Projekt SWIS – sprawozdanie

Kamil Kośnik 318380
Kacper Radzikowski 318401

1. **Założenie projektu**

W ramach projektu postanowiliśmy zrealizować urządzenie USB, które umożliwia jednoczesną obsługę dwóch czujników komunikujących się za pomocą interfejsu I2C. Odczytywane pomiary, stan urządzenia lub inne polecenia są przekazywane za pomocą interfejsu USB do komputera RPI5 pracującego pod systemem Raspberry Pi OS, na który został stworzony: sterownik urządzenia, a także prosta aplikacja użytkownika umożliwiająca odczytywanie na urządzeniu docelowym wyników pomiarów, a także kontrolę nad parametrami pracy urządzenia.

2. **Repozytoria z kodem**

Projekt jest przechowywany na dwóch repozytoriach. Jeden z nich przechowuje kod tylko dla urządzenia USB. Drugie repozytorium natomiast przechowuje cały kod działający na urządzeniu, czyli kod sterownika oraz kod aplikacji użytkownika.

Link do repozytorium z kodem urządzenia USB - https://github.com/FRSH-0109/SWIS_STM32_USB_Device

Link do repozytorium z kodem sterownika urządzenia oraz aplikacji użytkownika - https://github.com/FRSH-0109/SWIS_Linux_USB_driver_and_app

3. **Specyfikacja techniczna**

Urządzenie USB – platforma STM32F411E-DISCO

Wykorzystywane czujniki:

a) Czujnik temperatury i wilgotności **SHTC3**, interfejs komunikacyjny - I2C
b) Czujnik temperatury, wilgotności i ciśnienia **BME280**, interfejs komunikacyjny – I2C

Platforma docelowa – Raspberry Pi 5 2GB RAM

4. **Budowanie projektu**

Projekt urządzenia USB zalecane jest budować za pośrednictwem zintegrowanego środowiska programistycznego od firmy ST – STM32CubeIDE.

Projekt sterownika oraz aplikacji użytkownika budujemy natomiast za pomocą napisanego dedykowanego skrytpu Makefile. Budowanie projeku realizujemy z poziomu terminala będąc w katalogu z naszym projektem:

- Zbudowanie kodu sterownika i aplikacji jednocześnie realizujemy poleceniem make all.
- Kod samego sterownika urządzenia budujemy za pomocą polecenia make driver.
- Kod samej aplikacji użytkownika budujemy za pośrednictwem polecenia make app.
- Folder projektu czyścimy za pomocą polecenia make clean.

5. **Urządzenie USB**

Koncepcja działania

Urządzenie USB odpowiada za obsługiwanie pomiarów z podłączonych do niego czujników i przekazywanie ich wyników do naszego urządzenia docelowego. Urządzenie w trakcie swojej pracy inicjuję komunikację z połączonymi czujnikami, a następnie w trakcie swojej pracy kontroluje jaki tryb pracy jest wskazany przez aplikację użytkownika i w zależności od niego stosownie reaguje. Jeżeli aplikacja użytkownika zażąda pracy czujnika w trybie pojedyńczego pomiaru to urządzenie poleci czujnikowi wykonanie pomiaru, odczyta jego wynik i przekaże go protokołem USB do urządzenia docelowego, gdzie ten wynik zostanie obsłużony już przez najpierw sterownik, a następnie przez aplikację użytkownika. Samo urządzenie USB po obsłużenie polecenia ustawi czujnik w tryb oczekiwania (IDLE) i zawiesi wykonywanie pomiaru do otrzymania kolejnego polecenia. Inną opcją jest obsługa polecenia pomiaru cyklicznego, kiedy to czujnik będzie realizował zachowanie identyczne jak w trybie pojedyńczego pomiaru z tym wyjątkiem, że na koniec przeprowadzania pomiaru zamiast przejść w tryb oczekiwania czujnik będzie oczekiwał na kolejne żądanie pomiaru, które otrzyma od sterownika po zadanym okresie.

Ponadto urządzenie USB ma zaimplementowaną obsługę komunikacji po protokole USB poprzez napisane funkcje odpowiadające za rozpoczęcie komunikacji, wysyłanie komunikatów do urządzenia docelowego, parsowanie odbieranych ze sterownika poleceń na potrzeby ich interpretacji.

Resztę kodu stanowią funkcje odpowiadające za konfigurację i inicjację peryferiów połączonych z urządzeniem tj. zegary, porty GPIO, a także inicjację protokołu komunikacyjnego I2C oraz biblioteki HAL z której korzystamy z racji na to, że urządzenie jest utworzone na platformie STM32.

6. **Sterownik urządzenia**

Koncepcja działania

Sterownik urządzenia implementuje mechanizmy potrzebne do koordynowania komunikacji poprzez interfejs USB pomiędzy urządzeniem USB, a urządzeniem docelowym. Do zrealizowania tego celu kod sterownika deklaruje struktury *usb_device_id* odpowiedzialną za określenie jakie urządzenia są kompatybilne z tym sterownikiem (poprzez podanie listy par numerów vendor_id oraz product_id, które pozwalają systemowi zidentyfikować urządzenie) oraz *usb_driver*, który przechowuje podstawowe informacje o sterowniku w postaci jego nazwy oraz stosownej dla niego tablicy usb_device_id. Ponadto struktura ta przechowuje napisane przez nas implementacje funkcji probe oraz disconnect, które są wywoływane przez subsystem USB jądra Linux w momencie wykrywania urządzenia, a także jego odłączania od urządzenia docelowego. W naszym przypadku te dwie funkcje zostały zaimplementowane poprzez funkcje *vendor_probe()* oraz *vendor_disconnect()*. Funkcja vendor_probe() odpowiada za zadeklarowanie, zaalokowanie zasobów dla struktur potrzebnych do funkcjonowania sterownika USB w tym informacji o dostępnych enpointach, używanym interfejsie USB. Ponadto funkcja ta inicjuje wykorzystywany przez sterownik mutex, przygotowuje USB Request Block do nasłuchiwania na wskazanym endpoincie.

Oprócz zadeklarowania wersji struktury usb_driver dla naszego sterownika zaimplementowana została prywatna struktura usb_vendor, która odpowiada za przechowywanie wszystkich informacji potrzebnych do obsługi sterownika w tym między innymi: używanej struktury usb, interfejsu usb, stosowanego mutexa, najnowszą odebraną wiadomość (pole latest_data), dane konfiguracyjne urządzenia takie jak okres wysyłania pomiarów. Ponadto struktura ta posiada pole cmd, które służy do przekazywania nadanych z urządzenia docelowego komend do urządzenia USB.

Sterownik ponadto implementuje funkcje odpowiadające za operacje plikowe z racji na to, że urządzenie USB komunikuje się z systemem operacyjnym właśnie poprzez mechanizm urządzeń plikowych. Z racji na to zaimplementowane zostały własne wersje funkcji open (*vendor_open()*), release (*vendor_release()*), read (*vendor_read()*) i write (*vendor_write()*). Linux jest informowany o wykorzystaniu tych funkcji w celu komunikacji z urządzeniem plikowym za pomocą struktury typu *file_operations*.

Ostatnią częścią kodu sterownika jest funkcja obsługująca jego odpowiedź na przerwanie występujące w URB. W jego ramach sterownik odczytuje nadesłaną przez urządzenie USB ramkę danych poprzez URB i zapamiętuje ją jako ostatnią nadesłaną informację w strukturze usb_vendor (w polu latest_data).

7. **Aplikacja użytkownika**

Koncepcja działania

Zaimplementowana na urządzeniu docelowym aplikacja użytkownika pozwala nam na odczytywanie pomiarów urządzenia w dwóch trybach: jednorazowym, gdy po wywołaniu programu z odpowiednią flagą wskazującą oczekiwanie pojedynczej odpowiedzi z danego czujnika program odpowie na takie wywołanie wyświetlając ostatni zarejestrowany przez sterownik pomiar z danego czujnika (przechowywany w polu *latest_data* struktury *usb_vendor*) albo tryb ciągłego, gdy po wywołaniu kodu z odpowiednią flagą i przekazaniu okresu z jakim chcemy otrzymywać pomiary program będzie z zadanym okresem odczytywał pozyskane z urządzenia przez sterownik pomiary. Ponadto nasza aplikacja posiada komendy pozwalające na uzyskanie informacji o tym jaki jest aktualny stan urządzenia oraz informację o zadanym okresie podawania danych w ramach cyklicznego trybu pracy urządzenia.

Realizowane przez aplikację użytkownika komendy:

1. **Odczyt stanu czujnika 1 [SHTC3]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app s1
SHTC3 IDLE
```

Zwracany opis mówi o stanie, w którym znajduje się czujnik SHTC3

2. **Odczyt stanu czujnika 2 [BME280]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app s2
BME280 IDLE
```

Zwracany opis mówi o stanie, w którym znajduje się czujnik BME280

3. **Odczyt przechowywanego pomiaru czujnika 1 [SHTC3]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app d1
SHTC3 DATA: 25.658 49.605
```

Zwracany opis mówi o aktualnie przechowywanych w pamięci pomiarach, z których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika SHTC3.

4. **Odczyt przechowywanego pomiaru czujnika 2 [BME280]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app d2
BME280 DATA: 25.830 48.270
```

Zwracany opis mówi o aktualnie przechowywanych w pamięci pomiarach, z których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika BME280.

5. **Jednokrotny pomiar na żądanie czujnika 1 [SHTC3]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app i1
SHTC3 DATA: 25.720 50.386
```

Czujnik wykonuje jednokrotny pomiar a następnie zwraca świeżo zabrane dane, których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika SHTC3.

6. **Jednokrotny pomiar na żądanie czujnika 2 [BME280]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app i2
BME280 DATA: 25.820 48.257
```

Czujnik wykonuje jednokrotny pomiar a następnie zwraca świeżo zabrane dane, których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika BME280.

7. **Cykliczny pomiar czujnika 1 [SHTC3]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app cw1 1000
Parameter value: 1000
Writing new period value (1000) to driver!
SHTC3 DATA: 25.287 49.693
SHTC3 DATA: 25.229 49.707
SHTC3 DATA: 25.349 49.747
```

Czujnik wykonuje cykliczne pomiar, co zadany okres czasu, a następnie zwraca świeżo zabrane dane, których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika SHTC3.

8. **Odczyt okresu pomiarowego czujnika 1 [SHTC3]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app cr1
SHTC3 PERIOD: 1000
```

Zwracana informacja to aktualnie przypisany okres pomiary dla czujnika w ms. Przy czym wartość 0 oznacza brak cyklicznych pomiarów.

9. **Cykliczny pomiar czujnika 2 [BME280]**

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app cw2 1234
Parameter value: 1234
Writing new period value (1234) to driver!
BME280 DATA: 25.460 48.312
BME280 DATA: 25.470 48.348
BME280 DATA: 25.460 48.277
```

Czujnik wykonuje cykliczne pomiar, co zadany okres czasu, a następnie zwraca świeżo zabrane dane, których pierwszy to temperatura [*C] a drugi to wilgotność [%] czujnika BME280.

10. Odczyt okresu pomiarowego czujnika 2 [BME280]

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app cr1
SHTC3 PERIOD: 1000
```

Zwracana informacja to aktualnie przypisany okres pomiary dla czujnika w ms. Przy czym wartość 0 oznacza brak cyklicznych pomiarów.

11. Równoczesna praca czujników

Nic nie stoi na przeszkodzie by w ten sposób skonfigurować oba czujniki do równoczesnej pracy, nawet z różnymi okresami pomiarowymi. Wystarczy najpierw ustawić okres pracy jednego z nich, a następnie zrobić to dla drugiego. Poniżej widoczna jest praca dwóch czujników z okresami 1234ms oraz 500ms.

```
kamil@raspberrypi:~/SWIS $ sudo ./user_app cw1 500
Parameter value: 500
Writing new period value (500) to driver!
SHTC3 DATA: 25.442 49.200
SHTC3 DATA: 25.426 49.164
SHTC3 DATA: 25.415 49.200
BME280 DATA: 25.460 47.903
SHTC3 DATA: 25.301 49.187
SHTC3 DATA: 25.410 49.216
BME280 DATA: 25.470 48.009
SHTC3 DATA: 25.375 49.223
```

8. **Rozwój i testowanie**

Warto wspomnieć, że nieodłączonym elementem poprawnego i zrozumiałego uruchomienia całego urządzenia było wykorzystanie analizatora stanów logicznych. Pozwoliło to na sprawne debuggowanie i rozwiązywanie bieżących problemów implementacyjnych.