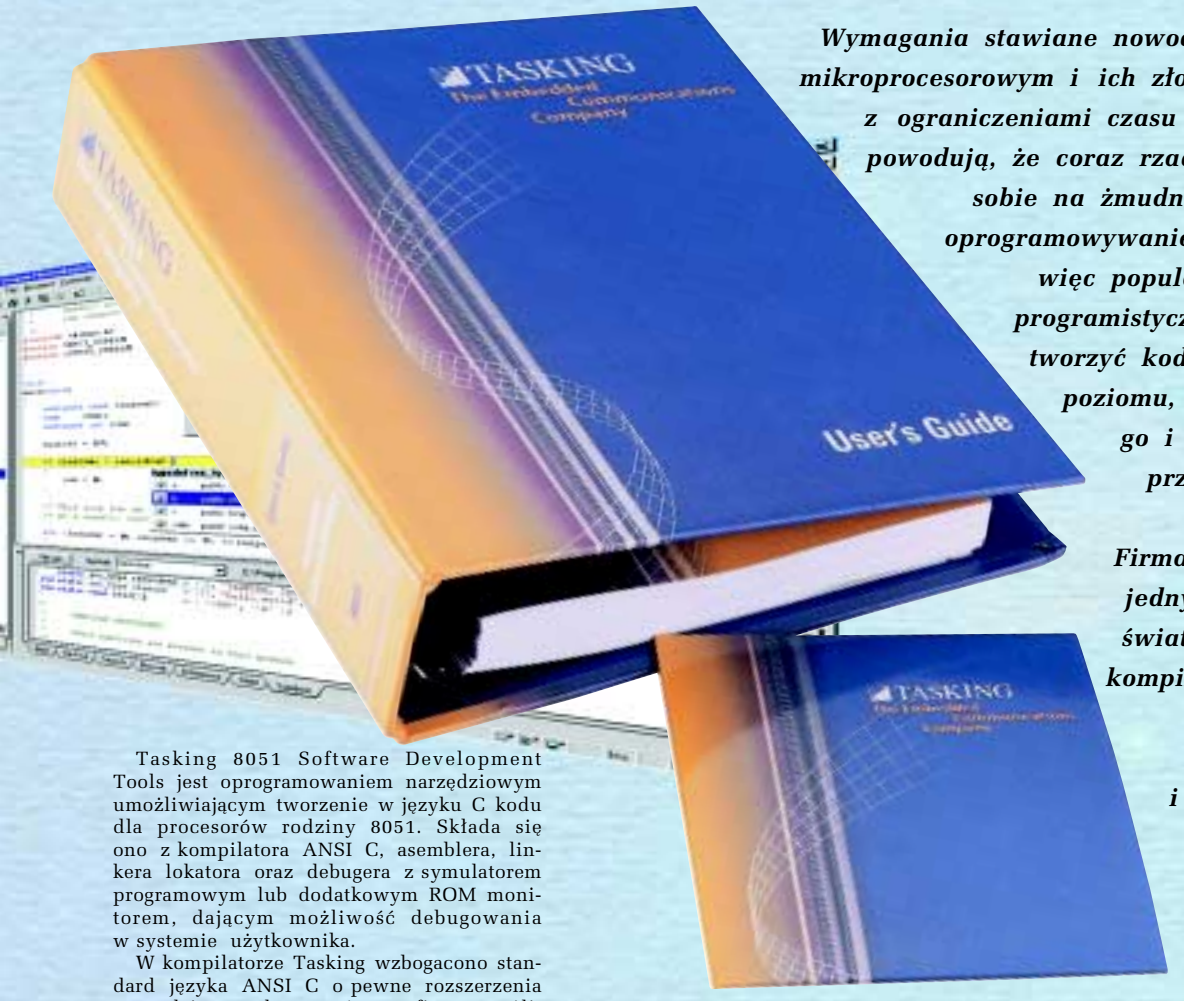
Oprogramowanie narzędziowe firmy Tasking Inc. dla mikrokontrolerów '51



Wymagania stawiane nowoczesnym układom mikroprocesorowym i ich złożoność w połączeniu z ograniczeniami czasu realizacji projektu powodują, że coraz rzadziej można pozwolić sobie na żmudne i długotrwałe oprogramowywanie urządzenia. Rośnie więc popularność pakietów programistycznych pozwalających tworzyć kod w języku wysokiego poziomu, a następnie testować go i uruchamiać jeszcze przed implementacją w prototypie. Firma Tasking Inc. jest jednym z czołowych światowych producentów kompilatorów języka C dla szerokiej gamy mikrokontrolerów i procesorów. Wśród nich jest także popularny mikrokontroler Intel 8051.

Tasking 8051 Software Development Tools jest oprogramowaniem narzędziowym umożliwiającym tworzenie w języku C kodu dla procesorów rodziny 8051. Składa się ono z kompilatora ANSI C, asemblera, linkera lokatora oraz debugera z symulatorem programowym lub dodatkowym ROM monitorem, dającym możliwość debugowania w systemie użytkownika.

W kompilatorze Tasking wzbogacono standard języka ANSI C o pewne rozszerzenia pozwalające wykorzystać specyficzne możliwości mikrokontrolera 8051. Rozszerzenia obejmują między innymi zmienne bitowe, które kompilator alokuje w bitowo adresowanej pamięci wewnętrznej mikrokontrolera, obsługę przerwań sprzętowych przez odpowiednio zadeklarowane funkcje języka C wraz z automatyczną zmianą wybranego banku rejestrów w momencie wywołania przerwania oraz odstęp do portów I/O za pomocą specjalnych zmiennych.

Ponieważ mikrokontroler 8051 może występować w różnych konfiguracjach (z pamięcią zewnętrzną lub bez), program wyposażono w cztery różne modele pamięci, które określają, w jaki sposób zmienne zadeklarowane w programie mają być alokowane w dostępnej pamięci RAM. Modele te to:

- *small* - korzystający tylko z pamięci wewnętrznej procesora,
- *auxpage* - wykorzystujący 256 bajtów pamięci zewnętrznej, stosowany dla procesorów wyposażonych we wbudowaną zewnętrzną pamięć RAM,
- *large* - dla rozwiązań z zewnętrzną pamięcią RAM do 64kB,
- *reentrant* - w którym procesor korzysta z dużej pamięci zewnętrznej, ale w odróżnieniu od poprzednich modeli (*auxpage* i *large*) zorganizowany jest tam wirtualny stos programu oraz sterta, dzięki czemu używając tego modelu pamięci programista może wykorzystywać takie techniki programowania jak wywołania rekurencyjne funkcji i dynamiczną alokację pamięci.

Oczywiście wszystkie wymienione modele pamięci wykorzystują wewnętrzny RAM procesora, umieszczając tam często wykorzystywane zmienne, dzięki czemu program zyskuje na szybkości działania, bowiem dostęp do wewnętrznego RAM-u jest dużo szybszy niż do zewnętrznego. Użytkownik dodatkowo ma możliwość samodzielnego wskazania, gdzie zadeklarowana zmienna ma zostać umieszczona, to znaczy w jakim obszarze pamięci i pod jakim adresem. I tak przykładowo:

```
_xdat unsigned char Display _at(0x2000);
```

deklaruje zmienną bajtową w pamięci zewnętrznej pod adresem 2000h.

Kompilator wyposażony jest w rozbudowane i w pełni sterowane funkcje optymalizacji kodu pod kątem szybkości i objętości. W praktyce na parametry kodu maszynowego można wpływać nie tylko ustawieniami kompilatora, ale również pisząc odpowiednio kod programu. Przykładowo, poprzez rozbijanie w przemyślany sposób zadań realizowanych przez program na odpowiednie bloki funkcyjne otrzymamy zwięzły, zajmujący niewiele miejsca kod wynikowy.

W razie potrzeby można przyspieszyć wykonanie programu, wywołując funkcje jako rozwijalne (dyrektywa *inline*), co pozwala uniknąć straty czasu procesora potrzebnego na przekazanie parametrów do funkcji i skok do jej kodu. W takim przypadku każdorazowo w miejscu wywołania funkcji rozwijalnej zostanie umieszczona kopia jej kodu. Niestety, takie rozwiązanie, choć przy-

śpiesza działanie programu, powoduje jednocześnie wzrost wielkości kodu maszynowego. Warto jednak pamiętać, że dla „krótkich", często wywoływanych funkcji może przynosić pozytywne rezultaty.

Mimo zastosowania funkcji optymalizujących kod programu, nie zaszkodzi jednak, jeśli programista będzie posiadał elementarną wiedzę o instrukcjach mikrokontrolerów rodziny 8051. Pozwoli to uniknąć pewnych pułapek. Dla przykładu, jeśli w pętli ma zostać przeprowadzone pewne zadanie dziesięć razy, to ze względu na istnienie rozkazu porównania zawartości akumulatora z zerem, korzystniej (o kilka instrukcji kodu maszynowego) będzie iterować od dziesięciu do zera, niż jak napisałaby większość początkujących użytkowników, od zera od dziesięciu.

Dzięki narzędziom optymalizacji kodu i rozważnemu stylowi pisania programu kompilator tworzy dość szybki i krótki kod wykonywalny, łatwo mieszczący się we współczesnych procesorach wyposażonych we wbudowany ROM.

Tasking pozwala ominąć przykre ograniczenie procesora 8051, jakim jest brak możliwości obsługi pamięci ROM o pojemności większej niż 64kB. Proponowanym rozwiązaniem jest bankowanie pamięci. Użytkownik może posłużyć się 256 bankami pamięci po 32kB każdy. Jeśli kompilator wykryje wywołanie funkcji zdefiniowanej w innym banku niż aktualnie używany, wówczas automatycznie umieści w kodzie wykonywalnym odpowiednie instrukcje przełączające banki poprzez przykładowo port P1.

Dodatkiem do kompilatora są biblioteki umożliwiające obliczenia zmiennopozycyjne (pojedyncza precyzja), obsługę portów I/O i dynamiczne zarządzanie pamięcią.

Asembler

Integralną częścią pakietu jest asembler. Narzędzia podobne do Tasking SDT przetwarzają bowiem kod C na kod asemblera, a następnie poddają go kompilacji, w wyniku czego otrzymywane są relokowalne pliki obiektowe. Stanowią one źródło dla linkera lokatora, który na ich podstawie generuje plik w formacie wyjściowym dla programatorów pamięci ROM.

Moduł asemblera jest w pełni funkcjonalny, dzięki niemu programista może nie tylko umieszczać w kodzie C wstawki asemblerowe, ale także kompilować projekty napisane w asemblerze.

Linker lokator

Zadaniem tego elementu pakietu jest łączenie wszystkich części kodu, powstałych w trakcie kompilacji różnych plików programu i lokowanie ich w pamięci, również z uwzględnieniem życzeń użytkownika. Linker lokator pozwala zadać wartość przesunięcia bloku kodu względem adresu zerowego lub umieścić kod wybranej funkcji we wskazanym miejscu obszaru pamięci programu. Jest również odpowiedzialny za generowanie w odpowiednim formacie pliku dla programatora pamięci. Dostępne formaty to: Intel Hex, Motorola S Record, Intel OMF51, IEE 695.

Debuger CrossViev Pro

Debuger wchodzący w skład pakietu jest w pełni samodzielnym programem o otwartej architekturze. Pozwala on na wykorzystanie różnych środowisk symulacyjnych. Standardowo wykorzystywany jest szybki symulator programowy. Do droższej wersji pakietu dołączony jest specjalny program nazywany ROM Monitorem. Użytkownik instaluje ten program w prototypie projektowanego urządzenia i bezpośrednio na nim przeprowadza proces uruchamiania i testowania. Komunikacja z debugerem odbywa się w takim przypadku poprzez port szeregowy procesora, który wykonuje testowany program.

Innym rozwiązaniem jest użycie typowego, proponowanego przez wielu producentów, emulatora sprzętowego. Dodatkowo, do zastosowań wymagających specjalnego środowiska uruchomieniowego, można poprzez stworzenie specjalnych bibliotek uzyskać sprzęg z dowolnym systemem, przy założeniu, że jest to system z procesorem kompatybilnym z 8051.

Sam debuger jest łatwą w obsłudze aplikacją. Podglądane wartości umieszczone są w czytelnych, skalowalnych oknach. Użytkownik ma dostęp do wszystkich rejestrów procesora, w tym również portów - może odczytywać i modyfikować ich zawartość. Podobnie istnieje możliwość odczytu i zapisu pamięci, jak również wyszukiwania w niej zadanej wartości. Dla ułatwienia pracy programiście nie zapomniano o klasycznym mechanizmie podglądu wartości zmiennych użytych w kodzie programu. W oknie inspektora zmiennych można oglądać ich wartości, a także dokonywać zmian.

Oprócz typowych mechanizmów służących sterowaniu wykonywaniem testowanego programu, czyli zwykłych breakpointów, jak również czułych na wartości danych, program udostępnia narzędzie *trace*, służące do śledzenia kolejności realizowania poszczególnych instrukcji. Niezwykła przydatność tej funkcji objawia się w przypadku, gdy błędy w programie są wynikiem losowego, występującego incydentalnie zbiegu okoliczności lub mają przyczynę w pozornie tylko bezbłędnym wykonaniu instrukcji poprzedzających miejsce awarii. Dzięki liście *trace* użytkownik może dosłownie „cofnąć" się po swoich śladach do miejsca, które było przyczyną błędu.

Podgląd wykonywanego kodu może odbywać się w trzech trybach: widoku kodu języka C, widoku kodu asemblera lub w trybie mieszanym. Sam debugowany program może być zależnie od wybranego trybu pracy wykonywany po instrukcji kodu maszynowego lub po instrukcji C.

CrossViev Pro wyposażony jest w narzędzia do testowania i profilowania aplikacji. Dzięki nim można określić, czy badany fragment kodu został wykonany podczas ostatniego uruchomienia programu, czy też nie. Można zbadać, ile czasu procesora zajmuje wykonanie poszczególnych części kodu. Informacja taka może być cenną wskazówką podczas poszukiwania w programie „wąskich gardeł", nad którymi warto jeszcze popracować, aby przyspieszyć wykonywanie programu.

Na zakończenie proponuję przeanalizowanie krótkiego przykładu programowania mikrokontrolera 8051 w języku C:

```
1  #include <reg51.sfr>
2
3  _xdat unsigned char Buffer[255];
4  unsigned int i;
5
6  _interrupt(4) void
   Serial_Interrupt (void)
   {
7  //ta funkcja obsługuje przerwanie
8  RI = 0; //zerowanie flagi -
   //gotowość do odbioru kolejny bajt
9  Buffer[i] = SBUF;
   //zapisz wartość buforu do
   //tablicy
10 i++; //inkrementacja licznika
   //elementów tablicy
11 }
12
13 void main (void)
   //początek programu głównego
15 {
16 TL1 = 0xFE;//ustawienie
   //transmisji portu szeregowego
17 TH1 = 0xFE;
18 TMOD = 0x20;
19 TCON = 0x40;
20 SCON = 0x50;
21 i = 0;//zerowanie licznika
   //elementów tablicy
22 EA = 1;//zezwól na przerwania
22 ES = 1;//włącz przerwanie od portu
   //szeregowego
23 do {} while (i <= 255);
   //oczekiwanie na zapełnienie
   //tablicy
24 }
```

Objaśnienie:

W wierszu pierwszym dyrektywą *#include* dołączono plik, w którym zdefiniowane są rejestry procesora 8051. W wierszu trzecim deklarowana jest 255-bajtowa tablica Buffer umieszczona w pamięci zewnętrznej (dyrektywa *_xdat*). Poniżej definiowana jest funkcja *Serial_Interrupt*, której zadaniem jest odczytanie z rejestru portu szeregowego bajtu i zapisanie go do tablicy *Buffer*. Widoczne jest, że funkcja realizująca przerwanie odwołuje się do rejestru SBUF i RI, w celu pobrania wartości bufora portu i ustawienia flagi przerwania.

Natomiast właściwy program zaczyna się od funkcji *main* (wiersz trzynasty) i to od tego kodu rozpocznie wykonywanie programu procesor. Najpierw zrealizowane zostaną instrukcje ustawiające zawartość rejestrów TL1, TH1, TMOD, TCON, SCON, mające na celu ustawienie parametrów transmisji szeregowej. Dalej, zapisując jedynki do rejestrów EA i ES, procesor uaktywnia przerwanie pochodzące od portu szeregowego i w pętli *do{}while* przechodzi do oczekiwania, aż cała 255-bajtowa tablica wypełni się danymi.

Przedstawiony przykład jest prostym programem realizującym odbiór danych z portu szeregowego. Daje on podstawowy obraz na temat programowania mikrokontrolerów w językach wysokiego poziomu, jakim jest ANSI C. Wszystkich zainteresowanych pobraniem demonstracyjnej wersji programu zapraszam na strony internetowe www.evatronix.com.pl.

Tomasz Jakóbiec
tomjak@bielsko.evatronix.com.pl

Demonstracyjne wersje programów firmy Tasking są dostępne na płycie CD-EP10/2001B oraz w Internecie pod adresem: www.evatronix.com.pl.