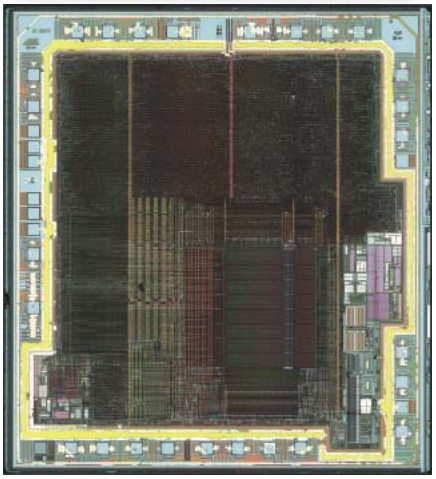
Samoprogramowanie AVR (2)

Opis bootloadera zgodnego z AVR109

Dodatkowe materiały na CD i FTP:
host: ep.com.pl, user: 12235, pass: 60u61csy
• pierwsza część artykułu



Kontynuujemy cykl artykułów poświęconych bootloaderom do mikrokontrolerów AVR. W artykule opiszemy bootloader udostępniany przez firmę Atmel w nocie aplikacyjnej AVR109. Umożliwia on samoprogramowanie różnych mikrokontrolerów z rodziny ATmega. Podany niżej użyteczny przykład będzie dotyczył ATmega8.

W tej części artykułu weźmiemy na warsztat darmowy bootloader udostępniony przez Atmela. Do kompilacji i uruchomienia będzie potrzebne AVR Studio, które można pobrać ze strony *http://www.atmel.com* oraz nota aplikacyjna AVR109 (*http://www.atmel.com/dyn/products/app_notes.asp?family_id=607*). W nocie oprócz kodu źródłowego znajduje się przydatny arkusz kalkulacyjny. Można też ściągnąć program *AVR-OSP II* (*http://www.esnips.com/web/AtmelAVR*), dzięki któremu będziemy mogli programować także nowsze mikrokontrolery.

Po pobraniu niezbędnych plików należy zainstalować programy oraz rozpakować kod źródłowy. Opisaną w tym artykule przykładową konfigurację oraz kompilację bootloadera wykonano dla mikrokontrolera ATMega8.

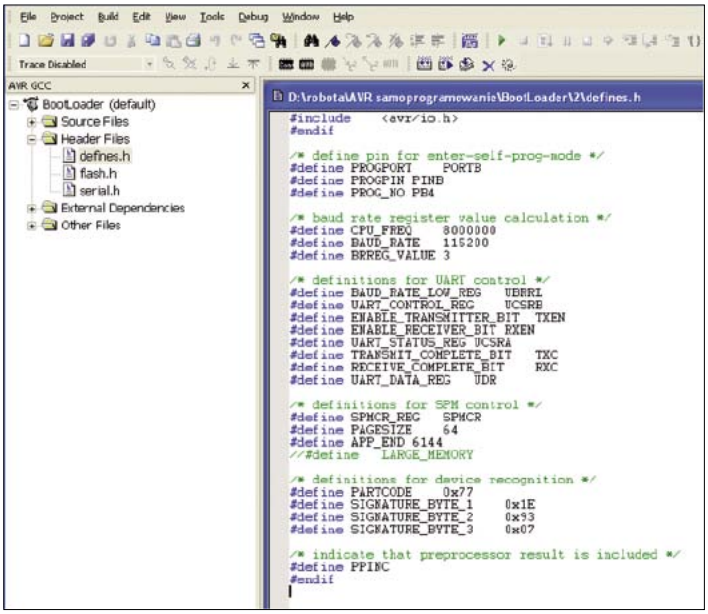
	A	B	C	D	E	F	G	H
1	Replace all code segment definitions in the linker file with the following line:							
2	-Z(CODE)INTVEC,FAR_F,SWITCH,CODE=1800-1FFF							
3								
4	Create a file (defines.h) containing the following:							
5	/* definitions generated by preprocessor, copy into defines.h */							
6	#ifndef	PPINC						
7	#define	_ATMEGA8	// device select: _ATMEGAxxxx					
8	#define	_B1024	// boot size select: _Bxxxx (words), powers of two only					
9	#ifdef	__ICCAVR__						
10	#include	"iom8.h"						
11	#endif							
12	#if	__GNUC__						
13	#include	<avr/io.h>						
14	#endif							
15								
16	/* define pin for enter-self-prog-mode */							
17	#define	PROGPORT	PORTB					
18	#define	PROGPIN	PINB					
19	#define	PROG_NO	PB4					
20								
21	/* baud rate register value calculation */							
22	#define	CPU_FREQ	8000000					
23	#define	BAUD_RATE	115200					
24	#define	BRREG_VALUE	3					
25								
26	/* definitions for UART control */							
27	#define	BAUD_RATE_LOW_REG	UBRRL					
28	#define	UART_CONTROL_REG	UCSRB					
29	#define	ENABLE_TRANSMITTER_BIT	TXEN					
30	#define	ENABLE_RECEIVER_BIT	RXEN					
31	#define	UART_STATUS_REG	UCSRA					
32	#define	TRANSMIT_COMPLETE_BIT	TXC					
33	#define	RECEIVE_COMPLETE_BIT	RXC					
34	#define	UART_DATA_REG	UDR					
35								
36	/* definitions for SPM control */							
37	#define	SPMCR_REG	SPMCR					
38	#define	PAGESIZE	64					
39	#define	APP_END	6144					
40	//#define	LARGE_MEMORY						
41								
42	/* definitions for device recognition */							
43	#define	PARTCODE	0x77					
44	#define	SIGNATURE_BYTE_1	0x1E					
45	#define	SIGNATURE_BYTE_2	0x93					
46	#define	SIGNATURE_BYTE_3	0x07					
47								
48	/* indicate that preprocessor result is included */							
49	#define	PPINC						
50	#endif							
51								

Rys. 6.

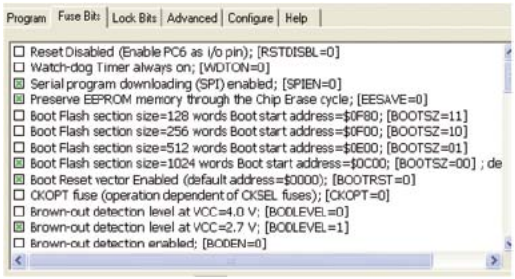
Konfiguracja bootloadera

Na początku należy otworzyć kod źródłowy w AVR Studio. Po otwarciu, w bocznym panelu w folderze *Header Files* znajdziemy plik *defines.h*. Należy go otworzyć do edycji (**rys. 7**). Jest to plik konfiguracyjny, w którym ustalamy typ procesora, rozmiar sekcji bootloadera, definiujemy porty, do którego podłączono przycisk włączający bootloader, podajemy częstotliwość taktowania mikrokontrolera oraz prędkość komunikacji przez port szeregowy (UART). W tym pliku również przypisujemy odpowiednim rejestrom nazwy. Wynika to z faktu, że wraz z pojawianiem się nowych modeli AVR-ów zmieniały się nazwy niektórych rejestrów (oczywiście nie tylko nazwy, ale to już wykracza poza ramy tego artykułu). Na szczęście nie trzeba tego robić ręcznie, skrypt w Excelu sam wygeneruje odpowiednie nazwy i dodatkowo sprawdzi, czy bootloader zmieści się w pamięci.

Arkusz kalkulacyjny wykonujący wspomniane obliczenia zapisano w pliku *preprocessor.xls*. Aby go otworzyć, należy oczywiście dysponować komputerem z zainstalowanym programem Excel. Arkusz po otwarciu ustawia się na pierwszej zakładce, w której jest umieszczona instrukcja posługiwania się kalkulatorem. W drugiej zakładce o nazwie *defines_h* (**rys. 6**) automatycznie jest generowany potrzebny kod. Należy pamiętać, aby modyfikować tylko żółte pola! W pierwszym polu ustawia się typ mikrokontrolera, następnie wielkość sekcji bootloadera (w słowach!). W kolejnych trzech polach należy wpisać oznaczenie portu, rejestru pin oraz numer pinu, dzięki któremu bootloader będzie się włączał. Chodzi o to, że po włączeniu zasilania lub zerowaniu



Rys. 7.



Rys. 8.

program sprawdza, czy na tym wejściu mikrokontrolera jest stan „0”, jeśli tak, to włącza boot loader, jeśli nie, to skacze do sekcji aplikacji. W kolejnym polu można ustalić wartość kwarcu jakim jest taktowany mikrokontroler, wartość podajemy w hercach, należy zdefiniować, z jaką prędkością ma działać port szeregowy (ale jeśli ustawimy inną niż 115.200, to bootloader nie będzie już mógł współpracować z AVR Studio, pozostanie „tylko” Avr-Osp II). Po ustawieniu tych wszystkich opcji należy skopiować wszystko od wiersza 5 do 50, a następnie wkleić do pliku *defines.h* zamiast danych, które tam są. Dalszej konfiguracji możemy dokonać w pliku *main.c* (boczny panel w „folderze” Source Files), programiści zaoferowali nam „regulacje” funkcjonalności bootloadera po linijce /* *Uncomment the following to save code space* */ można wyłączyć niektóre możliwości programu, np. możliwość programowania pamięci EEPROM. Dzięki temu możemy zmniejszyć rozmiar programu i „zmieścić” się w sekcji o rozmiarze 512 słów zamiast 1024 (bo tyle potrzeba na bootloader z pełną funkcjonalnością). Przed kompilacją trzeba jeszcze zakomunikować linkierowi, aby program nie znajdował się od adresu 0, lecz od początku sekcji bootloadera. Robi się to w następujący sposób: uruchamiamy z menu *Project–>Configuration Options*. Ukaże się okienko, w którym należy wskazać typ procesora oraz w zakładce *Custom Options* dla linkiera należy dodać następującą regułę – ***Ttext=0x1800***, gdzie 0x1800 to adres początku sekcji bootloadera, np. można go wziąć z pliku, w którym przygotowywało się konfiguracje (*preprocessor.xls*), na górze można znaleźć następującą linikę: *Z(CODE) INTVEC, FAR_F,SWITCH, CODE=1800-1FFF.* To właśnie z niej można wziąć adres. Pod dokonaniu tych zmian można już skompilować program.

Wgranie bootloadera

Podłączamy procesor do równoległego bądź szeregowego programatora. Nim wgramy program, trzeba jeszcze ustalić odpowiednie bity konfiguracyjne, mianowicie chodzi o ustawienie odpowiedniej wielkości sekcji bootloadera oraz ustawienie bitu Bootrst (Boot Reset Vector). Konfigurację Fuse Bits dla naszego przykładu możemy zobaczyć na **rys. 8**. Po dokonaniu tych rzeczy zostaje już tylko zaprogramowanie pamięci Flash.

Zasada działania

Teraz możemy już odłączyć programator i wypróbować samoprogramowanie. Po podłączeniu układu do portu szeregowego należy go wyzerować, podając jednocześnie „0” na zadeklarowaną wcześniej nóżkę mikrokontrolera. Teraz trzeba uruchomić program *Avr-Osp II* lub z Avr Studio program *AVR prog* (menu *Tools–>AVR prog*). Programy same wykrywają, jaki układ jest podłączony. Oba programy są bardzo intuicyjne, dlatego nie będę opisywał ich obsługi. Wszystko sprowadza się do otwarcia pliku *hex* i naciśnięcia przycisku *Write* lub *Program*. Programy weryfikują zapisaną pamięć, dlatego mamy pewność, że pamięć Flash lub EEPROM została dobrze zaprogramowana.

Paweł Klaja, pklaja@o2.pl