Politechnika Wrocławska

Sterowniki Robotów

Lab 03 - Liczniki i przerwania

Wydział i kierunek	W12N, Automatyka i Robotyka
Autor i numery albumów	Michał Markuzel 275417
Termin zajęć	Środa 15:15-16:55
Kod grupy zajeciowej	W12AIR-SI0102G
Prowadzący	dr inż. Wojciech Domski

1 Wstęp

Celem laboratorium było zapoznanie się z podstawowymi trybami pracy liczników w mikrokontrolerach STM32, w tym: generatorem podstawy czasu, trybem Input Capture oraz generowaniem sygnału PWM. Dodatkowo, ćwiczenie obejmowało obsługę przerwań wewnętrznych i zewnętrznych. Wykonano trzy zadania oraz zadanie domowe polegające na pomiarze wypełnienia sygnału PWM.

2 Zadanie 1: Generator podstawy czasu

2.1 Konfiguracja

- **Wybór licznika:** Wykorzystano licznik **TIM6**, który w mikrokontrolerze STM32L476RG jest 16-bitowym timerem bazowym, niewymagającym przypisania do pinów I/O. Został on skonfigurowany jako generator podstawy czasu, czyli układ generujący przerwania z określoną częstotliwością.
- **Parametry licznika:**
  - **Prescaler (TIM6_PRESCALER):** Ustawiono tak, aby odpowiednio podzielić sygnał taktujący timer. Wartość prescalera została dobrana w taki sposób, by po jego zastosowaniu oraz odpowiednim doborze okresu (period), przerwanie występowało co dokładnie 1 sekundę.
  - **Okres (TIM6_PERIOD):** Określa liczbę taktów, po której generowane jest przerwanie. Wraz z prescalerem umożliwia uzyskanie wymaganej częstotliwości przerwań (1 Hz).
- **Stałe konfiguracyjne:** Wartości `TIM6_PRESCALER` oraz `TIM6_PERIOD` zostały zdefiniowane w pliku `main.h` i przypisane w konfiguratorze STM32CubeMX, co pozwala na łatwą ich edycję bez bezpośredniego modyfikowania kodu źródłowego.
- **Przerwanie:** Włączono globalne przerwanie dla licznika TIM6 w zakładce *NVIC Settings*. Przerwanie to jest współdzielone z przetwornikiem DAC i nosi nazwę `TIM6_DAC_IRQHandler`.
- **Uruchomienie timera:** W funkcji `main.c` wywołano funkcję `HAL_TIM_Base_Start_IT(&htim6)`, która uruchamia timer w trybie generującym przerwania.

2.2 Kod programu

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim == &htim6) {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
}
```

Funkcja `HAL_TIM_PeriodElapsedCallback` została zdefiniowana w pliku użytkownika i służy do obsługi przerwań. Dzięki użyciu słowa kluczowego `__weak` w definicji domyślnej funkcji zwrotnej, możliwe jest jej nadpisanie. Warunek `if (htim == &htim6)` sprawdza, czy przerwanie pochodzi od właściwego licznika, ponieważ funkcja callback może być wspólna dla wielu timerów.

2.3 Działanie i teoria obsługi przerwań

Gdy licznik TIM6 osiągnie zadaną wartość (ustawioną przez prescaler i okres), generuje przerwanie. Wywoływana jest procedura `TIM6_DAC_IRQHandler`, która z kolei wywołuje funkcję `HAL_TIM_IRQHandler`, obsługującą flagi przerwania. W wyniku tego uruchamiana jest funkcja zwrotna `HAL_TIM_PeriodElapsedCallback`.

W obsłudze przerwania nie należy wykonywać operacji czasochłonnych. W tym przypadku ograniczono się jedynie do zmiany stanu diody LED, co jest operacją natychmiastową.

2.4 Wynik

Dioda LED zmieniała swój stan co 1 sekundę, co potwierdziło poprawność konfiguracji licznika jako generatora podstawy czasu oraz działanie przerwania.

3 Zadanie 2: Input Capture

3.1 Konfiguracja

- **Wybór licznika:** Użyto licznika **TIM5** w trybie Input Capture.
- **Parametry licznika:**
  - **Prescaler (TIM5_PRESCALER):** Ustawiono wartość odpowiadającą spowolnieniu zegara bazowego tak, by czas mógł być mierzony w milisekundach.
  - **Okres (TIM5_PERIOD):** Ustawiono na wartość wystarczająco dużą, aby umożliwić pomiar bez przepełnienia licznika.
- **Wejście:** Pin **PA0** skonfigurowano jako wejście typu **Input Capture** z detekcją zbocza **opadającego**. Dodatkowo ustawiono **pull-up**.
- **Przerwanie:** Włączono przerwanie dla licznika TIM5 w konfiguracji NVIC.
- **USART:** Skonfigurowano **USART2** w trybie asynchronicznym do wyświetlania danych pomiarowych.

3.2 Kod programu

```
#include "stdio.h"

volatile int flag;
volatile int period;

int _write(int file, char *ptr, int len) {
    HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, 50);
    return len;
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim == &htim5) {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
            period = __HAL_TIM_GET_COMPARE(&htim5, TIM_CHANNEL_1);
            __HAL_TIM_SET_COUNTER(&htim5, 0); // linia opcjonalna     do
    usuni cia przy sprz towym resetowaniu
            flag = 1;
        }
    }
}

int main(void)
{
    // [...] Inicjalizacja HAL, zegar w , GPIO, USART2, TIM5 itd.

```

```
    flag = 0;
    HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_1);

    while (1)
    {
        if (flag == 1)
        {
            HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

            printf("Wykryto zbocze po %d sekundach\r\n", period / 10000);
            flag = 0;
        }

        HAL_Delay(10);
    }
}
```

3.3 Wynik

Po podłączeniu pinu **PC13** (przycisk USER) do pinu **PA0**, program wykrywa zbocza opadające sygnału na pinie wejściowym. Po naciśnięciu przycisku na płytce, aktualny czas od ostatniego zbocza zostaje obliczony na podstawie wartości licznika i wyświetlony przez port szeregowy w formacie:

```
Wykryto zbocze po X sekundach
```

4 Zadanie 3: Generowanie sygnału PWM

4.1 Cel

Celem zadania było zapoznanie się z techniką modulacji szerokości impulsu (PWM) oraz zastosowaniem przerwań zewnętrznych (EXTI). Działanie PWM zilustrowano poprzez zmianę jasności diody LED podłączonej do pinu PA5, odpowiadającą zmianom wypełnienia sygnału PWM.

4.2 Konfiguracja

- **Płytka:** NUCLEO-L476RG
- **Licznik:** TIM2 skonfigurowany w trybie PWM Generation CH1.
- **Parametry licznika:**
  - **Prescaler (TIM2_PRESCALER):** 0
  - **Okres (TIM2_PERIOD):** 26666
- **Wyjście:** PA5 (TIM2_CH1) jako wyjście sygnału PWM.
- **Przerwania:** Skonfigurowano przerwanie zewnętrzne EXTI na pinie PC13 (przycisk użytkownika).

4.3 Kod programu

```c
int pwm_duty;
volatile int flag = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if (GPIO_Pin == GPIO_PIN_13) {
    flag++;
  }
}

int main(void)
{
  // [... inicjalizacja HAL, GPIO, TIM, UART ...]

  flag = 0;
  pwm_duty = 0;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

  while (1)
  {
    if (flag == 1)
    {
      pwm_duty = 0;
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
    }

    if (flag == 2)
    {
      pwm_duty = 26666 * 0.22;
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
    }

    if (flag == 3)
    {
      pwm_duty = 26666 * 0.65;
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
    }

    if (flag == 4)
    {
      pwm_duty = 26666;
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
    }

    if (flag >= 4)
    {
      flag = 0;
    }

    HAL_Delay(10);
  }
}
```

4.4 Wynik

Po każdym naciśnięciu przycisku użytkownika (PC13), zmieniało się wypełnienie sygnału PWM generowanego na pinie PA5, co przekładało się na wyraźną zmianę jasności diody LED. Ustawione wartości odpowiadały kolejno wypełnieniom: 0%, 22%, 65%, 100%. Po osiągnięciu 100% cykl rozpoczynał się od nowa.

5 Zadanie domowe: Pomiar wypełnienia sygnału PWM

5.1 Konfiguracja

- **Generator PWM (TIM3):**
  - **Tryb:** PWM Generation CH1
  - **Prescaler:** 720
  - **ARR (Auto-Reload Register):** 4000
  - **Częstotliwość:** około 25Hz
    (Obliczona według wzoru: $f = \frac{f_{clk}}{(PSC+1)\cdot(ARR+1)}$ przy $f_{clk} = 72\,\text{MHz}$)
  - **Kanał:** TIM3_CH1 (wyprowadzony na pin PA6)
- **Pomiar sygnału (TIM1):**
  - **Tryb:** Input Capture – pomiar czasu trwania impulsu wysokiego (HIGH)
  - **Prescaler:** 7200
  - **ARR:** 1000
  - **Rozdzielczość czasowa:** 100 ms (jeden overflow co ok. 100ms)
  - **Kanał:** TIM1_CH1 (podłączony do sygnału PWM z TIM3)
  - **Przerwania:** Włączone dla Input Capture i Update Event (przepełnienie)
- **Cel zadania:** Zmierzono czas trwania stanu wysokiego sygnału PWM (długość impulsu) oraz jego wypełnienie (duty cycle). Wyniki były wyświetlane przez port szeregowy UART.

5.2 Kod programu

```
// Zmienne globalne do przechowywania pomiar w
volatile uint32_t risingEdgeTime = 0;           // Czas zbocza narastaj cego
volatile uint32_t fallingEdgeTime = 0;          // Czas zbocza opadaj cego
volatile uint32_t overflowCount = 0;            // Licznik przepe nie  timera
volatile uint32_t overflowsRisingToFalling = 0; // Liczba przepe nie  mi dzy
    zboczem narastaj cym a opadaj cym
volatile uint32_t pulseDuration = 0;            // Czas trwania impulsu w
    cyklach timera
volatile uint8_t risingEdgeDetected = 0;        // Flaga wykrycia zbocza
    narastaj cego
volatile uint8_t measurementComplete = 0;       // Flaga zako czenia pomiaru

// Zmienne do oblicze
volatile float pwmDutyCycle = 0.0;      // Wype nienie PWM w %
volatile float pulseDurationMs = 0.0;   // Czas trwania impulsu w ms
volatile float pwmPeriodMs = 0.0;       // Okres sygna u PWM w ms
volatile uint8_t inputCaptureState = 0; // Stan pomiaru Input Capture (0-
    pocz tek , 1-zbocze narastaj ce , 2-zbocze opadaj ce)

// Zmienne do wy wietlania i zmiany wype nienia
static uint32_t lastTime_pwm = 0, lastTime_IC = 0;
static uint8_t pwmState = 0;
uint32_t currentTime = 0;

// [...]

// Callback wywo ywany przy rejestracji impulsu na kanale 1 (np. TIM1 CH1)
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
```

```
{
  if (htim->Instance == TIM1)
  {
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
    {
      // Timer 1 jest w trybie Input Capture w trybie Direct Mode -
      // zbocza narastaj ce i opadaj ce  s   rejestrowane na tym samym kanale

      // Sprawdzamy aktualny poziom sygna u , aby ustali  czy to zbocze
    narastaj ce czy opadaj ce
      if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8) == GPIO_PIN_SET)
      {
        // Zbocze narastaj ce
        risingEdgeTime = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
        overflowsRisingToFalling = 0;
        risingEdgeDetected = 1;
      }
      else
      {
        // Zbocze opadaj ce (tylko je li wcze niej by o narastaj ce)
        if (risingEdgeDetected)
        {
          fallingEdgeTime = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

          // Obliczanie czasu trwania impulsu
          if (fallingEdgeTime >= risingEdgeTime)
          {
            // Normalny przypadek - bez przepe nienia licznika mi dzy
    pomiarami
            pulseDuration = fallingEdgeTime - risingEdgeTime;
          }
          else
          {
            // Przypadek z przepe nieniem licznika mi dzy pomiarami
            pulseDuration = ((ARR_1 + 1) - risingEdgeTime) + fallingEdgeTime;
          }

          // Dodanie przepe nie  licznika kt re wyst pi y mi dzy zboczami
          pulseDuration += overflowsRisingToFalling * (ARR_1 + 1);

          // Resetowanie flagi zbocza narastaj cego i ustawienie flagi
    zako czenia pomiaru
          risingEdgeDetected = 0;
          measurementComplete = 1;
        }
      }
    }
  }
}

// Callback wywo ywany przy przepe nieniu licznika (np. TIM1 Update)
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM1)
  {
    overflowCount++;

    // Je li jeste my w trakcie pomiaru (po zboczu narastaj cym, przed
    opadaj cym)
    if (risingEdgeDetected)
    {
      overflowsRisingToFalling++;
    }
```

```
  }
}

// [...]

while (1)
  {
    currentTime = HAL_GetTick();

    if (currentTime - lastTime_IC >= 1000)
    {
      lastTime_IC = currentTime;

      if (measurementComplete)
      {
        // Obliczenie czasu trwania impulsu w milisekundach
        // Wz r: czas [ms] = (licznik * (prescaler+1)) / (cz stotliwo
    zegara [Hz] / 1000)
        float tickTime_us = ((prescaler_1 + 1) * 1000000.0) / SystemCoreClock;
    // Czas jednego tiku w mikrosekundach
        pulseDurationMs = (pulseDuration * tickTime_us) / 1000.0;
    // Konwersja us na ms

        // Obliczenie okresu sygna u PWM na TIM3 (teoretycznie)
        // Okres = (ARR_3 + 1) * (prescaler_3 + 1) / SystemCoreClock
        pwmPeriodMs = ((float)(ARR_3 + 1) * (prescaler_3 + 1)) / (
    SystemCoreClock / 1000.0);

        // Obliczenie wype nienia (duty cycle) PWM
        pwmDutyCycle = (pulseDurationMs / pwmPeriodMs) * 100.0;

        // Wy wietlenie wynik w
        printf("Czas impulsu: %.3f ms, Okres PWM: %.3f ms, Wypelnienie: %.2f%%\r
    \n",
               pulseDurationMs, pwmPeriodMs, pwmDutyCycle);

        // Resetowanie flagi zako czenia pomiaru
        measurementComplete = 0;
      }
      else
      {
        printf("Oczekiwanie na pomiar...\r\n");
      }
    }

    // Zmiana wype nienia PWM co 2 sekundy
    if (currentTime - lastTime_pwm >= 2000)
    {
      lastTime_pwm = currentTime;

      // Zmiana wype nienia PWM
      switch (pwmState)
      {
      case 0:
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, (ARR_3 * 10) / 100); // 10%
        printf("Ustawiono wypelnienie PWM: 10%%\r\n");
        pwmState = 1;
        break;
      case 1:
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, (ARR_3 * 40) / 100); // 40%
        printf("Ustawiono wypelnienie PWM: 40%%\r\n");
        pwmState = 2;
        break;
```

```
142         case 2:
143           __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, (ARR_3 * 80) / 100); // 80%
144           printf("Ustawiono wypelnienie PWM: 80%%\r\n");
145           pwmState = 0;
146           break;
147         }
148       }
149     }
```

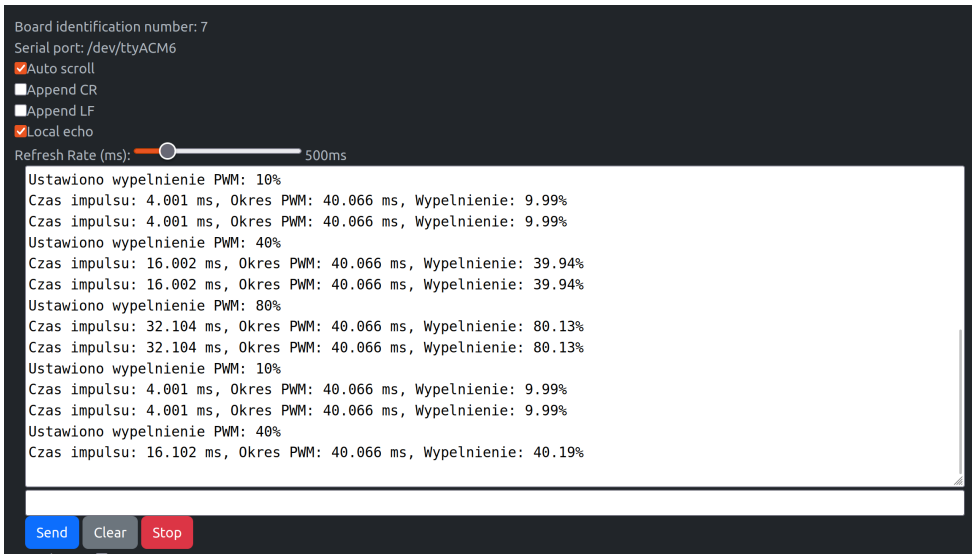
Program mierzył czas trwania impulsu i obliczał wypełnienie sygnału PWM, wyświetlając wyniki na porcie szeregowym.

5.3 Wynik

Program poprawnie mierzył czas trwania impulsu sygnału PWM oraz obliczał jego wypełnienie (duty cycle). Wyniki były prezentowane cyklicznie na porcie szeregowym co 1 sekundę, a wypełnienie sygnału PWM zmieniało się co 2 sekundy pomiędzy wartościami 10%, 40% oraz 80%.

Na podstawie wyświetlanych danych można było zauważyć:

- Zmierzony czas trwania impulsu odpowiadał rzeczywistej długości stanu wysokiego, np. dla 10% wypełnienia przy częstotliwości 25Hz (okres ok. 40ms), impuls trwał około 4ms.
- Wartość obliczonego wypełnienia (duty cycle) była zgodna z ustawioną wartością generatora PWM.
- Przepełnienia licznika były prawidłowo zliczane i uwzględniane w końcowych obliczeniach długości impulsu.



Rysunek 1: Output programu na Serial Port

6 Podsumowanie

W ramach przeprowadzonych ćwiczeń laboratoryjnych szczegółowo zapoznano się z działaniem oraz konfiguracją liczników (timerów) w mikrokontrolerach STM32. Szczególną uwagę poświęcono trzem podstawowym trybom pracy: trybowi generatora podstawy czasu (Time Base), trybowi pomiaru impulsów (Input Capture) oraz trybowi generowania sygnału PWM (Pulse Width Modulation).

Podczas zajęć praktycznych zrealizowano szereg eksperymentów, które umożliwiły:

- konfigurację timera do generowania przerwań w równych odstępach czasu,
- pomiar czasu trwania impulsu wejściowego przy użyciu trybu Input Capture, wraz z obsługą przepełnień licznika,
- generowanie sygnału PWM o określonym wypełnieniu i częstotliwości, a także jego dynamiczną zmianę,
- synchronizację dwóch timerów – jednego jako generatora PWM, drugiego jako analizatora tego sygnału.

Szczególnie cennym doświadczeniem było wykonanie zadania domowego, polegającego na skonfigurowaniu jednego timera jako generatora sygnału PWM o częstotliwości 25Hz, oraz drugiego jako odbiornika w trybie Input Capture, z tak dobranym preskalerem, aby zapewnić dokładność pomiaru rzędu 100s. Pozwoliło to na dokładny pomiar czasu trwania stanu wysokiego sygnału oraz obliczenie wypełnienia PWM w czasie rzeczywistym.

Dużym wyzwaniem okazała się prawidłowa konfiguracja timerów, właściwe wykrywanie zboczy, a także weryfikacja połączeń pinów na płytce Nucleo oraz przypisanie ich w narzędziu konfigurującym sprzęt (STM32CubeMX). Doświadczenie to pokazało, jak istotne jest dokładne sprawdzenie ustawień sprzętowych i funkcji pinów przed implementacją oraz debugowaniem kodu.

Wnioskiem na przyszłość jest konieczność wcześniejszego upewnienia się co do poprawności konfiguracji mikrokontrolera w środowisku CubeMX – szczególnie w kontekście mapowania pinów, preskalerów, wyboru kanałów i źródeł zegara – zanim przystąpi się do implementacji logiki programu. Takie podejście pozwala zaoszczędzić czas i uniknąć trudnych do wykrycia błędów.