

Informacje o wykonawcach

Imię	Nazwisko	Numer albumu
Konrad	Krupski	310 729
Julia	Polak	310 965

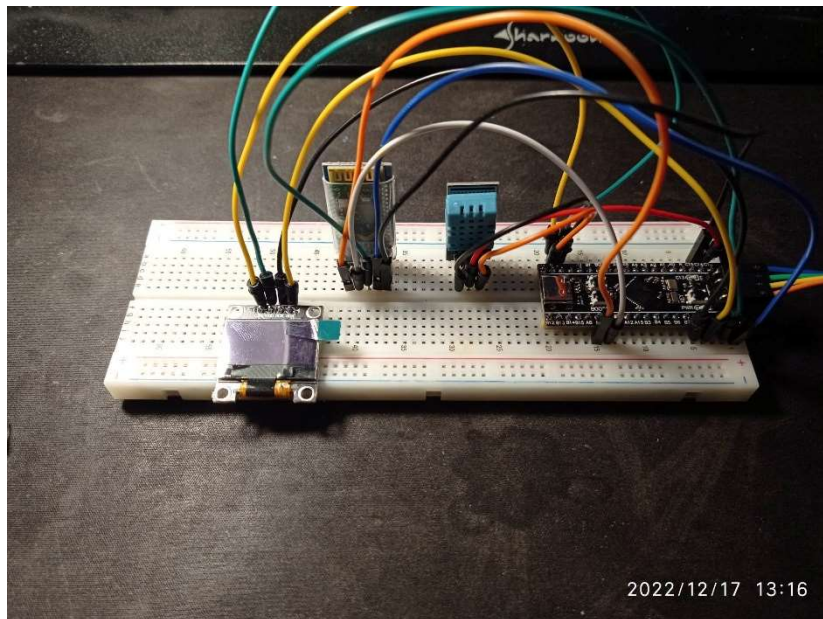
Opis projektu

Celem projektu jest zbudowanie układu służącego do pomiaru temperatury i wilgotności otoczenia. Układ będzie działał w temperaturach otoczenia od 0°C do 50 °C oraz dla wilgotności od 20% do 90%. Dodatkową funkcjonalnością systemu będzie możliwość zdalnego połączenia się z urządzeniem mobilnym np. telefonem poprzez interfejs bluetooth. Przy użyciu aplikacji na system android będziemy w stanie kontrolować działanie czujnika poprzez wysyłanie odpowiednich komend. W celach jest zrealizowanie komend do wyłączenia i włączenia czujnika oraz do wysyłania informacji, które dane urządzenie będzie pobierało.

Spis użytych urządzeń

Lp.	Nazwa urządzenia	Zasilanie	Dokumentacja
1	Płytką Blackpill z procesorem STM32F401CCU6	3.3V	LINK
2	Czujnik DHT11	3.3V	LINK
3	Programator/debugger do modułów z procesorami	-	-
4	Płytką stykową z modułem zasilającym i przewodami połączeniowymi	-	-
5	Moduł Bluetooth HC-05	3.3V	LINK
6	Wyświetlacz OLED SSD1306	3.3V	LINK

Zbudowany układ



Rysunek 1 Zbudowany układ

Czujnik DH11 został podłączony do napięcia 3.3V oraz jego wyjście zostało podłączone do pinu C13. Pin ten jest połączony z diodą płytki, dlatego łatwo będzie można zobaczyć kiedy czujnik pobiera i przetwarza informację o temperaturze i wilgotności powietrza.

Moduł Bluetooth HC-05 został podłączony do napięcia 3.3V oraz jego porty zostały podłączone w następujący sposób:

- RX -> A9
- TX -> A10
- EN -> C14
- STATE – port ten nie został podłączony, ponieważ w tym projekcie nie była używana jego funkcjonalność

Ekran OLED SSD1306 został podłączony do napięcia 3.3 V oraz jego porty zostały podłączone w następujący sposób:

- SCK -> B8
- SDA -> B9

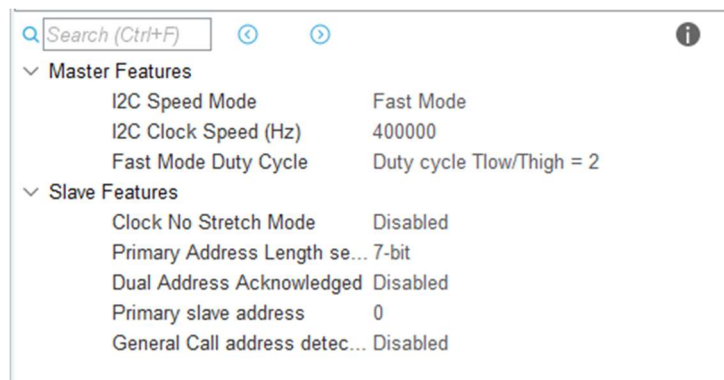
Do programowania płytki został użyty programator ST-LINK V2

Konfiguracja CUBE-IDE

Na początku utworzono projekt postępując według standardowej procedury. Wybrano odpowiedni procesor oraz zdecydowano się na niewprowadzanie domyślnej konfiguracji płytki zadanej przez producenta.

Konfiguracja I²C

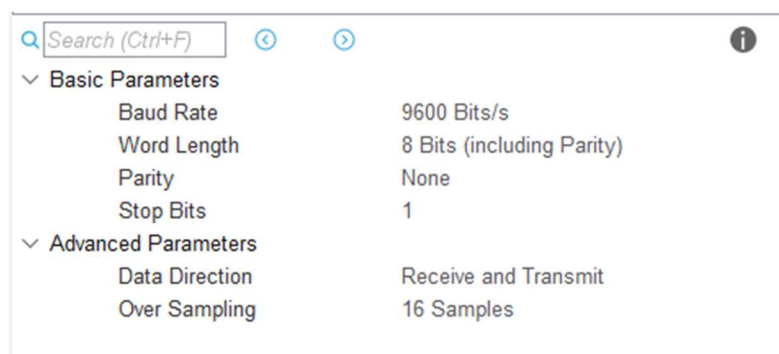
1. Uruchamiamy interfejs I²C1 korzystając z zakładki Connectivity
2. Ustawiamy parametr Speed Mode -> Fast Mode
3. Upewniamy się, że częstotliwość zegara wynosi 400 kHz



Rysunek 2 Parametry I2C

Konfiguracja USART

1. Uruchamiamy interfejs USART1 korzystając z zakładki Connectivity
2. Tryb Asynchronus
3. W ustawieniach parametrów zmieniamy Baud Rate na 9600 Bits/s



Rysunek 3 Parametry UASRT

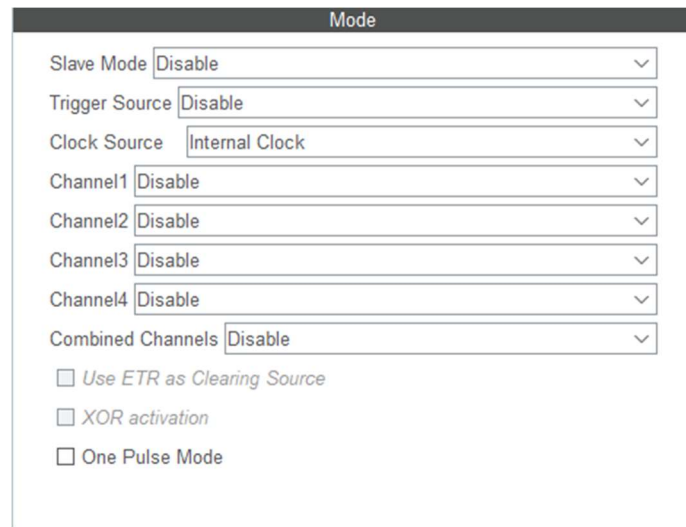
4. Uruchamiamy przerwania typu global

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	<input checked="" type="checkbox"/>	0	0

Rysunek 4 Uruchomienie przerwań UASRT

Konfiguracja Timer'a

1. Uruchamiamy TIM2 korzystając z zakładki Timers
2. Wybieramy Clock Source -> Internal Clock, czyli wyzwalanie zegarem wewnętrznym



Mode

Slave Mode Disable

Trigger Source Disable

Clock Source Internal Clock

Channel1 Disable

Channel2 Disable

Channel3 Disable

Channel4 Disable

Combined Channels Disable

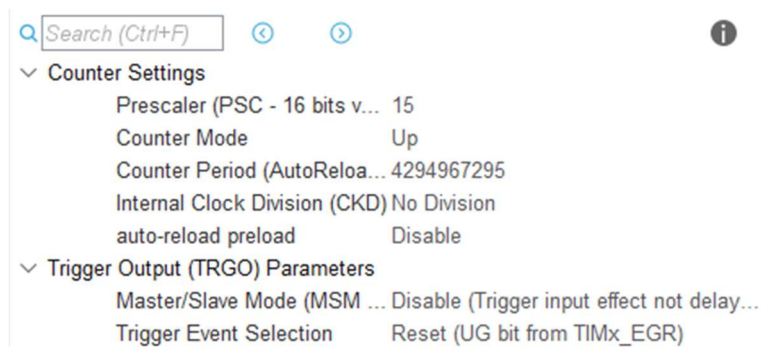
☐ Use ETR as Clearing Source

☐ XOR activation

☐ One Pulse Mode

Rysunek 5 Ustawienia TIM2

3. Ustawiamy częstotliwość zegara na 1 MHz poprzez ustawienie parametru Prescaler -> 15



Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits v... 15

Counter Mode Up

Counter Period (AutoReloa... 4294967295

Internal Clock Division (CKD) No Division

auto-reload preload Disable

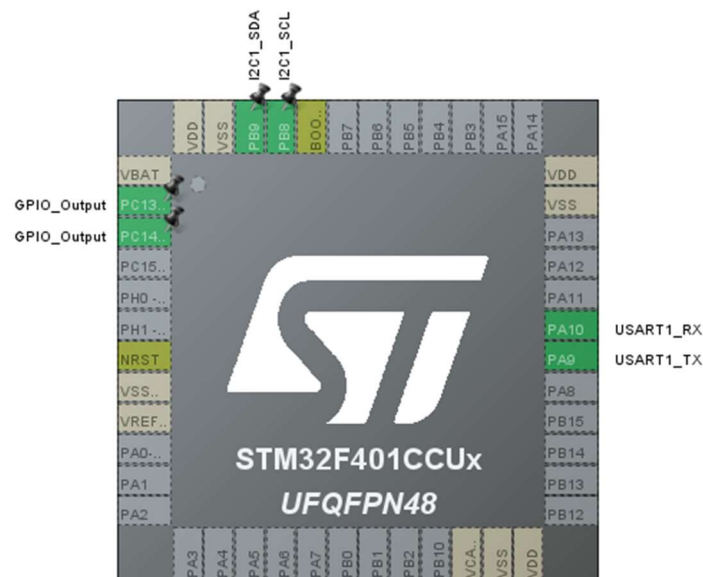
Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM ... Disable (Trigger input effect not delay...

Trigger Event Selection Reset (UG bit from TIMx_EGR)

Rysunek 6 Parametry TIM2

Przypisanie pinów



Rysunek 7 Schemat płytki

Zawarcie plików projektowych

W projekcie zostały użyte zewnętrzne biblioteki do obsługi czujnika oraz wyświetlacza OLED. W tej części sprawozdania opiszę jak odpowiednio je załączyć.

Czujnik DHT11

1. Klonujemy repozytorium

<https://github.com/mesutkilic/DHT11-STM32-Library.git>

2. Załączamy pliki mk_dht11.c oraz mk_dht11.h do naszego projektu odpowiednio do folderu Src i Inc

OLED SSD1306

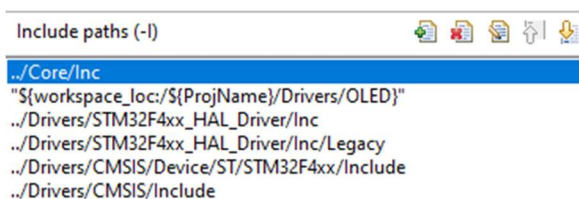
1. Klonujemy repozytorium

<https://github.com/afiskon/stm32-ssd1306.git>

2. Dla tego przypadku posłużę się funkcją załączenia plików konfiguracyjnych poprzez środowisko CUBE-IDE.

3. Uruchamiamy właściwości projektu

4. C/C++ Build -> Settings -> MCU GCC Compiler -> Include Paths -> Dodajemy folder OLED



5. Konfigurujemy plik `ssd1306_conf_temp.h` według tego co jest napisane na stronie github

Opis kodu

```
/* Private includes -----  
/* USER CODE BEGIN Includes */  
#include "ssd1306.h"  
#include "mk_dht11.h"  
#include <stdio.h>  
#include "string.h"  
#include "temp_bit.h"  
#include "hum_bit.h"  
/* USER CODE END Includes */
```

Rysunek 8 Załączenie plików

Na początek załączone są wszystkie pliki potrzebne do obsługi peryferiów. Pliki `xxxx_bit.h` zawierają bitmapy, które będą wyświetlane na ekranie wyświetlacza.

```
/* Private define -----  
/* USER CODE BEGIN PD */  
#define DHT11_PORT GPIOC  
#define DHT11_PIN GPIO_PIN_13  
/* USER CODE END PD */
```

Rysunek 9 Definicja portów

Zdefiniowany również port, do którego dołączony jest czujnik DHT11.

```
/* USER CODE BEGIN PV */  
  
//DHT  
char text_temp[20];  
char text_hum[20];  
dht11_t dht;  
uint8_t temp = 0;  
uint8_t hum = 0;  
  
//bluetooth  
uint8_t ParseBuffer;  
char data[50];  
uint8_t flaga = 1;  
  
/* USER CODE END PV */
```

Rysunek 10 Zmienne

W tym miejscu mamy zmienne służące do obsługi peryferiów. Tablice `text_temp` i `text_hum` przechowują znaki, które są wysyłane przez interfejs UART do urządzenia mobilnego np. telefonu. Zmienna `dht` jest zdefiniowana w celu obsługi czujnika według wymagań zewnętrznej biblioteki. Zmienne `temp` i `hum` przechowują wartości temperatury i wilgotności w formacie bitowym.

Zmienna ParseBuffer przechowuje wartość bitową, która jest przesyłana z urządzenia mobilnego do modułu bluetooth przez interfejs UART. Tablica znaków data przechowuje wartości przesyłane z odułu bluetooth do urządzenia mobilnego w celu ich wyświetlenia np. temperaturę czy wilgotność. Zmienna flaga jest zmienną pomocniczą służącą do włączenia i wyłączenia czujnika.

```
114  /* USER CODE BEGIN 2 */
115
116  ssd1306_Init();
117  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, SET);
118
119  /* USER CODE END 2 */
```

Rysunek 11 Inicjalizacja

Teraz inicjalizowane są peryferia, na początek czujnik a potem moduł bluetooth. Pin C14 jest podłączony do pinu ENABLE modułu bluetooth, dlatego należy ustawić go w stan wysoki, aby uruchomić urządzenie.

```
125  HAL_TIM_Base_Start(&htim2);
126  HAL_UART_Receive_IT(&huart1,&ParseBuffer,1);
127  init_dht11(&dht, &htim2, DHT11_PORT, DHT11_PIN);
128
129  while (1)
130  {
131
132      if(flaga){
133          readDHT11(&dht);
134          temp = dht.temperature;
135          hum = dht.humidity;
136      }
137
138      printScreen();
139
140
141      /* USER CODE END WHILE */
142
143      /* USER CODE BEGIN 3 */
144  }
145  /* USER CODE END 3 */
146 }
```

Rysunek 12 Nieskończona pętla programu

Uruchamiamy timer, który definiuje co, ile czujnik pobiera dane oraz następnie włączamy moduł bluetooth w stan gotowości w celu odebrania nadchodzących danych, ostatecznie uruchamiamy czujnik dht11 za pomocą odpowiedniej funkcji.

W linii 132 widzimy zastosowanie flagi do obsługi czujnika. Flaga domyślnie przyjmuje wartość 1 i pozwala to na pobranie odczytu z czujnika oraz zapisanie go do odpowiednich zmiennych. W przeciwnym wypadku sytuacja ta nie ma miejsca.


```

191 void printScreen()
192 {
193     sprintf(text_temp, " %d C", temp);
194     sprintf(text_hum, " %d %%", hum);
195
196     ssd1306_Fill(Black);
197
198     if(flaga) {
199         ssd1306_DrawBitmap(0, 0, weather, 32, 32, White);
200
201         ssd1306_DrawBitmap(0, 32, humidity, 32, 32, White);
202
203         ssd1306_SetCursor(35, 10);
204         ssd1306_WriteString(text_temp, Font_11x18, White);
205         ssd1306_SetCursor(35, 42);
206         ssd1306_WriteString(text_hum, Font_11x18, White);
207     }
208
209     ssd1306_UpdateScreen();
210
211
212
213
214
215
216 }

```

Rysunek 13 Funkcja printScreen()

Funkcja służy do wyświetlania na ekranie OLED informacji odczytanych z czujnika w zależności od flagi. Jeżeli flaga przyjmuje wartość 1 to na ekranie wyświetlamy informacje, jeżeli to to wszystkie piksele ekranu ustawiane są na kolor czarny. Z powodu tego, że jest to ekran typu OLED to czarny kolor piksela jest równoważny jego wyłączeniu, dlatego pobór mocy jest znikomy.

```

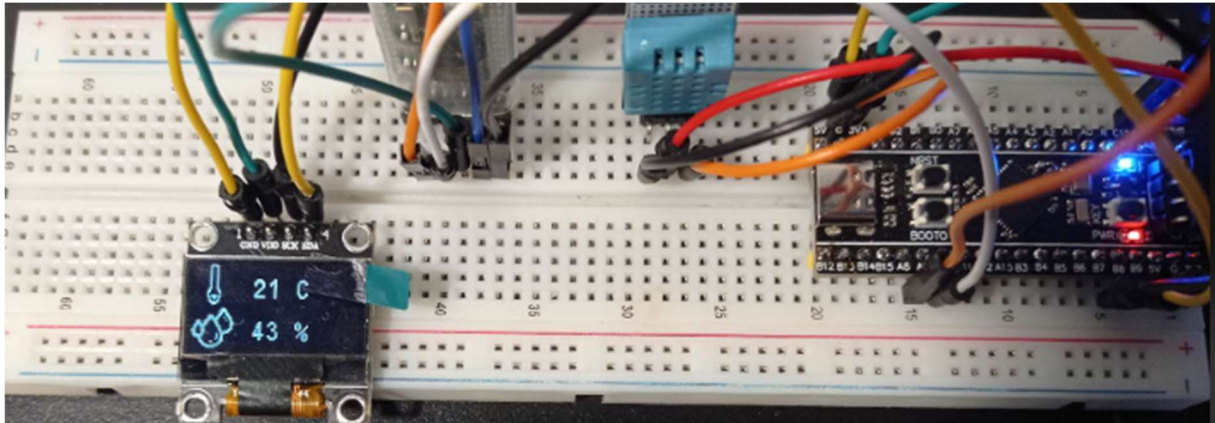
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance==USART1)
    {
        if(ParseBuffer==78) // 78 -> N w Ascii
        {
            flaga = 0;
        }
        else if (ParseBuffer==89) // 89 -> Y w Ascii
        {
            flaga = 1;
        }
        else if(ParseBuffer==84) // 84 -> T w Ascii
        {
            sprintf(data, "Temperatura: %d \r\n", temp);
            HAL_UART_Transmit(&huart1, (uint8_t *)data, sizeof(data), HAL_MAX_DELAY);
        } else if(ParseBuffer==72) // 72 -> H w Ascii
        {
            sprintf(data, "Wilgotnosc: %d \r\n", hum);
            HAL_UART_Transmit(&huart1, (uint8_t *)data, sizeof(data), HAL_MAX_DELAY);
        }
        HAL_UART_Receive_IT(&huart1,&ParseBuffer,1);
    }
}

```

Rysunek 14 Callback do obsługi modułu Bluetooth

W odpowiednim Callback'u zdefiniowano obsługę komunikacji modułu bluetooth -> urządzenie mobilne. Na początek sprawdzamy z jakiego interfejsu UART przychodzi wywołanie funkcji, dalej w zależności od odebranego znaku wysyłana jest odpowiednia informacja. Ten callback jest wywoływany, gdy moduł bluetooth skończy odbierać informacje z urządzenia mobilnego.

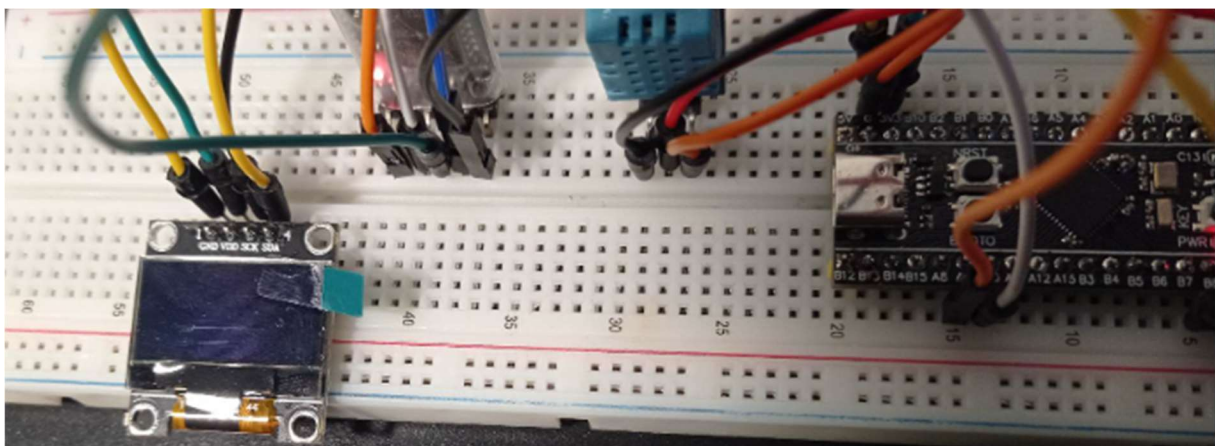
Działanie układu



Rysunek 15 Uruchomiony układ

```
14:14:35.507 Connected  
14:14:43.439 N
```

Rysunek 16 Wysyłamy polecenie do wyłączenia ekranu



Rysunek 17 Ekran uległ wyłączeniu

```
14:16:20.080 T  
14:16:20.493 Temperatura: 21
```

Rysunek 18 Pomiar temperatury

14:16:11.500 H
14:16:11.531 Wilgotnosc: 43

Rysunek 19 Pomiar wilgotności