

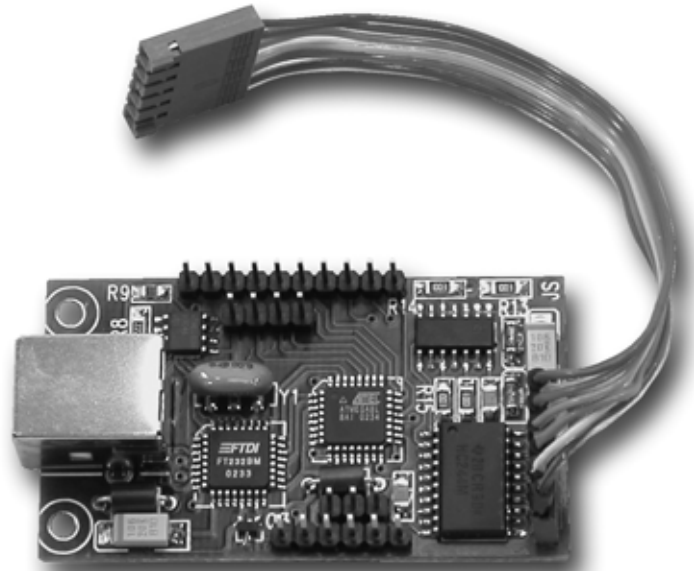
Programator USB mikrokontrolerów ATmega - ISP, część 1



AVT-524

Programatory ISP dla mikrokontrolerów AVR to temat od dawna znany i obfitujący w rozmaite rozwiązania - zarówno komercyjne jak i amatorskie (jedno z możliwych rozwiązań przedstawiamy w Miniprojektach). Pojawia się więc pytanie, czy warto od podstaw opracowywać coś nowego? Na pewno nie ma sensu powielanie typowego projektu opartego np. na porcie równoległym LPT. Co w zamian? Oczywiście USB!

Rekomendacje: programator dla wymagających konstruktorów, przede wszystkim tych, którzy korzystają z mikrokontrolerów ATmega.



Podczas konstruowania programatora prezentowanego w artykule nie koncentrowałem się na zapewnieniu jego maksymalnej prostoty i obniżeniu ceny, ważniejsze były jego cechy użytkowe. Dlatego też:

- komunikacja z komputerem sterującym jest zrealizowana za pomocą łącza USB, co często jest jedynym dostępnym sposobem dla posiadaczy laptopów,
- własne oprogramowanie sterujące pozwala na dowolne modyfikacje, poprawki i rozszerzenia a także na stosunkowo łatwą integrację z różnymi środowiskami,
- płytkę programatora posiada oprócz buforowanego interfejsu ISP wyprowadzone także inne interfejsy (I2C, SPI) oraz linie przetwornika A/C, PWM itd. co pozwala na jej wykorzystanie na rozmaite sposoby,

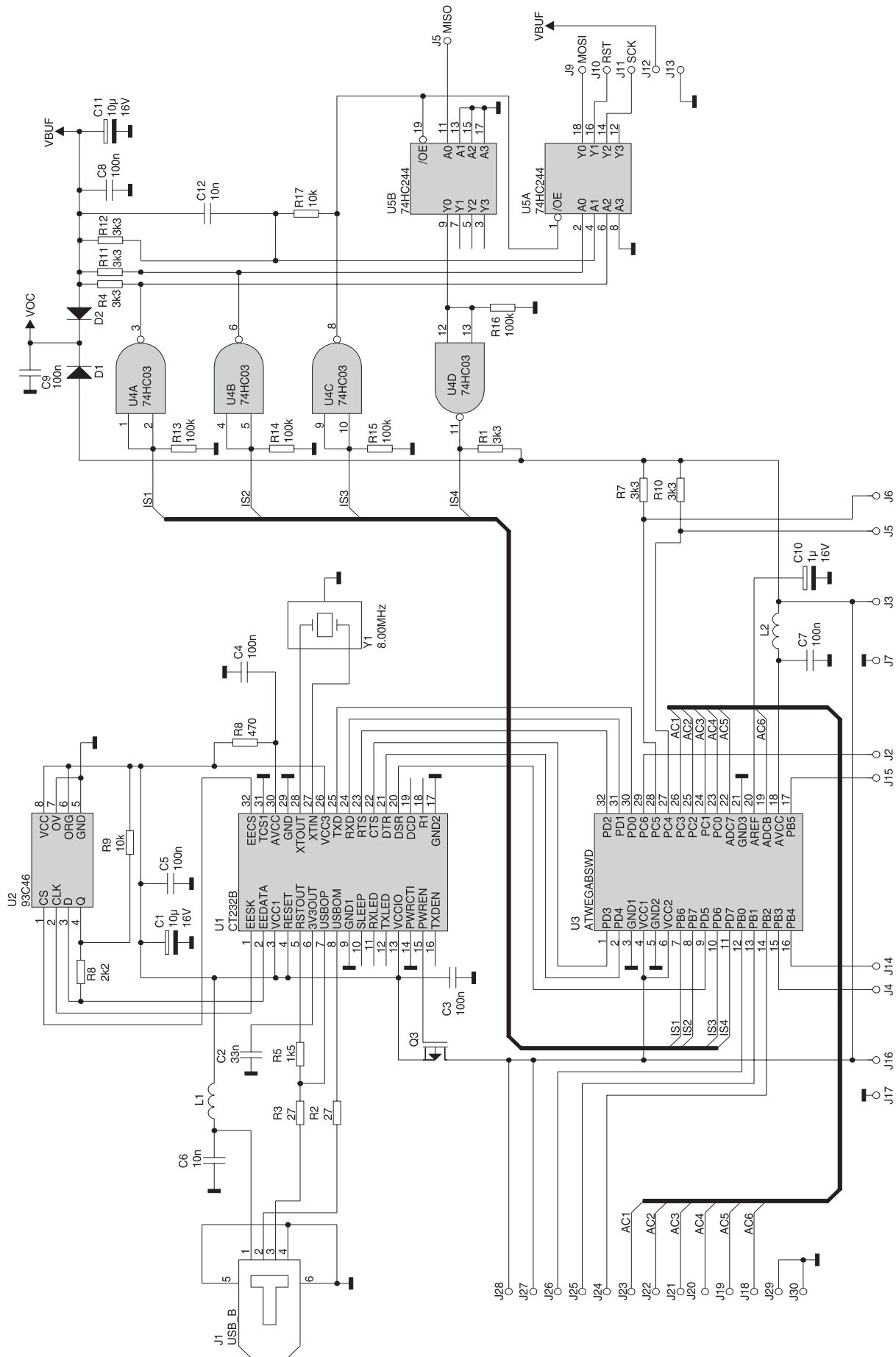
- mikrokontroler programatora można wyposażyć w bootloader, co pozwoli na szybkie i sprawne wymiany wersji oprogramowania za pośrednictwem tego samego łącza USB.

Konstrukcja programatora

Schemat elektryczny urządzenia przedstawiono na **rys. 1**. Można na nim wyróżnić trzy podstawowe bloki funkcyjne: interfejs USB, mikrokontroler sterujący oraz buforowany interfejs ISP.

Połączenie z magistralą USB jest zrealizowane z pomocą układu U1 - FT8U232BM - w typowej aplikacji proponowanej przez FTDI. Pamięć U2 pozwala na wprowadzenie własnych identyfikatorów programatora. Tranzystor Q3 zapewnia - zgodnie z wymaganym standardem USB - wyłączanie zasilania pro-

Programator prezentowany w artykule współpracuje ze środowiskiem projektowym dla AVR-GCC - AVRSide (dostępne bezpłatnie na stronie <http://www.avrside.fr.pl> oraz na CD-EP8/2003B). Możliwości jego wczesnej wersji opisaliśmy w EP1/2003).



Rys. 1. Schemat elektryczny programatora

gramatora podczas enumeracji oraz w stanie uśpienia komputera - hosta.

Mikrokontroler U3 steruje interfejsem ISP zgodnie z komendami i danymi otrzymywanymi z działającego na komputerze - hosta oprogramowania. Zwraca też informacje o prawidłowym zakończeniu kolejnych operacji lub o zaistniałych błędach. Dzięki zastosowaniu mikrokontrolera ATmega 8 nie ma konieczności dodawania praktycznie żadnych dodatkowych peryferiów. Mikrokontroler pracuje z wewnętrznym oscylatorem 8 MHz i ma wbudowany układ zerujący oraz watchdog. Jednocześnie dysponuje całą gamą dodatkowych sprzętowych interfejsów: I2C, SPI, wielokanałowym przetwornikiem analogowo-cyfrowym, wyjściami PWM. Odpowiednie styki wyprowadzono na płytce, co pozwoli na ewentualne wykorzystanie programatora do różnych innych celów (konwertery USB<->I2C, USB<->SPI, programatory pamięci szeregowych, przetwornik A/C do komputera itd.). Obwód filtrujący L2, C7 zasilają obwody A/C, natomiast kondensator C10 dodatkowo filtruje wewnętrzne napięcie odniesienia 2,56 V, które zostało przewidziane do wykorzystania w układzie.

Wyprowadzenia interfejsu SPI tworzą zarazem wejście magistrali ISP pozwalającej na szeregowe wpisanie do mikrokontrolera programu wykonawczego oraz ustawień konfiguracyjnych.

Interfejs ISP łączy linie I/O mikrokontrolera sterującego z wejściami ISP docelowego układu. Musi on spełnić następujące wymagania:

- całkowicie niezależnie zespół urządzeń od kolejności włączania zasilania,
- umożliwić dołączanie układu docelowego zasilanego innym napięciem,
- uaktywniać linie sterujące ISP tylko i wyłącznie w trakcie programowania - normalnie powinny pozostawać w stanie wysokiej impedancji nie wpływając w żaden sposób na działanie docelowego układu bez konieczności odłączania.

Zadania te zostały zrealizowane przy pomocy inwertera z wyj-

ściami OC (U4) oraz trójstanowego bufora HC244 (U5). Bufor U5 oraz *pull-up* y jego linii wejściowych są zasilane napięciem docelowego systemu - zawsze więc uzyskamy potrzebną zgodność poziomów logicznych. Inwerter U4 jest zasilany dwustronnie poprzez diody Schottky ego D1 i D2. Kostka pozostaje więc „pod napięciem“ niezależnie od chwilowego podłączenia zasilania - eliminuje to możliwość wysterowania wejść przy braku zasilania U4 (co układy cyfrowe niezbyt lubią). Wyjścia typu otwarty kolektor pozwalają na odpowiednie dopasowanie poziomów logicznych (dla poziomu niskiego 0 V, a dla poziomu wysokiego Vcc - niezależnie od jego wartości - oczywiście w ramach możliwości układu, czyli do 6 V). Nie ma to większego znaczenia przy zasilaniu obu układów jednakowym napięciem (czyli +5 V - tyle programator otrzymuje z magistrali USB). Natomiast przy np. 3,3 V systemu docelowego mamy następującą sytuację:

- zasilanie U5 = 3,3 V,
- wysokie poziomy logiczne na wejściach i wyjściach U5 = ok. 3,3 V,
- zasilanie U4 = ok. 4,7...4,8 V (ok. +5 V z USB pomniejszone o spadek na diodzie Schottky ego),
- wysokie poziomy logiczne na wejściach U4 od strony programatora - ok. 5 V (co jest dopuszczalne - nie przekraczamy napięcia zadziałania wejściowych wewnętrznych diod ochronnych kostki),
- wysoki poziom logiczny na wejściu U4 od strony bufora (wyprowadzenia 12, 13) = ok. 3,3 V. Jest to jedyna linia wymagająca dokładniejszego sprawdzenia: z danych serii HC wynika, że przy takiej wartości napięcia zasilającego minimalne napięcie rozpoznawane jako poziom wysoki wynosi nieco mniej niż 3,3 V, mieścimy się więc w wymaganym zakresie.

Wynika z tego, że układy z zasilaniem poniżej 3,3 V mogą sprawiać problemy. Jak nisko można zejść - pozostaje do praktycznego sprawdzenia (zazwyczaj rzeczywiste właściwości układów są lepsze niż gwarantowane wartości podawane w katalogach).

Zwróćmy też uwagę na sposób włączania linii RST interfejsu. Sekwencja przebiega następująco:

- ustawiamy programowo niski poziom SCK - bufor jest jeszcze w stanie wysokiej impedancji i na wyjściu nic się nie dzieje,
- ustawiamy programowo niski poziom RST - bufor 244 zostaje włączony podając na wyjście SCK poziom niski, ale linia RST zostaje przestawiona z poziomu wysokiego na niski dopiero po krótkiej chwili wynikającej ze stałej czasowej obwodu R17, C12. W ten sposób gwarantujemy stabilny stan niski SCK w chwili zakończenia zerowania, zgodnie z zaleceniami Atmela dotyczącymi wchodzenia w tryb programowania szeregowego.

Montaż układu

Całość układu została zmontowana na dwuwarstwowej płytce drukowanej, której schemat montażowy pokazano na **rys. 2**. Użyto głównie elementów SMD, co pozwoliło na zachowanie niewielkich wymiarów. Wyprowadzenia interfejsów możemy wyposażać w listwy *goldpin* albo pozostawić do podłączeń przewodami - w zależności od potrzeb. Do polutowania najlepiej użyć grota *mini wave*, ale ponieważ rastry kostek nie należą do najmniejszych - tradycyjne techniki też będą zupełnie wystarczające.

Po zlutowaniu sprawdzmy dokładnie, czy nie ma zwarc ani przerw. Wskazane jest też przemyć płytki jednym z dostępnych preparatów chemicznych (np. PCC). Podłączenie do ISP prototypowego docelowego układu wykonamy według własnych potrzeb niezbyt długim (najwyżej kilkanaście centymetrów) przewodem taśmowym.

Uruchamianie programatora

Sam sprzętowy układ bez skonfigurowania i zaprogramowania nie przyda się nam do niczego. „Ożywianie“ płytki przebiega w kilku etapach. Wszystkie przedstawione opisy i programy dotyczą systemu Windows. Adaptację układu do innych systemów operacyjnych pozostawiam inwencji Czytelników.

Niezbędne programy (łącznie ze środowiskiem AVRSide) oraz pliki pomocnicze publikujemy na płycie CD-EP8/2003B.

Etap 1 - konfiguracja układu interfejsu USB

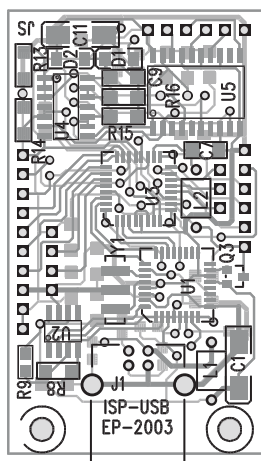
Najpierw musimy wyposażyć się w narzędzia ze strony producenta (www.ftdichip.com) - będzie nam potrzebny sterownik D2xx (najnowsza wersja z obsługą układów w wersji BM) oraz program *Ftd2xxst.exe* do obsługi szeregowej pamięci EEPROM 93C46. Rozpakowane pliki umieszczamy na dysku i zapamiętujemy lokalizację.

Instalacja sterowników

Teraz możemy po raz pierwszy podłączyć nasz układ do magistrali USB (bepośrednio do gniazda A komputera albo poprzez huba z własnym zasilaniem, nie ograniczającego poboru prądu do 100 mA).

Po włączeniu Windows wyświetli informację o obecności nowego urządzenia USB i poprosi o podanie lokalizacji sterownika - odnajdujemy i zatwierdzamy lokalizację folderu z plikami (jak wyżej) - po chwili instalacja jest zakończona. Menedżer urządzeń w kluczu *Kontrolery uniwersalnej magistrali szeregowej* pokaże teraz obecność urządzenia „FTDI FT8U2XX DEVICE“.

Uwaga! Sprawa może się poważnie skomplikować jeśli używamy już jakiegoś układu z kostką FTDI obsługiwaną jako wirtualny port szeregowy. System ma



Rys. 2. Rozmieszczenie elementów na płycie drukowanej

wtedy dla firmowych VID oraz PID zainstalowany inny zestaw sterowników (VCP), co wyklucza jednocześnie użycie D2xx. Jedyne wyjście będzie wtedy skonfigurowanie kostki FT8U232 na innym komputerze i zmiana np. PID. Taki sam zmieniony PID musimy ręcznie wpisać do pliku **.inf* sterownika D2xx, aby system umiał przypisać płytce z nowym PID właśnie ten sterownik. Operacje ze zmianą PID (VID) muszą być przeprowadzone z należytą starannością - jeśli się pomylimy system w ogóle nie będzie w stanie obsłużyć płytki (zgłaszając tylko obecność nieznanego urządzenia).

Wpisy do deskryptorów FT8U232

Służy do tego wspomniany powyżej program narzędziowy *Ftd2xxst.exe*. Sposób jego użycia jest dokładnie opisany w podręczniku, który w postaci elektronicznej publikujemy na CD-EP8/2003B.

Po wybraniu nowego pliku edytujemy wszystkie potrzebne pola:

Manufacturer = FTDI
Manufacturer ID = FT
Vendor ID = 0403

Product ID = 6001 (chyba, że wymagana jest zmiana zgodnie z uwagą powyżej)

Description = avr isp loader (tej pozycji używa program nadrzędny PC do lokalizacji urządzenia, nie może być więc ona zmieniana, chyba że równoległe z kodem programu).

W opcjach zaawansowanych ustawiamy:

Plug and Play - nie

Fixed Serial Number - dowolnie, w prototypach ustawiłem na „tak“, wprowadzając ręcznie numery 20000001, 20000002 itd.)

Self Powered - nie

Remote Wakeup - nie

Max Power (mA) - 250 mA (wprawdzie sam programator mieści się w podstawowych 100 mA, ale być może będziemy stosować płytkę w innym celu i przyda się możliwość zasilania dodatkowych zewnętrznych podzespołów).

WYKAZ ELEMENTÓW

Rezystory

R1, R4, R7, R10...R12: 3,3kΩ 1206
R2, R3: 27Ω 0805
R5: 1,5kΩ 0805
R6: 470Ω 1206
R8: 2,2kΩ 1206
R9: 10kΩ 0805
R13...16: 100kΩ 1206

Kondensatory

C1, C11: 10μF/16V 6032 tantal
C2: 33nF 0805
C3...C5, C13: 100nF 1206
C6, C12: 10nF 0805
C7...C9: 100nF 1206
C10: 1μF/16V 3216 tantal

Półprzewodniki

U1: FT8U232BM TQFP32
U2: 93C46 SO8
U3: Atmega 8 TQFP32
U4: 74HC03 SO14
U5: 74HC244 SO20
Q3: tranzystor P-MOS MMBF 2202
PT1 SOT23
D1, D2: dioda uniwersalna Schottky Minimelf

Różne

Y1: rezonator ceramiczny 6,00 MHz CSTCC6.00MG - TC (Murata) SMD (można też powierzchniowo przylutować rezonator przewlekany)
L1, L2: koralik ferrytowy przewlekany
J1: gniazdo USB typ B listwy *goldpin*, przewód taśmowy

Po zakończeniu edycji zapamiętujemy plik i programujemy EEPROM. Nowe ustawienia deskryptorów będą użyte po następnej enumeracji (czyli odłączeniu i ponownym podłączeniu płytki).

Etap 2 - konfiguracja mikrokontrolera ATmega 8

Używamy do tego dowolnego posiadanego programatora ISP. Prototypy były konfigurowane z poziomu AVRSide za pośrednictwem optoizolowanego programatora RS. Zwróćmy tylko uwagę na prawidłowe podłączenie linii - za pomocą schematu i rysunku płytki zlokalizujemy bez trudu wyprowadzenia interfejsu ISP. Mikrokontroler zastosowany w projekcie pracuje z wewnętrznym oscylatorem 8 MHz. Prze-

stawiamy więc fabryczne 0001 (1 MHz) na 0100. Pozostałych bitów możemy w naszym zastosowaniu nie zmieniać. Jednocześnie należy odczytać i zanotować wartość bajtu kalibracyjnego dla 8 MHz. ATmega po zerowaniu zawsze automatycznie ładuje do korekcyjnego rejestru OSCCAL wartość dla 1 MHz. Musimy więc w części inicjalizacyjnej programu przeładować samodzielnie OSCCAL odczytaną powyżej wartością. Wymaga to wprawdzie przekompilowania programu oddzielnie dla każdego egzemplarza urządzenia, co jednak na etapie testów i uruchamiania (czy też przy składaniu pojedynczych płytek na własne potrzeby) nie jest żadnym utrudnieniem.

Uwaga! Konfiguracja *fuse'ów* wymaga uwagi i staranności. Sprawdźmy dwa razy zanim coś ostatecznie wpisujemy do mikrokontrolera. Niektóre bity mogą nam bowiem całkiem zablokować interfejs ISP, ale nawet zmiana źródła taktowania (inne niż potrzebne ustawienie CSEL) sprawi w zmontowanym już urządzeniu mnóstwo kłopotów.

Etap 3 - zaprogramowanie mikrokontrolera

Jak zaznaczono wyżej, należy wpisać do programu właściwą dla danego egzemplarza ATmega wartość OSCCAL (na początku funkcji *main()* w module *u_main.c*). Kod dla programatora jest wol-

nym oprogramowaniem, całkowicie dostępnym - można więc przy okazji go poprawiać czy rozszerzać o nowe funkcje (na razie uruchomione są tylko podstawowe operacje dla serii ATmega). Otrzymany po kompilacji plik wynikowy *avrisp.hex* ładujemy do mikrokontrolera dowolnym programatorem ISP. Teraz możemy przystąpić do praktycznych prób ale o tym za miesiąc.

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/sierpien03.htm> oraz na płycie CD-EP8/2003B w katalogu PCB.

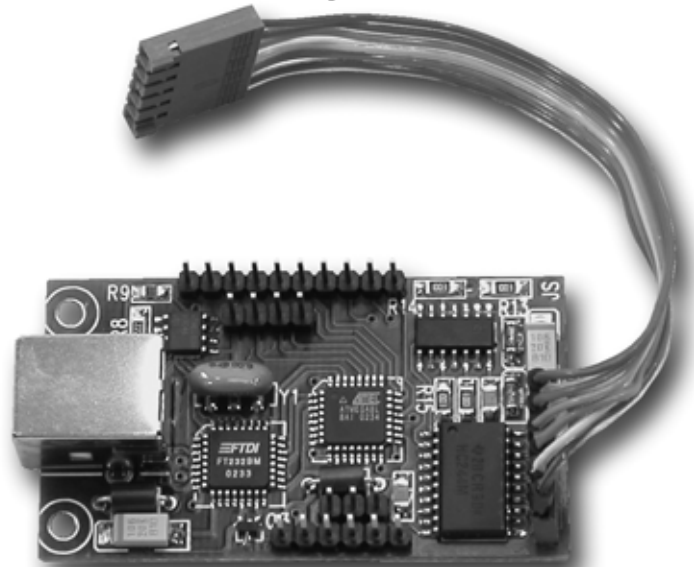
Programator USB mikrokontrolerów ATmega - ISP, część 2

AVT-524

Programatory ISP dla mikrokontrolerów AVR - zarówno komercyjne jak i amatorskie (jedno z możliwych rozwiązań układowych przedstawiamy w Miniprojektach) są od dawna opisywane. Nasuwa się więc pytanie, czy warto od podstaw opracowywać jakiś nowy układ programatora? Na pewno nie ma sensu powielanie typowego projektu opartego np. na porcie równoległym LPT. Co w zamian?

Oczywiście USB!

Rekomendacje: programator dla wymagających konstruktorów, pragnących zgłębić tajniki USB - przede wszystkim tych, którzy korzystają z mikrokontrolerów ATmega.



Program mikrokontrolera sterującego

Program dla mikrokontrolera ATmega został napisany w języku C, z użyciem najnowszej wersji WinAVR oraz śródowiska AVRSide. Kod źródłowy jest załączony do materiałów pomocniczych, jest też (najbardziej aktualna wersja) dostępny na stronie AVRSide.

Program nie jest zbyt skomplikowany i działa następująco:

- odbiera za pomocą sprzętowego UART-u komendy i dane wysyłane przez aplikację sterującą,
- po zdekodowaniu komendy wykonuje odpowiednią operację na interfejsie ISP,
- wysyła do aplikacji sterującej potwierdzenie wykonania lub informację o błędzie.

Komunikacja szeregową

Po konwersji w FT8U232 transmisja przez USB nie różni się z punktu widzenia mikrokontrolera (z wyjątkiem kilku szczegółów) od zwykłej, wielokrotnie omawianej transmisji przez UART. Główną (pozytywną) różnicą jest możliwość praktycznie dowolnego ustawiania szybkości transmisji, bez kłopotliwego doboru częstotliwości rezonansowej kwarców.

W tym programatorze używamy 250 kbd - wartości nietypowej dla zwykłego RS232, ale przy oscylatorze 8 MHz możliwej do precyzyjnego taktowania.

Aplikacja przesyła do programatora ramkę danych sformatowaną w następujący sposób:

- bajt [0] - długość obszaru danych (w 8-bajtowych blokach),
- bajt [1] - kod komendy,
- bajty [2]...[7] - informacje zależne od kodu komendy,
- obszar danych (wielokrotność 8-bajtowego bloku).

Zauważmy, że w strukturze ramki brak jest obszaru kontrolnego (CRC lub chociażby sumy kontrolnej). Wynika to nie tylko z uproszczeń w pierwszej, testowej wersji, ale ma również konkretny cel: praktyczne sprawdzenie stopnia niezawodności toru przesyłowego USB<->UART. Z dotychczasowych prób laboratoryjnych wynika, że transmisja jest praktycznie bezawaryjna. Jednak w razie potrzeby można bez problemu rozbudować program o procedury dodatkowej kontroli poprawności odbierania danych.

Odbiornik pracuje w oparciu o przerwanie. Po odebraniu bajtu

[0] zostaje odczytana i zapamiętana oczekiwana długość bieżącej ramki. Kolejne bajty są wpisywane do bufora odbioru aż do zakończenia ramki. Równocześnie pracuje licznik *timeoutu* odbioru, który pozwala na wykrycie ramek niekompletnych (bez jego zastosowania niekompletna ramka powodowałaby zawieszenie odbiornika w oczekiwaniu na zakończenie jej przesyłania). Skompletowanie ramki powoduje ustawienie flagi przyjęcia transmisji - jest to sygnał dla pętli głównej, że należy odczytać bufor odbiornika, zinterpretować zawarte w nim dane i wykonać odpowiednie dla danego polecenia operacje.

Po realizacji komendy programator wysyła do aplikacji odpowiedź. Ramka zwrotna ma stały rozmiar 32 bajty. Bajt [0] zawsze określa kod błędu. Dalsze bajty mogą zawierać dane zależne od komendy. Nadajnik również pracuje z użyciem przerwania.

Wyczerpujące rozpisanie formatów poszczególnych komend znajdziemy w kodach źródłowych.

Operacje interfejsu ISP

Opiszemy tylko komendy potrzebne do obsługi mikrokontrolera w podstawowym zakresie:

- *SET_PARAMS* - przekazuje do programatora typ docelowego mikrokontrolera oraz częstotliwość taktowania SCK (dla uproszczenia wyliczamy już w PC wartość argumentu dla pętli *_delay_loop_1*),
- *READ_FUSES* - odczytuje stan bitów konfiguracyjnych, wartość bajtów kalibracyjnych oraz sygnatur (w tej wersji nie ma samoczynnego uzgadniania sygnatury z wybranym typem mikrokontrolera),
- *WRITE_FUSES* - zapisuje ustawione w aplikacji sterującej bity konfiguracji,
- *WRITE_FLASH_PAGE* - do programatora przesłana jest zawartość jednej strony pamięci Flash (rozmiar zależy od typu ATmega), zostaje ona przeladowana do bufora strony docelowego układu i przepisana do jego pamięci (można ten proces rozszerzyć o weryfikację wpisanej strony - dotychczas nie wydaje się to konieczne, ale może okazać się dopiero przy większej liczbie testów),

- *WRITE_EEPROM* - dla uzyskania lepszej wydajności transmisji dane EEPROM są przesyłane w 32-bajtowych blokach (choć każdy bajt musimy programować oddzielnie).

Jeśli przeanalizujemy kod źródłowy, to stwierdzimy, że bardzo łatwo rozbudować programator o następne funkcje oparte na tym samym mechanizmie przesłania komendy. Mogą to być też polecenia w ogóle niezwiązane z obsługą AVR, ale np. wykorzystujące inne wbudowane interfejsy.

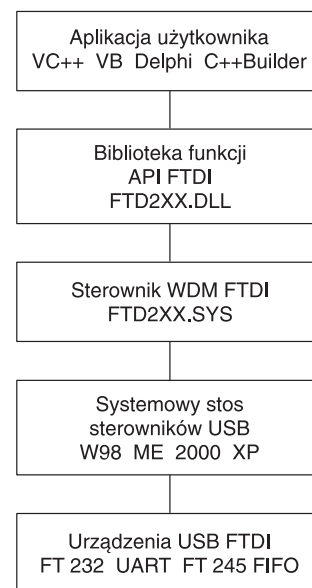
Opisane komendy można również rozszerzać o dodatkowe opcje (sposób kontroli czasu zapisu, pomijanie przy zapisie bajtów 0xFF itp.).

Sam przebieg sterowania interfejsem ISP oczywiście nie pozwala na dowolność - musi być całkowicie zgodny ze specyfikacją. Nie powtarzamy tu wielokrotnie już przedstawianych opisów obsługi ISP - zainteresowani mogą samodzielnie przeanalizować programową realizację tabeli komend programowania szeregowego.

Jak to wygląda od strony komputera

Do obsługi programatora wykorzystujemy również (jak przy konfiguracji deskryptorów) driver D2XX. Schemat blokowy przesyłania danych pokazano na **rys. 3**. Jest ono złożone i wieloetapowe, ale z punktu widzenia aplikacji użytkowej sprowadza się do odpowiedniego stosowania funkcji API zawartych w bibliotece *ftd2xx.dll*. Może się wydawać, że jest to zbędne dokładanie sobie roboty - są do wyboru także tradycyjne metody oparte na zwykłej obsłudze portu szeregowego. Jednak D2XX oferuje znacznie większą kontrolę nad pracą układu, co ma zwłaszcza znaczenie dla przyspieszenia dwustronnej wymiany danych z użyciem dużej liczby niewielkich, oddzielnych pakietów (szczegółowe omówienie tematu znajduje się w nacie aplikacyjnej AN232-04 *Latency and data throughput*). Dużo łatwiejsza jest także identyfikacja układu według opisu w deskrytorze oraz ustawianie praktycznie dowolnej szybkości transmisji.

Biblioteka *ftd2xx.dll* zawiera dwa - stosowane zamiennie -



Rys. 3. Organizacja przesyłu danych USB pomiędzy aplikacją a programatorem

zbiory funkcji do obsługi transmisji: interfejs klasyczny oraz nowy interfejs FT-Win32 API (wygodny w stosowaniu, gdyż jego funkcje - chociaż odwołują się bezpośrednio do stosu USB - mają składnię prawie zgodną z dotychczasowymi systemowymi funkcjami Windows API). Dodatkowo znajdziemy tu funkcje związane z rozszerzonymi możliwościami wersji BM oraz obsługę dostępu do pamięci EEPROM (wykorzystanie wolnego obszaru dla własnych potrzeb).

W przykładowej aplikacji sterującej programatorem używamy interfejsu FT-Win32. Aplikacja została napisana w Delphi, z czym wiązała się konieczność sporządzenia modułu nagłówkowego tłumaczącego wywołania C funkcji bibliotecznych na Object Pascal - znajdziemy go w materiałach pomocniczych.

Komponent komunikacji z układem FT8U232BM

Dla ułatwienia obsługi łączności z układem FTDI potrzebne funkcje zostały zgrupowane w łatwym do stosowania komponencie Delphi. Jest on przystosowany głównie do potrzeb przyrządów warsztatowych (jak opisywany właśnie programator) operujących na niezbyt wielkich przesyłach danych (zazwyczaj w trybie *half-duplex*) i często korzystających z dodatkowych linii kontrolnych UART. Dlatego opiera się na najprostszym, synchronicz-

nym trybie wywoływania funkcji i posługuje się tylko jednym dodatkowym wątkiem dla kontroli odbiornika i stanu linii.

Parametry pracy UART-u FT8U232 ustawiamy według potrzeb za pomocą właściwości typu *published* (dostępnych już na etapie projektowania). Większość z nich jest analogiczna jak dla zwykłego portu szeregowego. Należy jednak zwrócić uwagę na następujące aspekty:

- Mechanizm ustawiania szybkości jest zupełnie inny niż w tradycyjnym porcie; jest ona uzyskiwana z podziału częstotliwości wzorcowej 3,0 MHz przez programowo ustawiany dzielnik o części całkowitej i ułamkowej. Pozwala to na dużą elastyczność w doborze - zawsze jednak warto przeliczyć jak dokładnie układ będzie w stanie odwzorować wprowadzoną przez nas wartość. Np. dla programatora szybkość 250 kbd jest możliwa do ustawienia dokładnie, nawet bez użycia ułamkowego podzielnika ($250000 = 3000000/12$).
- Pakietowy sposób przesyłu danych magistralą USB zazwyczaj wymaga, w przypadku większych bloków danych, ustawienia kontroli przepływu (zewnętrzny układ podłączony do FT8U232 musi być informowany o konieczności wstrzymania nadawania w chwilach oczekiwania na odczyt przez USB wypełnionych buforów tego układu). Mamy do wyboru dwie pary linii *handshakingu* sprzętowego oraz *handshaking* programowy. Zauważmy, że w przypadku omawianego programatora możemy zrezygnować z kontroli przepływu, gdyż jednorazowo przesyłany blok danych jest zawsze mniejszy od pojemności bufora FT8U232.

Kilka właściwości dotyczy samego połączenia USB:

- *DeviceDescription*, *DeviceSerialNumber* oraz *DeviceOpenBy* pozwalają na łączenie z konkretnym urządzeniem na podstawie jego nazwy lub numeru wpisanych do deskryptora.
- *UsbInputBufferTimeout* oraz *UsbInputPacketSize* pozwalają na optymalizację transmisji USB pod kątem przewidywanej organizacji wymiany danych (szcze-

góły - jak wspomniano wyżej - w notach aplikacyjnych FTDI).

Obsługa transmisji z użyciem komponentu jest bardzo prosta - polega na użyciu tylko kilku właściwości *runtime*:

- *DevicePresent* służy do łączenia się i rozłączania z urządzeniem,
- *xxxState* pozwala na odczyt linii wejściowych i ustawianie wyjściowych,
- wpis do *WriteBuffer* wysyła dane, a z *ReadBuffer* pobieramy dane odebrane,
- *DisconnectError* pozwala zidentyfikować fizyczne odłączenie urządzenia w trakcie nawiązanej łączności,
- zdarzenia *Onxxxx* pozwalają na samodzielne oprogramowanie według potrzeb sytuacji typowych dla przebiegu transmisji (połączenie, rozłączenie, błąd przesyłu, odbiór bloku danych).

Kod źródłowy komponentu jest załączony w materiałach (na CD-EP8/2003B). Stwierdzone dotychczas usterki dotyczą możliwości wystąpienia błędu AV (*access violation*) podczas likwidacji instancji komponentu bezpośrednio po rozłączeniu urządzenia - wątek potomny nie zawsze zdąży się zatrzymać, odwołując się do nieistniejących już elementów. Pozostało to nieskorygowane, gdyż - jak można zobaczyć w przykładowym oprogramowaniu - bardzo łatwo ten efekt wyeliminować.

Przykład oprogramowania sterującego

Sterowanie programatorem zostało wbudowane w środowisko AVRSide dla kompilatora AVR-GCC. Pozwoliło to na wykorzystanie istniejących już modułów związanych z programowaniem (odczyt i kontrola plików *.hex, wskaźnik postępu, okno dialogowe bitów konfiguracji). Nic jednak nie stoi na przeszkodzie, aby - dysponując kodami i znając format komend - w razie potrzeby dopisać własne narzędzia.

Po wybraniu z menu AVRSide poleceń programowania lub konfiguracji układu docelowego zostaje wykonana następująca sekwencja czynności:

- jest dynamicznie tworzone i wyświetlone odpowiednie okno dialogowe (rys. 4),
- w przypadku programowania zostają załadowane (wraz z kont-



Rys. 4. Okno dialogowe konfiguracji po odczycie bitów fuse

rolą poprawności) pliki wyjściowe projektu (Intel hex lub binarne dla kodu programu oraz zawartości EEPROM),

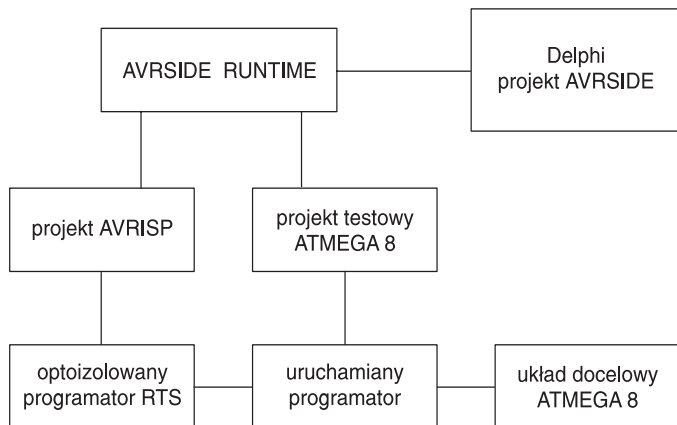
- zostaje nawiązane połączenie z wybranym typem programatora - w razie stwierdzenia błędu operacje zostają zatrzymane z odpowiednim komunikatem ostrzegawczym,
- realizowane są (samoczynnie podczas programowania lub w interakcji z użytkownikiem podczas ustawiania konfiguracji) potrzebne komendy,
- po zakończeniu połączenie z programatorem zostaje przerwane, a okno dialogowe zamknięte i zlikwidowane.

Zestaw AVRSide + programator obsługuje jedynie mikrokontrolery z rodziny ATmega. Wynika to z:

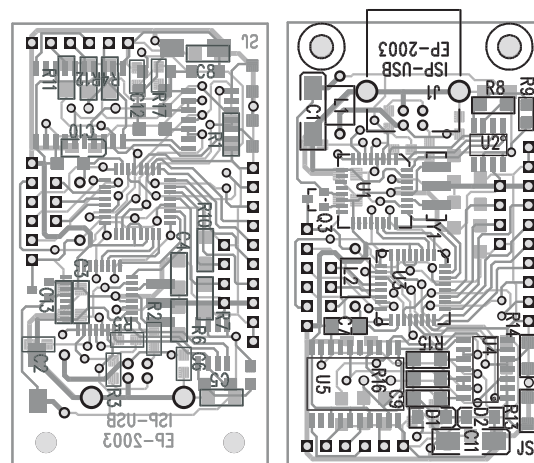
- braku możliwości programowania w AVR-GCC najmniejszych układów niewyposażonych w pamięć RAM,
- szybkiego wycofywania przez Atmela z produkcji układów ze starszej serii 90xx.

Ostateczne uruchomienie i sprawdzenie działania programatora polega na podłączeniu zmontowanej i zaprogramowanej płytki, dołączeniu do interfejsu ISP jakiegoś testowego (najlepiej już wypróbowanego) układu z mikrokontrolerem ATmega i próbie jego obsługi z poziomu AVRSide. Zawsze wskazane jest rozpoczęcie od odczytu bitów konfiguracji (*fuse*). Możliwe komunikaty o błędach obejmują:

- brak podłączonego programatora
- przyczyn szukajmy po stronie magistrali USB lub w błędnym zaprogramowaniu deskryptorów FT8U232,
- błędna konfiguracja programatora (przy odczycie *fuse*ów lub po pewnej chwili od otwarcia okna)



Rys. 5. Przykładowy schemat blokowy stanowiska uruchomieniowego - pozwala na równoległe rozwijanie i sprawdzanie programu sterującego Delphi oraz programu C dla mikrokontrolera



Rys. 6. Schemat montażowy płytki drukowanej

- świadczy o niewłaściwej pracy ATmega8 programatora (mikrokontroler nie odpowiedział na pierwszą komendę ustawiającą typ docelowego układu i częstotliwość pracy) lub o usterce na połączeniach UART; czasem taki objaw świadczy po prostu o nieudanym starcie przy podawaniu zasilania z magistrali USB - wtedy wystarczy odłączyć i po chwili ponownie podłączyć płytkę),
 - niedostępne urządzenie docelowe
 - brak poprawnej odpowiedzi programowanego mikrokontrolera na sekwencję wejścia w tryb ISP.

Z powyższego łatwo się domyślić, że - poza kilkoma banalnymi przypadkami błędnych połączeń - lokalizacja usterki może nie być łatwa i nie sposób podać recepty na każde możliwe niedomaganie układu. Jednak jest to cecha wszystkich składanych samodzielnie bardziej skomplikowanych urządzeń.

Gdy uzyskamy właściwy odczyt bitów *fuse* (można to ocenić np. po wartościach sygnatur), możemy wykonać próbne zapisy konfiguracji oraz programowania. Na razie nie ma opcji weryfikacji zawartości *flasha*, więc poprawność wpisanego kodu musimy ocenić po działaniu docelowego układu (miganie LED-a kontrolnego itp.). Dotychczasowe testy wykazały dużą niezawodność zapisu - więc wprowadzanie weryfikacji (znacznie spowalniającej cały proces) może okazać się całkiem zbędne. Dla testowanego docelowego ATmega8 uzyskano czas programowania 8 kB Flasha równy ok. 4 s. Nie stwierdzono znaczącej różnicy przy użyciu stałego (zgodnego ze specyfikacją) opóźnienia cyklu zapisu oraz przy aktywnej kontroli zakończenia cyklu. W chwili pisania artykułu testy na większych ATmegaach były dopiero w przygotowaniu - po wszelkie aktualności

sięgajmy na stronę AVRSide (www.avrside.fr.pl).

Jeśli chcemy samodzielnie rozwijać oprogramowanie układu (zarówno *firmware* mikrokontrolera, jak i współpracującą nadrzędną aplikację) - musimy się wyposażyć w bardziej rozbudowane stanowisko. Na rys. 5 przedstawiono przykład konfiguracji używany przy uruchamianiu pierwszych modeli programatora.

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

Dodatkowe informacje:

- strona firmowa FTDI: www.ftdichip.com,
- strona AVRSide: www.avrside.fr.pl,
- strona WinAVR: <http://sourceforge.net/projects/winavr>.

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: <http://www.ep.com.pl/pdf/wrzesien03.htm>.