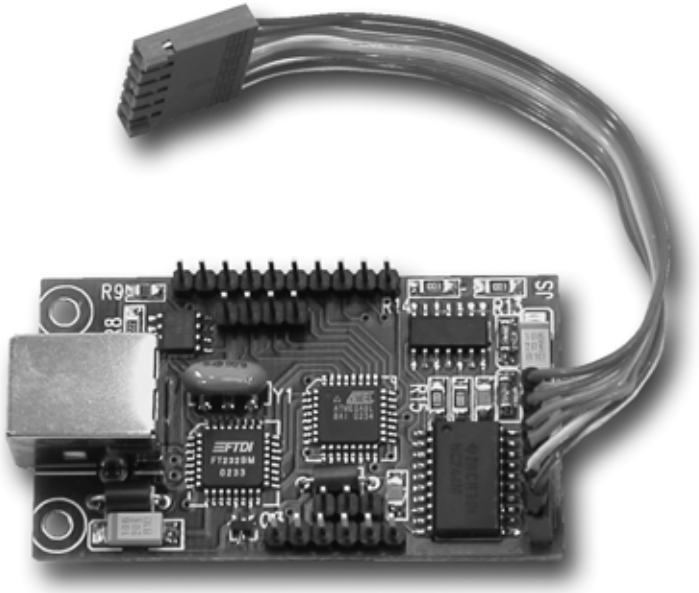
Programator USB mikrokontrolerów ATmega - ISP, część 2

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 środowiska 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 szeregowa

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 intefejsu 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 przeładowana 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 (chociaż 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.

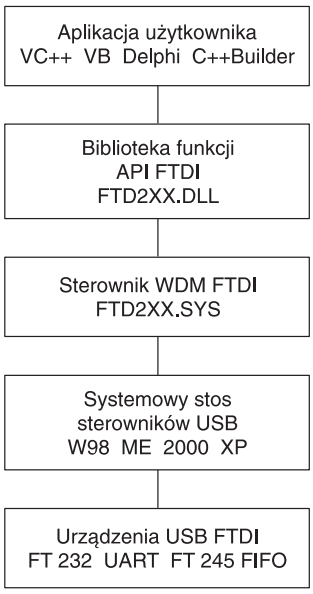
Opisane komendy można również rozszerzać o dodatkowe opcje (sposób kontroli czasu zapisu, pomijanie przy zapisie bajtów 0xFF itp.).

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 przyśpieszenia dwustronnej wymiany danych z użyciem dużej liczby niewielkich, oddzielnych pakietów (szczegółowe omówienie tematu znajduje się w nocie aplikacyjnej AN232-04 *Latency and data throughput*). Dużo łatwiejsza jest także identyfikacja układu według opisu w deskryptorze 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 (szczegóły - 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).

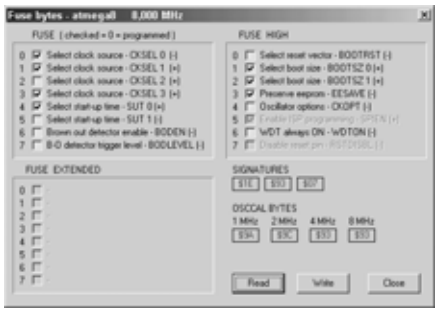
Kod źródłowy komponentu jest załączony w materiałach (na CD-EP8/2003B). Stwierdzone dotychczas usterki dotyczą możliwości wystąpienia błędu AV (*access violation*) podczas likwidacji instancji komponentu bezpośrednio po rozłączeniu urządzenia - wątek potomny nie zawsze zdąży się zatrzymać, odwołując 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 kontrolą 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.



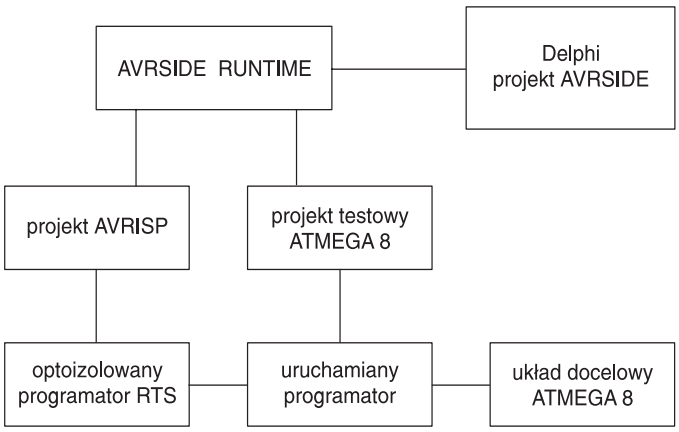
Rys. 4. Okno dialogowe konfiguracji po odczycie bitów *fuse*

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

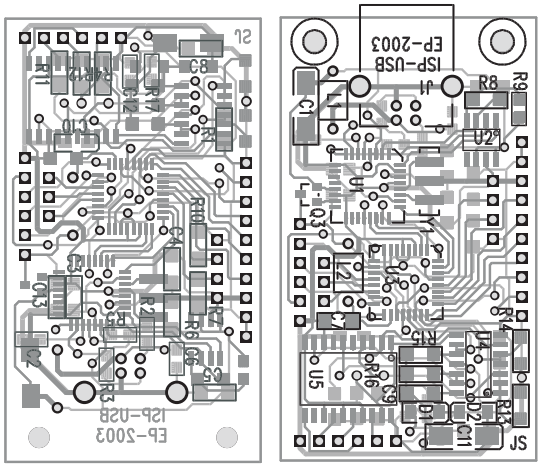
- braku możliwości programowania w AVR-GCC najmniejszych układów niewyposażonych w pamięć RAM,
- szybkiego wycofywania przez Atmela z produkcji układów ze starszej serii 90xx.

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 *fuseów* 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 podłączeń - 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 ATmegach 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***.