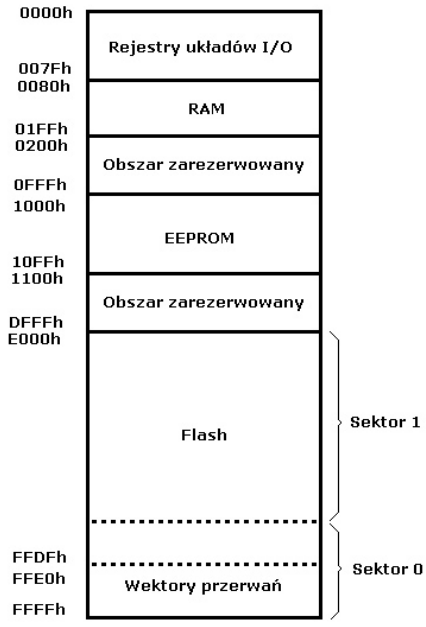
Bootloader dla mikrokontrolerów ST7FLITE3x

Gdy wtyczka zostanie już pomyślnie zainstalowana w równie delikatnym gnieździe konieczne jest odczekanie dłuższego, bądź też nieco krótszego czasu, w zależności od rozmiaru pamięci oraz częstotliwości pracy mikrokontrolera, podczas którego następuje programowanie pamięci. W większości przypadków programowany jest cały obszar pamięci mikrokontrolera, niezależnie czy program zajmuje sto bajtów czy osiem kilobajtów. Ponieważ perspektywa spędzenia połowy życia na cyklicznym podłączaniu i odłączaniu wtyczki programatora nie jest dla mnie zbyt kusząca, postanowiłem opracować jakiś alternatywny sposób programowania pamięci, który nie absorbowałby programisty tak bardzo, jak robi to programator ST7–STICK i robiłby to w nieco krótszym czasie.

Wgrywanie programu do pamięci Flash mikrokontrolerów ST7 za pomocą programatora ST7–STICK jest czynnością, która znacznie wydłuża całkowity czas poświęcony na opracowanie aplikacji oraz w pewnym sensie przyczynia się do spadku wydajności programisty. Konieczność każdorazowego podłączania (w celu zaprogramowania pamięci) i odłączania (w celu uruchomienia mikrokontrolera) wtyczki programatora niepotrzebnie rozprasza i dekoncentruje programistę, który zamiast skupić się na pisanej aplikacji zastanawia się czy delikatna wtyczka rozpadnie się tym, czy dopiero następnym razem.

Pamięć Flash mikrokontrolerów ST7LITE może być programowana na dwa sposoby: wewnątrzukładowo (ICP – *In–Circuit Programming*) oraz z poziomu działającego programu (IAP – *In Application Programming*). Pierwsza metoda programowania jest wykorzystywana przez programator ST7–STICK, a drugą metodę zastosowałem w opracowanym przeze mnie bootloaderze. Zrozumienie zasady jego działania wymaga zapoznania się z organizacją pamięci w przestrzeni adresowej mikrokontrolera, która jest przedstawiona na **rys. 1**. Jak widać, pamięć programu Flash jest podzielona na dwa sektory: 0 oraz 1. Rozmiar tych sektorów jest konfigurowalny w dość szerokim zakresie za pomocą bajtów konfiguracyjnych *Option Byte*. Niezmiernie istotną kwestią jest to, że w trybie programowania IAP mamy możliwość zaprogramowania tylko i wyłącznie sektora 1, niezależnie od jego rozmiaru. Przedstawiany w artykule bootloader zajmuje najmniejszy możliwy obszar sektora 0, czyli 512 bajtów, z czego znaczna część pozostaje niewykorzystana. Jak łatwo obliczyć, do dyspozycji ładowanego programu pozostaje obszar o rozmiarze 7,5 kB. Należy zwrócić uwagę na fakt, iż pewne istotne obszary pamięci są niemożliwe do zaprogramowania z poziomu aplikacji. Obszar wektorów przerwań zlokalizowany jest niestety w sektorze 0, tak więc nie mamy możliwości jego zmodyfikowania z poziomu działającego programu. Nie jest to jednak poważny problem i można go rozwiązać w dość prosty sposób, który zostanie przedstawiony w dalszej części artykułu. Natomiast nie do przeskoczenia jest kwestia bajtów konfiguracyjnych – mogą być one zaprogramowane wyłącznie w trybie ICP. W związku z tym na etapie programowania mikrokontrolera kodem bootloadera należy ustawić docelowe dla wgrywanej za jego pomocą aplikacji bity konfiguracyjne. Komunikacja komputera z bootloaderem odbywa się poprzez łącze szeregowe, przy czym po stronie mikrokontrolera



Rys. 1.

transmisja jest realizowana poprzez układ LINSCI. Ogranicza to zastosowanie bootloadera tylko do mikrokontrolerów z grupy LITE3x. Program bootloadera sprawdza, czy tuż po wyzerowaniu mikrokontrolera wyprowadzenie PB0 znajduje się w stanie niskim. Jeżeli tak, to następuje uruchomienie procedury programowania pamięci, w przeciwnym razie następuje skok do obszaru pamięci użytkownika. W związku z tym proces programowania pamięci rozpoczyna się od wymuszenia niskiego stanu na wyprowadzeniu PB0 oraz podaniu na wyprowadzenie RESET impulsu zerującego mikrokontroler. Następnie nawiązywana jest komunikacja po łączu szeregowym i rozpoczyna

List. 1.

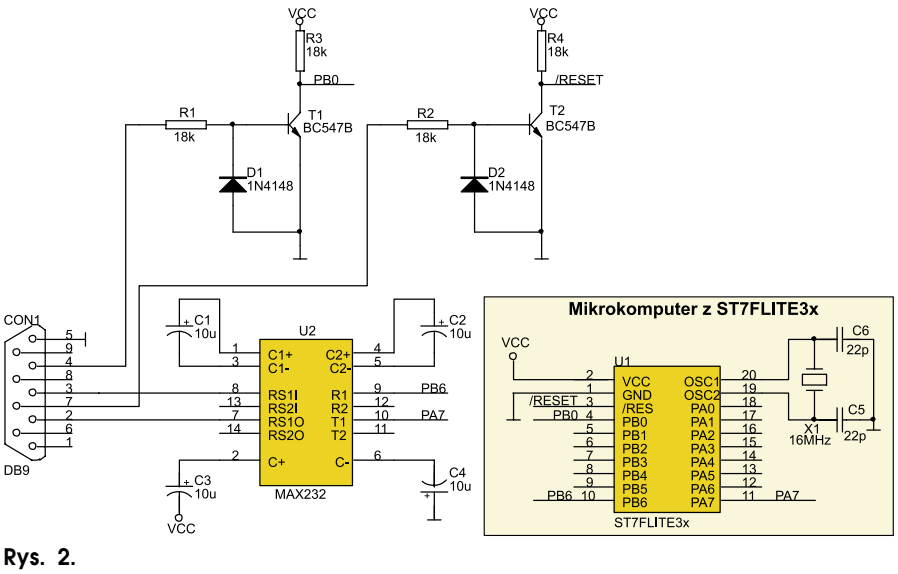
```
flashCode
     CLR   X
     BSET  FCSR, #1
fcLoop
     LD A,(bufor,X)
     LD ([flashPointer.w],X), A
     INC   X
     CP X, #32
     JRNE  fcLoop
     BSET  FCSR, #0
     BTJT  FCSR, #0, *
     RET
```

List. 2.

```
copyFlashToRam
     CLR   X
loop
     LD A, (flashCode,X)
     LD (ramCode,X), A
     INC   X
     CP X, #19
     JRNE  loop
     RET
```

się programowanie obszaru pamięci użytkownika. Po zaprogramowaniu pamięci następuje zwolnienie wyprowadzenia PB0 oraz ponowne wyzerowanie mikrokontrolera, po którym następuje wykonanie załadowanego uprzednio programu użytkownika.

Ponieważ, jak już wspominałem na wstępie, głównym celem zastosowania bootloadera była chęć wyeliminowania jakichkolwiek czynności przeprowadzanych ręcznie na sprzęcie, zadanie sterowania stanem wyprowadzeń PB0 oraz RESET spoczywa na komputerze PC oraz aplikacji sterującej. Schemat części sprzętowej niezbędnej do poprawnej i w pełni automatycznej pracy bootloadera przedstawiony jest na **rys. 2**. Oprócz układu MAX232 będącego konwerterem poziomów napięć, w skład „programatora" wchodzą dwa inwertery na tranzystorach NPN sterowane liniami DTR i RTS portu RS232. Rozwiązanie



Rys. 2.

List. 3.

```
.getch
     BTJF  SCISR, #SCISR_RDRF, *
     LD A, SCIDR
     RET
```

List. 4.

```
.putch
     BTJF  SCISR, #SCISR_TDRE, *
     LD SCIDR, A
     RET
```

takie jest znane chociażby z układu programowania pamięci Flash mikrokontrolerów LPC2100, który był dla mnie inspiracją przy opracowywaniu układu bootloadera.

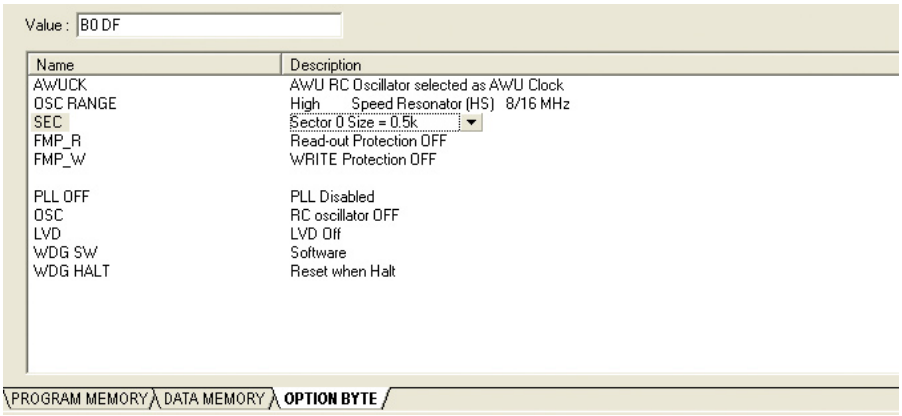
Program bootloadera

Kod bootloadera został napisany w asemblerze i składa się z kilku prostych procedur. Najważniejszą procedurą jest procedura kopiująca bufor pamięci RAM do pamięci Flash. Przed zapisem danych do obszaru pamięci Flash należy ustawić bit LAT, znajdujący się w rejestrze FCSR, natomiast po zapisaniu wszystkich 32 bajtów danych należy ustawić bit PGM, wyzwalając tym samym proces programowania. Po zakończeniu procesu programowania bit PGM zostanie sprzętowo wyzerowany, tak więc należy przed zapisaniem następnych danych poczekać na wyzerowanie bitu PGM. Kod procedury przedstawiono na **list. 1**.

Ze względu na fakt, iż podczas programowania pamięci Flash dostęp do niej jest niemożliwy, procedura programowania pamięci Flash musi zostać umieszczona w pamięci RAM. Kod procedury kopiującej obszar pamięci Flash do pamięci RAM przedstawiono na **list. 2**.

Procedura odbierająca dane nadchodzące z komputera jest bardzo prosta: oczekuje na ustawienie flagi RDRF (*Received Data Ready Flag*) znajdującej się w rejestrze SCISR, a następnie odczytuje rejestr SCIDR i jego zawartość zwraca w rejestrze A. Kod procedury przedstawiono na **list. 3**.

Procedura wysyłająca dane do komputera jest zbliżona w działaniu do procedury wysyłającej. Na początku procedury następuje sprawdzenie flagi TDRE (*Transmit Data Register Empty*). W przypadku, gdy jest wyzerowana następuje oczekiwanie na jej ustawienie, a następnie zawartość rejestru A jest przepisywana do rejestru SCIDR.



Rys. 3.

Kod procedury przedstawiono na **list. 4**.

Kod ładowanego przez bootloader programu jest przesyłany w 32–bajtowych pakietach, poprzedzonych dwubajtowym adresem w pamięci Flash, pod który pakiet ma zostać zapisany. Do odbioru pakietu została przygotowana procedura, której kod przedstawiono na **list. 5**.

Zasadnicza część programu bootloadera rozpoczyna się od sprawdzenia stanu wyprowadzenia PB0. Jeśli wyprowadzenie znajduje się w stanie wysokim, następuje skok względny do instrukcji skoku bezwzględnego pod adres wektora zerowania użytkownika. Jeśli wyprowadzenie PB0 znajdowało się w stanie niskim następuje uruchomienie programu bootloadera. Na początku inicjowany jest układ LINSCI, następnie procedura programowania pamięci Flash jest kopiowana do pamięci RAM. Kod programu przedstawiono na **list. 6**.

Kod bootloadera (po skompilowaniu programu, który udostępniamy na CD–EP10/2007B) należy zapisać w pamięci mikrokontrolera ST7FLITE3x za pomocą programatora ICP, przy czym należy zwrócić szczególną uwagę na poprawne zaprogramowanie na tym etapie bajtu konfiguracyjnego. Konieczne jest ustawienie rozmiaru sektora 0 na 0,5 kB (**rys. 3**) oraz ustawić źródło taktowania odpowiednie dla wgrywanego za pomocą bootloadera programu użytkownika. Zmiana wartości bajtu konfiguracyjnego nie jest możliwa z poziomu bootloadera i w razie konieczności zmiany konfiguracji należy skorzystać z programatora ICP (np. ST7–STICK lub podobny).

List. 5.
```
.getbuf
     CLR  X
getbufloop
     CALL getch
     LD (bufor,X), A
     INC  X
     CP X, #32
     JRNE getbufloop
     RET
```

List. 6.
```
._reset
  BSET PBOR, #0           ; włączamy pull-up
  BTJT PBDR, #0, endBoot  ; jeśli w stanie wysokim to skok do endBoot

  LD A, #%11000000        ; 38400bps @ 8 MHz (16MHz ext) / 19200bps @ 4MHz
                          ; (8MHz ext)
  ;LD  A, #%00001001      ; 115200bps  @ 3,6864 MHz (7,3728MHz ext)
  ;LD  A, #%00010010      ; 57600bps @ 3,6864 MHz (7,3728MHz ext)
  ;LD  A, #%01100100      ; 4800bps  @ 3,6864 MHz (7,3728MHz ext)
  LD SCIBRR, A            ;
  LD A, #%00001100        ; TX en / RX en
  LD SCICR2, A            ;

  CALL copyFlashToRam     ; skopiowanie kodu programu z pamięci Flash do RAM

  CALL getch              ; odbiór pierwszego bajtu
  CP A, #$55              ; sprawdzenie czy równy 0x55
  JRNE endBoot            ; jeśli nie to skok do endBoot
  PUSH A                  ; zapamiętanie odebranego znaku na stosie
  CALL getch              ; odebranie drugiego bajtu
  CP A, #$AA              ; sprawdzenie czy równy 0xAA
  JRNE endBoot            ; jeśli nie to skok do endBoot
  CALL putch              ; odesłanie drugiego bajtu
  POP  A                  ; odtworzenie ze stosu pierwszego bajtu
  CALL putch              ; odesłanie pierwszego bajtu
  CALL getch              ; odebranie pierwszego bajtu klucza odblokowującego
                          ; Flash
  LD FCSR, A              ; i zapisanie go do FCSR
  CALL getch              ; odebranie drugiego bajtu klucza odblokowującego
                          ; Flash
  LD FCSR, A              ; i zapisanie go do FCSR
  LD A, #$55              ; zgłoszenie gotowości na odebranie danych
  CALL putch
mainLoop
  CALL getch              ; odebranie starszego bajtu wskaźnika adresu
  LD flashPointer, A
  CALL getch              ; odebranie młodszego bajtu wskaźnika adresu
  LD {flashPointer+1}, A
  LD A, #$55
  CALL putch              ; zgłoszenie gotowości
  CALL getbuf             ; odebranie 32 bajtów danych
  CALL ramCode            ; wywołanie procedury programującej Flash
  LD A, #$55              ; zgłoszenie gotowości na odebranie danych
  CALL putch
  JRA  mainLoop   ; skok na początek pętli
```

Program sterujący

Program sterujący ładowaniem programów z wykorzystaniem bootloadera opsywanego w artykule jest aplikacją konsolową, którą w prosty sposób zintegrujemy ze środowiskiem ST7 Visual Develop (sposób jej przeprowadzenia opiszemy za miesiąc). Dzięki temu do zaprogramowania pamięci będzie wymagane tylko jedno kliknięcie w ikonę na pasku narzędzi lub wybranie polecenia z menu środowiska ST7Visual Develop. Program sterujący przyjmuje trzy parametry: ścieżkę dostępu do pliku z kodem programu, numer portu COM, do którego podłączony jest układ oraz prędkość transmisji w bitach na sekundę. Kod programu może być przygotowany w trzech formatach plików: Motorola S–record (**.s19*), Intel Hex (**.hex*) oraz w formacie binarnym (**.bin*). Prędkość transmisji może być wybrana dowolnie, spośród typowych prędkości transmisji łączem RS232.

Radosław Kwiecień, EP
radoslaw.kwiecien@ep.com.pl

Komplet materiałów (program bootloadera w postaci źródłowej oraz skompilowanej, program sterujący oraz przykładowy program testowy) znajduje się na płycie CD–EP3/2008B oraz jest dostępny do ściągnięcia ze strony internetowej EP.