Programator mikrokontrolerów AVR i AT89S8252

AVT-515

Kilka dni wytężonej pracy zajęło mi opracowanie programatora ISP dla mikrokontrolerów firmy ATMEL z rodziny AT90, ATmega i AT89S8252. Dla mnie to przykład kolejnego urządzenia, o którego wartości stanowi nie tyle liczba i nowoczesność zastosowanych podzespołów, a program wykonywany przez mikrokontroler.

***Rekomendacje**: nie ma wśród naszych Czytelników takich, którzy nie wiedzą, co to jest mikrokontroler. Dlatego właśnie projekt prezentowany w artykule może zainteresować praktycznie każdego, kto korzysta lub zamierza korzystać z niezwykle popularnych AVR-ów.*

Zapewne większość Czytelników zajmujących się mikrokontrolerami zna wiele różnych typów prostych programatorów. Opisy wielu z nich były również publikowane na łamach EP. Większość z nich jest sterowana sygnałami portu równoległego. Ma to swoje wady i zalety, nie chcę ich tutaj omawiać. Osobiście mam jednak pewne obawy przed podłączaniem czegoś do portu drukarkowego. Raz, że nie jest to zbyt wygodne, jeśli już jest tam podłączona drukarka. Dwa, że czasami drobna awaria programatora prowadzi do uszkodzenia portu, a to już jest duży kłopot. Zdecydowałem się na skonstruowanie programatora podłączanego do portu szeregowego komputera PC. Istnieje co prawda nota aplikacyjna firmy ATMEL (AVR910) opisująca taki programator, jednak wydał mi się on zbyt skomplikowany. Dodatkową trudnością było zdobycie i zaprogramowanie mikrokontrolera AT90S1200.

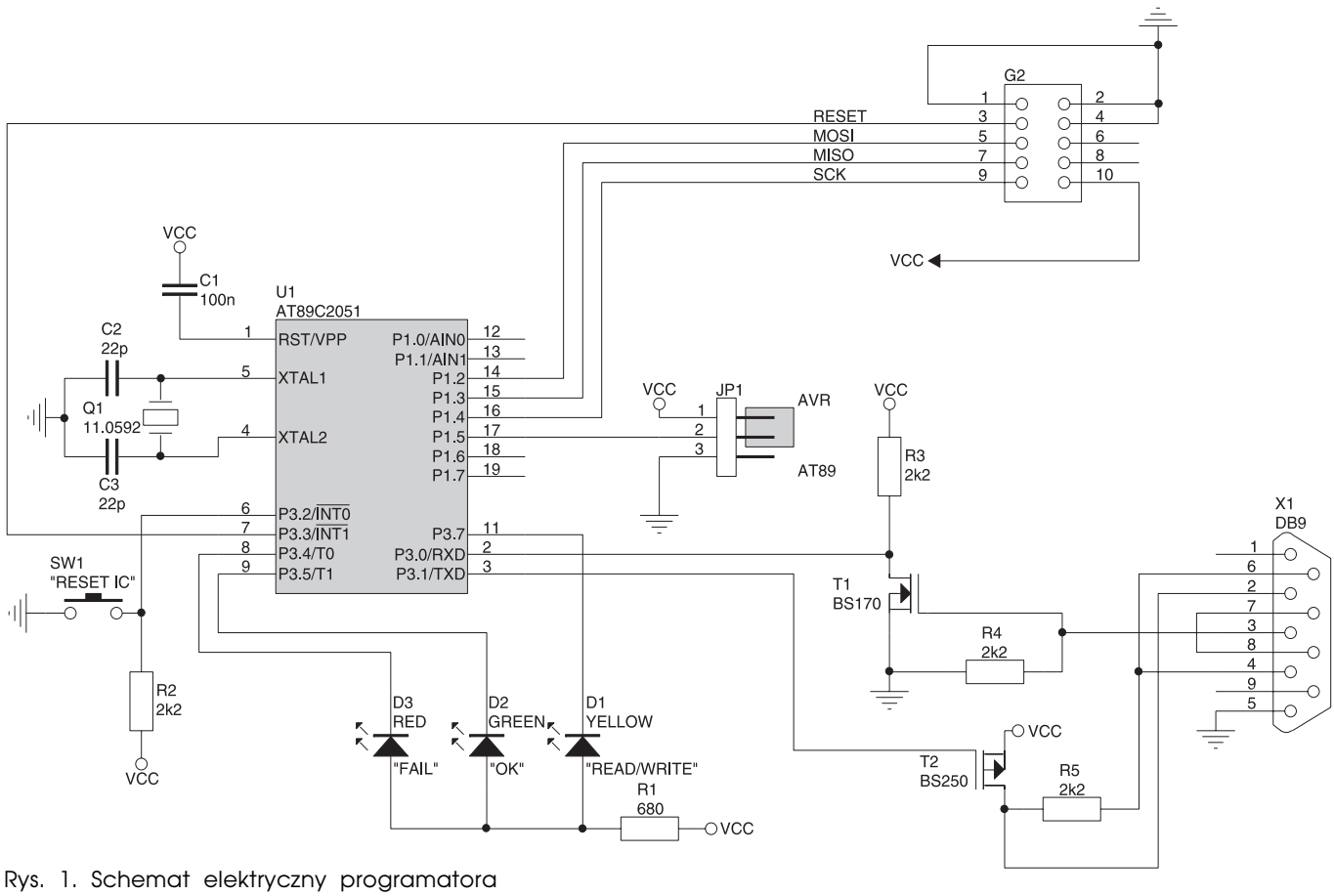
Opis działania

Schemat elektryczny programatora pokazano na **rys. 1**. Nie posiada on własnego stabilizatora napięcia zasilania, ponieważ zasilanie jest pobierane z programowanego układu. Założyłem również, że zapisywany lub odczytywany mikrokontroler posiada własny generator zegarowy podłączony do wejścia XTAL1. Oczywiście wystarczy, jeśli ma podłączony rezonator ceramiczny lub kwarcowy.

Sygnał interfejsu szeregowego z komputera PC doprowadzany jest na bramkę tranzystora T1 (BS170). Pełni on rolę układu translacji poziomów napięć RS232 (logiczna „1“: -12...-5V, logiczne „0“: +5...+12V) na poziomy TTL. Dalej sygnał podawany jest na wejście RxD interfejsu UART mikrokontrolera. Drugie wyprowadzenie UART - TxD - jest podłączone do bramki tranzystora BS250 pracującego jako driver dokonujący translacji poziomu napięć TTL na wymagane przez RS232. Uwaga: proszę nie zmieniać wartości re-

Programator opisany w artykule można wykorzystać do programowania następujących mikrokontrolerów:

- AT89S8252
- AT90S1200 i AT90S1200 rev.C
- AT90S2313
- AT90S2323
- AT90S2333
- AT90S2343
- AT90S4414
- AT90S4433
- AT90S4434
- AT90S8515
- AT90S8535
- ATMega 103
- ATMega 161
- ATMega 163
- ATMega 603
- ATMega 83



Rys. 1. Schemat elektryczny programatora

zystancji dołączonych do tranzystorów! Wartości rezystorów R3, R4 i R5 powinny być sobie równe i zawierać się w granicach 1,8...2,4 kΩ. Tranzystor T2 zasilany jest napięciem dodatnim pochodzącym z płytki programatora oraz napięciem ujemnym pochodzącym z linii sygnałowej interfejsu RS232. Zakres napięć wyjściowych waha się od -5...+5V. Niestety, interfejs RS232 starszych komputerów PC (386, 486 i być może niektóre Pentium I) może nie akceptować takich poziomów napięć. Nie mają z nim żadnego problemu nowsze modele PC, mające interfejs szeregowy zgodny ze specyfikacją EIA232.

Zastosowany w programatorze mikrokontroler AT89C2051 spełnia rolę terminala realizującego polecenia aplikacji sterującej. Zaimplementowałem w nim funkcje programowego interfejsu SPI. Programowanie docelowego układu odbywa się z jego wykorzystaniem. Diody LED (D1, D2 i D3) służą do sygnalizacji stanu programatora. Ich kolory dobrane są „intuicyjnie", zgodnie z ludzkimi przyzwyczajeniami. Świecenie diody zielonej (D2) oznacza poprawne nawiązanie komunikacji z komputerem PC oraz pozytywny rezultat przeprowadzanej operacji. Świecenie diody żółtej (D1) oznacza, że programator przesyła dane - jest zajęty. Dioda czerwona (D3) świeci się w sytuacji, gdy wystąpi błąd - operacja została zakończona niepowodzeniem.

Zwarcie przycisku SW1 powoduje wystawienie sygnału zerującego dla programowanego mikrokontrolera. Aktywny poziom sygnału zerującego (0 lub 1) jest wybierany zworką JP1.

Opis programu dla mikrokontrolera AT89C2051

Program dla mikrokontrolera U1 został napisany w języku C z niewielkimi wstawkami w języku asembler. Po skompilowaniu zajmuje on około 1,6 kB przy włączonej opcji optymalizacji kodu wynikowego pod kątem szybkości jego działania. Do napisania programu posłużył mi kompilator firmy Raisonance. Program kompilowany był przy włączonym modelu pamięci TINY. Każdy, kto będzie chciał wykonać modyfikacje, może się posłużyć darmową wersją demonstracyjną tego pakietu dostępną na stronie producenta (*http://www.raisonance.com*), umożliwiającą kompilację i uruchamianie programów do 4 kB kodu.

Program realizuje funkcję terminala dołączanego do komputera PC z wbudowanym interpreterem poleceń, wykonującego polecenia przesyłane przez program AVR Prog. Z małym wyjątkiem, do komunikacji używane są funkcje wejścia/wyjścia z biblioteki producenta. Ten „mały wyjątek" to funkcja przesyłająca znaki do komputera PC, która wymagała odrębnej implementacji. Standardowo bowiem *putchar()* po napotkaniu kodu 0x0A przesyła dodatkowo 0x0D, tworząc typową sekwencję końca linii znaków (powrót karetki i nowa linia). Nie jest to pożądane przy przesyłaniu danych w postaci binarnej zawartych w pamięci mikrokontrolera. Najprostszym rozwiązaniem była własna implementacja funkcji *putchar()*. Jak widać na **list. 1**, nie należy ona do zbyt skomplikowa-

Rys. 2. Wygląd okna programu AVR Prog

nych. Nie wykorzystuje mechanizmu przerwań, zeruje flagę TI oraz zapisuje dane do bufora UART i oczekuje na ustawienie TI będące sygnałem zakończenia transmisji bajtu.

Funkcja korzysta z nastaw predefiniowanych przez producenta pakietu. Szybkość transmisji określana jest przez polecenie *#pragma DEFJ(TIM1_INIT=0xFD)* zawierające wartość inicjującą Timer 1 sterujący pracą UART.

Funkcja *getchar()* nie nadaje się do zastosowania w naszej aplikacji, ponieważ zwraca ona echo odebranego znaku, co zgodne jest ze specyfikacją standardu ANSI C i przydatne w przypadku terminala znakowego, jednak zupełnie niepotrzebne w przypadku programatora. Na szczęście producent zdefiniował również inną funkcję, znacznie bardziej przydatną dla naszych celów. Jest to funkcja o nazwie *getkey()* zwracająca wartość bajtu odczytanego z UART.

Funkcje zapisu *(wrser)* i odczytu *(rdser)* interfejsu SPI napisane zostały w języku asembler. Są to niewielkie procedury, zaledwie po kilka bajtów każda. Ich implementacja w C, aczkolwiek możliwa, zajmowałaby więcej miejsca w pamięci programu oraz byłaby nieco bardziej skomplikowana. Dzięki asemblerowi łatwo jest zapanować nad funkcją i stanem flagi przeniesienia C mikrokontrolera, a także nad czasem wykonywania instrukcji. Z tego samego powodu również funkcja (pętla) absorbująca CPU na czas około 1 ms jest napisana w asemblerze *(delayms)*. Deklaracji bitów wyprowadzeń programowego interfejsu SPI (MISO, MOSI i SCK) dokonałem w module napisanym w języku asembler tak, aby były one dostępne zarówno z poziomu aplikacji w języku C, jak i z poziomu aplikacji w asemblerze.

Reszta programu to bardzo rozbudowany warunek *switch* rozpatrujący odebrane znaki i podejmujący akcję w zależności od poleceń przesłanych przez AVRProg. Protokoły komunikacyjne AT89 i AT90 różnią się znacznie pomiędzy sobą. Z tego też powodu, w wielu miejscach programu konieczny jest rozdział funkcji na realizowane przez AVR i realizowane przez AT89, mimo iż przesyłane polecenia mają identyczną postać. Podstawowa różnica w protokole polega na tym, że AT89 wymaga 3 bajtów dla każdego z poleceń, natomiast AT90 i ATMega wymagają 4.

Szczególną uwagę należy zwrócić na poprawną implementację funkcji *spiinit()*. Funkcja ta wprowadza kod rozkazu *Programming Enable*, umożliwiając zapis pamięci Flash i EEPROM. Układy z serii AVR wymagają, aby dokonana została synchronizacja interfejsów *Master* (programator) i *Slave* (mikrokontroler). Dodatkowo funkcja ta wywoływana jest na koniec cyklu zapisu, przed przełączeniem do cyklu odczytu.

Uwaga: bez poprawnej pracy funkcji *spiiinit()* nie jest możliwy zapis i odczyt danych z wykorzystaniem interfejsu SPI.

Programator uniwersalny

Programator można bez najmniejszych problemów zintegrować z Bascom AVR oraz AVR Studio. Alternatywnie do obsługi programatora można wykorzystać program AVR Prog.

Aplikacja sterująca - AVR Prog

Do sterowania pracą programatora wykorzystałem bezpłatny program AVR Prog (**rys. 2**), dostępny na stronie internetowej firmy Atmel. Jest to aplikacja wykonana dla środowiska Windows. Nie wymaga przeprowadzania żadnej instalacji. Program rozprowadzany jest jako archiwum w formacie ZIP, które zawiera plik wykonywalny EXE. Należy go po prostu skopiować na dysk twardy i utworzyć skrót ułatwiający uruchomienie.

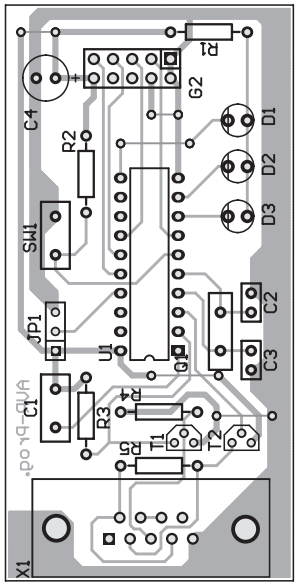
List. 1. Implementacja funkcji putchar()

```
//własna funkcja putchar (dla każdego 0x0A
//oryginalny putchar dodaje 0x0D)
int putchar (const int c) //nagłówek zgodny
                //z biblioteką producenta
{
  SBUF = c;     //zapis bufora UART
  TI = 0;       //zerowanie flagi TI,
                //początek transmisji
  while (!TI); //oczekiwanie na przesłanie
                //bajtu
}
```

Uwaga: program nie uruchomi się bez podłączonego urządzenia, które wyśle identyfikator zawierający 3 pierwsze znaki „AVR" podczas próby nawiązania komunikacji.

Po uruchomieniu program testuje porty szeregowe, sprawdzając możliwość komunikacji i w ten sposób sam wykrywa obecność (bądź też nieobecność) podłączonego programatora. Aplikacja jest bardzo prosta w użyciu. Obsługuje się ją identycznie jak większość programów napisanych dla środowiska Windows. Korzystając z pola *Device* umieszczonego na dole okienka, należy wybrać typ programowanego mikrokontrolera. Od tego wyboru zależeć będzie algorytm zapisu i odczytu pamięci mikrokontrolera. W górnej części okienka jest umieszczony przycisk *Browse* (*Hex file*). Korzystając z niego, należy wskazać zbiór w formacie HEX zawierający dane do zapisu do pamięci Flash lub EEPROM. W ten sam sposób wskazuje się również nazwę zbioru, w którym zostaną zapamiętane odczytane dane. Teraz, po wykonaniu wyżej opisywanych czynności, można przeprowadzić programowanie pamięci, odczytać ją lub porównać jej zawartość z danymi zapamiętanymi na dysku. Operacje te można przeprowadzać niezależnie dla obu rodzajów pamięci.

Uwaga: przed zapisem pamięci Flash wykonywana jest instrukcja kasująca zarówno zawartość pamięci Flash, jak i EEPROM. W przypadku zapisu danych do EEPROM - zawartość Flash nie jest usuwana.



Rys. 3. Schemat montażowy płytki programatora

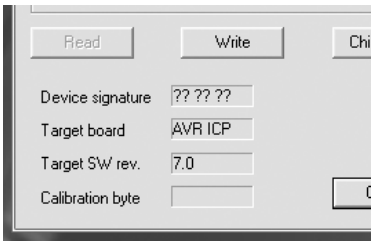
WYKAZ ELEMENTÓW

Rezystory
R1: 680Ω
R2, R3, R4, R5: 2,2kΩ
Kondensatory
C1: 0,1µF
C2, C3: 22pF
Półprzewodniki
U1: AT89C2051 (zaprogramowany)
T1: BS170
T2: BS250
D1: LED żółta
D2: LED zielona
D3: LED czerwona
Różne
X1: DB9
JP1: jumper 3x1
SW1: SW-PB
G2: złącze 5x2
Q1: 11,0592MHz

Uruchomienie układu

Schemat montażowy programatora pokazano na **rys. 3**. Poprawnie zmontowany układ nie powinien nastręczać żadnych trudności przy uruchomieniu. Aby przetestować jego działanie, należy podłączyć go do programowanego układu i typowym kablem do transmisji szeregowej do komputera PC. Pamiętajmy o tym, że programator pobiera zasilanie z programowanej płytki! Diody LED sygnalizujące stan programatora powinny zaświecić się, a następnie kolejno zgasnąć.

Teraz należy przystąpić do próby komunikacji z aplikacją AVR Prog. Tuż po jej uruchomieniu próbuje ona zidentyfikować typ podłączonego programatora oraz listę programowanych układów. Wysłanie komendy „t" powoduje zaświecenie się diody zielonej („OK"), co jest sygnałem poprawnej wymiany danych programatora z aplikacją sterującą. Po krótkiej chwili od nawiązania połączenia, na ekranie PC powinno się ukazać okno robocze programu AVR Prog. Teraz wybierzmy przycisk *Advanced*. Na ekranie powinna ukazać się informacja jak na **rys. 4**. Po zamknięciu okienka *Advanced*, najlepszym testem jest próba zapisu pamięci Flash i EEPROM. Pamiętajmy o wyborze właściwego typu mikrokontrolera z listy programowanych układów.



Rys. 4. Identyfikacja rodzaju programatora po wybraniu opcji *Advanced*

Typ programowanego układu przechowywany jest w pamięci RAM programatora. Czasami przy wyłączeniach płytki uruchomieniowej, mimo iż każdorazowo sekwencja programowania zaczyna się od przesłania kodu wyboru programowanego układu, może się zdarzyć, że przy pierwszej próbie zapisu lub odczytu danych otrzymamy komunikat *Cannot enter programming mode*. Należy wówczas ponowić próbę operacji, a jeśli nie da to rezultatu, zamknąć aplikację AVR Prog i uruchomić ją ponownie. Operacja taka przywraca wszystkie nastawy programatora, ponownie ustawiając poprawny tryb jego pracy.

Jacek Bogusz, AVT
jacek.bogusz@ep.com.pl

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