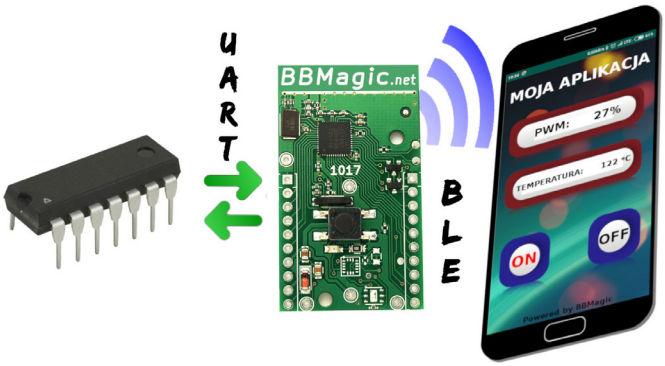
Jak napisać aplikację mobilną poprzez... UART (1)

W trakcie procesu tworzenia urządzeń elektronicznych projektanci napotykają szereg zdawałoby się nierozwiązywalnych problemów, z którymi muszą sobie sprawnie poradzić. Wracającym niezwykle często, niczym bumerang kłopotem jest zorganizowanie interfejsu użytkownika (HMI – Human Machine Interface). Wymagania są niejednokrotnie sprzeczne: wysoki poziom estetyki, ergonomia użytkowania i elastyczność funkcjonalna w opozycji do minimalnych kosztów oraz krótkiego czasu wdrożenia. Skłania to do poszukiwania nowych rozwiązań będących optymalnym kompromisem rozcinającym ten gordyjski węzeł.

Ciekawym pomysłem jest użycie dotykowych ekranów urządzeń mobilnych w roli HMI. Idea zobrazowana na **rysunku 1** jest niezwykle oczywista: programowanie i obsługa bezprzewodowych aplikacji mobilnych poprzez nieskomplikowany, popularny interfejs UART. Takie rozwiązanie otworzyłoby na oścież zamknięte dotąd drzwi do estetycznych i zaawansowanych interfejsów graficznych dla niemal każdego projektu opartego o mikrokontroler. Jeśli dołożyć do tego bezprzewodowość i niezwykłą energooszczędność układu z Bluetooth SMART to można pokusić się o stwierdzenie, że mamy mały przełom.

Sprzęt

Schemat modułu pokazuje **rysunek 2**. Mikrokontroler z Bluetooth Low Energy i interfejsem UART (typu NRF51822) realizuje funkcję mostu

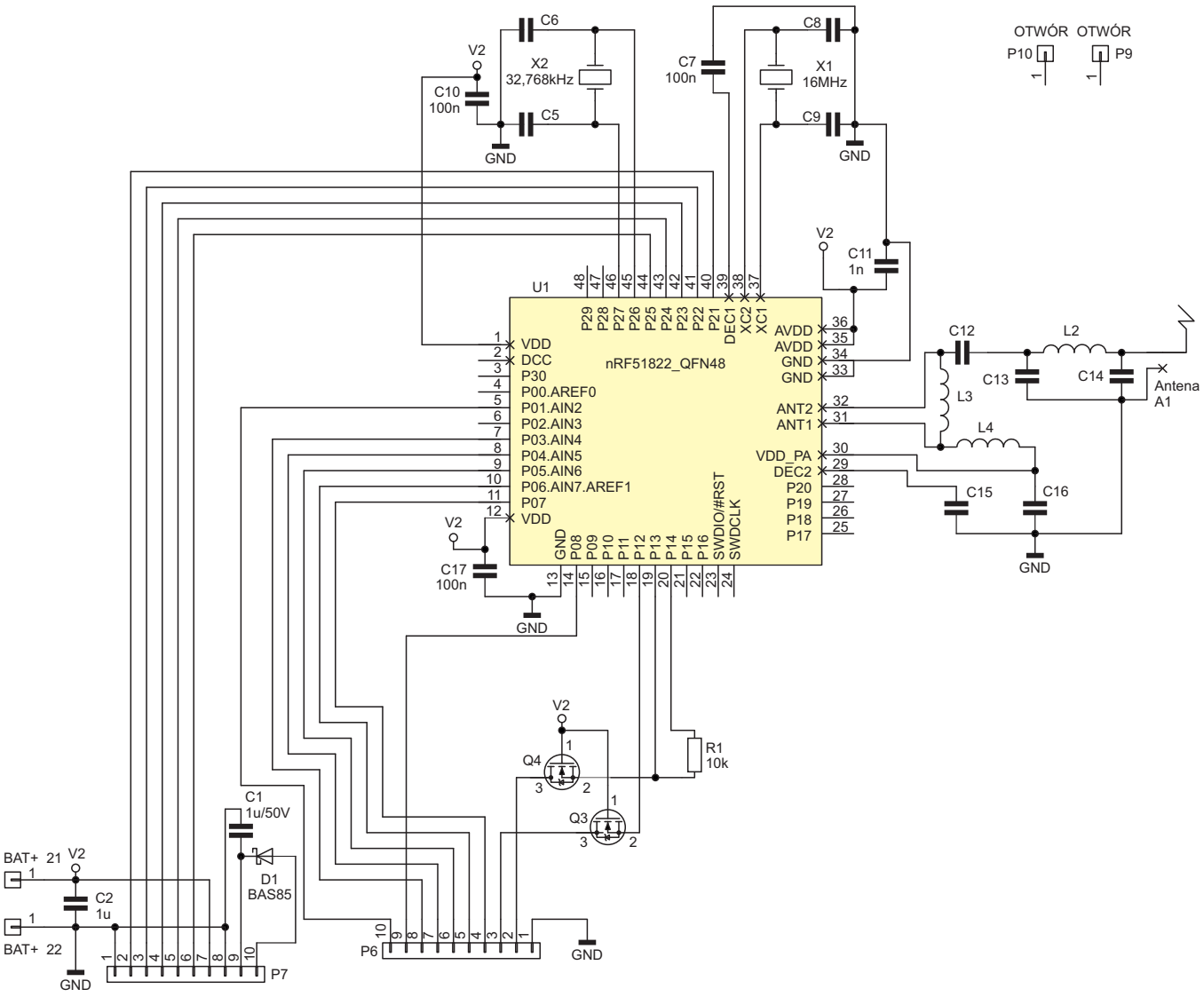


Rysunek 1. Idea zastosowania modułu

pomiędzy dowolnym mikrokontrolerem posiadającym funkcjonalność asynchronicznej komunikacji szeregowej (UART), a urządzeniem mobilnym z interfejsem Bluetooth w wersji minimum 4.0 (smartfon, tablet).

Zasilanie projektowanego systemu można zorganizować na kilka sposobów:

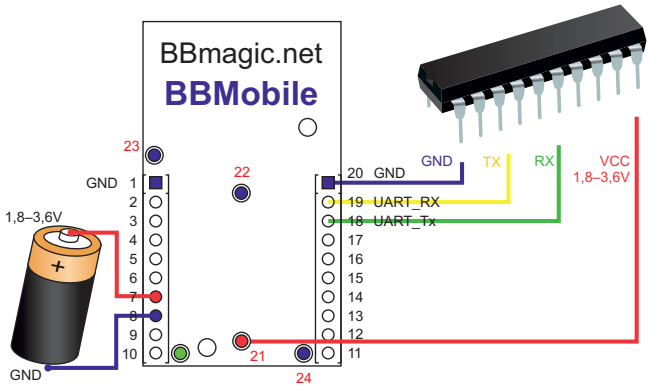
1. W aplikacjach ultra low power (**rysunek 3**) – bezpośrednio z baterii podłączonej do pinów 7 i 8 złącza P7. Możliwe jest też podłączenie źródła energii do padów 21 i 22 (np. wlutowanie gniazda pastylkowej baterii CR 2032). Zakres możliwych napięć zasilających waha się od 1,8 do 3,6 V.
2. W przypadku gdy mikrokontroler potrzebuje napięcia 5 V (**rysunek 4**) – zastosowanie stabilizatora na napięcie 3,3 V zainstalowanego w złączu P7 (pin 7 – wyjście stabilizatora, pin 8 – masa stabilizatora, pin 9 – wejście stabilizatora).



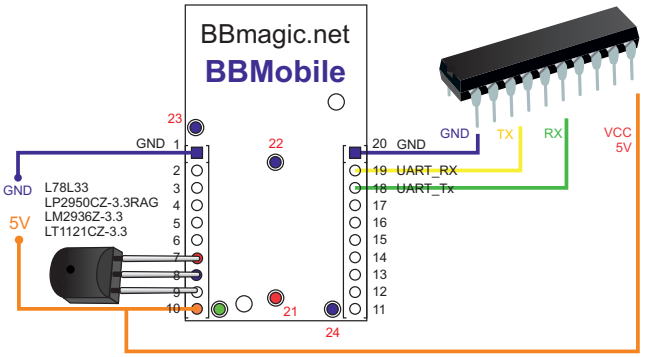
Rysunek 2. Schemat modułu

3. W aplikacjach gdzie dostępne są jedynie wyższe napięcia (**rysunek 5**) – przez stabilizator podłączony jak poprzednio. Jeśli użyć np. L78L33, LP2950CZ3.3RAG, LM2936Z-3.3 lub LT1121CZ-3.3 to do pinu 10 złącza P7 możemy podłączyć napięcie w granicach od 5 do 30 V. Wejście to jest zabezpieczone przed pomyłką odwrotnej polaryzacji za pomocą diody D1.

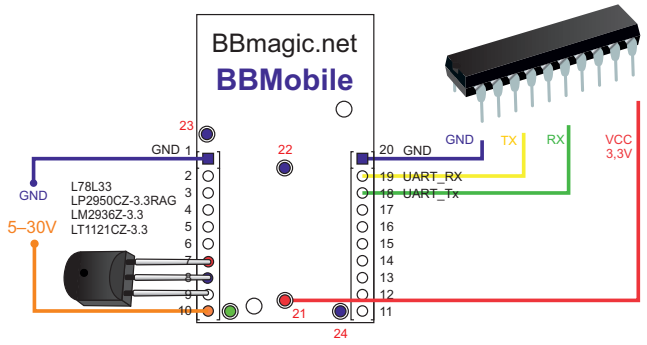
W komunikacji UART pośredniczą unipolarne tranzystory Q3 i Q4 umożliwiając dopasowanie poziomów napięć jeśli byłyby one różne w układzie mikrokontrolera i modułu. Dzięki nim port komunikacyjny można bez obawy łączyć z systemami zasilanymi napięciem 5 V, np. Arduino.



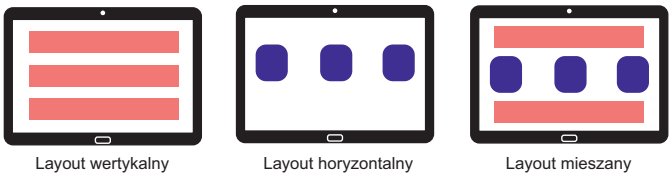
Rysunek 3. Konfiguracja dla aplikacji ultra low power



Rysunek 4. Konfiguracja dla mikrokontrolera zasilanego napięciem 5 V



Rysunek 5. Konfiguracja dla aplikacji zasilanych napięciem powyżej 5 V



Rysunek 6. Przykłady layoutów

Software

Ogólny schemat działania nie jest skomplikowany:

- uruchamiamy aplikację BBMobile,
- klikamy przycisk 'START' – na ekranie pojawia się lista dostępnych urządzeń znajdujących się w zasięgu,
- wybieramy urządzenie z którym chcemy zestawić połączenie,
- w wyniku nawiązania łączności na ekranie smartfona buduje się interfejs użytkownika – przyciski, pola tekstowe, przełączniki i inne kontrolki zgodnie z poleceniami przesłanymi przez port szeregowy UART modułu (rysunek 1).

Nasuwa się pytanie – w jaki sposób „upchnąć" tak złożony i wielowymiarowy proces jak tworzenie i obsługa aplikacji do tak prostego interfejsu jak UART? Aplikacja składa się przecież z głównego okna (layout) w którym obiekty (kontrolki) mogą być różnie rozmieszczone w zależności od potrzeb funkcjonalnych konkretnej aplikacji (**rysunek 6**): wertykalnie – pionowo jedna pod drugą lub horyzontalnie – obok siebie. Mogą też oczywiście występować dowolne kombinacje tych dwóch układów.

Dostępne są kontrolki realizujące wiele funkcji: pola wyświetlające informacje (TextView), przyciski (Button), paski postępu (ProgBar), edytowalne pola tekstowe (EditText), pola wyboru (Spinner) i inne. Każda kontrolka posiada swoje własne cechy jak np. rozmiar, treść wyświetlanego tekstu, kolor tła, kolor tekstu, rozmiar czcionki itd. Cechy te muszą zostać zdefiniowane podczas tworzenia interfejsu, a potem mogą być modyfikowane podczas pracy aplikacji. Dodatkowo kontrolki wyjściowe takie jak buttony, switche czy edytowalne pola tekstowe i numeryczne muszą mieć możliwość przekazywania informacji o akcjach wykonywanych przez użytkownika. Wpisanie danych do pola edytowalnego, przełączenie kontrolki switch czy naciśnięcie buttona powinno powodować przesłanie przez UART informacji jednoznacznie identyfikującej kontrolkę i wywołaną akcję.

Wobec tak szerokiego spektrum możliwych działań konieczne jest podzielenie komunikacji na elementy:

1. Po włączeniu zasilania modułu BBMobile, gdy połączenie Bluetooth SMART nie jest zestawione (lub po zamknięciu połączenia) komunikacja umożliwia ustawienie lub zmianę parametrów pracy. Dokonujemy tego wysyłając komendy rozpoczynające się znakiem '<' i kończące znakami nowej linii i powrotu karetki: '\n' i '\r'.
2. Po zestawieniu połączenia Bluetooth SMART konieczne jest wysłanie w formacie JSON opisu struktury interfejsu, który ma zostać stworzony na ekranie. Blok tych danych rozpoczyna się znakiem '{' i kończy znakiem '}'.
3. Gdy interfejs użytkownika zostanie już zbudowany na ekranie urządzenia mobilnego, komunikacja jest kontynuowana za pomocą krótkich komend rozpoczynających się od znaku '$' i kończących się znakami nowej linii i powrotu karetki: '\n' i '\r'.

JSON

JavaScript Object Notation czyli JSON to lekki, tekstowy format wymiany danych pochodzący od języka Java Script, który w czytelny dla człowieka sposób porządkuje informacje w struktury zwane obiektami. Ponadto:

- obiekty zamknięte są w nawiasach pazurkowych '{...}',
- dane wewnątrz obiektów zorganizowane są w pary "atrybut":"wartość",
- obiekty oraz pary "atrybut":"wartość" rozdzielone są przecinkami,
- obiekty mogą być zagnieżdżane,
- dane typu tablicowego zamknięte są w nawiasach kwadratowych '[...]',
- wielkość liter, spacje, tabulacje i znaki końca linii (enter) nie mają znaczenia informacyjnego, a używane są jedynie dla uzyskania większej przejrzystości kodu.

Dla lepszego zrozumienia formatu JSON przeanalizujmy element interfejsu jakim jest kontrolka TextView. Jest to pole tekstowe, którego zawartości użytkownik nie może modyfikować. Służy po prostu do wyświetlenia informacji. Oto jak pary "atrybut":"wartość" definiują taką kontrolkę jako obiekt JSON zawarty wewnątrz nawiasów:

```
{ „ty":"TextView", „n":"text_1", „te":"HELLO WORLD",
„bg":"0,0,0" }
```

Pierwszy nawias '{' i ostatni nawias '}' obejmują całą strukturę obiektu – oznaczają jego początek i koniec. Kolejne rozdzielone przecinkami pary "atrybut":"wartość" mają następujące znaczenie:

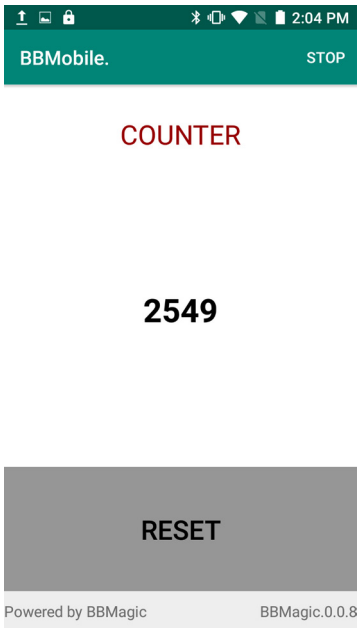
- „ty": „TextView" – definiuje typ kontrolki – w tym przykładzie jest to TextView; skrót od type:TextView,
- „n":"text_1" – określa unikalną, wybraną przez projektanta nazwę obiektu pozwalającą odwołać się do niego podczas działania aplikacji, skrót od name:text_1,
- „te":"HELLO WORLD" – definiuje tekst, który ma zostać wyświetlony w polu kontrolki, skrót od: text: HELLO WORLD,
- „bg":"0,0,0" – określa składowe RGB koloru tła kontrolki. Tutaj wybrano kolor czarny; skrót od: background:0,0,0.

Struktura JSON opisująca layout

Pokazany powyżej krótki kod zdefiniował jedną kontrolkę. Cały interfejs użytkownika tworzony na ekranie może zawierać wiele kontrolek. Konieczne jest zatem niewielkie rozbudowanie kodu JSON, aby zdołał opisać nie tylko same kontrolki, ale również ich układ. Na **listingu 1** pokazano najprostszy, kompletny kod definiujący layout, który zawiera dwie kontrolki: TextView oraz Button ułożone horyzontalnie, jedna obok drugiej. **Rysunek 7** pokazuje ekran z interfejsem utworzonym po przesłaniu kodu do modułu.

Kod jako obiekt JSON ograniczony jest oczywiście nawiasami '{}'. W drugiej linii zdefiniowano jego typ – jest to layout, czyli obiekt zawierający inne obiekty (jak stwierdziliśmy wcześniej, obiekty mogą być zagnieżdżone). Komponenty wewnątrz layoutu będą w układzie horyzontalnym o czym informuje linia 3. W linii 4 określono tło layoutu jako czarne.

Rysunek 7. Wygląd okna aplikacji z listingu 1



Rysunek 8. Wygląd okna aplikacji z listingu 2

Listing 1. Kod JSON definiujący layout horyzontalny z dwiema kontrolkami: TextView i Button

```
1   {
2   „ty”: „lout”,
3   „or”:”H”,
4   „bg”:”0,0,0”,
5   „cs”:[
6       {
7       „ty”:”TextView”,
8       „n”:”tv_1”,
9       „te”:”Hello World”,
10      „w”:”2”,
11      „tc”:”255,255,255”,
12      „bg”:”0,0,0”,
13      „ts”:”40”,
14      „tl”:”bold”
15      },
16      {
17      „ty”:”Button”,
18      „n”:”btn_1”,
19      „te”:”OK”,
20      „w”:”1”,
21      „tc”:”0,0,0”,
22      „bg”:”254,127,0”,
23      „ts”:”30”,
24      „tl”:”bold”
25      }
26      ]
27  }
```

Listing 2. Definicja layoutu wertykalnego z trzema kontrolkami

```
{
    „ty”: „lout”,
    „or”:”V”,
    „bg”:”255,255,255”,
    „cs”:[
    {
        „ty”:”TextView”,
        „n”:”tv_1”,
        „te”:”\nCOUNTER”,
        „w”:”2”,
        „tc”:”150,0,0”,
        „ts”:”25”
    },
    {
        „ty”:”TextView”,
        „n”:”tv_1”,
        „te”:”2549”,
        „w”:”2”,
        „tc”:”0,0,0”,
        „ts”:”30”,
        „tl”:”bold”
    },
    {
        „ty”:”button”,
        „n”:”btn_1”,
        „te”:”RESET”,
        „w”:”1”,
        „tc”:”0,0,0”,
        „bg”:”150,150,150”,
        „ts”:”25”
    }
    ]
}
```

Linia piąta rozpoczyna definicję tablicy obiektów znajdujących się wewnątrz layoutu głównego. Tablice komponentów definiowane są w następujący sposób:

```
cs:[ {obiekt_1}, {obiekt_2}, ... , {obiekt_n} ]
```

Skrót cs pochodzi od angielskiego components, a wewnątrz nawiasów kwadratowych znajdują się rozdzielone przecinkami kolejne obiekty. Obiektem może być kontrolka (Button, TextView, Switch, EditText, itd) lub inny layout.

W tym przykładzie tablica komponentów zawiera dwie kontrolki: TextView („ty”:”TextView”) oraz Butoon („ty”:”Button”). Kontrolka TextView nosi nazwę „tv_1” i wyświetla tekst „Hello World” w kolorze białym („tc”:”255,255,255”) na czarnym tle („bg”:”0,0,0”). Rozmiar tekstu ustalono na 40 („ts”:”40”), a czcionka jest pogrubiona („tl”:”bold”). Kontrolka tekstowa będzie zajmowała dwie trzecie szerokości ekranu ponieważ jej wielkość określono na 2 („w”:”2”), a suma wielkości obu kontrolek to 3.

Przyjrzyjmy się teraz Button'owi: nosi nazwę btn_1 („n”:”btn_1”), a wyświetlany na nim napis w kolorze czarnym („tc”:”0,0,0”) to OK („te”:”OK”). Napis ma rozmiar mniejszy o 10 punktów niż ten z pola tekstowego („ts”:”30”), ale czcionka jest również pogrubiona („tl”:”bold”). Tło buttona pokolorowano na pomarańczowo definicją z linii 22 („bg”:”254,127,0”). Button zajmie jedną trzecią szerokości ekranu ponieważ jego rozmiar zdefiniowano na 1 („w”:”1”), a suma wielkości obu kontrolek to 3. Aby zmienić układ layoutu na wertykalny (elementy ułożone jeden pod drugim) wystarczy zmodyfikować kod odpowiedzialny za orientację w trzeciej linii na „or”:”V”.

Poniżej pokazano przykład layoutu wertykalnego z trzema kontrolkami, którego analizę pozostawiamy czytelnikowi. Wskazówką weryfikującą poprawność niech będzie zrzut ekranu smartfona po uruchomieniu kodu, który pokazuje **rysunek 8**.

Co dalej?

W kolejnych odcinkach zajmiemy się opisaniem kontrolek wraz z ich cechami. Sprawdzimy również w jaki sposób najszybciej zaprojektować i przetestować nowy interfejs użytkownika. Zajmiemy się też interakcją z zaprojektowanym interfejsem, bo przecież działanie aplikacji mobilnych nie kończy się tylko na wyświetleniu informacji na ekranie.

Mariusz Żądło
iram@poczta.onet.pl