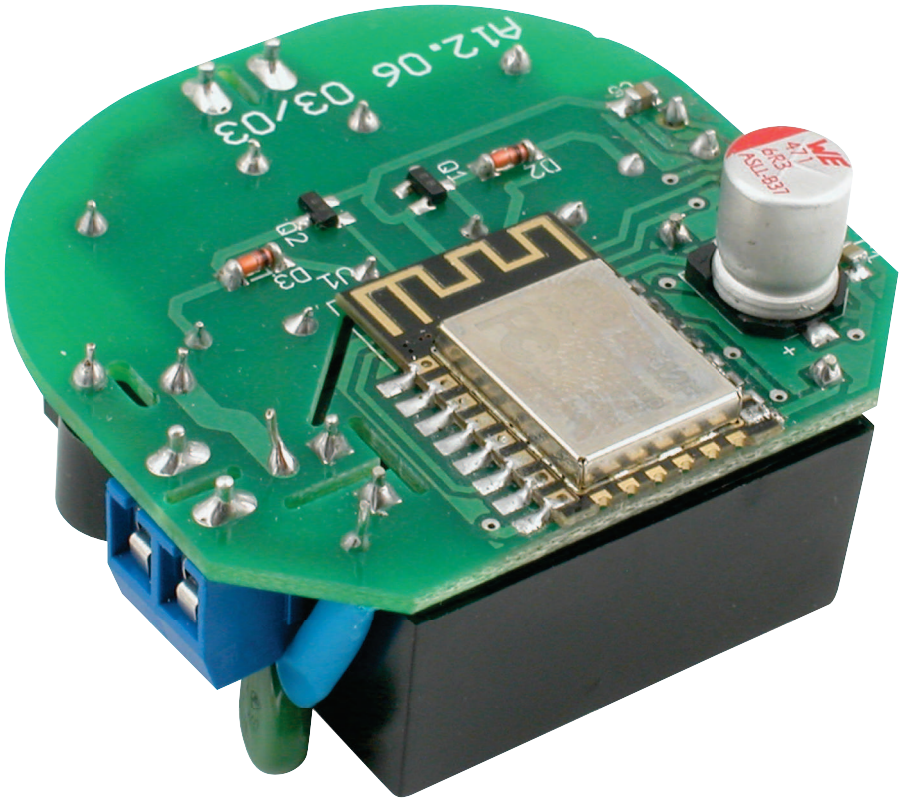
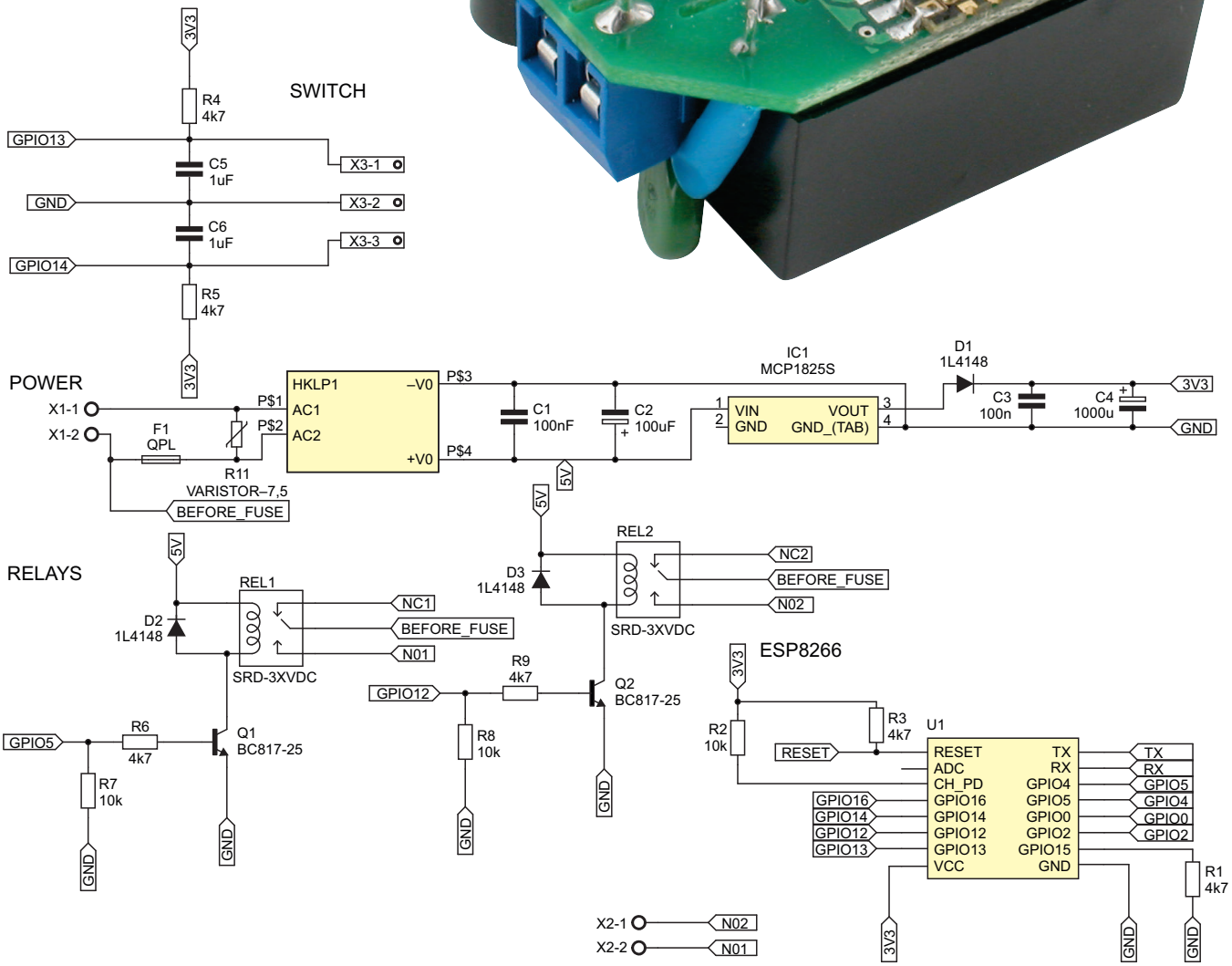
Sterownik oświetlenia na 230 V zgodny z HomeKit

Możliwość zdalnego kontrolowania oświetlenia jest komfortowym udogodnieniem, ale może także zwiększyć bezpieczeństwo naszego dobytku. Dzięki możliwości załączania oświetlenia w mieszkaniu możemy zasymulować obecność domowników i zniechęcić tym potencjalnego włamywacza. Prezentowane urządzenie jest interesujące jeszcze z innego powodu, można je połączyć z aplikacją Home firmy Apple.





Rysunek 1. Schemat sterownika

Dodatkowe materiały do pobrania ze strony www.media.avt.pl

W ofercie AVT* AVT-5713

Podstawowe parametry:
- zasilanie 230 V AC,
- możliwość podpięcia pod homekit,
- możliwość sterowania jednym lub dwoma punktami świetlnymi,
- złącza umożliwiające podpięcie kabli 1,5 mm² typu drut,
- wsparcie dla włącznika świecznikowego.

Projekty pokrewne na www.media.avt.pl:

AVT-5685	Mikrosterownik Wi-Fi (EP 7/2019)
AVT-5675	Moduł przekaźników sterowanych przez Wi-Fi (EP 4/2019)
AVT-5662	Kolorofon z Wi-Fi (EP 2/2019)
AVT-5530	Regulator natężenia oświetlenia z Wi-Fi (EP 1/2016)

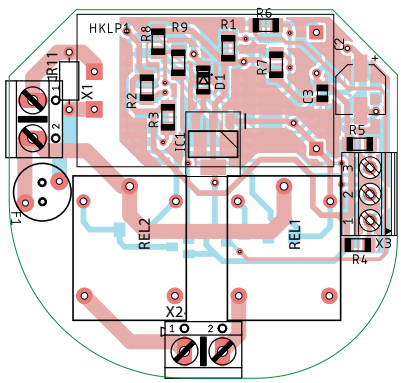
* **Uwaga! Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**

Podstawową wersją zestawu jest wersja [**B**] nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji [**B**] zawiera elementy elektroniczne (w tym [**UK**] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.

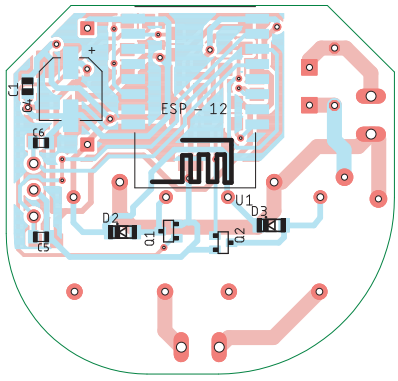
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- wersja [**C**] – zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] – płytka drukowana bez elementów i dokumentacji

Kity w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- wersja [**A+**] – płytka drukowana [**A**] + zaprogramowany układ [**UK**] i dokumentacja
- wersja [**UK**] – zaprogramowany układ

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! *http://sklep.avt.pl*. W przypadku braku dostępności na *http://sklep.avt.pl*, osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: kity@avt.pl.



Rysunek 2. Schemat płytki PCB sterownika, strona TOP

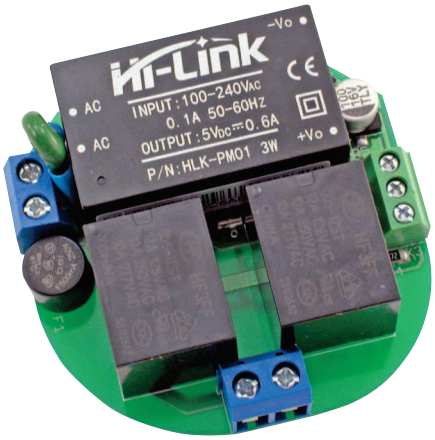


Rysunek 3. Schemat płytki PCB sterownika, strona BOTTOM

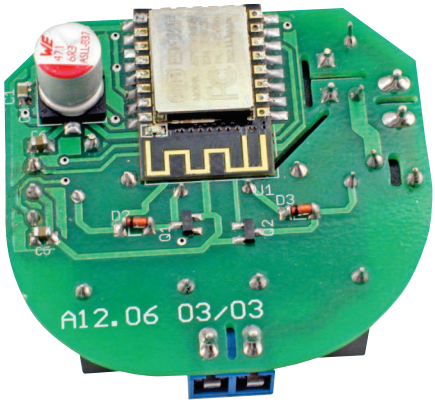
Cała przygoda z zaprojektowaniem sterownika zaczęła się od momentu nabycia modułu Wemos D1 mini z układem ESP8266 12F. Nie tracąc czasu, zacząłem testować moduł. Na początku były to proste programy, jak Web Serwer z funkcją włączania zwykłej diody led. Wtedy pomyślałem: Skoro można sterować diodą led, gdy jest się w zasięgu tej samej sieci co Wemos, to czy uda się zbudować sterownik oświetlenia, z którym będzie można się łączyć z każdego miejsca na ziemi?. Znalazłem informacje na temat protokołów MQTT (MQ Telemetry Transport) oraz HAP (HomeKit Accessory Protokol). Gdy przyswoiłem wiedzę z tego zakresu, wtedy uznałem, że przyszedł czas na zaprojektowanie sterownika.

Budowa urządzenia

Schemat sterownika został pokazany na **rysunku 1**. Sercem sterownika jest moduł ESP8266 12F, działa on w standardzie Wi-Fi 802.11 b/g/n na częstotliwości 2,4 GHz. Zawiera 11 cyfrowych wejść/wyjść, jeden interfejs SPI oraz I²C. Na płytce nie zabrakło

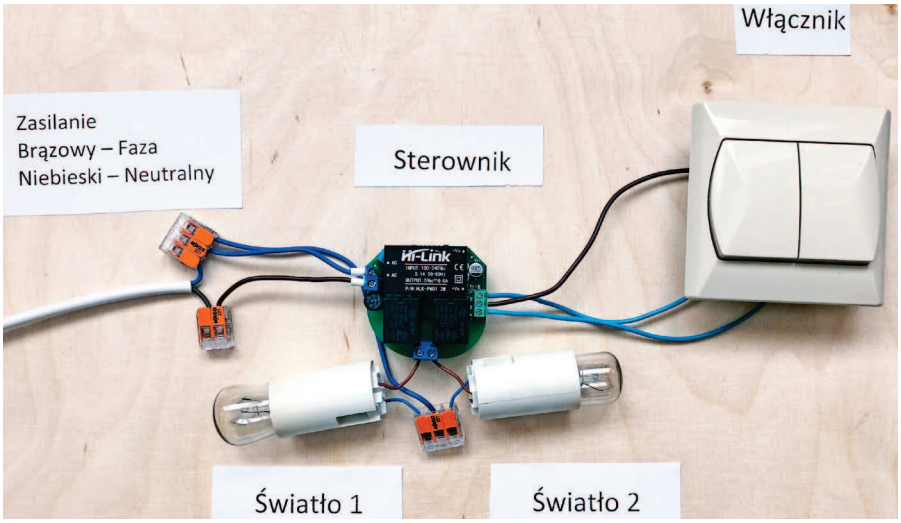


Fotografia 4. Zmontowany sterownik, widok od strony TOP



Fotografia 5. Zmontowany sterownik, widok od strony BOTTOM

Fotografia 6. Zmontowany sterownik umieszczony w puszce instalacyjnej



Fotografia 7. Sposób podłączenia sterownika

Wykaz elementów:

Rezystory: (SMD1206)
R1, R3…R6, R9: 4,7 kΩ
R2, R7, R8: 10 kΩ

Kondensatory:
C1, C3: 100 nF (SMD0805)
C2: 100 µF
C4: 330...1000 µF
C5, C6: 1 µF (SMD0805)

Półprzewodniki:
D1…D3: 1N4148 (SMD)
Q1, Q2: BC817-25
IC1: MCP1825S
U1: ESP8266 12F
HKLP1: HLK-PM01 5 V, 0,6 A

Pozostałe:
F1: topikowy, szybki, 500 mA, 250 VAC, raster: 5,08mm
R11: Warystor 275 VAC
REL1, REL2: SRD-05VDC-SL-C
X1, X3: ARK 2/5,0 mm
X3: ARK 3/2,54 mm

również przetwornika analogowo-cyfrowego (wejście ADC) oraz złącza UART (RX, TX). Wymiary tego modułu to tylko 24×16 mm. Dzięki temu jest to bardzo popularny moduł stosowany w wielu aplikacjach dla Internetu Rzeczy (Internet of Things).

Schemat płytki PCB pokazują **rysunki 2** i **3**. Na płytce został umieszczony zasilacz modułowy HLK-PM01 5V, dzięki czemu urządzenie jest zasilane bezpośrednio napięciem 230 VAC. Napięcie z zasilacza doprowadzone jest do przekaźników, a do zasilania modułu ESP8266 zastosowano dodatkowy stabilizator napięcia LDO typu MCP1825S. Aby zabezpieczyć płytkę przed przepięciami elektrycznymi występującymi w sieci, zastosowano bezpiecznik topikowy szybki 0,5 A i włączony za nim, do linii zasilających, warystor. Pomimo że zasilacz jest stabilizowany, zdecydowałem się zastosować parę kondensatorów – elektrolityczny 100 µF i ceramiczny 100 nF dla napięcia 5 V oraz parę kondensatorów – elektrolityczny 1000 µF i ceramiczny 100 nF dla napięcia 3,3 V.

Podczas projektowania płytki pomyślałem, że warto dodać złącze umożliwiające podpięcie łącznika świecznikowego. Jako zabezpieczenie przed efektem *bouncingu* użyłem kondensatorów ceramicznych 1 µF włączonych równolegle do zacisków tego złącza. Na płytce znajdują się również dwa przekaźniki do sterowania oświetleniem. Do załączania przekaźników przez ESP8266 zastosowałem tranzystory BC817-25. Istotne jest dodanie rezystorów podciągających 10 kΩ do masy na wyprowadzeniach sterujących tranzystorami. Dzięki temu żadne zakłócenia w działaniu układu nie spowodują przypadkowego załączenia wyjść. Z każdego przekaźnika poprowadzone jest jedno wyjście (NO – normally open) do złącza wyjściowego X2.

Listing 1. Definicje stałych i import bibliotek

```
#define _light_first 5
#define _light_second 12
#define _switch_first 14
#define _switch_second 13

#define _on 1
#define _off 0

#include <ESPAsyncWiFiManager.h>
#include <AceButton.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

Listing 2. Zmienne oraz definicje funkcji

```
// serwer będzie nadsłuchiwał na porcie numer 80
AsyncWebServer server(80);
DNSServer dns;

int state_light_first;
int state_light_second;

AceButton firstLightButton;
AceButton secondLightButton;

void buttonsAction(AceButton*, uint8_t, uint8_t);
```

Listing 3. Użycie biblioteki *AsyncWiFiManager*

```
  void setup() {
  AsyncWiFiManager wifiManager(&server, &dns);
  wifiManager.setConfigPortalTimeout(120);
  wifiManager.autoConnect(/*nazawaSieci*/"Ligth switch" /*,"opcjonalnieHasloDostepuDoSieci"*/);
}
```

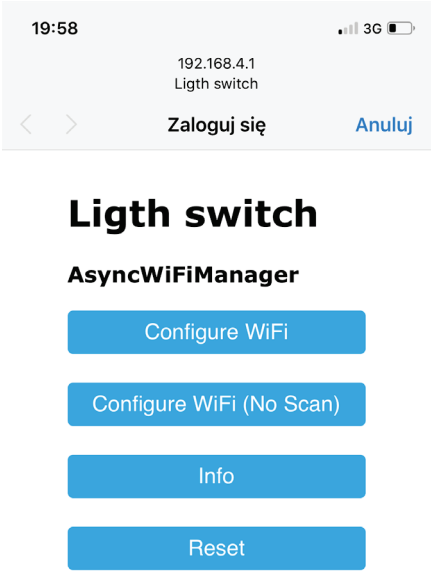
Listing 4. Metody realizowane przez serwer

```
//Na zapytanie localhost/reset urządzenie zostanie uruchomione ponownie,
//serwer jako odpowiedź wyśle wiadomość o ponownym uruchomieniu oraz kod 200
server.on("/reset", HTTP_GET, [](AsyncWebServerRequest * request) {
          request->send(200, "text/plain", "Device will be reseted");
          delay(1000);
          ESP.restart();
});


//Na zapytanie localhost/state_light_first urządzenie wyśle
//stan pierwszej żarówki oraz kod 200
server.on("/state_light_first", HTTP_GET, [](AsyncWebServerRequest * request) {
          request->send(200, "text/plain", String(state_light_first));
});


//Na zapytanie localhost/state_light_second urządzenie wyśle
//stan drugiej żarówki oraz kod 200
server.on("/state_light_second", HTTP_GET, [](AsyncWebServerRequest * request) {
          request->send(200, "text/plain", String(state_light_second));
});


//Na zapytanie localhost/on_light_first sterownik
//włączy pierwszą żarówkę, zmieni jej stan oraz wyśle kod 204
server.on("/on_light_first", HTTP_GET, [](AsyncWebServerRequest * request) {
          digitalWrite(_light_first, HIGH);
          state_light_first = _on;
          request->send(204);
});


//Na zapytanie localhost/on_light_second sterownik
//włączy drugą żarówkę, zmieni jej stan oraz wyśle kod 204
server.on("/on_light_second", HTTP_GET, [](AsyncWebServerRequest * request) {
          digitalWrite(_light_second, HIGH);
          state_light_second = _on;
          request->send(204);
});


//Na zapytanie localhost/off_light_first sterownik wyłączy
//pierwszą żarówkę, zmieni jej stan oraz wyśle kod 204
server.on("/off_light_first", HTTP_GET, [](AsyncWebServerRequest * request) {
          digitalWrite(_light_first, LOW);
          state_light_first = _off;
          request->send(204);
});


//Na zapytanie localhost/off_light_second sterownik wyłączy
//drugą żarówkę, zmieni jej stan oraz wyśle kod 204
server.on("/off_light_second", HTTP_GET, [](AsyncWebServerRequest * request) {
          digitalWrite(_light_second, LOW);
          state_light_second = _off;
          request->send(204);
});
```

Montaż

Podczas montażu sterownika należy pamiętać, żeby zacząć od najmniejszych części, takich jak rezystory, tranzystory czy diody prostownicze, gdyż niektóre z nich umieszczone są pod przekaźnikami lub zasilaczem. **Fotografie 4** i **5** pokazują zmontowane płytki. Zastosowane złącza ARK pozwalają na podpięcie kabli typu drut o przekroju 1,5 mm², używanego w instalacjach oświetlenia domowego. Sterownik ma wymiary 48×51 mm, dzięki czemu mieści się w puszkach podtynkowych ∅ 60 mm oraz ∅ 70 mm (**fotografia 6**).

Zasilanie sterownika (przewody neutralny i fazowy) podłączamy do złącza X1 (przy warystorze). Do złącza X2 (umieszczone między dwoma przekaźnikami) podłączamy przewody fazowe (brązowe) oświetlenia. Nie musimy do sterownika podłączać dwóch punktów świetlnych, sterownik może pracować z podłączonym jednym punktem świetlnym. Do złącza X3 podłączamy łącznik



Rysunek 8. Strona konfiguracyjna AsyncWiFiManager

Listing 5. Konfiguracja pinów
```
pinMode(_light_first, OUTPUT);
pinMode(_light_second, OUTPUT);

pinMode(_switch_first, INPUT);
pinMode(_switch_second, INPUT);

digitalWrite(_light_first, LOW);
digitalWrite(_light_second, LOW);

digitalWrite(_switch_first, LOW);
digitalWrite(_switch_second, LOW);
```

Listing 6. Konfiguracja metod wywoływanych dla przyciśnięcia przycisków
```
ButtonConfig* buttonsConfig = ButtonConfig::getSystemButtonConfig();
buttonsConfig->setEventHandler(buttonsAction);
buttonsConfig->setFeature(ButtonConfig::kFeatureClick);
```

Listing 7. Zainicjowanie przycisków
```
firstLightButton.init(_switch_first, LOW, 0);
secondLightButton.init(_switch_second, LOW, 1);

void loop(){
//...
firstLightButton.check();
secondLightButton.check();
//...
}
```

świecznikowy, środkowy zacisk to przewód wspólny (COM). Podłączenie instalacji testowej pokazuje **fotografia 7**.

Oprogramowanie

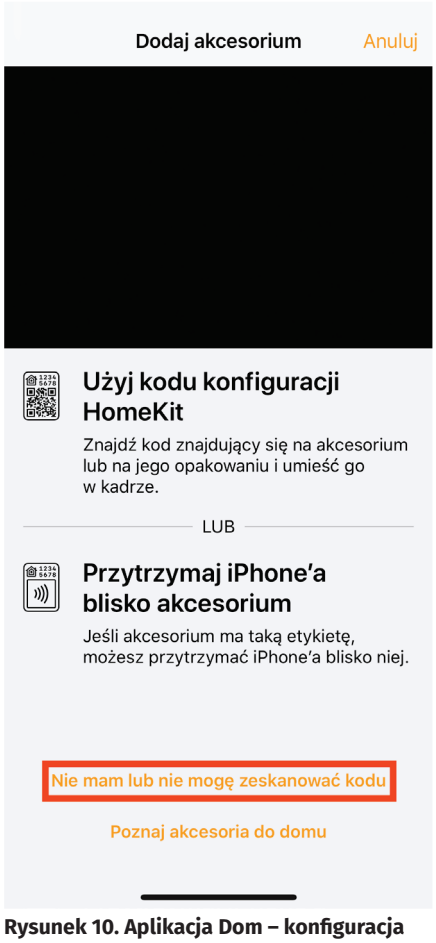
Definicje stałych oraz import bibliotek użytych w programie prezentuje **listing 1**. Z kolei zmienne oraz definicje funkcji pokazuje **listing 2**. Podczas programowania sterownika trudno jest przewidzieć, czy nasze hasło lub nazwa sieci się nie zmieni, dlatego zdecydowałem się na użycie biblioteki AsyncWiFiManager. Gdy moduł ESP8266 nie będzie w stanie podłączyć się do zapisanej sieci lub żadna sieć nie będzie zapisana, to zostanie uruchomiony punkt dostępu oraz strona pozwalająca na konfigurację połączenia Wi-Fi. Należy wtedy połączyć się z siecią Wi-Fi stworzoną przez ESP a następnie wejść na stronę 192.168.4.1 i dokonać konfiguracji (**rysunek 8**). Ustawienie Timeout-u na 120 sekund spowoduje, że jeśli nie zostanie podjęta próba konfiguracji połączenia wi-fi to strona konfiguracyjna zniknie po 120 sekundach i układ będzie pracował normalnie z uruchomionym punktem dostępu. Gdy będziemy chcieli, aby strona konfiguracyjna znów była dostępna to należy połączyć się z punktem dostępowym ESP8266 a następnie wejść na stronę 192.168.4.1/reset, urządzenie wtedy zostanie zresetowane. Ten zabieg ma na celu umożliwić korzystanie ze sterownika w przypadku braku sieci Wi-Fi.

Użycie AsyncWiFiManger pokazuje **listing 3**.

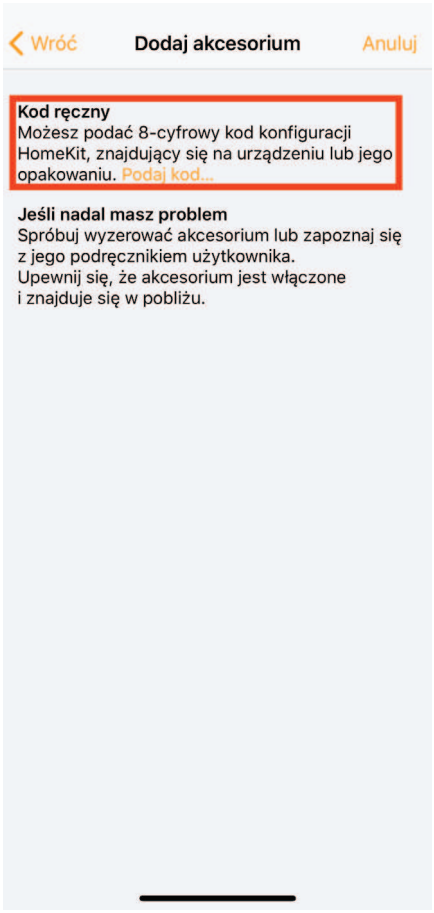
Do uruchomienia webserwera zdecydowałem się zastosować ESPAsyncWebServer. Serwer ten wyróżnia się tym, że jeśli gdzieś w środku kodu będziemy mieli wywołanie funkcji delay webserwer, cały czas będzie odpowiadał na wysyłane do niego zapytania. Należy tutaj wspomnieć, że wymieniona wyżej wersja biblioteki do konfiguracji połączenia Wi-Fi współpracuje z asynchroniczną wersją webserwera, dostępna jest również biblioteka, która nie współpracuje z tym serwerem.

Listing 8. Funkcja do sterowania oświetleniem poprzez użycie przycisków
```
void buttonsAction(AceButton * button, uint8_t eventType,
uint8_t /* buttonState */) {
switch (eventType) {
        case AceButton::kEventPressed:
                if (button->getId() == 0) {
                        if (state_light_first == _on) {
                                state_light_first = _off;
                                digitalWrite(_light_first, LOW);
                        } else if (state_light_first == _off) {
                                state_light_first = _on;
                                digitalWrite(_light_first, HIGH);
                        }
                        break;
                } else if (button->getId() == 1) {
                        if (state_light_second == _on) {
                                state_light_second = _off;
                                digitalWrite(_light_second, LOW);
                        } else if (state_light_second == _off) {
                                state_light_second = _on;
                                digitalWrite(_light_second, HIGH);
                        }
                }
        break;
        case AceButton::kEventReleased:
                if (button->getId() == 0) {
                if (state_light_first == _on) {
                                state_light_first = _off;
                                digitalWrite(_light_first, LOW);
                        } else if (state_light_first == _off) {
                                state_light_first = _on;
                                digitalWrite(_light_first, HIGH);
                        }
                } else if (button->getId() == 1) {
                        if (state_light_second == _on) {
                                state_light_second = _off;
                                digitalWrite(_light_second, LOW);
                        } else if (state_light_second == _off) {
                                state_light_second = _on;
                                digitalWrite(_light_second, HIGH);
                        }
                }
        break;
}
```

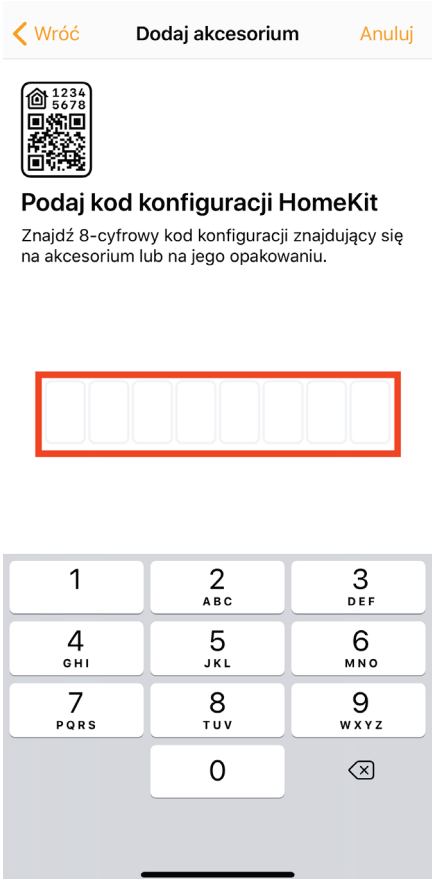
Rysunek 9. Aplikacja Dom – dodawanie akcesorium



Rysunek 10. Aplikacja Dom – konfiguracja



Rysunek 11. Aplikacja Dom – wybór ręcznego wprowadzenia kodu



Rysunek 12. Aplikacja Dom – wygenerowany kod

Listing 9. Zawartość pliku *config.json*

```
„accessories”: [
{
          „name”: „First light”,
          „accessory”: „HTTP-LIGHTBULB”,
          „onUrl”: „http://192.168.0.7/on_light_first”,
          „offUrl”: „http://192.168.0.7/off_light_first”,
          „statusUrl”: „http://192.168.0.7/state_light_first”
},
{
          „name”: „Second light”,
          „accessory”: „HTTP-LIGHTBULB”,
          „onUrl”: „http://192.168.0.7/on_light_second”,
          „offUrl”: „http://192.168.0.7/off_light_second”,
          „statusUrl”: „http://192.168.0.7/state_light_second”
}
]
```

Gdy już mamy utworzony serwer to należy dodać do niego metody, które będzie realizował jako odpowiedź na zapytanie. W funkcji void setup() umieszczamy kod z **listingu 4**. Na samym końcu wystarczy uruchomić serwer poleceniem: server.begin();

Konfiguracja pinów i początkowe ustawienie stanów pokazuje **listing 5**. Do obsługi przycisków zdecydowałem się użyć biblioteki AceButton, ponieważ zapewnia eliminację efektu *bouncingu*. Będziemy używać dwóch przycisków, więc należy stworzyć konfigurację (podać nazwę metody, która ma być wywołana, gdy przycisk zostanie naciśnięty, oraz po jakiej akcji metoda ma być wywołana). W funkcji void setup() umieszczamy kod z **listingu 6**. Następnie należy zainicjować przyciski. Pierwszy parametr to numer pinu, pod który podpięty jest przycisk, drugi to stan, którym jest aktywowany, a trzeci to numer id przycisku – **listing 7**. Na koniec w metodzie void loop() należy umieścić „sprawdzanie” przycisków.

Funkcja sterowania oświetleniem, przez użycie przycisków, wygląda jak na **listingu 8**.

Bardzo fajną rzeczą jest fakt, że sterownik działa z łącznikiem świecznikowym, dzięki czemu nie trzeba go wymieniać na taki, który jest używany w dzwonkach. Po wciśnięciu lub zwolnieniu przycisku sprawdzany jest numer id przycisku, który wywołał funkcję, potem sprawdzany jest stan danej żarówki, a następnie stan żarówki wraz ze stanem na pinie jest negowany, co gasi albo zapala światło. Oprogramowanie w sterowniku zostało przystosowane do współpracy z HomeKit, aczkolwiek bez problemu można stworzyć aplikację na dowolny system, która będzie się komunikowała ze sterownikiem dzięki temu, że zastosowano protokół HTTP.

Wszystkie linki do bibliotek użytych w tym projekcie są zamieszczone na końcu artykułu.

Opisane poniżej oprogramowanie dostępne jest również na moim githubie (link na końcu artykułu).

Czym jest HomeKit

Firma Apple udostępniła protokół HAP (HomeKit Accessory Protocol), aby móc podłączyć urządzenia firm trzecich do aplikacji Home. Można samemu stworzyć odpowiedni kod, który połączy nasze ESP z aplikacją, ale można wykorzystać tak zwany mostek, który zrobi to za nas. Mostek komunikuje się z naszymi urządzeniami na ESP8266 za pomocą protokołu HTTP oraz z urządzeniami firmy Apple za pomocą HAP. HAP jest od początku do końca szyfrowany. Dodatkowo urządzeniami dodanymi do aplikacji Home można sterować głosowo poprzez asystenta Siri.

Instalacja homeKita

Aby uruchomić Raspberry Pi, potrzebujemy karty micro sd dobrej klasy, gdyż to z niej będzie uruchamiany system, więc im wolniejsza karta, tym wolniej będzie się ładował system. Najnowszą wersję oprogramowania pobieramy ze strony Raspberry Pi. Dodatkowo potrzebujemy programu Etcher, aby nagrać system na kartę pamięci. Po wszystkim wkładamy kartę do minikomputerka, podpinamy klawiaturę i myszkę poprzez adapter USB OTG, podłączamy monitor i na końcu podpinamy zasilanie. Gdy Rasberry Pi już się nam uruchomi, zaczynamy instalować Homebridge.

Instrukcja instalacji dostępna jest również na stronie Homebridge.

Pierwszym krokiem jest otworzenie terminalu i aktualizacja pakietów oraz systemu.

Wpisujemy po kolei:

```
sudo apt-get update
sudo apt-get upgrade
```

Następnie zaczynamy instalować Homebridge:

```
sudo apt-get install curl
curl -sL https://deb.nodesource.
com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
sudo apt-get install
libavahi-compat-libdnssd-dev
sudo npm install -g --unsafe-perm
homebridge
```

Aby Homebridge wystartował, zawsze wraz z uruchomieniem systemu powinniśmy uruchomić demona Homebridge poprzez terminal. Na samym początku musimy edytować plik homebridge który znajduje się w katalogu /etc/default/

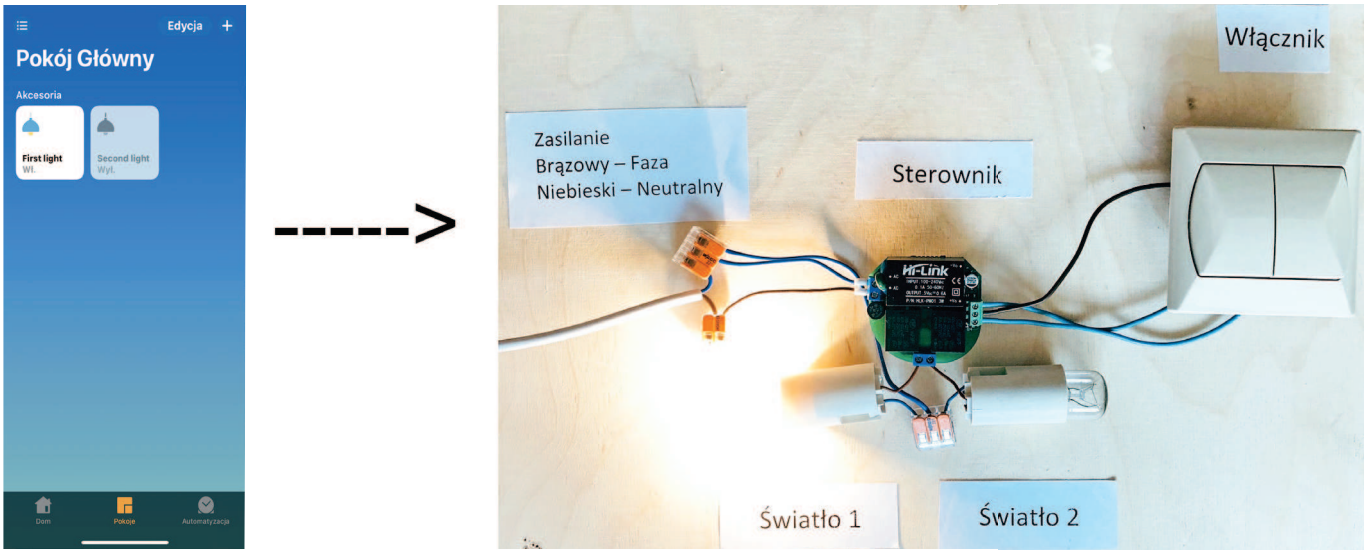
```
sudo nano /etc/default/homebridge
```

I wpisać

```
HOMEBRIDGE_OPTS=-U /var/homebridge
```

Linijka ta informuje demona homebridge gdzie znajdują się pliki konfiguracyjne.

Następnie musimy edytować plik homebridge.service który znajduje się w katalogu /etc/systemd/system/



Rysunek 13. Aplikacja Dom po skonfigurowaniu

```
sudo nano /etc/systemd/system/
homebridge.service
```

Wpisujemy poniższy tekst:

```
[Unit]
Description=Node.js HomeKit
Server
After=syslog.target network-
online.target

[Service]
Type=simple
User=homebridge
EnvironmentFile=/etc/default/
homebridge
ExecStart=/usr/local/bin/
homebridge $HOMEBRIDGE_OPTS
Restart=on-failure
RestartSec=10
KillMode=process

[Install]
WantedBy=multi-user.target
```

Następnym krokiem jest dodanie użytkownika, który będzie uruchamiał demona

```
sudo useradd --system homebridge
```

Gdy to już zrobiliśmy to musimy utworzyć katalog homebridge w katalogu /var i przekopiować do niego plik konfiguracyjny config.json, oraz katalog persist (na tym etapie urządzenia powinny być już dodane)

```
sudo mkdir /var/homebridge
sudo cp /root/.homebridge/config.
json /var/homebridge/
sudo cp -r /root/.homebridge/
persist /var/homebridge
```

Potem musimy zmienić zezwolenia dostępu do katalogu homebridge/

```
sudo chmod -R 0777 /var/homebridge
```

Przy edycji pliku config.json z katalogu *.homebridge* należy powtórzyć kopiowanie tego pliku i katalogu *persist*.

Gdy to wszystko już zrobiliśmy możemy uruchomić demona hombridge

```
sudo systemctl deamon-reload
sudo systemctl enable
homebridge sudo system start
homebridge
```

Teraz możemy sprawdzić status Homebridge aby odczytać wygenerowany kod służący do połączenia z mostkiem. W terminalu należy wpisać:

```
sudo system status homebridge
```

Jeśli kod jeszcze się nie wygenerował, to powtarzamy polecenie. Następnie otwieramy aplikację Dom i wybieramy „Dodaj akcesorium" (**rysunek 9**). W nowym oknie na dole wybieramy „Nie mam lub nie mogę zeskanować kodu" (**rysunek 10**), potem wybieramy „Podaj kod…" (**rysunek 11**) i wpisujemy wygenerowany przez Homebridge kod (**rysunek 12**). **Ważna uwaga:** urządzenie, na którym dodajemy mostek, jak i Raspberry Pi, powinny się znajdować w tej samej sieci. Po wpisaniu kodu nasz mostek zostanie dodany (może to chwilę potrwać, lecz nigdy nie trwa to dłużej niż 60 sekund).

Teraz, gdy mamy stworzony mostek, możemy zainstalować plugin do obsługi naszego sterownika. W tym celu wchodzimy na stronę ***https://www.npmjs.com***, na której są dostępne pakiety do Homebridge i w wyszukiwarce na stronie wpisujemy „homebridge-http-lightbulb". Autor pluginu przedstawił sposób instalacji oraz konfigurację. Zacznijmy od instalacji, wracamy do terminalu i wpisujemy:

```
sudo npm install -g
homebridge-http-lightbulb
```

Teraz zostanie pobrany i zainstalowany plugin, może to chwilę potrwać, czas instalacji zależy od kilku czynników (prędkości łącza, rodzaju Raspberry Pi czy prędkości karty micro sd). Teraz musimy dopisać konfigurację pluginu do pliku *config.json*. Wymieniony plik znajduje się w katalogu *~/.homebridge/*. Aby wejść do katalogu, wpisujemy w terminalu:

```
cd ~/.homebridge/
```

Następnie uruchamiamy plik w edytorze *nano:*

```
sudo nano config.json
```

Plugin pozwala również na sterowanie jasnością lub kolorami żarówki, nasz sterownik obsługuje włączenie/wyłączenie światła i dlatego skupimy się na konfiguracji tej funkcji. W tablicy *accessories* dodajemy nowe obiekty (**listing 9**). **Ważna uwaga:** 192.168.0.7 to adres IP modułu ESP, może to być inny adres. Aby zapobiec zmianie adresu IP, np. po restarcie routera, warto dodać rezerwację adresu ip dla naszego ESP. Gdy chcemy sterować jedną żarówką, dodajemy tylko jeden obiekt do tablicy. Na koniec zapisujemy plik (CTRL+O) i wychodzimy (CTRL+X).

Po tych czynnościach należy wykonać restart Homebridge poleceniem:

```
sudo systemctl restart homebridge
```

Po ponownym uruchomieniu do aplikacji Dom zostały dodane dwa nowe urządzenia (**rysunek 13**). Po kliknięciu na jedno z nich światło się zapali, a gdy klikniemy drugi raz, światło zgaśnie. Gdy przytrzymamy palcem akcesorium, to otworzy nam się nowy widok, w którym będzie zwykły suwak. Należy również wspomnieć, że aby sterować urządzeniami, gdy jest się poza siecią, w której znajduje się Raspberry Pi, trzeba mieć iPada, Apple TV lub głośnik Apple HomePod. Urządzenie to będzie pełniło funkcję centrum akcesoriów.

Michał Urban

Linki do bibliotek i programów:

1. ESPAsyncWiFiManager ***http://bit.ly/2Nhq6yF***
2. ESPAsyncWebServer ***http://bit.ly/33L1DHS***
3. AceButton ***http://bit.ly/2ZbHdo5***
4. HomeBridge ***http://bit.ly/2Z1BGFu***
5. Light bulb plugin homebridge ***http://bit.ly/2L15HeO***
6. Soft ***http://bit.ly/31R2Z1U***