Mikrokontrolery z rdzeniem ARM, część 11

Obsługa układów peryferyjnych: PLL, VPB, MAM

Kontynuujemy opis konfiguracji wybranych peryferiów mikrokontrolerów LPC213x. W tej części zajmujemy się modułem MAM, detektorem zaniku napięcia zasilającego oraz trybami oszczędzania energii.

Konfiguracja kontrolera pamięci MAM

„Kluczem" do uzyskania dużej szybkości pracy mikrokontrolerów LPC jest kontroler pamięci Flash umożliwiający uruchamianie programu w pamięci Flash z maksymalną częstotliwością taktowania wewnętrznego (60 MHz). Kontroler MAM może pracować w następujących trybach (**rys. 27**):

- tryb 0 – kontroler MAM jest wyłączony i wszystkie instrukcje są pobierane z wewnętrznej pamięci Flash. W tym trybie program umieszczony w pamięci Flash będzie się wykonywał wolno, ponieważ przy dostępie do pamięci muszą być wstawiane dodatkowe cykle oczekiwania (*Wait States*). Po wyzerowaniu mikrokontroler rozpoczyna pracę w tym trybie.
- tryb 1 – wszystkie instrukcje sekwencyjne są pobierane z kontrolera MAM, natomiast instrukcje rozgałęzione oraz dane stałe pobierane są bezpośrednio z pamięci Flash. Użycie tego trybu może dawać pewne korzyści w stosunku do trybu pełnego, gdy pracujemy z krótkimi fragmentami kodu rozgałęzionego lub gdy chcemy aby dane zawarte w pamięci Flash były z niej odczytywane bezpośrednio.
- tryb 2 – wszystkie instrukcje pobierane są z wykorzystaniem kontrolera MAM. Wówczas każdy dostęp do pamięci Flash odbywa się z wykorzystaniem kontrolera MAM. W większości przypadków zapewnia to maksymalną wydajność programu umieszczonego w pamięci Flash, więc podczas normalnej pracy będziemy korzystać z tego trybu.

W większości przypadków pracować będziemy przy całkowicie włączonym kontrolerze MAM (Tryb 2), co zapewni maksymalną szybkość wykonania programu. Do konfiguracji kontrolera pamięci służą następujące rejestry SFR:

Tab. 12. Funkcje bitów w rejestrze konfiguracji MAM

Bit	Nazwa	Opis	Wartość początkowa
[1:0]	MAMMC	00 – Kontroler MAM wyłączony (Tryb 0) 01 – Kontroler MAM częściowo włączony (Tryb 1) 10 – Kontroler MAM włączony (Tryb 2) 11 – Zarezerwowane.	0
[7:2]	–	Zarezerwowane	–

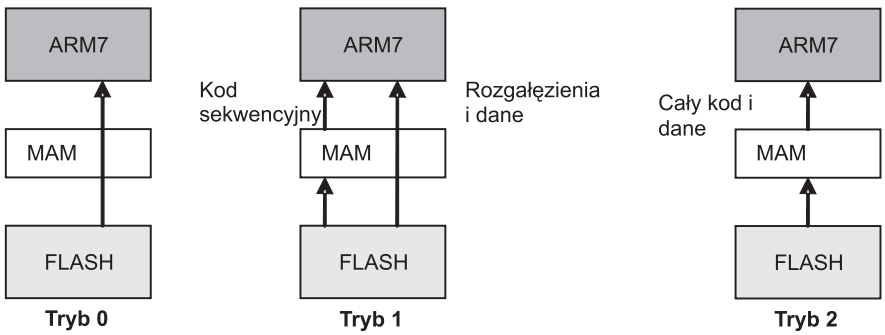
MAMCR – rejestr określający tryb pracy

MAMTIM – rejestr określający liczbę cykli zegarowych jaka jest użyta przy dostępie do pamięci Flash

Stan bitów rejestru MAMCR (0xE01FC000) określa tryb pracy kontrolera MAM w sposób pokazany w **tab. 12**.

Stan bitów rejestru MAMTIM (0xE01FC004) określa czas trwania dostępu do pamięci Flash w cyklach zegarowych procesora. W przypadku, gdy częstotliwość ta jest mniejsza niż 20 MHz, wystarczy, że czas ten będzie równy 1 cyklowi. W przypadku, gdy częstotliwość zawiera się w przedziale pomiędzy 20…40 MHz, najodpowiedniejszą wartością czasu będzie 2 cykle. Natomiast dla częstotliwości większych od 40 MHz musimy ustawić czas dostępu na 3 cykle (**tab. 13**).

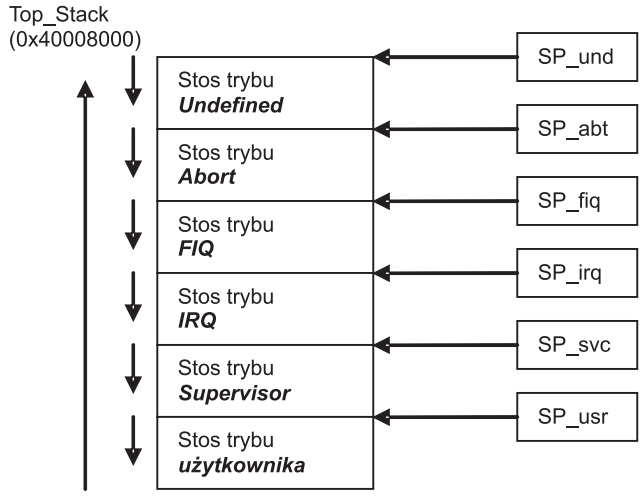
Aby zainicjalizować w pliku startowym *boot.s* kontroler pamięci MAM, musimy przypisać symbolowi *MAM_SETUP* wartość róż-



Rys. 27.

Tab. 13. Funkcje bitów w rejestrze MAMTIM

Bit	Nazwa	Opis	Wartość początkowa
[2:0]	MAMFCT	000 – Zarezerwowane 001 – 1 cykl częstotliwości procesora (CCLK) 010 – 2 cykl częstotliwości procesora (CCLK) 011 – 3 cykl częstotliwości procesora (CCLK) 100 – 4 cykl częstotliwości procesora (CCLK) 101 – 5 cykl częstotliwości procesora (CCLK) 110 – 6 cykl częstotliwości procesora (CCLK) 111 – 7 cykl częstotliwości procesora (CCLK)	0x07
[7:3]	–	Zarezerwowane	–



Rys. 28.

ną od 0. Spowoduje to dołączenie kodu odpowiedzialnego za inicjalizację. Należy również symbolowi *MAMCR_Val* przypisać odpowiednią wartość zgodnie z tab. 12 oraz symbolowi *MAMTIM_Val* przypisać ilość cykli dostępu do pamięci zgodnie z tab. 13. Poniżej przedstawiono definicję symboli dla najczęściej wykorzystywanego trybu pracy z częstotliwością w okolicy 60 MHz, z całkowicie włączonym kontrolerem MAM oraz zdefiniowanym czasem dostępu na 3 cykle zegarowe CCLK:

```
.equ    MAM_SETUP,      1
.equ    MAMCR_Val,      0x00000002
.equ    MAMTIM_Val,     0x00000003
```

Za inicjalizację kontrolera MAM odpowiada poniższy fragment kodu, który wpisuje do rejestrów kontrolera MAM odpowiednie wartości zgodnie z wartościami przypisanymi do symboli:

```
.if MAM_SETUP
LDR     R0, =MAM_BASE   ;Do R0 adres
bazowy MAM
MOV     R1, #MAMTIM_Val ;Do R1 zawar-
tość rejestru MAMTIM
STR     R1, [R0, #MAMTIM_OFS] ; Za-
pisz R1 do rejestru MAMTIM
MOV     R1, #MAMCR_Val    ;Do R1 za-
wartość rejestru MAMCR
STR     R1, [R0, #MAMCR_OFS]    ;Za-
pisz R1 do rejestru MAMCR
.endif
```

Działanie tego fragmentu programu sprowadza się do przepisania wartości symbolu *MAMTIM_Val* do rejestru SFR *MAMTIM* oraz symbolu *MAMCR_Val* do rejestru SFR *MAMCR*. Przepisanie zawartości odbywa się pośrednio poprzez wpisanie odpowiedniej liczby do rejestru R1, a następnie przepisaniu zawartości tego rejestru do odpowiedniego rejestru SFR. Jak pamiętamy rdzeń ARM nie może wykonywać żadnych operacji bezpośrednio na pamięci, którą także są rejestry SFR, dlatego zmiana zawartości SFR musi odbywać się z pomocą rejestrów ogólnego przeznaczenia.

Pozostałe elementy pliku startowego

Oprócz kodu związanego z konfiguracją urządzeń mających wpływ na działanie rdzenia mikrokontrolera, plik startowy ustawia rozmiar stosu dla każdego trybu ochrony procesora. Zapewnia inicjalizację pamięci zgodną ze standardem ANSI C/C++, czyli zeruje obszar pamięci danych niezainicjalizowanych oraz inicjalizuje obszar pamięci danych zainicjalizowanych. Wywołuje indywidualne konstruktory obiektów globalnych, czyli specjalne metody klas odpowiedzialne za inicjalizację obiektów, a na końcu wywołuje funkcję główną programu (*main*). Po zakończeniu działania programu kod zawarty w pliku startowym wywołuje destruktory obiektów globalnych, czyli specjalne metody wywoływane w celu zakończenia życia obiektu. Ostatnią czynnością jaką wykonuje kod startowy jest wejście w nieskończoną pętlę, ponieważ poza programem nie istnieje już żaden system nadrzędny. W przypadku, gdyby program działał pod kontrolą systemu operacyjnego, ostatnią czynnością byłoby oddanie kontroli do systemu. Za konfigurację stosów dla różnych trybów ochrony odpowiedzialne są następujące symbole w pliku startowym *boot.s*:

```
#Ustawienia stosu
.equ    Top_Stack,      0x40008000
;Adres określający wierzchołek stosu
.equ    UND_Stack_Size, 0x00000004
;Rozmiar stosu trybu Undefined
.equ    SVC_Stack_Size, 0x00000004
;Rozmiar stosu trybu Supervisor
.equ    ABT_Stack_Size, 0x00000004
;Rozmiar stosu trybu Abort
.equ    FIQ_Stack_Size, 0x00000004
;Rozmiar stosu trybu FIQ
.equ    IRQ_Stack_Size, 0x00000100
;Rozmiar stosu trybu IRQ
.equ    USR_Stack_Size, 0x00000200
;Rozmiar stosu trybu User
```

Zmieniając poszczególne wartości liczbowe w powyższych liniach, mamy możliwość kontrolowania indywidualnie poszczególnych rozmiarów stosów dla wybranego trybu ochrony. W przypadku, gdy dany tryb ochrony nie będzie wykorzystywany, należy ustawić najmniejszy dopuszczalny rozmiar stosu, czyli 4 bajty. Natomiast dla trybów ochrony, z których będziemy korzystać należy ustawić wielkość stosu od kilkudziesięciu do kilkuset bajtów w zależności od liczby zagłębień w programie oraz maksymalnego rozmiaru zmiennych automatycznych. Obszar stosu został umieszczony na końcu pamięci RAM mikrokontrolera, przy czym poszczególne wskaźniki stosów wskazują na wierzchołki stosów. Stos zorganizowany jest w kierunku adresów malejących. Odłożenie wybranego rejestru na stos powoduje zmniejszenie adresu wskaźnika wierzchołka, a następnie umieszczenie pod tym adresem wybranego rejestru. Na **rys. 28** przedstawiono organizację poszczególnych stosów w pamięci mikrokontrolera.

Aby ustawić wskaźnik stosu w wybranym trybie ochrony, należy wejść w ten tryb poprzez modyfikację rejestru znaczników CPSR, a następnie należy ustawić wskaźnik stosu SP (R13) tak, aby wskazywał na żądany adres.

Poniżej przedstawiono fragment kodu odpowiedzialnego za ustawienie wskaźnika stosu w trybie *Undefined*:

```
LDR R0, =Top_Stack      ;Do R0 wierz-
chołek stosu
#  Stos dla trybu Undefined
MSR CPSR_c, #Mode_UND|I_  Bit|F_Bit
;Wejdź w tryb undefined
MOV SP, R0          ;Ust wskaźnik sto-
su na wierzchołek
SUB R0, R0, #UND_Stack_Size
;Odejmij rozmiar stosu trybu UNDEF
```

Kod startowy przechodzi po kolei przez wszystkie tryby ochrony

Tab. 14. Funkcje bitów w rejestrze RSIR

Bit	Nazwa	Opis
[0]	POR	Ustawienie tego bitu oznacza, że mikrokontroler został wyzerowany w wyniku jego włączenia. Ustawienie tego bitu powoduje wyzerowanie pozostałych.
[1]	EXTR	Bit ten jest ustawiany w wyniku wystąpienia sygnału zewnętrznego zerowania
[2]	WDTR	Bit ten jest ustawiany w wyniku zadziałania układu Watchdog i jest kasowany w wyniku wystąpienia jakiejkolwiek innej przyczyny zerowania.
[3]	BODR	Bit ten jest ustawiany jeżeli zerowanie nastąpiło w wyniku zadziałania układu detektora napięcia. Bit ten nie jest kasowany w wyniku ustawienia bitu WDTR albo EXTR.
[7:4]	–	Zarezerwowane

i w każdym z nich oddzielnie ustawia wskaźnik stosu. Musimy pamiętać, że jako ostatni powinien być inicjalizowany wskaźnik stosu trybu użytkownika, ponieważ nie ma możliwości jawnego opuszczenia tego trybu. Po ustawieniu wskaźników stosów inicjalizowana jest pamięć RAM zgodnie z wymogami języka ANSI C/C++. Najpierw zerowana jest sekcja *.bss* zawierająca niezainicjalizowane zmienne globalne, które zgodnie z definicją powinny przyjąć wartość 0. Czynność tę realizuje poniższy fragment kodu:

```
# Wyczyść sekcje .bss (Inicjaliza-
cja 0)
MOV     R0, #0       ;Do R0 wpisz 0
LDR     R1, =__bss_start__   ;Do R1
adr. pocz. danych
LDR     R2, =__bss_end__     ;Do R2
adr. końca danych
LoopZI:         CMP     R1, R2
;Porównaj R2 z R3
STRLO   R0, [R1], #4      ;Do ko-
mórki pam. z R1 wpisz zawartość R0
zmniejsz index tylko wtedy gdy nie
koniec segmentu
BLO     LoopZI       ;Jeśli w obsza-
rze .bss to skocz
```

Następnie kopiowany jest do sekcji .data w obszarze pamięci RAM fragment pamięci Flash przechowujący wartości początkowe zmiennych zainicjalizowanych. Procedura kopiująca jest podobna do poprzedniej z tą różnicą, że dane odczytywane z pamięci Flash wpisywane są do rejestru R0, skąd następnie wędrują do sekcji .data zawartej w pamięci RAM. Pętla ta zapewnia odpowiednie ustawienie zmiennych zainicjalizowanych zgodnie z wartościami przypisanymi tym zmiennym w kodzie programu.

Po zainicjalizowaniu obszaru danych w pamięci RAM następuje uruchomienie po kolei wszystkich metod konstruktorów obiektów globalnych, co zapewnia odpowiednie zainicjalizowanie wszystkich obiektów klas. Za wywołanie konstruktorów odpowiedzialny jest poniższy kod:

```
  LDR R0, =__ctors_start__   ;do R0
początek tablicy adresów konstrukto-
rów ob. glob
  LDR R1, =__ctors_end__     ;do R1
koniec tablicy adresów konstruktorów
ob. glob
ctor_loop:
  CMP R0, R1      ;Porównaj czy to
ostatni adres
BEQ ctor_end      ;Jeżeli tak to
opuść pętle
  LDR R2, [R0], #+4     ;Do R2 wpisz
adres konstruktora
  STMFD SP!, {R0-R1}    ;Zapisz R0,R1
na stosie
MOV LR, PC      ;Do Link Reg (R14)
przepisz licznik rozk. (R15)
  MOV PC, R2      ;Do PC (R15) wpisz
adres konstruktora
  LDMFD SP!, {R0-R1}    ;Zdejmij ze
stosu R0,R1
B ctor_loop       ;Skocz do początku
pętli
ctor_end:
```

Chcielibyśmy zwrócić szczególną uwagę Czytelników na sposób, w jaki odbywa się uruchomienie konstruktorów. Widać tutaj wielką zaletę równouprawnienia wszystkich rejestrów rdzenia. Uruchomienie danej procedury sprowadza się do wprowadzenia do licznika rozkazów PC (R15) adresu podprogramu, co jest nie do pomyślenia w większości mikrokontrolerów o tradycyjnej architekturze. Po zainicjalizowaniu wszystkich obiektów poprzez wywołanie ich konstruktorów następuje uruchomienie funkcji głównej *main*:

```
#Wywołaj funkcję main
                LDR R2, =main
                MOV LR, PC
                MOV PC, R2
```

Jeżeli funkcja *main* zakończy działanie w pisanych przez nas programach, co w zasadzie nigdy nie będzie miało miejsca, wykonywany jest kod odpowiedzialny za wywołanie destruktorów. Destruktory klas są odpowiedzialne za wykonanie czynności mających na celu zakończenie istnienia obiektu. Na przykład zwolnienie pamięci przydzielonej dynamicznie. Kod odpowiedzialny za wywołanie destruktorów jest w zasadzie identyczny jak w przypadku wywołania konstruktorów. Różnica polega jedynie na podstawieniu adresów procedur destruktorów *__dtors_start__*. Po opuszczeniu kodu odpowiedzialnego za wywołanie destruktorów nie ma już więcej żadnych czynności do wykonania dlatego ostatnią czynnością jest wejście mikrokontrolera w nieskończoną pętlę.

Tab. 15. Funkcje bitów w rejestrze PCON

Bit	Nazwa	Opis	Wartość początkowa
[0]	IDL	Ustawienie tego bitu powoduje przejście mikrokontrolera w tryb Idle czyli sygnał zegarowy dla jednostki centralnej zostaje wyłączony, natomiast pozostałe układy peryferyjne pracują. Jeżeli wystąpi jakieś przerwanie od układu peryferyjnego, jednostka centralna jest uruchamiana i kontynuuje wykonanie przerwanego programu	0
[1]	PD	Ustawienie tego bitu powoduje przejście mikrokontrolera w tryb Power Down, powoduje wyłączenie generatora sygnału zegarowego taktującego mikrokontroler. Jednym sposobem na wyjście z tego trybu jest zerowanie mikrokontrolera lub obudzenie na skutek przerwania zewnętrznego lub alarmu od zegara RTC. Należy pamiętać, że gdy mikrokontroler wyjdzie z tego trybu musimy ponownie skonfigurować pętlę PLL.	0
[2]	PDBOT	Ustawienie tego bitu powoduje, że detektor zaniku napięcia zasilającego, gdy mikrokontroler wejdzie w tryb Power Down jest wyłączany, zapobiegając w ten sposób wyjściu z tego trybu w wyniku obniżenia napięcia zasilającego. Gdy ten bit jest ustawiony detektor zaniku napięcia pracuje również w trybie Power Down.	0
[7:3]	–	Zarezerwowane	

Detektor zaniku napięcia zasilającego oraz identyfikacja źródła zerowania

Zbyt niskie napięcie zasilające mikrokontroler może spowodować nieprawidłową pracę programu objawiającą się nieokreślonym zachowaniem, na przykład skokiem w nieokreślone miejsce pamięci, co w przypadku systemów mikroprocesorowych sterującymi ważnymi procesami może mieć opłakane skutki. Dlatego w większości współczesnych mikrokontrolerów wprowadzono detektor zaniku napięcia zasilającego, który w przypadku zbyt niskiego napięcia zasilającego utrzymuje mikrokontroler w stanie zerowania. W mikrokontrolerach LPC213x detektor posiada dwa progi zadziałania. W przypadku, gdy napięcie zasilające spadnie poniżej 2,9 V, ustawiana jest flaga BOD (bit 20) w rejestrze *VICRawIntr* (`0xFFFFF008`), która może zgłosić przerwanie informujące o zaniku napięcia zasilającego. Przerwanie to można wykorzystać na przykład do zapamiętania w pamięci EEPROM danych konfiguracyjnych. Natomiast, gdy napięcie spadnie poniżej 2,6 V, wówczas mikrokontroler jest utrzymywany w stanie zerowania, co zapobiega błędnemu działaniu mikrokontrolera. Z punktu widzenia programisty istotne może być określenie przyczyny z jakiej mikrokontroler został wyzerowany i w zależności od sytuacji podjęcie odpowiednich czynności. Na przykład jeżeli system został wyzerowany z powodu zadziałania układu Watchdog, możemy poinformować użytkownika o wystąpieniu sytuacji krytycznej. Przyczynę zerowania mikrokontrolera możemy określić poprzez zbadanie stanu bi-

Tab. 16. Funkcje bitów w rejestrze PCONP

Bit	Nazwa	Opis	Wartość początkowa
[0]	–	Zarezerwowane	–
[1]	PCTIM1	Załączenie TIMER0	1
[2]	PCTIM2	Załączenie TIMER1	1
[3]	PCUART0	Załączenie układu UART0	1
[4]	PCUART1	Załączenie układu UART1	1
[5]	PCPWM0	Załączenie układu PWM0	1
[6]	–	Zarezerwowane	–
[7]	PCI2C0	Załączenie układu I2C0	1
[8]	PCSPI0	Załączenie układu SPI0	1
[9]	PCRTC	Załączenie układu RTC	1
[10]	PCSPI1	Załączenie układu SPI1	1
[11]	–	Zarezerwowane	–
[12]	PCAD0	Załączenie układu A/D0. Bit PDN w rejestrze AD0CR musi być wyzerowany przed wyzerowaniem tego bitu. Również ten bit musi być ustawiony przed ustawieniem bitu PDN.	1
[18:13]	–	Zarezerwowane	–
[19]	PCI2CI	Załączenie układu I2C1	1
[20]	PCAD1	Załączenie układu A/D1. Bit PDN w rejestrze AD1CR musi być wyzerowany przed wyzerowaniem tego bitu. Również ten bit musi być ustawiony przed ustawieniem bitu PDN.	1
[31:21]	–	Zarezerwowane	–

tów rejestru RSIR (0xE01FC180), których znaczenie jak w **tab. 14**.

Wszystkie ustawione bity w rejestrze RSIR możemy skasować programowo poprzez wpisanie jedynki na wybranym bicie. Aby sprawdzić w praktyce działanie rejestru RSIR oraz działanie detektora zaniku napięcia napiszemy bardzo prosty program (ep4.zip) badający cyklicznie stan bitu BOD w rejestrze SFR *VicRawIntr* oraz sprawdzający przyczynę zerowania mikrokontrolera, którego listing znajduje się poniżej:

```
#include "lpc213x.h"
//Definicja LED-ow
#define LEDS (0xFF<<16)
#define LEDDIR IO1DIR
#define LEDPIN IO1PIN

//Funkcja glowna main
int main(void)
{
  LEDDIR = LEDS;        //ustaw linie
P1.16..P1.24 jako wyjściowe
  unsigned char r=0;
  while(1)
  {
    if(RSID & 0x01) RSID |= 0x0F;
    r = (RSID & 0x0f)<<4;
    if(VICRawIntr & (1<<20) ) r |=
1;
    else r |= 2;
    LEDPIN = ((unsigned int)r)<<16;
  }
}
```

Działanie tego programu jest bardzo proste, mianowicie na początku inicjalizowane są piny wyjściowe sterujące diodami LED. Następnie określana jest przyczyna zerowania i jeżeli bit POR (0) w rejestrze RSIR jest ustawiony, wszystkie bity w tym rejestrze są kasowane. Następnie do zmiennej r wpisywany jest stan rejestru RSIR z zamaskowanymi nieistotnymi bitami i przesunięty o 4 bity w lewo, tak aby diody D4... D7 odzwierciedlały stan rejestru RSIR. W następnej linii sprawdzany jest stan bitu BOD i jeżeli jest on skasowany (co oznacza prawidłowe napięcie zasilające), ustawiany jest w zmiennej r bit odpowiadający diodzie LED1 w zestawie ZL6ARM. Natomiast, gdy napięcie zasilające jest za niskie i zawiera się w przedziale 2,6...2,9 V, ustawiany jest bit odpowiadający diodzie LED0. Na końcu pętli zawartość zmiennej r jest wysyłana do portu P1, co powoduje ustawienie odpowiednich diod LED. Aby skompilować ten przykład należy po uruchomieniu *Eclipse* wybierać z menu polecenie *File–>New–>Project*. Pokaże nam się okno dialogowe, z którego wybieramy opcję: *Standard Make C++ Project*. Następnie klikamy przycisk *Next>*, wówczas pojawi się kolejne okno dialogowe. W polu tekstowym *Project Name* wpisujemy nazwę projektu, np. *Bod* i wciskamy klawisz *Finish*, co spowoduje utworzenie pustego projektu. Kolejną czynnością jest import plików do projektu. Możemy tego dokonać klikając prawym przyciskiem myszy na otwartym projekcie, a następnie wybrać polecenie *Import*. Pojawi się wówczas okno dialogowe, z którego wybieramy opcję *Archive File*, zobaczymy wtedy kolejne okno, w którym klikamy przycisk *Browse* i wybieramy plik *ep4.zip* z przykładowym projektem. Po tej czynności przystępujemy do kompilacji projektu, co możemy zrobić wydając z menu polecenie *Project–>Build All*. Jeżeli wszystko przebiegło poprawnie projekt zostanie skompilowany, w wyniku czego powstanie plik wynikowy *bodtst.hex*. Aby sprawdzić działanie poniższego programu musimy podłączyć zestaw ZL6ARM do zasilacza regulowanego i ustawić napięcia wyjściowe, np. na poziomie 10 V, powinna wówczas zaświecić się dioda LED1. Następnie stopniowo zmniejszamy napięcie zasilające do momentu aż zapali się dioda LED0 i zgaśnie LED1, co oznacza ustawienie flagi przerwania od detektora BOD. Możemy również przetestować działanie stanu rejestru RSIR obserwując stan diod D4...D0. Gdy włączymy napięcie zasilające do zestawu ZL6ARM, nie powinna się palić żadna z diod D4...D7, następnie jeżeli wciśniemy np. przycisk RESET powinna zapalić się dioda D5 odzwierciedlająca stan bitu zerowania sygnałem zewnętrznym. Jeżeli teraz, np. będziemy zmniejszać stopniowo napięcie aż do momentu, gdy układ BOD wystawi sygnał RESET, a następnie zwiększymy napięcie zasilające, powinna zaświecić się dioda D7. Oznacza to, że przyczyną zerowania był układ detektora zaniku napięcia.

Tryby oszczędzania energii

Mikrokontrolery LPC213x/214x posiadają tryby oszczędzania energii bardzo podobne jak w 8–bitowym mikrokontrolerze 8051, mianowicie tryb *Idle* oraz Tryb *Power Down*. W trybie *Idle* rdzeń mikrokontrolera zostaje zatrzymany, natomiast zegar oraz urządzenia peryferyjne nadal pracują. Nadejście najbliższego przerwania powoduje uruchomienie CPU i kontynuację wykonania programu. W trybie *Power Down* zatrzymana jest jednostka centralna, wszystkie urządzenia peryferyjne oraz zegar mikrokontrolera. Jedyną możliwością wyjścia z tego trybu jest zerowanie mikrokontrolera, zgłoszenie przerwania zewnętrznego lub przerwanie od alarmu zegara RTC. Ponadto wprowadzono dodatkowy rejestr konfiguracyjny umożliwiający wyłączenie nieużywanych układów peryferyjnych. Do uruchamiania trybów oszczędności energii służy rejestr PCON (0xE01FC0C0), natomiast za wyłączanie poszcze-

gólnych układów peryferyjnych odpowiada rejestr PCONP (0xE01FC0C4). Rejestr PCON zawiera dwa bity (**tab. 15**), których ustawienie powoduje wejście w wybrany tryb obniżonego poboru mocy. Jeżeli oba bity są ustawione wówczas mikrokontroler wchodzi w tryb uśpienia (*Power Down*).

Rejestr PCONP zawiera zestaw bitów umożliwiających wyłączenie nieużywanych układów peryferyjnych. Niektórych układów nie można wyłączyć, np. Watchdoga, portów wejścia wyjścia czy układów związanych z generacją sygnału zegarowego. Prawidłowy zapis i odczyt rejestrów danego układu peryferyjnego jest możliwy tylko wtedy, gdy ten układ peryferyjny jest włączony. Jeżeli wybrany bit w rejestrze PCONP jest **ustawiony** oznacza to, że odpowiadający mu układ peryferyjny jest **włączony**. Natomiast jeżeli wybrany bit jest **wyzerowany**, wówczas odpowiadający mu układ peryferyjny jest **wyłączony**.

Zestawienie bitów rejestru PCONP znajduje się w **tab. 16**.

Aby przejść w tryb IDLE, wystarczy, że w programie ustawimy bit IDL w rejestrze PCON, co można zrobić na przykład poprzez wywołanie następującej instrukcji:

```
PCON |= PCON_IDL;
```

Będzie to ostatnia instrukcja wykonana przez jednostkę centralną. Dodatkowe zmniejszenie poboru prądu można uzyskać poprzez wyłączenie nieużywanych układów peryferyjnych. Powiedzmy, że w programie nie będziemy nigdy używać Timera0 i interfejsów SPI, wówczas wystarczy wykonać następującą instrukcję:

```
PCONP &= ~( PCONP_PCSPIO| PCONP_
PCSPI1| PCONP_PCTIM0);
```

Zaowocuje to wyłączeniem wspomnianych wcześniej układów peryferyjnych. Jeszcze jedną możliwością zmniejszenia poboru prądu pobieranego przez mikrokontroler jest zmniejszenie częstotliwości taktującej urządzenia peryferyjne, poprzez ustawienie odpowiednio dużego podzielnika sygnału zegarowego *VPBDIV*.

Na tym kończymy opis układów peryferyjnych związanych z *System Control Block*. Zaprezentowane tutaj wiadomości nie wyczerpały tematu, miały one nauczyć czytelnika jak skonfigurować plik startowy do własnych wymagań oraz zapoznać go ze wstępnymi informacjami na temat trybów oszczędzania energii. Przykłady dotyczące tych trybów będą szczegółowo rozważanie podczas omawiania przerwań zewnętrznych.

Lucjan Bryndza SQ7FGB
lucjan.bryndza@ep.com.pl