Środowisko projektowe dla AVR-GCC

W artykule znajdziecie wskazówki, w jaki sposób samodzielnie tworzyć pomocnicze narzędzia programowe dla elektronika - amatora. Wprawdzie niemal do wszystkiego znajdziemy dzisiaj darmowe oprogramowanie w Internecie, ale nie zawsze spełnia ono nasze oczekiwania. Własny „wypiek“ pozwala na dużo lepszą realizację potrzebnych funkcji. Dostarcza przy tym wiele satysfakcji. Takie zajęcie tylko pozornie nie jest związane z czystą elektroniką. Komputer wraz z odpowiednimi programami jest już bowiem stałym elementem wyposażenia warsztatu elektronika - jak dawniej multimetr czy oscyloskop.

Przedstawione opisy dotyczą środowiska Windows oraz popularnego pakietu programistycznego Delphi. Oczywiście, jest to tylko jedna z wielu dostępnych obecnie platform, więc trudno mówić o jej wszechstronnej uniwersalności. Ale z tym problemem spotykamy się dzisiaj prawie w każdej dziedzinie.

O wyborze Delphi zdecydowało (obok szeregu niewątpliwych zalet) głównie udostępnienie przez firmę Borland bezpłatnej wersji pakietu. Jest to Delphi 6PE (*Personal Edition*), z którego możemy korzystać bez żadnych opłat po przeprowadzeniu rejestracji na stronie Borlanda. Wersja ta jest w pełni funkcjonalna, nie ma ograniczeń czasowych ani żadnych blokad uniemożliwiających normalną pracę (jak np. spotykane w różnych ewaluacyjnych edycjach ograniczenia długości kodu czy też zapisywania projektów na dysk itp.).

Oczywiście wyposażenie pakietu jest dużo skromniejsze niż wydań komercyjnych, na przykład:
- nie ma komponentów obsługi baz danych,
- brak kodów źródłowych biblioteki komponentów,
- wyłączone są niektóre ułatwienia edytora.

Zarazem obowiązuje bardzo istotne ograniczenie licencyjne: pakiet może być używany tylko do celów niekomercyjnych (tzn. nie może służyć do jakiejkolwiek działalności zarobkowej).

W naszym zastosowaniu żadne z powyższych ograniczeń nie będzie jednak stanowić poważniejszej przeszkody, ponieważ:
- pomocnicze programy warsztatowe na ogół zadowalają się gotowymi komponentami i nie ma potrzeby wnikania w ich kody (co jest zazwyczaj konieczne w przypadku modyfikacji lub dopisywania komponentów potomnych),
- raczej nie będziemy używać baz danych (a jeśli już, to w zakresie nie wymagającym stosowania wizualnych komponentów wspomagających),
- powstające programy są z założenia przeznaczone dla potrzeb własnych lub do niekomercyjnego rozpowszechniania na zasadzie *freeware*.

Delphi w wersji Personal Edition jest bezpłatną wersją pełnego pakietu komercyjnego. Możemy z niego korzystać bez żadnych opłat, wymagana jest jedynie rejestracja na internetowej stronie firmy Borland.

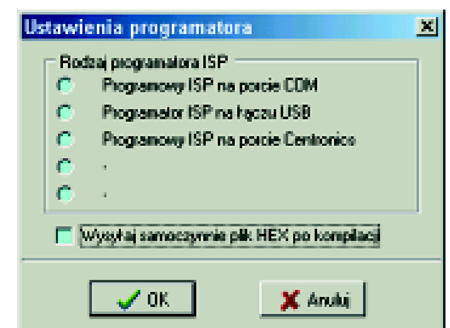
Przykładowy projekt

Jako przykładowy projekt spróbujmy opracować „okienkowe“ środowisko do obsługi bezpłatnego kompilatora C dla rodziny AVR. AVR-GCC jest znakomitym narzędziem o wszechstronnych możliwościach, ale pracuje wyłącznie w trybie tekstowym. Zmniejsza to w dużym stopniu wygodę jego użytkowania w porównaniu do standardowych, obecnie graficznych IDE (*Integrated Development Environment* - zintegrowane środowisko programistyczne).

Niedawno był opisywany w EP sposób używania AVR-GCC w takim środowisku - firmowym AVR Studio Atmela, które niestety nie do końca eliminuje pewne niedogodności. Szczególnie uciążliwy jest brak samoczynnej lokalizacji błędów w przygotowywanym kodzie źródłowym, konieczność edycji plików *makefile*, a także czyszczenia foldera przed każdą kompilacją (*clean*). Z kolei AVR Studio oferuje wspaniałe możliwości debugowania (z ustawianymi breakpointami i podglądem zmiennych). Nasze oprogramowanie będzie uzupełniające dla AVR Studio, pomocne zwłaszcza przy pierwszym - obfitującym w błędy składniowe - etapie przygotowania kodu źródłowego dla mikrokontrolera.

Czym powinno się charakteryzować nasze środowisko?

1. Edycja programu powinna być wspierana przez wszelkie możliwe udogodnienia, typowe dla współczesnych windowsowych edytorów: zaznaczanie, wycinanie, kopiowanie, cofanie operacji, wyszukiwanie i zamiana słów, obsługa schowka systemowego itd. Równie ważne jest wspomaganie typo-



Rys. 1. Okno obsługi sprzętowego programatora

Instalacja i licencjonowanie D6PE

Instalacja i rejestracja przebiega następująco:

- Na stronie Borlanda (*www.borland.com*) w sekcji *download* wybieramy Delphi 6 Personal. Obecnie ze strony można otrzymać tylko numer seryjny i klucz - sam plik instalacyjny trzeba zdobyć z innych źródeł (np. z CD załączanych do pism komputerowych). Wejście do sekcji *download* wymaga podania własnych danych łącznie z kontem e-mail. Przydzieloną nazwę użytkownika oraz hasło zapamiętujemy do dalszego wykorzystania.
- Wkrótce otrzymamy e-mail z numerem seryjnym (*Serial Number*) oraz kluczem upoważniającym (*Authorization Key*) i zapisujemy je.
- Uruchamiamy plik instalacyjny *BorlandDelphiPersonalEdition.exe*. Jest to samorozpakowujące się archiwum przygotowujące pakiet instalacyjny programu.
- W utworzonym folderze *BorlandDelphiPersonalInstaller* uruchamiamy właściwy instalator *install.exe*. Teraz konieczne będzie wprowadzenie otrzymanych kodów rejestracyjnych. Wybieramy też docelowy folder oraz typ instalacji - na ogół wystarczy pozostawić ustawienia domyślne (*Program Files\Borland\Delphi6*, instalacja typowa).
- Po zakończeniu instalacji konieczny jest restart systemu dla wprowadzenia wszystkich nowych ustawień.
- Dla uruchomienia Delphi musimy jeszcze zarejestrować zainstalowany pakiet - odpowiednie okno otwiera się samoczynnie przy pierwszym starcie. Można to wykonać *on line* albo wchodząc na stronę *http://register.borland.com* i wypełniając formularz dotychczasowymi danymi rejestracyjnymi (okienko Delphi podaje nam dodatkowo kod rejestracyjny) - otrzymamy wtedy e-mailem dodatkowy kod aktywacyjny - jego wprowadzenie ostatecznie kończy instalację.
- Teraz uruchamiamy środowisko Delphi i wstępnie sprawdzamy, czy wszystko działa prawidłowo, np. próbnie kompilując najprostszy projekt z pustym oknem.
- Pozostaje nam jeszcze instalacja łaty z poprawkami - pobieramy ze strony Borlanda (*http://www.borland.com/*) lub BSC (*http://www.bsc.com.pl*) ostatnią wersję (numer 2 - *D6_upd2_std.exe*) i uruchamiamy - koniecznie przy wyłączonym Delphi. Ponownie musimy wprowadzić dane rejestracyjne. Dodatkowo kopiujemy do folderu *Help* poprawki z pliku *d62per.zip*.

Teraz środowisko jest gotowe do używania. Nie zamierzam tutaj oczywiście prezentować tutoriala Delphi dla początkujących - ilość informacji na ten temat jest ogromna i każdy może w sieci lub w księgarni wybrać coś najbardziej odpowiedniego dla własnych potrzeb.

wo programistyczne, jak kolorowanie elementów języka (np. słów kluczowych) czy ustawianie zakładek (*bookmarków*) dla łatwej lokalizacji wybranych, istotnych w danej chwili fragmentów.

2. Łatwe zarządzanie plikami. Musimy mieć możliwość tworzenia projektów (czyli zespołów plików potrzebnych do utworzenia kodu wykonywalnego) i ich łatwej modyfikacji oraz zapisywania i odtwarzania. To samo dotyczy także poszczególnych plików projektu.

3. Uruchamianie dla plików zawartych w projekcie właściwego „silnika" wykonawczego, czyli AVR-GCC.

4. Łatwe wprowadzanie potrzebnych opcji i parametrów (które w trybie tekstowym ręcznie wpisujemy do pliku *makefile)*.

5. Przechwycenie i dogodne wyświetlenie oraz interpretacja wyników działania kompilatora, z możliwością automatycznego zlokalizowania stwierdzonych błędów w tekście kodu źródłowego.

6. Możliwość obsługi sprzętowego programatora, który załaduje uzyskany plik wykonywalny do pamięci Flash uruchamianego mikrokontrolera (**rys. 1**).

Powyższe wystarczy do przeprowadzenia pierwszych testów i oceny skuteczności i łatwości obsługi naszego IDE. Potem przyjdzie czas na dalsze udoskonalanie, ale na razie zrealizujmy założenia podstawowe.

Edytor tekstowy

Jako edytor zostanie wykorzystany znakomity, bezpłatny komponent *TSynEdit*. Jest on dostępny pod adresem *http://synedit.sourceforge.net*, w zestawie z demonstracyjnymi przykładami wykorzystania. Ściągnięty plik ZIP rozpakowujemy do folderu przeznaczonego na komponenty (np. *\Delphi6\Komponenty\SynEdit1_1*). Aktualnie dostępna wersja 1.1 posiada już pakiet przewidziany specjalnie dla D6PE i nie musimy wprowadzać żadnych poprawek (poprzednio koniecznych ze względu na brak w PE komponentów bazodanowych).

W Delphi zamykamy wszystkie projekty i otwieramy (*File>Open*) plik pakietu *SynEdit_D6_PE.dpk*. W opcjach musimy podać ścieżkę wyszukiwania, wskazującą na folder *SynEdit1_1\Source*, a następnie kompilujemy plik źródłowy i instalujemy komponenty.

TSynEdit dysponuje wszystkimi potrzebnymi możliwościami, jego istotną wadą jest jednak brak opisów i pomocy - dla zapoznania się z właściwościami i metodami trzeba sięgać do kodu źródłowego.

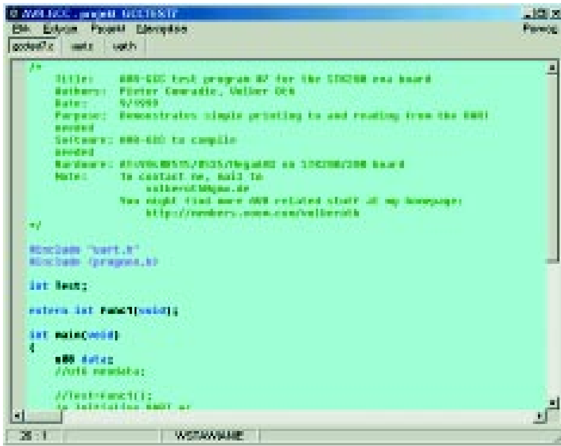
Następną ważną sprawą jest przestrzeganie warunków licencji MPL/GPL, ale w przypadku naszego otwartego programu nie grozi ich naruszenie.

TSynEdit współpracuje z serią komponentów kolorowania składni (*syntax highlighters*), opracowanych dla popularnych języków programowania (**rys. 2**). Ponieważ C dla mikrokontrolera ma własną specyfikę, nie wybieramy żadnego dedykowanego highlightera, ale uniwersalny *TSynGeneralSyn*, który pozwala na wprowadzanie dowolnej listy słów kluczowych. Dotyczy to także asemblera. Dużo bardziej eleganckim rozwiązaniem byłoby oczywiście dopisanie highlighterów specjalnie dla C i asm AVR, ale na razie pozostawiamy to na przyszłość.

Pakiet AVR-GCC i struktura katalogów

Na stronie *http://www.avrfreaks*.net jest dostępny gotowy instalator AVR-GCC dla platformy Windows. Po zainstalowaniu kompilatora dobrze jest sprawdzić jego działanie w trybie tekstowym, posługując się załączonymi przykładami. Następnie w głównym folderze kompilatora *c:/avrgcc* tworzymy podkatalogi [*ide*] oraz [*proj*] przeznaczone na plik wykonywalny naszego projektu Delphi oraz na foldery poszczególnych projektów AVR.

Po otwarciu w Delphi nowej aplikacji ustawiamy w opcjach folder pliku wynikowego *exe* na *c:/avrgcc/ide*. Ponieważ dla uproszczenia będziemy się posługiwać w programie ściśle określoną strukturą katalogów, testowe uruchomienia będą wymagały startu właś-



Rys. 2. Składnia wyświetlanego tekstu jest automatycznie kolorowana, co znakomicie upraszcza analizę programów

nie z tego docelowego folderu. Po zapisaniu nowego projektu Delphi, wszystko jest gotowe do rozpoczęcia właściwego programowania.

Zaczynamy od pomysłu na interfejs ekranowy

To jest wybór w znacznej mierze subiektywny. Dwie podstawowe możliwości to SDI albo MDI (*single* lub *multi document interface*). MDI w ramach głównego okna - ramki pozwala na otwarcie wielu potomnych okienek oraz upraszcza zarządzanie nimi (ustawia obok, w kaskadzie, udostępnia wykaz itd.). Większość profesjonalnych IDE opiera się właśnie na modelu MDI, przydatnym zwłaszcza podczas debugowania. W naszym - pozbawionym debugera - programie nie zachodzi konieczność otwierania dużej liczby pomocniczych okienek. Dlatego możemy pozostawić jak najwięcej miejsca dla obszaru edycji kodu. W takim przypadku całkowicie wystarczy model SDI. Istotne informacje przedstawiane zazwyczaj w dodatkowych oknach spróbujemy udostępniać w inny sposób.

Do nawigacji w obrębie projektu C wybrałem gotowy komponent *TTabControl*. Każda zakładka (*Tabs*) odpowiada jednemu plikowi projektu. Dynamiczne dodawanie i usuwanie zakładek odpowiada dodawaniu i usuwaniu plików z projektu. W ten sposób na ekranie mamy zawsze całą zawartość projektu i możemy szybko przełączać się pomiędzy plikami (w taki sam sposób jest zorganizowany edytor w Delphi). Nie ma specjalnych ograniczeń liczby zakładek, ale liczba plików w przeciętnym projekcie dla mikrokontrolera nie jest nigdy zbyt duża. Ze względu na znaczne uproszczenie programu, przyjąłem ograniczoną do 32 liczbę plików - w praktyce nigdy nie powinno to być za mało.

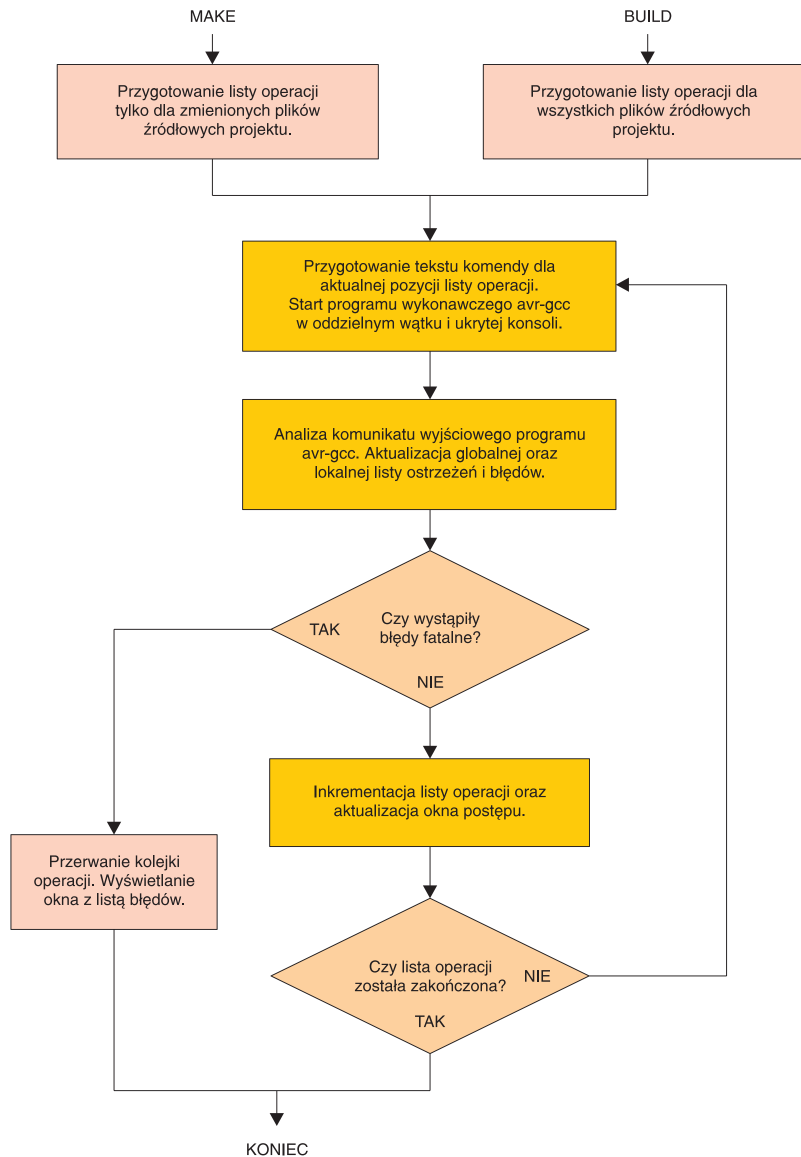
Na potrzeby edycji tworzymy tylko jeden obiekt klasy *TSynEdit*. Przełączanie pomiędzy plikami polega na ich otwieraniu w tym samym obiekcie. Alternatywnym rozwiązaniem byłoby utworzenie oddzielnego obiektu *SynEdit* dla każdego pliku - jest to jednak zbędne (i błędne) szafowanie zasobami systemu. Jedynym argumentem „za" mogłaby być większa szybkość przełączania. Jednak przy parametrach współczesnych dysków twardych opóźnienia są praktycznie nieazauważalne (można dodatkowo wprowadzić buforowanie plików w pamięci, ale po pierwszych próbach stwierdziłem, że to też nie jest konieczne).

Najważniejsze bieżące informacje o działaniu programu pokażemy w tradycyjny sposób - na belce statusu. Dokładamy więc do naszego interfejsu komponent *TStatusBar*.

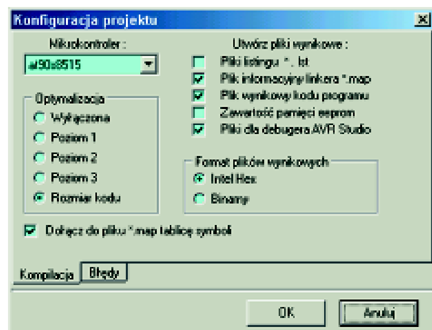
Potrzebne też będzie sterowanie funkcjami programu. Do wyboru mamy tradycyjne menu, skróty klawiszowe, paski narzędziowe z przyciskami. Ponieważ przy tworzeniu kodu dla mikrokontrolera posługujemy się głównie klawiaturą, starałem się maksymalnie wyeliminować użycie myszy. Jest to wybór subiektywny, jednakże zgrupowanie wszystkich komend w komponencie *TActionList* pozwala na bardzo proste rozszerzenie zestawu kontrolek sterujących, zgodnie z własnymi upodobaniami.

Struktura i przechowywanie informacji o projektach C-AVR

Na projekt w C składają się:
- poszczególne pliki kodu źródłowego (**.c*, **.s*),



Rys. 3. Przebieg kompilacji w AVR-GCC



Rys. 4. Okno konfiguracji projektu

- pliki nagłówkowe (*.h),
- zestaw wybranych opcji kompilatora i linkera (np. informacja o typie mikrokontrolera, poziomie optymalizacji, formatach wynikowych, zakresie raportowania ostrzeżeń itd.),
- informacje pomocnicze (np. ustawienia edytora, zakładki w tekście).

Przyjąłem następujące rozwiązanie:
- W folderze [*avrgcc\proj*] są umieszczone podkatalogi projektów - oddzielny folder na każdy projekt.
- W folderze projektu umieszczamy wszystkie jego pliki oraz plik konfiguracyjny.*cfg* zawierający informacje o projekcie. Na podstawie tego pliku oprogramowanie będzie umiało załadować projekt w takiej postaci, w jakiej został zapisany.
- W folderze [*avrgcc\proj*] umieścimy też generowany automatycznie plik *lastwork.cfg*, zapisujący dane projektu w chwili zamykania programu. Posłuży on do odtworzenia roboczego projektu przy ponownym uruchomieniu.

Plik.*cfg* jest jednym dużym rekordem zawierającym wszystkie potrzebne informacje. Ponieważ trzeba się liczyć z możliwością późniejszego rozszerzenia zestawu opcji - przewidzimy spory zapas pól rezerwowych. Nie jest to ani jedyne, ani najbardziej eleganckie rozwiązanie, niemniej w naszym przypadku będzie w zupełności wystarczające. Odwzorowaniem tego rekordu jest w programie zmienna *ProjectInfo*.

Do zapisywania i otwierania projektów posłużą standardowe dialogi *TOpenFile* i *TSaveFile*. Mogłyby one zostać użyte również dla potrzeb poszczególnych plików projektu, jednak ze względu na występującą wtedy konieczność ciągłego przestawiania konfiguracji dialogów zastosowałem dla plików oddzielny zestaw tych samych komponentów.

Oczywiście musimy się też wyposażyć w narzędzie do wyboru wszelkich potrzebnych opcji - posłużą do tego celu oddzielne okna dialogowe z zestawami odpowiednich kontrolek. Na początek uwzględnimy tylko najbardziej potrzebne - później będzie można listę dowolnie rozszerzać.

Okno konfiguracji oraz wszystkie typy jej dotyczące zebrane są w module *u_options*. Użycie w dialogu komponentu *TPageControl* zapewnia niezbędną elastyczność w późniejszym dodawaniu opcji.

Jak to ma działać?

Przebieg pracy kompilatora AVR-GCC możemy prześledzić, analizując komunikaty programu *make* wywoływanego bezpośrednio z konsoli albo z poziomu AVR Studio wersja 3.5 (wersja 4 nie współpracuje - przynajmniej na razie - z zewnętrznym kompilatorem C). Zauważymy następujące etapy:
- kompilacja i asemblacja plików źródłowych *.c* do plików relokowalnych *.o*,
- asemblacja plików źródłowych *.s* do plików relokowalnych *.o*,
- linkowanie plików relokowalnych do formatu wyjściowego *.elf* z utworzeniem pliku informacyjnego *.map*,
- tworzenie na podstawie *.elf* plików wynikowych *.obj* oraz plików wynikowych kodu: *.hex* (kod) i *.eep* (zawartość wewnętrznej EEPROM),
- konwersja formatu *.elf* do *.cof* i *.sym* używanych przez debuger AVR Studio.

To samo chcemy zrealizować za pomocą naszego programu i jednocześnie mieć możliwość ustawiania z poziomu środowiska graficznego potrzebnych parametrów oraz przechwytywania i odpowiedniej obsługi komunikatów o błędach. Algorytm postępowania jest przedstawiony na **rys. 3**.

Kompilację rozpoczynają zdarzenia *OnExecute* akcji związanych z poleceniami *make* (dla plików zmienionych) lub *build* (dla wszystkich plików). Każdorazowo - w zależności od zestawu plików źródłowych oraz wybranych opcji kompilacji (**rys. 4**) - jest tworzona lista operacji, które należy kolejno wykonać dla uzyskania potrzebnych efektów. Wszystkie niezbędne do tego zmienne i metody zawarte są w obiekcie *TCompOperationList*. Ustawiamy też flagę *Gccbusy*, która informuje wszelkie inne procedury, że trwa kompilacja. Lista operacji zostaje dodatkowo przekazana do metody *PrepareCompileList* okna wizualizacji postępu kompilacji (moduł *u_multicomp*).

Następnie wywoływana jest metoda *StartGcc*. *StartGcc* najpierw wywołuje metodę *UpdateGccParams*, która - w zależności od opisu operacji (zawartego w aktualnej pozycji listy operacji) oraz ustawionych opcji - formatuje linie komend wykorzystywane później bezpośrednio do uruchamiania narzędzi pakietu AVR-GCC (linie komend są lokowane w zmiennych globalnych *GccCommand* i *GccParams*, aby były widziane przez potomne wątki procesu głównego). Po przygotowaniu komend powoływany jest dodatkowy wątek *CompThread*, w którym uruchamiamy wybrane narzędzie AVR-GCC, czekamy na zakończenie jego pracy i przechwytujemy komunikat wyjściowy, lokując go w zmiennej globalnej *Gccoutput*.

Wątek kończąc swój „żywot", wykonuje zdarzenie *OnTerminate*, któremu przypisaliśmy metodę *EndCompilation* okna głównego. W metodzie tej jest najpierw analizowany, za pomocą funkcji *ParseErrors*, komunikat otrzymany z AVR-GCC i wykrywane są błędy i ostrzeżenia. Funkcja ta jest cokolwiek zawikłana, gdyż musi zidentyfikować błędy w różnie sformatowanych odpowiedziach poszczególnych narzędzi. Prawdopodobnie w trakcie używania wyjdą na jaw jakieś jej braki i będzie podlegać modyfikacjom. Następnie *EndCompilation*, na podstawie rezultatu *ParseErrors* oraz danych dotyczących bieżącej operacji, decyduje co robić dalej:
- przerwać bezwarunkowo kolejkę operacji z informacją o wystąpieniu błędu fatalnego AVR-GCC (*fatal error* - błąd, który powoduje całkowicie nieprawidłową realizację operacji),
- dopisać do wykazu błędów ostrzeżenia AVR-GCC (*warning* - informuje o nieprawidłowościach, które na ogół nie wpływają na poprawność wykonania, ale mogą jednak stać się przyczyną nieprzewidzianych kłopotów) i warunkowo przerwać albo kontynuować kolejkę,
- bez żadnych zastrzeżeń kontynuować kolejkę aż do realizacji ostatniej pozycji listy operacji.

Kontynuacja kolejki polega na zwiększeniu indeksu listy operacji i ponownym wywołaniu *StartGcc*. Nie jest to jednak realizowane bezpośrednio w *EndCompilation*, ale poprzez wysłanie do własnego okna komunikatu funkcją *PostMessage*. Komunikat ten trafia na listę komunikatów okna, jest pobrany i zrealizowany (poprzez metodę *wmContinueCompilation*) z pewnym opóźnieniem pozwalającym na prawidłowe zwolnienie obiektu kończącego się wątku przed utworzeniem i uruchomieniem następnego.

Zauważmy, że kompilację/asemblację przeprowadzamy w oddzielnej operacji osobno dla każdego pliku. Wprawdzie AVR-GCC może przyjąć do jednoczes-

nej „obróbki“ listę plików, ale rozdzielenie na cząstkowe procesy pozwala na większą elastyczność - zwłaszcza w zakresie analizy błędów i wizualizacji postępu kompilacji, nie wpływając zarazem w zauważalny sposób na szybkość.

Dodatkowo środowisko jest wyposażone w możliwość uruchomienia pojedynczej operacji kompilowania wybranego pliku źródłowego. Jest to bardzo pomocne przy lokalizacji błędów składniowych - zwłaszcza w pierwszym etapie pisania nowego programu C. Do sygnalizacji przebiegu kompilacji służy dodatkowe okienko zawarte w module *u_process*.

Dzięki inicjatywie i pracy Jerzego Szczesiula powstało pierwsze polskie środowisko integrujące kompilator AVR-GCC z programem AVR Studio. Powinno być ono nadal rozwijane, w związku z czym zachęcamy naszych Czytelników do podjęcia wyzwania: mamy szansę dać coś całemu elektronicznemu światu.

Uruchamianie programów AVR-GCC

Jak wspomniano wyżej, do uruchamiania narzędzi AVR-GCC jest powoływany oddzielny wątek. Pozwala to na zachowanie interakcji z oknem głównym podczas oczekiwania na zakończenie procesów potomnych. W naszym przypadku akurat nie ma to wielkiego znaczenia (i tak nic nie robimy w programie podczas kompilacji, a przy tym nie trwa ona zbyt długo), ale zastosowałem to rozwiązanie jako typowe dla takich zadań.

Do uruchomienia AVR-GCC jest użyta funkcja *CreateProcess* z linią komend pobraną ze zmiennych globalnych *GccCommand* i *GccParams*. Jednak najpierw musimy przeprowadzić pewne przygotowania (moduł *u_thread*):

- Uruchamiamy kanał, przez który konsolowy program AVR-GCC prześle do naszej aplikacji komunikat o wyniku swego działania. Służy do tego rurka (*pipe*). Tworzymy ją funkcją *CreatePipe*, uzyskując w wyniku uchwyty wejścia i wyjścia rurki. Przedtem musimy pamiętać, aby w atrybutach bezpieczeństwa (*Security Attributes*) powstającej rurki ustawić flagę *bInheritHandle*, co pozwoli na korzystanie z uchwytów rurki przez proces potomny.
- Następnie ustawiamy dodatkowe parametry startowe dla uruchamianego procesu:
  - standardowe wyjście komunikatów oraz błędów przypisujemy do uchwytu wejścia rurki,
  - żądamy ukrycia konsoli procesu,
  - odmaskowujemy wymagane ustawienia we flagach.

Teraz dopiero uruchamiamy proces. Wątek czeka na jego zakończenie za pomocą funkcji *WaitForSingleObject* (jest to funkcja synchroniczna, która zatrzymuje wątek na czas swego działania - to wyjaśnia znaczenie pierwszego akapitu). Następnie przepisuje komunikat wyjściowy kompilatora z wyjścia rurki do zmiennej globalnej *GccOutput* za pomocą funkcji *PeekNamedPipe* oraz *ReadFile*.

Na tym kończy się funkcja *Execute* wątku i zostaje wykonane zdarzenie *OnTerminate* obiektu *CompThread*. Jest on następnie automatycznie likwidowany, gdyż przy tworzeniu ustawiliśmy atrybut *FreeOnTerminate* - program nie musi już tym się zajmować.

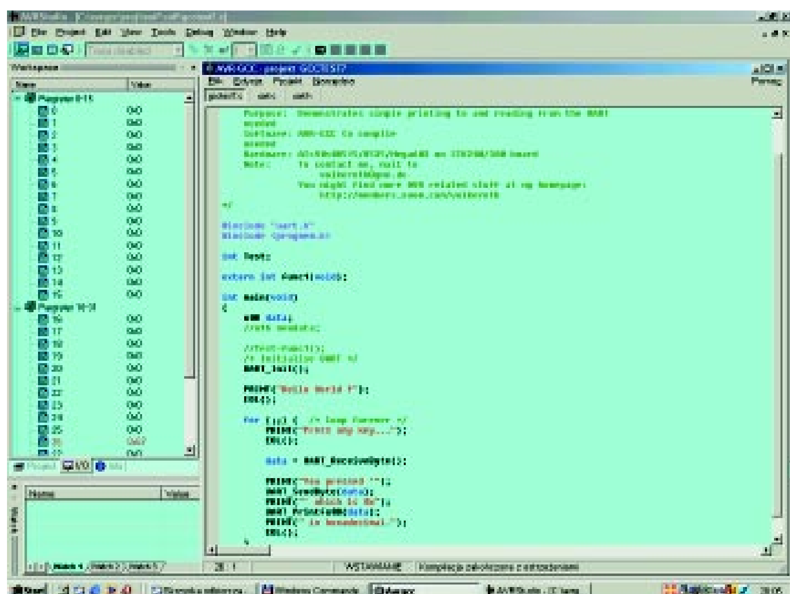
Co robimy z komunikatem kompilatora?

Jak wspomniano, na każdym etapie kompilacji komunikat *GccOutput* jest analizowany w funkcji *ParseErrors* pod kątem występowania błędów i ostrzeżeń. Zidentyfikowane opisy błędów i ostrzeżeń są lokowane w listach błędów *LocalErrorList* i *GlobalErrorList* (wykorzystywanych zamiennie przy różnych ustawieniach opcji zatrzymywania kolejki operacji). Do wyświetlenia w razie potrzeby tych list służy oddzielne okno z modułu *u_errwindow*. Wybrałem rozwiązanie z oknem niezależnym - możemy je przesuwać w dowolne miejsce, również poza główne okno programu. Poza funkcją informacyjną okno to umożliwia wskazanie lokalizacji błędu w kodzie źródłowym. Służą do tego powiązane ze sobą metody *ShowErrorLine* okna listy błędów i okna głównego. Do lokalizacji wykorzystujemy fakt, że w liniach opisu występuje nazwa modułu i numer linii. Na podstawie nazwy modułu funkcja *ShowTabByName* przełącza edytor na odpowiednią zakładkę. Następnie zostaje podświetlona i pokazana w edytorze linia o żądanym numerze.

Powyższy opis jest oczywiście jedynie orientacyjny - wszelkie szczegóły znajdziemy w kodzie źródłowym, który zawiera także wiele funkcji pomocniczych. Starałem się je jednak dosyć obszernie opisywać, co powinno znacznie ułatwić analizę ich przeznaczenia i działania.

Współpraca naszego programu z AVR Studio 4

Po uruchomieniu AVR Studio otwieramy w nim utworzony w podkatalogu [*coff*] plik **.cof*. Ustawiamy jako platformę AVR Simulator i wybieramy procesor z listy. Studio otwiera główny plik projektu i przechodzi w tryb



Rys. 5. Zrzut ekranowy zestawu AVR-GCC IDE+AVR Studio 4. Przy odpowiednim rozmieszczeniu okien wygląda to prawie jak jednolite środowisko

debugowania (**rys. 5**). Pozostałe moduły projektu pokażą się w chwili przejścia do ich kodu. Jeśli chcemy mieć wszystkie od razu - otwieramy je oddzielnie z listy plików źródłowych. Jednak ten mechanizm nie działa do końca prawidłowo - następne okna bezpośrednio po otwarciu znajdują się w trybie edycji - wystarczy jednak wykonać reset, aby środowisko debugera powróciło do właściwego stanu. Teraz można ustawić częstotliwość symulatora i przystąpić do debugowania. Jeśli musimy wprowadzić poprawki, przełączamy się z powrotem do naszego IDE i wykonujemy pełną kompilację. Następnie przechodzimy z powrotem do Studia - po chwili wykrywa ono zmianę pliku obiektowego **.cof* i pyta, czy ma go przeładować - potwierdzamy. Natomiast odmawiamy zapisania zmian w plikach źródłowych przed odświeżeniem, wprawdzie dotyczą one kopii w podkatalogu [*coff*] i nie uszkodzą plików projektu, ale w oknach debugera pozostanie nieprawdziwa informacja sprzed modyfikacji.

Po przerwie w pracy ponowne uruchomienie AVR Studia nie wymaga już żadnego ustawiania - po prostu uruchamiamy plik projektu **.aps* utworzony w folderze [*coff*] i cała konfiguracja zostaje samoczynnie odtworzona (wyjątkiem jest brak okien plików, które wcześniej były otwarte ręcznie i nie zostały ani razu wykorzystane przy debugowaniu).

Drobne niedostatki współpracy z AVR Studio nie mają większego znaczenia, bardziej uciążliwe są błędy w konwerterze *elfcoff* (błędne przetwarzanie zmiennych *volatile* czy struktur w strukturze). Środkiem zaradczym jest ściągnięcie poprawek z AVR Forum, aczkolwiek autorzy nie są do końca zadowoleni z ich działania i czasem jedynym wyjściem będzie zmiana składni kodu.

Obecna wersja projektu Delphi - chociaż osiągnęła poprawność działania podstawowych funkcji w wybranych prostych przykładach AVR C - jest tylko pierwszą przymiarką. Zawsze w trakcie używania pojawia się konieczność wprowadzania poprawek. Poza tym wypada środowisko sukcesywnie wyposażać w nowe możliwości. Zwłaszcza bogato zapowiada się wykorzystanie pełnego potencjału *TSynEdit*. Kolejne aktualizacje będą dostępne na stronie *www.easy-soft.tsnet.pl* oraz na stronie *www.ep.com.pl* w dziale *Download*. Najświeższe wersje można też znaleźć na *www.avrside.fr.pl*. Ponieważ mamy do czynienia z wolnym oprogramowaniem rozpowszechnianym na zasadach licencji GNU GPL (General Public License), to każdy może swobodnie korzystać z programu oraz kodów źródłowych do własnych prac i udoskonaleń. Należy jedynie pamiętać o obwarowaniach licencyjnych MPL/GPL dla komponentu *TSynEdit* przy rozpowszechnianiu własnych wersji programu.

Jerzy Szczesiul, AVT
jerzy.szczesiul@ep.com.pl

Aktualizacje środowiska będzie można znaleźć na stronie www.ep.com.pl w dziale Download.