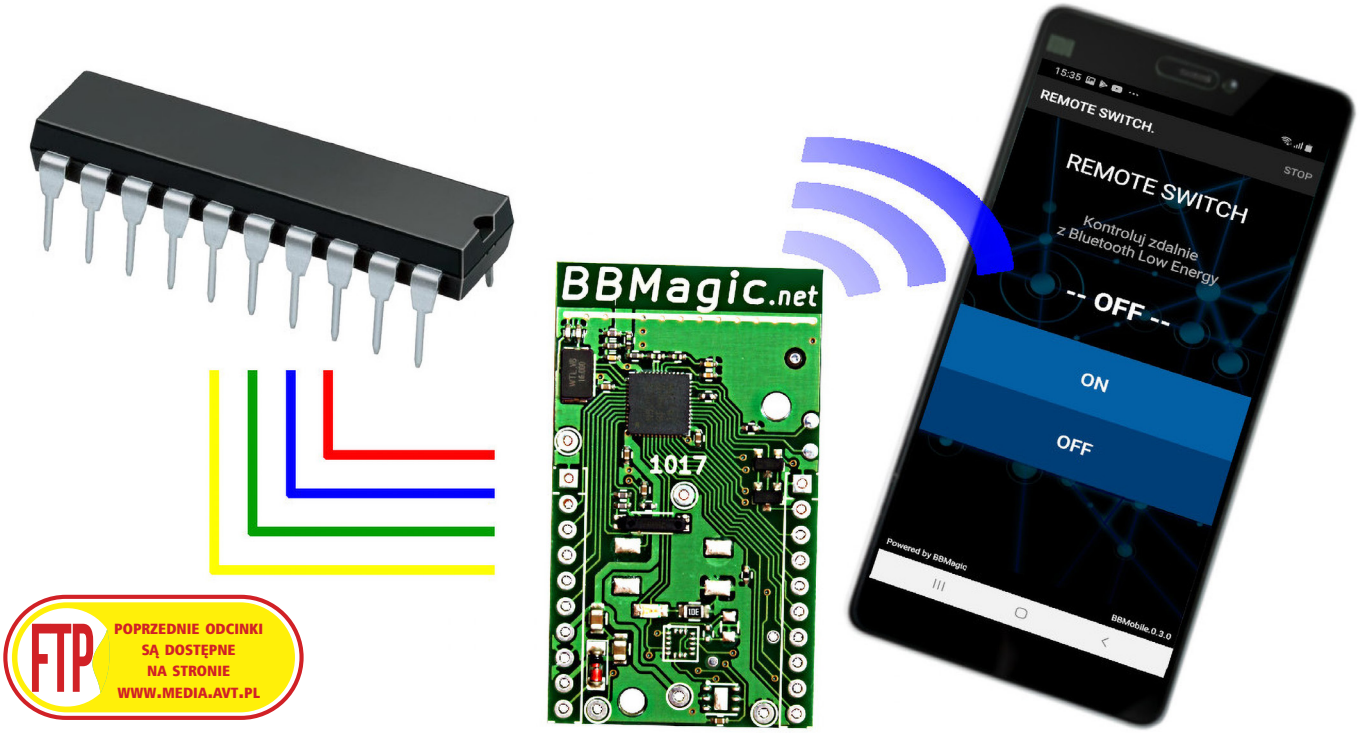
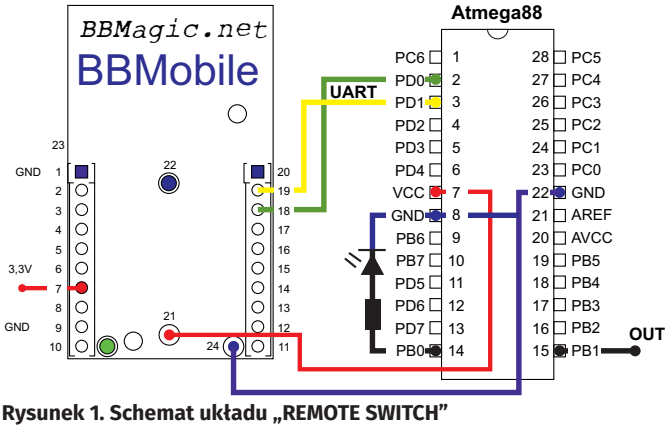


FTP POPRZEDNIE ODCINKI SĄ DOSTĘPNE NA STRONIE WWW.MEDIA.AVT.PL

Jak napisać aplikację mobilną poprzez UART (4)

W kolejnej części artykułu BBMobile, zajmiemy się oprogramowaniem mikrokontrolera, będącego sercem sterownika „REMOTE SWITCH”. Zbudujemy uniwersalną strukturę programu, która nieznacznie zmodyfikowana pozwoli zrealizować wiele nowych, ciekawych i użytecznych urządzeń sterujących: włączników czasowych, termostatów, sterowników kolektorów słonecznych i bojlerów, włączników z opóźnionym startem, sterowników wentylatorów itp. Wszystkie te urządzenia łączy wspólny element: bezprzewodowy interfejs użytkownika na ekranie urządzenia mobilnego.



Rysunek 1. Schemat układu „REMOTE SWITCH”

Hardware projektu

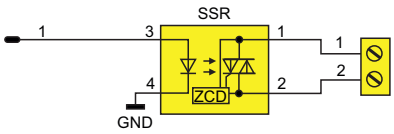
Sterownik „REMOTE SWITCH” składa się z trzech głównych elementów: mikrokontrolera sterującego, modułu BBMobile (most rozpięty pomiędzy interfejsem UART kontrolera, a interfejsem użytkownika na ekranie smartfona) oraz wykonawczego układu mocy. Jako układ programowalny wybrano chip ATmega88. Może go zastąpić dowolny inny kontroler wyposażony w port UART, z co najmniej 8 kB pamięci programu i 2 kB pamięci RAM (co nie powinno sprawić większych problemów).

Schemat układu prezentuje **rysunek 1**. Komunikacja szeregowa pomiędzy mikrokontrolerem a modułem BBMobile, odbywa się dwoma liniami portu UART (PD.0 i PD.1 kontrolera). Obydwa komponenty zasilane są tym samym napięciem o wartości 3,3 V, które dołączone jest do pinów 7 i 8 modułu BBMobile. Zasilanie dla ATmega88 pobierane jest natomiast z pinów 21 i 24. Do pinu PB.0 kontrolera podłączono diodę sygnalizującą stan pracy sterownika, a do pinu PB.1 należy dołączyć wspomniany wykonawczy układ mocy. W zależności od zastosowania sterownika, może to być przekaźnik półprzewodnikowy SSR lub układ wykonawczy z triakiem (**rysunek 2**), tranzystor mocy (**rysunek 3**) lub przekaźnik mechaniczny (**rysunek 4**). Do złącza śrubowego podłączamy sterowane urządzenie, a wejście układu wykonawczego podpinamy do PB.1 mikrokontrolera.

Biblioteka bbmobile_lib

Obsługa modułu BBMobile została zaimplementowana w dedykowanej bibliotece, której kod źródłowy dostępny jest w witrynie *bbmagic.net*. Składa się z dwóch plików: *bbmobile_uart04.c* i *bbmobile_lib04.c*. Pierwszy plik zawiera funkcje obsługujące hardware portu szeregowego mikrokontrolera. Wymaga edycji w przypadku chęci zastosowania biblioteki z innym chipem. Dostosować należy zmienną kompilatora UART_DATA_REG (definiuje nazwę rejestru danych UART) oraz dwie funkcje:

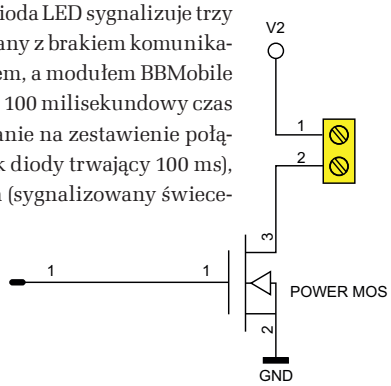
`bbmobile_uart_init`, konfigurującą sterownik UART i `bbmobile_uart_tx_wait_free`, oczekującą na możliwość wysłania kolejnego znaku UART-em. Drugi z plików zawiera funkcje obsługujące moduł BBMobile oraz stworzony interfejs mobilny i nie wymaga edytowania. W **tabeli 1** zostały zestawione funkcje biblioteki wraz z ich krótkim opisem.



Rysunek 2. Przekaźnik SSR

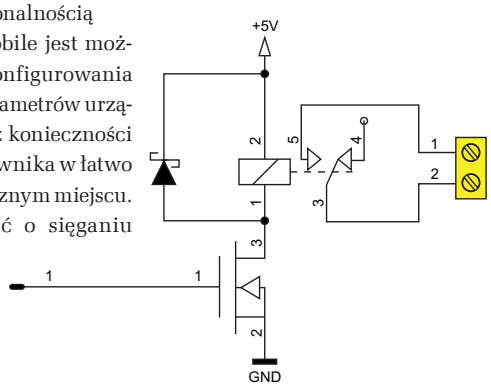
Funkcjonalność projektu

Demonstracyjny projekt „REMOTE SWITCH" umożliwia zdalne załączanie i wyłączanie dowolnego sprzętu elektronicznego, przy pomocy urządzenia mobilnego (smartfon, tablet). Dostęp do sterowania zabezpieczony będzie kodem PIN. Dioda LED sygnalizuje trzy stany sterownika: błąd związany z brakiem komunikacji pomiędzy mikrokontrolerem, a modułem BBMobile (długi okres świecenia i krótki 100 milisekundowy czas wygaszenia LED-u), oczekiwanie na zestawienie połączenia Bluetooth (krótki błysk diody trwający 100 ms), stan zestawionego połączenia (sygnalizowany świeceniem diody).



Rysunek 3. Tranzystor mocy

Na bazie tego projektu możliwe jest zbudowanie wielu ciekawych i użytecznych urządzeń. Poza prostotą tworzenia estetycznych wizualnie interfejsów użytkownika. Dodatkową funkcjonalnością projektów z BBMobile jest możliwość zdalnego konfigurowania i odczytywania parametrów urządzenia. Nie ma już konieczności instalowania sterownika w łatwo dostępnym i widocznym miejscu. Można zapomnieć o sięganiu do pokręteł i przycisków sterujących na panelu oraz wytężaniu wzroku, by odczytać wskazania.



Rysunek 4. Przekaźnik mechaniczny

Rysunek 5. Skanowanie w poszukiwaniu urządzeń

Rysunek 6. Funkcje biblioteki BBMobile

Wystarczy uruchomić aplikację i wszystko czego potrzebujemy, wyświetlone będzie na kolorowym ekranie.

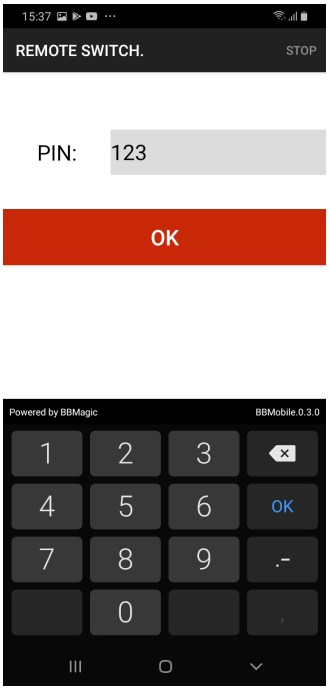
Projekt programu sterującego

Struktura programów dla mikrokontrolerów korzystających z modułu BBMobile, jest dość podobna. Wynika to ze sposobu użytkowania urządzeń z mobilnym interfejsem. Użytkownik wykonuje czynności, których obsługa jest zaimplementowana w kontrolerze sterującym tj. włączenie aplikacji BBMobile i uruchomienie skanowania w poszukiwaniu dostępnych peryferii (**rysunek 5**), zestawienie połączenia z wybranym z listy urządzeniem (**rysunek 6**), wprowadzenie kodu PIN umożliwiającego dostęp jeśli jest wymagany (**rysunek 7**), odczytywanie wartości i wprowadzanie zmian parametrów lub sterowanie urządzeniem (**rysunek 8**).

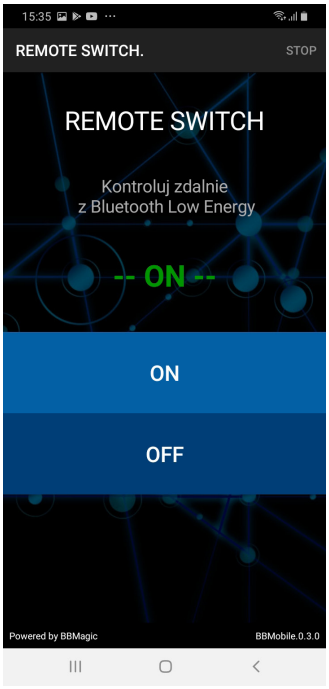
Program sterujący można podzielić na trzy intuicyjne bloki funkcjonalne: inicjalizację kontrolera i konfigurację wszystkich układów peryferyjnych wraz z modułem BBMobile (aby ten był gotowy do przyjęcia nadchodzących połączeń), oczekiwanie na zestawienie połączenia Bluetooth (a gdy to nastąpi stworzenie interfejsu użytkownika na ekranie), współpraca z interfejsem aż do chwili zamknięcia połączenia (obsługiwanie żądań i wysyłanie komend sterujących). Dwa ostatnie bloki wykonywane są w nieskończonej pętli, ponieważ po zamknięciu aktywnego połączenia, sterownik musi oczekiwać na zestawienie kolejnego i realizację nowych zadań.

Tabela 1. Funkcje biblioteki bbmobile

Funkcja	Opis	Przykład użycia
bbmobile_check_ble_conn	Sprawdza czy bezprzewodowe połączenie z urządzeniem mobilnym jest zestawione.	con = bbmobile_check_ble_conn() ;
bbmobile_is_data	Sprawdza czy nadeszły nowe dane od aplikacji mobilnej. Jeśli nadeszły, to funkcja zwraca wartość różną od zera, a dane znajdują się w buforze bbm_buf[].	if(bbmobile_is_data())
bbmobile_send_json	Wysyła do aplikacji mobilnej opis struktury interfejsu w postaci kodu JSON.	bbmobile_send_json(json_switch) ;
bbmobile_send_2chip_ack	Wysyła komendę do modułu BBMobile i oczekuje na odpowiedź: >OK lub >ERR. Wykorzystywana do ustawiania parametrów modułu.	bbmobile_send_2chip_ack("<hello\r\n") ;
bbmobile_send_2app_ack	Wysyła komendę do aplikacji i oczekuje na odpowiedź: $ok lub $nok. Służy do komunikacji z interfejsem użytkownika.	bbmobile_send_2app_ack("$set,t1:te=\"-- ON --\"tc=\"0,150,0\"\r\n") ;
bbmobile_send_dec_2app	Przesyła do interfejsu liczbę uint16_t w postaci dziesiętnej.	bbmobile_send_dec_2app(licznik,0) ;
bbmobile_send_sigdec_2app	Przesyła do interfejsu liczbę int16_t w postaci dziesiętnej ze znakiem.	bbmobile_send_sigdec_2app(napiecie,0) ;
bbmobile_send_hex_2app	Przesyła do interfejsu bajt w postaci szesnastkowej. Służy do prezentacji danych w formie hex.	bbmobile_send_hex_2app(hex_data) ;
cmp_flash_ram_bytes	Porównuje określoną liczbę znaków w buforze z podanym wzorcem. Wykorzystywana do walidacji komunikatów pochodzących z interfejsu użytkownika.	if(cmp_flash_ram_bytes("$by,b1", bbm_buf, 6))



Rysunek 7. Wprowadzanie pinu



Rysunek 8. Interfejs sterujący

Listing 1. Inicjalizacja programu

```
48  //=========================================
49  //-MAIN
50  //=========================================
51  int main(void)
52  {
53      uint8_t i, pk_state ;
54
55      DDRB = 0b00000011 ;
56      PORTB = 0b00000011 ;
57      DDRD = 0b00000010 ;
58      PORTD = 0b00001111 ;
59
60      PK_OFF ;
61      pk_state =0 ;
62
63      //bbmobile_uart_init((uint16_t)(F_CPU/(9600*16l)-1)) ;
        //-UBRR reg with double UART speed -> 960bps
64      bbmobile_uart_init(12) ;
65
66      sei() ; //-enable interrupts
67
68      //-wait for BBMobile module answer
69      do
70      {
            //-it takes about 500ms if timeout
71          if( bbmobile_send_2chip_ack("<hello\r\n") ==0 )
72              break ;
73          LED_OFF ; bbmobile_wait_ms(100) ; LED_ON ;
74      }while(1) ;
75
76      //-set Bluetooth device name
77      i = bbmobile_send_2chip_ack("<name,REMOTE SWITCH\r\n") ;
78
79      //-set PIN
80      i = bbmobile_send_2chip_ack("<pin,123\r\n") ;
```

Program sterujący – inicjalizacja

Na **listingu 1** został pokazany kod inicjujący sterownik „REMOTE SWTICH". Linie od 55...58 konfigurują porty mikrokontrolera, zgodnie z opisanym powyżej przeznaczeniem. W stanie początkowym sterowane urządzenie jest wyłączone, a flaga `pk_state` wyzerowana, o czym decydują linie 60 i 61. W linii 64 konfigurowany jest port UART do pracy z prędkością 9600 bps, przy wewnętrznym zegarze kontrolera 1 MHz. Linia 66 włącza obsługę przerwań, co jest niezbędne do poprawnej pracy biblioteki BBMobile.

Pętla rozpoczynająca się od linii 69 sprawdza czy komunikacja pomiędzy kontrolerem a modułem, działa w sposób niezakłócony. W tym celu wysyłana jest komenda `<hello` na którą poprawnie działający BBMobile odpowiada pozdrowieniem `>HI`. Zwróćmy uwagę, że każdy obieg pętli powoduje przygaśnięcie diody na 100 ms, co pozwala zidentyfikować ewentualny problem braku komunikacji pomiędzy modułem, a kontrolerem. Dłuższy okres świecenia diody z krótkimi przygaśnięciami, sygnalizuje właśnie ten problem. Dopiero gdy funkcja `bbmobile_send_2chip_ack("<hello\r\n")` zwróci wartość różną od zera, pętla zostaje przerwana, a w linii 77 definiowana jest nazwa urządzenia, która widoczna będzie na ekranie urządzenia mobilnego podczas skanowania. Kod dostępu do sterownika ustawiony jest wywołaniem kolejnej funkcji w linii 80, a jej pominięcie spowoduje, że „REMOTE SWITCH" nie będzie zabezpieczony hasłem.

Program sterujący – nieskończona pętla główna

Na **listingu 2** pokazano nieskończoną główną pętlę programu, która realizuje: oczekiwanie na zestawienie połączenia, wysłanie kodu JSON budującego interfejs użytkownika (po otwarciu kanału komunikacyjnego), aktualizację kontrolek interfejsu, obsługę interfejsu (reakcję na żądania użytkownika aż do chwili zamknięcia połączenia Bluetooth).

Przyjrzyjmy się jak poszczególne zadania zostały zrealizowane przy użyciu zasobów biblioteki *bbmobile_lib*. Pierwsza pętla mająca swój początek w linii 85, używa funkcji `bbmobile_check_ble_conn()` do sprawdzania stanu połączenia z aplikacją mobilną. Pulsująca dioda zaświecana jest w tym stanie na 100 ms, co sygnalizuje poprawną pracę sterownika i oczekiwanie na nadchodzące połączenia. Jeżeli kanał komunikacyjny z aplikacją mobilną zostanie otwarty, dioda świeci na stałe. Funkcja `bbmobile_send_json(json_switch)` wyśle kod opisujący strukturę interfejsu, zawarty w tablicy `json_switch`. Na **listingu 3** widoczna jest definicja

Listing 2. Nieskończona pętla

```
82      for(;;)     //-do it forever
83      {
84          //-wait for BLE connection
85          while( bbmobile_check_ble_conn() ==0 )
86          {
87              LED_OFF ; bbmobile_wait_ms(900) ;
88              LED_ON ; bbmobile_wait_ms(100) ;
89          } ;
90
91          //-send JSON structure to build mobile interface
92          bbmobile_send_json(json_switch) ;
93
94          //-update status
95          if(pk_state)    //-if switch is on
96              bbmobile_send_2app_ack("$set,t1:te=\"-- ON --\"tc=\"0,150,0\"\r\n") ;
97          else            //-if switch is off
98              bbmobile_send_2app_ack("$set,t1:te=\"-- OFF --\"tc=\"255,255,255\"\r\n") ;
99
100         //-play with interface
101         while(bbmobile_check_ble_conn())    //-until BLE connection is established
102         {
103             if( bbmobile_is_data() )        //-if there is new data from BBMobile
104             {
105                 if(cmp_flash_ram_bytes("$by,b1", bbm_buf, 6))       //-if ON button is pressed
106                 {
107                     PK_ON ;
108                     pk_state =1 ;
109                     bbmobile_send_2app_ack("$set,t1:te=\"-- ON --\"tc=\"0,150,0\"\r\n") ;
110                 }else if(cmp_flash_ram_bytes("$by,b2", bbm_buf, 6))//-if OFF button is pressed
111                 {
112                     PK_OFF ;
113                     pk_state =0 ;
114                     bbmobile_send_2app_ack("$set,t1:te=\"-- OFF --\"tc=\"255,255,255\"\r\n") ;
115                 }
116                 bbmobile_rx_enable() ;  //-enable rec new data from BBMobile
117             }
118         } ;
119         //-BLE connection is closed
120         LED_OFF ;
121     }
122
123     return(0) ;
124 }
```

Listing 3. Konfiguracja programu z pliku JSON

```
13  #include <avr/pgmspace.h>
14  #include <avr/io.h>
15  #include <avr/interrupt.h>
16
17  #include "bbmobile_lib04.c"
18
19  #define LED_PORT         PORTB
20  #define LED_PIN              0
21  #define LED_ON               LED_PORT |= (1<<LED_PIN)
22  #define LED_OFF              LED_PORT &= ~(1<<LED_PIN)
23
24  #define PK_PORT              PORTB
25  #define PK_PIN               1
26  #define PK_ON                PK_PORT |= (1<<PK_PIN)
27  #define PK_OFF               PK_PORT &= ~(1<<PK_PIN)
28
29
30  //-JSON structure - mobile interface definition
31  const char json_switch[] PROGMEM ="{\"ty\":\"lout\",\"or\":\"V\",\"img\":\"1\",\"cs\":[         \
32  {\"ty\":\"TextView\",\"te\":\"\nREMOTE SWITCH\",\"tc\":\"255,255,255\",\"ts\":\"30\",\"w\":\"1\"},         \
33  {\"ty\":\"TextView\",\"te\":\"Kontroluj zdalnie\n z Bluetooth Low Energy\",\"tc\":\"174,174,174\",\"ts\":\"20\",\"w\":\"1\"}, \
34  {\"ty\":\"TextView\",\"n\":\"t1\",\"tc\":\"255,255,255\",\"ts\":\"35\",\"tl\":\"bold\",\"w\":\"1\"},       \
35  {\"ty\":\"Button\",\"n\":\"b1\",\"te\":\"ON\",\"tc\":\"255,255,255\",\"bg\":\"3,96,164\",\"ts\":\"25\"},   \
36  {\"ty\":\"Button\",\"n\":\"b2\",\"te\":\"OFF\",\"tc\":\"255,255,255\",\"bg\":\"0,62,120\",\"ts\":\"25\"}, \
37  {\"ty\":\"TextView\",\"w\":\"3\"}]}" ;
```

Listing 4. Kod opisujący layout aplikacji

```
{
"ty":"lout",
"or":"V",
"img":"1",
"cs":[
    {
    "ty":"TextView",
    "te":"REMOTE SWITCH",
    "tc":"255,255,255",
    "ts":"30",
    "w":"1"
    },
    {
    "ty":"TextView",
    "te":"Kontroluj zdalnie\n z Bluetooth Low Energy",
    "tc":"174,174,174",
    "ts":"20",
    "w":"1"
    },
    {
    "ty":"TextView",
    "n":"t1",
    "te":"-- OFF --",
    "tc":"255,255,255",
    "ts":"35",
    "tl":"bold",
    "w":"1"
    },
    {
    "ty":"Button",
    "n":"b1",
    "te":"ON",
    "tc":"255,255,255",
    "bg":"3,96,164",
    "ts":"25"
    },
    {
    "ty":"Button",
    "n":"b2",
    "te":"OFF",
    "tc":"255,255,255",
    "bg":"0,62,120",
    "ts":"25"
    },
    {
    "ty":"TextView",
    "w":"3"
    }
    ]
}
```

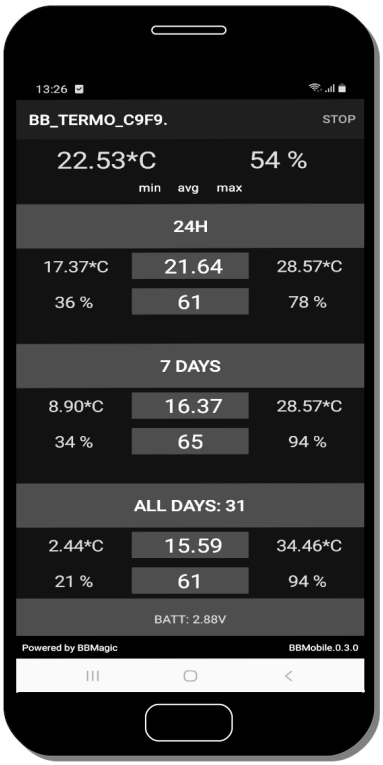
tablicy, zapisanej w pamięci programu. Jest to zaimplementowany przez nas kod JSON z **listingu 4**, który przygotowany był na etapie projektowania interfejsu w poprzednich częściach kursu. Tutaj został nieco zoptymalizowany pod kątem zajmowanego w pamięci miejsca. Ponadto, aby możliwa była kompilacja kodu wszystkie znaki " zostały escapeowane – zastąpione przez \".

Po zbudowaniu interfejsu, następuje uaktualnienie parametrów kontrolek. W sterowniku „REMOTE SWITCH" konieczny jest update tekstu informującego o aktualnym stanie wysterowania (urządzenie włączone lub urządzenie wyłączone). Kod w liniach 95...98 sprawdza stan flagi `pk_state` i wysyła do interfejsu za pomocą funkcji `bbmobile_send_2app_ack()` odpowiednią komendę, która wyświetli biały napis `OFF` lub czerwony `ON`.

Interfejs na ekranie jest wyświetlony i zaktualizowany, a program zostaje zamknięty w kolejnej pętli, która wykonywana będzie dopóki radiowe połączenie z aplikacją jest otwarte (linia 101). W pętli tej sprawdzamy cyklicznie, czy nadeszły nowe dane od interfejsu (linia 104). Jeśli tak, to reagujemy w zależności od odebranego komunikatu: włączamy sterowane urządzenie, gdy użytkownik dotknął guzika b1 (`if(cmp_flash_ram_bytes("$by,b1", bbm_buf, 6))`) i wyłączamy, gdy dotknął b2 (`else if(cmp_flash_ram_bytes("$by,b2", bbm_buf, 6))`). Za każdym razem aktualizujemy wartość flagi `pk_state` oraz parametry kontrolki TextView, wyświetlającej aktualny stan – napis i jego kolor w kontrolce t1. Funkcja `cmp_flash_ram_bytes("$by,b2", bbm_buf, 6)` porównuje sześć bajtów znajdujących się w buforze `bbm_buf` w pamięci RAM, z ciągiem podanym jako pierwszy argument funkcji. Jeśli ciągi znaków są różne, funkcja zwraca wartość zero, w przeciwnym wypadku jeden.

Krytyczna linia kodu 116 wywołująca funkcję `bbmobile_rx_enable()`, umożliwia odbieranie kolejnych komunikatów poprzez port UART. Bez jej wywołania, reagowanie na kolejne komendy płynące z interfejsu użytkownika byłoby niemożliwe, co spowodowałoby wrażenie zawieszenia się urządzenia.

Po dotknięciu przycisku „STOP" na ekranie interfejsu aplikacji mobilnej, połączenie Bluetooth zostanie zamknięte, a funkcja `bbmobile_check_ble_conn()` w linii 101 zwróci wartość zero. Spowoduje to zakończenie wykonywania pętli zawartej w liniach 101...118. Program wróci do linii 85 i ponownie rozpocznie oczekiwanie na kolejne połączenie, mrugając diodą LED.



Rysunek 9. Przykład innego zastosowania biblioteki BBMobile

Ewolucja

Ponieważ kod źródłowy projektu jest ogólnodostępny w witrynie *bbmagic.net*, a jego budowa jest prosta i przejrzysta, to z łatwością można poddać go modyfikacji, rozbudowując funkcjonalność, bądź całkowicie ją zmieniając. Aby stworzyć nowy interfejs użytkownika wystarczy zmienić zawartość tablicy z listingu 3. Można zmienić zabezpieczający kod PIN, edytując argument funkcji w linii 80 lub usunąć go, pomijając jej wywołanie. Zaprogramować reakcje na naciśnięcie kolejnych button-ów, możemy dodając kolejne warunki, rozpoczynając od linii 115. W ten sposób, w ciągu kilkunastu minut może powstać całkowicie nowy projekt, z interfejsem mobilnym na ekranie smartfona, jak na przykład na **rysunku 9**.

Mariusz Żądło
iram@poczta.onet.pl