AVR-GCC

Środowisko programistyczne dla mikrokontrolerów AVR

Wracamy do tematu środowiska AVR Studio z bezpłatnym kompilatorem C dla mikrokontrolerów AVR - GCC-AVR. W artykule przedstawimy instalację i konfigurację tego oprogramowania do pisania, kompilacji i uruchamiania programów w C pod system Windows.

Z pewnością większość elektroników zauważyła, że ostatnio nastała moda na stosowanie mikrokontrolerów jednoukładowych nawet w bardzo prostych układach. Szczególnym zainteresowaniem cieszy się rodzina AVR firmy Atmel. Są to dość nowoczesne, szybkie i bogato wyposażone mikrokontrolery. Aby ułatwić użytkownikowi uruchamianie układów, producent wprowadził możliwość programowania układów już po zamontowaniu w systemie docelowym. Bardzo interesujące cechy tych mikrokontrolerów są związane z ich architekturą i listą rozkazów stworzoną z myślą o stosowaniu kompilatorów języka C.

Nie chcę w tym miejscu wywoływać polemiki między osobami uznającymi za jedynie „słuszny" język asembler i ich przeciwnikami. Jak zwykle prawda leży pośrodku. Asembler pozwala ściśle kontrolować zasoby i przebieg programu, a języki wysokiego poziomu - skrócić czas przygotowywania programu, który jednocześnie może być bardziej czytelny.

Atmel udostępnia nieodpłatnie zintegrowane środowisko programistyczne dla AVR-ów - oprogramowanie AVR Studio. Posiada ono edytor (z rozpoznawaniem składni), asembler, symulator umożliwiający śledzenie wykonywania programów z podglądem wszystkich wbudowanych w mikrokontroler układów. Da się również tworzyć, kompilować i uruchamiać programy napisane w języku C. Ta ostatnia cecha daje duże możliwości, ale również sprawia najwięcej kłopotów w poprawnym skonfigurowaniu środowiska i dopasowaniu do niego kompilatora.

Rys. 1

Dobrym uzupełnieniem dla AVR Studio jest bezpłatny kompilator języka C - AVR-GCC. Pierwotnie powstał on z myślą o systemach typu Unix, ale po pewnym czasie przeniesiono go również na platformę Windows.

GCC to skrót od *GNU Compiler Collection*, a GNU to skrót od *GNU's Not Unix*. Więcej informacji o GCC i projekcie GNU można znaleźć na stronie głównej GCC (*http://gcc.gnu.org/*).

Instalacja kompilatora

Przed instalacją należy zaopatrzyć się w pakiet instalacyjny AVR-GCC w wersji 3.2 (opatrzonej datą 2002-06-25) dostępny w postaci jednego pliku do ściągnięcia ze strony AVR Freaks (*http://www.avrfreaks.net*) i AVR Studio (aktualnie wersja 3.55) udostępnione przez Atmela na stronie internetowej firmy.

Instalacja AVR-GCC sprowadza się do uruchomienia instalatora i postępowania według wskazówek. Należy pozwolić programowi instalacyjnemu założyć domyślny katalog. Podczas instalacji nie powinny się pojawić żadne błędy. Na koniec zostaniemy poinformowani o poprawnym zakończeniu instalacji.

Utworzony zostaje katalog *c:\avr-gcc*, a w nim kilka podkatalogów. Najważniejsze z nich to:

- *avr\include* - zawierający pliki nagłówkowe (.h),
- *avrfreaks* z plikiem *avr_make* (globalna część *makefile*), *makefile* (dla projektu) oraz plikiem wsadowym uruchamiającym kompilację *gcc_cmp.bat*,
- *bin* - w którym są zawarte programy wykonywalne pakietu.

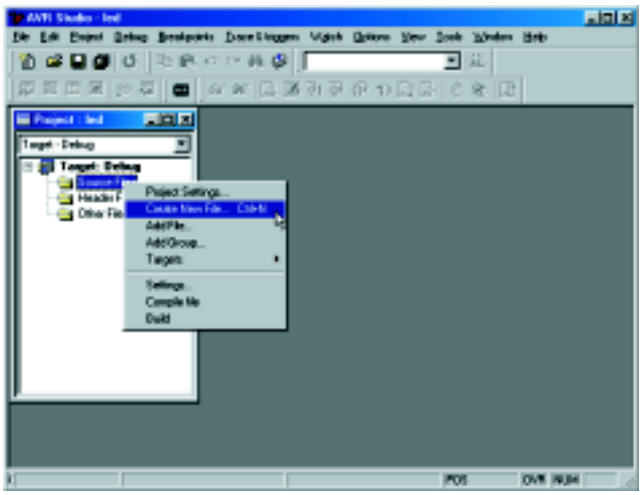
AVR Studio również instaluje się bardzo łatwo, przy czym domyślnym katalogiem jest *c:\Program Files\Atmel\AVR Studio*. W podkatalogu *appnotes* znajdują się przykładowe programy napisane w asemblerze i pliki definicyjne (*.inc*).

Konfiguracja i tworzenie projektu

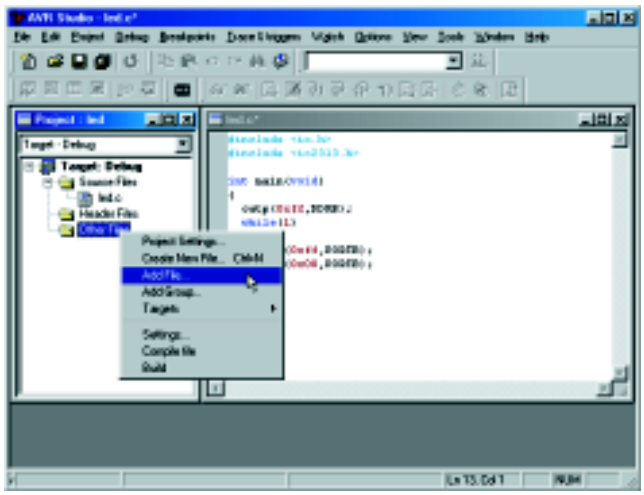
Opis konfiguracji przedstawię na przykładzie krótkiego programu. Po uruchomieniu AVR Studio, z menu *Project* wybieramy polecenie *New...*. Pojawi się okno *Select new project* (**rys. 1**). W polu *Project name* wpisujemy nazwę projektu *led*. Pole *Location* określa ścieżkę dostępu do tego projektu, przyjmijmy, że będzie to katalog *c:\avr\pierwszy*. Na koniec ważne jest, aby podświetlić w polu *Project type* pozycję *Generic 3rd Party C Compiler*.

Podczas kliknięcia prawym klawiszem myszy w oknie projektu otwiera się menu podręczne, z którego wybieramy *Create New File...* (**rys. 2**). W wywołanym oknie wpisujemy nazwę tworzonego pliku - *led.c* i ścieżkę dostępu do katalogu, w którym utworzyliśmy projekt. Aby zachować porządek, przeciągamy w oknie projektu ikonę z nazwą nowego pliku do folderu *Source Files*. W otwartym oknie zatytułowanym *led.c* wpisujemy krótki program w języku C (**list. 1**).

Spowoduje on, że mikrokontroler będzie wystawiał na wyjściach portu B na zmianę poziomy wysokie i niskie. Ważna uwaga: ostatnia linia każdego pliku przetwarzanego przez kompilator powinna być zakończona znakiem końca linii!



Rys. 2



Rys. 3

Mając już gotowy program, musimy go skompilować przy użyciu AVR-GCC. Pakiet kompilatora po naszej instalacji jest już w zasadzie przygotowany. Trzeba jedynie utworzyć plik *makefile* w katalogu projektu. Jest to plik tekstowy określający przebieg przetwarzania, kompilacji i konsolidacji. Dla ułatwienia, w pakiecie przygotowanym przez AVR Freaks plik ten zawiera parametry konfiguracyjne, które należy zmodyfikować dla konkretnego projektu. W końcowych liniach sterowanie jest przekazywane plikowi *avr_make* znajdującemu się w katalogu *c:\avrgcc\avrfreaks*, który zawiera wspólne dla wszystkich projektów polecenia i nie jest potrzebna jego modyfikacja.

Przykładowy plik *makefile* znajdziemy w wyżej wymienionym katalogu. Należy go skopiować do katalogu z projektem - w naszym przypadku do *c:\avr\pierwszy*. Na **list. 2** przedstawiam go w wersji w języku polskim, przystosowanej do prezentowanego przykładu.

Najprościej jest go modyfikować poprzez dodanie do projektu w AVR Studio. W tym celu klikamy prawym klawiszem w oknie projektu, z menu wybieramy *Add File* (**rys. 3**) i otwieramy plik *makefile*. Przeciągnijmy go myszką do folderu *Other Files*. Najważniejsze parametry, jakie musimy przede wszystkim określić to:

- MCU, czyli typ mikrokontrolera,
- TRG - nazwa pliku wynikowego, przy czym musi to być ta sama nazwa jak nazwa projektu i programu głównego,
- SRC - lista plików źródłowych do kompilacji,
- zależności obiektów wynikowych od innych plików (tutaj *led.o* jest zależny od *led.c*)

Zapis *$(TRG)* oznacza wywołanie zmiennej (makra) określonej wcześniej poprzez przyporządkowanie *TRG=*.

Teraz trzeba poinstruować AVR Studio, jak ma współpracować z kompilatorem GCC.

Upewnijmy się, że w oknie projektu jest wybrana zakładka *Target-Debug*. Wskazując myszką na rdzeń drzewa *Target-Debug*, klikamy prawym klawiszem i z menu wybieramy pozycję *Settings...* Pojawi się okno, które należy wypełnić tak, jak to pokazano na **rys. 4**. Ustawienia te powodują podjęcie odpowiednich akcji przy kompilowaniu i debuggowaniu programu. AVR Studio uruchamia program wsadowy (*gcc_cmp.bat*) przeprowadzający kompilację, a następnie, jeśli nie zasygnalizowano żadnych błędów, ładuje plik debuggera o nazwie takiej, jak nazwa projektu z rozszerzeniem *.cof*.

Przy okazji warto utworzyć dodatkową zakładkę *Clean*, która będzie służyła do usuwania plików tworzonych przez kompilator. W tym celu klikamy prawym klawiszem na oknie projektu i wybieramy *Targets>Add*. W polu *Name* wpisujemy *Clean*, a poniżej wybieramy *Copy settings from>Debug*. Wybieramy zakładkę *Target-Clean* i podobnie jak wyżej modyfikujemy ustawienia. Linia polecenia ma mieć postać *c:\avrgcc\avrfreaks\gcc_cmp.bat clean*, co oznacza, że program będzie wywoływany z parametrem *clean*. Oba okienka z grupy *Run Stage Settings* pozostawiamy puste.

W systemie Windows 2000 trzeba wywoływać program wsadowy z podkatalogu *win2000*, czyli *c:\avrgcc\avrfreaks\win2000\gcc_cmp.bat*. Jeśli w oknie wyjściowym AVR Studio pojawia się komunikat błędu, informujący o kłopocie z odnalezieniem pliku *c:\tmpout.txt*, należy w pliku *gcc_cmp.bat* podać pełną ścieżkę dostępu do pliku *gcc_cmp2.bat* (*@start*

List. 1.

```
#include <io.h>
#include <io2313.h>
int main(void)
{
 outp(0xff,DDRB);
 while(1)
 {
   outp(0xff,PORTB);
   outp(0x00,PORTB);
 }
}
```

List. 2.

```
# Simple Makefile Volker Oth (c) 1999
# edited by AVRfreaks.net nov.2001
# tłumaczenie: Michał Lankosz sierpień 2002
###### zmień poniższe parametry dla swojego projektu ##########
#wpisz typ mikrokontrolera (np. at90s8535, attiny22, atmega128 itd.)
MCU = at90s2313
#wpisz nazwę pliku docelowego (bez rozszerzenia!)
TRG = led
#wpisz nazwy plików źródłowych w C rozdzielając spacją
SRC = $(TRG).c
#wpisz dodatkowe pliki źródłowe w asemblerze
ASRC =
#dodatkowe biblioteki i pliki obiektowe do dołączenia (zlinkowania)
LIB =
#dodatkowe pliki include do kompilacji
INC =
#flagi dla asemblera
ASFLAGS = -Wa, -gstabs
#flagi dla kompilatora
CPFLAGS  = -g  -O3 -Wall -Wstrict-prototypes -Wa,-ahlms=$(<:.c=.lst)
#flagi dla linkera
LDFLAGS = -Wl,-Map=$(TRG).map,-cref
############ tej linijki nie powinieneś modyfikować ##############
include $(AVR)/avrfreaks/avr_make
# zależności
$(TRG).o: $(TRG).c
```

/MIN /wait cmd /c c:\avrgcc\avrfreaks\win2000\gcc_cmp2.bat %1).

Na tym etapie mamy już wszystko ustawione. W oknie projektu wybieramy zakładkę *Target-Debug* i naciskamy klawisz F7 (menu *Project>Build*), aby skompilować program. Komunikaty kompilatora są wyświetlane w oknie zatytułowanym *Project Output*. W katalogu projektu wygenerowany zostanie między innymi plik *led.hex*. Jest to gotowy plik w formacie Intel Hex, którym programuje się mikrokontroler, na przykład za pomocą programatora YAAP (EP7/2002) lub Ponyprog (*www.lancos.com*). Ten drugi szczególnie polecam ze względu na dużą liczbę obsługiwanych układów.

Cały projekt ze wszystkimi zmodyfikowanymi plikami najprościej jest zapisać poleceniem *Save All* (ikona z kilkoma dyskietkami).

Uruchamianie i debugging programu

Dość ważne podczas uruchamiania nowego kodu jest analizowanie działania niektórych jego fragmentów. W szczególności przydatna jest praca krokowa i wgląd w zawartość poszczególnych zmiennych. Środowisko AVR Studio jest przygotowane do śledzenia działania zarówno programów napisanych w asemblerze, jak i w języku C. Konieczne było jednak odpowiednie przystosowanie kompilatora. Głównym problemem jest to, że kompilator GCC normalnie generuje plik debuggera w formacie ELF, którego nie akceptuje AVR Studio. Najnowszy pakiet kompilatora zawiera już zewnętrzny program konwersji formatu ELF na COFF (*elfcof.exe*), który jest automatycznie uruchamiany według polecenia zawartego w pliku *avr_make*.

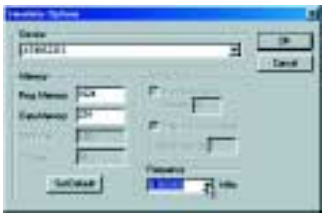
Z doświadczenia wiem, że konwersja formatów jest błędna lub AVR Studio zawiera błędy. Objawiają się one tym, że nie da się podglądać niektórych zmiennych i czasami nie działają pułapki (*breakpointy*). Dotyczy to szczególnie bardziej złożonych programów. Z tego względu dobrze jest, w celu analizy danego fragmentu programu, utworzyć pomocniczy projekt zawierający tylko istotne dla jego działania instrukcje.

Prześledzimy teraz skompilowany program *led*. Zauważmy, że po pierwszej kompilacji stały się aktywne niektóre przyciski na górnej listwie AVR Studio zawierającej zestaw poleceń do sterowania przebiegiem programu. Oznacza to, że został odnaleziony plik dla symulacji *led.cof*. Automatycznie również otworzyło się okno *Simulator Options* (**rys. 5**), w którym można wybrać typ mikrokontrolera i częstotliwość jego taktowania. Opcje te są dostępne z menu *Options>Simulator Options*.

Symulację rozpoczynamy klawiszem F5 (Run) lub wybierając ikonę 9. W oknie z programem źródłowym pojawia się żółta strzałka wskazująca linię kodu, która zostanie wykonana w kolejnym kroku. Aby wykonać linijkę wskazywaną strzałką, przyciskamy klawisz F11 (*Trace Into*, ikona 11).

To dopiero początek możliwości symulatora. Możemy bowiem „zobaczyć“ poziom, jaki występuje na wyprowadzeniach portu B. W tym celu klikamy na ikonę 6 (lub przyciskamy klawisze Alt+5) i rozwijamy gałąź PORTB. Wykonując kolejne kroki, obserwujemy zawartość rejestru portu wyjściowego (**rys. 6**).

Otwórzmy teraz okno procesora kombinacją klawiszy Alt+3 (lub ikoną 4). Znajdują się w nim przynajmniej dwie przydatne funkcje: stoper i licznik cykli zegarowych. Dzięki nim możemy łatwo zmierzyć czas wykonania danego fragmentu programu. W pętli *while* programu można na przykład zauważyć, że na wyjściu portu B poziom wysoki będzie się utrzymywał przez jeden



Rys. 5

cykl zegara, a niski przez trzy. Niesymetria wynika z budowy pętli. Przy tej okazji warto się również zapoznać z możliwością podglądu kodu maszynowego śledzonego programu.

Klikając na ikonce 21 (Ctrl+F11), otworzymy okno zawierające linijki tekstu w rodzaju:

```
+0000002C: BA18   OUT  0x18,R1
```

Powyższa linia zawiera informację, że pod adresem 2Ch znajduje się instrukcja o kodzie BA18h, której odpowiada zapis asemblerowy OUT 0x18,R1.

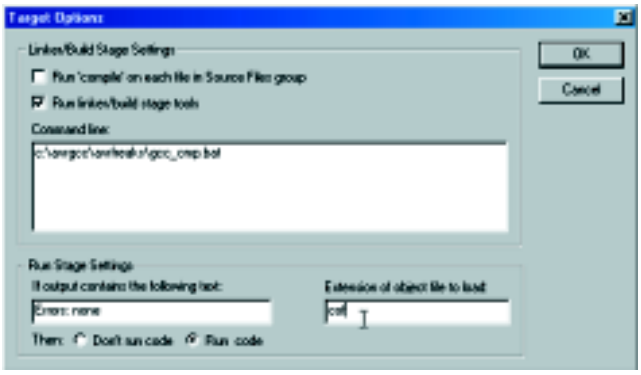
Uruchomienie pracy krokowej w tym trybie powoduje wykonywanie pojedynczych instrukcji asemblera. Aby powrócić do analizy kodu w C, ponownie używamy ikony 21.

Za pomocą ikony 19 (Shift+F5) zeruje się symulowany mikrokontroler, czyli doprowadza do jego stanu wyjściowego, jaki jest tuż po „resecie“ w rzeczywistym układzie.

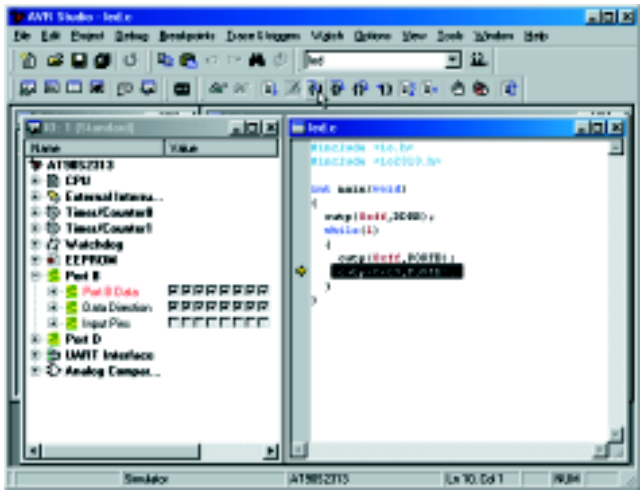
Podgląd zmiennych

Oprócz wewnętrznych rejestrów i pamięci procesora możemy również podglądać zmienne używane w programie. Dla przykładu zmodyfikujmy nasz program *led.c* do postaci pokazanej na **list. 3**.

Musimy jeszcze usunąć niepotrzebne pliki z poprzedniej kompilacji, zmieniając zakładkę w projekcie na *Target-Clean* i wciskając klawisz F7. Wracamy do *Target-Debug*, kompilujemy i uruchamiamy kombinacją kla-



Rys. 4

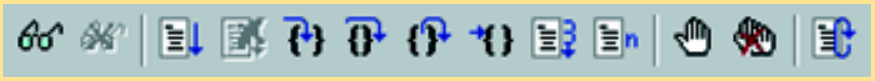


Rys. 6

Opis ikon menu AVR Studio

Ikony podglądu (*Views*):

1. Wartości zmiennych
2. Rejestrów
3. Pamięci
4. Istotnych parametrów pracy procesora
5. Informacji
6. Urządzeń wejścia/wyjścia



Ikony sterowania programem (*Debug*):

7. Dodanie zmiennej do podglądu
8. Usunięcie zmiennej z podglądu
9. Uruchomienie symulacji
10. Przerwanie symulacji
11. Krok z wejściem do funkcji
12. Krok z przeskokiem funkcji
13. Wyskoczenie z funkcji
14. Uruchomienie i zatrzymanie w miejscu kursora
15. Wykonanie n kroków
16. Wykonanie n kroków
17. Ustawienie lub usunięcie pułapki
18. Usunięcie wszystkich pułapek
19. Wyzerowanie procesora

Pozostałe

20. Kompilacja i uruchomienie programu

21. Przełącznik widoku źródła

wiszy Ctrl+F7. Klikamy na ikonę 7 (okularki) i podajemy nazwę interesującej nas zmiennej, czyli *x*. Rozpoczynamy pracę krokową. Instrukcja w pętli while powinna powodować przesuwanie o jeden bit w lewo bitów zmiennej *x* i zapisywanie tego bajtu do rejestru wyjściowego portu B. Rzeczywiście, widzimy efekt „węża" na bitach portu, podglądając stany jego wyjść w oknie IO. Jednak zmienna *x* ma przez cały czas wartość 0xFF. Taki stan rzeczy spowodował kompilator, który zoptymalizował kod. Kompilator „stwierdził", że nie ma potrzeby specjalnie czytać i zapisywać zmiennej *x* do pamięci RAM. Wystarczy operować na jednym z rejestrów, co skróci kod i jednocześnie przyspieszy działanie programu.

List. 3.

```
#include <io.h>
#include <io2313.h>
char x;
int main(void)
{
 outp(0xff,DDRB);
 x = 0xff;
 while(1)
   outp(x <<= 1,PORTB);
}
```

Dla celów uruchomieniowych można zmienić parametry kompilacji tak, aby kod nie był optymalizowany. Wystarczy w pliku *makefile* w linii zaczynającej się od wyrazu CPFLAGS= zamiast -O3 wpisać -O0. Po usunięciu plików poprzedniej kompilacji, ponownej kompilacji i uruchomieniu pracy krokowej, zmienna *x* jest już obserwowalna. Różnice można dostrzec, analizując kod asemblerowy tego fragmentu programu.

Gdy program jest złożony z kilku plików źródłowych i nagłówkowych, dodajemy ich nazwy do odpowiednich folderów w oknie projektu. W ten sposób mamy do nich szybki i łatwy dostęp. Należy pamiętać, aby w pliku *makefile* znalazła się lista plików do kompilacji i odpowiednie zależności.

Po przebrnięciu przez przedstawiony opis i uruchomieniu pierwszego programu, proponuję odwiedzić stronę AVR Freaks zawierającą mnóstwo przykładów, projektów i dokumentów pomocniczych.

Mam nadzieję, że niniejszy opis pozwoli pokonać trudy związane z przygotowaniem współpracy AVR Studio z kompilatorem AVR-GCC i zachęci Czytelników zainteresowanych programowaniem mikrokontrolerów do własnych eksperymentów.

Michał Lankosz
sq9fqq@sq9fqq.prv.pl
http://sq9fqq.prv.pl