IQRF – więcej niż radio (3)

Praca modułów radiowych TR52B w sieci

Wszystkie opisywane do tej pory przykłady stosowania modułów radiowych dotyczyły konfiguracji, w której pracują dwa moduły TR52B: jeden dołączony do hosta, a drugi zdalny. Ta prosta konfiguracja jest użyteczną w wielu zastosowaniach. Można ją na przykład wykorzystać do dołączenia zewnętrznego czujnika temperatury w stacji pogodowej. Jednak jak poradzić sobie, gdy do systemu pomiarowego trzeba dołączyć większą liczbę czujek? Rozwiązaniem może być budowa sieci radiowej.

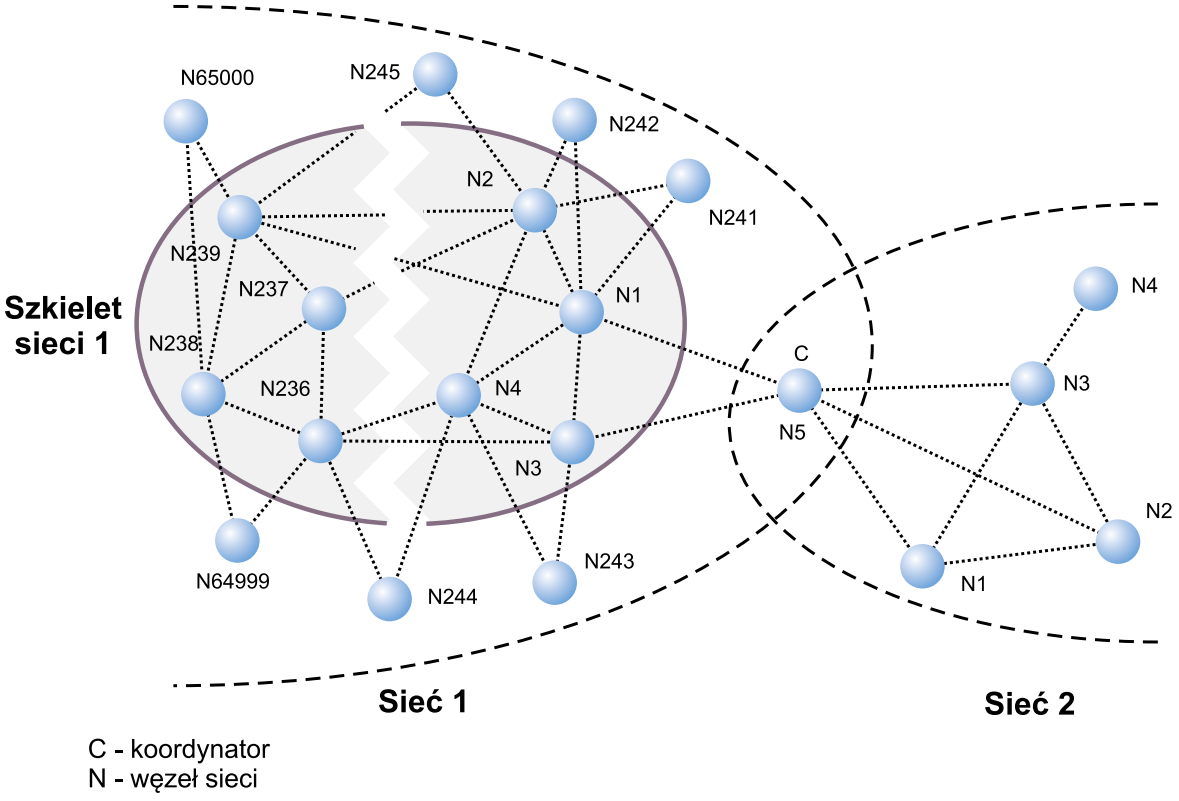
Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 18453, pass: 5eyp1854
• poprzednie części kursu

Połączenie radiowe daje możliwość dowolnego rozmieszczenia elementów sieci, pod warunkiem utrzymania prawidłowego poziomu sygnału i zwalnia od wykonywania kosztownego, i często niewygodnego okablowania. Problemy z okablowaniem będą narastały, kiedy zechcemy zwiększać liczbę miejsc, które chcemy nadzorować, lub z których będziemy odczytywać wyniki pomiarów. Przykładem może być system alarmowy z wieloma czujkami ruchu. Jeżeli w trakcie budowy domu, lub innego obiektu przewidzimy taką możliwość i instalacja zostanie wykonana przed położeniem tynków i pomalowaniem ścian, to nie ma większego problemu. Ale wykonanie takiej instalacji w gotowym obiekcie wiąże się z dużymi kosztami. Dlatego bardzo atrakcyjna jest możliwość łączenia wielu modułów w sieć. Nie dość, że nie trzeba wykonywać kosztownej instalacji, to dodatkowo można bez problemu zmieniać konfigurację – dodawać nowe obiekty lub zmieniać miejsce zainstalowania już istniejących.

W systemie IQRF moduły radiowe TR52B tworzące sieć mogą pracować w dwóch trybach:

- Peer-to-peer,
- IQMESH.

Peer-to-peer jest trybem domyślnym. Jest on używany do połączenia dwóch lub więcej obiektów bez systemowego Koordynatora sieci. Pakiety danych wysyłane przez moduł są dostępne dla wszystkich pozostałych modułów w sieci. Adresowanie i ruch pakietów nie jest wspierany przez IRQF OS i jego obsługa musi być zaimplementowana w warstwie aplikacji użytkownika. Liczba modułów w sieci nie jest limitowana. Można sobie wyobrazić sieć w topologii gwiazdy z modułami zaprogramowanymi do pracy w trybie *peer-to-peer*. Moduł połączony z hostem pracuje jako master i sekwencyjnie odpytuje wszystkie pozostałe (slave). Każdy moduł slave musi mieć przypisany na stałe unikalny adres i odpowiada po zaadresowaniu przez moduł master. Pozostałe moduły są nieaktywne do momentu zaadresowania. Takie rozwiązanie jest proste w implementacji, ale ma też wady. Podstawową jest sztywna konfiguracja. Każda zmiana topologii sieci będzie wymagała zmian w oprogramowaniu modułu połączonego z hostem. W wypadku dużej liczby modułów w sieci, czas odpowiedzi na zdarzenie może być długi. Dodanie usługi przekazywania pakietu z układu slave do następnego slave w celu zwiększenia zasięgu transmisji komplikuje protokół wymiany danych. Jednak dla sieci zawierających kilka modułów takie rozwiązanie może w zupełności wystarczyć.



Rysunek 1. Przykładowa struktura sieci IQMESH

Tryb IQMESH pozwala na utworzenie sieci kratowej (MESH). Z założenia taka sieć daje możliwość komunikacji pomiędzy elementami sieci bez konieczności używania wydzielonego elementu centralnego . Każde urządzenie sieciowe (moduł radiowy) może komunikować się z każdym innym urządzeniem bezpośrednio (jeśli jest zasięg radiowy) lub za pośrednictwem dowolnych modułów (gdy element docelowy jest poza bezpośrednim zasięgiem źródła). Taka topologia ma sporo zalet:

- łatwość poszerzania sieci,
- możliwość stosowania w trudnodostępnych obszarach,
- taka sieć może być samonaprawialna, ponieważ kiedy któryś z elementów (modułów) przestanie działać, inne elementy automatycznie przejmują funkcje pośredniczenia przy przesyłaniu danych,
- przy zastosowaniu odpowiedniej konfiguracji można osiągnąć olbrzymi zasięg i dużą przepustowość,
- oszczędność energii, gdyż komunikacja występuje tylko na niewielkie odległości pomiędzy bezpośrednio sąsiadującymi elementami.

Podstawy budowy sieci IQMESH

Liczba urządzeń (modułów) w sieci IQMESH jest ograniczona do 65000 elementów (urządzeń)końcowych. Jednak pełne wsparcie ruchu pakietów danych w sieci ma maksymalnie 239 węzłów przypisanych do sieci koordynatora. Te węzły tworzą szkielet sieci (backbone). Z tego wynika, że pozostałe elementy sieci IQMESH mogą pracować w trybie peer-to-peer. To jeszcze nie wszystko. Każdy z modułów sieci może jednocześnie pracować w dwu niezależnych sieciach. W jednej sieci może być węzłem, a w drugiej koordynatorem sieci. Takie rozwiązanie daje możliwość komunikowania miedzy sobą nieograniczonej ilości modułów, bo nie ma ograniczenia dotyczącego ilości sieci.

Przykładową strukturę sieci MESH składająca się z 2 podsieci pokazano na **rysunku 1**. Zacieniony obszar sieci 1 obrazuje jej szkielet. Numery węzłów wewnątrz szkieletu nie są większe niż 239. Węzły należące do sieci, ale niebędące węzłami szkieletu mają numery większe. Koordynator „C" sieci 1 jest jednocześnie węzłem N5 sieci 2. Każdy z węzłów szkieletu sieci może pełnić funkcję routera. Routing pozwala na przesłanie danych pomiędzy dwoma węzłami nawet wtedy, kiedy te węzły nie mają bezpośredniego połączenia radiowego pomiędzy sobą. W takim przypadku węzły szkieletu pełnią funkcję „cyfrowych przekaźników radiowych" przekazujących pakiety pomiędzy sobą aż dotrą one do adresata. Każda komunikacja w systemie IQMESH jest kodowana. W trakcie przesyłania do danych użytkownika system OS dodaje do nich dodatkowe informacje (bonding, routing itp.).

Ograniczenia. Pomimo tego, że IQMESH pozwalaja na elastyczne i dynamiczne konfigurowanie (w tym zmianę topologii), to w pierwszym rzędzie jest przeznaczony dla mniej lub bardziej statycznych systemów. Dołączanie (bonding), lub odłączanie (unbonding) modułów do szkieletu sieci jest raczej procesem instalacyjnym, a nie elementem dynamicznej zmiany konfiguracji w trakcie pracy sieci. Węzły sieci mogą inicjować transmisję w każdym momencie, ale mocno zaleca się korzystanie z tej właściwości w ostateczności . W typowych aplikacjach transmisję inicjuje zawsze Koordynator.

Sieć IQRF. Pakiet danych w trybie peer-to-peer składa się z 3 pól: pola nagłówka (PAH), pola danych i sumy kontrolnej CRC. Przesyłane danych w trybie IQMESH jest o wiele bardziej skomplikowane. System IQRF OS automatycznie do pola danych użytkownika dodaje dodatkowe pola:

- Nagłówka składającego się z bajtów PIN, DLEN
- Pola informacji o sieci NTWINFO
- Pola sumy kontrolnej CRCS

Każde z tych pól jest zabezpieczone wielomianem kontrolnym CRC, a dodatkowo całość jest zabezpieczona końcowym wielomianem CRC (**rysunek 2**). Te dość silne zabezpieczenia przed przekłamaniami dodawane automatycznie przez IQRF OS świadczą o poważnym podejściu producenta do niezawodności przesyłania danych w projekcie IQMESH. Pierwszy bajt nagłówka PIN zawiera podstawowe informacje o trybie pracy (peer-to-peer, lub IQMESH), potwierdzeniach i routingu. Zawartość PIN pokazano na **rysunku 3**. Rejestr PIN jest głównie używany do zapisywania bitu _NTWF i ustawiania trybu pracy modułu w sieci, oraz do testowania flagi _ROUTEF. Wartość zapisana w _ROUTEF określa, czy odebrany pakiet danych ma być przesłany dalej, czy nie (routing). W bajcie DLEN jest zapisana ilość bajtów przesyłanych w polu danych użytkownika. Pole NTWINFO zawiera blok informacji o sieci o długości zależnej od zawartości rejestru PIN. Pierwsze 5 bajtów (od RX

PIN	DLEN	CRCH	Networing info	CRCN	User data	CRCD	CRCS
Nagłówek (PAH)			NTWINFO		DATA		CRC

CRCH – suma kontrolna nagłówka
CRCN – suma kontrolna pola NTWINFO
CRCD – duma kontrolna pola danych
CRCS – końcowa suma kontrolna

Rysunek 2. Struktura pakietu danych w trybie IQMESH

B7	B6	B5	B4	B3	B2	B1	B0
_NTWF _ACKF _ROUTEF			_CRYPTF _MPRWF		_SYSPF _TIMEF		_AUXF

_NTWF=0 tryb peer-to-peer
_NTWF=1 tryb IQMESH
_ACKF żądanie potwierdzenia – określa żądanie wysłania pakietu potwierdzenia jako odpowiedź na odebrany pakiet. Zawartość tej flagi musi być odczytywana i obsługiwana przez użytkownika, przez OS jest ignorowana
_ROUTEF =0 rutowanie nie jest wymagane dla wysyłanego pakietu
_ROUTEF=1 rutowanie jest wymagane. Przy tym ustawieniu muszą być zdefiniowane rejestry RTDT0-3
_CRYPTF żądanie kodowania – zarezerwowane dla przyszłych zastosowań
_MPRWF flaga przeznaczona dla specjalnych aplikacji i wspiera bezpośredni dostęp do peryferii i usług
_MPRWF=0 zapis/odczyt modułu peryferyjnego nie aktywny
_MPRWF=0 zapis/odczyt modułu peryferyjnego aktywny. Dodatkowo OS dodaje bajty MPRW0-2 do NTWIFO
_SYSPF i _TIMEF flagi używane przez OS i nie przeznaczone dla użytkownika
_AUXF zarezerwowane dla przyszłych zastosowań

Rysunek 3. Struktura rejestru PIN

RX	TX	CLID0	CLID1	PID	RT0TX	RTDEF	RTDT0-3	MPRW0-2

RX – adres docelowy pakietu równy numerowi węzła, który odbiera dane w sieci. RX musi być zapisany przez użytkownika przed wysłaniem pakietu.
TX – adres źródłowy pakietu równy numerowi węzła, który nadaje dane w sieci. Jest automatycznie nadawany przez OS w trakcie wywołania RFTXpacket(). Jeżeli węzeł nie jest podłączony to TX oznacza typ urządzenia: 0x00 koordynator, 0x01 węzeł, 0x02 brama
CLID0-1 ID podłączonego węzła. Jeżeli ma wartość 0xFFFF, to jest to ID niepodłączonego węzła.
PID – Identyfikator pakietu – rydunrk 6
RTOTX – wykorzystywany tylko przez OS
RTDEF – algorytm routingu – rysunek 5
RTDT0-3 – cztery bajty konfiguracyjne routingu
MPRW0-2 – trzy bajty zarezerwowane dla bezpośredniego dostępu do układów peryferyjnych

Rysunek 4. Pole NTWINFO

do PID) jest umieszczane w każdym pakiecie IQMESH. Jeżeli topologia sieci nie wymaga routingu, czyli bit _ROUTEF=0 (na przykład siec w topologii gwiazdy), to NTWINFO ma wspomnianą długość 5 bajtów. Po ustawieniu bitu _ROUTEF do pola NTWINFO jest dopisywanych 6 dodatkowych bajtów: RTOTX, RTDEF i RTDT0-3. Ustawienie _MPRWF skutkuje dopisaniem do tego pola dodatkowych 3 bajtów MPRW0-2. Jeżeli jest ustawiona flaga _CRYPTF, to dodawane są dodatkowe 2 bajty. Pole NTWINFO pokazano na **rysunku 4**.

Wysyłanie danych. Jak już wspomniałem jest możliwe przesyłanie danych pomiędzy modułami zaprogramowanymi do pracy peer-to-peer i w trybie IQMESH (zależnie do bitu _NTWF rejestru PIN). W trybie peer-to-peer wystarczy zapisać bufor bufferRF, określić ilość ważnych danych zapisując rejestr DLEN i wysłać dane wywołując funkcję RFTXpacket(). W trybie IQMESH przed wysłanie pakietu trzeba określić czy moduł pracuje jako węzeł, czy jako koordynator. Jeżeli pakiet ma być routowany, to konieczne jest określenie parametrów przez zapisanie rejestrów RTDEF i RTDT0-3. Węzeł musi być wcześniej dołączony (boned) do sieci koordynatora. Przed wysłaniem pakietu użytkownik musi tez zapisać numer węzła docelowego – rejestr RX. W komunikacji dwukierunkowej PIN i DLEN musi być zapisywany przed każdym wysłaniem danych następującym po uprzednim odebraniu danych.

Odbieranie danych. Funkcja *RFRXpacket ()* pozwala na odebranie pakietu danych droga radiową. Funkcja kończy swoje działanie po prawidłowym odebraniu pakietu lub po upłynięciu timeoutu. Czas oczekiwania na odebranie pakietu jest programowany przez użytkownika poprzez zapisanie zmiennej systemowej *toutRF*. Timeout jest odliczany jako ilość tick'ów. Czas tick'u jest zależny od trybu TX mode ustawianego funkcją *setRFmode()* i wynosi 10 ms dla STD RX, 40 ms dla LP RX i 600 ms dla XLP RX. Upłynięcie timeout'u w trakcie działania *RFRXpacket()* powoduje jej przerwanie z jednym wyjątkiem. Po zaprogramowaniu opcji WAIT Packet End działanie funkcji nie jest przerywane. Dla zapewnienia prawidłowego odbioru można testować poziom sygnału funkcją *checkRF()* i jeżeli ten poziom jest niższy od założonego pakiet może być programowo ignorowany. Po prawidłowym odebraniu pakietu dane użytkownika są zapisywane w buforze *bufferRF*, a ilość ważnych danych w rejestrze DLEN.

Dołączanie węzłów do sieci koordynatora – bonding. Jak już wspomniałem szkielet sieci jest tworzony przez węzły dołączane do koordynatora sieci. Koordynator przydziela każdemu węzłowi w sieci unikalny numer z zakresu 1…239. Ten numer można traktować jako adres węzła. Bonding jest zazwyczaj wykonywany w trakcie procesu instalacyjnego. Producent zaleca, żeby moduły pracowały wtedy z małą mocą nadajnika, a odległość pomiędzy dołączanym modułem i modułem koordynatora nie była duża. Jest to konieczne, żeby dołączyć właściwy moduł, a nie na przykład moduł który nie jest w szkielecie sieci i ma pracować w trybie peer-to-peer.

Do dołączania przez koordynatora nowych węzłów do szkieletu sieci jest wykorzystywana funkcja *bondNewNode(address)*. Uruchomienie tej funkcji w module jest możliwe tylko pracującym jako koordynator. Funkcja *bondNewNode(address)* wyszukuje droga radiową wolnego (jeszcze niedołączonego) modułu i przydziela mu numeru (adresu) z argumentu funkcji. Argument może mieć wartość z zakresu 1…239. Jeżeli dołączenie powiedzie się, to funkcja zwraca wartość 1. Do zmiennej systemowej *param2* jest wpisywany numer przydzielonego węzła, a w pakiecie zwrotnym w komórkach *bufferRF[0]* i *bufferRF[1]* są wpisywane 2 młodsze bajty numeru identyfikacyjnego module ID dołączonego modułu. Jeżeli przypisanie się nie powiedzie to funkcja zwraca wartość 0.

Jeżeli argument *address* jest zerowy, to koordynator przydziela pierwszy wolny numer (ostatnio przydzielony +1). Można to wykorzystać do przydzielania nowych numerów węzłów bez obawy o przerwy w numeracji. Można też wywołać funkcję *bondNewNode(address)* z argumentem równym 0xFE. Moduł z tym adresem jest przydzielony do szkieletu sieci, ale nie może brać udziału w routingu pakietów. W czasie działania funkcji *bondNewNode(address)* w przyłączanym module musi być aktywna funkcja *bondRequest()*. Wynik przyłączenia jest zapisywany w pamięci EEPROM koordynatora (dopisanie do listy węzłów) i w pamięci EEPROM węzła. Przyłączony węzeł może być usunięty z listy węzłów koordynatora funkcją *removeBonedNode(addr)*. Numer usuwanego węzła jest podawany w argumencie funkcji. Ponowne dopisanie usuniętego węzła do listy koordynatora wykonuje funkcja *rebondNode(addr)*. Usuniecie wszystkich węzłów z listy koordynatora jest wykonywane po wywołaniu funkcji *clearAllBonds*. Koordynator może w dowolnym momencie zapytać czy węzeł o określonym numerze jest dopisany do listy węzłów koordynatora wywołując funkcję *isBondedNode(addr)*. Jeżeli węzeł jest na liście to funkcja zwraca 1. W przeciwnym przypadku zwraca 0.

Jak wiemy w trakcie procesu dołączania w module musi być uruchomiona funkcja *bondRequest()*. Kiedy koordynator połączy się z modułem drogą radiową i dołączenie zostanie wykonane prawidłowo, OS zapisuje niezbędne informacje w pamięci EEPROM modułu węzła i funkcja kończy swoje działanie. Po prawidłowym przyłączeniu do szkieletu sieci funkcja *bondRequest()* zwraca 1, a w zmiennej systemowej *param2* jest zapisywany numer węzła. Jeżeli przyłączenie się nie powiedzie funkcja zwraca 0. Wyjście z trybu przyłączenia i wejście w tryb pracy w sieci wymaga wykonania funkcji *setNodeMode()*.

Usunięcie węzła z sieci wykonuje funkcja *removeBond()*. Trzeba pamiętać, że jest to tylko usunięcie po stronie węzła. W tym przypadku nie ma połączenia radiowego z koordynatorem i informacja o usunięciu do niego nie dociera. Jeżeli węzeł ma być usunięty również z listy koordynatora, to trzeba

B7	B6	B5	B4	B3	B2	B1	B0
VSCLID					RAL		

RAL – algorytm routingu
VSCLID – wirtualne ID

Rysunek 5. Struktura rejestru RTDEF

B7	B6	B5	B4	B3	B2	B1	B0
FROMC Reserve HCOUNT							

FROMC=1 pakiet pochodzi od koordynatora
FROMC=0 pakiet pochodzi z węzła
HCOUNT – licznik skoków w procesie routowania

Rysunek 6. Struktura rejestru PID

Algorytm	RTDEF	Zakres adresów	Adresowanie przez	adresowanie	Adres rozgłaszania
SFM	0x01 1…239 Adres logiczny			RX	0xFF
DFM	0x02	1…239	Adres logiczny	RX	0xFF
DFM2B	0x42	1..65000	Adres użytkownika	RTDT3=addrH RX=addrL	0xFFFF
Tree	0x08	1…239	Logiczny adres	RX	0xFF

Rysunek 9. Adresowanie IQMESH

na nim wywołać *removeBondNode(addr)*. Ponowne dołączenie do sieci jest możliwe tylko przez koordynatora po wywołaniu funkcji *bondNewNode* po stronie koordynatora i funkcji *bondRequest* po stronie węzła. Funkcja *amIBonded()* zwraca 1, gdy węzeł jest dołączony do szkieletu sieci. Jeżeli nie jest dołączona zwraca 0. Działa prawidłowo po prawidłowym wykonaniu funkcji *bondRequest()* i funkcji *removeBond()*. Jak widać pełne połączenie radiowe pomiędzy koordynatorem i węzłem jest wspierane przez OS tylko w czasie dołączania węzła do sieci. Jeżeli jest konieczna zdalna rekonfiguracja sieci, to musi być wykonywana w warstwie aplikacji użytkownika.

W sieci mogą pracować moduły niedołączone do szkieletu przez koordynatora. Do ich adresowania można wykorzystać 4-bajtowy numer identyfikacyjny Module ID. Po wykonaniu funkcji *moduleInfo()* do bufora *bufferINFO* jest wpisywanych 8 bajtów zapisanych w pamięci modułu przez producenta. 4 najmłodsze bajty to pole *Module ID*. Najstarszy bajt określa czy moduł jest węzłem (wartość 0x00), czy koordynatorem (wartość 0x01). Trzy najmłodsze bajty to numer seryjny modułu.

Routing pakietów danych. Routing umożliwia przesyłanie pakietów danych do odbiorcy (koordynatora lub węzła), który jest poza zasięgiem nadawcy (węzła lub koordynatora). Pozwala to na zmniejszenie mocy nadajników modułów radiowych, zwiększenie pewności przesyłania danych i odporności na zakłócenia. Poza tym moduły radiowe mogą dynamicznie zmieniać swoje położenie na przykład w aplikacji nadzorującej położnie osób, czy maszyn. Wszystkie węzły przyłączone przez koordynatora do szkieletu sieci mogą spełniać role routerów, ale w każdym węźle można to indywidualnie włączać lub wyłączać. Aby pakiety mogły niezawodnie docierać do miejsca przeznaczenia topologia sieci powinna być tak zaprojektowana by każdy przesyłany pakiet miał przynajmniej 2 niezależne drogi przesłania.

Żeby routing był możliwy muszą być spełnione następujące warunki:

- Moduły spełniające rolę routera muszą być dołączone do szkieletu sieci (Bonded)
- Pakiet został wysłany przez nadawcę z ustawioną flagą _ROUTEF
- W module routera jest aktywna funkcja RFRXpacket(), kiedy routowany pakiet jest przesyłany
- Flaga _ROUTEF jest powiązana z wychodzącym pakietem i nie ma żadnego wpływu na rutowanie przychodzących pakietów

Listing 1. Pętla routera

```
setNodeMode();
while(1)
{
  RFRXpacket();
  clrwdt();
}
```

Na **listingu 1** pokazano programową obsługę routingu pakietu w trybie STD (ustawianego funkcją *SetRFmode()*. Wszystkie czynności związane z routingiem są wykonywane przez OS w tle i nie są widoczne przez użytkownika. Routing pakietów danych w sieci IQMESH wykorzystuje mechanizm „skoków" (hops) pomiędzy modułami mogącymi nawiązać między sobą łączność radiową. Zależnie od konfiguracji i przeznaczenia sieci można wybrać jeden z 5 zaimplementowanych algorytmów.

Jak wiemy ustawienie flagi _ROUTEF powoduje dopisanie do pola NTWINFO 4 dodatkowych bajtów konfiguracyjnych RTDT0-3:

RDT0: zawiera ilość skoków na pakiet w zakresie od 0 do 239. Wpisanie zera oznacza przesłanie bezpośrednie (bez routingu). Liczba skoków jest powiązana z topologią sieci i na przykład może być równa numerowi węzła do którego jest wysyłany pakiet z koordynatora. RTDT0 jest dekrementowany przy każdym skoku.

RTDT1: czas szczeliny czasowej w tickach. Powinien być większy niż czas przesłania pakietu danych. Jest zależny od długości pola danych, wartości rejestru PIN, DLEN, trybu RF (ustawianego setRFmode()) i prędkości transmisji. Można go wyliczyć w przybliżeniu dla prędkości 19,2 kb/s: tryb STD 1 (lub2) +1 dla każdych 24 bajtów pola danych; tryb LP: 5(lub 6) +1 dla każdych 24 bajtów pola danych; tryb XLP: 120.

RTDT2: DID identyfikator algorytmu Discovery – nie jest zapisywany przez użytkownika.

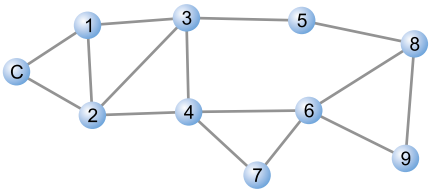
RTD3: starsza część adresu użytkownika – tylko w trybie kiedy takie adresowanie jest używane.

Pakiet jest dostarczany w czasie równym RTDT1 (czas trwania szczeliny)×RTDT0 (liczba skoków) i nie może być potwierdzany do momentu, aż ten czas nie upłynie. Niezachowanie tego warunku spowoduje kolizję w modułach routerów. Pierwsza szczelina czasowa jest przeznaczona dla koordynatora.

Rodzaj algorytmu jest zapisywany w polu RAL rejestru RTDEF (rysunek 5). W sieci IQMESH można wybrać jeden z 5 algorytmów:

- SFM Static Full MESH
- DFM Discovered Full MESH
- DOM Discovered Optymized MESH
- DFM2B Discovered Full MESH 2B
- Tree

Algorytm SFM wykorzystuje do adresowania modułów adres logiczny przypisany przy procesie dołączania (RX= logical address). Wektor rutowania jest statyczny (1. 2…239). Przyłączanie musi być wykonane przed umieszczeniem modułu w miejscu docelowym i adresy modułów musza być



Rysunek 7. Przykładowa topologia sieci przy routowaniu SFM

znane. Topologia sieci zależy od adresów. Moduły rozmieszcza się tak, by ze wzrostem odległości od koordynatora rosły adresy modułów. Na **rysunku 7** pokazano przykładową topologia sieci dla routingu SFM.

Ograniczeń topologii nie ma wbudowana w IQMESH funkcja Discovery i związane z nią algorytmy DFM, DOM i DFM2B. Węzły sieci mogą tak jak w przypadku SFM być rozmieszczane w stałym porządku określonym przez numery węzłów, lub w dowolnie (losowo) i właśnie dla możliwości losowego rozmieszczenia węzłów została zaprojektowana funkcja Discovery. Discovery tworzy z dołączonych węzłów sieci nowy wewnętrzny logiczny szkielet sieci (backbone) i ścieżka routowania jest znajdowana automatycznie. Węzły są przydzielane do stref (grup węzłów które mogą zostać osiągnięte przy takiej samej liczbie skoków). Liczba stref może być limitowana przez użytkownika.

Algorytm DFM wykorzystujący funkcję Discovery nie wymaga znajomości numerów węzłów, a same węzły mogą być rozmieszczone losowo. Discovery może być uruchomiony po rozmieszczeniu i dołączeniu modułów do szkieletu sieci przez koordynatora funkcją *bondNewNode()*. Routing używa wirtualnych adresów VRN (*Virtual Routing Numbers*) tworzonych w rezultacie procesu Discovery. Użycie automatycznie utworzonych przez OS VRN i stref powoduje, że w czasie routingu znajdowana jest najkrótsza droga pomiędzy koordynatorem a węzłem. Po każdej zmianie topologii sieci trzeba wywołać w programie koordynatora funkcję systemową *discovery(zones)*. Jej argumentem jest maksymalna liczba ustanawianych stref. Jest ona równa maksymalnej ilości skoków routowania na pakiet danych. Funkcja *discovery()* zwraca liczbę węzłów VRN przydzielonych do sieci Discovery.

Adresowanie węzła docelowego jest dokładnie takie samo, jak w wypadku *SFM(RX= logical address)*. Na **rysunku 8** pokazano schemat topologii sieci. Logiczne numery węzłów otworzone w czasie bondingu są symbolicznie umieszczone wewnątrz okręgów symbolizujących węzły. Wirtualne adresy VRN są umieszczone obok węzłów. Strefy Discovery są oznaczone jako Z1, Z2, Z3 i Z4. Algorytm DOM jest odmianą DFM. Różnica polega na tym, że w DOM jest optymalizowana (redukowana) liczba skoków przez wywołanie funkcji optymizeHops(). W wyniku

Listing 2. Inicjalizacja program koordynatora

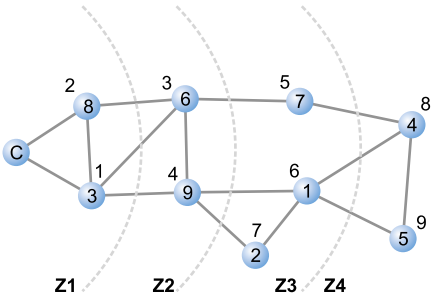
```
//deklaracja zmiennych pomocniczych
uns8 i, myTIMESLOT, myPOWER, myRTDEF, lastPID;
bit myRouting;
myTIMESLOT = 10;    // ustawienie szczeliny czasowej większej
                    // od czasu przesyłania routowanego pakietu
myPOWER = 7;  //ustawienie mocy nadajnika
myRTDEF = 1;  //routing SFM bez discovery
myRouting = 1;  //włączenie routingu
disableRFPGM();
enableSPI();  //odblokowanie SPI do komunikacji z hostem
setRFmode(_WPE); //tryb STD RX mode-czekanie na koniec pakietu
toutRF = 3;  //[ticks] musi być większe niż długość pakietu RF; dla
             //odbioru z węzła DLEN=17 i toutRF=3
appInfo();
copyBufferINFO2COM();
startSPI(32);  // informacja o pakiecie przesłana do hosta
_LEDG = 1;  // sygnalizacja zerowania
waitDelay(100);
_LEDG = 0;
PID = 0;
lastPID = 0;
```

Listing 3. Odebranie pakietu danych przez radio i przesłanie go do hosta

```
while (1)//początek pętli koordynatora
{
  clrwdt();
  if (checkRF(RX_FILTER))
  {
    if (RFRXpacket())  //czy coś odebrano przez radio?
    {
      if (RX == 0xFF)  //czy jest to broadcast?
      continue;
      if (PID == lastPID)  //taki pakiet był juz raz odebrany
      continue;
      lastPID = PID;  //ten pakiet został odebrany pierwszy raz
      bufferCOM[0] = 'T';  //tworzenie komunikatu dla hosta
      bufferCOM[1] = 'X';
      bufferCOM[2] = ':';
      bufferCOM[3] = '#';
      bufferCOM[4] = TX;
      bufferCOM[5] = ' ';
//skopiowanie odebranego pakietu danych do bufora bufferCOM
      copyMemoryBlock(bufferRF, bufferCOM + 6, DLEN);
      startSPI(6 + DLEN); // przesłanie do hosta przez SPI
    }
  }
  while (getStatusSPI())
  continue;
```

Listing 4. Odebranie, interpretacja i wykonanie komendy

```
if (_SPIRX) //czy odebrano coś z hosta przez SIP?
{
  if (!_SPICRCok)  /jeżeli CRC jest błędna to restart SPI
  goto restart_SPI;
  switch (bufferCOM[0])   // testowanie kodu komendy
    case 'g':  //tryb debugowania
      debug();
      break;
    case 'c':  //usuń wszystkie połączenia
      clearAllBonds();     /
      break;
    case 'W':  //ustawienie poziomu sygnału
      if (SPIpacketLength < 2)  // spodziewane 'W'<level>
      goto restart_SPI;
      myPOWER = bufferCOM[1];
      break;
    case 'T':  // ustawienie szczeliny czasowej
      if (SPIpacketLength < 2)  // spodziewane 'T'<time>
      goto restart_SPI;
      myTIMESLOT = bufferCOM[1];
      break;
    case 'R':  // włączenie/wyłączenie routingu
      if (SPIpacketLength < 2)  //spodziewane 'R'<value>
      goto restart_SPI;
      myRouting = 1;  //ON
      if (bufferCOM[1] == 0) myRouting = 0; //OFF
      break;
    case 'F':  // ustawienie RTDEF
      if (SPIpacketLength < 2)  // spodziewane'F'<value>
      goto restart_SPI;
      myRTDEF = bufferCOM[1] & 0x07;
      break;
    case 'I':  // pobranie informacji o systemie
      showSystemParams();
    case 'i':  // pobranie bieżących ustawień
      stopSPI();
      bufferCOM[0]  = 'W';
      bufferCOM[1]  = '=';
      bufferCOM[2]  = '#';
      bufferCOM[3]  = myPOWER;
      bufferCOM[4]  = ' ';
      bufferCOM[5] = 'T';
      bufferCOM[6] = '=';
      bufferCOM[7] = '#';
      bufferCOM[8] = myTIMESLOT;
      bufferCOM[9] = ' ';
      bufferCOM[10] = 'F';
      bufferCOM[11] = '=';
```



Rysunek 8. Topologia sieci Discovery i routing DFM

jej działania zapisywany jest rejestr RTDT0. Powoduje to szybsze dostarczanie pakietu kosztem niezawodności jego dostarczania (redukowane są niezależne drogi).

Zasada działania algorytmu DFM2B jest bardzo podobna do DFM. Jedyna różnica to możliwość dołączenia 65000 węzłów i stworzenia nich wirtualnego szkieletu sieci Discovery. Zasadniczy szkielet sieci ma maksymalnie 239 węzłów. Do adresowania węzłów używany jest 2 bajtowy adres. Młodszy bajt adresu jest standardowo zapisywany w rejestrze RX, a starszy w rejestrze RTDT3 z pomocą funkcji *setUserAddress(addr)*. W DFM2B nie stosuje się optymalizacji skoków *optymizeHops()*. Algorytm *Tree* dopuszcza obecność tylko jednego routera w strefie. Pakiety danych są dostarczane najszybciej jak to tylko możliwe, ale nie ma wcale dróg alternatywnych i pewność dostarczania nie jest wysoka. Algorytm *Tree* jest przeznaczony dla transferu z węzła do koordynatora. W obecnej wersji OS ten algorytm nie jest w pełni zaimplementowany i przetestowany.

Praktyczna realizacja sieci IQMESH

Konfigurowanie i działanie sieci IQMWSH najlepiej jest sprawdzić w praktyce. Producent przygotował przykład E11 składający się z plików źródłowych: jeden dla koordynatora IQMESH-C.c i drugi dla węzłów sieci IQMESH-N.c.

Analizę projektu siec rozpoczniemy od koordynatora. Moduł TR52B pełniący funkcje koordynatora musi być połączony z hostem poprzez interfejs SPI. W czasie testów rolę hosta pełni aplikacja IQRF IDE uruchomiona na komputerze PC i połączona z modułem poprzez USB i moduł CK-USB-04. Po zerowaniu moduł oczekuje na komendę wysłaną przez hosta przez SPI. Komendy są przeznaczone do kontrolowania sieci: konfiguracji i przesyłania danych. Do prawidłowego działania przykładów jest potrzeby IQRF IDE w wersji 2.07 i wyższej oraz IQRF OS w wersji 3. W katalogu projektu jest umieszczone zdefiniowane makro *E11-IQMESH-C.mcr*. to makro trzeba zaimportować do IQRF IDE.

Host może przesyłać do koordynatora następujące komendy:

'c' – zeruj wszystkie połączenia koordynatora.
'g' – wprowadź tryb debugowania.
'i' – pobierz informację o konfiguracji.
'I' – pobierz informację i węźle.
'@' – uruchom Discovery.
'b' <addr> – przyłącz (Bond) nowy węzeł <addr> z zakresu 1…239, 0 –automatyczne adresowanie. Przykład: „b#003" przyłącz węzeł o logicznym adresie 3
'u'<addr> odłącz węzeł o adresie addr z listy koordynatora i z węzła. Adres 0xff usuwa wszystkie połączenia. Przykład „u$0xff" – odłącz wszystkie węzły od koordynatora
'W'<level> ustaw moc nadajnika. Argument <level> ma zakres 0…7.
'T'<time> Ustaw szczelinę czasową. Argument <time>ma zakres 0..255. Przykład: „T#005" ustawia szczelinę na 50 ms.
'F'<value> ustawia algorytm routowania value=1 SFM, value=2 DFM. Przykład „F#002 algorytm DFM (Discovery).
'R'<value> włączenie/wyłączenie routingu 0= wyłącz, 1 włącz Przykład „R#001" włącz routing.
't'<level> ustaw moc nadawania TX do celów testowych. Argument <level> ma zakres 0...7.
's'<addr> ustaw węzeł o adresie zawartym w argumencie addr w stan uśpienia (Steep). Przykład „s#010" uśpij węzeł o adresie 10. Zakres adresów 1…239, adres 0xFF ogłoszeniowy (do wszystkich węzłów).
'r'<addr> zeruj adres węzła o numerze zawartym w argumencie addr. Przykład „r#008" zeruj węzeł o adresie 10, lub „r" zeruj wszystkie węzły.
'p'<addr><cmd_LED> wyślij komendę cmd_LED sterowania diodą LED w węźle addr. Komenda nie wymaga potwierdzenia. Komendy: 0 – wyłącz diodę, 1 – załącz diodę, 2 – włącz miganie diody.
'?'<addr><cmd_LED> – komenda działa tak samo jak 'p', tylko wymaga potwierdzenia

Po włączeniu lub zerowaniu modułu TR52B pełniącego rolę koordynatora zapisywane są zmienne inicjujące wartościami domyślnymi: szczelina czasowa *myTIMESLOT=10*, moc nadajnika *myPOWER=7*, algorytm *routingu myRTDEF=1* (SFM bez Discovery). Tryb odbierania pakietów ustawiany jest komendą *setRFmode(_WPE)* na czekanie na pakiet niezależnie od ustawionego timeout'u (**listing 2**).

Obsługa koordynatora odbywa się w pętli nieskończonej i jest podzielona na kilka etapów. Pierwszy z nich, to sprawdzenie czy moduł odebrał jakiś pakiet danych. Sygnał radiowy musi być silniejszy ustawionego przez wartość *RX_FILTER*. Wykrycie, że jest to pakiet rozgłoszeniowy o adresie 0xFF (przeznaczony dla wszystkich węzłów) powoduje, że program wraca do początku oczekiwania na pakiet danych. Jeżeli jednak nie jest to pakiet roz-

Listing 4. c.d.

```
      bufferCOM[12] = '#';
      bufferCOM[13] = myRTDEF;
      bufferCOM[14]= ' ';
      bufferCOM[15] = 'R';
      bufferCOM[16] = '=';
      bufferCOM[17] = '#';
      bufferCOM[18] = 1;
      if (myRouting == 0) bufferCOM[18] = 0;
      startSPI(19);  //I przesłanie przez SPI do hosta
      pulseLEDG();
      continue;
    case 't':
      if (SPIpacketLength < 2)  // spodziewane 't'<level>
      goto restart_SPI;
      //poziom 0…7
      setTXpower(bufferCOM[1] & 0x07);
      bufferCOM[1] = 0;
      goto sendPacket;
    case 'p':  //pakiet danych bez potwierdzenia
    case '?':  // pakiet danych z potwierdzeniem
      if (SPIpacketLength < 3)  //spodziewane <cmd><addr><LED_cmd>
      goto restart_SPI;
      setTXpower(myPOWER);
      bufferRF[2] = bufferCOM[2];  // komenda LED
      RTDT1 = myTIMESLOT;  // szczelina czasowa
      goto sendPacket;
    case 'u':  // usunięcie połączeń koordynatora i węzłów
      if (SPIpacketLength < 2)   // spodziewane 'u'<addr>
      goto restart_SPI;
      if (bufferCOM[1] < 0xF0)  // max. adres węzła 239
      removeBondedNode(bufferCOM[1]);
    case 'r':  // zerowanie adresów węzłów
    case 's':  // uśpienie węzłów
      setTXpower(7);
      RTDT1 = 2;  //szczelina czasowa
      //kompletowanie pakietu danych wysyłanych do węzłów
      sendPacket:
      bufferRF[0] = 0xEE;  //identyfikacja pakietu
      bufferRF[1] = bufferCOM[0];  //komenda
      PIN = 0;
      //czy komenda bez adresu?
      if ((SPIpacketLength == 1) || (bufferCOM[1] == 0))
      setNonetMode();  //wysłanie do wszsytkich modułów - tryb
                       //z wyłączeniem sieci
      else
      {  // pakiet wysyłany siecią
        setCoordinatorMode();  //tryb koordynatora
        _ROUTEF = myRouting;  //włączenie routingu
        RX = bufferCOM[1];  //adres węzła
        RTDEF = myRTDEF;  //algorytm routingu
        RTDT0 = eeReadByte(0x00);  // SFM: liczba skoków węzłom
        if (RTDEF > 1)  //czy discovery
        {
          if (isDiscoveredNode(RX)) // czy numer węzła jest w sieci
          optimizeHops(0xff);  //RTDT0 jest ustawiane automatycznie
        }
      }
      DLEN = 3;
      PID++;  //każdy pakiet ma inny identyfikator
      RFTXpacket();
      if (bufferCOM[0] == '?')
      {
        pulseLEDG();
        startSPI(0);
        continue;//z potwierdzeniem
      }
      break;
    case '@':  //uruchom Discovery
      pulseLEDG();
      setTXpower(myPOWER);
      i = discovery(eeReadByte(0));  //maksymalna liczba stref
                       //jest równa liczbie dołączonych węzłów
      SWDTEN = 1;
      bufferCOM[0]  = 'D';
      bufferCOM[1]  = ':';
      bufferCOM[2]  = '#';
      bufferCOM[3]  = i;  // liczba węzłów discovery
      bufferCOM[4]  = ' ';
      bufferCOM[5]  = '/';
      bufferCOM[6]  = ' ';
      bufferCOM[7]  = 'B';
      bufferCOM[8]  = ':';
      bufferCOM[9]  = '#';
      bufferCOM[10] = eeReadByte(0);
      startSPI(11);
      continue;
    case 'b':
      if (SPIpacketLength < 2)  // spodziewane 'b'<value>
      goto restart_SPI;
      pulsingLEDR();
      if (bondNewNode(bufferCOM[1]))
      {
        bufferCOM[0] = 'O';
        bufferCOM[1] = 'K';
        bufferCOM[2] = ':';
        bufferCOM[3] = '#';
        bufferCOM[4] = param2; // adres węzła
        bufferCOM[5] = ' ';
        bufferCOM[6] = '$';
        bufferCOM[7] = bufferRF[1]; // Node ID [1]
        bufferCOM[8] = '$';
        bufferCOM[9] = bufferRF[0]; // Node ID [0]
```

głoszeniowy, to pole identyfikatora PID jest porównywane ze zmienna lokalną *ladtPID*. Jeżeli obie wartości są sobie równe, to taki pakiet jest ignorowany i oznacza to, że taki sam pakiet już został odebrany (poprzez inną drogę routingu). Jeżeli PID nie jest równy *lastPID*, to zawartość odebranego pakietu jest wpisywana do *bufferCOM* od adresu 0x06 i przesyłana poprzez SPI do hosta (**listing 3**). W kolejnym kroku program sprawdza czy odebrano komendy z hosta przez SPI. Jeżeli tak, to komenda jest kompletowana i wysyłana (**listing 4**).

Komendy t, p, u, ?, r, oraz s są przesyłane do węzłów. Kompletowanie zwartości bufferRF, oraz ustawień sieciowych jest wykonywane we fragmencie od etykiety sendPacket. Komendy, które nie mają argumentu w postaci adresu są wysyłane do wszystkich węzłów jednocześnie przy wyłączonym trybie sieciowym (komenda setNonetMode()). Jeżeli komenda, która ma być przesłana do węzła ma argument, to jest prowadzany tryb koordynatora przez funkcję setCoordinatorMode();. Konieczne jest wtedy zapisanie ustawień sieciowych: routingu, adresu węzła, algorytmu routowania i ilości skoków. Obsługa węzła zaczyna się od testowania czy jest on przyłączony do koordynatora. Jeżeli nie to wykonywana jest pętla z **listingu 5**.

Moduł TR52B musi być umieszczony w module ewaluacyjnym DK-EVAL-03 lub DK-EVAL-04.Węzeł jest przyłączany do sieci po wywołaniu funkcji *bondNewNode()* po stronie koordynatora i funkcji *bondRequest()* po stronie przyłączanego węzła. Koordynator uruchamia przyłączenie węzła po otrzymaniu z hosta komendy *b<addr>*. W przyłączanym module trzeba wtedy włączyć zasilanie i w ciągu 20 sekund przycisnąć w module DK-EVAL przycisk. Spowoduje to uruchomienie funkcji *bondRequest()* i proces przyłączania może zostać wykonany. Przypomnę, że producent zaleca wykonywać przyłączanie kiedy oba moduły są blisko siebie i pracują z małymi mocami nadajników, tak aby mieć pewność, że to właśnie ten moduł został przyłączony, a nie inny umieszczony w sieci i przeznaczony do pracy jako urządzenie końcowe. Jeżeli w ciągu 20 sekund przycisk nie zostanie przyciśnięty, to jest wywoływana funkcja *mySleep()*. Ta funkcja usypia moduł, ale przedtem przygotowuje

Listing 4. c.d.

```
        i = 10;
      }
      else
      {
        bufferCOM[0] = 'E';  // bond error
        bufferCOM[1] = 'R';
        bufferCOM[2] = 'R';
        bufferCOM[3] = 'O';
        bufferCOM[4] = 'R';
        i = 5;
      }
      stopLEDR();
      startSPI(i);
      continue;
      default:
      goto restart_SPI;
    }
  }
}
```

Listing 5. Pętla przyłączenia węzła do koordynatora

```
while (!amIBonded())  //czy węzeł jest przyłączony do sieci
{
  sleepTimeout = 0;
  while (!buttonPressed)  //przycisk musi być przyciśnięty
                          //przed dołączeniem
  {
    clrwdt();
    if (sleepTimeout>100)  //uśpienie kiedy przycisk nie jest
                           //przyciśnięty w ciągu 20 s.
    {
      mySleep();
      break;  //po wybudzeniu do procedury przyłączenia
      }
        sleepTimeout++;
        pulseLEDR();  // wait for button press indication
        waitDelay(25);
      }
      stopLEDR();
      _LEDR = 1;
        bondRequest();  //czekaj na dane do przyłączenia
      _LEDR = 0;
      waitDelay(50);
      clrwdt();
    }
  }
}
```

REKLAMA

go do wybudzenia poprzez skonfigurowanie przerwania od zmian stanów na liniach wejściowych PORTB. Po przyciśnięciu klawisza w uśpionym module zostanie zgłoszone przerwanie od zmiany stanów PORTB, moduł zostanie wybudzony i program przejdzie do wykonywania funkcji *bondRequest()*.

Listing 6. Pętla główna programu węzła

```
if (RFRXpacket()) //czy odebrano pakiet danych
{
  if (wasRouted())  // czy odebrany pakiet routingu
  pulseLEDG();
  if (_ROUTEF)   // czy pakiet ma włączony routing
  {  //tak - czekanie na koniec routingu
    while (RTDT0)  //w  RTDT0 ilość skoków do wykonania
    {
      waitDelay(RTDT1);  // RTDT1 - szczelina czasowa
      RTDT0--;
    }
  }
  if (bufferRF[0] == 0xEE)  //czy to pakiet przesłany z koordynatora
  {
    bufferRF[0] = 0x00;  //tak-zeruj tą informację
    switch (bufferRF[1])  //testowanie komendy
    {
      case 'p':  //potwierdzenie nie jest wymagane
      case '?':  //potwierdzenie jest wymagane
      switch (bufferRF[2])  //komenda sterownia diodą LED
      {
        case 0:
          _LEDR = 0;
          break;
        case 1:
          _LEDR = 1;
        break;
          case 2:
          pulseLEDR();
          break;
      }
      if (bufferRF[1] == '?')  //kompletowanie i wysyłanie potw.
      {
        if (RX != 0xFF)  //to do not answer broadcast packets
        {
          waitDelay(5);  //rekomendowane opóźnienie pomiędzy
                         //odbiorem i nadawaniem
          moduleInfo();
          bufferRF[0] = 'I';
          bufferRF[1] = 'D';
          bufferRF[2] = ':';
          bufferRF[3] = '$';
          bufferRF[4] = bufferINFO[2];// module ID
          bufferRF[5] = '$';
          bufferRF[6] = bufferINFO[1];
          bufferRF[7] = '$';
          bufferRF[8] = bufferINFO[0];
          bufferRF[9] = ' ';
          bufferRF[10] = 'R';
          bufferRF[11] = 'S';
          bufferRF[12] = 'S';
          bufferRF[13] = 'I';
          bufferRF[14] = ':';
          bufferRF[15] = '#';
          bufferRF[16] = lastRSSI;
          DLEN = 17;
          getNetworkParams();  // adres węzła w  param2
          if (RTDEF == 1)
          RTDT0 = param2;  //SFM: ilość skoków dla potwierdzenia
                           //= adres tego węzła
          else  //algorytm DFM
          {
            RTDEF = 0x02;
            RTDT0 = 0xFF;  // DOM: optymalizacja liczby skoków
          }
// RTDT1 (timeslot) taki sam jak w odebranym pakiecie
          RX = 0;  // wysyłamy do koordynatora
        RFTXpacket();
        }
      }
      break;
      case 't':  //testowanie zakresu
        pulseLEDG();
        pulseLEDR();
        break;
      case 'r':  //reset
       pulseLEDR();
       waitDelay(5);
       reset();
      case 'u':  //odłączenie od sieci
        pulseLEDR();
        setNodeMode();
        removeBond();
        reset();
      case 's':  //uśpienie
         _LEDG = 1;
        SWDTEN = 0;
        startLongDelay(400);
        while (isDelay());
        mySleep();
        break;
      default:
        break;
    }
  }
}
```

Po dołączeniu do sieci program wychodzi z pętli i przechodzi do sprawdzania czy poziom odbieranego sygnału nie jest mniejszy niż założono. Zapobiega to błędom działania przy zbyt słabym, zakłóconym sygnale. Jeżeli sygnał jest wystarczający, to testujemy, czy został odebrany pakiet danych (**listing 6**). Po odebraniu pakietu program sprawdza czy odebrany pakiet nie jest przeznaczony do routingu. Ustawienie flagi _ROUTEF wymusza odczekania czasu potrzebnego na zakończenie routowania pakietu, Dopiero po tym czasie węzeł może wysłać swoje dane.

Koordynator przesyła w polu danych na pierwszej pozycji *bufferRF[0]* bajt 0xEE identyfikujący naszą sieć – jest to identyfikacja na poziomie aplikacji użytkownika. Po zidentyfikowaniu sieci program testuje zawartość *bufferRF[1]*, w którym jest zapisany kod komendy przesyłany przez koordynatora i bajtu *bufferRF[2]* z kodem komendy sterowania diodą LED.

Podsumowanie

Małe sieci radiowe przeznaczone do systemów nadzoru i akwizycji danych są coraz bardziej popularne, co widać po coraz szerszej ofercie wielu producentów. Konkurencja staje się coraz większa co widać po szybko malejących cenach i coraz większych możliwościach sieciowych modułów radiowych.

Sieć tworzona za pomocą modułów TR52B i przy wsparciu OS IQRF ma spore możliwości przy wykorzystaniu niewielkich zasobów sprzętowych. Takie funkcje, jak zdalne przyłączanie węzłów czy routing pakietów są dostępne w rozwiązaniach znacznie bardziej skomplikowanych programowo i sprzętowo. Konfiguracja i zarządzania siecią IQMESH wymaga przyswojenia sobie pewnej porcji wiadomości, ale za to daje możliwość elastycznego połączenia mechanizmów sieciowych z własną aplikacją. Czy to rozwiązanie ma szansę na szersze upowszechnienie się w naszym kraju będzie zależeć od dostępności, jakości wsparcia technicznego i ceny. Według cennika ze strony producenta cena pojedynczych modułów ma wynieść nieco ponad 16 euro za sztukę, a CK-USB-04 – 55 euro. Mając te 2 elementy można oszacować koszty budowy własnych rozwiązań.

Mnie cały system: oprogramowanie IDE, moduły programowe i ewaluacyjne, programy przykładowe bardzo się spodobał. Mam jednak niewielkie zastrzeżenia do dokumentacji – mogłaby być nieco obszerniejsza i wyposażona w przewodnik początkującego, tak aby ktoś, kto pierwszy raz styka się z systemem nie musiał szukać informacji w kilku notach aplikacyjnych i innych dokumentach.

Tomasz Jabłoński, EP