Ćwiczenie 2: Konfiguracja sygnałów zegarowych. Porty wejścia/wyjścia ogólnego przeznaczenia

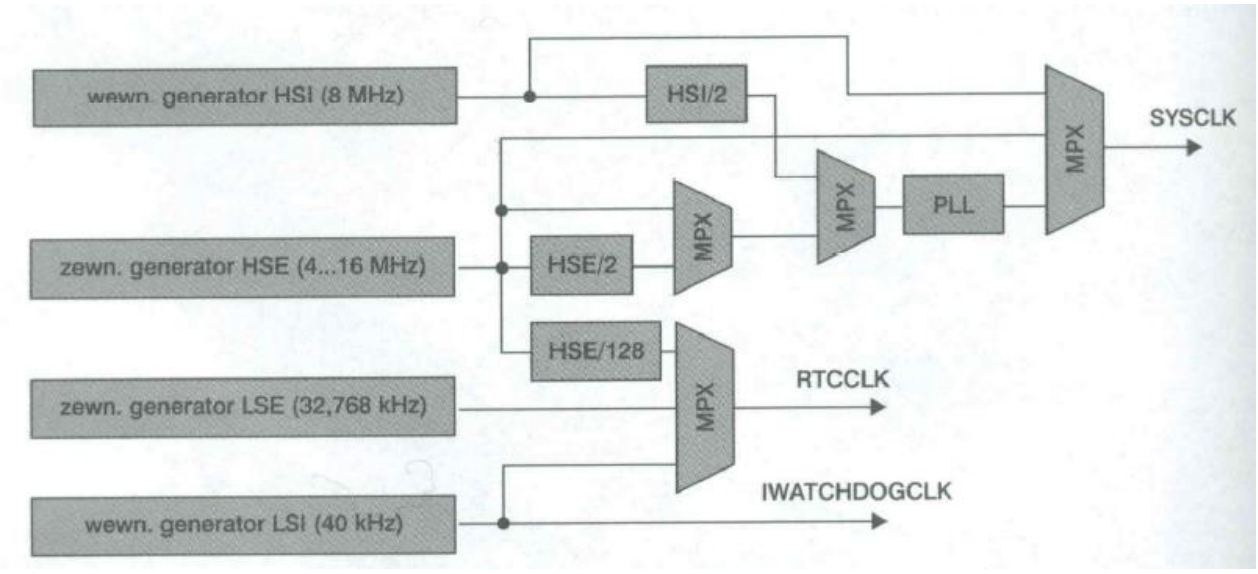
Zakres materiału

1. Sygnały zegarowe i ich konfiguracja.
2. Budowa portów wejścia/wyjścia ogólnego przeznaczenia w mikrokontrolerze STM32F103.
3. Programowanie portów i sterowanie kierunkiem portu.
4. Techniki programowania pętli opóźniających (instrukcje nop, goto, decfsz).

Wprowadzenie do ćwiczeń

1. Konfiguracja sygnału zegarowego

Sygnał zegara systemowego może w mikrokontrolerach z rodziny STM32 pochodzić z trzech źródeł: zewnętrznego (HSE — High Speed External), którym może być rezonator kwarcowy (ceramiczny), odpowiedni przebieg podawany bezpośrednio na wejście generatora lub z dwóch źródeł wewnętrznych: powielacza częstotliwości z PLL (Phase Locked Loop) lub wewnętrznego generatora RC (HSI — High Speed Internal). W mikrokontrolerach STM32 dostępne są dwa dodatkowe źródła sygnałów zegarowych, są to: wewnętrzny generator RC o częstotliwości 40 kHz (LSI — Low Speed Internal) oraz zewnętrzny rezonator o częstotliwości „zegarkowej", czyli 32,768 kHz (LSE — Low Speed External). Pierwszy z nich może być wykorzystywany jako źródło sygnału zegarowego dla sprzętowego watchdoga i zegara czasu rzeczywistego (RTC), natomiast drugi może być stosowany tylko do taktowania RTC (Rys 1).



Rys. 1. Sygnały zegarowe w STM32

Pierwszą czynnością wymaganą do poprawnej pracy mikrokontrolerów z rdzeniem Cortex-M3 jest konfiguracja sygnałów zegarowych i zerujących. Domyślnie po włączeniu zasilania mikrokontroler wykorzystuje do pracy wewnętrzny szybki oscylator (HSI). Jeżeli aplikacja ma pracować z zewnętrznym rezonatorem, to należy go w pierwszej kolejności włączyć.

Pierwszą czynnością jest ustawienie wszystkich rejestrów RCC (Reset and Clock Control) do ich wartości domyślnych, służy do tego wywołanie funkcji RCC_Delnit(). Następnie, aby system współpracował z zewnętrznym rezonatorem, to trzeba go uruchomić, do tego celu wykorzystywana jest funkcja RCC

HSEConfig() wywoływana z argumentem RCC_HSE_ON. Uruchomienie się rezonatora kwarcowego wymaga nieco czasu, a zatem mikrokontroler musi poczekać, aż sygnał zegarowy będzie stabilny. Jeśli wszystko uruchomiło się bez błędów, to mikrokontroler przechodzi do kolejnych niezbędnych konfiguracji. W przypadku, kiedy zewnętrzny rezonator nie dostarczy poprawnego sygnału, mikrokontroler będzie korzystał z wewnętrznego szybkiego oscylatora HSI. Informację o stanie źródła sygnału zegarowego zawiera zmienna HSEStartUpStatus i na podstawie jej wartości mogą być podejmowane dalsze decyzje zapobiegające niepoprawnemu działaniu systemu. Gdy zewnętrzny rezonator poprawnie wystartuje, to konfigurujemy sposób pracy pamięci Flash. Oprócz włączenia bufora dla pamięci ważne jest ustalenie współczynnika opóźnienia dostępu do pamięci w stosunku do zegara systemowego SYSCLK. W zależności od tego, z jaką częstotliwością pracuje mikrokontroler wybieramy stosowne opóźnienie. Jeżeli częstotliwość SYSCLK zawiera się w przedziale od 48 do 72 MHz, to opóźnienie dostępu musi wynosić dwa cykle, gdy mikrokontroler pracuje z zegarem do 24 MHz, to nie jest wymagane opóźnienie. W pozostałych przypadkach, czyli dla częstotliwości z przedziału 24 do 48 MHz, współczynnik opóźnienia dostępu do pamięci musi wynosić 1 cykl.

Kolejnym etapem uruchamiania systemu jest ustalenie źródeł i częstotliwości taktowania poszczególnych wewnętrznych magistral i samego rdzenia. Argument w wywołaniu funkcji RCC_HCLKConfig() określa, przez jaką liczbę ma być podzielony sygnał zegarowy dla rdzenia. W naszym przypadku sygnał SYSCLK jest tożsamy z HCLK (zegarem rdzenia) — nie dzielimy sygnału SYSCLK. Ustawienia częstotliwości taktowania wymagają także magistrale bloków peryferyjnych. Peryferia mikrokontrolerów STM32 są podłączone do jednej z dwóch magistral: APB1 lub APB2. Sygnały taktujące te magistrale nazywają się odpowiednio: PCLK1 i PCLK2. Magistrala APB 1 może pracować z maksymalną częstotliwością taktującą 36 MHz. Ponieważ docelowo rdzeń ma być taktowany z częstotliwością 72 MHz, to należy częstotliwość sygnału HCLK (z którego jest wytwarzany sygnał PCLK I) podzielić przez 2. Maksymalną częstotliwością pracy APB2 jest 72 MHz, a więc HCLK nie musi (ale może) być dzielony i może bezpośrednio stanowić sygnał PCLK2.

Ponieważ wykorzystujemy rezonator kwarcowy o częstotliwości 8 MHz, to aby uzyskać sygnał o częstotliwości 72 MHz należy częstotliwość rezonatora powielić 9 razy. Do tego celu wykorzystano wbudowaną w mikrokontroler pętlę PLL. Funkcja RCC_PLLConfig() jest wywoływana w celu ustawienia źródła powielanego sygnału i wartości mnożnika. Następnie układ PLL jest włączany i po ustaleniu sygnału wyjściowego za pomocą funkcji RCC_SYSCLKConfig() zostaje wybrany jako źródło zegara systemowego. Ostatnią czynnością, niezbędną do uruchomienia systemu z zewnętrznego oscylatora, jest poinformowanie mikrokontrolera z jakiego źródła ma pochodzić sygnał SYSCLK. W przedstawianym przykładzie jest to oczywiście sygnał pochodzący z układu PLL. Na koniec pozostaje już tylko włączenie wykorzystywanych w danym projekcie peryferii, np. portów we/wy GPIOx.

Kod funkcji RCC_Config, konfigurującej sygnał zegarowy pokazano w przykładzie 1.

2. Porty wejścia/wyjścia GPIO

Aby używać portów wejścia/wyjścia należy je uprzednio skonfigurować. Najprościej to zrobić używając modułu GPIO z biblioteki StdPeriph i struktury typu GPIO_InitTypeDef zawierającej 3 pola:

- GPIO_Pin – numery pinów, które konfigurujemy (np. GPIO_Pin_1)
- GPIO_Speed – szybkość taktowania pinów (2, 10, 50 MHz)
- GPIO_Mode – tryb pracy

Linie portów wejścia/wyjścia (GPIO – General Purpose Input/Output) w mikrokontrolerach STM32 mogą pracować w jednym z 6 trybów:

- Jako wejścia:
    - „pływające” (bez wewnętrznego podciągania) *GPIO_Mode_IN_FLOATING*
    - Z podciąganiem do plusa zasilania (pull-up) *GPIO_Mode_IPU*
    - Z podciąganiem do masy (pull-down) *GPIO_Mode_IPD*

- Analogowe *GPIO_Mode_AIN*

- Jako wyjścia:
  - Z otwartym drenem GPIO_Mode_Out_OD
  - Symetryczne push-pull GPIO_Mode_Out_PP

Biblioteka StdPeriph oferuje następujące funkcje do zmiany stanu i odczytu portów GPIO:

- uint8_t GPIO_ReadInputDataBit (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
  - Czyta wartość bitu GPIO_Pin z portu wejściowego GPIOx, gdzie x – „litera" portu.
- uint16_t GPIO_ReadInputData (GPIO_TypeDef *GPIOx)
  - Czyta wartość całego portu wejściowego GPIOx.
- uint8_t GPIO_ReadOutputDataBit (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
  - Czyta wartość bitu GPIO_Pin z portu wyjściowego GPIOx.
- uint16_t GPIO_ReadOutputData (GPIO_TypeDef *GPIOx)
  - Czyta wartość całego portu wyściowego GPIOx.
- void GPIO_SetBits (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
  - Ustawia wybrane bity portu na 1.
- void GPIO_ResetBits (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
  - Zeruje wybrane bity portu.
- void GPIO_WriteBit (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
  - Ustawia lub zeruje odpowiednie bity portu, BitVal = Bit_SET lub Bit_RESET.
- void GPIO_Write (GPIO_TypeDef *GPIOx, uint16_t PortVal)
  - Zapisuje dane do podanego portu.

3. Opóźnienia

Do realizacji opóźnień można stosować następującą technikę (liczbę powtórzeń pętli można wyznaczyć doświadczalnie w symulatorze):

```
for (i = 0; i <0x320000; i++); //około 0.5s przy 72MHz
```

Należy pamiętać, że pętle takie pozwalają na w miarę dokładne odmierzanie interwałów czasowych wyłącznie wtedy, gdy system przerwań jest wyłączony. Jednakże nie są polecane do tego celu. Dokładne odmierzanie czasu może być realizowane za pomocą przerwań od zegara systemowego.

4. Diody LED i przyciski

Dioda LED (dioda elektroluminescencyjna, LED – ang. *Light Emitting Diode*) jest to półprzewodnikowy element elektroniczny, który emituje światło pod wpływem przepływającego prądu elektrycznego. Świecenie diody następuje, gdy dioda LED jest spolaryzowana w kierunku przewodzenia, tzn. do anody podłączony jest biegun dodatni, a do katody biegun ujemny.

Sterowanie diodą LED jest bardzo proste i wymaga wykonania dwóch czynności:

- ustawienie wyprowadzenia portu, do którego jest podłączona dioda LED, w tryb pracy wyjścia Czynność tą należy wykonać na początku programu.
- w zależności od potrzeb należy wyzerować lub ustawić odpowiedni bit portu GPIOx:
  - jeśli chcemy zapalić diodę, to bit należy ustawić,
  - jeśli chcemy zgasić diodę, to bit należy wyzerować.

Bardzo często istnieje potrzeba sterowania systemem wbudowanym za pomocą przycisku lub zestawu przycisków, a nawet klawiaturą. Przyciski najczęściej wykonywane są w postaci mikroprzełączników, które po naciśnięciu zwierają styki. Do obsługi przycisków należy wiedzieć, że:

1. Stan przycisków sygnalizowany jest poprzez tzw. logikę ujemną, co oznacza, że przycisk wciśnięty powoduje pojawienie się stanu niskiego (zera) na wejściu, natomiast przycisk zwolniony powoduje pojawienie się stanu wysokiego (jedynki) na wejściu.
2. Wciśnięcie przycisku jest procesem dynamicznym, tzn. jest sygnalizowane zmianą stanu. W związku z tym należy sprawdzić czy nastąpiła zmiana wartości na wejściu z jedynki na zero.
3. W czasie wciskania przycisku występuje iskrzenie (bouncing). Odczytywane jest ono jako trwająca kilka milisekund sekwencja następujących po sobie zer i jedynek. Aby wyeliminować błędy odczytywania stanu przycisków (tzw. debouncing) należy stosować opóźnienia (iskrzenie nie trwa z reguły dłużej, niż kilka milisekund).

Pytania kontrolne:

1. W jaki sposób można określić kierunek pracy portu (wejście lub wyjście)?
2. W jaki sposób jest ustawiana wartość na wyjściu portu?
3. W jaki sposób można sprawdzić stan logiczny na liniach portu?
4. W jaki sposób można generować opóźnienia?
5. W jaki sposób konfigurujemy mikrokontroler do pracy z zewnętrznym i wewnętrznym sygnałem zegarowym?
6. Co to jest iskrzenie i w jaki sposób można to zjawisko zlikwidować?
7. W jaki sposób można sprawdzić wciśnięcie przycisku?

Przykłady:

1. Konfiguracja zegara

Funkcja która ustawia mikrokontroler do pracy z sygnałem zegarowym zewnętrznym o częstotliwości 8 MHz, taktowanie rdzenia – 72MHz. (Funkcja znajduje się zarówno w pliku **cw2p1.c** jak i **cw2p2.c**)

```
void RCC_Config(void)
//konfigurowanie sygnalow taktujacych
{
    ErrorStatus HSEStartUpStatus; //zmienna opisujaca rezultat uruchomienia HSE
    RCC_DeInit();                                   //Reset ustawien RCC
    RCC_HSEConfig(RCC_HSE_ON);                      //Wlaczenie HSE
    HSEStartUpStatus = RCC_WaitForHSEStartUp();     //Odczekaj na gotowosc HSE
    if(HSEStartUpStatus == SUCCESS)
    {
           FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
           FLASH_SetLatency(FLASH_Latency_2);       //ustaw zwloke dla pamieci Flash;
                                                    //zaleznie od taktowania rdzenia
                                                    //0:<24MHz; 1:24~48MHz; 2:>48MHz

           RCC_HCLKConfig(RCC_SYSCLK_Div1); //ustaw HCLK=SYSCLK
           RCC_PCLK2Config(RCC_HCLK_Div1); //ustaw PCLK2=HCLK
           RCC_PCLK1Config(RCC_HCLK_Div2); //ustaw PCLK1=HCLK/2

           //ustaw PLLCLK = HSE*9 czyli 8MHz * 9 = 72 MHz
           RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
           RCC_PLLCmd(ENABLE); //wlacz PLL

           //odczekaj na poprawne uruchomienie PLL
           while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);

           //ustaw PLL jako zrodlo sygnalu zegarowego
           RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

           //odczekaj az PLL bedzie sygnalem zegarowym systemu
           while(RCC_GetSYSCLKSource() != 0x08);
    } else {
           }
```

```
    //wlacz taktowanie portów GPIO B i A
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOA, ENABLE);
    }
```

2. Migające diody

Program, który miga diodami LED z częstotliwością około 1Hz. Przed uruchomieniem programu należy połączyć przewodami: PB0 - LED0, PB1 – LED1.(kod źródłowy znajduje się w pliku **cw2p1.c**).

```
void GPIO_Config(void)
{
  //konfigurowanie portow GPIO
  GPIO_InitTypeDef  GPIO_InitStructure;

  /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO potrzebnych w
programie*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 |
GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}

int main(void)
{
  volatile unsigned long int i;

  //konfiguracja systemu
  RCC_Config();
  GPIO_Config();

  /*Tu nalezy umiescic ewentualne dalsze funkcje konfigurujace system*/
  GPIO_ResetBits(GPIOB,  GPIO_Pin_0  |  GPIO_Pin_1  |  GPIO_Pin_2  |  GPIO_Pin_3  |  GPIO_Pin_4  |
GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);

  while (1) {
    /*Tu nalezy umiescic glowny kod programu*/
       GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);
       for(i = 0; i < 0x320000ul; i++);                 //okolo 0.5s przy 72MHz
       GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_RESET);
       GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
       for(i = 0; i < 0x320000ul; i++);
       GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
  };
  return 0;
}
```

3. Przyciski

Program, który zmienia stan diody LED0 po naciśnięciu przycisku SW0 Przed uruchomieniem programu należy połączyć przewodami: PB0 - LED0, PA0 – SW0.(kod źródłowy znajduje się w pliku **cw2p2.c**).

```
void GPIO_Config(void)
{
  //konfigurowanie portow GPIO
  GPIO_InitTypeDef  GPIO_InitStructure;

  /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO potrzebnych w
programie*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 |
GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_In_Floating;      //wejście bez podciągania
  GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
}
int main(void)
{
  volatile unsigned long int i;
  uint8_t button_state=0xFF, temp=0, port_data ;

  //konfiguracja systemu
  RCC_Config();
  GPIO_Config();

/*Tu nalezy umiescic ewentualne dalsze funkcje konfigurujace system*/
  GPIO_ResetBits(GPIOB,  GPIO_Pin_0  |  GPIO_Pin_1  |  GPIO_Pin_2  |  GPIO_Pin_3  |  GPIO_Pin_4  |
  GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);
  GPIO_ResetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);

  while (1) {
       /*Tu nalezy umiescic glowny kod programu*/
       port_data = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
       temp = port_data ^ button_state; // czy stan przycisków sie zmienil?
       temp &= button_state; // czy to byla zmiana z 1 na 0?
       button_state = port_data; // zapamietaj nowy stan

       if (temp & 0x01)                              // czy to przycisk SW0?

       // zmien stan LED na przeciwny
       GPIO_WriteBit(GPIOB,GPIO_Pin_0,(BitAction)(1-GPIO_ReadOutputDataBit(GPIOB,GPIO_Pin_0)));
       };

  return 0;
}
```

Zadania do samodzielnego wykonania:

1. Napisz program, który miga 4 diodami w taki sposób, że każda następna miga 2 razy szybciej niż poprzednia.

2. Napisz program, który zapala i gasi diodę przyporządkowaną danemu przyciskowi (SW0 – LED0, SW1 - LED1, itd.)

3. Napisz program, który wykorzystuje joystick do sterowania przesuwającą się cyklicznie zapaloną diodą. (W – przesuw w lewo, E – w prawo, N – zwiększ szybkość przesuwania, S – zmniejsz szybkość przesuwania). Dodatkowo przyciskami SW0 i SW1 należy zwiększać i zmniejszać liczbę przesuwających się zapalonych diod.

UWAGA: W przypadku wykorzystywania wyprowadzeń portu GPIOB może zachodzić konieczność przełączenia pinów 3 i 4 z trybu JTAG/SW w tryb GPIO, aby to zrobić należy:

A) W funkcji GPIO_Config() umieścić na początku następującą linię (zremapować piny na GPIO):

```
GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
```

B) W funkcji RCC_Config() dodać taktowanie dla alternatywnych funkcji GPIOB:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
```