System sterowania oświetleniem Philips Hue

z użyciem środowiska Zerynth oraz języka Python (1)

Programowanie systemów wbudowanych już od dobrych kilku lat nie jest wyłącznie domeną miłośników asemblera czy niskopoziomowych operacji na rejestrach mikrokontrolera. Dynamiczny rozwój Internetu Rzeczy spowodował, że znaczna część programistów języków wysokiego poziomu, przeniosła swoje zainteresowania zawodowe z obszaru wysoko wydajnych komputerów klasy PC, w kierunku małych mikroprocesorów i mikrokontrolerów. Trend ten nie pozostał niezauważony przez producentów sprzętu i oprogramowania, którzy udostępniają deweloperom gotowe rozwiązania, pozwalające szybko, a przede wszystkim wysokopoziomowo, przygotować kompletny projekt sprzętowo-programowy.

W artykule zaprezentujemy prosty system sterowania żarówkami z serii Philips Hue zbudowany przy użyciu zintegrowanego modułu wyświetlacza Riverdi IoT Display oraz środowiska programistycznego Zerynth – umożliwiającego łatwe i szybkie tworzenie aplikacji z użyciem języka Python.

Warstwa sprzętowa – moduł wyświetlacza oraz system Philips Hue

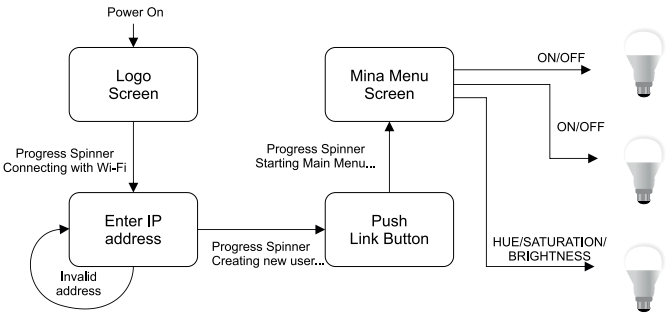
Zagadnienia związane z przygotowaniem projektu od strony sprzętowej ograniczono do minimum, dlatego do jego realizacji zastosowano gotowy moduł Riverdi IoT Display [1] – zintegrowany system wyświetlacza LCD (o przekątnej 5 cali i rozdzielczości ekranu 800×480), kontrolera graficznego Bridgetek BT81x [2] oraz dobrze znanego mikroprocesora ESP32.

Głównym zadaniem stawianym modułowi wyświetlacza będzie nawiązanie komunikacji z systemem Philips Hue [3] oraz zapewnienie możliwości sterowania oświetleniem poprzez prosty, graficzny interfejs użytkownika. W opisywanym przypadku w skład systemu oświetlenia wchodzą dwie żarówki ze sterowaniem włącz/wyłącz oraz jedna żarówka RGB, sterowana poprzez określenie parametrów barwy, jasności oraz nasycenia koloru. Serce systemu stanowi Philips Hue Bridge, podłączony do sieci lokalnej za pomocą przewodu Ethernet i komunikujący się z żarówkami z użyciem protokołu ZigBee. Blokowy schemat istniejącej konfiguracji oraz jej planowej rozbudowy o moduł inteligentnego panelu sterującego pokazuje **rysunek 1**.

Przygotowanie aplikacji w środowisku Zerynth

Zastosowanie gotowego rozwiązania sprzętowego pozwala nam dość szybko przejść do zagadnień związanych bezpośrednio z oprogramowaniem. Pracę nad projektem rozpoczynamy od pobrania zintegrowanego i wieloplatformowego (udostępnionego dla systemów Windows, Linux oraz macOS) środowiska programistycznego Zerynth SDK [4]. Całość procesu instalacji przebiega w sposób standardowy dla wybranego systemu operacyjnego. Przy pierwszym uruchomieniu instalatora użytkownik zostanie poproszony o zaakceptowanie umowy licencyjnej oraz wybór sposobu instalacji (instalacja online lub offline – o ile użytkownik uprzednio pobrał repozytoria bibliotek). Ostatnim krokiem w procesie instalacji jest wybór wersji oprogramowania (w czasie

Rysunek 1. Schemat blokowy projektowanego systemu sterowania oświetleniem



Rysunek 2. Schemat blokowy prezentujący poszczególne etapy działania projektowanej aplikacji

tworzenia artykułu ostatnią dostępną wersją jest r2.6.0). Po podłączeniu modułu wyświetlacza oraz zarejestrowaniu nowego urządzenia w Zerynth SDK nasze środowisko jest już gotowe do pracy.

Zanim jednak przystąpimy do omówienia pierwszych linii kodu, warto krótko przeanalizować poszczególne etapy działania projektowanej aplikacji. Jak pokazuje **rysunek 2**, działanie aplikacji rozpoczynamy od skonfigurowania wyświetlacza oraz wyświetlenia prostego logo, które pozwoli nam upewnić się, że proces pierwszej konfiguracji przebiegł prawidłowo. W kolejnym kroku nawiązujemy połączenie ze zdefiniowaną w programie siecią Wi-Fi (zapisanie identyfikatora SSID sieci oraz hasła dostępu nie jest najbezpieczniejszym rozwiązaniem, jednak na potrzeby testowego uruchomienia systemu w pełni wystarczającym). Po ustanowieniu połączenia użytkownik systemu zostanie poproszony o wpisanie adresu IP urządzenia Philips Hue Bridge. Poprawność wpisanego adresu IP jest weryfikowana poprzez próbę odczytu statusu urządzenia. Przedostatnim krokiem – przed wyświetleniem głównego interfejsu aplikacji – jest proces utworzenia nowego użytkownika systemu oraz jego autoryzacja, które poprawnie zakończone pozwolą nam przejść do ostatniego poziomu menu, jakim jest wyświetlenie panelu sterującego. Ponieważ istniejący system oświetlenia składa się wyłącznie z trzech żarówek (w tym jednej z możliwością sterowania kolorem), panel kontrolny może zostać zrealizowany w postaci jednego ekranu, co pokazuje **fotografia 1**.

Proces przygotowania aplikacji rozpoczynamy od utworzenia nowego projektu w Zerynth Studio. Zgodnie z przyjętym schematem blokowym z rysunku 2, pierwszym zadaniem aplikacji będzie konfiguracja wyświetlacza oraz wyświetlenie prostego logo. W module

Fotografia 1. Główne menu sterowania oświetleniem Philips Hue

Riverdi IoT Display, funkcję silnika graficznego pełni układ BT815 firmy Bridgetek, który stanowi jednocześnie „most" pomiędzy wyświetlaczem LCD (podłączonym do układu BT815 za pomocą 24-bitowego interfejsu RGB) a mikrokontrolerem (komunikującym się poprzez interfejs SPI/QSPI). Typową aplikację układu BT815 ilustruje **rysunek 3**.

Dzięki zastosowaniu układu BT815 aplikacja użytkownika jest zwolniona z obowiązku utworzenia bufora ramki w obszarze pamięci RAM, oraz zaimplementowania niskopoziomowych funkcji związanych z rysowaniem interfejsu. Układ BT815 definiuje szereg prostych obiektów graficznych (przyciski, przełączniki, suwaki itp.), umożliwiających szybkie tworzenie aplikacji, oraz bardziej złożone funkcje, związane z kompresją grafiki czy odtwarzaniem multimediów. Kompletna dokumentacja układu jest dostępna pod adresem: https://bit.ly/2JU2ReW.

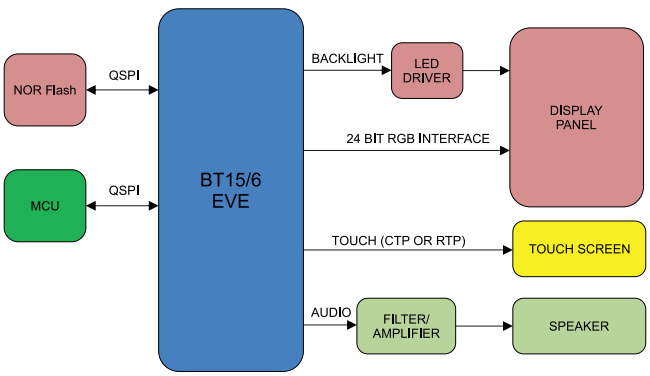
Wraz z zapewnieniem wsparcia dla modułu Riverdi IoT Display programiści środowiska Zerynth przygotowali API dla układów BT81x, umożliwiające tworzenie interfejsu z użyciem kilku prostych funkcji – https://bit.ly/3a1hZlt. Korzystając z dostarczonej dokumentacji, przystępuję do wprowadzenia pierwszych zmian w domyślnie utworzonym pliku *main.py*. Edycję rozpoczynamy od zaimportowania modułów bibliotek dla układów BT81x oraz 5-calowego wyświetlacza Riverdi:

```
from bridgetek.bt81x import bt81x
from riverdi.displays.bt81x import ctp50
```

W kolejnym kroku inicjalizujemy wyświetlacz i komunikację poprzez wybrany interfejs SPI, określając dodatkowo przypisanie linii Chip Select, Interrupt, Power Down:

```
bt81x.init(SPI0, D4, D33, D34)
```

Po poprawnej inicjalizacji układu możemy przystąpić do wyświetlenia prostego logo. W tym celu wybrana grafika musi zostać uprzednio załadowana do wewnętrznej pamięci GRAM kontrolera BT81x. Aby już na pierwszym etapie prac zadbać o odpowiednią strukturę kodu, funkcję `loadImage()` umieśćmy w wydzielonym pliku *gui.py*:



Rysunek 3. Typowa aplikacja układu BT81x (źródło: https://bit.ly/3qOiK7m)

```
def loadImage(image):
    bt81x.load_image(0, 0, image)
```

Zadaniem funkcji `loadImage()` jest wywołanie metody `bt81x.load_image()`, której parametrami są: adres w obszarze pamięci GRAM, dodatkowe flagi sterujące oraz nazwa pliku z wybraną grafiką. W przygotowanej aplikacji – w obszarze pamięci GRAM – będziemy w danej chwili przechowywać wyłącznie jedną grafikę, więc adres w pamięci będzie ustawiony zawsze na wartość 0. Do celów testowych użyjemy grafiki w formacie PNG, o rozmiarze 642×144 pikseli – pozostały obszar obrazu (dla wyświetlacza 5 cali wynosi on 800×480 pikseli) uzupełniony zostanie białym tłem. W pliku *main.py* załadowanie grafiki *gui_logo.png* do pamięci GRAM wykonamy następującym zestawem wywołań:

```
import gui

new_resource('images/gui_logo.png')
gui.loadImage('gui_logo.png')
```

Do uzyskania pełnej funkcjonalności zaimplementujmy również funkcję `showLogo()`, która w określonych współrzędnych wyświetli załadowany obrazek. W tym celu – w pliku *gui.py* – umieśćmy fragment kodu (**listing 1**). Za pomocą metody `bt81x.dl_start()` rozpoczynamy tworzenie tzw. Display List, która opisuje aktualnie tworzoną ramkę. Wywołaniem `bt81x.clear_color()` ustalamy kolor dla operacji czyszczenia obrazu, która jest realizowana poprzez metodę `bt81x.clear()`. Wykorzystując klasę Bitmap, definiujemy parametry wyświetlanego obrazu, m.in. źródło obrazu (w omawianym przypadku jest to pamięć GRAM) oraz jego rozmiar. Za pomocą wywołań `bt81x.prepare_draw()` oraz `bt81x.draw()` dodajemy do tworzonej listy wskazaną grafikę. Tworzenie listy za pomocą wywołania `bt81x.dl_start()` musi zostać zakończone wywołaniem `bt81x.display()` oraz `bt81x.swap_and_empty()`, które zmieni aktualnie wyświetlaną zawartość ekranu.

Listing 1. Funkcja *showLogo()* realizująca prostą operację wyświetlenia grafiki

```
def showLogo():

    bt81x.dl_start()
    bt81x.clear_color(rgb=(0xff, 0xff, 0xff))
    bt81x.clear(1, 1, 1)

    image = bt81x.Bitmap(1, 0, (bt81x.ARGB4, 642 * 2), (bt81x.BILINEAR,
        bt81x.BORDER, bt81x.BORDER, 642, 144))

    image.prepare_draw()
        image.draw(((bt81x.display_conf.width - 642)//2,
        (bt81x.display_conf.height - 144)//2), vertex_fmt=0)

    bt81x.display()
    bt81x.swap_and_empty()
```

Listing 2. Nawiązanie połączenia z wybraną siecią Wi-Fi

```
from wireless import wifi
from espressif.esp32net import esp32wifi as wifi_driver

ssid = "ssid_value"         # this is the SSID of the WiFi network
wifiPWD = "password_value"      # this is the Password for WiFi

gui.showSpinner("Connecting with predefined WiFi network...")
wifi_driver.auto_init()

for _ in range(0,5):
    try:
        wifi.link(ssid,wifi.WIFI_WPA2,wifiPWD)
        break
    except:
    gui.showSpinner("Trying to reconnect...")
else:
    gui.showSpinner("Connection Error - restarting...")
    mcu.reset()
```

Listing 3. Wyświetlenie wbudowanego widżetu *Spinner*

```
def showSpinner(msg):

    bt81x.dl_start()
    bt81x.clear(1, 1, 1)
    txt = bt81x.Text(400, 350, 30, bt81x.OPT_CENTERX | bt81x.OPT_CENTERY, msg, )
    bt81x.add_text(txt)

    bt81x.spinner(400, 240, bt81x.SPINNER_CIRCLE, 0)

    bt81x.display()
    bt81x.swap_and_empty()
```

Komunikacja pomiędzy panelem sterującym a mostkiem Philips Hue Bridge będzie się odbywała poprzez sieć lokalną. W tym celu niezbędne jest podłączenie modułu wyświetlacza do sieci Wi-Fi. Dzięki wykorzystaniu układu ESP32, integrującego moduły łączności Wi-Fi oraz Bluetooth, użytkownik nie musi podłączać żadnych dodatkowych modemów bezprzewodowych. Fragment kodu odpowiedzialny za nawiązanie połączenia z wybraną siecią Wi-Fi (określoną zmienną `ssid` oraz hasłem dostępu – zmienna `wifiPWD`), pokazuje **listing 2**.

Za pomocą pętli `for()` oraz metody `wifi.link()` aplikacja pięciokrotnie próbuje nawiązać połączenie ze zdefiniowaną siecią. W przypadku pełnego niepowodzenia operacji funkcja `mcu.reset()` wykona reset urządzenia. W zależności od siły sygnału sieci Wi-Fi proces łączenia może się wydłużać. Aby użytkownik systemu otrzymał informację zwrotną z interfejsu graficznego, o postępie procesu jest on informowany poprzez wywołanie funkcji `gui.showSpinner()`, która wyświetla wbudowany, animowany obiekt *Spinner* wraz z określonym przez argument komunikatem. Ciało funkcji `showSpinner()` pokazuje **listing 3**.

Wykorzystując powyższą funkcję, w utworzonej liście umieszczamy elementy typu *Spinner* oraz *Text* – całość listy zamykamy wywołaniami `bt81x.display()` oraz `bt81x.swap_and_empty()`. Rezultat pokazuje **fotografia 2**.

Fotografia 2. Ekran z obiektem typu *Spinner*, informujący użytkownika o postępie procesu

W kolejnym kroku, po poprawnym nawiązaniu połączenia z siecią Wi-Fi, możemy przystąpić do utworzenia interfejsu graficznego, umożliwiającego wprowadzenie adresu IP, przypisanego do urządzenia Philips Hue Bridge. Od tego zaczniemy kolejną część artykułu.

Łukasz Skalski
contact@lukasz-skalski.com

Przypisy:
[1] https://bit.ly/34k3Gov
[2] https://bit.ly/3m6BeMQ
[3] https://bit.ly/3gDUIHf
[4] https://bit.ly/3qOad43