„Rewolucyjne" MSP430 z pamięcią FRAM

Przykładowy rejestrator zdarzeń alarmowych

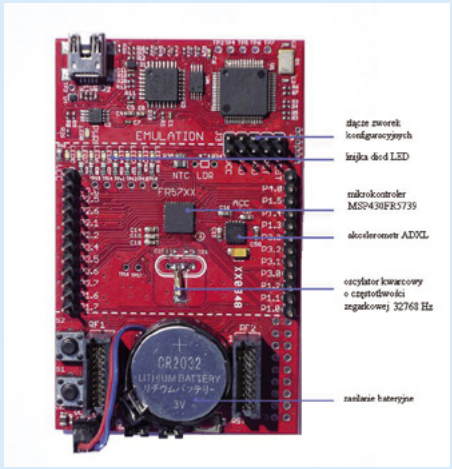
O MSP430 z pamięcią FRAM pisaliśmy w EP08/2011. Wówczas, prezentowane były układy serii FR57xx. Omówione zostały właściwości pamięci FRAM, oraz budowa płytki startowej z zestawu MSP-EXP430FR5739. W dzisiejszym artykule na przykładzie projektu rejestratora alarmów pokażemy w jaki sposób „FRAMowe" MSP430 zastosować w praktyce.

Oprogramowanie rejestratora zostało napisane przy użyciu środowiska *IAR Embedded Workbench Kickstart Edition* (darmowa wersja z ograniczeniem kodu wynikowego 8 kB, do pobrania ze strony www.iar.com). Za bazę sprzętową posłużyła płytka startowa z zestawu *MSP-EXP430FR5739* (opisana w EP 08/11). W module płytki startowej zainstalowano koszyczek na baterie CR2032, oraz dołączony do zestawu oscylator kwarcowy o częstotliwości zegarkowej 32768 Hz. Wygląd płytki startowej „po poprawkach" ilustruje **rysunek 1**.

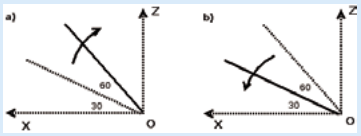
Kod źródłowy projektu został napisany w języku programowania C. Autor oprogramowania zaimplementował biblioteki obsługi modułów sprzętowych (RTC, WDT, UART, ADC, DMA, CRC, FRAM itd.), oraz programowych (bufory cykliczne, transmisja danych, konwersja liczb, wykrywanie alarmów, logowanie danych, diagnostyka błędów). Biblioteki zostały skonstruowane w sposób, który ułatwia ich użycie w innych projektach.

Sposób działania

Zaprojektowany rejestrator alarmów korzysta z zamontowanego na płytce startowej *MSP-EXP430FR5739* akcelerometru ADXL. Mikrokontroler cyklicznie monitoruje wskazania akcelerometru (poziom napięcie na linii kanału „Z", który odzwierciedla odchylenie akcelerometru w pozycji pionowej). W momencie wykrycia odchylenia o kąt powyżej 60% (próg alarmowy w górę), bądź o kąt poniżej 30% (próg alarmowy w dół) zgłaszane są *alarmy otwarcia*/zamknięcia. Informacja o wystąpieniu alarmu, wraz z czasem jego wystąpienia zapisywana jest w wewnętrznej pamięci FRAM mikrokontrolera. W sposób graficzny działanie rejestratora przedstawia **rysunek 2**.



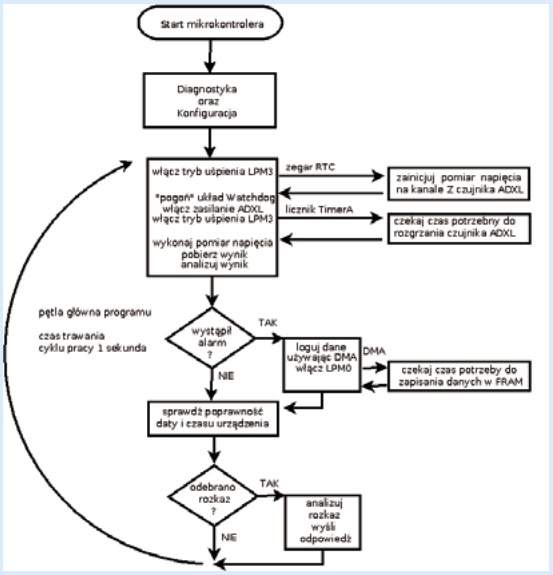
Rysunek 1. Płytka startowa MSP-EXP430FR5739



Rysunek 2. Praca rejestratora. a) alarm otwarcia b) alarm zamknięcia *zestaw startowy należy zamontować w pozycji poziomej, krótszą krawędzią (z złączem USB) stycznie do osi OY, dłuższą krawędzią stycznie do osi OX

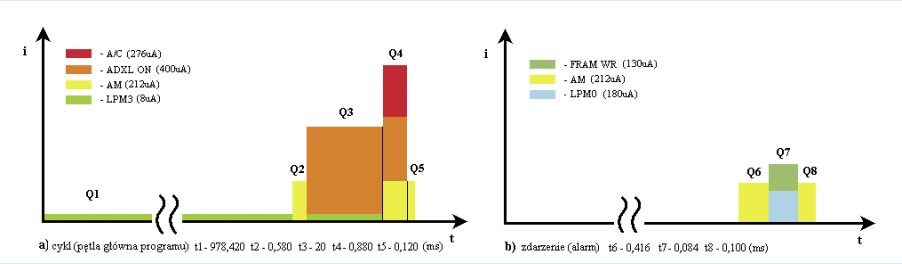
Wykonanie

Jednym z założeń podczas realizacji projektu było zapewnienie możliwości pracy urządzenia na zasilaniu bateryjnym. Żeby wydłużyć czas pracy rejestratora na jednym komplecie baterii, konieczne było zadbania o energooszczędność urządzenia. W tym celu, w projekcie autor skorzystał z trybów uśpienia MSP430, w tym z trybu LPM3 w którym pobór prądu mikrokontrolera wynosi 6,8 µA. Po starcie mikrokontrolera wykonywana jest diagnostyka i konfiguracja, a następnie rozpoczyna się wykonanie pętli głównej programu. Na początku pętli mikrokontroler



Rysunek 3. Schemat blokowy programu

wprowadzany jest w tryb uśpienia LPM3, w którym przebywa około 978 ms. Z uśpienia mikrokontroler budzi przerwanie od zegara RTC. Wówczas inicjowany jest pomiar napięcia na linii kanału „Z" akcelerometru ADXL. Włączane jest zasilanie akcelerometru, i inicjowane jest opóźnienie o czasie trwania 20 ms (czas potrzebny do „rozgrzanie" układu ADXL). Mikrokontroler wprowadzany jest w tryb uśpienia LPM3 z którego po upływie 20 ms budzony jest przez przerwanie od licznika TimerA. Wykonywany jest pomiar napięcia, oraz analizowany wynik. W przypadku wykrycia wystąpienia alarmu, informacja o alarmie zapisywana jest w pamięci mikrokontrolera (z wykorzystaniem modułu DMA i trybu uśpienia LPM0). Na zakończenie pętli głównej programu sprawdzana jest poprawność czasu i daty urządzenia (wewnętrzny zegar RTC), oraz komunikacja urządzenia z PC (analiza rozkazów). Schemat blokowy przebiegu działania programu ilustruje **rysunek 3**.



Rysunek 4. Zużycie ładunku baterii. A/C – pomiar napięcia, ADXL ON – włączenie akcelerometru, AM- normalny tryb pracy, LPM0/3 – tryb uśpienia, FRAM WR – zapis alarmu w pamięci FRAM

Pętla główna programu wykonywana jest w czasie 1 s. Przez większość czasu jej realizacji mikrokontroler przebywa w trybie uśpienia LPM3. Wówczas pobór prądu urządzenia wynosi 8 µA. W trybie normalnej pracy pobór prądu urządzenia wzrasta do 212 µA. Włączony akcelerometr konsumuje 400 µA, a pomiar napięcia analogowego dodatkowe 276 µA. W trakcie wykonania pętli głównej programu urządzenia pobiera z baterii ładunek o wartości 16,9172 µAs [µC]. W momencie wykrycia alarmu mikrokontroler potrzebuje czasu na jego obsługę (min. logowanie alarmu w pamięci FRAM). Skraca się czas przebywania MSP430 w trybie uśpienia LPM3 wzrasta pobór ładunku. Im więcej alarmów jest zgłaszanych tym czas pracy urządzenia na jednym komplecie baterii jest krótszy. W graficzny sposób zużycie ładunku elektrycznego baterii ilustruje **rysunek 4**.

Żeby obliczyć czas pracy urządzenia na baterii należy wykonać obliczenia matematyczne korzystając ze wzorów zamieszczonych w **ramce**. Przykładowo montując w urządzeniu baterię CR2032 o pojemności 240 mAh (przy 10 alarmach na godzinę) możemy zagwarantować 591 dni ciągłej pracy urządzenia. Jest to zadowalający wynik, choć trzeba przyznać, że MSP430 z pamięcią FLASH były bardziej energooszczędne (w trybie uśpienia LPM3 mikrokontroler pobierał 0,8 µA, zamiast 6,8 µA).

Przejdźmy do obsługi pamięci FRAM. Zamontowany w zestawie startowym mikrokontroler MSP430FR5739 ma (15,5 kB + 256 B) pamięci FRAM (pamięć programu + pamięć informacyjna) oraz 1 kB pamięci SRAM. Właściwości pamięci FRAM pozwalają, zrezygnować z użycia pamięci SRAM w projekcie. Mimo tego autor postanowił w pamięci SRAM ulokować stos i stertę programu, oraz zmienne typu „no init". W pamięci FRAM umieszczono zmienne globalne programu! (konfiguracja pamięci w pliku: *lnk430FR5739.xcl*). Dodatkowo pamięć programu została podzielona na 3 segmenty/partycje (kod programu, dane rejestratora, wektor przerwań), a autor zdefiniował zasady dostępu do pamięci (konfiguracja w module MPU). Segmenty w których umieszczono kod aplikacji oraz wektor przerwań posiadają prawa do wykonania, partycja z danymi rejestratora (alarmami) prawa do odczytu i zapisu, natomiast pamięć informacyjna ma ustawione prawa tylko do odczytu (**rysunek 5**).

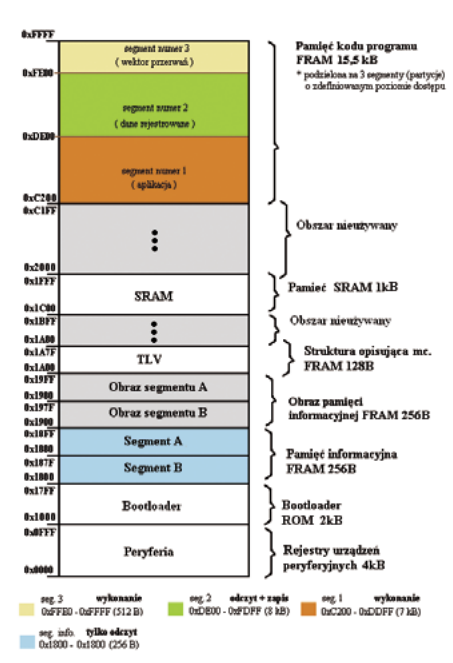
W przypadku naruszenia zasad dostępu do pamięci (np.: próba zapisania danych, w obszarze pamięci informacyjnej), wykonywany jest restart PUC mikrokontrolera. Program główny zlicza takie sytuacje (moduł diagnostyka), a liczniki błędów można odczytać (rozkaz ^Diagnostyka?). Dodatkowo w programie wykrywane są błędy pamięci FRAM (moduł FRCTL). Wykrywane są tzw. błędy podwójne (błąd krytyczny, niepoprawna wartość

Tabela 1. Protokół komunikacyjny

Rozkaz*	Opis	Przykład
^ Info?	Odczytaj informacje o projekcie.	Info? Elektronika Praktyczna 08/12 FRAM PROJEKT, wersja 1.4 Autor: Łukasz Krysiewicz .::WYKONANO::.
^ Diagnostyka?	Pobranie informacji o działaniu programu. (restart watchdog, błędy pamięci FRAM, naruszenie zasad dostępu do pamięci FRAM)	^ Diagnostyka? wdt: 0, framDB: 0, framSB: 0 mpuInfo: 0, mpuSeg1: 0, mpuSeg2: 0, mpuSeg3: 0 .::WYKONANO::.
^ Data=	Ustaw datę. Format: rok-miesiąc-dzień	^ Data=2012-04-10 .::WYKONANO::.
^ Czas=	Ustaw czas urządzenia. Format: godzina:minuta:sekunda	^ Czas=22:03:00 .::WYKONANO::.
^ Data?	Odczytaj datę.	^ Data? 2012-04-10 .::WYKONANO::.
^ Czas?	Odczytaj czas.	^ Czas? 22:06:05 .::WYKONANO::.
^ Rejestrator!	Kasuj dane rejestrowane.	^ Rejestrator! .::WYKONANO::.
^ Rejestrator?	Odczytaj dane rejestrowane.	^ Rejestrator? 2012-04-10 22:17:50 ZAMKNIECIE 2012-04-10 22:18:11 OTWARCIE 2012-04-10 22:18:15 ZAMKNIECIE .::WYKONANO::.

*rozkaz - strumień danych rozpoczynający się znakiem' ^ ' zakończony znakiem nowej linii <cr><lf>
- prawidłowe wykonanie rozkazu sygnalizowane jest komunikatem .::WYKONANO::.
- odebranie rozkazu spoza zdefiniowanej listy generuje komunikat .::NIEZNANY ROZKAZ::.
- w przypadku błędu wykonania rozkazu urządzenie zwraca komunikat .::BLAD::.

komórki pamięci, restart PUC mikrokontrolera), oraz błędy pojedyncze (błąd pamięci który został skorygowany, poprawna komórka pamięci). W obu przypadkach wystąpienia błędów są zliczane i podobnie jak w przypadku naruszenia zasad dostępu do pamięci mogą zostać odczytane poleceniem diagnostycznym. Zapis alarmów do pamięci FRAM autor zrealizował korzystając z modułu transmisji DMA, oraz trybu uśpienia LPM0. Równie dobrze dane do pamięci FRAM można zapisywać korzystając z funkcji języka „C" memcpy, bądź też z operacji rzutowania, przypisania. (identycznie jak w przypadku pamięci SRAM). Dodatkowo w programie autor wykorzystał zmienne typu __persistent (zmienne inicjowane tylko raz w momencie programowania mikrokontrolera). W połączeniu z właściwościami pamięci FRAM zmienne tego typu dają nową jakość w programowaniu mikrokontrolerów. Można tworzyć „nieulotne zmienne" (bo zapisane w pamięci FRAM), których nie trzeba inicjować po starcie mikrokontrolera.



Rysunek 5. Mapa pamięci MSP430FR5739

Czas pracy urządzenia – obliczenia

Żeby określić czas pracy rejestratora na jednym komplecie baterii trzeba obliczyć średni pobór ładunku elektrycznego. Korzystając z danych prezentowanych na rysunku 4 obliczamy pobór ładunku w trakcie wykonania pętli głównej programu (pętla bez alarmu wzór 1, pętla z obsługą alarmu wzór 2).

(1) $$Q1 = \sum_{5}^{i=1} Q_i$$

(2) $$Q2 = i_1 * (t_1 - \sum_{8}^{6} t_i) + \sum_{5}^{i=2} Q_i + \sum_{8}^{i=6} Q_i$$

gdzie: $Q_i = (i_i * t_i)$, dla i=1,2,..,8 ([uAs]=[uA]*[s])

Następnie uwzględniając okresowość zgłaszania alarmów (alarm co N sekund) otrzymujemy wzór 3 opisujący średni pobór ładunku elektrycznego rejestratora:

(3) Qs [uAs]=((N-1)*Q1+Q2)/N

Odczytujemy pojemność baterii (zazwyczaj podawana w mAh), podstawiamy wartości do wzoru 4 i obliczamy czas pracy rejestratora na jednym komplecie baterii (*).

(4) T [s]=C[As **]/Qs[As]

* przybliżony czas pracy urządzenia na jednym komplecie baterii, w praktyce czas działania urządzenia zależy od kilku dodatkowych zmiennych (zjawisko samorozładowania baterii, spadek napięcia baterii pod koniec jej okresu żywotności, wpływ temperatury otoczenia na charakterystykę pracy baterii, etc.)
** konwersja jednostek C[As]=C[mAh]/1000*3600, Q[As]=Q[uAs]/10^6

Uruchomienie

Należy zainstalować środowisko IAR, otworzyć projekt rejestratora (FR-PRJ.eww), do portu USB komputera PC podłączyć zestaw startowy MSP-EXP430FR5739, oraz zaprogramować mikrokontroler (*Project–> Download Active Application*). Następnie należy obserwować stan diod komunikacyjnych urządzenia. Włączona dioda LED8, sygnalizuje błąd oscylatora kwarcowego. Jest to błąd krytyczny, który należy niezwłocznie usunąć (poprawić jakość montażu kwarcu zegarkowego XT1). Włączone diody LED7, LED6 sygnalizują niepoprawną wartość daty i czasu urządzenia. Żeby ustawić zegar urządzenia należy uruchomić terminal transmisji UART (dostępny w systemie Windows HyperTerminal, bądź inny), ustawić parametry transmisji 8N1 9600 BAUD i wykonać rozkazy „ustaw date", „ustaw czas" (opis w **tabeli 1**). W momencie, gdy urządzenie przestanie zgłaszać błędy (diody sygnalizacyjne zostaną wyłączone), należy podłączyć zasilanie bateryjne, odłączyć przewód USB od portu komputera PC, oraz zdemontować zworki konfiguracyjne zestawu *MSP-EXP430FR5739*. Po wykonaniu wyżej opisanych operacji urządzenie jest gotowe do pracy. Rejestrator może zalogować 8110 alarmów (10 bajtów na rekord). W momencie zapełnienia pamięci rejestratora włączana jest dioda sygnalizacyjna LED5. Alarmy zalogowane w rejestratorze możemy odczytać za pomocą rozkazu: ^ Rejestrator?. Pamięć rejestratora „kasujemy" (zerujemy indeks logera) korzystając z rozkazu: ^ Rejestrator! .

Podsumowanie

Szczegółowy opis działania rejestratora zamieszczono w plikach źródłowych projektu (opis kodu programu). Czytelnicy zainteresowaniu dodatkowymi informacjami na temat działania urządzenia i pracy z pamięcią FRAM, proszeni są o bezpośredni kontakt z autorem.

Łukasz Krysiewicz
lukasz_krysiewicz@interia.pl