IQRF – więcej niż radio

Sprzęt, IQRF OS i podstawy konfigurowania IDE

Jeśli do aplikacji jest potrzebny nieskomplikowany moduł do transmisji radiowej, ale jednocześnie dający unikalne możliwości, warto zainteresować się ofertą czeskiej firmy Microrisc. Czesi wpadli na pomysł, aby połączyć moduł radiowy z mikrokontrolerem. Taki pomysł nie wydaje się czymś odkrywczym i rzeczywiście by tak było, gdyby na tym poprzestano. Ale produkt Microrisc'a to znacznie więcej, ponieważ zaprojektowano cały system zbudowany z małych, łatwych w użyciu i programowaniu modułów radiowych, zestawów rozszerzających DCC Kits, firmowych narzędzi sprzętowych i programowych, stanowiących kompleksowe wsparcie dla konstruktora.

Projektanci protokołów przeznaczonych do przesyłania danych droga radiową starają się w nich upakować jak największą liczbę usług. Nie jest czymś nadzwyczajnym stosowanie rozwiązań sieciowych z mechanizmami powtarzania zagubionych lub przekłamanych ramek. Dane mogą być szyfrowane, aby uodpornić transmisję na ataki. Jednak stosowanie rozbudowanych protokołów często powoduje znaczące zmniejszenie rzeczywistej prędkości przesyłania danych. Dodatkowo, potrzebne są spore zasoby: szybki mikrokontroler z dużą pamięcią. Nie można też pominąć faktu, że programowe zaimplementowanie protokołu nie jest banalne i może być kosztowne. Można stosować specjalizowane układy ze stosem protokołów zaimplementowanym sprzętowo, ale może to być istotnym ograniczeniem na przykład w wypadku zaprzestania produkcji układu lub konieczności wykonania większej modyfikacji naszej aplikacji.

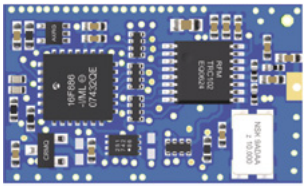
Stosowanie układów lub modułów ze stosem protokołów czasami jest całkowicie uzasadnione, ale bardzo często potrzebne są prostsze rozwiązania, gdzie nie potrzeba transmisji z wykorzystaniem standardów Bluetooth, Wi-Fi, czy ZigBee. W takich sytuacjach można wybrać „gołe" moduły produkowane przez wiele firm. Zwykle mają one interfejs SPI lub UART i poza przesyłaniem danych nie wspierają transmisji na przykład przez obliczenie i sprawdzanie sumy kontrolnej. Programista sam musi zadbać o odpowiedni protokół transmisji i retransmisję w wypadku zakłóceń. Przykładem bardziej zaawansowanego rozwiązania, którego też oczywiście można użyć, jest moduł radiowy TLX2401. Wbudowany sterownik zapewnia wyliczanie CRC i adresowanie docelowego użytkownika. Poza tym ma zaimplementowaną unikalną właściwość – ShockBurst. Mikrokontroler wpisuje dane do wewnętrznego bufora lub je odczytuje z dowolną prędkością, a wbudowany sterownik sam zajmuje się wysyłaniem i odbieraniem danych.

System IQRF – podstawowe właściwości

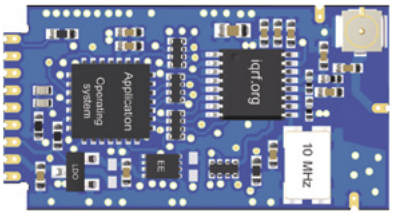
Jak wspomniano, w produktach Microrisc mikrokontroler współpracuje z nieskomplikowanym modułem radiowym. Jak można się spodziewać, bierze on na siebie cały ciężar obsługi transmisji danych. W protokole wymiany danych zaimplementowano tylko te funkcje, które są niezbędne do przesyłania danych pomiędzy modułami. Nie oznacza to jednak rezygnacji z maksymalnej, dostępnej funkcjonalności, ograniczonej zasobami małego mikrokontrolerem. Oprócz możliwości wymiany danych pomiędzy dwoma modułami w trybie *peer-to-peer* zaimplementowano własny protokół sieciowy o nazwie IQMESH.

Oprócz samych modułów, producent – firma IQRF, oferuje szereg gotowych wyrobów pracujących w sieciach IQMESH, jednak dla elektronika konstruktora najważniejsze będą moduły radiowe i narzędzia potrzebne do ich zaprogramowania oraz uruchomienia. Obecnie w ofercie producenta można znaleźć 3 typy modułów: TR52B (**rysunek 1**), TR52D i TR53B (**rysunek 2**). Zapowiadany jest kolejny – TR54D. Przyjrzymy się im bliżej.

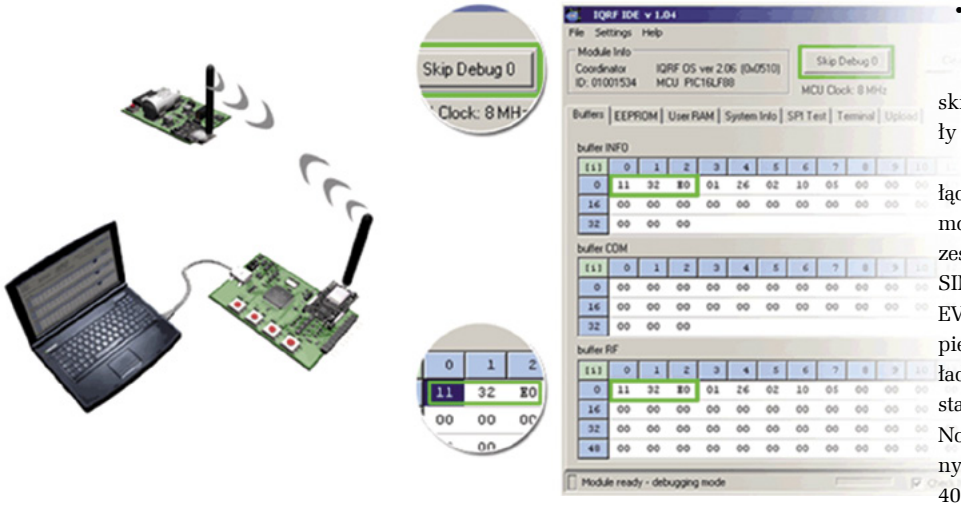
Jak już wiemy, moduł głównie składa się z transceivera radiowego i mikrokontrolera. TR52B jest dodatkowo wyposażony w czujnik temperatury i pamięć EEPROM o pojemności 16 kb. Moduł radiowy elektrycznie łączy się z urządzeniami zewnętrznymi przez typowe złącze telefonicznych kart SIM (na płytce modułu radiowego są wykonane złocone kontakty). W pamięci mikrokontrolera nadzorującego pracę każdego z modułów umieszczono system operacyjny IQRF OS. Nie zajmuje on jednak całej pamięci programu, dzięki czemu użytkownik może dodać niezbędne funkcje. Aplikacja jest uruchamiana pod kontrolą wspomnianego IQRF OS. Dzięki temu potencjalnemu programiście jest oddawanych do dyspozycji wiele funkcji systemowych, bez konieczności ich implementowania. Program dla modułu jest pisany w języku C i kompilowany kompilatorem CC5X. Producent zadbał o to, aby nie trzeba było martwić się o niezbędne narzędzia. Darmowa wersja kompilatora zupełnie wystarcza do napisania sporego programu. Trzeba pamiętać, że taki moduł jest tylko inteligentnym układem peryferyjnym i nie są na nim uruchamiane żadne duże zadania. Tworzenie aplikacji umożliwia firmowy pakiet IRQF IDE, który mimo iż na pierwszy rzut oka wygląda na nieskomplikowane narzędzie, to ma spore możliwości. Za jego pomocą możemy



Rysunek 1. Moduł TR52B

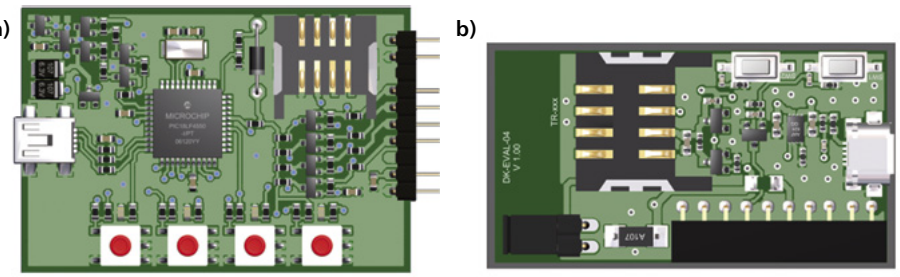


Rysunek 2. TR53B



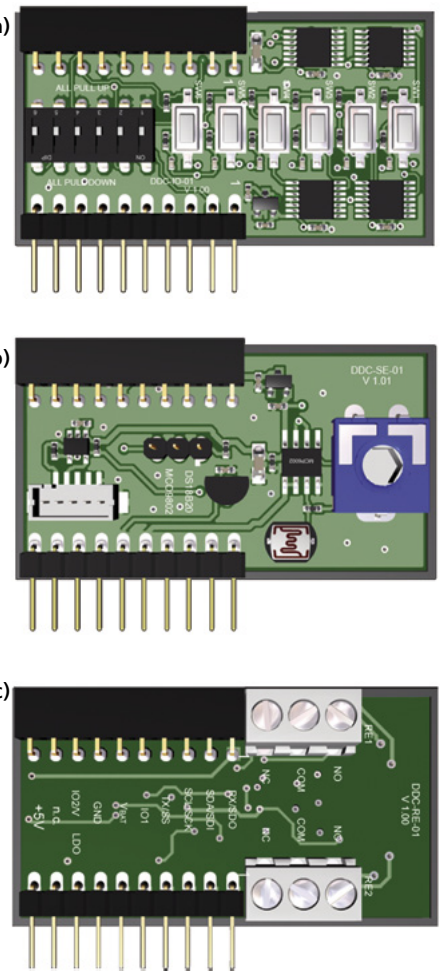
Rysunek 3. Narzędzia projektowe IQRF

a) b)



Rysunek 4. Zestawy ewaluacyjne CK-USB-02 (a) i CK-USB-04 (b)

a) b) c)



Rysunek 5. Moduły zestawu DCC Kits: DCC-IO-01 (a), DCC-SE-01 (b), DCC-RE-01 (c)

wykonać wszystkie czynności projektowe: edytować (z użyciem zewnętrznego edytora) plik źródłowy, kompilować plik źródłowy, zaprogramować mikrokontroler modułu radiowego i wyszukiwać błędy w działającym programie. IDE zawiera również interfejs programatora modułów, który umożliwia zapisanie pamięci Flash mikrokontrolera modułu kodem programu użytkownika.

Zestawy ewaluacyjne CK-USB-02 i CK-USB-04

Głównym elementem zestawów CK-USB jest mikrokontroler PIC18LF4550 z wbudowanym interfejsem USB. Po zainstalowaniu pakietu IRQF IDE i firmowych driverów USB, można z poziomu pakietu IDE programować pamięć Flash i debugować wykonywany program. Moduły TR52B i TR53B łączy się z zestawami ewaluacyjnymi poprzez złącze karty SIM.

Nowszy zestaw CK-USB-04 ma wyprowadzone na złączu zewnętrzne sygnały z modułu radiowego. Dzięki temu można do niego dołączać zewnętrzne moduły z zestawu DCC kits:

- DCC-IO-01 – moduł portów I/O z przyciskami zwiernymi i diodami LED sygnalizującymi stan przycisku (zwarty/rozwarty),
- DCC-SE-01 – moduł czujników zawierający czujnik temperatury MCP9802 (magistrala I²C), czujnik temperatury DS18B20 (magistrala 1-wire), fotorezystor i potencjometr,
- DCC-RE-01 – moduł z 2 miniaturowymi przekaźnikami FTRB3GB003Z.

Każdy moduł DCC Kits ma 2 złącza: męskie i żeńskie. Za pomocą tych złączy moduły DCC kits można łączyć szeregowo.

Żeby można było łatwo testować działanie łącza radiowego składającego się z minimum 2 modułów przygotowano dodatkowe specjalne zestawy z zasilaniem bateryjnym i złączami SIM dla modułów radiowych. Zestaw DK-EVAL-03 ma wbudowany akumulator o napięciu 3,7 V i pojemności 240 mAh. Za jego ładowanie odpowiada zamontowany na płytce stabilizator MCP7383 zasilany ze złącza USB. Nowszy zestaw DK-EVAL-04 ma wbudowany akumulator o napięciu 3,7 V i pojemności 400 mAh. Ładowanie jest wykonywane tak samo, jak w DK-EVAL-03. Ponadto, na płytce jest umieszczone złącze pozwalające na dołączenie modułów z zestawu DCC Kits.

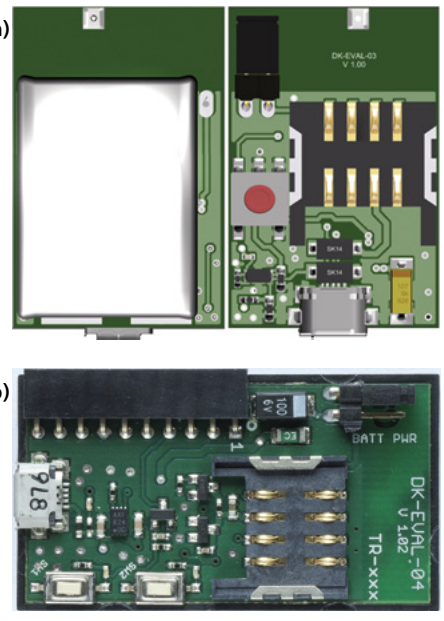
System IQRF to system łączy radiowych o zadziwiająco dużych możliwościach. Dobrze przemyślany zestaw modułów ewaluacyjnych i rozszerzeń zestawu DCC Kits pozwalają na szybkie i bezproblemowe uruchamianie własnych aplikacjami.

Moduł radiowy TR52B

Konstrukcja modułów radiowych zostanie omówiona na przykładzie TR52B. Jego schemat blokowy został pokazano na **rysunku 7**, a ideowy na **rysunku 8**. Moduł zmontowano na dwustronnej płytce drukowanej o wymiarach 25 mm×19,9 mm. Składa się on z kilku zasadniczych części:

- mikrokontrolera PIC16F886 (Microchip),
- modułu radiowego TRC102 (RFM),
- czujnika temperatury MCP9700 (Microchip),
- pamięci EEPROM typu 24AA16,
- stabilizatora napięcia +3,3 V typu MCP1700 (Microchip).

a) b)



Rysunek 6. Zestawy ewaluacyjne DK-EVAL-03 (a) i DK-EVAL-04 (b)

Wszystkie elementy modułu wykonano w technologii SMD i przylutowano na umownej górnej stronie elementów płytki drukowanej (*component side*). Na dolnej stronie płytki są umieszczone złocone kontakty przeznaczone do połączenia wyprowadzeń z typowym gniazdem dla karty SIM.

Mikrokontroler PIC16F886. Mikrokontrolery rodziny PIC16F mają dobrze sprawdzony rdzeń RISC, doskonałe układy peryferyjne, pobierają niewiele mocy. PIC16F886 może pracować z maksymalną częstotliwością oscylatora wynoszącą 20 MHz. Alternatywnie może być taktowany wbudowanym wewnętrznym oscylatorem RC o częstotliwości 8 MHz. Tę częstotliwość można programowo dzielić wewnętrznie, a stopień podziału ustalać przez zapisywanie jednego z rejestrów sterujących SFR.

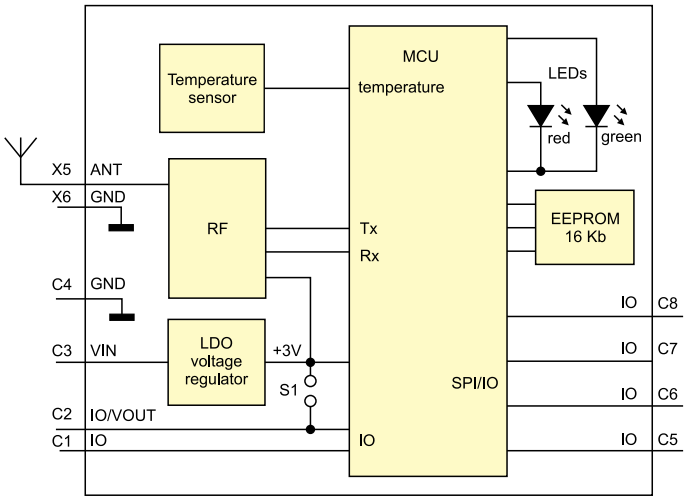
Mikrokontroler pracuje w szerokim zakresie napięć zasilających 2…5,5 V. Rozbudowany układ zerowania z blokami PWRT (Power up Timer) i POR (Power On Reset) zapewnia bezproblemowy start mikrokontrolera w czasie włączenia zasilania. Pamięć programu typu Flash ma rozmiar 8 kSłów, a pamięć danych RAM 386 bajtów. Dodatkowo, mikrokontroler ma wbudowaną pamięć EEPROM o pojemności 256 bajtów.

Lista dostępnych peryferii jest jak na mały i tani mikrokontroler dość obszerna. Oprócz 24 dwukierunkowych linii portów I/O są do dyspozycji: 3 timery (w tym jeden 16-bitowy), 10-bitowy przetwornik A/C, moduł CCP połączony z układem PWM, moduł transmisji szeregowej USART i rozbudowany interfejs szeregowy MSSP (transmisja SPI i I²C w trybach *Master* i *Slave*). Ze schematu pokazanego na rys. 8 można wywnioskować, że wykorzystywany jest przetwornik A/C (pomiar temperatury z układu MCP9700A), interfejs I²C, UART lub SPI (komunikacja z układem hostem poprzez wyprowadzenia C5…C8) oraz linie portów I/O. Na pewno są używane liczniki/timery do generowania tick'ów systemowych systemu IQRF OS i ewentualnie odliczania opóźnień.

Transceiver TRC102. Część radiową modułu zbudowano w oparciu o układ TRC102 firmy RFM. Do przesyłania danych zastosowano modulację FSK z maksymalną prędkością transmisji wynoszącą 256 kbps. Częstotliwość nośnej może się zmieniać w zakresie od 400 MHz do 1 GHz i pokrywa 3 pasma częstotliwości:

- 433 MHz (95 kanałów o szerokości 100 kHz),
- 868 MHz (190 kanałów o szerokości 100 kHz),
- 915 MHz (285 kanałów o szerokości 100 kHz).

Moduł ma dobrą czułość wynoszącą –112 dBm. Jest zasilany napięciem z zakresu 2,2….3,8 V. W trybie odbioru danych pobiera prąd o natężeniu ok. 11 mA. Pobór prądu można ograniczyć wprowadzając moduł w tryb *Standby*, w którym to pobór prądu wynosi zaledwie 0,3 μA.

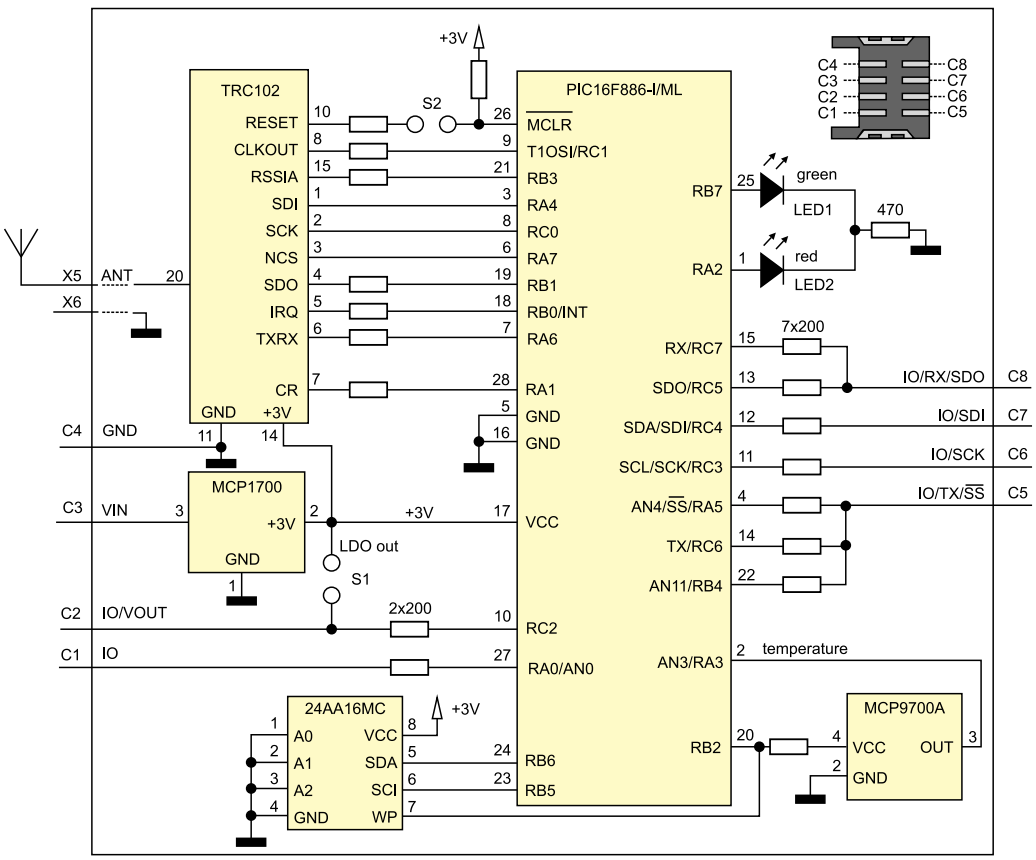


Rysunek 7. Schemat blokowy modułu TR52B

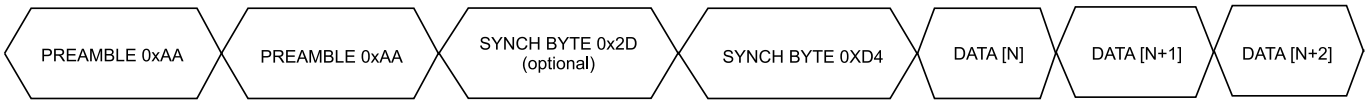
Z punktu widzenia programowej obsługi przesyłania danych, najważniejszym elementem toru radiowego jest interfejs SPI z wbudowanymi mechanizmami wspierającymi poprawny odbiór danych.

Logiczna część TRC102 jest sterowana osiemnastoma 16-bitowymi rejestrami, włączając w to rejestr danych przeznaczonych do wysłania (TxReg) i rejestr danych odbieranych z bufora FIFO (*FIFO Read*). Tematem tego artykułu nie jest jednak programowanie TRC102, więc nie będziemy się dokładnie zajmować opisywaniem rejestrów sterujących i pełnionych przez nie funkcji. Dane te można znaleźć w dokumentacji układu.

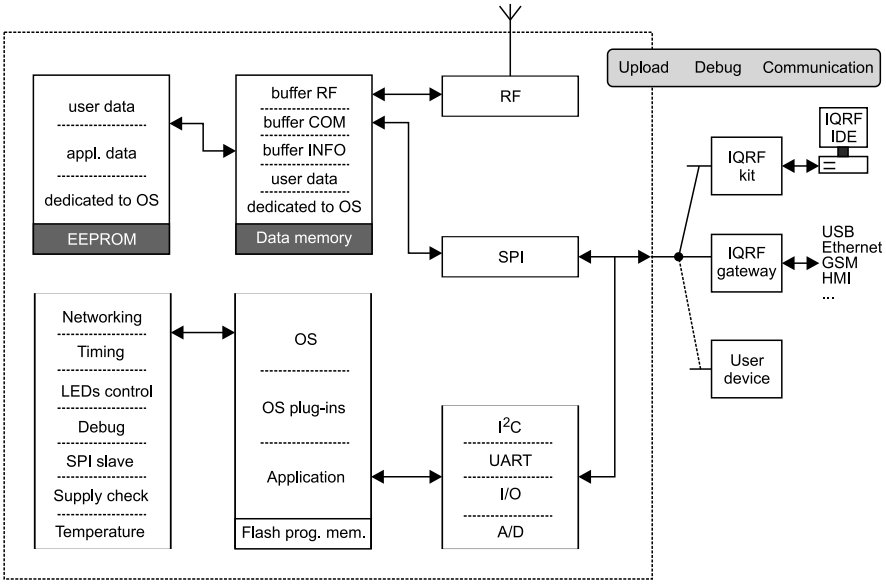
16-bitowy rejestr TxReg jest zbudowany z dwu 8-bitowych rejestrów przesuwnych połączonych szeregowo. Po włączeniu zasilania (POR) rejestry są inicjowane wartościami 0xAA. Dwa bajty o tej zawartości są preambułą nadawanych danych. Po odblokowaniu nadajnika, jeżeli mikrokontroler nie prześle do TxReg żadnych danych, to drogą radiową w sposób ciągły są transmitowane bajty 0xAA. Przesłanie danych zapisanych do TxReg jest sygnalizowane stanem niskim na wyprowadzeniu IRQ.



Rysunek 8. Schemat ideowy TR52B



Rysunek 9. Przykład transmisji danych



Rysunek 10. Architektura systemu IQRF OS

Rejestr odbiornika FIFO również jest 16-bitowy. Układ można tak skonfigurować, że po odebraniu zaprogramowanej liczby bitów (1…16) zostanie zgłoszone przerwanie na wyprowadzeniu IRQ.

Odbiornik można zaprogramować do odbioru jednego lub dwóch bajtów synchronizacyjnych. Jeden bajt ma wartości 0xD4, drugi jest opcjonalny i może być zdefiniowany przez użytkownika. Odbiór tych bajtów (jednego lub dwóch) może być sygnalizowany stanem wysokim na wyprowadzeniu 16 (w module nieużywane).

Na **rysunku 9** pokazano przykładową transmisję danych. Każdy wysyłany pakiet niezależnie od tego czy wysyłane są bajty synchronizacyjne czy nie musi być poprzedzony jednym lub dwoma (to jest programowane) bajtami preambuły.

Odczytywanie rejestru FIFO jest możliwe kiedy na wyprowadzeniu 6 (TxRx) jest zostanie wymuszony stan niski. Jeżeli odbiornik ma dane gotowe do odczytania, to wyprowadzenie 7 (CR) przechodzi w stan wysoki i jest w tym stanie do momentu odczytania danej przez interfejs SPI.

Czujnik temperatury MCP9700A. MCP9700 jest scalonym czujnikiem zasilanym napięciem +3,3 V i mierzącym temperaturę w zakresie –40…+125°C. Dokładność pomiaru w zakresie 0…+70°C wynosi ±4°C przy typowym poborze prądu na poziomie 6 μA. Elementem mierzącym jest złącze diody PN. Sygnałem wyjściowym jest napięcie zmieniające się zgodnie ze współczynnikiem 10,0 mV/°C:

$$Vout = Tc1 \times Ta + V0C$$

gdzie:

- ***Ta*** – temperatura otoczenia,
- ***Tc1*** współczynnik 10,0 mV/C,
- ***V0C*** – napięcie na wyjściu dla 0C (500 mV).

Niezbyt wysoką dokładność można stosunkowo łatwo skorygować poprzez kalibrację programową. Jeżeli skalibrujemy czujnik dla temperatury +25°C, to możemy uzyskać dużo większą dokładność w całym zakresie pomiarowym. W dokumentacji opisano metody uzyskiwania większych dokładności i określania wpływu samonagrzewania się układu na pomiar temperatury.

Pamięć EEPROM 24AA16. Pamięć EEPROM ma pojemność 16 kb i jest zasilana napięciem +2,5…5,5 V. Jest wyposażona w szeregowy interfejs komunikacyjny I²C. Sygnał zegarowy magistrali może mieć częstotliwość 100 lub 400 kHz.

Architektura IQRF OS

Żeby móc wykorzystywać zapisany w pamięci mikrokontrolera system operacyjny IQRF OS, trzeba poznać przynajmniej w ogólnym zarysie sposób jego działania. Głównym zadaniem OS jest obsługa transmisji danych przez kanał radiowy i transmisji SPI pomiędzy modułem a zewnętrznym hostem. Obsługa tych zadań jest wykonywana w tle programu głównego na poziomie „niższym". Programista ma do dyspozycji funkcje systemowe pozwalające na szybkie i łatwe organizowanie wymiany danych Nie trzeba wczytywać się w dokumentację modułu radiowego i mikrokontrolera, opracowywać i testować optymalnych rozwiązań transmisji radiowej – to wszystko jest już gotowe. Można skupić się na tworzeniu własnych algorytmów dostosowujących działanie modułu do konkretnych potrzeb. Oprócz tego, funkcje systemowe pomagają w organizacji i obsłudze dość rozbudowanej sieci IQMESH.

Struktura oprogramowania modułu składa się z dwu głównych warstw:

- Zbioru procedur opracowanych i programowanych przez producenta. Te procedury składają się na system operacyjny IQRF OS.
- Warstwy aplikacji. Ta warstwa zawiera procedury napisane przez użytkownika.

Tu ważna uwaga: wszystkie procedury IQRF OS są skompilowane i wpisane do pamięci Flash mikrokontrolera na etapie produkcji u producenta. W trakcie pracy nad własnym projektem kompilacji podlegają tylko procedury warstwy aplikacji. Po skompilowaniu do pamięci Flash jest wgrywany tylko kod wynikowy aplikacji. Dlatego jest możliwe efektywne wykorzystanie darmowej wersji kompilatora z ograniczeniem długości kodu. Producent nie musi też udostępniać wersji źródłowej (ani wynikowej) procedur OS, co zapewne ograniczy piractwo. Wadą takiego rozwiązania jest to, że zmiana wersji systemu na nowszą jest możliwa tylko u producenta. Możliwe jest za to rozszerzenie funkcjonalności IQRF OS poprzez instalacje dodatkowych „wtyczek" (plug-in). Tę operację może wykonać użytkownik.

Działanie pod kontrolą systemu operacyjnego polega na używaniu funkcji systemu, ale także na wykorzystywaniu mechanizmu przydzielania procesorowi zasobów do wykonywanych zadań. Jak przystało współczesnemu systemowi operacyjnemu, pod kontrolą IQRF OS można uruchomić więcej niż jedno zadanie jednocześnie. Zastosowany mikrokontroler nie pozwala na typową wielozadaniowość, ale dla tego specjalizowanego zadania wystarczy. W aplikacjach czasu rzeczywistego musi działać zegar odmierzający interwały czasowe, tzw. tiki (*tick*). W IQRF OS pojedynczy tik ma długość 10 ms. Interwały zlicza 16-bitowy licznik przepełniający się co 2^{16} impulsów, czyli co 655 sekund. Wykorzystując metodę przechwytywania zawartości tego licznika można łatwo odmierzać opóźnienia.

Jak już wspomniałem, podstawowym zadaniem wykonywanym przez OS w tle programu głównego jest obsługa dwu protokołu SPI:

- do komunikacji łączem radiowym RF z wykorzystaniem układu TRC102. Komunikacja pomiędzy modułami może się odbywać w topologii *peer-to-peer* lub z użyciem własnego protokołu sieciowego IQMESH.
- do komunikacji z układami peryferyjnymi w tym z hostem mikroprocesorowym poprzez standardowy interfejs SPI pracujący w trybie Slave.

Jak dla takiego prostego systemu z małym mikrokontrolerem to całkiem sporo. Oprócz tego funkcje systemowe obsługują odmierzanie czasu, sterowanie diodami LED, pomiar temperatury przez wbudowany czujnik i zapewniają kontrolowanie napięcia zasilania. Bardzo użyteczna jest funkcja systemowa pozwalająca na debugowanie pisanego kodu. Architekturę IQRF OS pokazano na **rysunku 10**.

W mikrokontrolerach z rodziny PIC16F w obszarze pamięci danych RAM są umieszczone rejestry sterujące SFR i rejestry ogólnego przeznaczenia (pamięć danych). Dostęp do przestrzeni pamięci zajętej lub używanej przez system może być ograniczony, lub niemożliwy. Na przykład aplikacja użytkownika nie może używać rejestrów INDF (adresowanie pośrednie RAM), EECON1, EECON2, EEADR (dostęp do EEPROM), PIR, PIE1 i PIE2 (konfigurowanie systemu przerwań). Jeżeli to konieczne, to dostęp do wolnej przestrzeni zapewniają funkcje systemowe. Zaleca się dokładne przestudiowanie opisu dostępnych zasobów mikrokontrolera.

W pamięci RAM dedykowanej dla OS są umieszczone 3 bufory systemowe:

– *bufferRF* o długości 64 bajtów (adresy 0x110…0x14F) przeznaczony do obsługi transmisji danych przez moduł radiowy RF,
– *bufferCOM* o długości 41 bajtów (adresy 0xA0…0xC8) przeznaczony do obsługi transmisji danych przez interfejs SPI do komunikacji z hostem. Obecnie można wykorzystywać 35 bajtów od adresu 0xA0. Pozostały obszar jest zarezerwowany.
– *bufferINFO* o długości 35 bajtów (adresy 0x20…0x42) przeznaczony do odczytywania informacji systemowych z pamięci EEPROM.

OS ma funkcje systemowe specjalnie przygotowane do kopiowania danych pomiędzy tymi buforami. Poza tym jest zdefiniowany bufor ogólnego przeznaczenia *bufferAUX* o pojemności 22 bajtów. W pamięci danych zarezerwowano obszary dla danych użytkownika i zmiennych systemowych. Bufory zajmują w pamięci danych 172 bajty, a do dyspozycji aplikacji użytkownika jest przeznaczony obszar o rozmiarze 40 bajtów.

Pamięć EEPROM pokazana na rys. 7 jako jeden blok w rzeczywistości składa się z dwóch obszarów: wewnętrznej pamięci EEPROM mikrokontrolera o pojemności 256 bajtów i zewnętrznej pamięci EEPROM 24AA16 o pojemności 2 kB. Dane systemowe są przechowywane w wewnętrznej pamięci EEPROM mikrokontrolera. Jedną z takich danych jest identyfikator modułu odczytywany komendą systemową *moduleInfo()* do bufora *bufferINFO*. Format identyfikatora pokazano na **rysunku 11**.

W pamięci Flash są zapisane procedury OS, wtyczki OS i aplikacja użytkownika. Procedury systemu zajmują 1024 słowa instrukcji. W dokumentacji *IQRF OS User's Guide* jest

Adres w buffINFO	7	6	5	4	3	2	1	0
	wersja softu OS		Typ PIC	Wersja OS	Koordynator	Numer seryjny		
				Identyfikator modułu				

Rysunek 11. Struktura identyfikatora modułu OS

umieszczona mapa pamięci RAM i EEPROM z zaznaczeniem obszarów zajętych przez system i dostępnych dla użytkownika.

Dane odebrane z modułu radiowego są zapisywane w buforze *bufferRF* przez pracujący w tle wątek systemowy. Stamtąd, po przepisaniu do *bufferCOM*, mogą być odczytane przez zewnętrzny hosta (*User Device*) lub przesłane do firmowej bramki (*IRQF gateway*). Bramka (produkowana i sprzedawana oddzielnie przez Microrisc) pozwala na przesyłanie danych dalej poprzez inne interfejsy np. USB, Ethernet itp. Moduł może komunikować się z hostem z wykorzystaniem pozostałych interfejsów, alternatywnych dla SPI: I²C lub UART. Niestety, ich obsługę użytkownik musi zaimplementować sobie sam. Wyprowadzenia C5...C8 mogą też być wykorzystywane jako uniwersalne linie I/O lub wejścia przetwornika A/C.

Po przesłaniu danych łączem radiowym połączenie jest automatycznie zamykane i układ przechodzi do trybu odbioru danych. Jeżeli są spodziewane nowe dane, to można użyć funkcji systemowych *checkRF()* lub *RFRXpacket()*. Dane są wysyłane po użyciu funkcji *RFTXpacket()*.

W systemach sterowania i nadzoru zasilanych bateryjnie jest ważne ograniczenie pobieranej mocy. Główne elementy modułu, czyli mikrokontroler PIC18F886, transceiver TRC102 i czujnik temperatury mają wbudowane mechanizmy usypiania. Wprowadzanie w tryb uśpienia całości realizuje funkcja systemowa *iqrfSleep()*. Jej wywołanie musi być jawnie umieszczone w programie użytkownika. Uśpienie zatrzymuje liczniki mikrokontrolera, odłącza podciąganie linii portów do plusa zasilania i wyłącza moduł odbiorczo – nadawczy TRC102.

Uzycie funkcji *irqfSleep()* musi być poprzedzone skonfigurowaniem możliwości wybudzania mikrokontrolera poprzez zmianę stanu na wyprowadzeniu. Sekwencja usypiania/wybudzania może wyglądać tak:

```
GIE=0; //zablokuj przerwania
RBIE=1;//odblokuj zgłaszanie
przerwań od zmian stanów na
liniach PORTB
iqrfSllep(); uśpij moduł
RBIF=0; //jeżeli konieczne
```

Powrót do normalnego działania (stanu aktywnego) następuje po:

- przepełnieniu watchdog'a (jeżeli jest odblokowany).
- zmianie poziomu na linii portu PORTB – typowo RB4 na wyprowadzeniu C5.

Moduł radiowy może być usypany oddzielnie po wywołaniu funkcji *setRFsleep()*. Może to być potrzebne gdy mikrokontroler pracuje, ale nie jest potrzebna transmisja danych. Pobór prądu obniża się wówczas do 0,6 mA. Ponowne włączenie toru radiowego następuje po wywołaniu funkcji *RFRXpacket()*, *RFTXpacket()*, *checkRF()* lub *getSupplyVoltage()*.

PIC16F886 ma wbudowany licznik watchdog'a. Po jego przepełnieniu następuje automatyczny restart mikrokontrolera. Okres przepełnienia przy częstotliwości zegara taktującego wynoszącej 8 MHz można ustawić od 1 do 268 ms. Można go znacznie wydłużyć włączając preskaler. Domyślnie czas przepełnienia jest ustawiony na około 4 sekundy. Watchdog można programowo włączać lub wyłączać. Do jego zerowania służy instrukcja *clrwdt()*. OS nie ma wbudowanej funkcji obsługi watchdog'a i jeżeli zdecydujemy się na jego zastosowania, to jego obsługę trzeba zaimplementować w programie użytkownika.

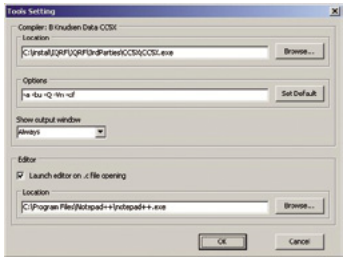
Łącze radiowe pracuje w dwóch pasmach: 868 MHz lub 916 MHz. Pasma i kanały są wybierane programowo. Programowo można też wybierać prędkość transmisji z jednej z wartości: 1,2 kb/s, 19,2 kb/s, 57,6 kb/s i 86,2 kb/s. Producent zaleca wybór 19,2 kb/s, a pozostałe prędkości przeznaczyć do testowania. Maksymalna moc wyjściowa to 3,5 mW. Można ją regulować programowo w 8 krokach za pomocą funkcji systemowej.

Narzędzia firmowe

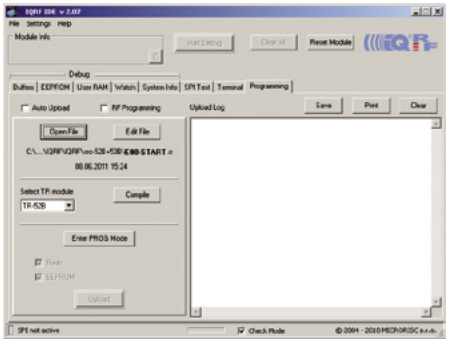
Pakiet IQRF IDE. Wiemy jak jest zbudowany moduł radiowy i znamy przynajmniej w zarysie jaka jest filozofia systemu IQRF OS. Ta wiedza jest niezbędna do napisania efektywnie działającego programu dla modułu. Jednak aby napisać nawet najprostszy program, potrzebne będą narzędzia: środowisko IDE z kompilatorem i programator. Do testowania otrzymałem 2 różne zestawy ewaluacyjne: jeden z modułem CK-USB-02, a drugi CK-USB-04. Każdy z nich zawierał komplet modułów radiowych TR52B, moduł ewaluacyjny pełniący funkcję programatora i płytę z narzędziami programowymi.

Opis narzędzi firmowych rozpocznę od programu IDE. Program z kompilatorem i dokumentacją jest dystrybuowany na płycie CD. W pierwszym kroku trzeba rozpakować plik *IQRF-Startup-Package*. Po rozpakowaniu są tworzone 4 katalogi:

– *3rdParties* z kompilatorem CC5X, edytorem graficznym Notepad++ i driverem USB do firmowego programatora,
– *Documentation* z notami aplikacyjnymi i podręcznikami użytkownika,
– *IQRF_IDE* z programem środowiska IDE i plikami pomocy,



Rysunek 12. Konfigurowanie ścieżek dostępu



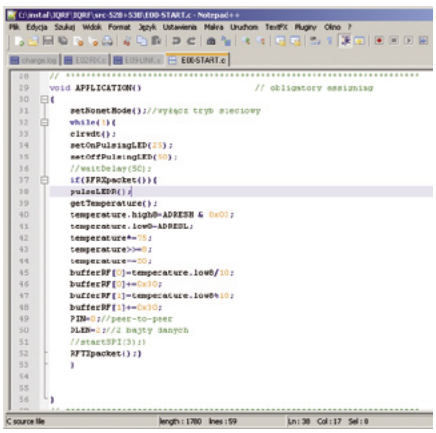
Rysunek 13. Okno IQRF IDE

– *Src-52B+53B* z plikami źródłowymi przykładów programów użytkowych.

Dokładny opis instalacji narzędzi programowych jest zamieszczony w nocie aplikacyjnej *AN003 IQRF Development Tools Installation Guide*.

Połączenie programatora z komputerem, na którym jest zainstalowane IDE jest realizowane przez złącze USB. Do jego obsługi trzeba zainstalować sterownik USB autorstwa IQRF. Sposób jego zainstalowania jest dokładnie opisany w nocie AN003.

Aplikacja IRQF IDE w wersji V2.07 zapisanej na płycie CD dostarczonej wraz z zestawami nie wymaga instalowania i można ją uruchamiać z katalogu IQRF_IDE. Ze strony producenta można też pobrać nowszą wersję (w momencie tworzenia artykuły była to V2.08), która – w odróżnieniu od poprzedniej – wymaga zainstalowania. Po pierwszym uruchomieniu trzeba w zakładce *Settings –> Tool Settings* podać ścieżkę dostępu do katalogu kompilatora i ścieżkę dostępu do katalogu edytora plików źródłowych (**rysunek 12**).



Rysunek 14. Otwarte okno edytora notepad ++

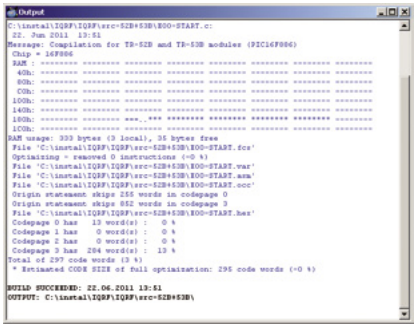
Kompilowanie pliku projektu i programowanie modułu radiowego. Na **rysunku 13** pokazano okno programu. Pracę nad własnym projektem rozpoczyna się od zakładki *Programming*. Najpierw otwieramy plik, w którym jest zawarta wersja źródłowa programu użytkownika. Jeżeli program jest skonfigurowany jak na rys. 12, to kliknięcie na przycisk *Open File* i wybranie pliku źródłowego spowoduje automatyczne jego otwarcie w edytorze. Do prób użyłem dostarczonego na płycie CD edytora notepad++. Ten edytor jest dostępny bezpłatnie w Internecie i dobrze znany programistom. Na **rysunku 14** pokazano okno z otwartym plikiem źródłowym. Wybranie tego edytora znacznie ułatwia prace z plikiem pisanym w języku C. Słowa kluczowe, komentarze i stałe są wyróżniane kolorami. Dodatkowo notepad ++ wspiera parowanie nawiasów.

Zmodyfikowany plik źródłowy jest kompilowany po kliknięciu na przycisk *Compile*. Ważne, aby przed uruchomieniem kompilatora prawidłowo wybrać moduł docelowy– w naszym wypadku będzie to TR-52B. Wynik kompilacji jest wpisywany do okna *Output* (**rysunek 15**). Jeżeli rezultat pracy kompilatora jest bezbłędny, to plik wynikowy można przesłać do mikrokontrolera modułu. *IRQF IDE* jest tak skonfigurowane, że użytkownik nie musi się martwić o to, gdzie należy zapisać program w pamięci Flash mikrokontrolera – o wszystko „dba" oprogramowanie.

Wejście w tryb programowania następuje po kliknięciu na przycisk *Enter PROG Mode*. W tym momencie moduł ewaluacyjny pełniący rolę programatora musi być połączony kablem USB z komputerem. Jeżeli w dolnej linijce okna IQRF IDE jest zaznaczona opcja *Check Mode* i to pojawi się komunikat „*Module ready – programming mode*". Obok tego komunikatu powinna migać zielona kontrolka. Jest to sygnał gotowości do programowania pamięci Flash. Po wejściu do trybu programowania jest automatycznie odczytywany i wyświetlany w oknie *Module Info* identyfikator wersji IQRF OS.

Plik wynikowy zapisuje się do modułu radiowego po kliknięciu na przycisk *Upload*. Wynik operacji przesłania do modułu jest wyświetlany w oknie *Upload Log*.

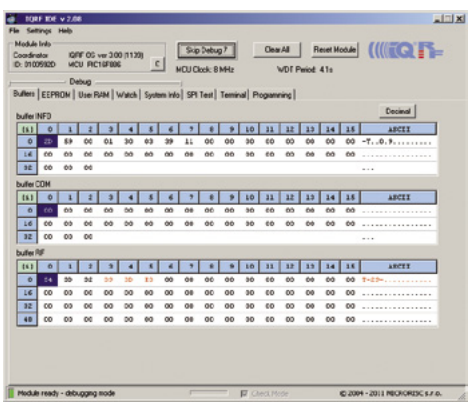
Debugowanie programu. Użytkownicy mikrokontrolerów Microchipa są przyzwyczajeni do dobrych narzędzi typu ICD (In circuit debuger) na przykład ICD2, PICKIT2, czy PICKIT3. Z praktyki wynika, że debugowanie sprzętowe znacznie przyśpiesza proces uruchamiania programu. W module radiowym użycie narzędzia tego typu nie jest możliwe, ale projektanci systemu wprowadzili dość użyteczną namiastkę debugera w postaci funkcji systemowej o nazwie *debug()*. Jeżeli chcemy mieć możliwość podejrzenia zawartości buforów transmisyjnych, pamięci RAM użytkownika, czy zmiennych systemowych (*watch*), to programie trzeba umieścić wywołanie *debug()*. Tak zmodyfikowany program musi być wgrany do modułu radiowego. Kiedy uruchomiony program w module radiowym połączonym z modułem ewaluacyjnym napotka wywołanie *debug()*, to zatrzymuje swoje działanie i w IQRF IDE uaktywnia się przycisk *Start Debug*. Oczywiście, warunkiem jest aktywne połączenie z komputerem za pomocą USB. Po kliknięciu na *Start Debug* moduł ewaluacyjny przesyła do IDE całą zawartość wszystkich zmiennych możliwych do podejrzenia.



Rysunek 15. Okno z komunikatami kompilatora

Na **rysunku 16** pokazano okno z podglądem zawartości buforów systemowych w czasie debugowania programu przesyłającego wartość zmierzonej temperatury i przesyłanej przez zdalny moduł TR52B z zasilaniem bateryjnym do modułu TR52B dołączonego do modułu ewaluacyjnego. Pierwsze 2 bajty bufora *bufferRF* zawierają kody ASCII temperatury równej 27 stopni Celsjusza. Klikając na klawisz *Clear All* można wyczyścić zawartość wszystkich przeczytanych zmiennych, zawartości buforów i pamięci RAM i EEPROM. Program zatrzymany przez wywołanie funkcji *debug()* jest ponownie uruchamiany po kliknięciu na przycisk *Skip Debug*.

Dość istotnym elementem pozwalającym na szybkie uruchomienie własnej aplikacji jest możliwość testowania interfejsu SPI przeznaczonego do komunikacji pomiędzy modułem radiowym i hostem zewnętrznym. Do tego celu jest przeznaczona zakładka SPI test (**rysunek 17**). W oknie TX data są wpisywane dane do przesłania w formacie heksadecymalnym rozdzielone kropkami. Na końcu ciągu liczb można wyliczyć i dodać sumę kontrolną kli-



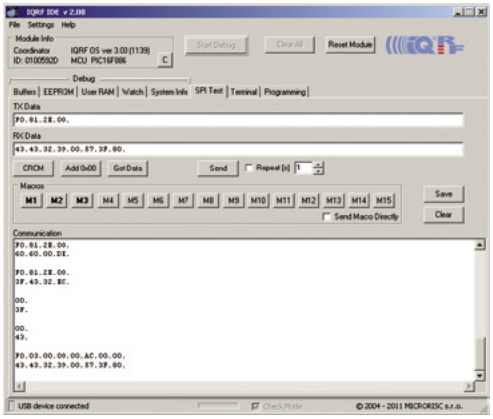
Rysunek 16. Okno z podglądem zawartości buforów systemowych

Tabela 1. Zestawienie wszystkich funkcji systemowych	
Sterowanie	
reset()	Restart system IQRF OS i aplikacji
calibrateTimer()	Kalibrowanie timera generującego tik
iqrfSleep()	Uśpienie całego modułu
setRFsleep()	Uśpienie transceiver'a radiowego
setRFreaady()	Wybudzenie transceiver'a radiowego
debug ()	Wejście w tryb debugowania
uns8 getSupplyVoltage()	Odczytanie napięcia baterii
getTemperature()	Pomiar temperatury
Odliczanie aktywnych opóźnień	
waitMS(uns8 ms)	Aktywne odliczanie opóźnienia w ms
waitDelay(uns8 ticks)	Aktywne odliczanie opóźnień w tikach
waitNewTick()	Aktywne czekanie na rozpoczęcie nowego tika
Odliczanie opóźnień w tle	
startCapture()	Zerowanie i start odliczania licznika OS ticks
captureTicks()	Przechwytuje wartość licznika OS ticks
startDelay(uns8 ticks)	Zlicza opóźnienie przez zadaną liczbę tików
startLongDelay(uns16 ticks)	Zlicza długie opóźnienie przez zadaną liczbę tików
bit isDelay()	Test czy trwa odliczanie
Sygnalizacja za pomocą LED	
setOnPulsingLED(uns8 ticks)	Zaświecenie diody LED na czas określony przez argument
setOffPulsingLED(uns8 ticks)	Zgaszenie diody LED na czas określony przez argument
pulsingLEDR()	Miganie diody czerwonej przez czasy określone przez setOnPulsingLED i setOffPulsingLED
pulseLEDR()	Miganie czerwonej diody LED w tle. Czasy ustawione tak jak w pulsingLEDR()
stopLEDR()	Zatrzymanie migania w tle diody czerwonej
pulsingLEDG()	Miganie diody zielonej przez czasy określone przez setOnPulsingLED i setOffPulsingLED
pulseLEDG()	Miganie zielonej diody LED w tle. Czasy ustawione tak jak w pulsingLEDG()
stopLEDG()	Zatrzymanie migotania w tle diody zielonej
EEPROM	
eeReadByte(uns8 addr)	Odczytanie komórki EEPROM. Adresy z zakresu od 0x00 do 0xBF
eeReadData(uns8 addr, uns8 lenght)	Odczytanie bloku danych z EEPROM
eeWriteByte(uns8 addr, uns8 data)	Zapisanie komórki EEPROM
eeWriteData(uns8 addr, uns8 lenght)	Zapisanie bloku danych do EEPROM
RAM	
uns8 ReadfromRAM(uns8 addr)	Odczytanie komórki banku pamięci RAM. Przed odczytaniem trzeba ustalić aktywny bank
writeToRAM(uns8 addr, uns8 val)	Zapisanie komórki banku pamięci RAM. Przed zapisaniem trzeba ustalić aktywny bank
Bufory, bloki danych	
copyBufferINFO2COM()	Kopiuje zawartość bufferINFO do bufferCOM
copyBufferINFO2RF()	Kopiuje zawartość bufferINFO do bufferRF
copyBufferRF2COM()	Kopiuje zawartość bufferRF do bufferCOM
copyBufferRF2INFO()	Kopiuje zawartość bufferRF do bufferINFO
copyBufferCOM2RF()	Kopiuje zawartość bufferCOM do bufferINFO
copyBufferCOM2INFO()	Kopiuje zawartość bufferCOM do bufferINFO
compareBufferINFO2RF(uns8 lenght)	Porównuje zawartość danych o ilości zadanej w argumencie buforów bufferINFO i bufferRF
copyMemoryBlock(uns8 from, uns8 to, uns8 lenght)	Kopiuje zawartość jednego obszaru pamięci RAM do drugiego obszaru
swapBufferINFO()	Zapisuje zawartość bufferINFO do bufferAUX
clearBufferINFO()	Zeruje bufferINFO
clearBufferRF()	Zeruje bufor bufferRF
moduleInfo()	Odczytuje identyfikację modułu i zapisuje w bufferINFO
appInfo();	Odczytuje dane aplikacji i zapisuje w bufferINFO
Interfejs SPI	
enableSPI()	Aktywuje interfejs sprzętowy SPI
disableSPI()	Wyłącza sprzętowy interfejs SPI i konfiguruje jego linie jako linie portów I/O
startSPI(uns8 lenght)	Inicjuje pakiet SPI przesyłany za Slave(modułu) do Master (hosta). Dane są pobierane z bufora bufferCOM
stopSPI()	Kończy komunikacje po SPI
restartSPI()	Pozwala na kontynuowanie transmisji po SPI
bit getStatusSPI()	Testuje zajętość interfejsu SPI pracującego w tle
Moduł radiowy RF	
setTXpower(uns8 level)	Ustawia 7 poziomów mocy wyjściowej
setRFspeed(uns8 speed)	Ustawia prędkość transmisji w bps
setRFband(uns8 band)	Wybiera pasmo trascivera 886MHz lub 916MHz

Tabela 1. Zestawienie wszystkich funkcji systemowych	
setRFchannel(uns8 channel)	Wybiera kanał radiowy
setRFmode(uns8 mode)	Wybiera tryb poboru mocy
checkRF()	Detekcja poziomu sygnału
TFTXpacket()	Wyślij pakiet z buforRF przez radio
bit RFRXpacket()	Odbierz pakiet i zapisz go do buforRF
Sieć	
setCoordinatorMode()	Urządzenie pracuje jako koordynator
setNodeMode()	Urządzenie pracuje jako węzeł
setNonetMode()	Sieć wyłączona
setNetworkFilteringOn()	Pakiet akceptowany tylko przez bieżącą sieć
setNetworkFilteringOff()	Pakiet akceptowany przez obie sieci
setUserAddr(uns16 address)	Nadanie węzłowi identyfikatora użytkownika
Uns8 getNetworkParams()	Pobranie informacji o stanie sieci
Routing	
setRoutingOn()	Wychodzące pakiety są rutowane w tle przez inne moduły
setRoutingOff()	Wychodzące pakiety nie są rutowane w tle przez inne moduły
uns8 dscovery (uns8 zones)	Wykrywanie węzłów do rutingu
answerSystemPacket()	Odblokowanie odpowiedzi koordynatora na wykrywanie
bit isDiscoveredNode(uns8 N)	Testowanie wykrywania węzłów
bit wasRouted()	Sygnalizuje czy przychodzący pakiet był rutowany
optimizeHops(uns8 x)	Optymalizacja numeru przeskoku dla danego węzła
Bonding – węzeł	
bit bondrequest()	Zezwolenie na przyłączenie
bit amIBonded()	Czy węzeł jest przyłączony?
removeBond()	Usuniecie przyłączenia
Bonding – kordynator	
bit bondnweNode(address)	Przyłącz nowy węzeł
bit isBondedNode(N)	Czy węzeł jest przyłączony?
removeBondeNode(N)	Odłącz węzeł
bit rebondNode(N)	Przyłącz ponownie węzeł
clearAllBonds()	Usuń wszystkie połączenia

kając na przycisk CRCM. Suma jest niezbędna kiedy do komunikacji wykorzystujemy funkcje systemowe IQRF OS.

Przygotowane sekwencje danych przeznaczonych do wysłania można zapamiętywać w makrach oznaczonych przez *M1*... *M15*. Po kliknięciu na przycisk z numerem makra przypisana sekwencja jest automatycznie wpisywana do okna TX data. Przebieg wymiany danych można obserwować w oknie *Communication*. Wymiana informacji przez SPI zostanie dokładniej opisana przy okazji omawiania przykładów programowania modułu. Do testowania transmisji



Rysunek 17. Okno testowania transmisji po SPI

po SPI jest przeznaczona też zakładka *Terminal*. Służy do wysyłania pakietów z tekstowym polem danych. W oknie *Tx Text* jest wpisywany ciąg znaków alfanumerycznych, a program sam automatycznie konstruuje kompletny pakiet danych (dodaje bajty startowe i sumę kontrolną).

Programowanie w systemie IRQF OS

Programowanie w każdym systemie operacyjnym wymaga miedzy innymi poznania funkcji systemowych, przynajmniej w zakresie potrzebnym do wykonania zamierzonego zadania. IQRF OS ma całkiem sporo wbudowanych, użytecznych funkcji. Ich skrócone zestawienie zamieszczono w **tabeli 1**. Dokładny ich opis można znaleźć w dokumencie *IQRF OS Operating system Reference Guide* dostępnym na stronie internetowej producenta. W kolejnym numerze EP opiszę szerzej te funkcje, które będą wykorzystywane w programach przykładowych.

Tomasz Jabłoński, EP