Jaszczurki zaciskają pasa

Możliwości energooszczędnych mikrokontrolerów EFM32 w teorii i praktyce (2)

Producenci układów scalonych prześcigają się w coraz śmielszych i bardziej zaawansowanych technologicznie rozwiązaniach dotyczących oszczędzania energii w aplikacjach. Artykuł ten przedstawia, w jaki sposób możemy wydłużyć autonomiczny czas działania konstruowanych przez nas urządzeń zasilanych bateryjnie, opartych na mikrokontrolerach rodziny EFM32 firmy Energy Micro.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12147, pass: 2e7u6a2a
- pierwsza część artykułu

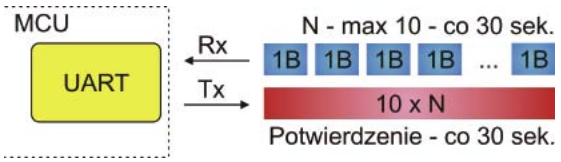
Chińska filozofia

„Usłyszę – zapomnę, zobaczę – zapamiętam, ale kiedy zrobię sam, to zrozumiem."

Powiedzenie to, przypisywane chińskiemu filozofowi Konfucjuszowi, towarzyszy ludzkości od ponad dwóch tysięcy lat. Można je odnieść do różnych dziedzin życia i nauki, w tym również do elektroniki. Nic tak nie uczy elektroniki, zarówno analogowej, jak i cyfrowej, jak samodzielne zaprojektowanie, złożenie i przetestowanie pewnego systemu. Wiadomo, podstawy teoretyczne są potrzebne do wytłumaczenia zjawisk i zasady działania układów, jednak nic tak nie cieszy elektronika praktyka jak „prawie gotowy" prototyp własnoręcznie wykonanego urządzenia. Dlatego w tej części artykułu zostaną poddane próbie i przeanalizowane niektóre dostępne metody oszczędzania energii w mikrokontrolerach *EFM32* firmy *Energy Micro*, przedstawione w poprzedniej części artykułu.

Problem do rozwiązania

Dla wyjaśnienia stopnia oszczędności energii przy odpowiednim zastosowaniu naszego gekona wymyślony został prosty przykład z transmisją szeregową przez *UART* (9600 bodów). Zastosowano ten interfejs, ponieważ jest on bardzo dobrze znany wszystkim elektronikom, od amatorów po profesjonalistów. Załóżmy, że pewna część całości systemu ma realizować transmisję szeregową. W czasie półminutowych interwałów może trafić do mikrokontrolera maksymalnie dziesięć bajtów danych. Po tym czasie wysyłana jest kontrolna informacja zwrotna w postaci dziesięciu takich samych bajtów zawierająca liczbę bajtów, które dotarły od wysłania ostatniego potwierdzenia (**rysunek 17**). Przykład ten nie powinien

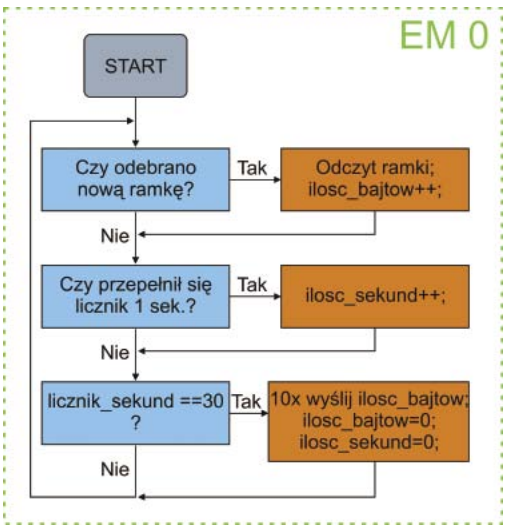


Rysunek 17. Przykładowe zadanie do rozwiązania

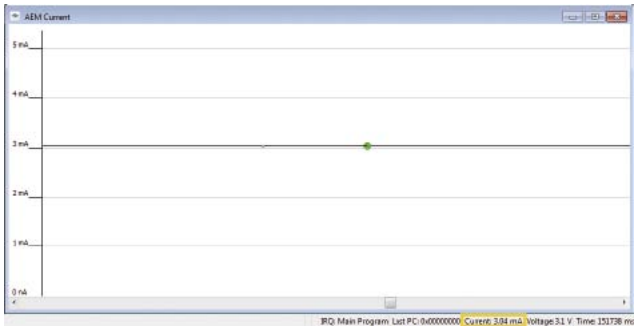
być postrzegany jako algorytm potwierdzeń podczas przesyłu danych, lecz jedynie jako sposób prezentacji możliwości energetycznych mikrokontrolerów *EFM32*. Kod programu został napisany na podstawie przykładów dostępnych na stronie *www.energymicro.com*, w większości z użyciem biblioteki *CMSIS*. Problem może być rozwiązany na trzy różne, niżej opisane sposoby, zaczynając od najprostszego zużywającego najwięcej energii, stopniowo przechodząc do bardziej wyrafinowanych funkcji, korzystających z zalet naszych gekonów, dzięki którym średnie zużycie energii znacząco maleje. Wybór poszczególnych rozwiązań odbywa się poprzez zmianę w pliku *energysaving.h* wartości dyrektywy *#define PRZYKLAD (X)*, gdzie *X* może przyjmować wartości 1, 2 lub 3. Należy pamiętać, że w przypadku, gdy mamy wyłączoną opcję automatycznego czyszczenia projektu, podczas kompilacji należy przy zmianie tylko naszej dyrektywy samodzielnie wyczyścić projekt, ponieważ kompilator może nie doszukać się zmiany w kodzie.

Rozwiązanie nr 1

Najprostsze, a zarazem najmniej efektywne energetycznie. Polega na zastosowaniu zwykłego *UART-a* i licznika zliczającego sekundowe czasy, które taktowane są przez wysokoczęstotliwościowe zegary. Przez cały czas procesor sprawdza, czy w buforze odbiorczym *UART-a* nie ma dostępnej nowej ramki oraz czy licznik nie jest przepełniony. Jeżeli jest dostępna nowa ramka, to od-



Rysunek 18. Algorytm rozwiązania nr 1



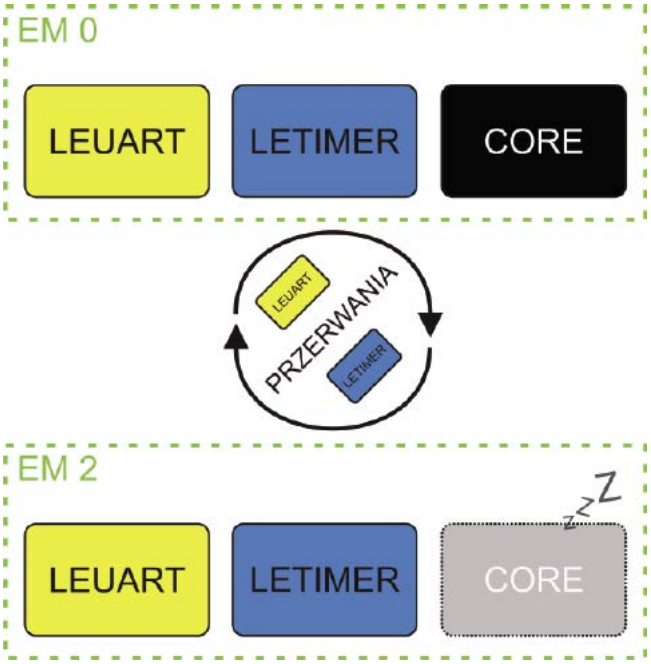
Rysunek 19. Pobór prądu w rozwiązaniu nr 1

czytujemy ją oraz zwiększamy zmienną określającą liczbę odebranych ramek. Jeżeli nastąpi przepełnienie sekundowego licznika, to zwiększamy zmienną określającą, ile sekund upłynęło. W przypadku gdy minęło już pół minuty, wysyłana jest przez *UART* liczba ramek, które zostały odebrane w ostatnim czasie, procesor czeka, aż przesyłanie się powiedzie, a następnie zerowany jest licznik sekundowy oraz licznik liczby ramek. Algorytm ten wykonywany jest cyklicznie (**rysunek 18**). Rozwiązanie to, choć spełnia zadaną funkcję, jest niepoprawne pod względem energetycznym w aplikacjach o niskim poborze mocy. Procesor przez większość czasu nie jest używany, a jednak jest nadal taktowany wysokoczęstotliwościowym zegarem i pobiera prąd, którego średnie zużycie zostało sprawdzone w programie *energyProfiler* i przez cały czas wynosiło 3,04 mA (**rysunek 19**). Na szczęście każdy mikroprocesor można zaprogramować tak, aby w bardziej elegancki sposób zrealizować funkcjonowanie naszej części urządzenia poprzez obsługę przerwań. Dzięki temu rdzeń będzie mógł sobie przejść w tryb uśpienia *EM1* i budzić się jedynie, gdy dostępna będzie nowa ramka w buforze odbiorczym *UART-a* lub gdy przepełni się licznik. Taką modyfikację kodu i sprawdzenie rezultatu w programie *energyProfiler* zalecamy jako „zadanie domowe", gdyż jest to rozwiązanie powszechnie stosowane dla każdej rodziny mikrokontrolerów. My z kolei pójdziemy o krok dalej.

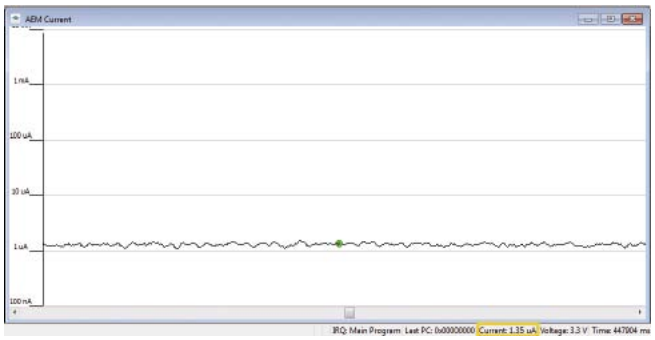
Rozwiązanie nr 2

Ponieważ nasz *UART* ma pracować z prędkością 9600 bodów, możemy zamiast zwykłego *UART'a* użyć zaprojektowanego specjalnie w celu oszczędzania energii peryferium *LEUART-a* (*Low Energy UART*). Ma on tę zaletę, że może być taktowany z osobnego zegara niskoczęstotliwościowego 32,768 kHz i osiągać prędkość do 9600 bodów, czyli tyle, ile potrzebujemy. Również zamiast stosowania zwykłego licznika możemy zastosować jego niskoenergetycz-

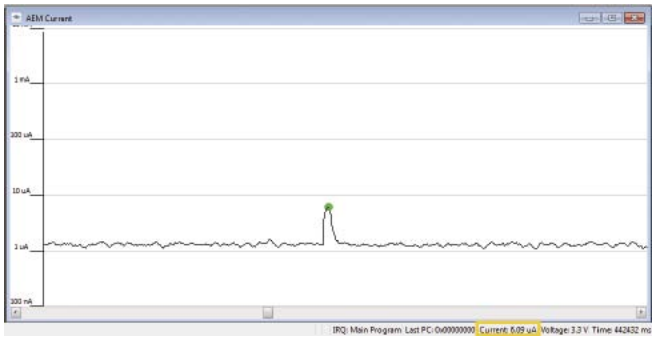
Rysunek 20. Usypianie zegara wysokoczęstotliwościowego w trybie EM2



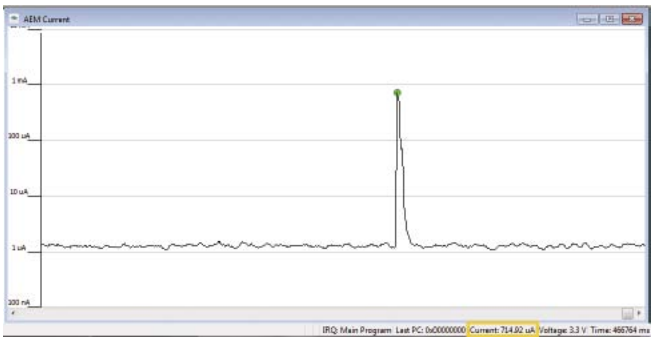
Rysunek 21. Rozwiązanie nr 2



Rysunek 22. Pobór prądu w rozwiązaniu nr 2 podczas bezczynności

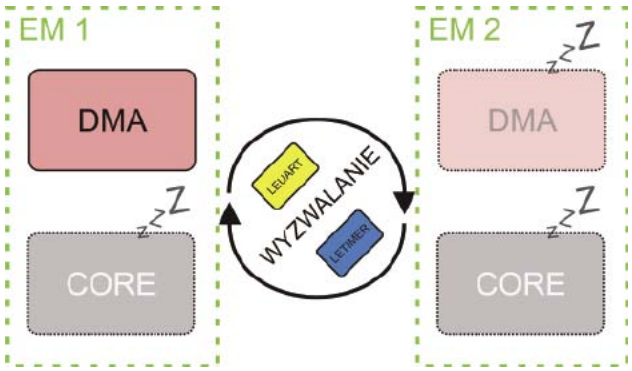


Rysunek 23. Pobór prądu w rozwiązaniu nr 2 podczas odbioru danych

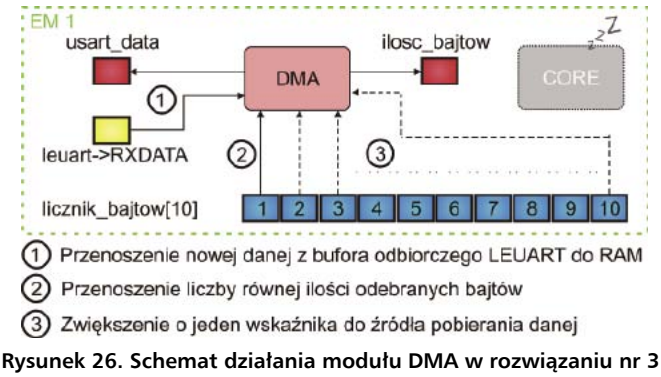


Rysunek 24. Pobór prądu w rozwiązaniu nr 2 podczas wysyłania danych

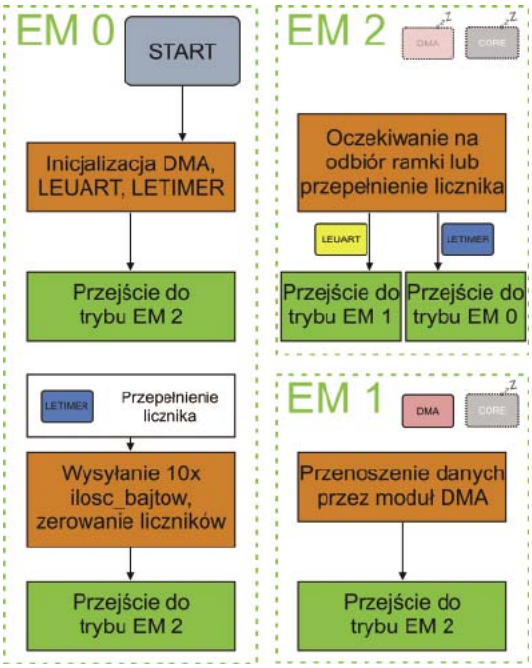
ny odpowiednik *LETIMER* (*Low Energy TIMER*), który także może działać z niskoczęstotliwościowym taktowaniem. Dzięki takiemu doborowi specjalistycznych peryferii możemy wprowadzić procesor w tryb uśpienia *EM2*, w którym zużycie energii jest rzędu mikroamperów dzięki wyłączaniu zegarów wysokoczęstotliwościowych (**rysunek 20**). W tym samym czasie na niskoczęstotliwościowym zegarze będą pracować peryferia *LE*, zachowując w pełni funkcjonalność naszej części systemu, uruchamiając wysokoczęstotliwościowy zegar i wybudzając rdzeń (przejście do trybu *EM0*) tylko gdy będzie to konieczne (**rysunek 21**). Podczas testowania zużycia energii w programie *energyProfiler* przesłanych zostało ciągiem 8 bajtów. Zastosowana została skala logarytmiczna, aby lepiej zobrazować wyniki. Średnie natężenie prądu podczas oczekiwania (*EM2*) wynosiło 1,35 µA (**rysunek 22**), a zatem ponad 2000 razy mniej niż w poprzednim rozwiązaniu. W chwili przyjmowania bajtów na wykresie można zauważyć charakterystyczny skok zużycia prądu (**rysunek 23**) spowodowany potrzebą przejścia procesora w tryb *EM0*, aby obsłużyć przerwanie – odebrać daną i zwiększyć licznik przychodzących bajtów. Pod koniec testowania można zauważyć kolejny, tym razem większy, skok poboru prądu (**rysunek 24**). Również i tym razem procesor przechodzi z trybu *EM2* w tryb *EM0*, w celu obsługi przerwania 30-sekundowego licznika – wysłania dziesięciu bajtów, z których każdy zawiera liczbę odebranych w ostatnim czasie bajtów, a następnie oczekiwania na zakończenie wysyłania danych. Na pierwszy rzut oka widać, że w odniesieniu do poprzedniego rozwiązania, dzięki zastosowaniu przerwań oraz specjalnych energooszczędnych peryferii, redukcja średniego natężenia prądu zasilania jest znaczna. Jest to spowodowane tym, że przez większość czasu procesor jest w trybie uśpienia, a peryferia działają na niskich częstotliwościach. Nieznacznym usprawnieniem może być dodanie kolejnego przerwania, tym razem od nadajnika *LEUART-a*. Dzięki temu procesor będzie mógł przechodzić w tryb uśpienia pomiędzy umieszczeniem danej w buforze nadawczym a potwierdzeniem zakończenia transmisji i rozpoczęciem przesyłania kolejnego bajtu. Niech to będzie zadaniem domowym numer 2. Oszczędność energetyczna wydaje się już dość imponująca. Czy da się jeszcze bardziej zacisnąć pasa naszemu gekonowi? Oczywiście tak!



Rysunek 25. Autonomiczne przejście pomiędzy trybami EM1 i EM2 przy wyzwalaniu transmisji DMA z LEUART i LETIMER



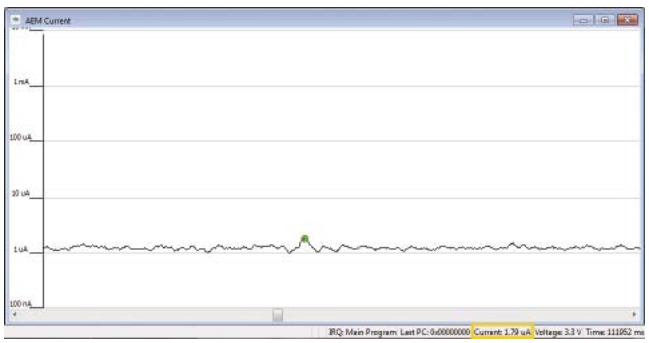
Rysunek 26. Schemat działania modułu DMA w rozwiązaniu nr 3



Rysunek 27. Algorytm rozwiązania nr 3

Rozwiązanie nr 3

Podstawą dużej oszczędności energii jest napisanie takiego kodu, aby funkcjonowanie mikrokontrolera było oparte na autonomicznej pracy peryferii (szczególnie tych niskoenergetycznych) i maksymalizowaniu czasu, w jakim procesor pozostaje w uśpieniu. Zatem, czy jesteśmy w stanie wyeliminować potrzebę udziału rdzenia podczas odbioru danych z *LEUART-a*? Tak – w bardzo prosty sposób – dzięki użyciu modułu *DMA*, w który wyposażony został nasz gekon (*DMA* wbudowany w rodzinę *EFM32* jest licencjonowanym produktem firmy *ARM*). Odpowiednio skonfigurowany kontroler *DMA* będzie przenosił daną z bufora odbiorczego *LEUART-a* do pamięci *RAM*. Choć *DMA* pracuje w trybie *EM1*, możliwe jest wybudzanie go za pomocą niskoenergetycznych peryferii pracujących w trybie *EM2*. Przejście pomiędzy tymi trybami odbywa się autonomicznie bez potrzeby ingerencji użytkownika, zużywając bardzo mało prądu (**rysunek 25**). A co ze zwiększaniem licznika przychodzących bajtów? Prostym sposobem można tego dokonać bez potrzeby wybudzania procesora. Podczas inicjalizacji zmiennych tworzymy wektor o liczbie elementów odpowiadającej maksymalnej liczbie bajtów, które mogą zostać odebrane. Przy z naszeym założeniu jest to 10 bajtów. Do tego wektora wstawiamy kolejno liczby: 1, 2, …, 10. Teraz tak konfigurujemy kanał kontrolera DMA, żeby przy każdej nowo-przyjętej danej przenosił wartość z naszego wektora do adresu, pod którym znajduje się licznik bajtów, za każdym razem zwiększając adres źródłowy przenoszenia o 1 pozycję (**rysunek 26**). Podsumowując, rozwiązanie to jest lepsze od poprzedniego, ponieważ nie ma potrzeby wybudzania procesora podczas obsługi przychodzących bajtów – tym zajmuje się kontroler *DMA*. Rdzeń jest aktywowany tylko przy zakończeniu 30-sekundowych okresów podczas wysyłania danych (**rysunek 27**). Dzięki temu następuje przechodzenie jedynie pomiędzy trybami *EM1* i *EM2* podczas odbioru, a nie jak w poprzednim przypadku *EM0*, co w tym przypadku powoduje zmniejszenie amplitudy skoku poboru prądu (**rysunek 28**), która z użyciem *DMA* wyniosła 1,79 μA. Oczywiście kod można byłoby jeszcze optymalizować pod względem energetycznym, ale celem tego artykułu jest ogólne zaprezentowanie energooszczędnych rozwiązań, w które wyposażone zostały mikrokontrolery z rodziny *EFM32*. Na uwagę zasługuje również jeszcze jedno ciekawe narzędzie, które możemy często wykorzystywać.



Rysunek 28. Zmniejszony pobór prądu w rozwiązaniu nr 3 podczas odbioru danych

PRS (Pefipheral Reflex System)

Moduł *PRS* służy do autonomicznej komunikacji pomiędzy peryferiami mikrokontrolera *EMF32*. Dzięki niemu możliwe jest odciążenie procesora w pewnych sytuacjach, jak na przykład rozpoczęcie próbkowania sygnału zgodnie z pewną częstotliwością dyktowaną przerwaniami z licznika. Niestety w naszym przykładzie nie możemy użyć tego modułu, ponieważ nie obsługuje on peryferii niskoenergetycznych. Jednak jeżeli zachodziłaby potrzeba użycia zwykłego *UART-a* i *TIMER-a* (przykładowo w przypadku używania większej prędkości transmisji niż 9600 bodów), to moglibyśmy dzięki zaprzęgnięciu do pracy pary: *DMA* i *PRS*, całkowicie wyeliminować konieczność obsługi założonej funkcji przez procesor. Ze strony *www.energymicro.com* możemy pobrać i przetestować program (nota aplikacyjna *AN0025*) pokazujący działanie modułu *PRS* (konwersja *ADC* wyzwalana licznikiem, transmisja *UART* wyzwalana stanem pinu, konwersja *DAC* wyzwalana programowo, pomiar szerokości impulsu wyzwalana komparatorem i licznikiem).

Podsumowanie

Mamy nadzieję, że przedstawione praktyczne i teoretyczne przykłady pomogą Czytelnikom przy trudnym i czasochłonnym rozwiązywaniu problemów z optymalizacją zużycia energii w projektach. Warto samodzielnie, na spokojnie przyswoić sobie przedstawioną wiedzę i rozwiązania, przetestować i przeanalizować kod, dostrzec zmiany, jakie powoduje on na wykresie energetycznym w programie *energyProfiler*. Należy pamiętać, że jeżeli jesteśmy w trybie debugowania procesora, to nie może on przejść niżej niż tryb *EM1* (uniemożliwia to otwarta sesja debuggera). Zatem, jeżeli chcemy zaobserwować przejście w tryb *EM2* i niższe, należy wgrać kod do pamięci Flash, zakończyć sesje debugowania, zrestartować mikrokontroler i gotowe.

Każdy projekt krytyczny energetycznie potrzebuje indywidualnego podejścia zarówno od strony sprzętowej, jak i programowej. Kluczowymi są: dobór układów i samo oprogramowanie. Przy testowaniu można po pierwsze próbować uruchomić działający funkcjonalnie prototyp, a następnie optymalizować go od strony energetycznej, jednak często zdarza się, że potem musimy zamienić piny zwykłego *UART-a* z jego wersją niskoenergetyczną, co może być kłopotliwe. Dlatego zalecamy od samego początku zwracać uwagę, aby pisać kod w wersji *low-energy*. Z przedstawionym arsenałem specjalnych peryferii i rozwiązań, jakimi dysponują mikrokontrolery z rodziny *EFM32* oraz wsparciem programu *energyProfiler*, z pewnością będzie to łatwiejsze. Elektronikom amatorom sprawi to dużo satysfakcji i dostarczy nowej wiedzy, a profesjonalistom pozwoli na znaczne skrócenie krytycznego czasu *time-to-market*.

Wojciech Gelmuda
Piotr Bratek
Andrzej Kos
Katedra Elektroniki
Akademia Górniczo-Hutnicza