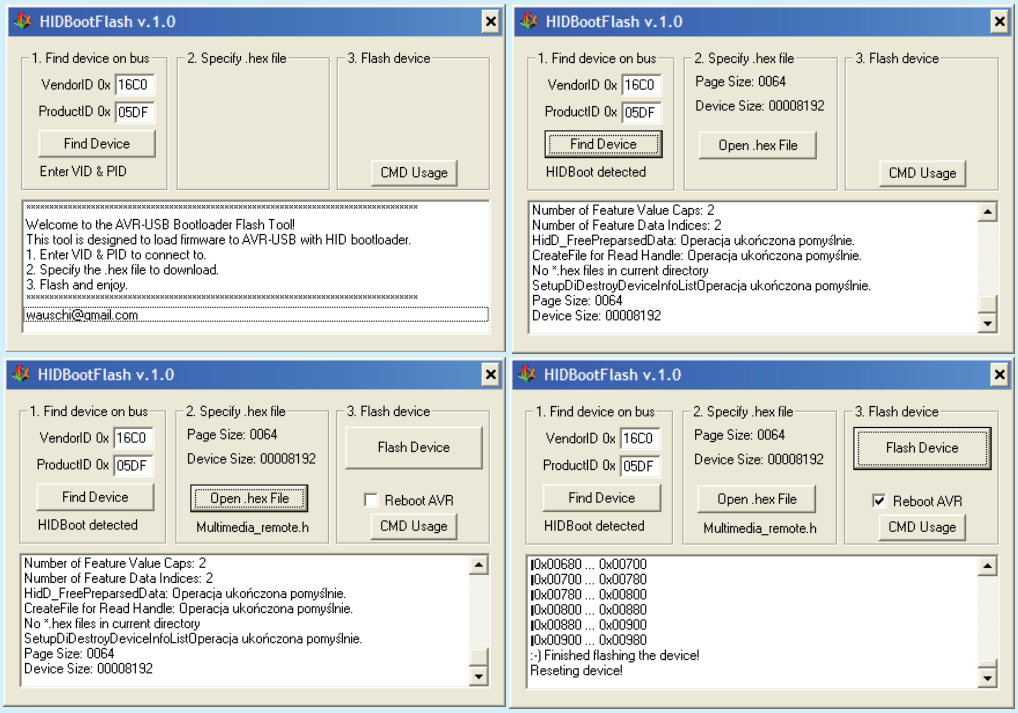
Wykaz elementów

R1, R2	68Ω
R3	1,5kΩ
R4	100Ω
R5	220Ω
C1, C2	22pF
C3, C4	10uF
D1, D2	dioda Zenera 3,6V
S	uswitch
IR	TSOP4836
LED	LED 3mm
X	12MHz
U	ATmega8

Podstawka wąska 28 pin
Kabel USB A-B

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-3057.

przycisk BOOT (na czas programowania) i podłączyć kabel USB. Po poprawnym zainstalowaniu urządzenia, w programie klikamy **Find device, Open .hex File** i wybieramy plik hex, który chcemy wgrać do mikrokontrolera (jaki to plik, napisałem wcześniej). Po wyborze wsadu pokaże się przycisk **Flash device**, który klikamy. Trzeba również zaznaczać **Reebot AVR**, co spowoduje automatyczne uruchomienie wgranego wsadu. Po zaprogramowaniu, które trwa sekundy, puszczamy przycisk.



Rys. 8

Całość widać na **rysunku 8**. Potem już można korzystać z pilota. Jeśli zajdzie potrzeba wgrania zmienionego wsadu, postępujemy analogicznie.

Aleksander Bernaczek
olo11111@gmail.com