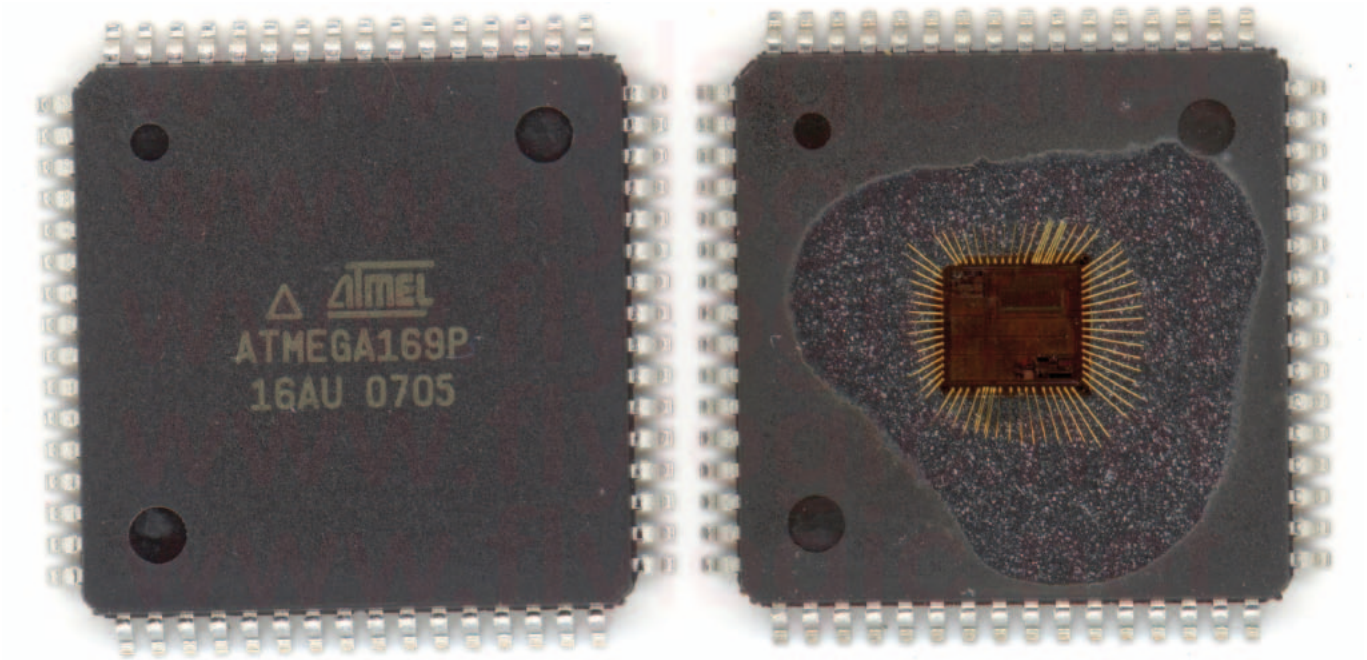


Samoprogramowanie AVR (1)

Opis gotowego bootloadera

Konstruktorom i elektronikom zajmującym się systemami embedded bardzo dobrze znane są zalety stosowania specjalnych programów startowych, tzw. bootloaderów. Dzięki nim można w prosty sposób zbudować mechanizm zdalnej aktualizacji oprogramowania, bez konieczności podłączania programatora. W przeszłości pisaliśmy w EP na temat bootloaderów przeznaczonych dla mikrokontrolerów AVR. W związku z bardzo dużym zainteresowaniem tym tematem oraz ogromną popularnością mikrokontrolerów ATmega, zdecydowaliśmy się opisać ciekawe rozwiązania bootloaderów.

W Internecie można znaleźć kilka darmowych propozycji gotowego programu do samoprogramowania mikrokontrolerów AVR. Interesującą propozycją jest program MegaLoad, który obsługuje prawie każdy mikrokontroler z rodziny ATmega. Dodatkowo, jest on prosty w konfiguracji. Kolejną jego zaletą jest to, że program obsługi napisano w C#.NET, co zapewnia jego względnie łatwą migrację pomiędzy różnymi systemami. Drugą interesującą propozycją jest program napisany przez inżynierów Atmela. Do jego atutów można zaliczyć to, że bootloader jest obsługiwany z poziomu AVR Studio. Na początku zajmiemy się MegaLoadem, gdyż jest najbardziej uniwersalnym rozwiązaniem. W kolejnej części cyklu opiszemy bootloader zgodnym z notą aplikacyjną AVR109.

Zaczynamy

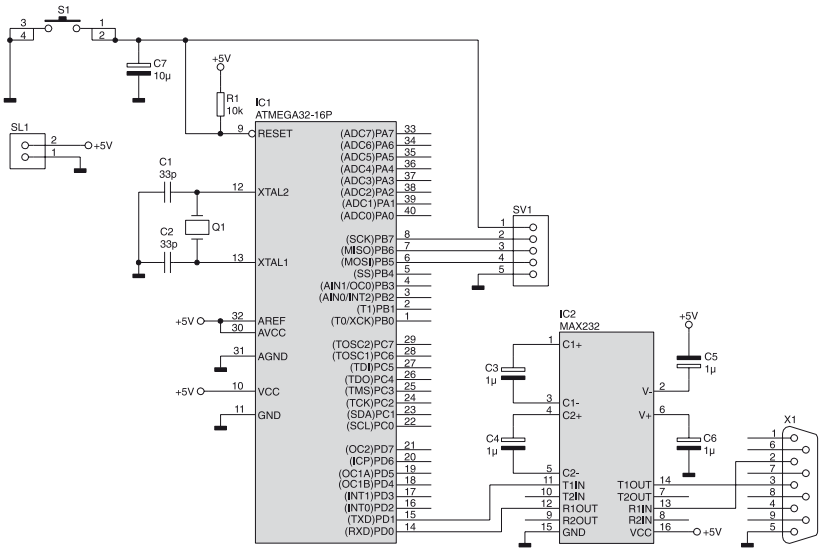
Program bootloadera komunikuje się z komputerem za pomocą portu szeregowego (istnieje również możliwość wykorzystania interfejsu RS485). Na potrzeby tego artykułu zbudowano prostą aplikację, której schemat pokazano na **rys. 1**. Wykorzystano w niej mikrokontroler ATmega32, a do konwersji poziomów popularny MAX232. Dzięki temu prostemu układowi można przetestować bootloader.

Do uruchomienia będzie potrzebny dowolny programator procesorów AVR, np. PonyProg, który jest prosty w budowie, a oprogramowanie i schemat można za darmo ściągnąć ze strony *http://www.lancos.com/prog.html*. Jedyne, co trzeba zakupić, to części; nie trzeba nawet wykonywać płytki – układ można zmontować w postaci tzw. pająka. Do kompilacji programu zastosujemy program ICCAVR, którego wersję demonstracyjną można ściągnąć ze strony *http://www.imagecraft.com*. Jest ona wystarczająca na potrzeby tego artykułu. Program MegaLoad oraz pliki bootloadera możemy pobrać ze strony autora, tj. *http://www.microsyl.com*.

Konfiguracja i kompilacja bootloadera

Pierwszym krokiem jest wybór możliwości bootloadera, czyli czy wystarcza nam tylko programowanie pamięci programu, czy też potrzeba, aby mógł on również zaprogramować wewnętrzną pamięć EEPROM i bezpieczniki BLB (*Boot Lock Bits*). Funkcja programowania EEPROM wiąże się ze zwiększeniem rozmiaru programu, a co za tym idzie, innych ustawień bezpieczników decydujących o wielkości pamięci poświęconej na bootloader, czyli sekcji BLS (z 256 na 512 słowa). W artykule będzie zaprezentowana opcja z możliwością programowania pamięci Flash, EEPROM oraz bezpieczników BLB, czyli wszystkiego, co oferuje program MegaLoad.

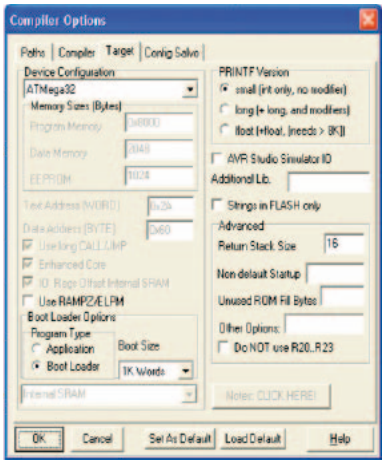
Program bootloadera możemy skompilować, gdy mamy zainstalowany kompilator ICC, program MegaLoad oraz rozpakowany



Rys. 1.

(ze strony ściągamy w postaci archiwum zip) kod źródłowy bootloadera. Po uruchomieniu ICC, otwieramy projekt (wybieramy z menu *Project–>Open) BootLoad.prj*. Znajdziemy go oczywiście w katalogu, do którego rozpakowaliśmy kod źródłowy. Po prawej stronie otwartego okna ukazują się pliki, które wchodzą w skład projektu; klikamy dwa razy na plik *main.c*, co spowoduje otwarcie go do edycji.

Teraz można rozpocząć konfigurację. Na początek w *MCU selection* ustawiamy dla jakiego typu procesora ma być skompilowany bootloader. Robimy to poprzez usunięcie znaku komentarza „//” przed odpowiednią nazwą typu mikrokontrolera (w naszym przypadku *#define MEGATYPE 32)*. Jednocześnie należy pamiętać, aby dodać „//” przed wszystkimi innymi nazwami typów, gdyż inaczej przy kompilacji zostanie zgłoszony błąd. W sekcji *MCU Frequency* należy wpisać wartość częstotliwości oscylatora w Hz, którego używa się do taktowania mikrokontrolera (np. dla kwarcu 8 MHz, *#define XTAL 8000000*). Teraz w sekcji *Bootload on UART x* trzeba wybrać, z którego portu szeregowego ma korzystać mikrokontroler do komunikacji z komputerem. Jeśli procesor ma tylko jeden port USART, to wybór jest oczywisty: *#define UART 0*. Tak też jest w przypadku ATmegi32. Gdy mikrokontroler ma więcej portów, wtedy można wybrać, z którego ma korzystać program. Postępujemy tak samo jak wcześniej, kasując „//” przed wybranym portem, a wstawiając „//” przed pozostałymi. W kolejnej sekcji o nazwie *BaudRate* ustawia się prędkość transmisji danych. Podaje się ją w bodach, czyli bitach/sekundę (np. *#define BAUDRATE 9600*). Teraz w sekcji *EEPROM programming* zaznacza się, czy bootloader ma mieć możliwość programowania wewnętrznej pamięci EEPROM, co jak już wcześniej wspomniałem, wiąże się z większym zapotrzebowaniem na pamięć programu.

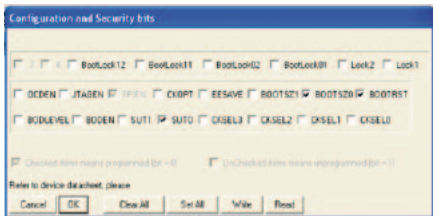


Rys. 2.

W sekcji *LockBit programming* decydujemy, czy potrzebna jest nam możliwość programowania bezpieczników.

Zamiast interfejsu RS232 *MegaLoad* może używać do komunikacji RS485 half duplex. Jeśli chcemy skorzystać z tej opcji, to wyłączamy RS232 i konfigurujemy sekcję *RS485*. Zostało jeszcze tylko otwarcie pliku *assembly.*s i wstawienie 1 przy nazwie mikrokontrolera, dla którego ma być skompilowany bootloader (przy innych musi być 0!). Po zapisaniu zmian można przejść do kompilacji.

Należy zacząć od konfiguracji kompilatora, to znaczy wyboru procesora, dla którego kompilowany jest program oraz że plik wynikowy będzie używany jako bootloader. Trzeba też podać wielkość sekcji BSL. Dokonuje się tego, wybierając z menu *Project–>Options...* Po otwarciu nowego okna w zakładce *Target* wybieramy procesor, a na dole w polu *Program type* zaznaczamy *Boot Loader* i wybieramy *Boot size*, czyli wielkość sekcji BSL. Tak jak wspomniano wcześniej, ustawiamy 256 Words w przypadku bez programowania EEPROM, a z jego programowaniem – 512 Words. Na **rys. 2** pokazano przykładowy wygląd ekranu konfiguracji.



Rys. 3.

Po tych czynnościach można już skompilować projekt, naciskając ikonkę *Project Build* lub *F9* . Jeśli wszystko zostało wykonane poprawnie, to kompilacja powinna zakończyć się sukcesem.

Programowanie procesora

Czas najwyższy na to, żeby zaprogramować procesor. Zastosujemy do tego programator PonyProg, ale schemat postępowania jest oczywiście taki sam dla każdego programatora. Zaczynamy od ustawienia bezpieczników. Sposoby na wywoływanie bootloadera są dwa: albo z programu aplikacyjnego, albo podczas restartu procesora. W przykładzie skorzystamy z drugiej opcji. W tym celu należy ustawić odpowiednie bezpieczniki *BOOTSZx*. Ich kombinacja jest uzależniona od typu procesora oraz od wielkości BSL, jaką, potrzebujemy (kombinacje znajdziemy w dokumentacji procesora) oraz jeśli chcemy, by po resecie startował program bootloadera, to ustawiamy bezpiecznik *BOOTRST*. Dla przykładowego układu będziemy potrzebowali BSL o wielkości 512 słowa, co odpowiada konfiguracji BOOTsz1 = 1, BOOTSZ0 = 0. Ustawienia bezpieczników przedstawia **rys. 3**.

Po zaznaczeniu odpowiednich bezpieczników naciskamy przycisk *Write*, w ten sposób zapisują się nasze ustawienia do procesora. Teraz trzeba załadować program bootloadera. Zaczynamy od otwarcia skompilowanego wcześniej bootloadera, wybieramy z menu *File–>Open Program (FLASH) File...* i otwieramy plik *BootLoad.hex*. Musimy także wybrać typ mikrokontrolera, który chcemy zaprogramować. Dokonujemy tego w menu *Device*, a następnie programuje jego pamięć Flash poprzez wybranie z menu *Command–>Write Program (FLASH)*. Pamięć programu powinna zostać zaprogramowana, a następnie zweryfikowana. Na **rys. 4** przedstawiono programowanie procesora z przykładu. Jeśli wszystko pójdzie bez przeszkód, to przechodzimy do ostatniego i moim zdaniem najprzyjemniejszego etapu, w którym będziemy mogli zobaczyć efekt naszej pracy.

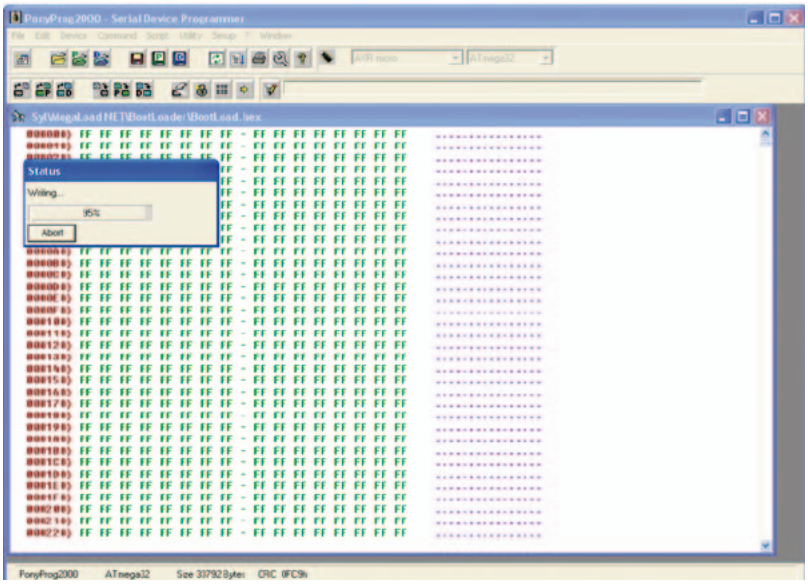
Programowanie

Ostatnim etapem będzie podłączenie do komputera układu i zaprogramowanie mikrokontrolera (sekcji aplikacji). Uruchamiamy program MegaLoad i podłączamy układ poprzez port *com*. Po uruchomieniu programu wyświetlane jest okienko jak na **rys. 5**.

Interfejs programu jest przejrzysty i czytelny, więc z jego obsługą nie powinno być problemów.

Wybieramy port, do którego jest podłączony nasz procesor i z jaką prędkością ma się odbyć transmisja (koniecznie taka sama, jak zostało ustawione podczas konfiguracji). Następnie wybieramy plik, w którym jest skompilowany program (w sekcji *File to be programed in the Flash*, klikamy przycisk *Open*) oraz jeśli chcemy zaprogramować EEPROM, to musimy wskazać plik (w sekcji *File to be programed in the EEPROM*, klikamy przycisk *Open*), w którym jest zawartość EEPROM. Teraz jeśli chcemy, wybieramy ustawienia bezpieczników (tylko ostrożnie), robimy to w sekcji *BootLoader Lock Bits to be programed* (znaczenia poszczególnych ustawień bitów omawiałem w poprzednim artykule). Po dokonaniu tych czynności zostaje nam tylko zresetowanie procesora, co powinno rozpocząć proces programowania. Jeśli wszystko przebiegło bez kłopotów, to wgrany program powinien działać od razu, jednak jeśliby się tak nie stało, może pomóc naciśnięcie przycisku *Send reset* w sekcji *Command* lub odłączenie zasilania układu. Jeśli te operacje nie pomogą, to znaczy, że gdzieś podczas konfigurowania bootloadera popełniliśmy błąd lub prędkość z jaką komunikuje się mikrokontroler z PC, jest źle dobrana.

Możemy sprawdzić teraz, czy nasz program rzeczywiście został bezbłędnie zapisany w pamięci mikrokontrolera. W tym celu należy podłączyć programator i zweryfikować pamięć procesora z plikiem, który wgraliśmy. Należy pamiętać, że w pamięci znajduje się nie tylko nasz program, ale również bootloader, dlatego jeśli będziemy weryfikować całą pamięć (włącznie z sekcją BSL) i porównywać ją z plikiem, który wgraliśmy, to rezultat weryfikacji będzie negatywny.

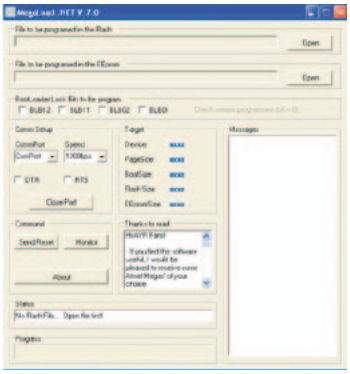


Rys. 4.

Teraz, po każdorazowym uruchomieniu/restarcie układu program bootloadera sprawdzi, czy ma programować pamięć. Jeśli wynik będzie negatywny, wówczas normalnie zostanie uruchomiona aplikacja zapisana w pamięci mikrokontrolera.

Podsumowanie

Bootloader autorstwa Sylvain Bissonnette jest bardzo elastyczny, przez co nadaje się do wielu różnych zastosowań. W oknie programu znajduje się sekcja *Thanks to read*, w której autor prosi o wsparcie jego pracy, do czego gorąco zachęcam, gdyż jego program jest wykonany solidnie, działa naprawdę stabilnie i nadaje się do różnych zastosowań, o czym mam nadzieję również przekonają się Czytelnicy.



Rys. 5.

Paweł Klaja
pklaja@o2.pl