AUTOMATYKA • 2011 • Tom 15 • Zeszyt 2

Piotr Szymczyk*, Magdalena Szymczyk*

Energooszczędne algorytmy w systemach wbudowanych**

1. Wprowadzenie

Współczesne systemy wbudowane są bardzo często także systemami mobilnymi zasilanymi z ograniczonych źródeł energii (baterie jednorazowe, akumulatory, baterie słoneczne, energia pozyskiwana z mikrodrgań, energia pozyskiwane z fal radiowych itp.) [2, 8, 9]. W ostatnim czasie ilość takich systemów zwiększa się lawinowo ze względu na ich niską cenę i zastosowanie ich niemal w każdej dziedzinie życia. Wymagania stawiane takim systemom to bardzo długi czas pracy bez wymiany baterii lub bez doładowania akumulatora. W celu sprostania takim wymaganiom należy odpowiednio zaprojektować urządzenie pod względem elektrycznym przez dobór odpowiedniego (energooszczędnego) sprzętu, na przykład odpowiednich mikrokontrolerów [4, 7]. Drugim aspektem związanym z energooszczędnym projektowaniem jest dobór odpowiedniego źródła energii – efektywnego ze względu na specyfikę projektowanego systemu wbudowanego. Trzecim zagadnieniem mającym bezpośredni i bardzo często znaczący wpływ na oszczędzanie energii w systemach wbudowanych jest oprogramowanie tych systemów. Stosowanie odpowiednich algorytmów oraz struktur danych pozwala znacząco zmniejszyć zapotrzebowanie na energię zasilania [6].

2. Energooszczędne techniki programistyczne

W rozdziale tym zostaną opisane pewne techniki programistyczne pozwalające na optymalne wykorzystanie energii zasilania systemu wbudowanego. Można wyróżnić techniki stosowane na poziomie oprogramowania systemowego oraz na poziomie samej aplikacji.

2.1. Duża pętla a system operacyjny

Duża część aplikacji wbudowanych bazuje na wykonywaniu w tak zwanej dużej pętli, w której poszczególne zadania są uruchamiane sekwencyjnie jako kolejne procedury

* AGH Akademia Górniczo-Hutnicza, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Katedra Automatyki

** Projekt został sfinansowany ze środków Narodowego Centrum Nauki 6693/B/T02/2011/40

wewnątrz tej pętli [5, 22]. Takie rozwiązanie jest mało elastyczne i nie nadaje się do implementowania bardziej złożonych i wymagających systemów oraz nie daje możliwości obsługi asynchronicznych zdarzeń w czasie rzeczywistym. Programowanie z użyciem dużej pętli ma jednak niezaprzeczalną zaletę w postaci prostoty i przejrzystości, nadaje się do stosunkowo prostych aplikacji realizujących pojedyncze funkcje. Programowanie z użyciem dużej pętli jest niemożliwe lub bardzo uciążliwe dla skomplikowanych i wymagających implementacji złożonych systemów wbudowanych. Jedynym uniwersalnym rozwiązaniem jest zastosowanie wielozadaniowego systemu operacyjnego, zawierającego mechanizm wywłaszczania, który umożliwia stworzenie aplikacji składającej się z wielu zadań pracujących niezależnie, posiadających swoją ważność z punktu widzenia czasu rzeczywistego oraz komunikujących się ze sobą oraz z otoczeniem [10, 11, 19]. Należy jednak zawsze mieć na uwadze, że sprzęt tutaj stosowany, na przykład mikrokontrolery, mają i będą miały mimo dużego ich rozwoju szereg ograniczeń i w związku z tym nie można stosować tu systemu operacyjnego czasu rzeczywistego o zbyt wysokich narzutach i wymaganiach na zasoby (system operacyjny dla własnych potrzeb nie może zabierać zbyt dużo pamięci i obciążać procesor) ponieważ uniemożliwi to realizację wielu aplikacji.

2.2. Uaktywnianie zadań zdarzeniami

Uaktywnianie zadań zdarzeniami jest optymalną techniką w przypadku systemu, którego zadaniem jest reagowanie na więcej niż jedno zdarzenie wewnętrzne lub zewnętrzne. Przez większość czasu system znajduje się w stanie uśpienia z obniżonym poborem energii zasilania. W chwili nadejścia zdarzenia system wychodzi z tego stanu, obsługuje to zdarzenie i po obsłużeniu go oczekuje na następne zdarzenie, oszczędzając energię.

2.3. Rezygnacja z pełnego systemu operacyjnego na rzecz prostego planisty

Każdy system operacyjny powoduje pewien narzut na właściwe obliczenia. Zwiększa potrzebną moc obliczeniową oraz wykorzystuje część zasobów na własne potrzeby, co z kolei bezpośrednio zwiększa zapotrzebowanie na energię zasilania. Jeśli w projektowanym systemie nie jest koniecznie potrzebne wykorzystanie pełnego systemu operacyjnego, to można zrezygnować z jego części lub wręcz ograniczyć się do prostego planisty, który zapewni jedynie wielozadaniowość.

2.4. Kompiltor wysikiego poziomu i opcje kompilacji

Pisząc program w języku wysokiego poziomu, na przykład w języku C, używamy kompilatora w celu uzyskania kodu maszynowego. Jakość kompilatora ma duże znaczenie i zasadniczy wpływ na postać kodu wynikowego. Bardzo często w takich przypadkach wykorzystywany jest darmowy kompilator GCC, nawet jeśli sam interfejs jest stworzony przez

jakąś komercyjną firmę. Opcje kompilacji kompilatora wysokiego poziomu określają wiele parametrów procesu kompilacji. Poprzez użycie odpowiednich ustawień kompilatora należy optymalizować kod wynikowy, ponieważ im mniejszy i bardziej efektywny kod, tym bardziej sprawny i energooszczędny system.

2.5. Rezygnacja z funkcji

W trakcie pisania kodu zdarza się dość często, że pewne fragmenty kodu powtarzają się i zwykle w takich przypadkach definiuje się funkcję, która jest wywoływana z odpowiednimi parametrami w odpowiednim miejscu w kodzie programu. Wywołanie funkcji powoduje szereg koniecznych czynności i pracochłonnych operacji. Jeśli możemy zrezygnować ze stosowania funkcji lub ograniczyć ich użycie, to możemy w ten sposób zmniejszyć czas wykonania poszczególnych etapów obliczeń.

2.6. Używanie funcji online i naked

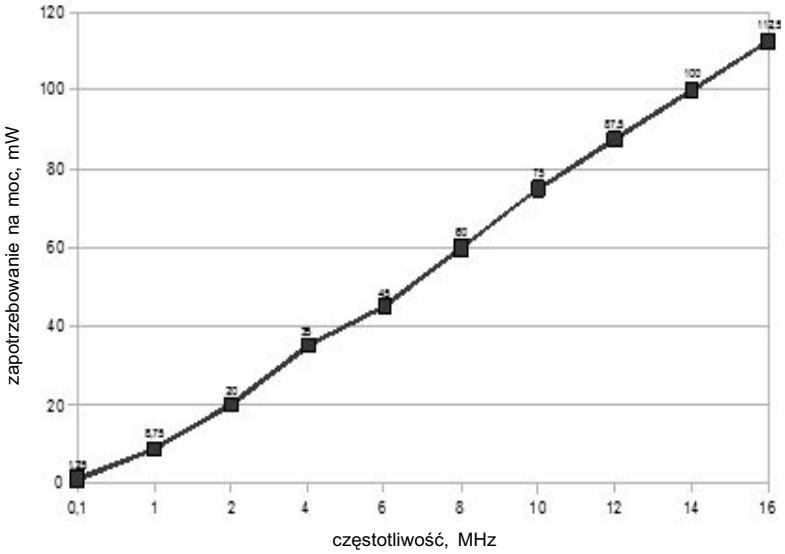
Czasami nie można zrezygnować z użycia wszystkich funkcji. W takim przypadku można użyć funkcje online lub naked. Funkcje tego typu dają minimalny narzut na wykonanie kodu programu, jednocześnie pozwalając programiście na komfort pisania złożonych programów z użyciem funkcji. Jeśli zaznaczymy w kodzie źródłowym, że dana funkcja ma być typu naked, wtedy kompilator nie stworzy prologu i epilogu tej funkcji [1] i wywołanie jej będzie bardzo szybkie. Dla funkcji online jej kod wynikowy jest wprost umieszczany w kodzie wynikowym w miejscu odpowiadającym kodowi źródłowemu wywołującego ją, powoduje to, że nie ma skoku do kodu funkcji oraz szeregu operacji na stosie, które dla zwykłych funkcji są niezbędne i bardzo czasochłonne.

2.7. Zmienne globalne

Użycie zmiennych globalnych zmniejsza kod wynikowy programu, ponieważ nie trzeba przekazywać danych poprzez parametry wywołania funkcji. Niemniej jednak taka technika niesie z sobą pewne niedogodności i zagrożenia polegające na tym, że trzeba uważać na nazwy, aby nie nastąpiło na przykład przesłanianie zmiennych wewnątrz funkcji. Trzeba również bardzo uważnie kontrolować użycie zmiennych, ponieważ błędne obliczenia mogą mieć znacznie większe skutki niż w przypadku zmiennych lokalnych.

2.8. Wstrzymywanie pracy całego mikrokontrolera, tryby pracy mikrokontrolera, częstotliwość rdzenia na przykładzie mikrokontrolera ATmega32

Na rysunku 1 pokazano zależność zapotrzebowania na energię zasilania mikrokontrolera ATmega32 w zależności od jego częstotliwości pracy. Im wyższa częstotliwość, tym potrzebna jest większa moc zasilania.



Rys. 1. Zapotrzebowania na moc w zależności od częstotliwości zegara dla ATmega 32

Mikrokontroler ATmega 32 posiada sześć trybów pracy:

- Tryb Idle – CPU i wykonanie programu zostają wstrzymane, zegar taktujący oraz wszystkie układy peryferyjne działają normalnie.
- Tryb ACD Nosie Reduction – jest on wykorzystywany w trakcie pracy przetwornika A/C aby poprawić dokładność wyniku dzięki wyeliminowaniu wewnętrznych źródeł zakłóceń (praca CPU oraz niektóre moduły wejścia wyjścia powodują takie zakłócenia).
- Tryb Power-down – jest najgłębszym trybem uśpienia, wyłączany jest zegar systemowy oraz prawie wszystkie układy peryferyjne, pozostają aktywne jedynie moduły, które pracują asynchronicznie względem głównego układu taktującego.
- Tryb Power-save – podobny tryb jak Power-down za wyjątkiem układu licznika 0, który może w tym trybie dalej działać.
- Tryb Standby – podobny tryb do Power-down z tą różnicą, że jest możliwy tylko z rezonatorem zewnętrznym i nie wyłącza go.
- Tryb Extended Standby – podobny tryb do Power-save z tą różnicą, że jest możliwy tylko z rezonatorem zewnętrznym i nie wyłącza go.

Jak wynika z przeprowadzonych badań [20] (tab. 1), najlepszym rozwiązaniem pod względem energetycznym jest prowadzenie obliczeń z maksymalną dostępną prędkością (maksymalną częstotliwością mikrokontrolera), a następnie przeprowadzenie mikrokontrolera w stan uśpienia. Dzięki temu można dodatkowo zaoszczędzić do około 19% energii zasilania w stosunku do prowadzenia obliczeń z obniżoną częstotliwością.

Tabela 1
Porównanie zapotrzebowania na moc dla różnych rozwiązań

Rodzaj pracy	Czas wykonania	Średnia moc [mW]
Praca ciągła tryb aktywny 16 MHz	T/16	112,50
Praca ciągła tryb aktywny 1 MHz	T	8,75
Praca ze zmianą trybów aktywny 16MHz/Power-down	T/16	7,11

2.9. Wstrzymywanie pracy poszczególnych modułów mikrokontrolera ARM Cortex M3 (STM32L)

W mikrokontrolerach STM32L które są przedstawicielami mikrokontrolertów ARM Cortex M3 można włączać i wyłączać poszczególne magistrale (APB1, APB2 i AHB) oraz poszczególne moduły (TIM2, TIM3, …). W tabeli 2 pokazano pobór prądu dla poszczególnych modułów mikrokontrolera STM32L [12, 18] dla różnych napięć zasilania rdzenia mikrokontrolera.

Tabela 2
Pobór prądu dla poszczególnych peryferii mikrokontrolera STM32L

Peryferia		Typowy pobór prądu [µA/MHz] dla V_{DD}=3,0V i T_A=25ºC		
		V_{CORE}=1,8V	V_{CORE}=1,5V	V_{CORE}=1,2V
APB1	TIM2	13	10,5	8
	TIM3	14	12	9
	TIM4	12,5	10,5	8
	TIM6	5,5	4,5	3,5
	TIM7	5,5	5	3,5
	LCD	5,5	5	3,5
	WWDG	4	3,5	2,5
	SPI2	5,5	5	4
	USART2	9	8	5,5
	USART3	10,5	9	6
	I2C1	8,5	7	5,5
	I2C2	8,5	7	5,5
	USB	12,5	104	6,5
	PWR	4,5	4	3
	DAC	9	7,5	6
	COMP	4,5	4	3,5

Tabela 2 cd.

Peryferia		Typowy pobór prądu [µA/MHz] dla V_{DD}=3,0V i T_A=25ºC		
		V_{CORE}=1,8V	V_{CORE}=1,5V	V_{CORE}=1,2V
APB2	SYSCFG & RI	3	2,5	2
	TIM9	9	7,5	6
	TIM10	6,5	5,5	4,5
	TIM11	7	6	4,5
	ADC1	11,5	9,5	8
	SPI1	5	4,5	3
	USART1	9	7,5	6
AHB	GPIOA	5	4,5	3,5
	GPIOB	5	4,5	3,5
	GPIOC	5	4,5	3,5
	GPIOD	5	4,5	3,5
	GPIOE	5	4,5	3,5
	GPIOF	5	4,5	3,5
	CRC	1	0,5	0,5
	DMA1	12	10	8
Wszystko włączone		**166**	**138**	**106**

Gdzie V_{DD} oznacza napięcie zasilania mikrokontrolera, T_A temperaturę mikrokontrolera, V_{CORE} napięcie zasilania rdzenia.

Często zdarza się, że w trakcie pracy systemu potrzebne są w danej chwili jedynie niektóre moduły mikrokontrolera. Wstrzymywanie niepotrzebnych modułów mikrokontrolera umożliwia znaczące zmniejszenie zapotrzebowania na energię [3]. Optymalne włączanie i wyłączanie modułów i magistral nie jest zagadnieniem prostym, ponieważ trzeba rozważyć, czy takie działanie ma sens ze względu na to, że operacje te wymagają czasu i energii. Jeśli dany moduł jest bardzo często używany, to może się okazać, że nie można go w ogóle wyłączać, ponieważ wymagania czasowe systemu byłyby nie spełnione.

2.10. Programistyczne dynamiczne zmiany częstotliwości pracy modułów mikrokontrolera

Zmniejszenie częstotliwości zegara poszczególnych modułów mikrokontrolera podobnie jak w przypadku częstotliwości taktowania rdzenia mikrokontrolera, prawie liniowo obniża zapotrzebowanie na moc zasilania poszczególnych modułów.

2.11. Programowanie w asemblerze

W wielu przypadkach można dodatkowo zoptymalizować kod programu, pisząc go w asemblerze lub stosując wstawki asemblerowe. Rozwiązanie takie jest dość uciążliwe ze względu na niski komfort pisania złożonego kodu w języku niskiego poziomu, ale z drugiej strony można stworzyć najbardziej optymalny kod wynikowy. Każdy kompilator języka wysokiego poziomu wnosi pewien narzut na kod i jeśli można się tego narzutu pozbyć, pisząc wprost w asemblerze, to należy to zrobić.

2.12. Kod programu w pamięci RAM

Kod programu w systemie wbudowanym może znajdować się w różnych rodzajach pamięci, może to być zarówno pamięć RAM, jak i EPROM oraz FLASH. Ponieważ pamięć RAM jest najszybsza, w związku z tym kod programu powinien znajdować się w pamięci RAM. Dlatego po uruchomieniu systemu bardzo często kopiuje się kod z wolniejszej pamięci do szybkiego RAM.

2.13. Tablice funkcji

Można również, gdzie jest to możliwe, stosować tablice funkcji. Pisząc kod źródłowy, możemy wypełnić tablicę funkcji wartościami, oczywiście o ile jest to wykonalne, ze względu na dostępne miejsce w pamięci i możliwość ich wyznaczenia na tym etapie. Tak stworzona tablica symuluje obliczenia i natychmiastowo podaje wynik. Jest to dobre rozwiązanie dla funkcji z parametrami, które mają w miarę ograniczoną liczbę wartości oraz gdy funkcje te są bardzo złożone obliczeniowo, a przez to czasochłonne.

3. Podsumowanie

W artykule przedstawiono programistyczne metody umożliwiające oszczędzanie energii zasilania dla systemów wbudowanych. W celu maksymalnego oszczędzania energii zasilania zaleca się więc: stosowanie uaktywniania zadań zdarzeniami, rezygnację z tak zwanej „dużej pętli" na rzecz systemu operacyjnego lub samego planisty, dobór optymalnego kompilatora wysokiego poziomu i właściwe ustawienie opcji kompilacji. Ponadto sugeruje się rezygnacje z używania funkcji wszędzie tam, gdzie jest to możliwe, a tam gdzie funkcje muszą pozostać, rozważenie zastosowania funkcji typu online lub nacked. Kolejne możliwości ograniczania mocy zasilania to stosowanie zmiennych globalnych, wstrzymywanie i dynamiczne zmiany częstotliwości pracy mikrokontrolera i jego poszczególnych modułów, kodowanie bezpośrednio w asemblerze, umieszczanie kodu aplikacji w pamięci RAM oraz stosowanie tablic funkcji. Stosowanie takich technik umożliwia stworzenie optymalnego pod względem energetycznym systemu wbudowanego, ale równocześnie może powodować dodatkowe komplikacje w trakcie tworzenia oprogramowania dla systemów

wbudowanych. Należy również nie zapominać o wymaganiach czasowych określonych dla takich systemów, aby nie doszło do sytuacji, w której system będzie dążył do maksymalnej oszczędności energii potrzebnej mu do pracy i nie spełni ograniczeń czasowych.

Literatura

[1] Baranowski R., *Mikrokontrolery AVR ATmega w praktyce.* BTC, Warszawa 2005.
[2] Berger A.S., *Embedded Systems Design: An Introduction to Processes, Tools and Techniques*. CMP Books, 2002.
[3] Bryndza L., *LPC2000 Mikrokontrolery z rdzeniem ARM7*. BTC, Warszawa 2007.
[4] Curtis K.E., *Embedded Multitasking With Small Microcontrollers*. Newnes Elsevier, Oxford 2006.
[5] Jasio L.D., *Programming 32-bit Microcontrollers in C: Exploring the PIC32*. Newnes, Oxford 2008.
[6] Jerraya A.A., Yoo S., Wehn N., Verkest D., *Embedded Software for SoC*. Kluwer Academic Publishers, Boston 2003.
[7] Kościelnik D., *Mikrokontrolery Nitron Motorola M68HC08*. WKŁ, Warszawa 2005.
[8] Lamie E.L., *Real-Time Embedded Multithreading: Using ThreadX and ARM*. CMP Books, 2005.
[9] Labrosse J. (ed), *Embedded Software.* Newnes Elsevier, Oxford 2008.
[10] Qing Li, *Real-Time Concepts for Embedded Systems*. CMP Books, San Francisco 2003.
[11] Sinha A., *Energy Efficient Operating Systems and Software*. Massachusetts Institute of Technology, Massachusett 2001.
[12] Starak T., *Mikrokontrolery ST w wersji L. Energooszczędne rozwiązania dla każdej aplikacji.* Elektronika Praktyczna, 1/2011.
[13] STM Datasheet – *AN3193 Application note STM32L15x ultralow power features overview.*
[14] STM Datasheet – *AN3216 Application note STM32L1xxx hardware development: getting started.*
[15] STM Datasheet – *PM0062 Programming manual STM32L151xx and STM32L152xx Flash and EEPROM programming.*
[16] STM Datasheet – *RM0038 Reference manual STM32L151xx and STM32L152xx advanced ARM-based 32-bit MCUs.*
[17] STM Datasheet – *STM32L151xx and STM32L152xx Errata sheet STM32L151xx and STM32L152xx ultralow power limitations.*
[18] STM Datasheet – *STM32L151xx STM32L152xx Ultralow power ARM-based 32-bit MCU with up to 128 KB Flash, RTC, LCD, USB, USART, I2C, SPI, timers, ADC, DAC, comparators.*
[19] Szymczyk P., *Systemy operacyjne systemów wbudowanych.* Pomiary Automatyka Robotyka, kwiecień 2009.
[20] Szymczyk P., *Energooszczędne oprogramowanie systeów czasu rzeczywistego mikrokontrolerów*. Materiasły konferencyjne SOC'2009.
[21] Szymczyk P., *Systemy operacyjne czasu rzeczywistego*. UWND AGH, Kraków 2003.
[22] Wilmshurster T., *Designing Embedded Systems with PIC Microcontrollers*. Principles and Applications, Newnes, Oxford 2007.
[23] Zbysiński P., *Energooszczędnie z Cortex M3*. Elektronika Praktyczna, 9/2010.