MSP430 Termometr z rejestracją

Przykładowa aplikacja na podsumowanie kursu

W styczniowym numerze Elektroniki Praktycznej (EP 01/2009) rozpoczęliśmy publikację kursu programowania mikrokontrolerów MSP430 (seria x1xx). Na zakończenie kursu autor prezentuje projekt termometru z możliwością rejestracji zmierzonej temperatury. Projekt został wykonany w oparciu o dostępną w ofercie AVT płytkę ewaluacyjną eMeSPek.

Sprzęt

Urządzenie testowano i uruchamiano na platformie sprzętowej eMeSPek. Przed rozpoczęciem pracy z płytką, należy wykonać zmianę w mozaice obwodów drukowanych. Konieczne jest przecięcie pokazanej na **fot. 1** ścieżki drukowanej. Modyfikacja układu sprawi, że zworka JP4 zacznie pełnić swoją funkcję.

Zworka JP4 odpowiada za konfigurację źródła zasilania układu. Ustawiona w pozycji 1-2 powoduje, że układ jest zasilany z baterii. Konfiguracja 2-3, doprowadza do układu zewnętrzne zasilanie. W aplikacji wykorzystywane będą diody LED, więc przed jej uruchomieniem zworki JP1 oraz JP2 należy ustawić w pozycji 1-2. Konfiguracji wymaga również źródło sygnału zegarowego LFXT1CLK. Zastosowano kwarc zegarkowy XT2 o częstotliwości 32768 Hz. W związku z tym zworki JP5 i JP6 należy ustawić w pozycji 1-2. Pomiary analogowe nie korzystają z zewnętrznego napięcia referencyjnego, nie ma więc konieczności dołączania do pinu P2.4 kondensatora C10 i zworka JP7 może zostać zdjęta, albo ustawiona w pozycji 1-2.

Funkcjonalność

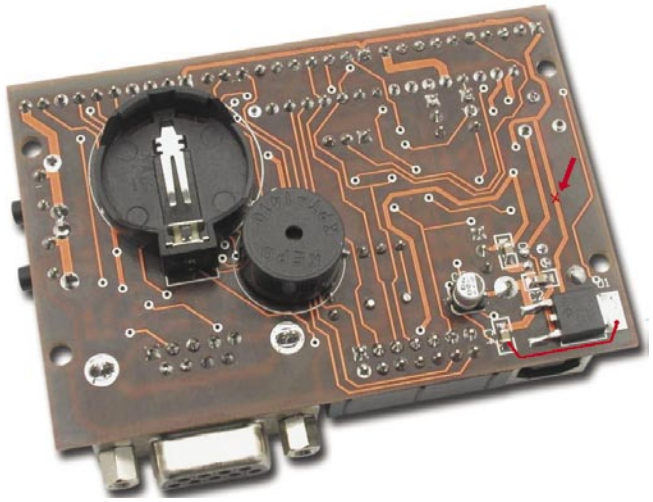
Jak sama nazwa wskazuje, termometr mierzy temperaturę otoczenia. Pomiar wykonywany jest cyklicznie. Użytkownik ma możliwość konfiguracji czasu pomiędzy pomiarami. Minimalna wartość to 1 sekunda, maksymalna – 65534 sekund (ok. 1,5 miesiąca). Poza cyklicznym pomiarem temperatury możliwe jest wykonanie pomiaru na żądanie. W tym celu należy wcisnąć jeden z dostępnych na płycie eMeSPek przycisków – SW1 lub SW2.

Po zakończeniu pomiaru, wynik zapisywany jest w wewnętrznej pamięci Flash mikrokontrolera. Wraz z temperaturą do pamięci zapisywany jest czas wykonania pomiaru. W sumie to 8 bajtów danych.

Maksymalnie w pamięci można zapisać 448 rekordów danych (czas+temperatura). W dowolnym momencie pracy urządzenia użytkownik ma możliwość odczytu zarejestrowanych danych. Istnieje również możliwość skasowania danych i rozpoczęcia cyklu pomiarowego od początku.

Organizacja pamięci FLASH

Darmowa wersja środowiska IAR posiada ograniczenie na rozmiar kodu wynikowego. Ograniczenie dotyczy projektów pisanych w języku C oraz C++. W przypadku tego typu projektów; czyli również projektu przenośnej stacji meteorologicznej; maksymalny rozmiar kodu wynikowego nie może być większy niż 4 kB. Należy pamiętać, że ograniczenie dotyczy jedynie wielkości kodu stworzonego przez programistę. Używając wbudowanych bibliotek (np.: funkcji strlen, sprintf) można wygenerować kod o większej objętości.

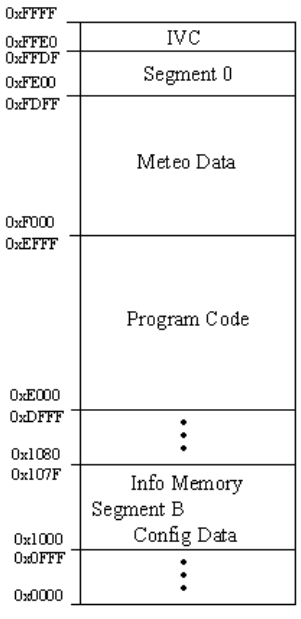


Fot. 1.

Mikrokontroler MSP430 f1232 ma 8 kB pamięci FLASH przeznaczonej na kod programu. Część z tej pamięci wykorzystano do rejestracji zmierzonej temperatury.

Do projektu dołączono plik lnk430F1232.xcl (*Project->Options, Linker->Config*) opisujący pamięć FLASH mikrokontrolera. W pliku tym obszar pamięci kodu programu ograniczono do 4 kB (–P(CODE-)CODE=E000–EFFF). Uzyskane w ten sposób miejsce przeznaczono na rejestrowane dane (–P(CODE)METEO=F000–FDFE). W sumie przeznaczono na nie 3584 B pamięci FLASH, co przy rozmiarze jednego rekordu danych 8 B daje możliwość zarejestrowania 448 rekordów. Dane rejestrowane są począwszy od adresu 0xF000, aż do momentu zapełnienia się pamięci. Wówczas pomiar temperatury oraz rejestracja danych zostają wstrzymane i włączana jest dioda D1, której świecenie informuje użytkownika o zapełnieniu się pamięci i tym samym zakończeniu cyklu pomiarowego.

W projekcie wykorzystywana jest pamięć informacyjna mikrokontrolera. W segmencie B tejże pamięci, pod adresem 0x1000, zapisywany jest parametr konfigurujący czas pomiędzy pomiarami.



Rys. 2.

Programowanie

Zworkę JP4 należy ustawić w pozycji 2-3. Następnie podłączyć zewnętrzne zasilanie oraz programator. Otworzyć projekt `Meteo.eww` i zaznaczyć w opcjach debuger kasowanie pamięci programu oraz pamięci informacyjnej podczas programowania. Po zaprogramowaniu mikrokontrolera (*Project->Debug*), uruchomić program (*Debug->Go*), a następnie wyjść z debugera (*Debug->Stop Debugging*). Urządzenie zostało zaprogramowane. Można odłączyć programator, usunąć zworkę JP4 oraz odłączyć zewnętrzne zasilanie.

IAR – tricks & tips
W trybie debugera środowiska IAR, programista ma możliwość podglądania pamięci FLASH mikrokontrolera (View->Memory). Programując urządzenie (*Project->Options, FET Debugger*) ma natomiast możliwość wyboru czy będzie kasowana tylko pamięć programu (*Erase main memory*), czy też wraz z nią pamięć informacyjną (*Erase main and Information memory*). W pamięci informacyjnej znajduje się parametr konfiguracyjny, a w pamięci programu – poza kodem programu – rejestrowane dane. Jeśli, programista nie chce kasować znajdujących się w pamięci danych, to może skorzystać z trzeciej dostępnej opcji programowania urządzenia (*Retain unchanged memory*). Wówczas kod programu zostanie podmieniony, a obszar danych konfiguracyjnych oraz danych rejestrowanych pozostaną nienaruszone.
W projekcie pamięć kodu programu ograniczono do 4 kB. Kod wynikowy aplikacji nie może być większy niż zadeklarowana wielkość pamięci. Z tego powodu w projekcie włączono optymalizacją kodu (*Project->Options, C/C++ compiler->Optimizations*). Rozmiar kodu wynikowego programu programista może sprawdzić w oknie kompilacji projektu (*Project->Rebuild All*). Jeśli programista potrzebuje bardziej szczegółowych informacji o kodzie wynikowym, to powinien w opcjach projektu, w kategorii *C/C++ compiler*, zakładce *List* zaznaczyć opcję *Output list file*. Wówczas na etapie kompilacji, dla każdego z pliku projektu zostanie wygenerowany plik z rozszerzeniem *.lst* (katalog *Debug/List*). W pliku poza listingiem kodu programu znajdą się informacje na temat zajętości stosu programu, oraz objętości kodu wynikowego programu. Informacje te są wręcz niezbędne podczas optymalizacji oprogramowania.
W aplikacji układ watchdog odświeżany jest co 0,5 sekundy. Jeśli programista zamierza debugować projekt, to powinien zatrzymać układ watchdog. W projekcie (*Project->Options, C/C++ compiler->Preprocessor*) zadeklarowano dyrektywę preprocesora o nazwie `WDT_ON`. Zmiana tej nazwy powoduje, że na etapie kompilacji generowany jest kod programu z wyłączonym układem watchdog.
Przed rozpoczęciem debugowania projektu, należy wyłączyć aktywne pułapki sprzętowe. Zaprogramować mikrokontroler i dopiero wówczas ustawić pułapki. W przypadku mikrokontrolerów serii x1xx, programista ma możliwość ustawienia od 2 do 8 pułapek sprzętowych (w zależności od typu mikrokontrolera). Ograniczenie liczby pułapek sprzętowych bywa uciążliwe. Można je jednak ominąć. W opcjach projektu należy wybrać stosowanie pułapek programowych (*Options, FET Debugger, Breakpoints->Use Software Breakpoints*). Wówczas możliwe będzie korzystanie z nieograniczonej liczby pułapek. Niestety nie ma niczego za darmo. Decydując się na korzystanie z pułapek programowych godzimy się na to, że podczas debugowania program nie będzie wykonywany w czasie rzeczywistym.

Uruchomienie, konfiguracja

Aby płytka eMeSPek stała się urządzeniem przenośnym, należy w koszyku B1 umieścić baterię typu CR2032, a następnie przełączyć zasilanie układu na bateryjne (JP4 w poz. 1-2). Rozpocznie się inicjalizacja układów peryferyjnych, a następnie zostanie wykonany pomiar napięcia zasilania mikrokontrolera – w naszym przypadku napięcia baterii. Jeśli zmierzona wartość będzie mniejsza niż 2,7 V, to zostaną włączone diody D1 oraz D2, a urządzenie przejdzie w tryb pracy awaryjnej. Przejście w tryb awaryjny oznacza wstrzymanie normalnej pracy urządzenia. Zabezpieczenie zostało wprowadzone ze względu na minimalną wartość napięcia wymaganą podczas zapisu i kasowania pamięci FLASH. W przypadku serii x1xx do programowania pamięci wymagane jest napięcie o wartości co najmniej 2,7 V.

Jeśli po uruchomieniu urządzenia diody nie są włączone, oznacza to, że bateria dostarcza napięcia o właściwej wartości, a urządzenie przeszło w tryb pracy normalnej. Oprogramowanie rozpoczyna przeszukiwanie pamięci danych rejestrowanych w celu znalezienia indeksu ostatnio zarejestrowanej próbki (algorytm połowienia). Inicjowany jest parametr określający czas pomiędzy pomiarami. Następnie rozpoczyna się wykonywanie pętli głównej programu. W związku z tym, że urządzenie nie zostało skonfigurowane, to nie posiada pełnej funkcjonalności (nie są wykonywane cykliczne pomiary temperatury). Uruchomione zostały: moduł zegara RTC (od bazowej wartości 2001-01-01 00:00:00), komunikacja RS-232, oraz pomiary na żądanie.

Aby urządzenie było w pełni funkcjonalne należy je skonfigurować. W tym celu konieczne jest podłączenie urządzenia do komputera z uruchomionym terminalem portu szeregowego. Parametry pracy terminala to 8N1, wyłączona kontrola sprzętową, prędkość transmisji 9600 bps. Komunikacja z układem odbywa się przy pomocy komend tekstowych opisanych w tabeli numer 1. Komendy muszą być zakończone znakiem CR, gdyż tylko wówczas są analizowane przez wbudowane oprogramowanie mikrokontrolera.

Na początku należy ustawić datę, oraz czas („DateTime="), zgodnie z formatem „rr-mm-dd gg:mm:ss". Aktualny czas i datę można odczytać przy pomocy komendy „DateTime?" i tym samy upewnić się czy moduł programowego zegara RTC pracuje prawidłowo. Kolejnym etapem konfiguracji urządzenia jest zdefiniowanie czasu pomiędzy pomiarami. Czas jest podawany w sekundach, a dostępne nastawy parametru to od 1 sekundy do 65534 sekund (około 1,5 miesiąca). Po ustawieniu parametru („Interval="), i upłynięciu zdefiniowanego czasu urządzenie rozpoczyna cykliczne pomiary temperatury oraz zapis wyników w pamięci danych rejestrowanych. Użytkownik może zatrzymać pomiary, konfigurując czas pomiędzy pomiarami na jedną z wartości 0, albo 65535.

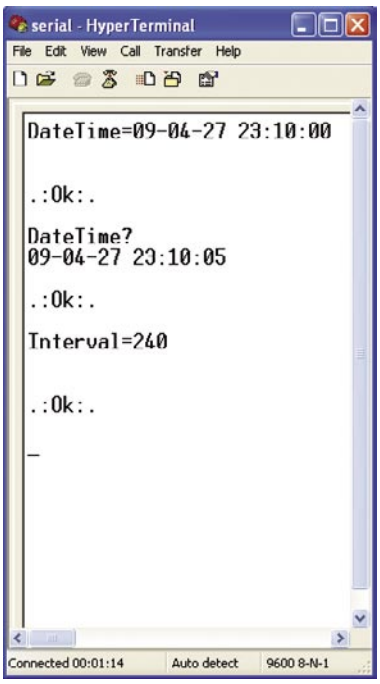
W dowolnym momencie działania urządzenia użytkownik może odczytać zarejestrowane parametry („MeteoData?"). Możliwe jest również skasowanie danych rejestrowanych („Erase!"). Podczas kasowania pamięci danych rejestrowanych włączona jest dioda D1. Proces kasowanie trwa niecałe 90 ms, dlatego też użytkownik może zaobserwować jedynie krótkotrwałe mignięcie diody D1.

Po zakończeniu konfiguracji (**rys. 3**), termometr jest gotowy do użycia. Autor artykułu, wykorzystał urządzenie do pomiaru temperatury w przydomowym ogródku. Pomiary wykonywane były co 4 minuty przez 24 godziny, a ich wyniki były zapisywane w pamięci danych. Rezultat pomiarów pokazano na **rys. 4**.

Struktura programu

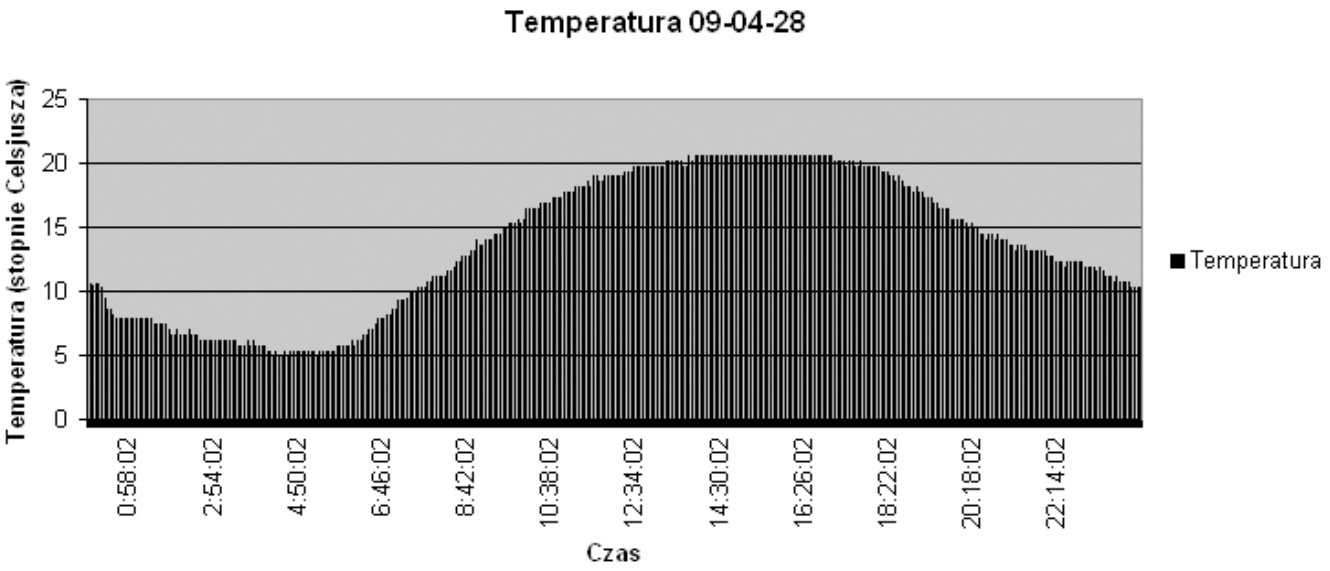
Aby maksymalnie wydłużyć czas pracy urządzenia na jednym komplecie baterii, wbudowane oprogramowanie zostało zoptymalizowane pod kątem energooszczędności. Jako źródło sygnału LFXT1CLK skonfigurowano kwarc niskiej częstotliwości (kwarc zegarkowy 32768 Hz). Częstotliwości sygnałów zegarowych mikrokontrolera dobrano następująco: ACLK=LFXT1CLK=32768 Hz, MCLK=SMCLK =DCO= ok. 720 kHz.

Przez większość czasu działania programu mikrokontroler przebywa w trybie uśpienia LPM3 (pobór prądu 0,7 μA, ACLK pozostaje włączone, pozostałe sygnały zegarowe są wyłączone). Przerwania od modułów peryferyjnych budzą CPU. W momencie zgłoszenia przerwania uruchamiany jest zegar DCO. Po około 6 μs (czas startu DCO), mikrokontroler przechodzi w tryb pracy aktywnej AM (wszystkie sygnały zegarowe są włączone, pobór prądu ok. 170 μA), wykonuje zdefiniowane w kodzie programu zadania, po czym powraca w tryb uśpienia LPM3.



Rys. 3

W trybie normalnej pracy mikrokontroler jest budzony w wyniku przerwań generowanych przez moduł zegara TimerA. Cyklicznie, zgłaszane są przerwania od liczników TACCR0, TACCR1. Przerwania od licznika TACCR1 są zgłaszane co sekundę. W procedurze obsługi przerwania odświe-



Rys. 4.

żany jest rejestr układu watchdog (czas wykonania kodu programu około 35 µs).

Licznik TACCR0 również generuje przerwania, co jedną sekundę. W procedurze obsługi przerwania odświeżany jest również układ watchdog. Dodatkowo uaktualniany jest czas zegara RTC, a oprogramowanie sprawdza czy powinno rozpocząć (tzw. cykliczny) pomiar danych rejestrowanych (czas wykonania około 75 µs). Reasumując, zaledwie przez czas około 110 µs mikrokontroler przebywa w trybie aktywnym AM (170 µA), pozostałą część sekundy pozostaje uśpiony LPM3 (0,7 µA).

W momencie, gdy w procedurze obsługi przerwania od licznika TACCR0 zostanie wykryta potrzeba wykonania pomiaru danych rejestrowanych, nastąpi opuszczenie trybu energooszczędnego LPM3. Oprogramowanie rozpocznie wykonywanie pętli głównej aplikacji. W niej zostanie wykonany pomiar temperatury (około 800 µs), oraz zapis danych do pamięci FLASH (około 1 ms). Następnie oprogramowanie powróci w tryb normalnej pracy. W identyczny sposób aplikacja zachowa się w przypadku wykrycia potrzeby wykonania pomiaru na żądanie (procedura obsługi przerwań portu drugiego).

Opuszczenie trybu pracy LPM3 i przejście do wykonywania pętli głównej ma również miejsce w przypadku procedury obsługi przerwania od modułu komunikacji UART. Sytuacja taka ma miejsce w momencie, gdy zostanie wykryte odebranie poprawnej komendy (zakończonej znakiem CR). W pętli głównej odbędzie się analiza odebranej komendy, ewentualne jej wykonanie oraz wysłanie odpowiedzi. Podczas trwania wymienionych operacji, pomiar danych rejestrowanych będzie wstrzymany (odczyt danych, kasowanie danych, etc.).

Komunikacja pomiędzy procedurami przerwań, a pętlą główną oprogramowania odbywa się przy wykorzystaniu procedur zdefiniowanych w pliku `Control.c`.

W trakcie wykonywania pętli głównej oprogramowania mikrokontroler przebywa w trybie pracy aktywnej AM. Pobór prądu wynosi około 170 µA. Należy jednak pamiętać, że pomiar temperatury oraz zapis danych do pamięci FLASH, wpływają na zwiększenie poboru energii.

Źródła projektu

Projekt termometru jest swego rodzaju podsumowaniem cyklu artykułów na temat MSP430. Wykorzystano w nim wszystkie moduły peryferyjne omawiane we wcześniejszych artykułach. Pliki źródłowe projektu umieszczono na płycie CD-EP5/2009A. Aplikację napisano w taki sposób, aby jej wartość dydaktyczna była jak największa (pomiar temperatury z wewnętrznego czujnika, sekcja danych konfiguracyjnych, danych pomiarowych, implementacja programowego zegar RTC i inne). Czytelnik ma możliwość modyfikacji kodu programu aplikacji. Może również wykorzystywać ów kod w przypadku własnych rozwiązań tworzonych w oparciu o energooszczędne mikrokontrolery MSP430.

Łukasz Krysiewicz

Tab. 1 – Komendy protokołu komunikacyjnego

Komenda	Opis	Format danych wejściowych	Odpowiedź	Przykład, wywołania oraz odpowiedzi.
`DateTime?`	Odczyt daty, oraz czasu.	Brak.	rr-mm-dd gg:mm:ss .:Ok.:.	DateTime? 09-04-28 16:56:23 .:Ok.:.
`DateTime=`	Ustawienie daty, oraz czasu.	rr-mm-dd gg:mm:ss	1) .:Ok.:. 2) .:Error:.	DateTime=09-04-28 16:44:02 .:Ok.:.
`Interval=`	Ustawinie czasu pomiędzy pomiarami.	Liczba z zakresu 0-65535.	1) .:Ok.:. 2) .:Error:.	Interval=240 .:Ok.:.
`MeteoData?`	Odczyt danych rejestrowanych.	Brak.	.:Ok.:. rr-mm-dd gg:mm:ss temperatura.	MeteoData? :Ok.:. 09-04-28 06:58:02 7.79 09-04-28 07:02:02 8.20 09-04-28 07:06:02 8.20
`Erase!`	Kasowanie danych rejestrowanych.	Brak.	.:Ok.:.	Erase! .:Ok.:.
Niepoprawna komenda	Wprowadzenie niepoprawnej komendy.	Brak.	.:Unknown:.	DaTeTiMe! .:Unknown:.