grafika pochodzi z http://7art-screensavers.com

Opóźnienia w STM32 (2)

Precyzyjne odmierzanie czasu w połączeniu z trybami oszczędzania energii

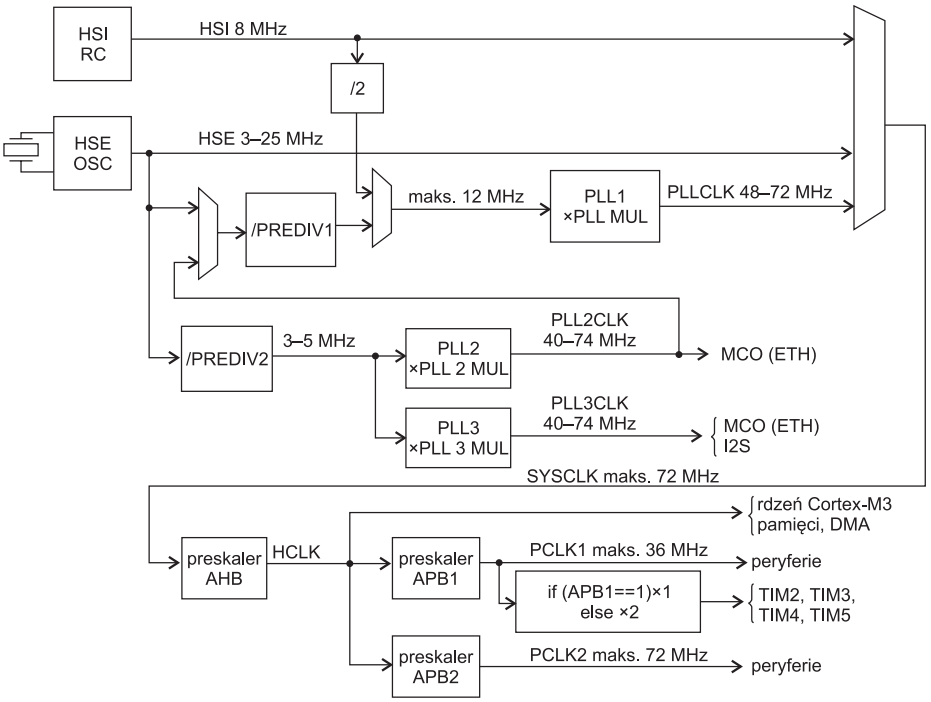
W pierwszej części artykułu opisano funkcje realizujące precyzyjne odmierzanie opóźnień w połączeniu z trybami oszczędzania energii dla mikrokontrolerów STM32 z rdzeniem Cortex-M3. W części drugiej omówiono konfigurowanie sygnałów zegarowych, przykładowy program testowy i układ, na którym wykonano testy. Zamieszczono też wyniki pomiarów prądu zasilania w poszczególnych trybach obniżonego poboru energii oraz rzeczywistych czasów opóźnień.

Dystrybucja sygnałów zegarowych

Aby wykorzystać opisane w pierwszej części artykułu funkcje opóźniające, musimy skonfigurować oscylator kwarcowy mikrokontrolera oraz określić sygnały taktujące rdzeń i peryferie. Dystrybucja sygnałów zegarowych w STM32F107 jest w uproszczeniu przedstawiona na **rysunku 1**. Rdzeń jest taktowany sygnałem HCLK, którego maksymalna częstotliwość wynosi 72 MHz. Peryferie są taktowane sygnałami PCLK1 i PCLK2 otrzymywanymi z HCLK za pomocą dzielników wstępnych (*prescalers*) APB1 i APB2. Sygnały PCLK1 i PCLK2 mogą mieć maksymalne częstotliwości odpowiednio 36 i 72 MHz, dlatego ustawiamy wartości dzielników APB1 i APB2 odpowiednio na 2 i 1. Częstotliwość taktowania liczników TIM2, TIM3, TIM4 i TIM5 zależy od współczynnika podziału APB1. Jeśli ma on wartość 1, to liczniki te są taktowane sygnałem PCLK1 o tej samej częstotliwości co HCLK. Jeśli natomiast dzielnik APB1 ma wartość różną od 1, to PCLK1 ma częstotliwość APB1 razy mniejszą niż HCLK, ale omawiane liczniki są taktowane sygnałem o częstotliwości dwa razy większej niż PCLK1. W naszym przy-

Listing 4. Plik nagłówkowy clock.h

```
#ifndef _CLOCK_H
#define _CLOCK_H 1

void AllPinsDisable(void);
int ClockConfigure(unsigned);

#endif
```



Rysunek 1. Uproszczony schemat dystrybucji sygnałów taktujących w STM32F107

Listing 5. Implementacja konfigurowania sygnałów taktujących, umieszczona w pliku clock.c

```
#include "check.h"
#include "clock.h"
#include <stm32f10x_flash.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>

#ifndef STM32F10X_CL
  #error Przykład jest dla STM32F107.
#endif

/* Konfiguruj wszystkie wyprowadzenia w analogowym trybie
   wejściowym (ang. analog input mode, trigger off), co
   redukuje zużycie energii i zwiększa odporność na
   zakłócenia elektromagnetyczne. */
void AllPinsDisable() {
  GPIO_InitTypeDef GPIO_InitStruct;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
                         RCC_APB2Periph_GPIOB |
                         RCC_APB2Periph_GPIOC |
                         RCC_APB2Periph_GPIOD |
                         RCC_APB2Periph_GPIOE,
                         ENABLE);
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_All;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_InitStruct.GPIO_Speed = 0;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
  GPIO_Init(GPIOB, &GPIO_InitStruct);
  GPIO_Init(GPIOC, &GPIO_InitStruct);
  GPIO_Init(GPIOD, &GPIO_InitStruct);
  GPIO_Init(GPIOE, &GPIO_InitStruct);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
                         RCC_APB2Periph_GPIOB |
                         RCC_APB2Periph_GPIOC |
                         RCC_APB2Periph_GPIOD |
                         RCC_APB2Periph_GPIOE,
                         DISABLE);
}

#ifndef HSE_VALUE
  #error W projekcie nie zdefiniowano częstotliwości kwarcu.
#endif

#if HSE_VALUE == 25000000
  #define RCC_PREDIV2_DivX RCC_PREDIV2_Div5
#elif HSE_VALUE == 20000000
  #define RCC_PREDIV2_DivX RCC_PREDIV2_Div4
#elif HSE_VALUE == 15000000
  #define RCC_PREDIV2_DivX RCC_PREDIV2_Div3
#elif HSE_VALUE == 10000000
  #define RCC_PREDIV2_DivX RCC_PREDIV2_Div2
#elif HSE_VALUE == 5000000
  #define RCC_PREDIV2_DivX RCC_PREDIV2_Div1
#else
  #error Błędna wartość HSE_VALUE
#endif

int ClockConfigure(unsigned freqMHz) {
  static const int maxAttempts = 1000000;
```

padku APB1 ma wartość 2, zatem liczniki te są taktowane z częstotliwością HCLK.

Sygnały taktujące są konfigurowane za pomocą funkcji ClockConfigure, której argumentem jest częstotliwość HCLK (w MHz), jaką chcemy ustawić. Dopuszczalne wartości argumentu to 12, 14, 16, 18, 24, 28, 32, 36, 48, 56, 64, 72 MHz. Funkcja ta zwraca zero, gdy konfiguracja powiedzie się lub wartość ujemną, gdy podano nieprawidłową wartość argumentu, gdy nie uda się wystartować oscylatora kwarcowego HSE lub któraś z PLL nie zsynchronizuje się. Odpowiednia deklaracja jest umieszczona na **listingu 4**. Znajduje się tam też deklaracja funkcji AllPinsDisable, która konfiguruje wszystkie porty wejścia-wyjścia tak, aby zminimalizować pobór prądu przez mikrokontroler. Implementacja zamieszczona jest na **listingu 5**.

Spójrzmy ponownie na rysunek 1. Do wejścia dzielnika AHB jest dołączony sygnał SYSCLK. Może to być sygnał 8 MHz z wewnętrznego oscylatora RC HSI (tak dzieje się po włączeniu zasilania, po resecie lub po wyjściu z trybu głębokiego uśpienia), sygnał HSE z oscylatora kwarcowego lub sygnał PLLCLK z wyjścia PLL1. Do wejścia PLL1 można dołączyć sygnał HSI podzielony przez 2 lub sygnał z wyjścia dzielnika częstotliwości PREDIV1. Do wejścia dzielnika PREDIV1 można dołączyć sygnał HSE lub sygnał PLL2CLK z wyjścia PLL2. Do wejścia PLL2 podłączony jest sygnał HSE przez dzielnik PREDIV2. Dobierając wartości współczynników podziału i mnożniki PLL, możemy uzyskać wiele różnych częstotliwości HCLK taktowania rdzenia.

W omawianym przykładzie zakładamy, że oscylator kwarcowy HSE ma częstotliwość 5, 10, 15, 20 lub 25 MHz. Sygnał z tego oscylatora jest podawany przez dzielnik PREDIV2 na wejście PLL2. Wartość dzielnika dobieramy tak, aby na wejściu PLL2 był zawsze sygnał o częstotliwości 5 MHz. Mnożnik PLL2MUL ustawiamy na 8, zatem na wyjściu PLL2 uzyskujemy sygnał PLL2CLK o częstotliwości 40 MHz. Sygnał ten podajemy na wejście dzielnika PREDIV1, który ustawiamy na 5. Zatem na wyjściu dzielnika PREDIV1 mamy sygnał o częstotliwości 8 MHz, który podajemy na wejście PLL1. Dobierając mnożnik PLLMUL spośród wartości 6, 7, 8 lub 9, uzyskujemy na wyjściu PLL1 sygnał PLLCLK o częstotliwości odpowiednio 48, 56, 64 lub 72 MHz, który podajemy na wejście dzielnika AHB. Ustawiając wartość tego dzielnika na 1, 2 lub 4, uzyskujemy żądaną częstotliwość sygnału HCLK.

Omawiany przykład napisany jest na mikrokontroler STM32F107, który należy do linii zorientowanej na komunikację (*connectivity line*). Używamy Standard Peripheral Library. Podczas kompilacji wszystkich plików źródłowych przykładu oraz plików bibliotecznych muszą być zdefiniowane stałe

preprocesora STM32F10X_CL i USE_STDPERIPH_DRIVER. Stałe te należy wyspecyfikować w bloku konfigurowania projektu lub w wierszu poleceń kompilatora. Analogicznie powinna też zostać zdefiniowana wartość stałej HSE_VALUE, która oznacza częstotliwość podłączonego do mikrokontrolera kwarcu w Hz. Przyjęta konfiguracja sygnałów taktujących dopuszcza częstotliwości kwarcu będące wielokrotnościami 5 MHz. Taka sama wartość tej stałej musi być użyta podczas kompilowania Standard Peripheral Library. Należy dodać, że jeśli nie zdefiniujemy tej stałej, to przy zdefiniowanej stałej STM32F10X_CL domyślna wartość częstotliwości wynosi 25000000 Hz. Taki właśnie kwarc jest zamontowany na „motylu", na którym testowany był omawiany przykład. Należy też zwrócić uwagę, że w wersjach biblioteki starszych niż 3.2.0 stała ta nazywała się HSE_Value. Trzeba wtedy przed pierwszym użyciem stałej HSE_VALUE dodać definicję:

```
#define HSE_VALUE HSE_Value
```

Program i układ testowy

Układ testowy przedstawiony jest na **rysunku 2**. Użyto dwóch diod świecących. LED1 do informowania o poprawnym lub niepoprawnym zakończeniu konfigurowania mikrokontrolera oraz do sygnalizacji, że mikrokontroler jest w stanie aktywnym. LED2 sygnalizuje, że mikrokontroler jest w trakcie obsługi przerwania, które jest zgłaszane narastającym zboczem na wyprowadzeniu PA0 za pomocą przycisku SW. Pozwala to testować wybudzanie z trybów uśpienia i zatrzymania w celu obsłużenia przerwania. Po jego obsłużeniu mikrokontroler powinien ponownie zasnąć. Czas opóźnienia nie powinien zależeć od liczby zgłoszonych i obsłużonych w jego trakcie przerwań. Przycisk SW ma jeszcze jedną funkcję. Wyprowadzenie PA0 skonfigurowano tak, aby narastające zbocze na nim wyprowadzało mikrokontroler z trybu czuwania. Wyjście FREQ służy do pomiaru czasu opóźnień, poprzez pomiar częstotliwości generowanego na nim przebiegu.

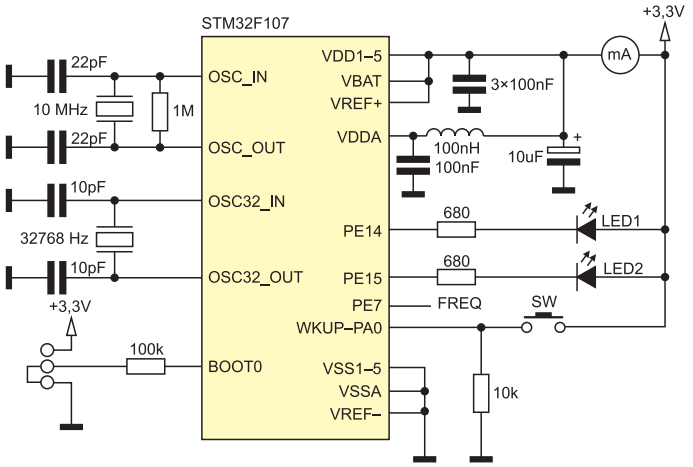
Sygnatury funkcji obsługujących wyjścia przedstawione są na **listingu 6**. Funkcje te są standardowe i dlatego nie zamieszczono ich szczegółowej implementacji. Funkcja OutConfigure konfiguruje porty wyjściowe. Wyprowadzenia LED i FREQ konfigurujemy jako wyjścia przeciwsobne (*push-pull*). Wyjścia LED konfigurujemy dla maksymalnej częstotliwości 2 MHz, a wyjście FREQ – 50 MHz. Funkcje LED1on, LED1off, LED2on i LED2off, zgodnie z nazwami, włączają lub wyłączają odpowiednią diodę LED. Funkcja FREQon ustawia wysoki poziom na wyjściu FREQ, a funkcja FREQoff – poziom niski. Funkcja Error gasi LED1, a potem wykonuje podaną jako argument liczbę mignięć LED1, po czym odczekuje dłuższą chwilę. Funkcja ta sygnalizuje błędy w połączeniu z makrem error_check. Makro to wywołujemy, aby sprawdzić, czy jakaś funkcja zakończyła się prawidłowo. Jako pierwszy argument tego makra umieszczamy wywołanie funkcji, która zwraca zero przy prawidłowym zakończeniu, a wartość ujemną, gdy wystąpi błąd. Jako drugi argument podajemy numer błędu – liczbę mignięć LED1. Przykładowo:

```
error_check(SleepConfigure(HCLK_
MHZ), 2);
```

Sygnatura funkcji KeyConfigure konfigurującej wyprowadzenie, do którego podłączony jest przycisk, przedstawiona jest na **listingu 7**. Jej implementacja zamieszczona jest na **listingu 8**. Znajduje się tam też procedura obsługi przerwania EXTI0_

Listing 5. cd.

```
  RCC_DeInit();
  RCC_HSEConfig(RCC_HSE_ON);
  /* Wykonuje maksymalnie HSEStartUp_TimeOut = 1280 prób. */
  if (RCC_WaitForHSEStartUp() == ERROR)
    return -1;

  FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
  if (freqMHz > 48)
    FLASH_SetLatency(FLASH_Latency_2);
  else if (freqMHz > 14) /* Dokumentacja mówi o 24 MHz. */
    FLASH_SetLatency(FLASH_Latency_1);
  else
    FLASH_SetLatency(FLASH_Latency_0);

  if (freqMHz >= 48)
    /* preskaler AHB, HCLK = SYSCLK = freqMHz */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
  else if (freqMHz >= 24)
    /* preskaler AHB, HCLK = SYSCLK / 2 = freqMHz */
    RCC_HCLKConfig(RCC_SYSCLK_Div2);
  else
    /* preskaler AHB, HCLK = SYSCLK / 4 = freqMHz */
    RCC_HCLKConfig(RCC_SYSCLK_Div4);

  /* preskaler APB1, PCLK1 = HCLK / 2 = freqMHz / 2 */
  RCC_PCLK1Config(RCC_HCLK_Div2);

  /* preskaler APB2, PCLK2 = HCLK = freqMHz */
  RCC_PCLK2Config(RCC_HCLK_Div1);

  /* PLL2: PLL2CLK = HSE / RCC_PREDIV2_DivX * 8 = 40 MHz */
  RCC_PREDIV2Config(RCC_PREDIV2_DivX);
  RCC_PLL2Config(RCC_PLL2Mul_8);
  RCC_PLL2Cmd(ENABLE);
  active_check(RCC_GetFlagStatus(RCC_FLAG_PLL2RDY), maxAttempts);

  /* PLL1: PLLCLK = (PLL2 / 5) * RCC_PLLMul */
  RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2, RCC_PREDIV1_Div5);
  if (freqMHz == 48 || freqMHz == 24 || freqMHz == 12)
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_6);
  else if (freqMHz == 56 || freqMHz == 28 || freqMHz == 14)
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_7);
  else if (freqMHz == 64 || freqMHz == 32 || freqMHz == 16)
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_8);
  else if (freqMHz == 72 || freqMHz == 36 || freqMHz == 18)
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_9);
  else
    return -1;
  RCC_PLLCmd(ENABLE);
  active_check(RCC_GetFlagStatus(RCC_FLAG_PLLRDY), maxAttempts);

  /* Ustaw SYSCLK = PLLCLK i czekaj aż PLL zostanie ustawiony
     jako zegar systemowy. */
  RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
  active_check(RCC_GetSYSCLKSource() == 0x08, maxAttempts);

  return 0;
}
```



Rysunek 2. Schemat układu testowego

Listing 6. Sygnatury funkcji obsługujących wyjścia, umieszczone w pliku out.h

```
#ifndef _OUT_H
#define _OUT_H 1

void OutConfigure(void);
void LED1on(void);
void LED1off(void);
void LED2on(void);
void LED2off(void);
void FREQon(void);
void FREQoff(void);
void Error(int);

#define error_check(expr, err)  \
  if ((expr) < 0) {             \
    for (;;) {                  \
      Error(err);               \
    }                           \
  }

#endif
```

Listing 7. Sygnatura funkcji konfigurującej przycisk, umieszczona w pliku key.h

```
#ifndef _KEY_H
#define _KEY_H 1

void KeyConfigure(void);

#endif
```

Listing 8. Implementacja obsługi przycisku, umieszczona w pliku key.c

```
#include "delay.h"
#include "key.h"
#include "out.h"
#include <misc.h>
#include <stm32f10x_exti.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>

void KeyConfigure(void) {
  GPIO_InitTypeDef GPIO_InitStruct;
  EXTI_InitTypeDef EXTI_InitStruct;
  NVIC_InitTypeDef NVIC_InitStruct;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

  /* Stanem aktywnym PA0 jest poziom wysoki.
     Jest zewnętrzny rezystor ściągający do masy. */
  GPIO_StructInit(&GPIO_InitStruct);
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* Narastające zbocze na PA0 wyzwala przerwanie EXTI0. */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
  EXTI_StructInit(&EXTI_InitStruct);
  EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStruct.EXTI_Line = EXTI_Line0;
  EXTI_InitStruct.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStruct);

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
  NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 2;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);
}

/* Przerwanie zgłaszane, gdy wciśnięto przycisk. */
void EXTI0_IRQHandler(void) {
  if (EXTI_GetITStatus(EXTI_Line0)) {
    EXTI_ClearITPendingBit(EXTI_Line0);
    /* Symuluj jakąś pracę. */
    LED2on();
    Delay(1000);
    LED2off();
  }
}
```

IRQHandler, która symuluje wykonywanie jakichś obliczeń za pomocą funkcji Delay. Na **listingu 9** zamieszczony jest przykładowy program główny. Częstotliwość (w MHz) taktowania rdzenia i liczników ustawiamy za pomocą stałej HCLK_MHZ. Rodzaj testu wybieramy za pomocą makrodefinicji:

```
#define SUPPLY_CURRENT
#define USLEEP_TIME 1
#define MSLEEP_TIME 1
```

Pobór prądu zasilania możemy mierzyć, definiując stałą SUPPLY_CURRENT. Wprowadzenie symbolu komentarza w linii tej definicji umożliwia pomiar czasu opóźnień mikrosekundowych i milisekundowych. Na wyjściu FREQ pojawia się wtedy sygnał o częstotliwości będącej w przybliżeniu połową odwrotności opóźnienia zadanego definicjami USLEEP_TIME i MSLEEP_TIME. Jeśli zdefiniowana jest stała USLEEP_TIME, to w pętli wywoływana jest dwukrotnie funkcja USleep z argumentem równym tej stałej. Analogicznie jeśli zdefiniowana jest stała MSLEEP_TIME, to w pętli wywoływana jest dwukrotnie funkcja MSleep z argumentem równym tej stałej. W czasie pierwszego wywołania każdej z tych funkcji, na wyjściu FREQ jest poziom wysoki, a w czasie drugiego – poziom niski. Jeśli żadna z tych stałych nie jest zdefiniowana, mierzymy częstotliwość iteracji „pustej" pętli, bez funkcji opóźniających. Zmierzony czas opóźnienia obliczamy jako połowę różnicy miedzy okresem iteracji pętli z funkcjami opóźniającymi, a okresem iteracji pętli bez funkcji opóźniających. Sposób pracy regulatora 1,8 V w trybie zatrzymania wybieramy za pomocą stałej PWR_REGULATOR_MODE, zdefiniowanej w pliku sleep.c, co zostało omówione w pierwszej części artykułu.

Aby skompilować przykład, potrzebny jest jeszcze plik nagłówkowy delay.h zawierający sygnaturę funkcji Delay, która została przedstawiona na początku pierwszej części artykułu. W pliku delay.c należy umieścić jej implementację, a w pliku out.c – implementacje funkcji, których sygnatury są na listingu 6. Ponadto Standard Peripheral Library wymaga obecności pliku stm32f10x_conf.h, który w naszym przypadku zawiera tylko puste makro:

```
#define assert_param(expr) \
((void)0)
```

Wyniki testów

Układ z **rysunku 2** można uruchomić na „motylu", który trzeba doposażyć w kwarc zegarkowy i dwa kondensatory – odpowiednie miejsce zostało przewidziane na płytce. Potrzebne dwie diody świecące są podłączone tak samo jak na rysunku 2. Do wyprowadzenia PA0 trzeba podłączyć rezystor ściągający do masy i przycisk do zasilania. Wyprowadzenie PA0 jest dostępne na PIN2 złącza JP1 modułu ethernetowego – w omawianym przykładzie moduł ten nie jest używany. Do styków 1 i 2 złącza JP6 zamiast zworki podłączamy amperomierz, który mierzy prąd pobierany przez mikrokontroler. Do wyprowadzenia FREQ (PE7) podłączamy częstościomierz.

W **tabeli 1** zestawiono zmierzone wartości prądu zasilania dla sześciu wartości HCLK, po trzy dla minimalnej i maksymalnej częstotliwości SYSCLK, jakie można uzyskać z wyjścia PLL1. Pomiary zostały wykonane przy odłączonym interfejsie

JTAG, którego działanie wpływa na ich wyniki, szczególnie istotnie zwiększając pobór prądu w trybie zatrzymania. W trybie aktywnym i uśpienia prąd zasilania powinien zależeć od częstotliwości taktowania. Widać jednak, że nie jest on proporcjonalny do częstotliwości HCLK i zależy też od częstotliwości SYSCLK. Najprawdopodobniej duża część prądu zasilania jest pobierana przez układy taktujące. Potwierdzają to pomiary dla trybu uśpienia, w którym przestają być taktowane rdzeń i pamięci. Ponieważ w testowanym układzie aktywna jest stosunkowo niewielka część peryferii, to za pobór prądu w tym trybie muszą odpowiadać głównie oscylator HSE i pętle PLL. W trybach głębokiego uśpienia (zatrzymania i czuwania) pobór prądu nie zależy od częstotliwości taktowania, gdyż wtedy HSE jest wyłączony. Wartości prądu zasilania w tych trybach są, zgodnie z oczekiwaniem, rzędu mikroamperów. Należy podkreślić, że dla uzyskania tak niskiego poboru prądu konieczne jest wywołanie funkcji AllPinsDisable. Jej pominięcie i pozostawienie „wiszących" wejść niweczy uzyskane oszczędności, zwłaszcza w trybie zatrzymania, gdyż mikrokontroler pobiera wtedy dodatkowo ponad 1,5 mA.

W **tabeli 2** zestawiono wyniki pomiarów opóźnień. Minimalne opóźnienie, jakie daje się uzyskać, wywołując funkcję USleep z parametrem 1, zależy od częstotliwości taktowania rdzenia, czego należało oczekiwać, gdyż wynika ono z liczby instrukcji, które musi wykonać ta funkcja i procedura obsługi przerwania licznika. Przy maksymalnej dopuszczalnej częstotliwości taktowania 72 MHz nie udało się uzyskać opóźnienia krótszego niż 3 μs. Należy uznać, że dokładność realizacji opóźnień jest zadowalająca.

Marcin Peczarski

Listing 9. Program testowy umieszczony w pliku main.c

```
#include "clock.h"
#include "delay.h"
#include "key.h"
#include "out.h"
#include "sleep.h"

#define HSI_MHZ    8
#define HCLK_MHZ   72
// #define SUPPLY_CURRENT
// #define USLEEP_TIME 1
// #define MSLEEP_TIME 1
#define HSI_DELAY  (HSI_MHZ  * 1000000)
#define HCLK_DELAY (HCLK_MHZ * 1000000)

int main() {
  AllPinsDisable();
  OutConfigure();
  KeyConfigure();

  LED1on();
  Delay(HSI_DELAY);
  error_check(ClockConfigure(HCLK_MHZ), 1);
  error_check(SleepConfigure(HCLK_MHZ), 2);
  LED1off();

  #ifdef SUPPLY_CURRENT
    USleep(65535);
    LED1on();
    Delay(HCLK_DELAY);
    LED1off();
    MSleep(8000);
    LED1on();
    Delay(HCLK_DELAY);
    LED1off();
    Sleep(12);
    LED1on();
    Delay(HCLK_DELAY);
    /* Dioda przestaje świecić po wejściu w tryb czuwania. */
    Standby(16);
    /* Program nigdy tu nie dochodzi. */
    error_check(-1, 3);
  #else
    for (;;) {
      FREQon();
      #ifdef USLEEP_TIME
        USleep(USLEEP_TIME);
      #endif
      #ifdef MSLEEP_TIME
        MSleep(MSLEEP_TIME);
      #endif
      FREQoff();
      #ifdef USLEEP_TIME
        USleep(USLEEP_TIME);
      #endif
      #ifdef MSLEEP_TIME
        MSleep(MSLEEP_TIME);
      #endif
    }
  #endif
}
```

Tabela 1. Prąd zasilania w zależności od częstotliwości taktowania w poszczególnych trybach oszczędzania energii

Częstotliwość nominalna HCLK	12 MHz	18 MHz	24 MHz	36 MHz	48 MHz	72 MHz
Częstotliwość nominalna SYSCLK	48 MHz	72 MHz	48 MHz	72 MHz	48 MHz	72 MHz
Dzielnik AHB	4	4	2	2	1	1
Tryb aktywny	10,8 ± 0,5 mA	15,4 ± 0,8 mA	17,3 ± 1,0 mA	24,5 ± 1,3 mA	28,3 ± 1,3 mA	35,7 ± 1,4 mA
Tryb uśpienia	5,9 ± 0,2 mA	7,6 ± 0,4 mA	7,1 ± 0,4 mA	9,4 ± 0,6 mA	9,3 ± 0,6 mA	12,7 ± 1,0 mA
Tryb zatrzymania – regulator 1,8 V włączony	35 ± 1,5 μA					
Tryb zatrzymania – regulator 1,8 V w stanie niskiego poboru energii	27 ± 1,4 μA					
Tryb czuwania	3,36 ± 0,04 μA					

Tabela 2. Pomiary rzeczywistych czasów opóźnień

Częstotliwość nominalna HCLK	18 MHz	36 MHz	72 MHz
Częstotliwość f_0 iteracji pętli bez funkcji opóźniających	300 kHz	563 kHz	912 kHz
Częstotliwość f_1 iteracji pętli z opóźnieniami 1 μs + 1 μs	47,9 kHz	85,3 kHz	136,4 kHz
Częstotliwość f_{20} iteracji pętli z opóźnieniami 20 μs + 20 μs	22,6 kHz	24,3 kHz	24,4 kHz
Częstotliwość f_{1m} iteracji pętli z opóźnieniami 1 ms + 1 ms	0,498 kHz	0,499 kHz	0,500 kHz
Najkrótsze możliwe opóźnienie: $0{,}5(1/f_1 - 1/f_0)$	8,8 ± 0,1 μs	4,97 ± 0,04 μs	3,12 ± 0,02 μs
Opóźnienie 20 μs: $0{,}5(1/f_{20} - 1/f_0)$	20,4 ± 0,4 μs	19,7 ± 0,3 μs	20,0 ± 0,3 μs
Opóźnienie 1 ms: $0{,}5(1/f_{1m} - 1/f_0)$	1002 ± 7 μs	1001 ± 7 μs	999,5 ± 7 μs
Dokładność pomiaru częstotliwości ± (0,1% + 3)			