Projektowanie energooszczędnych układów elektronicznych.

Część VII : Oprogramowanie

Przez kilkadziesiąt lat rozwoju informatyki optymalizacja oprogramowania dotyczyła głównie wydajności i zwięzłości kodu. Optymalizacja pod kątem zużycia energii przez procesor jest zagadnieniem nowym, a zasady takiej optymalizacji dopiero powstają. W dziedzinie oprogramowania mikrokontrolerów problem jest o tyle trudniejszy, że sposoby optymalizacji muszą być powiązane z rozwiązaniami sprzętowymi, zastosowanymi w konkretnej rodzinie układów.

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 10142, pass: 5x7bu87r
- poprzednie części kursu

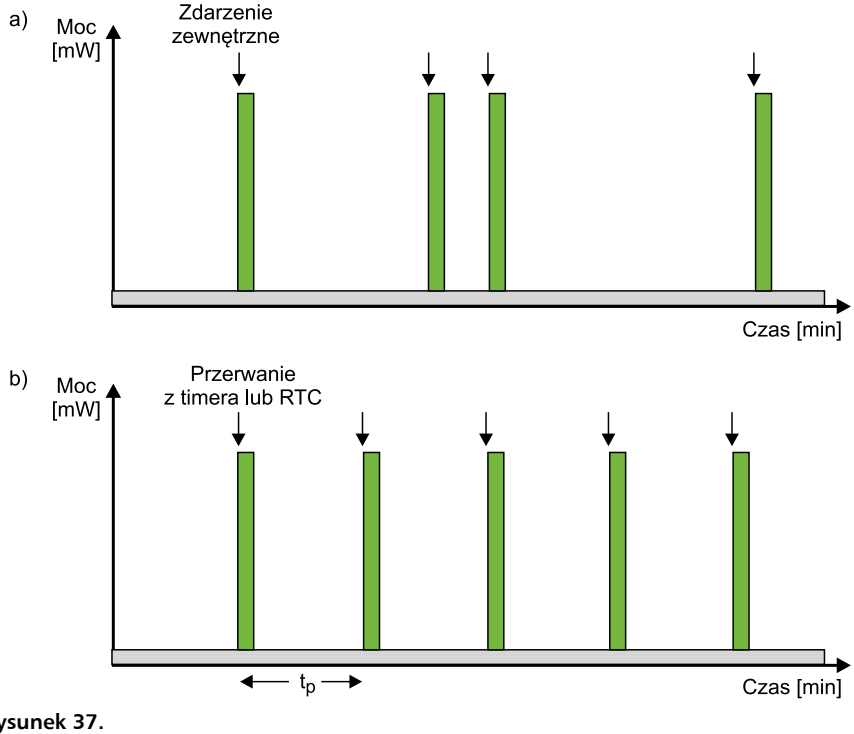
Podstawy tworzenia aplikacji energooszczędnych

Tradycyjne zasady tworzenia oprogramowania zaowocowały tym, że na całym świecie miliony mikrokontrolerów i mikroprocesorów przez 90% swojego czasu pracy nie zajmują się żadną użyteczną działalnością. W aplikacjach energooszczędnych należy zastosować nieco inne podejście do struktury programu, co ilustruje kilka przewrotnych definicji, które pewnie oburzą część profesjonalnych informatyków:

Pętla główna programu - stan, w którym mikrokontroler wykonuje w kółko te same zbędne operacje, czekając aż coś się wydarzy.

Odpytywanie urządzeń peryferyjnych (*polling*) – uzasadnienie istnienia pętli głównej poprzez udawanie, że procesor ma coś do zrobienia. Obowiązuje tu zasada: im większa częstotliwość odpytywania, tym większa pewność, że od poprzedniego razu nic się nie zmieniło.

Cykliczne odświeżanie stanu wyjść – kolejne uzasadnienie dla pętli głównej. Obowiązuje zasada: przy odpowiednio dużej częstotliwości odświeżania, zapisuje się do portów wielokrotnie te same wartości. Odświeżanie jest wygodą programisty: wystarczy napisać procedurę odpowiednio częstego przesyłania stanów zmiennych do portów wyjściowych, i można już operować tylko na zmiennych programu. Np. jeżeli dioda LED ma się świecić na stałe, to co za problem wysłać do niej stan aktywny 1000 razy na sekundę. Przecież CPU i tak nie ma nic innego do roboty.

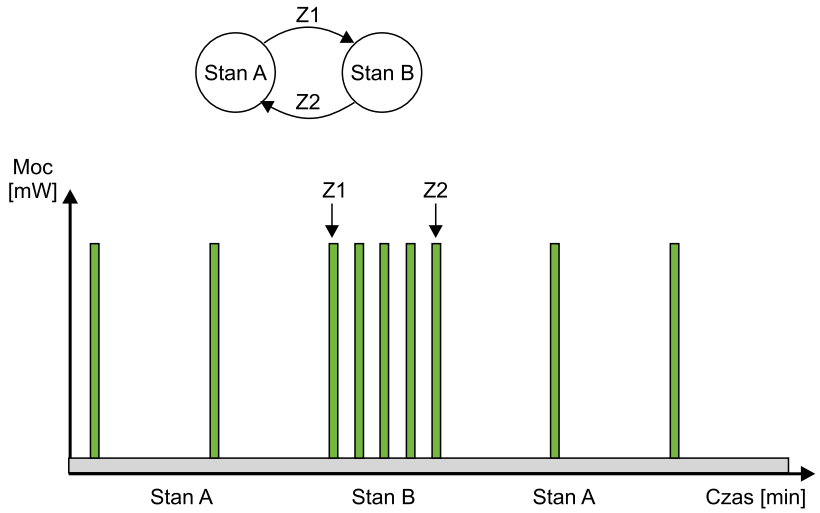


Rysunek 37.

Instrukcje skoków warunkowych i bezwarunkowych – według profesjonalnych programistów, używanie tych rozkazów świadczy o braku umiejętności programowania. Muszą być pętle, wywołania podprogramów i procedur. Oczywiście fakt, że procesor musi przeskakiwać tam i z powrotem do odległych obszarów pamięci w celu wykonania kilku bajtów kodu nie ma znaczenia, ma być elegancko i już.

Dobry program jest całkowicie niezależny od sprzętu – im mniejszy komputer, tym gorzej się sprawdza ta zasada. W mikrokontrolerach oprogramowanie jest zawsze powiązane ze sprzętem. Pisanie w języku wysokiego poziomu jest szybkie i efektywne, ale nie zapewnia optymalizacji energetycznej.

W aplikacjach energooszczędnych mamy do czynienia z programowaniem zdarzeniowym i cykliczną pracą mikrokontrolera. Podstawowym stanem mikrokontrolera jest stan uśpienia, z wyłączonym CPU i wszystkimi zbędnymi urządzeniami. Stan ten jest przerywany krótkimi okresami aktywności, uruchamianymi za pomocą systemu przerwań. Spełnienie warunku kilkuletniej pracy urządzenia na jednej baterii wymaga, aby stosunek czasu aktywności do czasu uśpienia dla najbardziej energooszczędnych mikrokontrolerów wynosił przynajmniej 1:50.



Rysunek 38.

W praktyce spotyka się proporcje 1:100, 1:1000 i więcej.

Tryby pracy cyklicznej

Istnieją dwa podstawowe tryby pracy cyklicznej : asynchroniczny i synchroniczny (**Rys. 37**). W trybie asynchronicznym wszystkie generatory zegarowe mikrokontrolera są wyłączone, a wybudzenie ze stanu uśpienia może aktywować wyłącznie zdarzenie zewnętrzne (zmiana stanu wejścia). W trybie synchronicznym pracuje jeden zegar (RTC, Watchdog) z układem czasowym generującym przerwania, a stany aktywności CPU pojawiają się w określonych odstępach czasu t_P. Tryb asynchroniczny jest bardziej oszczędny energetycznie, bo interwały czasowe między zdarzeniami mogą liczyć wiele godzin lub dni. Niestety jego zastosowanie jest ograniczone do tych aplikacji, w których ciągłe odmierzanie czasu nie jest konieczne.

W trybie synchronicznym oprogramowanie zarządzające przedziałami czasowymi pełni rolę nadrzędną w stosunku do innych procesów – czyli jest to miniaturowy system operacyjny RTOS. W celu zaoszczędzenia energii, stosuje się pracę synchroniczną ze zmiennym czasem t_P. System zarządzający pracuje wtedy jako maszyna stanów, której działanie ilustruje **Rys. 38**. W stanie A częstotliwość cykli aktywności mikrokontrolera jest mała, natomiast wystąpienie zdarzenia Z1 powoduje przejście do stanu B, ze znacznie większą częstotliwością budzenia. Zdarzenie Z2 powoduje powrót do stanu A. W identyczny sposób można zrealizować mieszany tryb asynchroniczno / synchroniczny. W stanie A (asynchronicznym) czas nie jest odmierzany w sposób ciągły, ale zdarzenie zewnętrzne powoduje przejście do trybu synchronicznego na określony interwał czasowy. Przykład: elektroniczny termometr lekarski. Naciśnięcie przycisku powoduje aktywowanie trybu synchronicznego i wykonywanie serii pomiarów temperatury. Po kilku minutach bezczynności urządzenie ponownie przechodzi do trybu asynchronicznego, oczekując na wciśnięcie przycisku.

Pomiar i obliczanie zużycia energii

Pomiar poboru prądu pobieranego przez mikrokontroler pracujący w trybie cyklicznym nie jest prostym zadaniem. Nie można po prostu włączyć miliamperomierza w obwód zasilania, potrzebny jest szybki rejestrator lub oscyloskop cyfrowy z pamięcią. Jeżeli mierzymy wartości na poziomie mA, to dodatkowym utrudnieniem jest możliwość zaburzenia wyniku poprzez prąd upływu obwodu pomiarowego. Niektórzy producenci oferują zestawy uruchomieniowe, wyposażone w układ pomiaru chwilowego i średniego poboru prądu. Tego typu pomiary zwykle wykonuje się na etapie testowania i optymalizacji aplikacji, natomiast pierwszym krokiem jest dokonanie szacunkowych obliczeń.

W celu oszacowania zużycia energii należy obliczyć średni pobór prądu I_S. W trybie synchronicznym ze stałym czasem powtarzania można wykorzystać wzór:

$$I_S = \frac{t_A}{t_A + t_U} \cdot I_A + \frac{t_U}{t_A + t_U} \cdot I_U$$

przy czym $t_A + t_U = t_P$ [okres powtarzania]

gdzie:

t_A , t_U - czas aktywności, czas uśpienia

I_A , I_U - prąd w stanie aktywności i w stanie uśpienia

Wygodniejsze może być bezpośrednie obliczenie zużycia energii z baterii, w następujący sposób:

Ilość cykli aktywności / uśpienia w ciągu godziny: $n = 3600 / t_P$

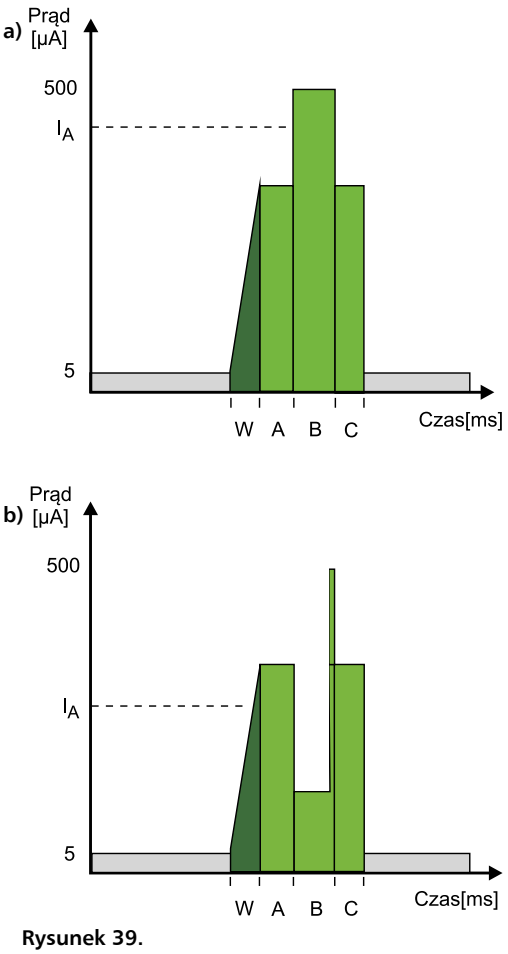
Pobór energii na godzinę dla stanu aktywności: $E1 = n \cdot t_A \cdot I_A$

Pobór energii na godzinę dla stanu uśpienia: $E2 = (3600 - n \cdot t_A) \cdot I_U$ lub $E2 = n \cdot t_U \cdot I_U$

Łączna energia zużyta w ciągu godziny wyniesie $E = E1 + E2$. Jeżeli wszystkie czasy będą wyrażone w sekundach, a prąd w mA, to otrzymamy godzinne zużycie energii w mAh.

Dla trybu asynchronicznego i mieszanego obliczenie jest trudniejsze, ponieważ należy najpierw oszacować statystyczną częstotliwość występowania stanów aktywności (na przykład w cyklu dobowym lub tygodniowym) i zsumować łączny czas ich trwania. Kolejnym krokiem jest zsumowanie czasów aktywności i wyliczenie proporcji t_A / T oraz $(T - t_A) / T$, gdzie T jest wybranym okresem analizy (doba, tydzień). Podstawiając je do podanych powyżej wzorów, można obliczyć odpowiednio średni prąd lub zużycie energii. Jeżeli dysponujemy wszystkimi danymi, to jest to zadanie na poziomie szkoły średniej, niewiele trudniejsze od obliczenia zużycia energii przez lodówkę.

Prąd w stanie uśpienia I_U dla wybranych warunków pracy można odczytać bezpośrednio z karty katalogowej, a czas uśpienia t_U wynika z założeń projektu. Oszacowanie prądu w stanie aktywności I_A jest najtrudniejszą częścią całego zadania. Należy wziąć pod uwagę prąd pobierany przez CPU oraz przez wszystkie aktywne układy peryferyjne i generatory zegarowe. Jeżeli aplikacja została zoptymalizowana pod kątem zużycia energii, to wartość prądu podczas cyklu aktywności ulega dużym wahaniom. Przykłady zmienności poboru prądu dla dwóch aplikacji realizujących zbliżone zadania



Rysunek 39.

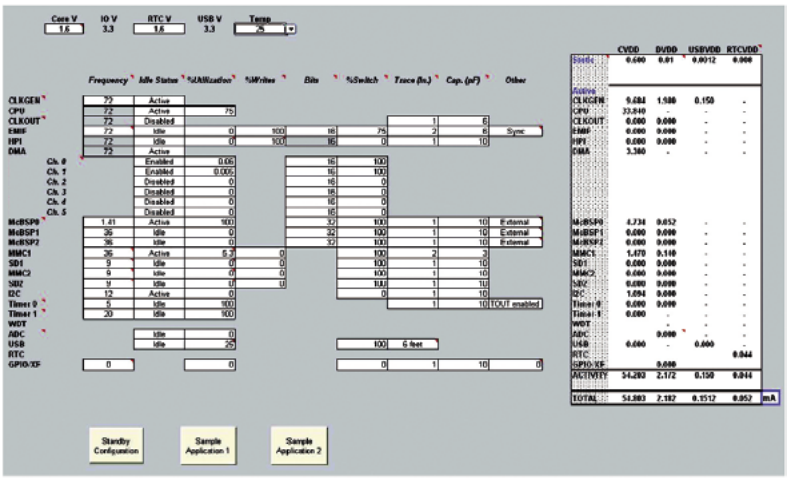
przedstawia **rysunek 39**. W obu aplikacjach cykl aktywności składa się z fazy wybudzania (W), fazy przygotowawczej (A), fazy wykonywania pomiarów (B) i fazy końcowej (C). Założono, że w fazie wybudzania prąd narasta liniowo, co pozwala przyjąć, że prąd w tej fazie wynosi ½ I_A. W takim przypadku można uwzględnić zużycie energii w fazie wybudzania poprzez dodanie do łącznego czasu aktywności połowy czasu wybudzania mikrokontrolera. Zakładamy, że w fazach A i C urządzenia peryferyjne nie pracują, czyli prąd ma trzy składowe: CPU i dwa generatory zegarowe (główny + RTC). W fazie B urządzenie wykonuje serię pomiarów analogowych napięcia. W przykładzie (a) każdy wynik jest na bieżąco przeliczany przez CPU i zapisywany w pamięci EEPROM. W przykładzie (b) rdzeń mikrokontrolera w fazie B nie pracuje, a „surowe" wyniki z przetwornika A/C są zapisywane w buforze RAM za pomocą funkcji DMA. Po zakończeniu serii pomiarów obliczana jest wartość średnia i tylko ten wynik zapisuje się w EEPROM. Wartość średnia prądu I_A, obliczona dla całego cyklu aktywności jest dużo niższa w przykładzie (b). Jak widać, przyjęcie określonego algorytmu pracy i odpowiednie wykorzystanie możliwości sprzętowych pozwala osiągnąć bardzo duże oszczędności energii.

Czas aktywności mikrokontrolera t_A jest sumą czasów wykonywania procedur W, A, B i C (do bilansu energii przyjmujemy ½ W). Ręczne obliczenie t_A jest żmudną operacją, ale większość kompilatorów i środowisk IDE posiada odpowiednie narzędzia do obliczania czasu wykonywania programu.

Niektórzy producenci mikrokontrolerów oferują proste programy narzędziowe – arkusze kalkulacyjne, przeznaczone do obliczania poboru prądu lub czasu eksploatacji baterii przy zadanych warunkach początkowych (napięcie zasilania, częstotliwość taktowania, aktywne urządzenia peryferyjne). Na **rysunku 40** przedstawiono ekran roboczy programu *Power Estimation Spreadsheet* firmy Texas Instruments. Program ten obsługuje procesory DSP z rodzin *TMS320* oraz *Sitara*, niestety nie jest dostępny dla MSP430. Inne programy o zbliżonych funkcjach to: *Battery Life Estimator* przeznaczony dla mikrokontrolerów z modułem radiowym firmy Silicon Labs, oraz *XLP Battery Life Estimator* firmy Microchip, dedykowany dla mikrokontrolerów serii *nanoWatt XLP*.

Wybór częstotliwości zegarowej

W mikrokontrolerach pracujących w trybie cyklicznym należy rozdzielić pojęcia poboru mocy i zużycia energii. W przypadku dużej częstotliwości taktowania chwilowy pobór mocy jest większy, ale zadania obliczeniowe zostaną zrealizowane w krótszym czasie niż przy małej częstotliwości taktowania. Zużycie energii w całym cyklu aktywności



Rysunek 40.

będzie prawie identyczne dla obu przypadków. Daje to konstruktorowi większą swobodę wyboru częstotliwości zegarowej, niż w mikrokontrolerach pracujących w trybie ciągłym. Jednakże w układach z zasilaniem bateryjnym pojawia się inne ograniczenie: przy wysokich częstotliwościach taktowania, chwilowy pobór prądu jest duży, co zmniejsza efektywną pojemność baterii. Z punktu widzenia oszczędności energii ważna jest nie tyle wartość bezwzględna częstotliwości zegarowej, co dopasowanie częstotliwości taktowania rdzenia i układów peryferyjnych. Przykład: pobranie 1 bajtu danych z zewnętrznej pamięci z interfejsem I²C wymaga przesłania 32 bitów (bity sterujące + adres układu + adres komórki + dane). Przy częstotliwości SCL = 400 kHz czas transmisji wyniesie 80 ms. Dla CPU taktowanego zegarem 16 MHz jest to bardzo długi czas oczekiwania na dane, natomiast przy taktowaniu CPU częstotliwością 32 kHz są to niecałe 3 cykle zegara. Może też wystąpić sytuacja odwrotna: szybki przetwornik A/C zakończył pracę i wystawił sygnał przerwania. Jednak CPU nie może odczytać danych i uśpić przetwornika, ponieważ obsługuje inne przerwanie o wyższym priorytecie. Optymalizacja energetyczna aplikacji może wymagać wielokrotnych zmian częstotliwości taktowania rdzenia i układów peryferyjnych w trakcie wykonywania programu, w celu zminimalizowania okresów bezczynności współpracujących układów. Częstotliwość taktowania można zmieniać poprzez wybór odpowiedniego generatora zegarowego lub zmianę jego częstotliwości. Jeżeli zmiany są wykonywane w czasie pracy programu, to korzystniej jest używać do tego celu programowanych dzielników częstotliwości (preskalerów). Zarządzanie taktowaniem poszczególnych modułów mikrokontrolera w czasie rzeczywistym można zrealizować na dwa sposoby:

- Przypisanie na stałe odpowiednich częstotliwości taktowania do poszczególnych procedur lub fragmentów kodu. Metoda prosta w realizacji, ale może być kłopotliwa w przypadku aplikacji wielozadaniowych.
- Implementacja mini-systemu operacyjnego RTOS, wyposażonego w funkcję monitorowania aktywności CPU i urządzeń peryferyjnych. System na bieżąco dostosowuje częstotliwości taktowania zasobów sprzętowych do potrzeb aplikacji. Metoda ta może być skuteczna w bardziej rozbudowanych programach, gdzie korzyści wynikające z optymalizacji będą większe niż dodatkowa moc obliczeniowa potrzebna do działania RTOS.

Zasady pisania programów

Chociaż tworzenie energooszczędnych aplikacji jest zagadnieniem nowym, to pojawiły się już pierwsze zasady i zalecenia dla programistów:

- **Konfiguracja portów we/wy**. W większości przypadków linie portów utrzymują swoje stany logiczne po przejściu mikrokontrolera w stan uśpienia. Dlatego przed uśpieniem systemu należy tak skonfigurować porty, aby zużycie energii było jak najmniejsze. Niewykorzystane piny najlepiej skonfigurować jako wyjścia cyfrowe. Wszystkie piny wyjściowe (używane i nieużywane) wyposażone w wewnętrzne lub zewnętrzne rezystory podciągające muszą być ustawione w taki stan logiczny, aby przez rezystory nie płynął prąd. Piny skonfigurowane jako wejścia cyfrowe powinny być zabezpieczone przed pojawieniem się nieokreślonego stanu logicznego. Jeżeli piny wejściowe nie muszą przyjmować przerwań w stanie uśpienia, to korzystnym rozwiązaniem jest skonfigurowanie ich jako wejść analogowych. Uzyskuje się w ten sposób znacznie mniejsze prądy upływu.
- **Rozmiar pamięci**. W każdej rodzinie mikrokontrolerów dostępne są układy o różnej wielkości pamięci RAM i Flash. Im większa pamięć, tym bardziej rozbudowane dekodery adresowe i większa

```
1  'Program przed optymalizacją
2  for i:=1 to N do B[i]:=f(A[i]);
3  for i:=1 to N do C[i]:=g(D[i]);
4
5  'Program po optymalizacji
6  for i:=1 to N do
7  begin
8      B[i]:=f(A[i]);
9      C[i]:=g(D[i]);
10 end;
11
```

Rysunek 41.

pojemność obciążenia na szynach danych i adresowych. Dlatego też nie należy wybierać układów z pojemnością pamięci znacznie przekraczającą potrzeby aplikacji. Z tych samych powodów mikrokontrolery z kilkoma wewnętrznymi szynami danych / adresów będą w cyklach dostępu do pamięci pobierać mniejszy prąd niż układy z jedną wspólną szyną. Dzieje się tak dlatego, że w przypadku kilku szyn, ilość wejść bramek dołączonych do każdej z nich jest mniejsza. Przy zmianie stanu szyny przeładowywana jest mniejsza pojemność obciążenia. Oczywiście przy ograniczaniu wielkości pamięci trzeba zachować umiar. Przykład: przewidywany rozmiar aplikacji 7 kB, a dostępne są mikrokontrolery z pamięcią 8, 16, 32 i 64 kB. Wybór 16 kB zapewni wystarczającą rezerwę, a większa pamięć będzie marnowaniem energii.

- **Korzystanie z pamięci i rejestrów**. Operacje arytmetyczne i logiczne wykonywane na wewnętrznych rejestrach CPU kosztują znacznie mniej energii niż wykonywane na zmiennych przechowywanych w pamięci RAM. Dlatego też warto tak budować kod programu, aby zminimalizować ilość przesłań do / z pamięci. Na **rysunku 41** przedstawiony jest przykład przetwarzania zmiennych tablicowych. W pierwotnej wersji kodu, tablica B jest zbyt duża, aby mogła być przechowywana w rejestrach, konieczne są transfery wartości pośrednich do / z pamięci RAM. Przebudowanie pętli pozwala zastąpić 2N cykli dostępu do pamięci przez mniej kosztowne energetycznie przesłania między rejestrami CPU. Dodatkową zaletą jest zwolnienie N komórek pamięci RAM. Z tych samych względów należy unikać najbardziej złożonych trybów adresowania pośredniego, w których pobranie wartości zmiennej wymaga kilku cykli dostępu do pamięci. W niektórych typach mikrokontrolerów, stosowanie się do tych zaleceń oznacza ograniczenie używania instrukcji wykonywanych w dwóch lub trzech cyklach zegarowych, co dodatkowo poprawia efektywność kodu.
- **Kod programu w pamięci RAM**. Wykonywanie kodu programu załadowanego do pamięci RAM pozwala zaoszczędzić ok. 40% energii w porównaniu z programem wykonywanym z pamięci Flash lub EPROM. Do niedawna nie było to możliwe, ale obecnie nawet 8-bitowe mikrokontrolery często dysponują 1 – 4 kB RAM, a w systemach 32-bitowych standardem jest 16 – 64 kB pamięci RAM. Jeżeli mikrokontroler oferuje taką możliwość, to warto wykonywać w pamięci RAM przynajmniej najczęściej używany fragment kodu programu.
- **Instrukcje skoków i wywołań podprogramów**. Podczas wykonywania skoku z obszaru początkowego do obszaru końcowego pamięci programu, następuje jednoczesna zmiana stanu kilkuset bramek w dekoderze adresów oraz prawie wszystkich linii adresowych. Impuls prądowy podczas skoku do odległego obszaru pamięci jest znacznie większy, niż podczas pobierania instrukcji z kolejno następujących adresów. Podczas wywołań i powrotów z podprogramów dochodzą jeszcze sekwencje dostępu do pamięci RAM – zapis / przywrócenie adresu powrotu i stanu rejestrów. W programach energooszczędnych należy w miarę możliwości ograniczać ilość wywoływanych podprogramów. W pierwszej kolejności powinny być eliminowane podprogramy składające się z kilku bajtów kodu, znajdujące się w obszarze pamięci odległym od miejsca wywołania. Co prawda powtarzanie tej samej sekwencji rozkazów w wielu miejscach programu spowoduje zwiększenie objętości kodu, ale poprawi szybkość wykonania i efektywność energetyczną.
- **Używanie języków wysokiego poziomu**. Współczesne kompilatory języków C/C++ dysponują bogatym zestawem funkcji i procedur bibliotecznych, co nawet w przypadku mikrokontrolerów umożliwia tworzenie kodu źródłowego oddzielonego od warstwy sprzętowej. Wygenerowanie kodu wynikowego, przeznaczonego dla konkretnej rodziny mikrokontrolerów jest zadaniem kompilatora, a na razie żaden dostawca kompilatorów nie zadeklarował, że jego produkt generuje kod zoptymalizowany pod kątem zużycia energii. Pisanie programów energooszczędnych wymaga bardzo silnego powiązania programu ze sprzętem, czasem nawet napisania fragmentów kodu w asemblerze. Korzystanie z zaawansowanych procedur i funkcji bibliotecznych powoduje utratę kontroli nad sposobem realizacji zadań przez mikrokontroler. Dla programistów przyzwyczajonych do operowania klasami i obiektami, jest to duży krok wstecz. Jako pocieszenie można przyjąć, że nawet w XXI wieku prawdziwe dzieła sztuki wykonywane są ręcznie, przy pomocy prostych narzędzi.
- **Tablice funkcji**. Dzięki zastosowaniu tablic funkcji można uzyskać duże oszczędności energii i wydajności. Przyjmijmy jako przykład prostą funkcję nieliniową :

$$Y = 256 / (255 - 0.7 * X)$$

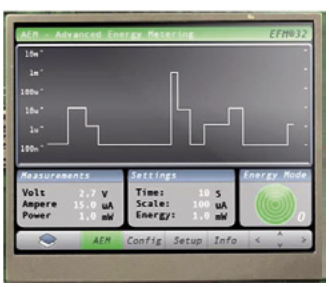
Nawet jeżeli zmienne X i Y są 8-bitowe, to do obliczenia wyniku trzeba użyć arytmetyki 16-bitowej. Stablicowanie tej funkcji „kosztuje” tylko 256 bajtów pamięci Flash, a zysk energetyczny będzie znaczący. Jeżeli argumentem funkcji jest 10-bitowy pomiar z przetwornika A/C, to tablica zajmie 2 kB pamięci przy 16-bitowej rozdzielczości wyniku. Im bardziej złożona formuła matematyczna, tym większe korzyści ze stosowania tablic.

Warto zauważyć, że powyższe zalecenia są zaprzeczeniem metody przetwarzania DSP. Typowy procesor DSP dysponuje wydajną jednostką centralną, rozbudowanym programem obliczeniowym w pamięci Flash, oraz dużą pamięcią RAM do przechowywania danych i wyników. W mikrokontrolerze energooszczędnym mamy program użytkownika w pamięci RAM, tablice funkcji w pamięci Flash, a wymagana wydajność obliczeniowa spada prawie do zera.

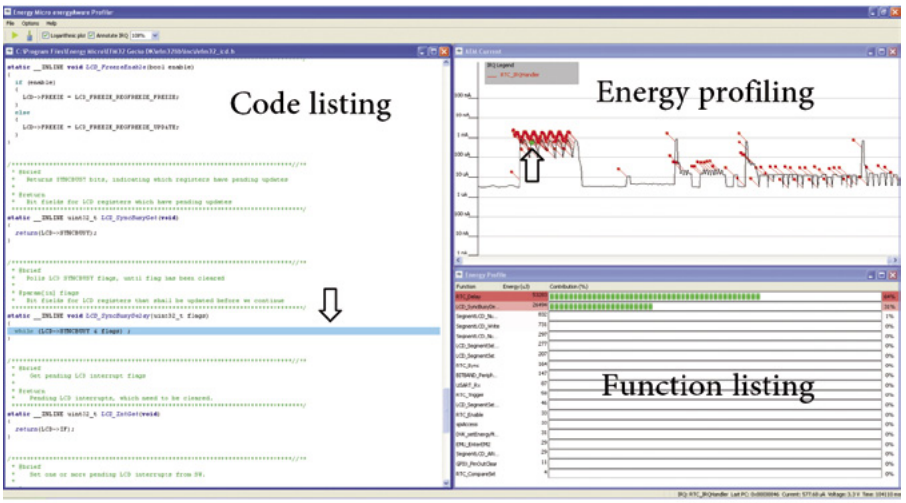
Optymalizacja energetyczna programu

Klasyczna optymalizacja zwięzłości i czasu wykonywania kodu nie jest tutaj wystarczająca. O ile krótszy czas wykonania kodu zwykle jest korzystny energetycznie, to zmniejszanie objętości kodu może dać wynik przeciwny do zamierzonego. Wykonane na początku projektu szacunkowe obliczenia zużycia energii także wymagają praktycznej weryfikacji. Optymalizacja energetyczna w fazie uruchamiania i testowania programu określana jest w literaturze anglojęzycznej terminem *Energy debugging* lub *Power debugging*. Pojawiają się już pierwsze narzędzia do optymalizacji energetycznej oprogramowania.

Narzędzia do optymalizacji wchodzą w skład zestawu startowego mikrokontrolera EFM32 firmy Energy Micro. Płytka uruchomieniowa jest wyposażona w sprzętowy moduł pomiaru zużycia energii AEM (*Advanced Energy Monitoring*), zawierający oddzielny mikrokontroler z wyświetlaczem LCD i łączem USB. Na ekranie LCD wyświetlany jest wykres poboru prądu w funkcji czasu oraz wartości liczbowe napięcia, prądu i mocy (**Rys. 42**). Bardziej szczegółową analizę umożliwia współpracujący z modułem AEM program *Energy Aware Profiler*. Program otrzymuje wyniki pomiaru prądu oraz próbki stanu licznika rozkazów CPU, a następnie synchronizuje chwilowy pobór prądu z aktualnie wykonywanym kodem



Rysunek 42.



Rysunek 43.

Fotografia 44.

programu. Ekran roboczy programu składa się z trzech okien (**Rys. 43**). Okno *Energy Profiling* prezentuje pobór prądu mikrokontrolera w funkcji czasu. Kliknięcie na wybrany punkt wykresu powoduje wyświetlenie odpowiedniego fragmentu kodu źródłowego w oknie *Code Listing* oraz tabeli zużycia energii przez aktywne procedury w oknie *Function Listing*. Firma IAR Systems zapowiada wprowadzenie podobnego oprogramowania w nowej wersji popularnego środowiska *IAR Embedded Workbench*.

Firma STMicro oferuje możliwość pomiaru prądu w zestawach startowych dedykowanych dla mikrokontrolerów STM8L15x. Rysunek 44 przedstawia najprostszy zestaw STM8L-Discovery, w którym pobór prądu może być przedstawiony w postaci cyfrowej na wyświetlaczu LCD. System pomiarowy składa się z przetwornika prąd / napięcie oraz sprzętowo sterowanego układu próbkująco-pamietającego z kondensatorem pamięciowym o pojemności 1 mF. Dzięki temu mikrokontroler może sam zmierzyć własny pobór prądu w ostatnio wykonanym cyklu aktywności. Do zestawu jest dołączone przykładowe oprogramowanie realizujące funkcje pomiaru i wyświetlania prądu. System ten nie zapewnia takich możliwości i komfortu pracy jak opisany wcześniej, ale jest znacznie prostszy i tańszy. Firma STMicro publikuje schemat zastosowanego rozwiązania, co może być inspiracją dla konstruktorów do budowy własnych układów monitorujących.

Jacek Przepiórkowski