kpt. dr inż. Mariusz BODJAŃSKI
Wojskowy Instytut Techniczny Uzbrojenia

METODY I NARZĘDZIA PROJEKTOWANIA APLIKACJI DLA MIKROKONTROLERÓW JEDNOUKŁADOWYCH

W artykule opisano proces projektowania oprogramowania dla systemów z mikrokontrolerami jednoukładowymi. Przedstawiono podstawowe metody i narzędzia tworzenia aplikacji. Porównano jakość kodów wynikowych i efektywność projektowania za pomocą prezentowanych metod.

1. Wstęp

Programowanie systemów mikroprocesorowych jest procesem różniącym się od programowania komputerów klasy PC (jednostka centralna, monitor, klawiatura). Projekt aplikacji przeznaczonej do uruchomienia w systemie zawierającym mikrokontroler jednoukładowy musi uwzględniać wiele aspektów: między innymi obsługę urządzeń peryferyjnych, zależności czasowe sygnałów nimi sterujących, długość kodu wynikowego itp. W związku z tym, programista chcąc spełnić powyższe kryteria, optymalizuje projekt pod względem szybkości lub rozmiaru.

2. Proces projektowania aplikacji

Pierwszym krokiem w procesie projektowania oprogramowania dla systemu z mikrokontrolerem jest dogłębne przestudiowanie architektury wewnętrznej mikrokontrolera, otoczenia sprzętowego oraz zapoznanie się ze wszystkimi sygnałami współpracującymi z urządzeniami peryferyjnymi systemu. Ma to szczególne znaczenie dla kształtu procedur składowych programu, komunikujących się z osprzętem mikrokontrolera.

Następnym etapem jest zgrubne ustalenie struktury programu, czyli:

- analiza zadania,
- projekt algorytmu programu,
- wyznaczenie zależności do obliczeń numerycznych,
- określenie kształtu procedur odpowiedzialnych za realizację poszczególnych zagadnień,
- wytyczenie rozgałęzień warunkowych.

Kolejne kroki to edycja kodu źródłowego, kompilacja, symulacja i debuggowanie, czyli śledzenie pracy programu oraz usuwanie błędów. Praktyka pokazuje, że etap debuggowania jest równie czasochłonny jak tworzenie kodu aplikacji. Proces tworzenia aplikacji kończy się oczywiście uruchomieniem programu, czyli załadowaniem kodu wynikowego do pamięci mikrokontrolera.

Częstym błędem popełnianym przez niedoświadczonych programistów jest rozpoczynanie pracy od tworzenie kodu pętli głównej. Tymczasem rozwiązywanie postawionego problemu należy zacząć od analizy i implementacji zadań cząstkowych, przybliżając się stopniowo do celu pracy. W przypadku programowania mikrokontrolerów jednoukładowych, pracę należy rozpocząć od budowy zbioru procedur obsługujących otoczenie sprzętowe mikrokontrolera.

3. Narzędzia tworzenia aplikacji

Wszystkie powyższe czynności wykonywane są przy pomocy odpowiednich narzędzi:

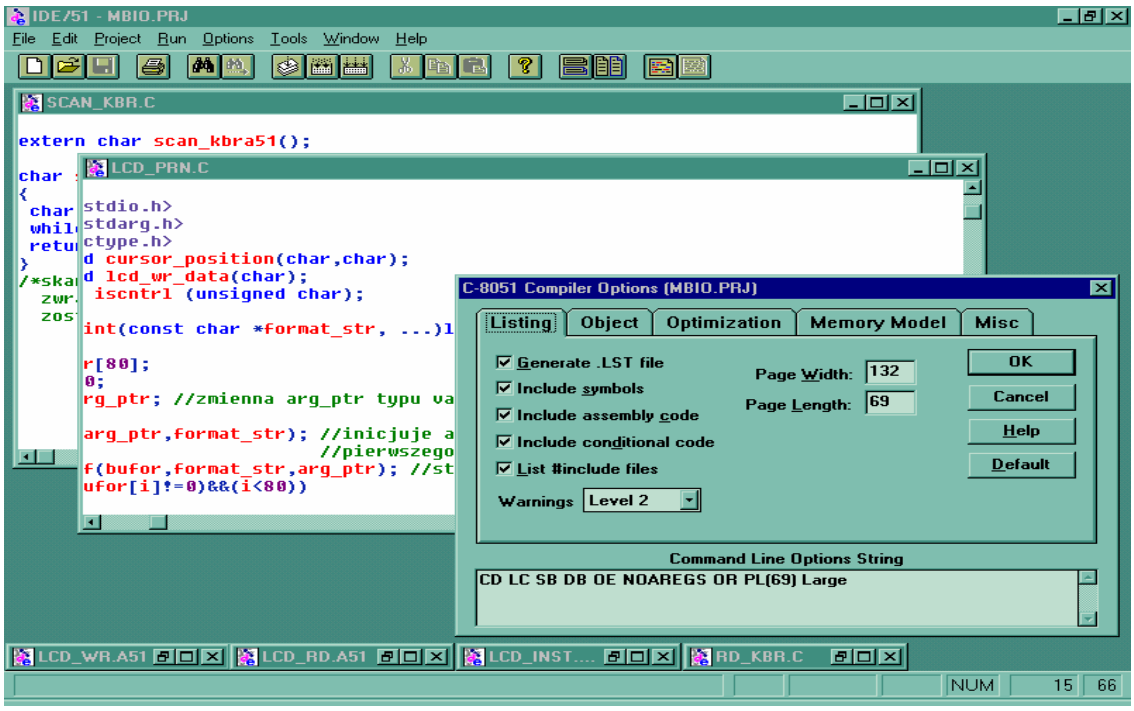
- Do budowy i edycji kodów źródłowych programów służą edytory tekstu wyposażone w standardowe opcje kopiowania, przenoszenia, szukania, itd.
- Kompilator jest programem tłumaczącym zapis kodów źródłowych z języka wysokiego poziomu (np. C/C++) na język niskiego poziomu. Kompilator analizuje również zapis źródłowy pod kątem błędów składniowych (wyświetla rodzaj błędu oraz miejsce jego wystąpienia).
- Asembler jest programem tłumaczącym zapis kodów źródłowych z języka niskiego poziomu (języka asemblera) do kodu wynikowego w postaci relokowalnego (przemieszczalnego) modułu typu *object* (ang. *object code*). Moduły *obj* nie nadają się jeszcze do załadowania do pamięci mikrokontrolera.
- Linker jest narzędziem łączącym moduły typu *obj* oraz moduły biblioteczne w ostateczny nierelokowalny kod wynikowy, który można załadować do pamięci mikrokontrolera i uruchomić.
- Usuwanie błędów merytorycznych z programu przeprowadza się za pomocą narzędzia zwanego debuggerem bądź symulatorem, który pozwala na śledzenie aplikacji krok po kroku i obserwację zmian w pamięci oraz w buforach urządzeń peryferyjnych mikrokontrolera. Debuggera nie należy mylić z emulatorem układowym, który jest fizycznym urządzeniem współpracującym z komputerem i emulującym pracę mikrokontrolera.

Obecnie na rynku dostępne są tzw. zintegrowane środowiska projektowe, łączące w jednej aplikacji wszystkie wspomniane powyżej narzędzia (edytor kodów źródłowych, kompilator, asembler, linker i debugger). Przykładem takiego programu jest *KEIL 8051* [3]. Jest to kompletny, wielozadaniowy system wspomagający projektowanie oprogramowania dla mikrokontrolerów rodziny 8051 i pochodnych. W zintegrowanym środowisku graficznym, pracującym w systemie Windows, łączy takie narzędzia jak:

- *IDE 8051_251* – platforma narzędziowa,
- *SimCASE 8051_251* – symulator/debugger,
- optymalizowany ANSI C kompilator,
- asembler skrośny,
- linker,
- menedżer projektów,
- menedżer bibliotek,

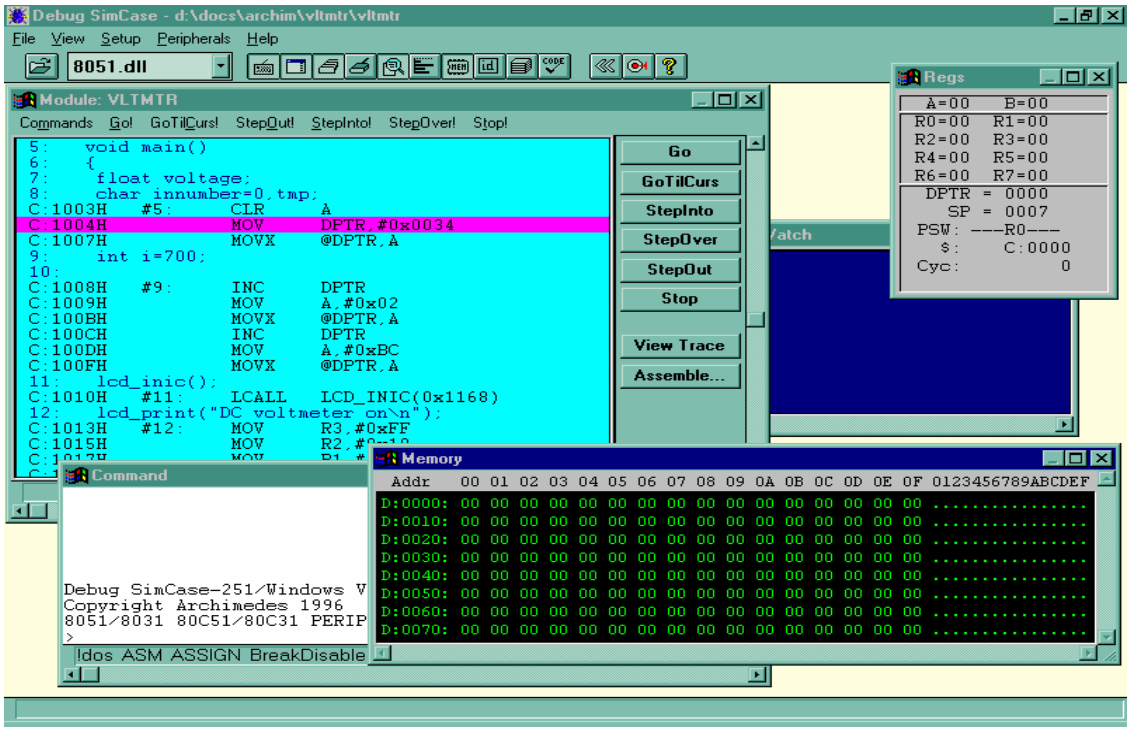
IDE 8051_251 (ang. *Integrated Development Environment*) charakteryzuje się między innymi następującymi własnościami:

- możliwość edycji wielu zbiorów,
- wielofunkcyjny edytor z kolorowaniem elementów składni języka C,
- rozbudowany system memu i paski narzędzi,
- konfiguracja klawiszy skrótów,
- menedżer aplikacji pozwalający uruchamiać inne programy,
- menedżer projektów, kontrolujący zbiory należące do projektu,
- okienka edycyjne konfigurujące ustawienia wszystkich narzędzi.



Rys. 1. Platforma narzędziowa *IDE 8051*

SimCASE 8051_251 jest debuggerem przeprowadzającym analizę programów kompilowanych przez *IDE* oraz symulującym wszystkie układy mikrokontrolera. Analiza może odbywać się na poziomie asemblera, języka C oraz w trybie mieszanym. Pozwala na standardową procedurę debuggowania programów poprzez pracę krokową, zakładanie pułapek, symulację wymuszeń itp.



Rys. 2. Debugger *SimCASE*

SimCASE jest środowiskiem graficznym z szeregiem konfigurowalnych przez użytkownika okienek dialogowych, będących właściwymi narzędziami analizy:

- okno debuggera, pokazujące kod źródłowy programu w trzech trybach wyświetlania,
- okienko poleceń wprowadzanych w linii poleceń,
- okienko śledzące zmianę zawartości rejestrów,
- okienko śledzące zawartość deklarowanych zmiennych,
- okienko wyświetlające dane wyprowadzane przez port szeregowy,
- okienko zawartości pamięci danych,
- *symbol browser* pozwala na analizę deklarowanych zmiennych lokalnych i globalnych oraz wszystkich deklarowanych funkcji wraz z ich parametrami,
- analizator wydajności pokazuje procentowy czas wykonywania deklarowanych funkcji w odniesieniu do czasu wykonywania programu,
- okienko wywołań stosu wyświetla aktualnie zagnieżdżone funkcje,
- *code coverage* informuje o procentowym udziale kodu deklarowanych funkcji w programie,
- okienko wywołań makr, definiowanych przez użytkownika w *command window*.

4. Kompilatory języka C/C++

Kompilatory C/C++ dla mikrokontrolerów są przeważnie kompilatorami skrośnymi (ang. *cross compilers*). Oznacza to, że kompilator pracuje na komputerze klasy PC i generuje kody dla mikrokontrolera określonego typu. Typowe kompilatory języka C/C++ dla mikrokontrolerów jednoukładowych posiadają następujące cechy [5]:

- dostęp do standardowych bibliotek ANSI C (*stdio*, *ctype*, *math* czy *stdarg*),
- implementacja typów ANSI C (*unsigned*, *short*, *long*, *char*, *int*, *float*, *double*, *pointer*) oraz typów związanych z architekturą mikrokontrolerów (np. typów bitowych lub typów definiujących rejestry specjalnego przeznaczenia),
- implementacja struktur, unii oraz tablic i typów wyliczeniowych,
- deklaracje funkcji rekurencyjnych i przerwań,
- implementacja różnych modeli pamięci (*tiny*, *small*, *compact*, *large*) z podziałem na obszar danych (*data*), programu (*code*), obszar pierwszych 256 bajtów danych (*idata*) oraz obszar pamięci zewnętrznej (*xdata*),
- dostęp do obszarów adresowanych bitowo, rejestrów specjalnego przeznaczenia *SFR* (ang. *Special Function Register*) i rejestrów ogólnego przeznaczenia,
- implementacja wskaźników o długości zależnej od typu pamięci (np. 1-bajtowe w obszarze *data* i 2-bajtowe w obszarze *xdata*),
- optymalizacja kodu wynikowego (usuwanie nie wykonywanych fragmentów kodów, optymalizacja pętli, eliminacja powtarzających się fragmentów kodów, eliminacja niepotrzebnych skoków, łączenie uzależnionych wyrażeń arytmetycznych i logicznych, umieszczanie zmiennych w szybkich rejestrach mikrokontrolera, optymalizacja wyrażeń zmiennoprzecinkowych).

5. Efektywność i jakość oprogramowania

Istnieją dwie podstawowe metody programowania: za pomocą języka niskiego poziomu (asemblera) [4] i języka wysokiego poziomu [1]. Najnowsze środowiska systemów

wspomagających projektowanie oprogramowania dla mikrokontrolerów udostępniają obydwie wspomniane metody programowania.

Wydajność i jakość oprogramowania uzależniona jest od wielu czynników. Nie tylko od umiejętności i indywidualnego podejścia programisty, ale również od wybranej metody oraz przyjętego kryterium optymalizacji projektu.

Używając wyłącznie języka wysokiego poziomu programista zyskuje przede wszystkim komfort pracy i oszczędność czasu spędzonego nad projektem aplikacji. Przejrzystość i oszczędność kodu źródłowego programów napisanych w języku wysokiego poziomu jest czynnikiem mającym duży wpływ na jakość aplikacji i efektywność procesu tworzenia oprogramowania. Dostępne obecnie na rynku kompilatory C/C++ dla mikrokontrolerów jednoukładowych dostarczają wielu zoptymalizowanych (pod względem szybkości lub rozmiaru kodu wykonywalnego) i zgodnych ze standardem ANSI C bibliotek, zawierających wiele różnorodnych funkcji (np. funkcji trygonometrycznych oraz innych funkcji przetwarzania danych). Wyręcza to programistę z obowiązku budowania takich procedur, co dalece upraszcza projektowanie oprogramowania dla mikrokontrolerów.

W sytuacjach, kiedy kryterium jest szybkość wykonywania kodu nie można niestety pominąć języków niskiego poziomu. Umiejętnie zaprojektowane w asemblerze procedury charakteryzują się dużą szybkością działania i powinny być używane w komunikacji mikrokontrolera z urządzeniami peryferyjnymi. Istnieją przypadki, kiedy należy rezygnować ze standardowych funkcji bibliotecznych (nie zawsze dostatecznie zoptymalizowanych) dostarczanych przez kompilatory języków wysokiego poziomu na rzecz szybkich procedur asemblerowych. Dla przykładu na wykonanie bibliotecznej funkcji *sin( )*, dostarczanej przez większość kompilatorów mikrokontrolera C51, potrzeba ponad trzech tysięcy cykli maszynowych.

6. Hybrydowe metody projektowania oprogramowania

Oprócz przedstawionych powyżej dwóch podstawowych metod projektowania (w asemblerze i w językach wysokiego poziomu) istnieje metoda mieszana, łącząca zalety obu języków. Takie udogodnienie jest dostępne dla programisty pracującego w nowoczesnym, zintegrowanym środowisku graficznym.

Język asemblera wykorzystuje się najczęściej w projektowaniu procedur współpracujących z otoczeniem sprzętowym mikrokontrolera [2], gdzie konieczna jest szybkość obsługi poszczególnych urządzeń oraz wymagana ścisła kontrola nad sygnałami sterującymi, przychodzącymi do i wychodzącymi z mikrokontrolera.

Natomiast języki wysokiego poziomu stanowią najczęściej w tej metodzie element łączący w pewną logiczną całość procedury napisane w asemblerze. Wykorzystywane są w tych blokach programu, które odpowiedzialne są za wszelkiego rodzaju pętle i skoki warunkowe oraz funkcje wymagające odpowiedniej wiedzy i dużego nakładu pracy przy ich projektowaniu (np. zmiennoprzecinkowe funkcje matematyczne – *log( ), exp( ), sqrt( )* itp.).

Taki sposób podejścia do problemu programowania mikrokontrolerów jednoukładowych usprawnia proces budowy aplikacji. Daje jasny i przejrzysty kształt programu, a zarazem efektywność i szybkość spotykaną przy wykorzystaniu języków niskiego poziomu.

7. Wnioski

Dotychczasowe rozważania prowadzą do wniosku, że największą wydajnością, ale zarazem najniższym komfortem pracy podczas programowania, charakteryzują się aplikacje zaprojektowane całkowicie w asemblerze. Języki wysokiego poziomu dalece upraszczają

proces tworzenia aplikacji, ale nie do końca optymalizują kod wynikowy programu. Wobec powyższego często słusznym rozwiązaniem stają się hybrydowe metody projektowania oprogramowania z wykorzystaniem języków niskiego i wysokiego poziomu. Prawidłowo zaprojektowane i zapisane w języku asemblera aplikacje są bez wątpienia efektywne w działaniu (zwłaszcza w odniesieniu do problemów sterowania), jednak czytelność, stopień skomplikowania, długość kodu źródłowego oraz nakład pracy przeznaczony na zbudowanie programu, w wielu przypadkach dyskwalifikuje języki niskiego poziomu. Oferowane przez wiele firm kompilatory C/C++ dla mikrokontrolerów jednoukładowych, choć wydajne i wygodne, nie zawsze w dostatecznym stopniu, optymalizują kody wynikowe pod względem szybkości i rozmiaru. Wobec tego często słusznym rozwiązaniem są hybrydowe metody projektowania oprogramowania z wykorzystaniem języków niskiego i wysokiego poziomu.

Literatura

[1] Faison T.: *Borland C++ 4.5 programowanie obiektowe*, Oficyna wydawnicza READ ME, Warszawa 1996.

[2] Gałka Paweł, Gałka Piotr.: *Podstawy programowania mikrokontrolera 8051*, Zakłady Nauczania Informatyki „MIKOM", Warszawa 1995.

[3] http://www.keil.com

[4] Kruk S.: *Programowanie w języku assembler*, Wydawnictwo PLJ, Warszawa 1992.

[5] Pełka R.: Mikrokontrolery - architektura, programowanie, zastosowania, Wydawnictwa Komunikacji i Łączności, Warszawa 1999.