Pamięci nieulotne w systemach mikroprocesorowych, część 1

Konstruując urządzenia z mikrokontrolerami często stajemy przed koniecznością zapewnienia przechowywania danych także po wyłączeniu lub zaniku zasilania. Łatwo jest, jeśli są to stałe jak: napisy menu (dla przykładu w różnych językach), obrazy, stałe parametry nastaw. Gorzej, gdy musimy przechować zmienne. Jeszcze trudniej, gdy muszą one być zapamiętane również w przypadku awarii napięcia zasilania.

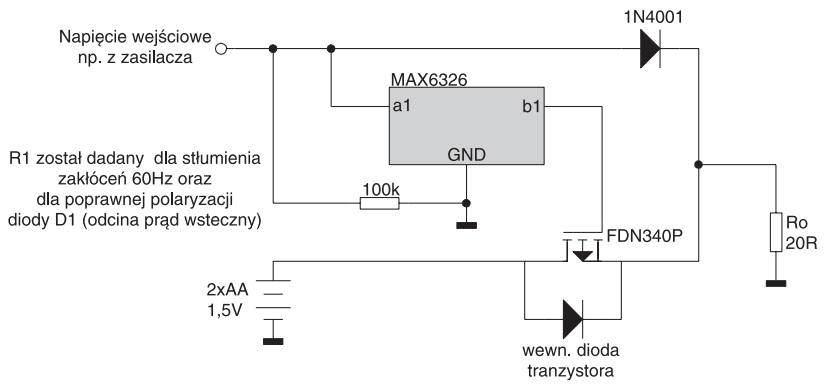
Jedną z najbardziej efektywnych i jednocześnie tanich metod przechowywania danych jest wykorzystanie pamięci RAM mikrokontrolera. Ma to jedną, ważną zaletę: w przypadku awarii napięcia zasilającego dane już znajdują się tam, gdzie będą potrzebne po przywróceniu „normalnych" warunków pracy. Nie trzeba zatem wykonywać żadnych dodatkowych kroków w celu ich przechowania. Oczywiście pozostaje kwestia kontroli, czy zapamiętane dane nie zostały uszkodzone, ale nie zmienia to cech, o których była mowa wcześniej. To od strony programowej. A jak wygląda to samo zagadnienie od strony sprzętowej?

Niestety, niezbędne są pewne dodatkowe połączenia i elementy zewnętrzne. Jak zapewne pamiętamy, pamięć DRAM jest zbudowana z wielu miniaturowych kondensatorów, które przechowują ładunek elektryczny niosący informację o stanie bitu. Pamięci DRAM wymagają odświeżania zawartości, czyli okresowego doładowywania kondensatorów. Pamięć SRAM nie wymaga odświeżania (a w taką zazwyczaj są wyposażane mikrokontrolery) i może przechowywać dane dowolnie długi czas, ale pod warunkiem zapewnienia zasilania. Rolę źródła zasilającego może pełnić kondensator elektrolityczny o bardzo dużej pojemności, akumulator lub bateria. Wszystko zależy od wymagań aplikacji. W swoich projektach często wykorzystuję baterię litową o napięciu 3 V. Zasilanie mikrokontrolera wyłącznie z baterii jest raczej nieefektywne, toteż stosuje się automatyczne przełączniki zasilania przełączające zasilanie mikrokontrolera pomiędzy źródłem stacjonarnym (np. zasilaczem) a baterią czy akumulatorem. Przełącznik można zbudować w oparciu o specjalizowany układ scalony lub za pomocą diod krzemowych lub lepiej - germanowych. Przykładami fabrycznych rozwiązań w formie układów scalonych mogą być MAX6326, -75, -81. Inne, bardziej skomplikowane, wyposażone są również w funkcję nadzoru napięcia zasilania MAX6365...68, wejście sygnału zewnętrznego *reset* (MAX6366) i układ *watchdog* (MAX6368). Przykłady ich zastosowań, zaczerpnięte z not aplikacyjnych producenta, umieszczono na **rys. 1** i **2**. Na **rys. 3** przedstawiono schemat najprostszego przełącznika diodowego.

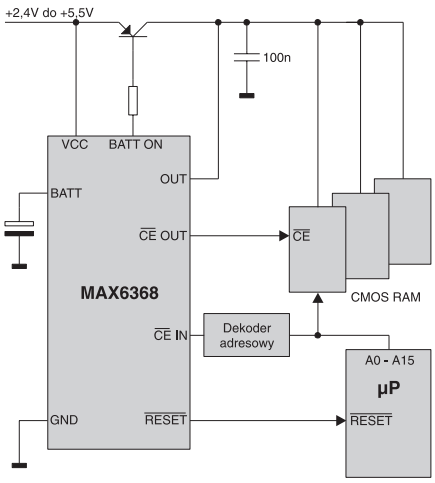
Oczywiście, oprócz przytoczonych, można zaproponować również wiele innych rozwiązań sprzętowych. Opracowując konstrukcje własnych przełączników, zwłaszcza w połączeniu z układem nadzoru napięcia zasilania, trzeba mieć na uwadze kilka istotnych czynników, między innymi:

- zanik napięcia zasilającego powinien zostać wykryty wcześniej niż pojawi się sygnał *reset* dla mikrokontrolera,
- napięcie zasilające z baterii musi być przełączone w sposób „pewny", eliminujący wahania napięcia mogące zakłócić pracę mikrokontrolera,
- spadek napięcia na diodzie krzemowej spolaryzowanej w kierunku przewodzenia to około 0,6 V, natomiast na diodzie germanowej 0,2 V; wartość tą należy odjąć od napięcia baterii zasilającej i ta różnica zasila mikrokontroler,
- prąd pobierany przez pamięć CMOS lub mikrokontroler w stanie IDLE jest tak mały, że również detekcyjne diody germanowe z powodzeniem mogą spełniać rolę przełącznika napięcia zasilającego; odnosząc tę uwagę do rys. 3: dioda D2 może być diodą germanową.

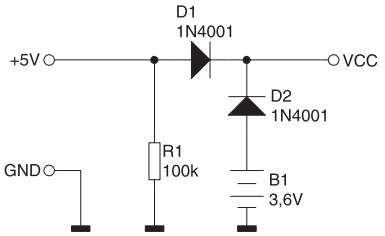
Układy mikrokontrolerów wprowadzone w tryb IDLE pobierają znikomą ilość energii i zachowują zawartość pamięci RAM. Typowo prąd potrzebny do zasilania w trybie IDLE ma wartość mniejszą niż 100 µA. W sytuacji jak opisywana wyżej, gdy do przechowywa-



Rys. 1. Przykład zastosowania układu MAX6326 do przełączania źródeł napięcia zasilającego



Rys. 2. Przykład użycia układu MAX6368 do przełączania źródła napięcia zasilającego oraz generowania sygnału *reset* dla mikrokontrolera



Rys. 3. Budowa taniego, diodowego przełącznika napięcia zasilania

nia danych używana jest wewnętrzna pamięć CMOS, układ mikrokontrolera musi być zasilany z baterii a zanik napięcia zasilającego powinien być sygnalizowany w celu wprowadzenia mikrokontrolera w stan obniżonego poboru energii (IDLE). Abstrahując od rozwiązań sprzętowych, mikrokontroler należy wyposażyć w tak zwane wejście pomiarowe. Oczywiście dla większości zastosowań wystarczające jest wejście cyfrowe pozwalające stwierdzić obecność stanu logicznego wysokiego lub niskiego przyłożonego do wyprowadzenia.

Często do sygnalizacji zaniku napięcia wykorzystuje się wejście zewnętrznego przerwania. Dla mikrokontrolera z rodziny 8051, wejścia te oznaczane są jako INT0 i INT1. Doskonale nadają się one do takiej sygnalizacji, ponieważ każde z wejść posiada przerzutnik Schmitta pozwalający jednoznacznie określić stan logiczny. Dodatkowo poziom napięcia (lub jego zmianę) na wejściu łatwo jest powiązać z odpowiadającym mu fragmentem programu. Fragment ten może zawierać na przykład ustalenie stanu portów wyjściowych oraz wprowadzenie mikrokontrolera w tryb obniżonego poboru energii. Przykład takiej funkcji obsługi przerwania zewnętrznego przedstawiono na **list. 1**.

W tym miejscu ważna uwaga: często programista wykorzystując wejście przerwania zewnętrznego w sposób taki, jak opisałem wyżej (sam również popełniałem taki błąd) ustawia bit powodujący wygenerowanie sygnału przerwania w momencie pojawienia się opadającego zbocza sygnału na wejściu INTx. Funkcja może zawierać rozkaz wprowadzenia mikrokontrolera w tryb IDLE w celu obniżenia poboru energii z baterii. Później pojawia się sygnał *reset* generowany przez układ nadzorcy napięcia zasilającego. Jeśli układ ten nie jest zasilany z baterii (a najczęściej w celu oszczędzania energii tak jest), to aktywny *reset* zaniknie wraz z zanikiem napięcia zasilania całego obwodu poza mikrokontrolerem. W takiej sytuacji, zasilany energią z baterii mikrokontroler, zaczyna wykonywać program począwszy od adresu 0, ponieważ aktywny sygnał *reset* wyprowadził go z trybu IDLE a opadające zbocze sygnału na wejściu przerwania nie pojawi aż do momentu załączenia i ponownego wyłączenia zasilania. A my dziwimy się, gdzie podziewa się energia z baterii, która powinna wystarczyć na co najmniej pół roku pracy...

Omówiony wyżej problem można rozwiązać na szereg różnych sposobów, zarówno programowo jak i sprzętowo. Najprostszym jest programowe testowanie wejścia pomiarowego lub ustawienie przerwań aktywowanych poziomem niskim sygnału a nie jego zboczem. Oczywiście zarówno część sprzętowa jak i programowa muszą ze sobą współgrać w celu osiągnięcia jak najlepszego efektu. Pamiętajmy, że mimo wprowadzenie w stan IDLE mikrokontroler nadal jest zasilany! W przypadku niektórych aplikacji ważnym może być również właściwe ustawienie stanów portów wejściowych i wyjściowych. Praktyka pokazuje, że mimo iż porty nie powinny być aktywne i dopuszczać do „wycieków" energii

List. 1. Przykład procedury obsługi przerwania wprowadzającej mikrokontroler w stan obniżonego poboru energii IDLE

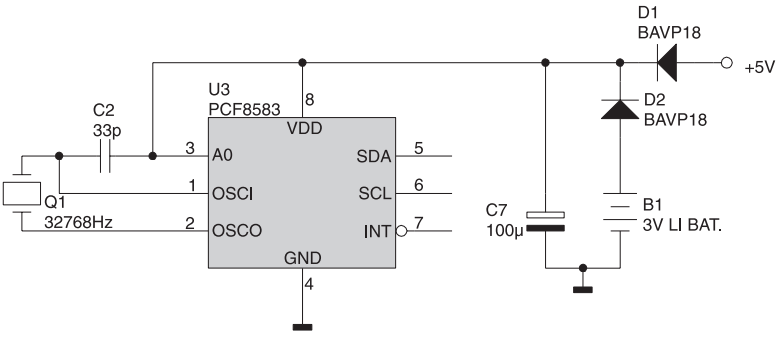
```
;********** PRZERWANIE ZEWN.INT1 ****************
IrqExt1:
clr    EA
   mov P1,INIT_FOR_P1
   mov P2,INIT_FOR_P2
   mov P3,INIT_FOR_P3
   orl PCON,#3            ;wyłączenie mikrokontrolera, przechowywanie zawartości RAM
   ajmp   $               ;ponowne załączenie po sygnale RESET (PD=IDL=1 -> PD)
```

List. 2. Przykład realizacji funkcji zapisu/odczytu danych w EEPROM

```
;***************** OBSŁUGA DANYCH PAMIĘTANYCH W EEPROM **********************
;Testowanie bitu gotowości pamięci EEPROM po operacji zapisu
EE_BusyTest:  mov A,WMCON
   anl A,#00000010B
   jz  EE_BusyTest
   ret

;Wyłączenie EEPROM
EE_Disable:   anl WMCON,#11100111B
   ret

;Zapamiętanie bajtu danych w pamięci EEPROM,16-bitowy adres w DPTR, bajt w ACC
;po odczytaniu bajtu wartość DPTR jest zwiększana o 1
EE_SaveByte:  orl WMCON,#00011000B     ;Włączenie trybu zapisu
   movx   @DPTR,A                      ;Zapis bajtu
call   EE_BusyTest                     ;Test zakończenia operacji
   inc DPTR
   ret

;Odczyt bajtu danych z pamięci EEPROM,16-bitowy adres w DPTR, bajt zwracany w ACC
;po odczytaniu bajtu wartość DPTR jest zwiększana o 1
EE_ReadByte:  anl WMCON,#11100111B
   orl WMCON,#00001000B                ;Tryb odczytu
   movx   A,@DPTR                      ;Odczyt bajtu
   inc DPTR
   ret
```

List. 3. Przykład użycia funkcji do obsługi pamięci EEPROM z list. 2

```
;Zapamiętanie parametrów ustawionych po kalibracji
EE_SaveParams:
   mov DPTR,#EE_PLCSTADDR
   mov A,PLCSTATE              ;<- nastawy maszyny
   call   EE_SaveByte
   mov A,PRNR                  ;<- numer aktywnego programu
   call   EE_SaveByte
   mov A,CALSTEPS
   call   EE_SaveByte
   mov A,CALSTEPS+1            ;<- liczba kroków kalibracji
   call   EE_SaveByte
   mov A,CALSTEPS+2
   call   EE_SaveByte
   call   EE_Disable
   ret

;Odczyt parametrów systemowych
EE_RestParams:
   mov DPTR,#EE_PLCSTADDR
   call   EE_ReadByte
   mov PLCSTATE,A
   call   EE_ReadByte
   cjne   A,#0FFH,EE_RestP1    ;<- numer aktywnego programu
   clr A
   mov PRNR,A
   call   EE_ReadByte
   mov CALSTEPS,A
   call   EE_ReadByte
   mov CALSTEPS+1,A            ;<- liczba kroków kalibracji
   call   EE_ReadByte
   mov CALSTEPS+2,A
   call   EE_ReadProg          ;<- odczyt ostatniego programu
   ret
```



Rys. 4. Również układ popularnego RTC zawiera 240 bajtów pamięci RAM, którą można wykorzystać do przechowywania danych

zasilającej, to bywa z tym różnie w przypadku różnych mikrokontrolerów od różnych producentów.

Innym zagadnieniem jest właściwa inicjalizacja zmiennych znajdujących się w pamięci RAM. Należy bardzo uważać na wszelkiego rodzaju polecenia testujące stan RAM. Często działają one w taki sposób, że zapisują do i odczytują z pamięci pewną wartość. Dobrze jest w pamięci mikrokontrolera wydzielić dla tego celu pewien obszar a zawarte w nim dane opatrzyć dodatkowo sumą kontrolną.

Inną możliwość tworzy dodanie zewnętrznej pamięci CMOS RAM. Najlepiej, gdy jest to pamięć z interfejsem szeregowym SPI lub I2C. Wówczas cały ciężar związany z obsługą jej spoczywa na oprogramowaniu. Nie jest konieczne projektowanie i budowa skomplikowanych dekoderów adresu. Jednak w przypadku użycia pamięci zewnętrznej, należy dobrze przemyśleć zarówno budowę sprzętową jak i sposób funkcjonowania programu. O ile bowiem w przypadku użycia wewnętrznego RAM mikrokontrolera do jego zawartości można uzyskać dostęp już po wykonaniu pojedynczego rozkazu, o tyle dostęp do danych poprzez interfejs SPI czy I²C wymaga wykonania kilku lub kilkunastu rozkazów.

Przykład podłączenia zasilania do układu zewnętrznej pamięci np. nastaw przedstawiony jest na **rys. 4**. Wykorzystano tu popularny układ zegara czasu rzeczywistego (RTC) PCF8583 zawierający w swojej strukturze 240 bajtów pamięci RAM do wykorzystania przez użytkownika. W układzie pokazanym na rys. 4 wykorzystano prosty przełącznik diodowy. Po zaniku głównego napięcia zasilającego +5 V, samoczynnie załącza on zasilanie PCF8583 z baterii litowej 3 V.

A co w takiej sytuacji z zasilaniem mikrokontrolera? Jeśli zewnętrzna pamięć RAM jest jedynie pamięcią nastaw, można odłączyć zasilanie awaryjne od mikrokontrolera. Jednak w przypadku, gdy zawiera również zmienne, należy wyposażyć mikrokontroler w wejście pomiarowe i wykonać pewien fragment programu w celu zapamiętania zmiennych, po zaniku głównego napięcia zasilającego. Gdy będzie on sygnalizowany wystarczająco szybko a w obwodzie zasilania mikrokontrolera umieścimy kondensator elektrolityczny o dużej pojemności, nie jest konieczne doprowadzanie awaryjnego napięcia zasilania. Mikrokontroler „zdąży" przesłać zmienne. Oczywiście nic nie stoi na przeszkodzie a nawet jest to rozwiązanie bezpieczniejsze, aby mikrokontroler był podłączony do zasilania awaryjnego wspólnie z układem pamięci. Może on np. po wykonaniu procedury awaryjnej przechodzić do trybu wyłączenia - POWER DOWN lub oszczędnego - IDLE w celu wydłużenia żywotności baterii.

Rozwiązania z zastosowaniem pamięci zewnętrznych SRAM można mnożyć w nieskończoność. Często do zasilania zamiast baterii stosuje się akumulator wyposażony w układ ładowania. Można używać pamięci równoległych lub szeregowych. Można stosować układy pamięci wykonane w technologiach Flash i EEPROM. Wydają się bardzo dobrą alternatywą w porównaniu z wszelkimi odmianami RAM. Nie wymagają bowiem użycia żadnych dodatkowych źródeł zasilania. Ponadto oferta handlowa związana zwłaszcza z pamięciami Flash jest bardzo szeroka. Spotkać można pamięci wykonane przy użyciu różnych technologii o pojemnościach od kilkuset bitów do 2 a nawet i więcej Mb. Dla przeciętnych zastosowań często wystarczającą jest pamięć EEPROM wbudowana w strukturę mikrokontrolera. Ich wadą w porównaniu z RAM jest długi czas konieczny na zapamiętanie danych.

Jacek Bogusz, AVT
jacek.bogusz@ep.com.pl

List. 4. Funkcje zapisu i odczytu pamięci EEPROM w języku AVR asembler wraz z przykładami ich użycia

```
.def   eedata = r23      ;bajt do zapisu w eeprom
.def   eeaddr = r24      ;adres zapisu bajtu

;----------------------------------------
;obsługa pamięci eeprom
;----------------------------------------
;zapis bajtu do EEPROM
;adres bajtu w eeaddr, bajt w eedata
ee_write: sbic eecr,EEWE           ;czekaj na EEWE = 0
   rjmp ee_write
   out eear,eeaddr                 ;podaj adres danych w eeprom
   out eedr,eedata                 ;zapisz dane
   sbi eecr,EEMWE                  ;ustaw bit „master ee write enable"
   nop
   sbi eecr,EEWE                   ;ustaw bit „ee write enable"
ret

;odczyt bajtu z EEPROM
;adres bajtu w eeaddr, bajt zwracany w eedata
ee_read:  sbic eecr,EEWE           ;czekaj na EEWE = 0
   rjmp ee_read
   out eear,eeaddr                 ;podaj adres w eeprom
   sbi eecr,EERE                   ;ustaw bit „ee read enable"
   in eedata,eedr                  ;czytaj dane
   ret

;zapis danych z bufora w pamięci RAM do EEPROM
;spodziewane: adres danych w eeaddr, koniec oznaczony jako EOD
ee_write_buf: ldi yl,LOW(buffer)
   clr yh
ee_write_loop:   ld eedata,y       ;sbuf = (Y)
   cpi eedata,EOD                  ;if (eedata != EOD) goto ee_write_next
   brne ee_write_next
   rcall ee_write                  ;zapisz EOD do eeprom
   ret
ee_write_next:   rcall ee_write    ;zapisz bajt do eeprom
   inc eeaddr                      ;eeadr = eeaddr+1
   inc yl                          ;Y = Y + 1
   rjmp ee_write_loop              ;goto ee_write_loop

;odczyt danych z EEPROM do bufora w pamięci RAM
;adres danych w eeaddr, koniec oznaczony jako EOD,
;adres miejsca,do którego przesyłane są dane w rej.Y
ee_read2buf: rcall ee_read
   cpi eedata,EOD                  ;if (eedata != EOD) goto ee_read_next
   brne ee_read_next
   st  y,eedata                    ;(Y) += EOD
   ret
ee_read_next: st  y,eedata
   inc yl                          ;Y += 1
   inc eeaddr                      ;eeadr += 1
   rjmp ee_read2buf
   ret
```