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]

INTERFEJS KOMUNIKACYJNY SPI (CYFROWY CZUJNIK TEMPERATURY/CIŚNIENIA, BIBLIOTEKI PRODUCENTA)

(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 22-*11-2021* 9-45

(*DATA I GODZINA ZAJĘĆ*)

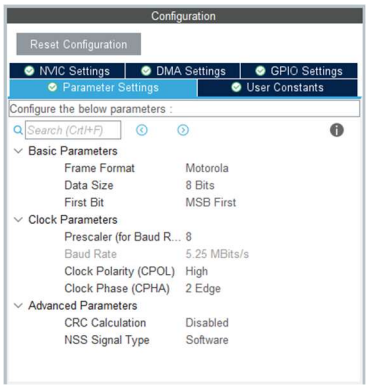
Spis treści

Zadanie #1

1.1 Specyfikacja

Program co 1 sekundę odczytuje pomiar z czujnika i przesyła go na terminal. Do zweryfikowania poprawnego działania programu zostanie użyty terminal.

1.2 Implementacja



Rys. 1 Konfiguracja SPI

Listing 1 Funkcja zapisu

```
int8_t user_spi_write(uint8_t reg_addr, uint8_t *reg_data, uint32_t length, void
*intf_ptr)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */
    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi2, ® addr, 1, 500);
    HAL_SPI_Transmit(&hspi2, reg data, length, 5000);
    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin, GPIO_PIN_SET);
          if(rslt == HAL_OK)rslt=0; else rslt=1;
    return rslt;
}
```

Listing 2 Funkcja odczytu

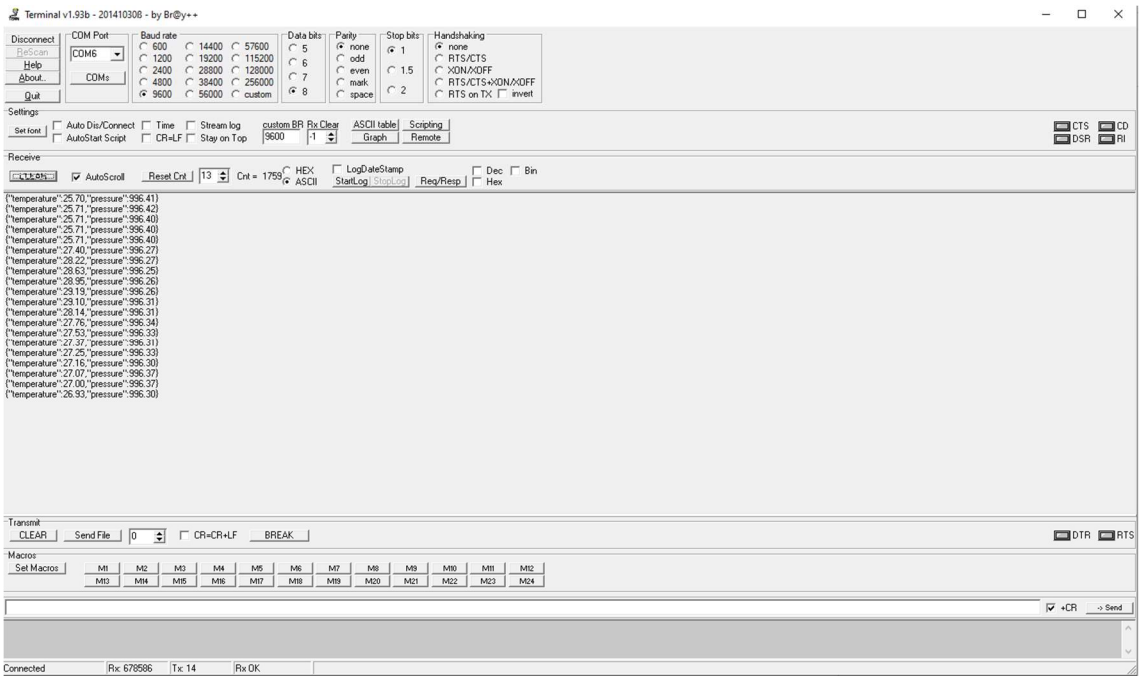
```
int8_t user_spi_read(uint8_t reg_addr, uint8_t *reg_data, uint32_t length, void
*intf_ptr)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */
    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi2, &reg_addr, 1, 500);
    HAL_SPI_Receive(&hspi2, reg_data, length, 5000);
    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin, GPIO_PIN_SET);
          if(rslt == HAL_OK)rslt=0; else rslt=1;
    return rslt;
}
```

Listing 3 Główna pętla programu

```
while (1)
  {
          BMP2_user_app_stream_sensor_data_by_uart(&dev);
          HAL_Delay(1000);
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

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

Odczyt temperatury i ciśnienia jest przesyłany na uart a następnie wprowadzone jest opóźnienie 1000 ms.

1.3 Wynik testów



Rys. 2 Terminal

1.4 Wnioski

Odczyty temperatury i ciśnienia są prawidłowe i przesyłane są na terminal.

Zadanie #2

1.1 Specyfikacja

Na komendę wysłaną z terminalu program zmienia częstotliwość wykonywania pomiaru z czujnika. Do weryfikacji poprawnego działania zostanie nagrane video terminalu.

1.2 Implementacja

Listing 2 Główna pętla programu

```
while (1)
  {
    if(start_send){
          BMP2_user_app_stream_sensor_data_by_uart(&dev);
          start_send=0;
    }
      HAL TIM SET AUTORELOAD(&htim10,ARR);
    /* USER CODE END WHILE */
    MX USB HOST Process();

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

Listing 3 Funkcja zwrotna dla przerwania na USART

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
          if(huart->Instance == USART2){
                    if(sscanf(bufforRx,"freq=%d",&tmp_int) && tmp_int>0 && tmp_int <10){
                              ARR=(uint16 t)((10000/tmp int)-1);
                    }
          }
          HAL_UARTEx_ReceiveToIdle_IT(&huart2, bufforRx, BUFFOR_RX_SIZE);
}
```

Listing 4 Funkcja zwrotna dla przerwania na timerze

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

W momencie wywołania przerwania na uarcie analizowana jest komenda. Na podstawie liczby zawartej komendzie ustawiana jest zmienna ARR która to w pętli głównej będzie ustawiać wartość rejestru AutoReload timera tak by funkcja zwrotna timera była uruchamiana z zadaną częstotliwością. Funkcja zwrotna timera ustawia flagę do wywoływania funkcji wysyłającej przez uart dane z czujnika w pętli głównej programu.

1.3 Wynik testów

https://drive.google.com/file/d/1dD6xGgGTdKEh8kfh2nD5NcFMYMn6SygP/view?usp=sharing

(19 sekund)

1.4 Wnioski

Odczyty wysyłane są z zadaną częstotliwością, program działa poprawnie.

Zadanie #3

1.1 Specyfikacja

Program na komendę print_on będzie wysyłał na terminal pomiar z czujnika. Na komendę print_off zaprzestanie wysyłania pomiarów. Gdy użytkownik wprowadzi komendę logger=liczba to program wyśle na terminal tablice JSON z liczbą pomiarów z czujnika wskazana w komendzie. Dane z czujnika będą pobierane z częstotliwością 8 Hz. Do weryfikacji poprawnego działania programu zostanie zamieszczony link do pliku video.

1.2 Implementacja

Listing 3 Funkcj zwrotna dla przerwania na USART

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
          if(huart->Instance == USART2){
                    sscanf(bufforRx,"%5c%d",&tmp_string,&tmp_int);
                    if(!strcmp(tmp_string,"freq=") && tmp_int>0 && tmp_int <10){
                              ARR=(uint16 t)((10000/tmp int)-1);
                    }else{
                              memset(tmp string, 0, 20);
                              tmp_int=0;
                    }
                    sscanf(bufforRx,"%7c%d",&tmp string,&logger);
                    if(!strcmp(tmp_string,"logger=") && logger>0 &&
logger<MAX DATA SIZE){
                              start_collectData=1;
                              starting_index=0;
                              print on=0;
                    }else{
                              memset(tmp string, 0, 20);
                    }
                    if(!strcmp(bufforRx,"print_on")){
                              print_on=1;
                    }
                    if(!strcmp(bufforRx,"print_off")){
                              print on=0;
                    }
          }
          memset(bufforRx, 0, BUFFOR_RX_SIZE);
          HAL UARTEx ReceiveToIdle IT(&huart2, bufforRx, BUFFOR RX SIZE);
}
```

Listing 4 Funkcja zwrotna dla przerwania na timerze

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

Listing 5 Główna pętla programu

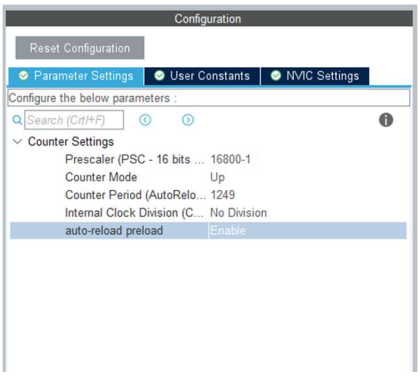
```
while (1)
  {
          if(period_elapsed && print_on){
                    BMP2 user app stream sensor data by uart(&dev);
                    period elapsed=0;
          }
          if(period_elapsed && start_collectData){
                    collectData();
                    period_elapsed=0;
          }
          if(end_of_collectData){
                    createJSON(data, starting index);
                    end of collectData=0;
          }

          //  HAL TIM SET AUTORELOAD(&htim10,1249);
    /* USER CODE END WHILE */
    MX USB HOST Process();

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

Listing 6 Funkcja collectData

```
void collectData(){
                    memset(tmp_string, 0, 20);
                    if(counter<logger){
                              bmp2_get_sensor_data(&(data[counter]), &dev);
                              counter++;
                    }else{
                              end_of_collectData=1;
                              counter=0;
                    }
}
```

Listing 7 Funkcja createJSON

```
void createJSON(struct bmp2_data *data,uint32_t starting_index){
                    uint16_t i=starting_index;
                    strcat(data_json,"[[");
                    for(i;i<logger;i++){
                              sprintf(tmp_string,"%0.1f",data[i].temperature);
                              strcat(data_json,tmp_string);
                              memset(tmp_string, 0, 20);
                              if(!(i==logger-1)) strcat(data_json,",");
                    }
                    strcat(data_json,"],[");
                    i=starting_index;
                    for(i;i<logger;i++){
                              sprintf(tmp_string,"%0.2f",data[i].pressure*0.01);
                              strcat(data_json,tmp_string);
                              memset(tmp_string, 0, 20);
                              if(!(i==logger-1)) strcat(data_json,",");
                    }
                    strcat(data_json,"]]");
                    HAL_UART_Transmit(&huart2, data_json, strlen(data_json), 1000);
                    memset(data_json, 0, MAX_JSON_SIZE);
                    start_collectData=0;
}
```



Rys. 3 Konfiguracja timera

Po wywołaniu przerwania na usarcie wykonywana jest funkcja zwrotna gdzie dokonywana jest analiza odebranej komendy i ustawiane są flagi. W pętli głównej znajduje się kod odpowiadający za włączenie cyklicznego przesyłania danych a także wywołania funkcji do zapisywania odczytów z czujnika do tablicy. Dalej jest funkcja która na podstawie tablicy odczytów tworzy format zapisu danych JSON i wysyła go na usart.

1.3 Wynik testów

https://drive.google.com/file/d/1MluwGIQ3vwseuAjVgRWsql86lzSgXn30/view?usp=sharing (21 sekund)

1.4 Wnioski

Program odpowiednio reaguje na 3 komendy.

Zadanie #4

1.1 Specyfikacja

Program wysyła na terminal 3 ostatnie odczyty dla temperatury i ciśnienia z ostatniej serii pomiaru, jeśli takowej serii nie ma to program zwraca zera.

1.2 Implementacja

Listing 8 Funkcja zwrotna dla przerwania na USART

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
	if(huart->Instance == USART2){
		sscanf(bufforRx,"%5c%d",&tmp string,&tmp int);
		if(!strcmp(tmp_string,"freq=") && tmp_int>0 && tmp_int <10){
			ARR=(uint16_t)((10000/tmp_int)-1);
		}else{
			memset(tmp string, 0, 20);
			tmp int=0;
		}
		sscanf(bufforRx,"%7c%d",&tmp_string,&logger);
		if(!strcmp(tmp_string,"logger=") && logger>0 &&
logger<MAX_DATA_SIZE){
			start collectData=1;
			starting index=0;
			print on=0;
		}else{
			memset(tmp_string, 0, 20);
		}
		sscanf(bufforRx,"%12c%d",&tmp string,&logger last);
		if(!strcmp(tmp_string,"logger_last=") && logger_last>0 &&
logger_last<MAX_DATA_SIZE){
			if(logger==0){
				starting_index=0;
				logger=logger last-1;
			}else{
				starting index=logger-logger last;
			}
			print_on=0;
		}else{
			memset(tmp string, 0, 20);
			logger last=0;
		}
		if(!strcmp(bufforRx,"print_on")){
			print_on=1;
		}
		if(!strcmp(bufforRx,"print_off")){
			print on=0;
		}
	}
	memset(bufforRx, 0, BUFFOR_RX_SIZE);
	HAL_UARTEx_ReceiveToIdle_IT(&huart2, bufforRx, BUFFOR_RX_SIZE);
}
```

Listing 9 Funkcja zwrotna dla przerwania na timerze

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

Listing 10 Główna pętla programu

```
while (1)
  {
	if(period_elapsed && print_on){
		BMP2_user_app_stream_sensor_data_by_uart(&dev);
		period elapsed=0;
	}
	if(period_elapsed && start_collectData){
		collectData();
		period_elapsed=0;
	}
	if(end_of_collectData || logger_last>0){
		createJSON(data, starting index);
		end_of_collectData=0;
		logger_last=0;
```

```
        }

        //  HAL TIM SET AUTORELOAD(&htim10,1249);
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

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

Listing 11 Funkcja collectData

```
void collectData(){
                    memset(tmp string, 0, 20);
                    if(counter<logger){
                              bmp2_get_sensor_data(&(data[counter]), &dev);
                              counter++;
                    }else{
                              end of collectData=1;
                              counter=0;
                    }
}
```

Listing 12 Funkcja createJSON

```
void createJSON(struct bmp2_data *data,uint32_t starting_index){
                    uint16_t i=starting_index;
                    strcat(data_json,"[[");
                    for(i;i<logger;i++){
                              sprintf(tmp string,"%0.1f",data[i].temperature);
                              strcat(data json,tmp string);
                              memset(tmp string, 0, 20);
                              if(!(i==logger-1)) strcat(data_json,",");
                    }
                    strcat(data json,"],[");
                    i=starting index;
                    for(i;i<logger;i++){
                              sprintf(tmp string,"%0.2f",data[i].pressure*0.01);
                              strcat(data_json,tmp_string);
                              memset(tmp_string, 0, 20);
                              if(!(i==logger-1)) strcat(data_json,",");
                    }
                    strcat(data json,"]]");
                    HAL_UART_Transmit(&huart2, data_json, strlen(data_json), 1000);
                    memset(data_json, 0, MAX_JSON_SIZE);
                    start collectData=0;
}
```

Po wywołaniu przerwania na usarcie wykonywana jest funkcja zwrotna gdzie dokonywana jest analiza odebranej komendy i ustawiane są flagi. W pętli głównej znajduje się kod do zbierania odczytów z czujnika i kod do tworzenia tablicy JSON z argumentem starting_index. Ten argument wskazuje index tablicy danych od którego ma być tworzona tablica JSON. Jeśli tablica danych nie została zapełniona to po wywołaniu logger_last=”x”, w funkcji zwrotnej dla usart, starting_index jest ustawiany na 0 a logger na x-1 by nie przekroczyć ilości liczby próbek x.

1.3 Wynik testów

https://drive.google.com/file/d/1ghHDaHAUTeLqCxAzQZsjbv7QKYe2vpEk/view?usp=sharing

(27 sekund)

1.4 Wnioski

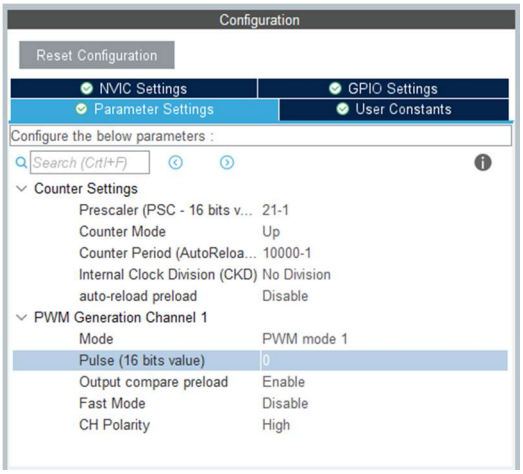
Program reaguje poprawnie na zadane komendy.

Zadanie #5

1.1 Specyfikacja

Program po otrzymaniu komendy tekstowej zmienia wypełnienie sygnału timera. Sygnał ten dociera do bramki tranzystora MOSFET by dalej sterować większym prądem w obwodzie z rezystorem cementowym. Do weryfikacji poprawnego działania programu będzie mierzone napięcie na rezystorze mocy po zadaniu komendy z terminala.

1.2 Implementacja



Rys. 4 Konfiguracja timera

Listing 13 Funkcja zwrotna dla przerwania na USART

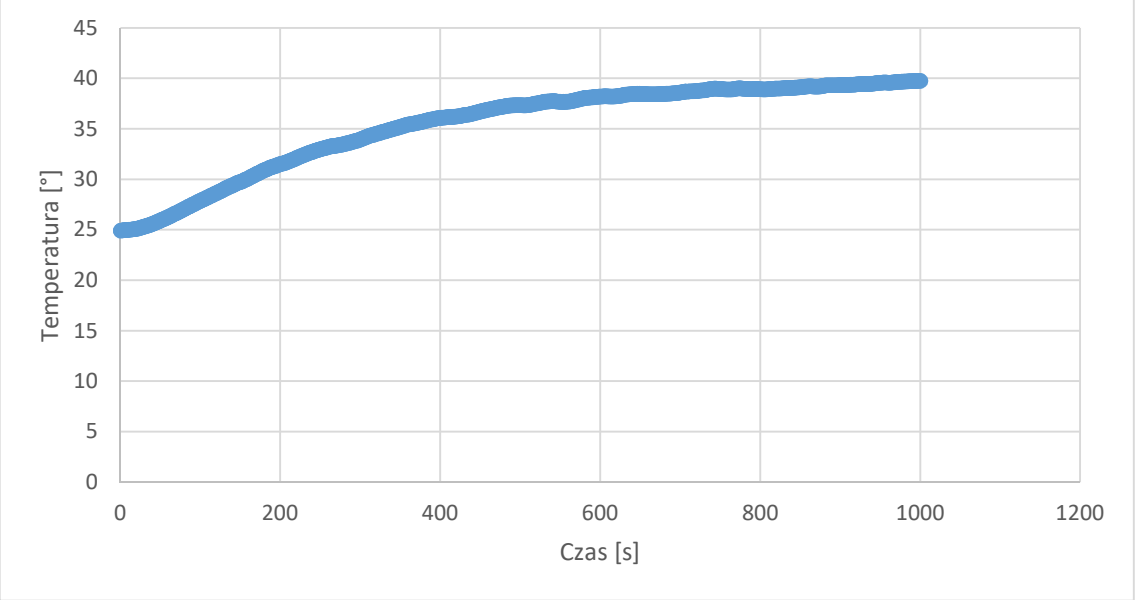
```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
          if(huart->Instance == USART2){
                    sscanf(bufforRx,"%5c%f",&tmp_string,&duty);
                    if(!strcmp(tmp_string,"PWM1=") && duty>=1 && duty<=100){
                              uint16_t CCRx=(uint16_t)(duty*68);
                              if(CCRx>=0 && CCRx<=6800){
                                        HAL_TIM_SET_COMPARE(&htim10,TIM_CHANNEL_1,CCRx);
                              }
                    }
          }
          memset(bufforRx, 0, BUFFOR_RX_SIZE);
          HAL_UARTEx_ReceiveToIdle_IT(&huart2, bufforRx, BUFFOR_RX_SIZE);
}
```

Listing 14 Główna pętla programu

```
while (1)
  {
    BMP2_user_app_stream_sensor_data_by_uart(&dev);
    HAL_Delay(1000);
    time++;
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

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

Po wykryciu przerwania na USART wykonywana jest funkcja zwrotna która analizuje odebraną komendę z terminala. Liczba skojarzona z wypełnieniem jest wykorzystywana do ustawienia wartości rejestru CCRx timera. CCRx nie może przekroczyć wartości 6800 ponieważ wartość 6800 oznacza generowanie sygnału PWM o wartości skutecznej równej 2 V a to maksymalne dopuszczalne napięcie bramki tranzystora MOSFET. W pętli głównej dokonywany jest pomiar temperatury co 1000 ms z inkrementacją zmiennej time do wykreślenia zależności odczytanej temperatury z czujnika od czasu.

1.3 Wynik testów



Rys. 5 Wykres zależności temperatury od czasu dla wypełnienia 90%

https://drive.google.com/file/d/1uNb_LunRKjg22JCP_94czqz5i8wXa-D9/view?usp=sharing

(23 sekundy)

1.4 Wnioski

Po wysłaniu komendy zmieniającej wypełnienie, napięcie na rezystorze mocy zmienia się. Oznacza to, że przewodność tranzystora MOSFET się też zmienia.

Podsumowanie

Warto przed rozpoczęciem pisania kodu narysować sobie schemat blokowy programu. W zadaniu 5, timer został skonfigurowany do pracy z częstotliwością 800 Hz, wynikało to z konieczności przyjęcia dużej wartości rejestru ARR by zachować zadawanie wypełnienia z krokiem 0.1%.