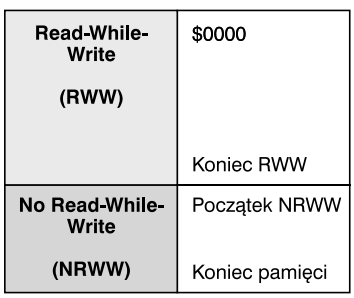
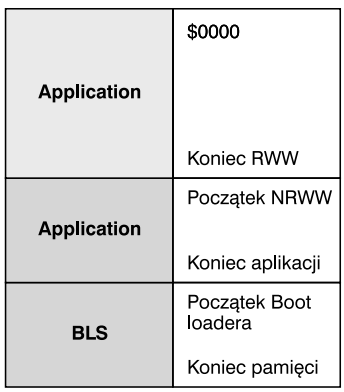
Bootloader dla mikrokontrolerów AVR

Procesory AVR mogą być programowane kilkoma sposobami: równolegle, szeregowo (ISP), przez interfejs JTAG. Wszystkie wymienione sposoby wymagają jednak programatora i niosą za sobą pewne ograniczenia. Programowanie równoległe na przykład wymaga wyciągnięcia mikroprocesora z podstawki, programowanie ISP i JTAG jest co prawda możliwe w systemie, ale w przypadku JTAG–a nie możemy używać linii, do których jest podpięty programator. Przy programowaniu ISP mogą natomiast wystąpić zakłócenia (gdy do wyprowadzeń mikrokontrolera oprócz programatora jest dołączone jeszcze inne urządzenie), które uniemożliwią poprawne zaprogramowanie pamięci. Firma Atmel proponuje nam jednak jeszcze jeden sposób programowania pamięci Flash, mianowicie samoprogramowanie i właśnie tym zagadnieniem zajmiemy się w artykule. Postaram się dokładnie opisać ten sposób programowania, zarówno od strony teoretycznej, jaki i praktycznej.

Gdy używamy *bootloadera*, pamięć procesora jest podzielona na dwie części *Read–While–Write* (RWW) oraz *No Read–While–Write* (NRWW). W przypadku kiedy nie korzystamy z samoprogramowania takiego podziału nie ma. Organizacja pamięci Flash jest narzucana przez konstruktorów i jest inna dla każdego układu, dlatego szczegółów należy szukać w dokumentacji konkretnego procesora (w podrozdziale *Boot Loader Parameters*), którego zamierzamy użyć w naszym projekcie (np. dla ATmega8(L) RWW to obszar pomiędzy adresami 0x000––0xBFF, a NRWW obejmuje adresy 0xC00–xFFF). Podział pamięci zilustrowano na **rys. 1.** Czym różnią się te dwie sekcje? Otóż podczas zapisu RWW praca procesora nie zostaje zatrzymana i możliwe jest działanie programu pod warunkiem, że znajduje się on w NRWW. W czasie ładowania programu do RWW **nie można** odwoływać się do komórek pamięci znajdującej się w tej sekcji, ponieważ może to spowodować zawieszenie układu i błąd przy zapisie. Zapis obszaru NRWW wiąże się z zatrzymaniem pracy CPU.



Rys. 1. Podział pamięci Flash mikrokontrolera AVR na obszary RWW i NRWW



Rys. 2. Podział pamięci Flash mikrokontrolera AVR na obszary RWW i NRWW i BLS

Boot Loader Section

Pamięć Flash jest podzielona również na części, w których przechowywany jest program użytkownika (*Application section*) oraz program *bootloadera* (*Boot Loader Section* w skrócie BLS). Obszar BLS znajduje się w obszarze NRWW, wynika to z tego, że instrukcja służąca do zapisu pamięci (SPM) może być wywoływana tylko z sekcji NRWW. Program może znajdować się zarówno w sekcji RWW i NRWW, zależy

Uwaga!
Czytelników zainteresowanych kursem FPGA przepraszamy – jego kolejna część, z przyczyn technicznych, ukaże się w kolejnym wydaniu EP.

to od tego czy wykorzystamy cały obszar NRWW dla programu *bootloadra* (wielkość wybieramy bezpiecznikami BOOTSZx). Jak już wcześniej wspominałem wielkość obszaru NRWW jest narzucona przez konstruktorów, co do wielkości BLS mamy jednak dużo do powiedzenia. Jej wielkość możemy ustalić bezpiecznikami BOOTSZx. Mamy do wyboru cztery możliwości. Wielkość BLS może się wahać w zależności od ustawień (BOOTSZx) i modelu procesora od 256 B, aż do 8 kB.

Na **rys. 2** przedstawiono przykładowy podział.

Informacji na temat możliwych wielkości możemy znaleźć w dokumentacji procesora (w podrozdziale *Boot Loader Parameters*).

Włączenie bootloadera

Przed korzystaniem z samoprogramowania, niezbędne jest odpowiednie skonfigurowanie mikrokontrolera. Nowsze ATmega posiadają dodatkowy bit konfiguracyjni (SELFPRGEN), który po zaprogramowaniu umożliwia korzystanie z instrukcji SPM. Kolejnym krokiem jest ustalenie bitów konfiguracyjnych BOOTSZ, czyli wielkości BLS. Teraz ustalamy (jeśli to konieczne) poziom zabezpieczenia pamięci (bezpieczniki BLB). Będzie o tym szerzej mowa w końcowej części artykułu. Musimy się też zdecydować, jak będziemy wywoływać program *bootloadera*? Mamy do wyboru dwie możliwości: albo skokiem z programu do BLS na skutek jakiegoś zdarzenia, albo ustawiając bit konfiguracyjny BOOTRST. Dzięki temu wektor inicjalizujący zostaje przeniesiony do BLS, czyli procesor zawsze bezie startował od BLS i dopiero w BLS wywołujemy skok do sekcji aplikacji, aby wykonywać program właściwy.

System przerwań

Oczywiście nie zamierzam omawiać całego układu przerwań tylko jego aspekt związany z *bootloaderm*. Jak wiemy, bezpiecznik BOOTRST przenosi wektor inicjalizacji do BLS. Istnieje jednak możliwość przeniesienia wszystkich wektorów przerwań do BLS. Robimy to przez ustawienie bitu IVCE, a następnie w czasie czterech kolejnych cykli maszynowych ustawiamy bit IVSEL w rejestrze GICR. W nowszych procesorach bity te znajdują się w rejestrze MCUCR. Teraz wektory znajdują się na początku sekcji BLS w tym samym porządku i z tym samym przesunięciem jak w normalnym przypadku. Przeniesienie przerwań daje nam pewność, że nie odbędzie się skok do RWW (co może się stać, gdy nie zablokujemy przerwań) podczas jej programowania, co może spowodować błąd zapisu.

Bit	7	6	5	4	3	2	1	0
Nazwa bitu	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN

Rys. 3. Budowa rejestru SPMCR

Instrukcja SPM

Wszystkie operacje samoprogramowania wykonujemy używając instrukcji SPM. Do sprecyzowania operacji, jaką ma wykonać instrukcja służy rejestr funkcyjny SPMCR (w niektórych modelach SPMCSR). Organizację tego rejestru przedstawiono na **rys. 3**.

Bit 7 – SPMIE: Przerwanie SPM (*SPM Interrupt Enable*)

Kiedy SPMIE jest ustawiony i układ przerwań jest włączony, przerwanie SPM jest aktywne. Przerwanie SPM będzie generowane dopóki bit SPMEN (w SPMCR) jest wyzerowany. Jeśli zdecydujemy się na to, by używać tego przerwania, należy pamiętać o przeniesieniu wektorów przerwań do sekcji BLS.

Bit 6 – RWWSB: Zajętość sekcji RWW (*Read–While–Write Section Busy*)

Podczas wykonywania operacji zapisu lub czyszczenia pamięci w obszarze RWW bit ten jest ustawiony i nie wolno korzystać z sekcji RWW. Bit RWWSB zostanie wyzerowany, gdy ustawimy bit RWWSRE lub automatycznie jeśli zaczniemy wykonywać ładowanie bufora strony.

Bit 5 – Bit zarezerwowany

Bit 4 – RWWSRE: aktywowanie sekcji RWW (*Read–While–Write Section Read Enable*)

Programowanie sekcji RWW powoduje zablokowanie jej odczytywania. Odblokowanie RWW polega na poczekaniu aż operacje programowania zostaną zakończone (SPMEN=0), następnie ustawieniu bitu RWWSRE i SPMEN oraz wykonaniu instrukcji SPM. Po poprawnym wykonaniu tych czynności, sekcja RWW jest już odblokowana.

Bit 3 – BLBSET: Programowanie i odczytywanie Lock Bit (*Boot Lock Bit Set*)

Ustawienie bitów BLBSET i SPMEN, a następnie wykonanie instrukcji SPM spowoduje zapisanie bitów sterujących zgodnie z wcześniej ustawioną maską w rejestrze R0, rejestry R1 i Z są ignorowane. Ustawienie bitów BLBSET i SPMEN, a następnie wykonanie instrukcji LPM spowoduje odczytanie *Lock bits* lub *Fuse bits* (zależnie od ustawienia Z0). Szczegóły w dalszej części artykułu.

Bit 2 – PGWRT: Zapisywanie strony (*Page Write*)

Ustawienie bitów PGWRT i SPMEN oraz wykonanie instrukcji SPM, spowoduje zapisanie bufora tymczasowego w pamięci programu, rejestr Z adresuje pamięć.

Bit 1 – PGERS: Kasowanie strony (*Page Erase*)

Ustawienie bitów PGWRT i SPMEN oraz wykonanie instrukcji SPM, spowoduje skasowanie strony. W rejestrze Z znajduje się adres strony do skasowania.

Bit 0 – SPMEN: Zezwolenie na programowanie (*Store Program Memory Enable*)

Ten bit aktywuje na 4 cykle zegarowe po jego ustawieniu instrukcję SPM. Jeśli ustawimy oprócz SPMEN któryś z bitów: RWWSRE, BLBSET, PGWRT, PGERS, to instrukcja SPM nabiera specjalnego znaczenia (opis przy objaśnieniu bitu). Natomiast gdy ustawimy tylko SPMEN, to wywołanie SPM spowoduje załadowanie wartości z R0: R1 do bufora tymczasowego adresowanie odbywa się przez rejestr Z.

Adresowanie pamięci

Adresowanie pamięci programu podczas samoprogramowania odbywa się w sposób pośredni przez rejestr Z. Ponieważ rejestr ten wskazuje na słowa (16 bitów), a nie bajty, najmłodszy bit powinien być równy zero. Sam rejestr Z może zaadresować 32 k słów, czyli 64 kB. Z tego powodu przy użyciu procesorów o większej pamięci (np. ATmega2560) trzeba używać dodatkowego rejestru RAMPZ do adresowania pamięci. Pamięć Flash jest podzielona na strony. Z tego powodu

Bit	23	15	0
Rejestr	RAMPZ	Rejestr Z	
Znaczenie	Adres strony	Adres słowa	0

Rys. 4. Adresowanie pamięci z użyciem rejestrów Z i RAMPZ

adresowanie wygląda następująco: najstarsze bity wskazują na adres słowa w pamięci, młodsze na adres słowa w stronie (**rys. 4**).

Konkretna granica między adresem strony, a adresem słowa zależy od wielkości pamięci. W zależności od wielkości pamięci różna jest liczba słów na stronie, np. Atmega8 ma 32 słowa na stronie, a Atmega32 ma tych słów 64. Znowu musimy poczytać dokumentację konkretnego procesora, aby dowiedzieć się, jak należy prawidłowo zaadresować pamięć. Odpowiedź znajdziemy w wspominanym już wcześniej podrozdziale *Boot Loader Parameters*, gdzie w tabeli przedstawiono podział rejestru Z.

Samoprogramowanie pamięci Flash

Przedstawię teraz schemat postępowania podczas uaktualniania oprogramowania mikrokontrolera. Pamięć programujemy strona po stronie. Nim wydamy rozkaz programowania musimy wykasować stronę, którą chcemy zaprogramować oraz wypełnić bufor tymczasowy. Wszystkie te operacje są opisane w dalszej części. Kolejność w której wykonujemy kasowanie i ładowanie bufora jest dowolna i zależy tylko od nas. Ważne, by je wykonać przed zapisem strony. Należy jednak uważać co kasujemy, żeby przypadkiem nie skasować strony, która miała zostać nie modyfikowana. Utraconych w ten sposób danych już nie odzyskamy. Może się tak stać, gdy programujemy fragmenty pamięci, a nie całą.

Kasowanie

Nim zapiszemy nowe dane do pamięci musimy skasować stronę. Najpierw wpisujemy adres strony, którą chcemy wymazać do rejestru Z. Następnie ustawiamy bity PGERS i SPMEN w rejestrze SPMCR, a następnie w najbliższych 4 cyklach maszynowych musimy wykonać instrukcję SPM. Jeśli tego nie zrobimy, bity w rejestrze SPMCR zostaną wyczyszczone i pamięć nie zostanie wymazana. Z tego powodu zaleca się wyłączenie przerwań podczas operacji tego typu – przerwanie może nastąpić między ustawieniem bitów, a wywołaniem instrukcji SPM, co w rezultacie doprowadzi do niewykonania operacji.

Teraz trochę praktyki – krótki programik w asemblerze (**list. 1**), zawiera przykładową procedurę kasowania strony. Adres strony do zmazania znajduje się w rejestrach R13:R14.

List. 1. Program kasowania strony pamięci Flash

```
mov ZL, R13  ;Załadowanie LBS adresu strony
mov ZH,R14  ;Załadowanie MBS adresu strony
ldi R20, (1<<PGERS) | (1<<SPMEN)  ; ustawianie maski odpowiednich bitów aby in-
strukcja spm wykonała kasowaniem
cz_spm:  ;sprawdzanie czy ostatnia instrukcja spm zakończyła działanie
in R17, SPMCR
sbrc R17, SPMEN
rjmp Wait_spm
cli  ;zablokowanie układu przerwań
out SPMCR,R20  ;ładowanie ustawień instrukcji spm i jej wykonanie
spm
sei  ; odblokowanie układu przerwań
```

List. 2. Program ładowania danych do bufora tymczasowego

```
cz_ee:  ;sprawdzanie czy zapis eeprom sie zakończył
sbic EECR, EEWE
rjmp cz_ee
wr: pętla w której pobieramy dane z ram i zapisujemy do buforu tymczasowego
ld R0, Y+ ;pobieranie danych z ram'u
ld R1, Y+
ldi R20, (1<<SPMEN)  ; ustawianie maski odpowiednich bitów aby instrukcja SPM wy-
konała zapis danej do buforu
cz_spm:  ;sprawdzanie czy ostatnia instrukcja SPM zakończyła działanie
in R17, SPMCR
sbrc R17, SPMEN
rjmp cz_spm
cli  ;zablokowanie układu przerwań
out SPMCR,R20  ;ładowanie ustawień instrukcji SPM i jej wykonanie
SPM
sei  ; odblokowanie układu przerwań
adwi ZH:ZL, 2  ;zwiększenie adresu słowa
sbiw R25:R24, 2 zmniejszenie ilości słów w stronie
brne wr  ; jeśli strona nie jest zapełniona cała to skacze do wr
```

List. 3. Program zapisujący stronę pamięci Flash

```
mov ZL, R13  ;Załadowanie LBS adresu strony
mov ZH, R14  ;Załadowanie MBS adresu strony
ldi R20, (1<<PGWRT) | (1<<SPMEN)  ;ustawianie maski odpowiednich bitów aby in-
strukcja SPM wykonała zapis strony
 cz_spm:  ;sprawdzanie czy ostatnia instrukcja SPM zakończyła działanie
in R17, SPMCR
sbrc R17, SPMEN
rjmp cz_spm
cli  ;zablokowanie układu przerwań
out SPMCR,R20  ;ładowanie ustawień instrukcji SPM i jej wykonanie
SPM
sei  ; odblokowanie układu przerwań
```

Wypełnianie bufora

Kolejnym krokiem, jaki trzeba podjąć przed zapisaniem Flasha jest wypełnienie bufora, który ma rozmiar jednej strony i jest on odseparowanym (nie SRAM) buforem tylko do zapisu. Bufor wypełniamy słowo po słowie. Rejestru Z używamy do zaadresowania słowa w buforze. Zapisywanie słowa w buforze odbywa się następująco: zapisujemy słowo do rejestrów R1:R0, zapisujemy adres w rejestrze Z, ustawiamy bity SPMEN w rejestrze SPMCR, wykonujemy instrukcję SPM w najbliższych czterech cyklach maszynowych. Pamiętajmy o wyłączeniu przerwań. Nim zaczniemy operację ładowania bufora, musimy sprawdzić czy operacja zapisu pamięci EEPROM została zakończona, gdyż podczas zapisu EEPROM–a zostaje niszczona zawartość bufora tymczasowego. Robimy to przez sprawdzenie bitu EEWE w rejestrze EECR (jeśli jest wyzerowany, to znaczy że procesy zapisu się skończyły). Warto o tym pamiętać, jeśli gdzieś w programie (zarówno aplikacji, jak i *bootloadera*) zapisujemy dane do pamięci EEPROM. Nadszedł czas na przykładowy program (**list. 2**). Będzie on ładował do bufora tymczasowego dane umieszczone w pamięci RAM. Adres komórki RAM–u, w której zaczynają się dane znajduje się w rejestrze Y, natomiast adres słowa znajduje się w rejestrze Z. Rejestry R24:R25 przechowują wielkość strony (która jest różna dla różnych typów procesora).

Programowanie

Po poprawnym wykonaniu wcześniej opisanych czynności możemy w końcu przystąpić do upragnionego zapisu pamięci. Skoro nowe dane są już w buforze, to czas je zapisać. Najpierw adresujemy, którą stronę chcemy zapisać. Robimy to identycznie jak w przypadku kasowania strony, następnie ustawiamy bity PGWRT i SPMEN w rejestrze SPMCR i w kolejnych czterech cy-

Tab. 1. Znaczenie bitów Boot Lock Bits

BLB0 Mode	BLB02	BLB01	Opis
1	1	1	Pełny dostęp zapis/odczyt, brak zabezpieczeń
2	1	0	Instrukcja SPM nie może zapisywać sekcji aplikacji
3	0	0	Instrukcja SPM nie może zapisywać sekcji aplikacji, instrukcja LPM wywoływana z sekcji BLS nie może odczytywać sekcji aplikacji. Wektory przerwań umieszczone w BLS są zablokowane podczas wykonywania programu.
4	0	1	Instrukcja LPM wywołana z BLS nie może odczytać sekcji aplikacji. Wektory przerwań umieszczone w BLS są zablokowane podczas wykonywania programu.
BLB0 Mode	**BLB12**	**BLB11**	**Opis**
1	1	1	Pełny dostęp zapis/odczyt, brak zabezpieczeń
2	1	0	Instrukcja SPM nie może zapisywać sekcji BLS
3	0	0	Instrukcja SPM nie może zapisywać BLS, instrukcja LPM wywoływana z sekcji aplikacji nie może odczytywać BLS. Wektory przerwań umieszczone w sekcji aplikacji są zablokowane podczas wykonywania programu bootloadera.
4	0	1	Instrukcja LPM wywołana z sekcji aplikacji nie może odczytać BLS. Wektory przerwań umieszczone w sekcji aplikacji są zablokowane podczas wykonywania programu bootloadera.

klach zegarowych wykonujemy instrukcję SPM. Bit SPMEN zostanie automatycznie wyzerowany po zakończeniu zapisu. Dopóki to się nie stanie, nie wolno rozpocząć zapisu następnej strony. Zakończenie programowania może również sygnalizować przerwanie. Znamy już teorię, możemy teraz napisać przykładowy program, który będzie programował stronę pamięci Flash (**list. 3**). Zakładamy że bufor tymczasowy jest już załadowany, a strona jest już wykasowana. W rejestrach R13:R14 znajduję sie adres strony do zapisania.

Bity zabezpieczające

Kolejną fantastyczną sprawą związaną z *bootloadrem* są *Boot Lock Bits*. Pozwalają one zabezpieczyć pamięć w wybrany przez nas sposób (**tab. 1**). Mamy do wyboru cztery poziomy zabezpieczenia, dla każdej sekcji. Bezpieczniki BLB0 zabezpieczają sekcję aplikacji, a BLB1 sekcję *bootloadera*.

Po przemyśleniu jaki poziom zabezpieczeń nas interesuje przystępujemy do zaprogramowania bitów. Pamiętajmy, że raz ustawiony bezpiecznik można wyczyścić tylko podczas programowania szeregowego lub równoległego. Programowanie zaczynamy od załadowania do rejestru R0 odpowiedniej maski bitów. Robimy to według następującego wzorca:

Bit	7	6	5	4
R0	1	1	BLB12	BLB11

3	2	1	0
BLB02	BLB01	1	1

Jeśli rejestr jest już odpowiednio załadowany, ustawiamy bit BLBSET i SPMEN w rejestrze SPMCR, następnie wykonujemy instrukcję SPM. Wartość rejestru Z jest ignorowana, ale projektanci Atmela zalecają, by załadować do niego wartość 0x0001. Pozostało nam tylko napisać odpowiedni program, który będzie zapisywał bity zabezpieczające (**list.4**).

Odczyt bitów sterujących i zabezpieczających

Oprócz możliwości ustawiania bitów zabezpieczający, *bootloader* ma możliwość odczytu bitów zarówno zabezpieczających, jak i konfiguracyjnych. Odczyt odbywa się bardzo podobnie jak zapis. Do odczytu używamy instrukcji LPM, która odczytuje różne bajty w zależności od adresu podanego w rejestrze Z. Zaczynamy więc od załadowania rejestru Z odpowiednim adresem (**tab. 2**). Teraz ustawiamy bity BLBSET i SPMEN w rejestrze SPMCR. Następnie podczas najbliższych 3 cykli musimy wydać polecenie LPM, które spowoduje odczytanie bajtu wskazanego przez rejestr Z do wybranego przez nas rejestru.

Tab. 2. Znaczenie adresu zawartego w rejestrze Z

Rejestr Z	Odczytane bity
0x0000	Bity konfiguracyjne LSB
0x0001	Bity zabezpieczające
0x0002	Bity konfiguracyjne MSB
0x0003	Bity konfiguracyjne EMSB

W tab. 2 nie przedstawiłem znaczenia konkretnych bitów, ponieważ jest ono różne dla każdego modelu mikrokontrolera (oprócz bitów zabezpieczających). Znaczenia bitów w wybranym przez nas modelu szukamy oczywiście w dokumentacji, na początku rozdziału *Memory Programming*. Wszelkie wątpliwości powinien wyjaśnić kolejny program (list. 5), który odczyta bity zabezpieczające do rejestru R15.

Podsumowanie

Artykuł ten miał na celu przybliżenie teoretycznej części samoprogramowania mikrokontrolerów z rodziny AVR. Myślę że informacje zawarte w tekście pozwolą na samodzielne napisanie programu *bootloadera* pod warunkiem, że ma się pewne doświadczenie w programowaniu AVR. Wszelkie wątpliwości jak zwykle są wyjaśnione w dokumentacji dostarczanej przez Atmela, w tym Application note (AVR109).

Paweł Kłaja

Literatura:

– *Dokumentacje ATmega*
– *Application note AVR109*
– *strona http://www.avrfreaks.net*
– *„Mikrokontrolery AVR ATmega w praktyce” Rafał Baranowski*

List. 4. Program zapisywania bitów zabezpieczających

```
ldi ZL,0x01 ;załadowanie adresu 0x0001
ldi ZH,0x00
ldi R0, 0b11111011 ;załadowanie maski bitów która ustawi bit BLB01
ldi R20, (1<<BLBSET)|(1<<SPMEN) ;ustawianie maski odpowiednich bitów żeby zapisać
bity
cli  ;zablokowanie układu przerwań
out SPMCR, R20  ;ustawienie bitów BLBSET i SPMEN w rejestrze SPMCR
spm ;zapis bitów
sei  ; odblokowanie układu przerwań
```

List. 5. Program odczytywania bitów zabezpieczających

```
ldi ZL,0x01 ;załadowanie adresu 0x0001 aby odczytać bity zabezpieczające
ldi ZH,0x00
ldi R20, (1<<BLBSET)|(1<<SPMEN) ;ustawianie maski odpowiednich bitów żeby odczytać
bity
cli  ;zablokowanie układu przerwań
out SPMCR, R20  ;ustawienie bitów BLBSET i SPMEN w rejestrze SPMCR
lpm R15, Z  ;odczytanie bitów zabezpieczających do rejestru R15
sei  ; odblokowanie układu przerwań
```