Mikrokontrolery STR91x od podstaw, część 4

Obniżony pobór mocy

Pisząc oprogramowanie dla urządzenia zasilanego z sieci najczęściej nie musimy martwić się o pobór energii, więc rdzeń mikrokontrolera oraz wszystkie urządzenia peryferyjne ustawiamy na największą możliwą częstotliwość pracy. Ponieważ mikrokontroler STR91x posiada bardzo wydajny rdzeń ARM9 oraz wiele zaawansowanych układów peryferyjnych pobór prądu może być znaczący (nawet setki Ma). W przypadku urządzeń zasilanych za pomocą baterii, nie możemy sobie pozwolić na pracę mikrokontrolera z maksymalną wydajnością, ponieważ bateryjne źródło energii ma ograniczoną pojemność, a dla użytkownika końcowego bardzo ważny jest jak najdłuższy czas działania urządzenia z jednego kompletu baterii.

Sposoby oszczędzania energii w STR91x

Mikrokontrolery STR91x mają zaawansowane mechanizmy ograniczenia poboru energii, pozwalające znacząco wydłużyć czas pracy mikrokontrolera przy zasilaniu bateryjnym. Najprostszym sposobem zmniejszenia poboru energii, jest wyłączenie nieużywanych urządzeń peryferyjnych. Włączanie oraz wyłączanie poszczególnych układów peryferyjnych mikrokontrolera umożliwiają rejestry PRR0...PRR1, – sterujące zerowaniem wybranego układu peryferyjnego, oraz rejestry PCGR0… PCGR1 sterujące dołączaniem sygnału zegarowego do wybranych układów peryferyjnych. Ponieważ układ STR91x został wykonany w technologii CMOS, odłączenie sygnału zegarowego oraz uaktywnienie sygnału zerującego skutkuje tym, że odłączony układ peryferyjny praktycznie nie pobiera prądu.

Konstruktorzy mikrokontrolerów STR91x zastosowali taktykę domyślnego wyłączania wszelkich peryferii (poza systemowymi). Przed użyciem układu peryferyjnego należy go włączyć poprzez odblokowanie sygnału zegarowego oraz sygnału zerującego. W przypadku, gdy dane urządzenie peryferyjne używane jest sporadycznie warto zastanowić się nad włączaniem wybranego urządzenia peryferyjnego tylko na moment jego użycia, co wbrew pozorom pozwala zaoszczędzić dość spore ilości energii.

Inną możliwością ograniczenia poboru prądu przez układy peryferyjne jest ograniczenie częstotliwości taktowania tych urządzeń, ponieważ stosunkowo rzadko istnieje konieczność taktowania ich z maksymalną częstotliwością (48 MHz). Mikrokontroler STR91x jest wyposażony w podzielniki AHBDIV oraz APBDIV, pozwalające podzielić częstotliwość F_{rclk} wychodzącą wyjścia z dzielnika RCLKDIV, osobno dla układów peryferyjnych dołączonych do magistrali AHB oraz APB. Duża część układów peryferyjnych posiada również indywidualne bloki preskalerów pozwalające na niezależne ustalenie częstotliwości taktowania tych układów.

Dzięki wspomnianym mechanizmom możemy znacznie ograniczyć pobór prądu pobieranego przez poszczególne układy peryferyjne mikrokontrolera. Jednak sama jednostka centralna CPU pracując z maksymalną częstotliwością taktowania równą 96 MHz, pobiera również znaczne ilości energii. Ponieważ duża moc obliczeniowa rdzenia ARM966E–S, jest zazwyczaj potrzebna tylko w dość krótkich odcinkach czasu, mikrokontroler STR91x, podobnie jak większość innych mikrokontrolerów posiada mechanizmy pozwalające znacząco ograniczyć pobór prądu przez CPU. Mikrokontroler STR91x może znajdować się w następujących trybach oszczędzania energii: *SPECIAL INTERRUPT MODE, IDLE MODE, POWER DOWN MODE*.

- Tryb *SPECIAL INTERRUPT MODE* jest trybem pracy pozwalającym zmniejszyć częstotliwość taktowania rdzenia mikrokontrolera podczas wykonywania głównego programu, oraz osiągać maksymalną szybkość podczas obsługi przerwań. Istota tego trybu polega na tym, że podczas normalnej pracy jednostka centralna CPU taktowana jest za pomocą sygnału zegarowego F_{rclk}, z wyjścia preskalera RCLK, natomiast w momencie wejścia jednostki centralnej w tryb obsługi przerwania podzielnik RCLK jest pomijany i jednostka centralna taktowana jest za pomocą głównego sygnału zegarowego F_{mstr}. Po zakończeniu procedury obsługi przerwania następuje ponowne przełączenie rdzenia mikrokontrolera na taktowanie za pomocą sygnału zegarowego F_{mstr}.
- Tryb *IDLE MODE* występuje praktycznie we wszystkich mikrokontrolerach. W momencie wejścia w ten tryb sygnał zegarowy taktujący jednostkę centralną CPU jest odłączany do czasu nadejścia najbliższego przerwania. Dzięki temu mechanizmowi mamy możliwość wyłączenia jednostki centralnej w momencie gdy nie jest ona potrzebna.
- Tryb *POWER DOWN MODE* polega na wyłączeniu wszystkich sygnałów zegarowych, oraz układów peryferyjnych, za wyjątkiem zegara RTC. Wyjście z tego trybu pracy jest możliwe tylko poprzez wyzerowanie mikrokontrolera, przerwanie zewnętrzne wyzwalane poziomem, lub przerwanie od zegara RTC. Jest to najgłębszy tryb uśpienia, podczas którego

List. 3.
```
#define VIC1_WIU0 (1<<10)
#define VIC1_WIU0_BIT 10
#define VIC_SLOT_EN (1<<5)
#define WIU_INT_EN 0x02
#define WIU_WKUP_INT 0x01
#define WKUPSEL_EXTINT7 0x07

//Setup external interrupt
static void setup_extint(void)
{
  /*** Ustawienie WIU (WakeUP Interrupt Unit) ***/
  //Wylacz reset
  SCU->PRR1 |= __WIU;
  //Wlacz zegar
  SCU->PCGR1 |= __WIU;            [1]
  //Ustawienie rejestru maski tylko P3.4 ... P3.7 zglaszaja przerania
  WIU->MR |= 0x80;                [2]
  //Typ przerwania (zbocze opadajace) P3.4 ... P3.7
  WIU->TR &= ~0x80;               [3]
  //Odblokuj przerwanie od WIU i wybudzenie procesora
  WIU->CTRL = WIU_INT_EN | WIU_WKUP_INT;          [4]
  //Ustaw VIC1.10 jako linie P3.7
  SCU->WKUPSEL = WKUPSEL_EXTINT7;         [5]

  /*** Ustawienie kontrolera przerwan VIC ***/
  //Zalaczenie clocka dla VIC
  SCU->PCGR0 |= __VIC;            [6]
  //Reset
  SCU->PRR0 &= ~__VIC;
  SCU->PRR0 |= __VIC;
  //Do wektora podstaw adres procedury obslugi
  VIC1->VAiR[0] = (unsigned long)irqIntHandler;
  //Ustaw slot na przerwanie od WIUOC
  VIC1->VCiR[0] = VIC1_WIU0_BIT | VIC_SLOT_EN;
  //Zakwalifikuj przerwanie WIU jako IRQ
  VIC1->INTSR &= ~VIC1_WIU0;
  //Vic interrupt enable register
  VIC1->INTER = VIC1_WIU0;
}
```

List. 4.
```
//Przerwanie niemaskowalne fiq
static void irqIntHandler(void) __attribute__
((interrupt(„IRQ”)));

void irqIntHandler(void)
{
  //Licznik przerwan
  static uint8_t irqCnt;
  //Przepisz kto zglosil do ledow
  GPIO7->DR[0xff<<2] = ++irqCnt;
  //Kasuj zrodlo przerwan
  WIU->PR = 0x80;
  //Informacja dla VIC (koniec przerwania)
  VIC1->VAR = 0;
}
```

List. 5.
```
#define SCU_PWRMNG_DOWN_MODE 0x02
#define SCU_PWRMNG_IDLE_MODE 0x01

//Funkcja zmienia tryb na power down
static inline void sleep_down(void)
{
    uint32_t tmp = SCU->PWRMNG;
    tmp &= ~0x7;
    tmp |= SCU_PWRMNG_DOWN_MODE;
    SCU->PWRMNG = tmp;
}

//Funkcja zmienia tryb na power down
static inline void sleep_idle(void)
{
    uint32_t tmp = SCU->PWRMNG;
    tmp &= ~0x7;
    tmp |= SCU_PWRMNG_IDLE_MODE;
      SCU->PWRMNG = tmp;
}
```

pobór prądu wynosi zaledwie kilkanaście μA.

Praca z obniżonym poborem mocy – przykład praktyczny

Po zapoznaniu się z podstawowymi zagadnieniami związanymi z oszczędzaniem energii, pokażemy teraz na przykładzie (*ex24_wakeup*), jak zmusić mikrokontroler STR91x do tego aby pobierał mniej prądu. Działanie niniejszego programu jest bardzo proste, mianowicie w momencie wciśnięcia klawisza K4, wartość licznika programowego zwiększana jest o 1, a stan tego licznika wyświetlany jest na diodach D0...D7. Ponieważ mikrokontroler nie ma tutaj specjalnie żadnych zadań do wykonania, po zwiększeniu zmiennej oraz wpisaniu danych do portu GPIO, w zależności od konfiguracji będziemy wyłączać samą jednostkę centralną CPU (tryb *IDLE*) lub usypiać mikrokontroler (tryb *POWER DOWN*). Wybudzanie mikrokontrolera odbywać się będzie za pomocą przerwania generowanego przez układ zgłaszania przerwań zewnętrznych i wybudzania (WIU), który był omówiony w jednym z poprzednich rozdziałów.

Jeżeli na początku programu (**list. 3**) zdefiniowano stałą #define PWR_MODE_IDLE mikrokontroler w chwilach bezczynności przełączany będzie w trybie *IDLE*, natomiast w przypadku braku tej definicji w tryb całkowitego uśpienia *POWER DOWN*. Funkcja *setup_extint()*, służy do konfiguracji bloku WIU, tak aby wciśnięcie klawisza K4, wygenerowało przerwanie oraz sygnał wybudzający mikrokontroler.

W [1] włączany jest kontroler przerwań zewnętrznych WIU, poprzez odblokowanie sygnału zegarowego oraz zerującego. W [2] włączane jest zgłoszenie przerwania od linii P3.7 przez ustawienie bitu 7 w rejestrze maski przerwania WIU–>MR. W [3] przerwanie od linii P3.7 konfigurowane tak, aby warunkiem jego zgłoszenia było wystąpienie zbocza opadającego. W trybie uśpienia (*Power Down*), sygnał taktujący blok WIU, jest wyłączony, nie ma więc możliwości badania zboczy sygnałów, dlatego ustawienie zbocza opadającego oznacza wybudzenie mikrokontrolera w momencie wystąpienia stanu niskiego na linii P3.7. W [4] za pomocą rejestru kontrolnego *WIU–>CTRL* włączamy globalne zezwolenie na zgłaszanie przerwań przez blok WIU, poprzez ustawienie bitu *INT_EN* (bit 1), oraz zezwolenie na wybudzanie mikrokontrolera ze stanu uśpienia poprzez ustawienie bitu *WKUP_INT* (bit 0). W [5], poprzez konfigurację multipleksera bloku SCU, wybieramy linię, która będzie generowała przerwanie, oraz wybudzała mikrokontroler ze stanu uśpienia. Na zakończenie w [6], konfigurowany jest kontroler VIC, tak aby przerwanie z kanału #10 kontrolera VIC1, przyporządkowane do grupy linii portu P3 powodowało wywołanie procedury obsługi przerwania *irqIntHandler()* – **list. 4**.

Procedura obsługi tego przerwania jest bardzo prosta, i sprowadza się jedynie do zwiększenia zmiennej *irqCnt* oraz przepisania stanu bitów do rejestru *DR* portu P7, w wyniku czego na diodach LED D1...D7 zestawu wyświetlana jest binarna reprezentacja tej zmiennej. Na zakończenie procedury obsługi przerwania, kasowana jest flaga zgłoszenia przerwania w układzie WIU, a do kontrolera VIC jest wysyłana informacja o zakończeniu obsługi przerwania.

List. 6.

```
//Funkcja glowna main
int main(void)
{
  //Poczatkowe opoznienie (JTAG secure)
  delay(1000000);

  //Ustawienie portow gpio
  setup_pll();

  //Ustawienie petli PLL (czestotliwosc 96MHz)
  setup_gpio();

  //Ustawienie przerwan zewnetrznych WIU /WIC
  setup_extint();

  //Enable IRQ interrupt in ARM core
  enable_irq();

#ifndef PWR_MODE_IDLE
  while(1)
  {
    //Uspij procesor
    sleep_down();
    //Ponowna inicjalizacja PLL
    setup_pll();
  }
#else
    while(1)
    {
      //Przejdz w tryb jalowy idle
      sleep_idle();
    }
#endif
  //Koniec main teraz przerwanie zmieni stan diody
  return 0;
}
```

Za wprowadzenie mikrokontrolera w tryb *IDLE* oraz *POWER DOWN* odpowiedzialne są funkcje *sleep_idle()*, oraz *sleep_down()* – **list. 5**.

Działanie tych funkcji sprowadza się do odczytania bieżącej zawartości rejestru *SCU–>PWRMNG*, ustawienia odpowiednich bitów *PWR_MODE*, a następnie ponowny zapis tego rejestru. W momencie wpisania nowej wartości do rejestru *SCU–>PWRMNG* mikrokontroler przechodzi w tryb *IDLE* lub *POWER DOWN*. Część główną programu przedstawiono na **list. 6**.

Na początku inicjalizowana jest pętla PLL mikrokontrolera (*setup_pll()*), konfigurowane są linie portów GPIO (*setup_gpio()*), inicjalizowana jest obsługa przerwań zewnętrznych i wybudzania (*setup_extint()*), oraz włączane są przerwania w jednostce centralnej (*enable_irq()*). Następnie program wchodzi do pętli głównej, w której cyklicznie w zależności od konfiguracji wywoływana jest funkcja przełączająca mikrokontroler w tryb *IDLE* lub *POWER DOWN*. Dodatkowo w pętli trybu *POWER DOWN*, po wywołaniu funkcji usypiającej ponownie wywoływana jest funkcja inicjalizująca pętlę PLL, ponieważ po przejściu mikrokontrolera w tryb uśpienia ulega ona wyłączeniu. Efektem działania poniższego programu jest to, że mikrokontroler praktycznie cały czas znajduje się trybie całkowitego uśpienia, lub trybie jałowym, a wybudzany jest tylko w momencie wciśnięcia klawisza K4. Dzięki temu znacząco został zredukowany prąd pobierany przez mikrokontroler. Należy tutaj zaznaczyć, że w trybie całkowitego uśpienia pobór prądu przez diody LED jest wielokrotnie większy od prądu pobieranego przez sam układ STR91x.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl