POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

SPRAWOZDANIE

SYSTEMY MIKROPROCESOROWE (LABORATORIUM)
[WARIE_2021-22_AIR_DZ_1_5_D_LUCZAK_21/22]

STEROWANIE MODULACJĄ SZEROKOŚCI IMPULSÓW
(PWM, LED RGB)

(TEMAT ZAJĘĆ)

KAROL DĘBSKI

(AUTOR I: KAROL.DEBSKI@STUDENT.PUT.POZNAN.PL)

FORMA ZAJĘĆ: LABORATORIUM

PROWADZĄCY:

DR INŻ. DOMINIK ŁUCZAK

DOMINIK.LUCZAK@PUT.POZNAN.PL

POZNAŃ *8-11-2021* 9-45

(*DATA I GODZINA ZAJĘĆ*)

Spis treści

Zadanie #1

1.1 Specyfikacja

Program zadaje moc LED za pomocą wypełnienia sygnału PWM. Do weryfikacji poprawności działania kodu zostanie zmierzony prąd dla wypełnienia 50% i 100%.

Użyte elementy elektroniczne: 3x rezystor 470 Ω, dioda RGB z wspólną katodą

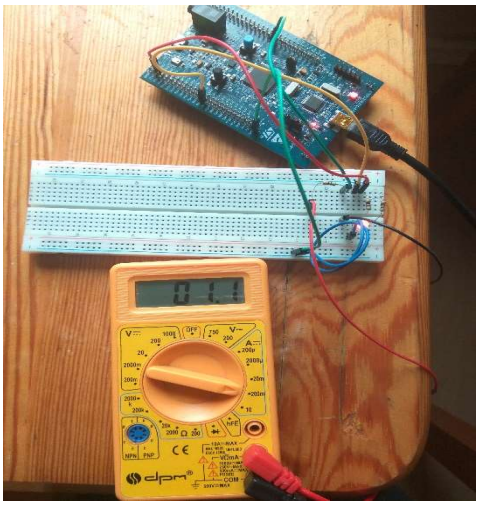
1.2 Implementacja

Listing 1 Inicjalizacja timera w trybie generowania sygnału PWM

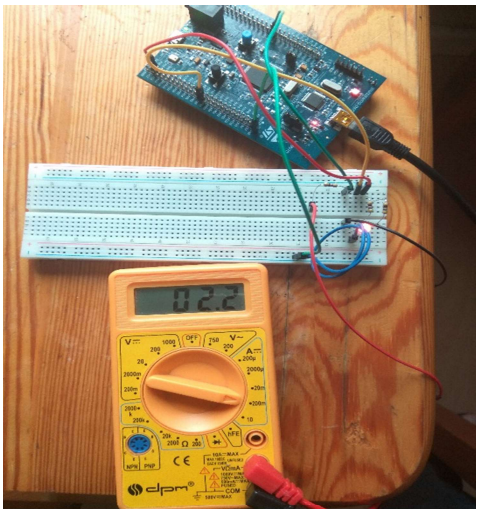
```
/* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
/* USER CODE END 2 */
```

Poniższe linijki kodu inicjalizują 3 kanały timera do generowania sygnału PWM.

1.3 Wynik testów



Rys. 1 Pomiar prądu dla wypełnienia 50%



Rys. 2 Pomiar prądu dla wypełnienia 100%

1.4 Wnioski

Prąd dla wypełnienia 50% jest mniejszy o połowe w stosunku do wypełnienia 100% co prowadzi do wniosku że timer zadaje sygnał PWM poprawnie.

Zadanie #2

1.1 Specyfikacja

Program będzie zmieniał wypełnienie LED RGB od 0% do 100% z krokiem 10% co 0.5 sekundy. Do sprawdzenia poprawnego działania programu zostanie zamieszczone nagranie video.

1.2 Implementacja

Listing 2 Główna pętla programu

```
  while (1)
  {
      HAL TIM SET COMPARE(&htim3,TIM CHANNEL 1,duty);
    __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_2,duty);
    __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,duty);
    /* USER CODE END WHILE */
    MX USB HOST Process();

    /* USER CODE BEGIN 3 */
  }
```

Listing 3 Funkcja zwrotna dla przerwania na TIM10

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
          if(htim->Instance == TIM10){
                    if(duty==1000){
                              duty=0;
                    }else{
                              duty+=100;
                    }
          }
}
```

Po naliczeniu 0,5 sekundy uruchamia się przerwanie na timerze. Zmienna duty jest zwiększana o 100 od 0 do 1000. Po osiągnięciu wartości 1000 zmienna jest zerowana. W pętli głównej programu zmieniane jest wypełnienie na podstawie zmiennej duty.

1.3 Wynik testów

https://drive.google.com/file/d/1rkNCyhtAYz7ECI-gcltXHhU8RG2ssFNK/view?usp=sharing (8 sekund)

1.4 Wnioski

Na filmie widać że prąd diody się zwiększa co 0,5 sekundy a następnie osiąga zero po upłynięciu 5 sekund.

Zadanie #3

1.1 Specyfikacja

Program po dostaniu komendy z UART'a zmienia wypełnienie poszczególnych diod RGB. Do sprawdzenia poprawnego działania programu zostanie zamieszczone nagranie video.

1.2 Implementacja

Listing 4 Funkcja zwrotna dla przerwania na UART2

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
          if(huart->Instance==USART2){
                    sscanf(msg,"%c%3d",&color,&duty);
                    switch (color){
                    case 'R'://PB0
                              HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,duty);
                              break;
                    case 'G':
                              __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_2,duty);
                              break;
                    case 'B':
                              HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_1,duty);
                              break;
                    }
          HAL_UART_Receive_IT(&huart2, msg, 4);
          }
}
```

Gdy nadejdzie przerwanie analizowany jest tekst który został wysłany. Pierwszy symbol to oznaczenie koloru diody a 3 kolejne cyfry to stopień wypełnienia sygnału PWM.

1.3 Wynik testów

https://drive.google.com/file/d/1sZRlljqtY7LSmxH22ZMwVh6QnqWbXNzL/view?usp=sharing (12 sekund)

1.4 Wnioski

Na filmie odpowiednie diody się załączają po wysłaniu komendy z terminala. Program działa poprawnie.

Zadanie #4

1.1 Specyfikacja

Program odbiera 12 bajtowe dane które po zdekodowaniu ustawiają wypełnienie dla diod RGB. Do sprawdzenia poprawnego działania programu zostanie zamieszczone nagranie video.

1.2 Implementacja

Listing 6 Funkcja zwrotna dla USART2

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
          if(huart->Instance==USART2){

          sscanf(msg,"%c%3d%c%3d%c%3d",&color1,&duty1,&color2,&duty2,&color3,&duty3);
                    set_duty(color1, duty1);
                    set duty(color2, duty2);
                    set duty(color3, duty3);
          HAL UART Receive IT(&huart2, msg, 12);
          }
}
```

Listing 7 Funkcja ustawiająca wypełnienie dla diody

```
void set_duty(uint8_t color,uint32_t duty){
          switch (color){
                    case 'R'://PB0
                              HAL TIM SET COMPARE(&htim3,TIM CHANNEL 3,duty);
                              break;
                    case 'G':
                              __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_2,duty);
                              break;
                    case 'B':
                              HAL TIM SET COMPARE(&htim3,TIM CHANNEL 1,duty);
                              break;
          }
}
```

Po wywołaniu przerwania na USART2 analizowany jest odebrany tekst. Po jego zdekodowaniu wywoływane są funkcje ustawiające wypełnienie dla poszczególnych diod.

1.3 Wynik testów

https://drive.google.com/file/d/1sdNPtGRaeTT7KLT_Ryd2ifT9S6aK-oCi/view?usp=sharing

(10 sekund)

1.4 Wnioski

Na filmie dioda RGB reaguje na komendy przesyłane przez terminal. Program działa poprawnie.

Zadanie #5

1.1 Specyfikacja

Program zlicza sygnały z impulsatora. Do weryfikacji poprawnego działania programu zostanie użyty SWV.

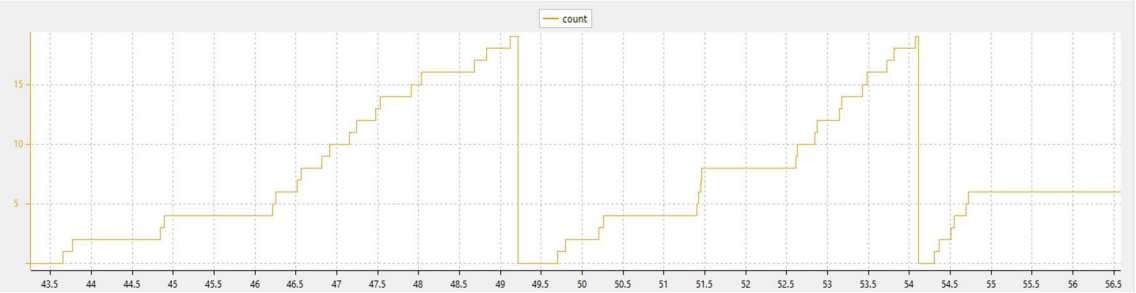
1.2 Implementacja

Listing 6 Główna pętla programu

```
while (1)
{
  count=  HAL_TIM_GET_COUNTER(&htim3);
  HAL_Delay(10);
  /* USER CODE END WHILE */
  MX_USB_HOST_Process();

  /* USER CODE BEGIN 3 */
}
```

Przy każdym przebiegu pętli while pobierana jest wartość ilości kroków o jakie został przekręcony impulsator od momentu inicjalizacji timera w trybie „encoder mode". Po każdym sprawdzeniu zostało dodane opóźnienie wynoszące 10 ms by SWV działał poprawnie.

1.3 Wynik testów



Rys 3 Podgląd SWV

1.4 Wnioski

Na wykresie wartość zmiennej zmienia się w zakresie od 0 do 19 co świadczy o poprawnym działaniu impulsatora który ma 20 kroków na obrót.

Zadanie #6

1.1 Specyfikacja

Program co 1 sekunda wysyła ilość kroków o jaką została przekręcona gałka impulsatora. Do weryfikacji poprawności działania programu zostanie użyty UART a także SWV.

1.2 Implementacja

Listing 6 Główna pętla programu

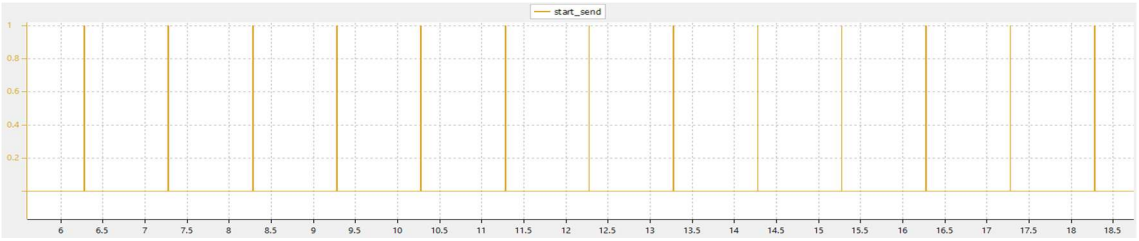
```
  while (1)
  {
          count=  HAL_TIM_GET_COUNTER(&htim3);
          sprintf((uint8_t *)msg,"%d\r\n", count);
          if(start_send){
                    HAL_UART_Transmit(&huart2, msg, strlen(msg),1000);
                    start_send=0;
          }
          HAL_Delay(10);
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

    /* USER CODE BEGIN 3 */
  }
```

Listing 7 Funkcja zwrotna dla TIM10

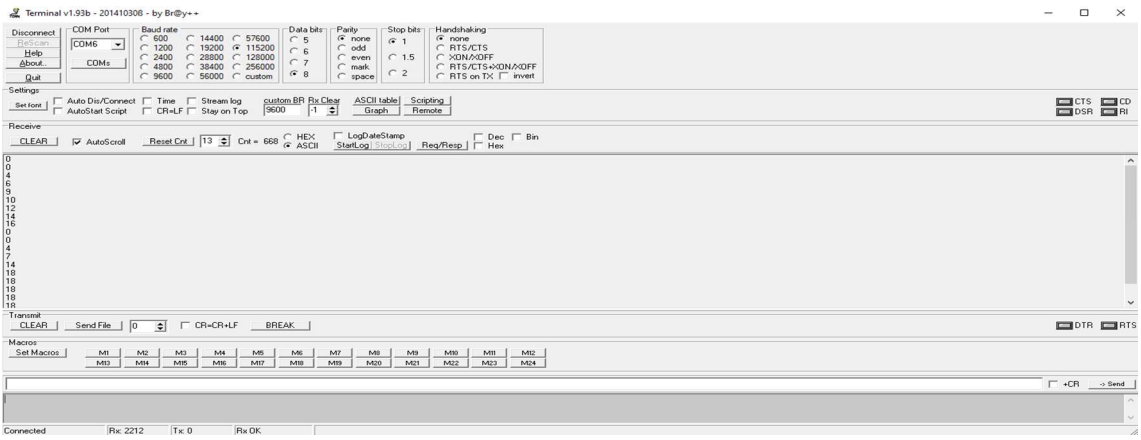
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
          if(htim->Instance==TIM10){
                    start_send=1;
          }
}
```

Przy każdej iteracji pętli while odczytywana jest liczba kroków o jaką został przekręcony impulsator. Gdy zostanie wywołane przerwanie na TIM10, wysyłana jest wiadomość o liczbie kroków o jakie został przekręcony impulsator.

1.3 Wynik testów



Rys 4 Podgląd SWV



Rys 5 Podgląd terminala na PC

1.4 Wnioski

SWV potwierdza, że wiadomość jest wysyłana co jedną sekundę a terminal na PC informuje że wiadomości zostają odebrane. Program działa poprawnie.

Podsumowanie

SWV nie nadaje się w roli oscyloskopu, potrzebuje co najmniej 10 ms by pobrać wartość danej zmiennej. Przy opóźnieniu poniżej 10 ms pojawiają się przekłamania.