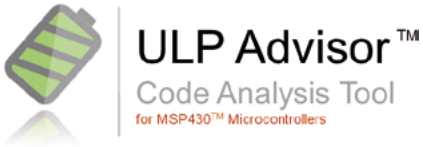
Poradnik oszczędzania energii



Wiele urządzeń zasilanych z baterii ma długi czas eksploatacji, niekiedy nawet do 20 lat. Sam staranny dobór podzespołów to jednak nie wszystko. Aby było możliwe osiągniecie założonej funkcjonalności, jest konieczne odpowiednie napisanie oprogramowania, odpowiednio korzystającego z możliwości mikrokontrolera i jego zasobów sprzętowych.

W nowoczesnych układach CMOS pobór prądu w stanie aktywnym jest powodowany ładowaniem wewnętrznych pojemności elementów przełączających w każdym cyklu zegarowym. Istnieje wiele możliwości zmniejszenia zapotrzebowania na energię, wśród których warto wymienić:

1. Zmniejszenie pojemności wewnętrznych
   Układ scalony może być wykonany za pomocą procesu technologicznego o mniejszych elementach przełączających. Jednak ich użycie z reguły zwiększa prądy upływu spowodowane cieńszą warstwą izolacji, a tym samym mniejszymi rezystancjami.
2. Zwiększenie wydajności CPU
   Wydajna jednostka centralna pozwala na szybkie wykonywanie instrukcji. Krótsze czasy wykonywania instrukcji pozwalają na spędzenie przez CPU większej ilości czasu w trybie niskiego poboru mocy i przez to zmniejszają zużycie energii. Dlatego też wydajność CPU jest bardzo ważnym parametrem energooszczędnych mikrokontrolerów. Niemniej, mogą tu występować ograniczenia, ponieważ nie każda operacja wykonywana przez CPU ma odpowiadającą jej instrukcję i czasami na pojedyncze operacje muszą składać się ciągi instrukcji. Niemniej jednak i w takiej sytuacji jest pomocne wydajne CPU.
3. Zmniejszenie liczby używanych bramek
   Przemyślany podział zadań pomiędzy zoptymalizowane peryferia (np. timery, DMA itd.) może zmniejszyć liczbę pracujących elementów logicznych (bramek) i ma duży wpływ na oszczędność energii. Również tutaj występują limity, ponieważ nie dla każdego typu zadania dostępna jest dedykowana jednostka wykonawcza.
4. Niższe napięcie zasilania
   Obniżenie napięcia zasilającego wywiera bardzo pozytywny wpływ na zmniejszenie zużycia energii. Niestety, również tutaj szybko osiąga się granicę możliwości, ponieważ zmniejszenie napięcia zasilającego wymusza obniżenie częstotliwości taktowania.
5. Odłączanie bloków funkcjonalnych mikrokontrolera
   Technologia „bramkowania mocy" (*Power Gating*) pozwala na całkowite odłączenie zasilania nieużywanych bloków funkcjonalnych mikrokontrolera, co powoduje znaczne ograniczenie prądów upływu. Niestety, również tutaj szybko osiąga się granice możliwości spowodowane prądem upływu kluczy odłączających. Upływ ten jest oczywiście znacznie mniejszy, niż prąd pobierany przez odłączany blok.

W artykule ograniczymy się do omówienia technik ograniczania poboru energii silnie uzależnionych od oprogramowania.

Do środowiska programistycznego *Code Composer Studio* (CCS) od wersji 5.2 jest dołączone użyteczne narzędzie — program ULP Advisor. Instruuje on programistę, jak tworzyć bardziej wydajny kod zwracając mu uwagę na niepowtarzalne, charakterystyczne cechy mikrokontrolera MSP430, które on może w pełni wykorzystywać. ULP Advisor to program przeznaczony do analizy programu dla mikrokontrolera, który sprawdza go w czasie kompilowania pod kątem kryteriów zapewniających niski pobór mocy. Jeśli jakaś część programu narusza te zasady, narzędzie wyróżnia odpowiednie miejsca w programie źródłowym za pomocą komunikatów i uwag. Oprócz opisu zasad programista otrzymuje wartościowe informacje na temat sposobu optymalizowania kodu.

Używając przykładu chcielibyśmy przybliżyć i zaprezentować, jak w praktyce wygląda praca z programem ULP Advisor. Użyjemy w tym celu mikrokontrolera MSP430G2553 z rodziny *MSP430 Value Line*. Przykładowa aplikacja, w której wykorzystamy płytkę *LaunchPad*, powinna mieć przycisk i interfejs szeregowy do transmisji danych do zewnętrznego odbiornika (np. komputera lub innego mikrokontrolera). Tuż po włączeniu zasilania aplikacja pozostaje nieaktywna. Po naciśnięciu przycisku, aplikacja aktywuje się. W okresach sekundowych jest mierzona wewnętrzna temperatura mikrokontrolera i przeliczana na stopnie Celsjusza i Fahrenheita. Ponadto, jest obliczana wartość średnia z ostatnich 16 pomiarów. Po zapamiętaniu wyników, dane transmitowane są za pomocą interfejsu szeregowego. Ponowne naciśnięcie przycisku kończy pracę, a aplikacja powraca w stan spoczynku i czeka na kolejne naciśnięcie przycisku. Pętlę główną opisywanego programu pokazano w uproszczeniu na **rysunku 1**.

Zastosowanie trybu niskiego poboru energii

Po utworzeniu nowego projektu, napisaniu programu zgodnie z podanym algorytmem i bezbłędnym skompilowaniu kodu źródłowego, środowisko programistyczne wyświetla porady programu ULP Advisor w oknie *Problems*.

Komunikaty wyświetlane są w kategorii *Info*. Pierwszy wpis: *„(ULP 1.1) Detected no uses of low power mode state changes using LPMx or _bis_SR_register() or __low_power_mode_x() in this project"*. Za pomocą hiperłącza można przejść na zintegrowaną z CCS stronę Wiki ze szczegółowymi informacjami na temat zmian proponowanych przez ULP Advisor 1.1. Uzyskujemy tam informacje, jakie sytuacje mogą wystąpić w razie nieprzestrzegania ustalonych zasad oraz wyjaśnienia, dlaczego program ULP Advisor wyświetla daną poradę i w jaki sposób można zaradzić problemom. W powyższym wypadku nie zastosowaliśmy w naszym programie żadnych trybów niskiego poboru mocy, tylko permanentnie eksploatowaliśmy mikrokontroler w trybie aktywnym. Zwykle program powinien być zoptymalizowany, tak aby zapewnić pracę w trybie energooszczędnym. Oznacza to wyłączenie nieużywanych komponentów mikrokontrolera i w efekcie znacznie obniżenie poboru mocy. Powszechne jest obniżenie poboru prądu 1000 razy lub nawet więcej. Często stosuje się technikę „bramkowania mocy" (*Power Gating*). Program powinien również wykorzystywać przerwania. Umożliwia to wydłużenie czasu uśpienia mikrokontrolera i skrócenie czasu prac w trybie aktywnym. Dopiero gdy wystąpi przerwanie CPU jest aktywowane, za-

łączane są odpowiednie układy peryferyjne i jest wykonywany kod obsługi przerwania, a następnie mikrokontroler zostaje uśpiony do czasu następnego zdarzenia. Poprzez maksymalizację czasów spoczynku oraz minimalizację czasów aktywności, zwiększa się wydajność energetyczna i do minimum zmniejsza średnie zużycie prądu.

Jeśli chodzi o wykorzystanie przerwań, przykładowy kod wykazuje szereg słabych punktów. Poniżej na liście porad ULP Advisor znajduje się kolejna uwaga oraz powtórka tego, czego się już nauczyliśmy. Komunikat brzmi *„(ULP 3.1) Detected flag polling using ADC10IFG. Recommend using an interrupt combined with enter LPMx and ISR."* Podwójnym kliknięciem na treści komunikatu przechodzimy bezpośrednio do wiersza w programie źródłowym, w którym rzeczywiście widzimy programowe sprawdzanie flagi sygnalizującej zakończenie konwersji przetwornika A/D. ULP Advisor sugeruje uśpienie procesora i wybudzanie go za pomocą przerwania wywołanego zakończeniem konwersji przetwornika A/D.

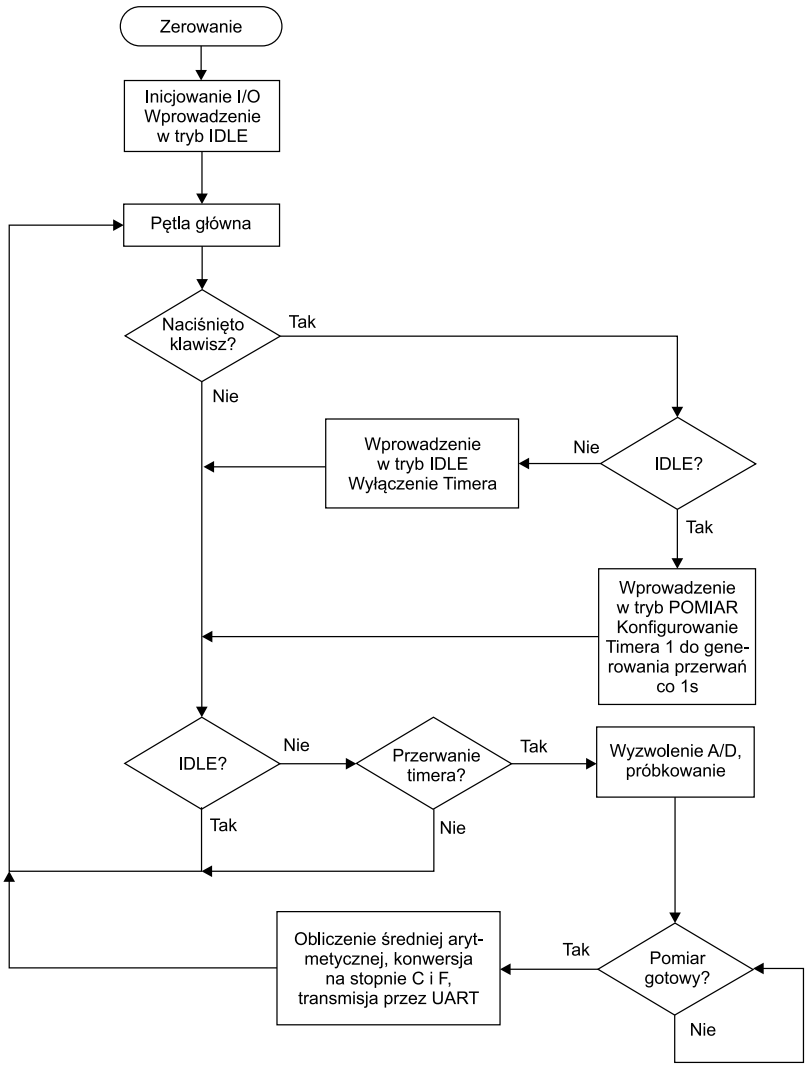
Architektura programu sterowana przerwaniami

Potencjalnymi źródłami przerwań w naszym przykładzie jest sygnał z wejścia GPIO pochodzący od przycisku, timer sekundowy i zidentyfikowany przez program ULP Advisor przetwornik A/DC. Po zmianie kodu mikrokontroler jest wprowadzany w stan uśpienia bezpośrednio po zainicjowaniu i skonfigurowaniu niezbędnych urządzeń peryferyjnych oraz ich przerwań. W stanie spoczynkowym zużycie prądu jest minimalne do czasu wybudzenia układu zdarzeniem przerwania. Sztuką jest przy tym maksymalne skrócenie czasu obsługi przerwania, tak aby skrócić czasy oczekiwania i reakcji. W tym przykładzie w procedurach obsługi przerwań jest ustawiany jedynie odpowiednia flaga stanu dla programu głównego, a stan spoczynku jest dezaktywowany przy wyjściu z procedury obsługi przerwania. Przetwarzanie przynależnego zadania odbywa się w programie głównym. Za pomocą flagi stanu można dowiedzieć się, co jest aktualnie do wykonania. Po wykonanej pracy pętla główna zamyka się, a mikrokontroler powraca w stan spoczynkowy. Na **rysunku 2** pokazano optymalny przebieg programu.

Przenoszenie zadań na inteligentne komponenty peryferyjne

W przykładzie do obsługi przycisku jest używana programowa pętla opóźnienia. Program ULP Advisor sygnalizuje to wskazaniem na zasadę *„(ULP 2.1) Leverage timer modules for delay loops"*. Obsługa przycisku może odbywać się w sposób energooszczędny przy użyciu timera i odpowiednich trybów niskiego poboru mocy. W mikrokontrolerze MSP430 jednostka CPU znajduje się w stanie spoczynkowym, a taktowany jest tylko timer. Dzięki temu przełączają się tylko obwody, które rzeczywiście potrzebne są do realizacji zadania. Po odliczeniu czasu timerem przerwanie budzi jednostkę centralną mikrokontrolera (CPU) z trybu energooszczędnego, co powoduje kontynuowanie obsługi procesów. Timer jest używamy do dwóch zadań:

- Obsługi przycisku.



Rysunek 1.

Listing 1. Funkcja *ProcessCurrentSample* używająca wskaźników do pobrania argumentów

```
//  Process samples, calculate the temperatures
void ProcessCurrentSample(unsigned short smpl, MEASUREMENT_INFO_T* x)
{
  // Ringbuffer definition to hold values for average calculation
  #define RINGBUFFERSIZE 16
  static unsigned short RingBuffer[RINGBUFFERSIZE];
  static unsigned char Idx = 0;
  unsigned short i;
// add new value to ring buffer, increment buffer index
  RingBuffer[Idx++] = smpl;
  if(Idx >= RINGBUFFERSIZE)  //handle buffer overflow
  {
    Idx = 0;  // roll over index when buffer is full
  }
  x->rs.A = smpl;
  // Temperature in Celsius
  // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
  x->rs.B = ((smpl - 673) * 423) >> 10;
  // Temperature in Fahrenheit
  // oF = ((A10/1024)*1500mV)-923mV)*1/1.97mV = A10*761/1024 - 468
  x->rs.F = ((smpl - 630) * 761) >> 10;
  // calculate average value
  x->rs.aA = 0;  // set to 0
  for(i = RINGBUFFERSIZE; i > 0; i--)
  {
    x->rs.aA += RingBuffer[i];  // accumulate stored values
  }
    x->rs.aA = x->rs.aA >> 4;  // divide by 16
  // finally copy current record to transmit buffer
  memcpy(tr.b, x->b, MEASUREMENT_INFO_SIZE);
}  // ProcessCurrentSample
```

- Do wyzwolenia pomiaru za pomocą przetwornika A/D. Używając mikrokontrolera MSP430 można posunąć się jeszcze o krok dalej i całkowicie pominąć wykonywanie kodu obsługi przerwania. Sygnał wyjściowy z timera może być użyty do wyzwalania przetwarzania A/D, bez konieczności wybudzania CPU z trybu uśpienia.

Konfigurowanie nieużywanych portów I/O

Po restarcie linie portów MSP430 pracują jako wejściowe. Ponieważ te wejścia mają wysoką impedancję, nie są odporne na sygnały zakłócające, jak np. od napięcia zasilającego 50 Hz czy na skutek bliskiej pracy świetlówek. W takim wypadku wejścia ciągle ulegałyby przełączeniom co zwiększałyby zużycie prądu. W układach z dużą liczbą wyprowadzeń może to oznaczać pobór nawet kilku mA. Środkiem zaradczym jest przełączenie nieużywanych portów w tryb wyjściowy lub ustalenie stanu wejściowego za pomocą zewnętrznego rezystora podciągającego (*pull up*) lub obniżającego (*pull down*). Zależnie od liczby dostępnych portów, program ULP Advisor generuje mniej lub więcej komunikatów z powodu naruszenia zasady *„(ULP 4.1) Terminate unused GPIOs”*.

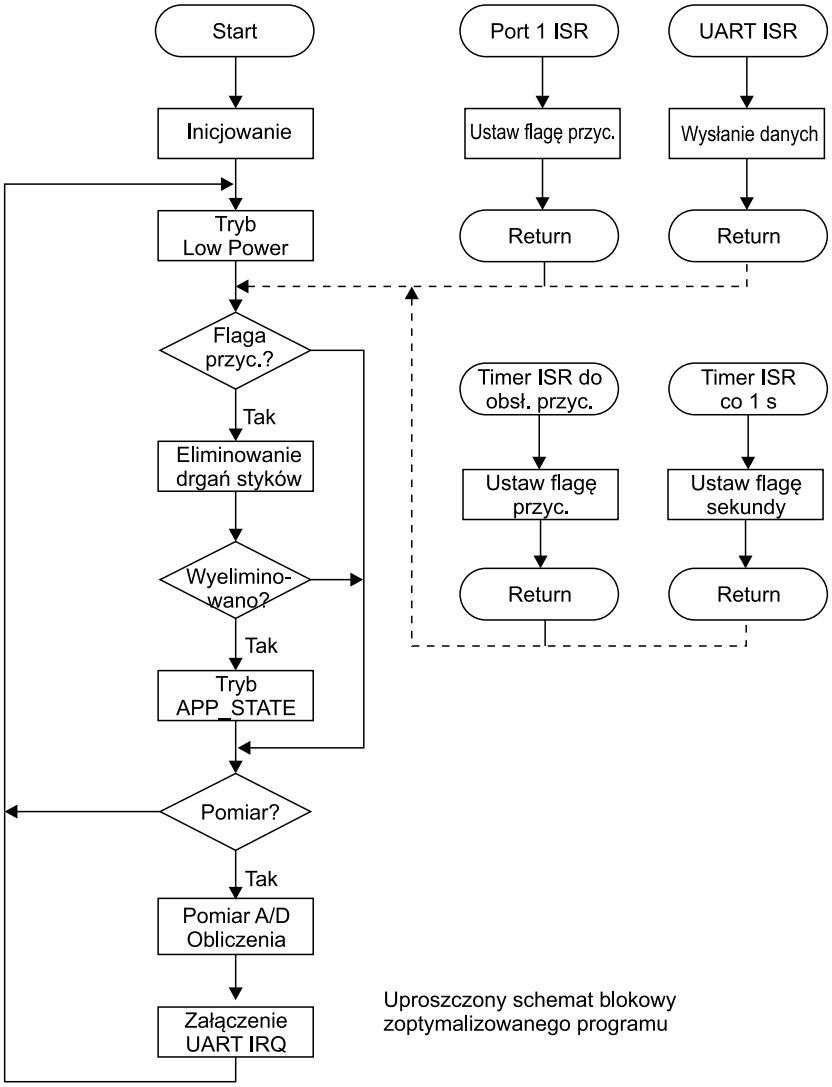
Mikrokontroler użyty przez nas w przykładzie ma 3 porty. Funkcja *Setup_Ports* służy do zainicjowania trybu ich pracy.

Wykorzystanie zmiennej lokalnej

Jeśli zadeklarowano zmienne globalne używane w czasie wykonywania programu tylko w kontekście jednej funkcji, ULP Advisor wyświetla komunikat *„(ULP 7.1) Use local instead of global variables where possible”*. Kompilator C zawsze będzie próbował optymalizować zmienne lokalne umieszczając je w rejestrach CPU. Redukuje to liczbę dostępów do pamięci, dzięki czemu zużywa się mniej prądu. Liczba rejestrów dostępnych dla kompilatora jest jednak ograniczona. Powyżej pewnego maksymalnej liczby zmiennych lokalnych kompilator C zaczyna korzystać ze stosu. W takim wypadku, lepiej będzie powrócić do części zmiennych zadeklarowanych globalnie, gdyż operacje zakładania i porządkowania elementów stosu wymagają dodatkowych cykli maszynowych i powodują nadmierne zużycie energii.

Przekazywanie dużych obszarów danych za pomocą referencji

W przypadku przekazywania do funkcji dużych ilości danych należy korzystać ze wskaźników. Kompilatory C tworzą lokalną kopię argumentów funkcji przekazywanych bezpośrednio. Wymaga to użycia większej ilości pamięci RAM oraz powiększa kod programu. Zamieszczona na **listingu 1** funkcja *ProcessCurrentSample* wykorzystuje wskaźnik do przekazywania argumentów.



Rysunek 2.

Wykonywanie intensywnych pod względem obliczeniowym operacji z pamięci RAM

Dalsze komunikaty ULP Advisor odnoszą się do naszych obliczeń stopni Celsjusza, Fahrenheita lub wartości średnich. Odpowiednia zasada brzmi: *„(ULP 5.1) Avoid processing-intensive operations: modulo, divide”*. Zalecamy skopiowanie kodu procesów obliczeniowych do pamięci RAM i wykonywanie ich stamtąd. Wykonywanie programu z pamięci RAM zużywa mniej energii niż z pamięci nieulotnej takiej jak Flash czy FRAM. Zwykle mikrokontroler dysponuje znacznie większą ilością pamięci Flash/FRAM w porównaniu do RAM. W związku z tym należy dokładnie przemyśleć jaką część RAM można zarezerwować do tego typu przeznaczenia. Dlatego w naszej aplikacji, w której pamięć RAM ma tylko 512 bajtów, rezygnujemy z wykonywania kodu z RAM. Zwłaszcza, że nasze przykładowe obliczenia dzielenia przez potęgi liczby 2 nie są wymagające pod względem obliczeniowym i mogą być zrealizowane z poziomu języka C przy użyciu instrukcji przesunięcia w prawo.

Jeśli dzielenie w tekście źródłowym zostaje zastąpione operacją przesunięcia, znika komunikat ULP Advisor. W naszym przykładowym kodzie ULP Advisor zauważa również mnożenie i zwraca uwagę na to, że wybrany przez nas mikrokontroler nie oferuje żadnego wspomagania sprzętowego (moduł MPY). Jeśli aplikacja wymaga jednak częstego wykonywania operacji mnożenia, należy pomyśleć o zastosowaniu układu wyposażonego w odpowiedni moduł sprzętowy.

Podczas przesyłania większych ilości danych lub często powtarzanych procesów kopiowania zaleca się stosowanie kontrolera DMA (brak w naszym przykładzie). Kontroler DMA jest dedykowany do tego rodzaju operacji i zużywa w porównaniu do CPU mniej cykli taktowania i mniej bramek do przesłania takiej samej ilości danych z jednego miejsca w pamięci do drugiego. Generalnie obowiązuje zasada: każdy cykl taktowania odpowiada określonej ilości energii, która jest zużywana i nigdy nie powraca. Mniej cykli taktowania oznacza mniej zużytej energii.

Roger Neumair, TI
Wolfgang Lutsch, TI