



**KAPITAŁ LUDZKI**  
NARODOWA STRATEGIA SPÓJNOŚCI

**UNIA EUROPEJSKA**  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Instrukcja współfinansowana przez Unię Europejską  
w ramach Europejskiego Funduszu Społecznego  
w projekcie

*„Innowacyjna dydaktyka bez ograniczeń  
– zintegrowany rozwój Politechniki Łódzkiej – zarządzanie Uczelnią,  
nowoczesna oferta edukacyjna i wzmacniania zdolności  
do zatrudniania osób niepełnosprawnych”*

Instrukcja jest dystrybuowana bezpłatnie.

## **Instrukcja do laboratorium**

---

Piotr Wasilewski

# **Laboratorium bezprzewodowych sieci sensorów**

Zadanie nr 14 – Studia podyplomowe „Bezprzewodowe systemy nadzoru i monitorowania”



**Politechnika Łódzka**  
Instytut Elektroniki

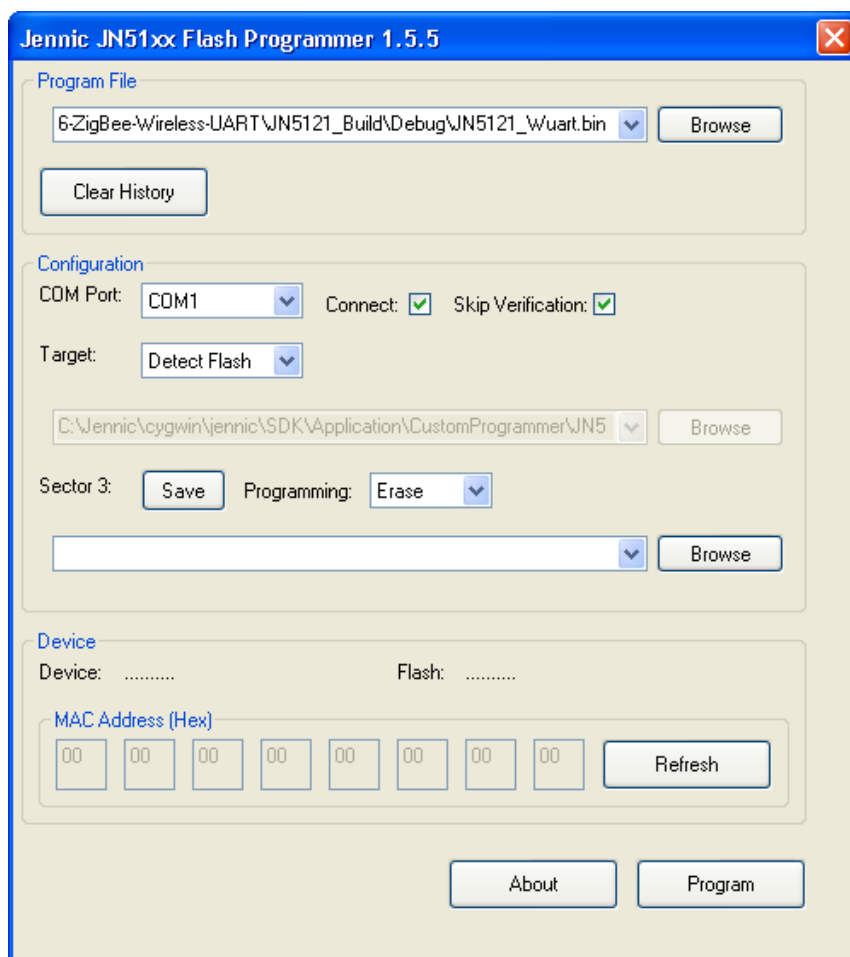
90-924 Łódź, ul. Żeromskiego 116,  
tel. 042 631 28 83  
[www.kapitalludzki.p.lodz.pl](http://www.kapitalludzki.p.lodz.pl)

# 1. Wstęp – programowanie modułów JN51xx za pomocą programu Jennic Flash Programmer

## 1.1. Programowanie modułu

Aby załadować plik z rozszerzeniem **.bin** do modułu należy wykonać następujące kroki:

1. Połączyć komputer z modulem Jennic za pomocą kabla USB. Sposób połączenia kabla z modulem Jennic pokazany jest na poniższym rysunku.
2. Po uruchomieniu programu Jennic Flash Programmer (skrót znajduje się na pulpicie i w menu **Start->Wszystkie programy->Jennic**) widoczne jest okno jak na rysunku poniżej

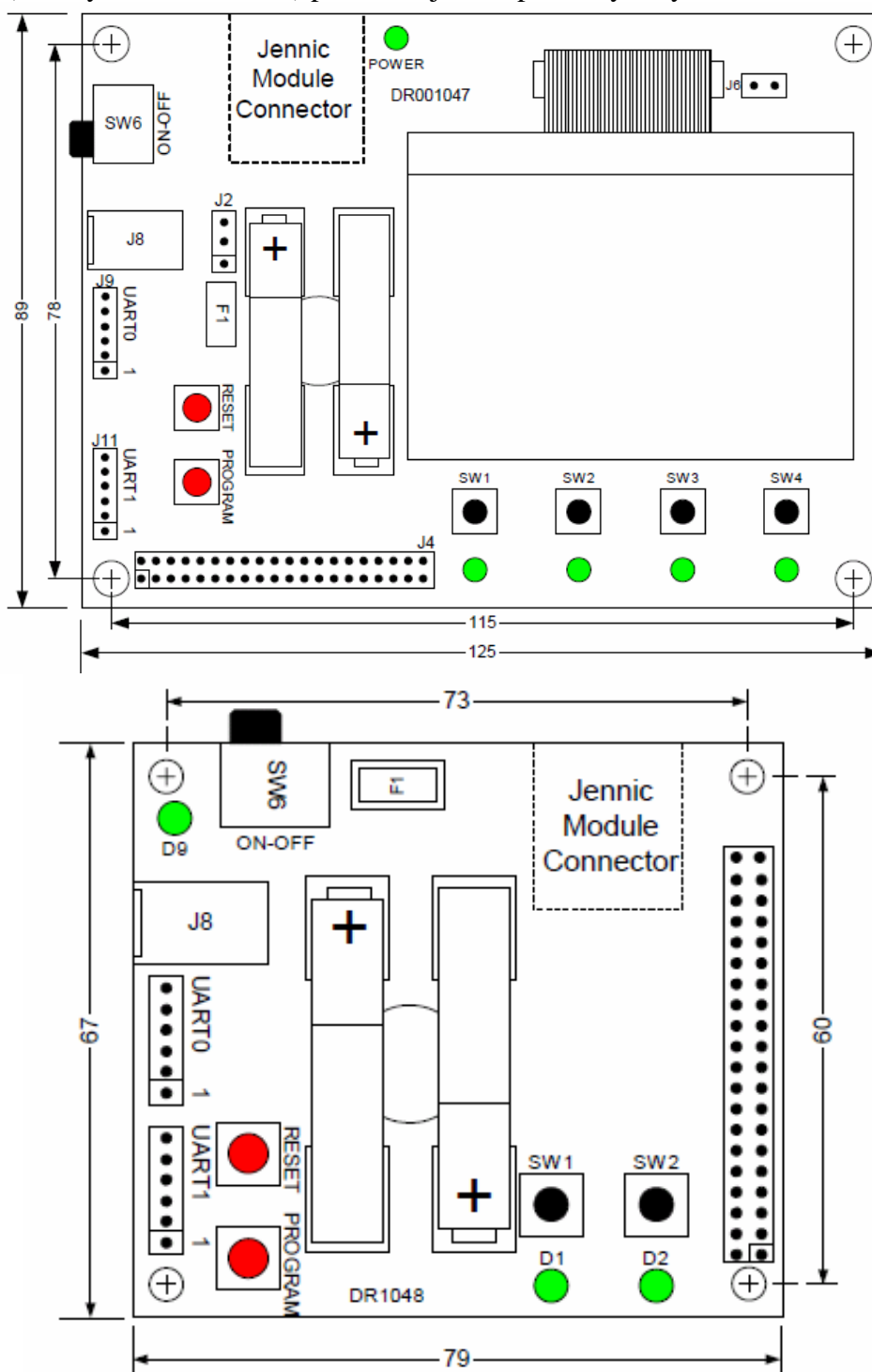


Po uruchomieniu aplikacji należy wybrać port szeregowy służący do komunikacji. Zwykle jest to port COM3.

3. Zresetować moduł i wprowadzić go tryb programowania:
  - a. Weisnąć i przytrzymać przycisk **Program** na płycie.

- b. Nacisnąć i zwolnić przycisk **Reset** na płycie.
- c. Puścić przycisk **Program**.

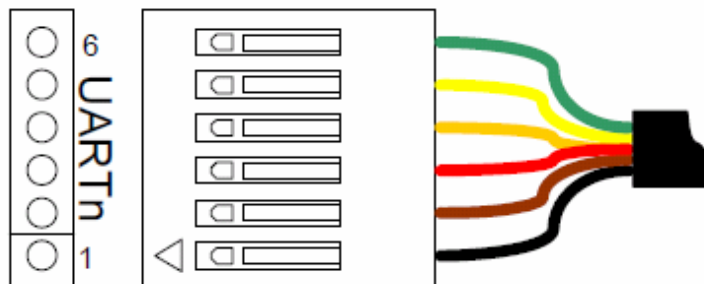
Umieszczenie elementów sterujących na płycie kontrolera (z wyświetlaczem LCD) i płycie sensorów (bez wyświetlacza LCD) pokazane jest na poniższych rysunkach.



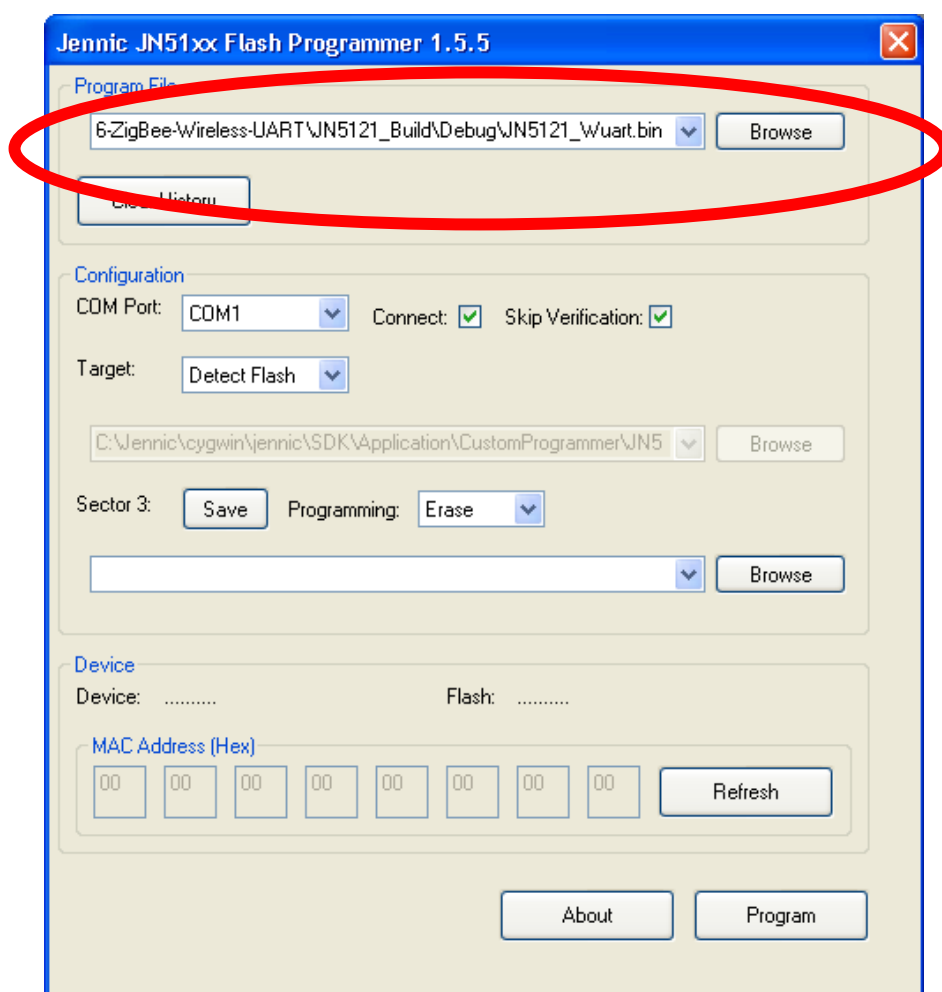
Przełącznik SW6 służy do włączania i wyłączania modułu. Gniazdo J8 służy do podłączenia zewnętrznego zasilacza prądu stałego (napięcie od 5–7V, polaryzacja wyprowadzeń nie jest

istotna) lub zmiennego o napięciu od 4,5–6V, W przypadku wykorzystania zewnętrznego zasilacza zworę na złączu J2 należy ustawić w położeniu 2-3, w przypadku zasilania baterijnego należy ją ustawić w położeniu 1-2.

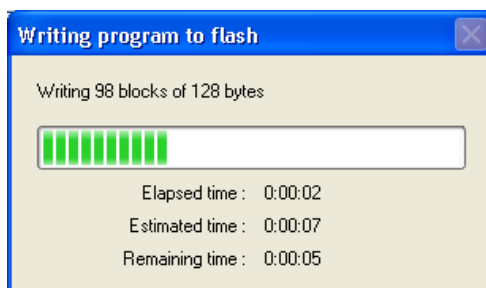
Poniższy rysunek pokazuje sposób połączenia kabla USB do modułu. W celu programowania układu należy podłączyć kabel do portu UART0. UWAGA: Odwrotne podłączenie może uszkodzić zarówno moduł ZigBee jak i elektronikę wewnątrz interfejsu USB!



4. Wybrać plik binarny do załadowania naciskając przycisk **Browse** w aplikacji Jennic Flash Downloader. Aplikacja przechowuje ostatnio użyte pliki, które są widoczne na liście rozwijanej.



5. Po wybraniu pliku nacisnąć przycisk **Program** celu rozpoczęcia programowania. W tym czasie wyświetlane jest okno dialogowe ze wskaźnikiem postępu:

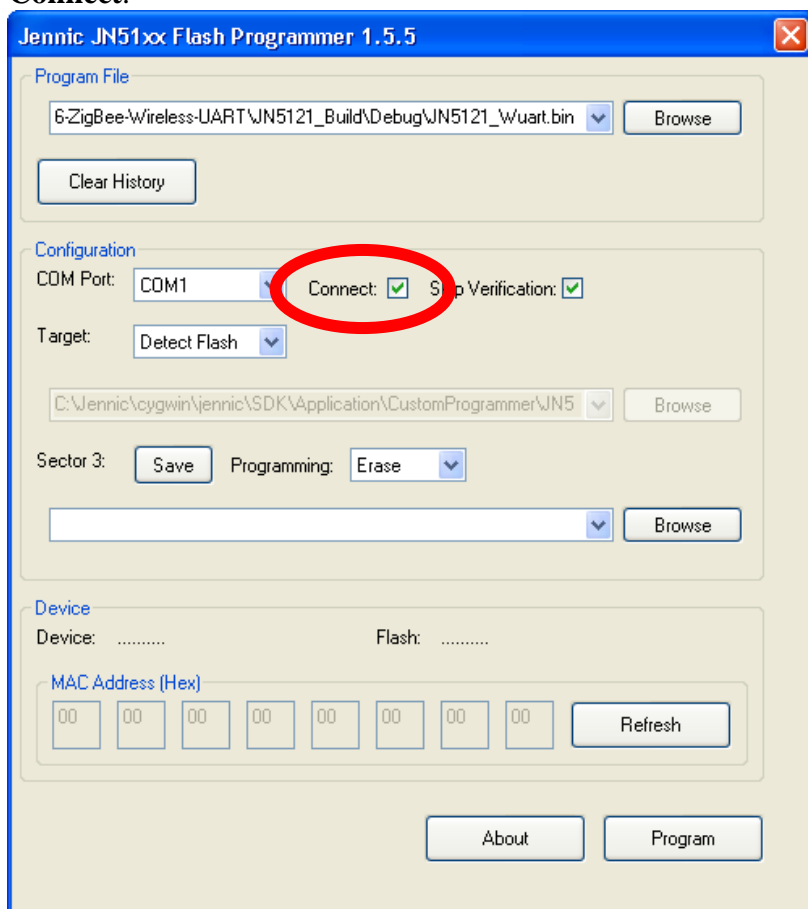


Po zakończeniu programowania zostanie wyświetlony komunikat o sukcesie lub błędzie.

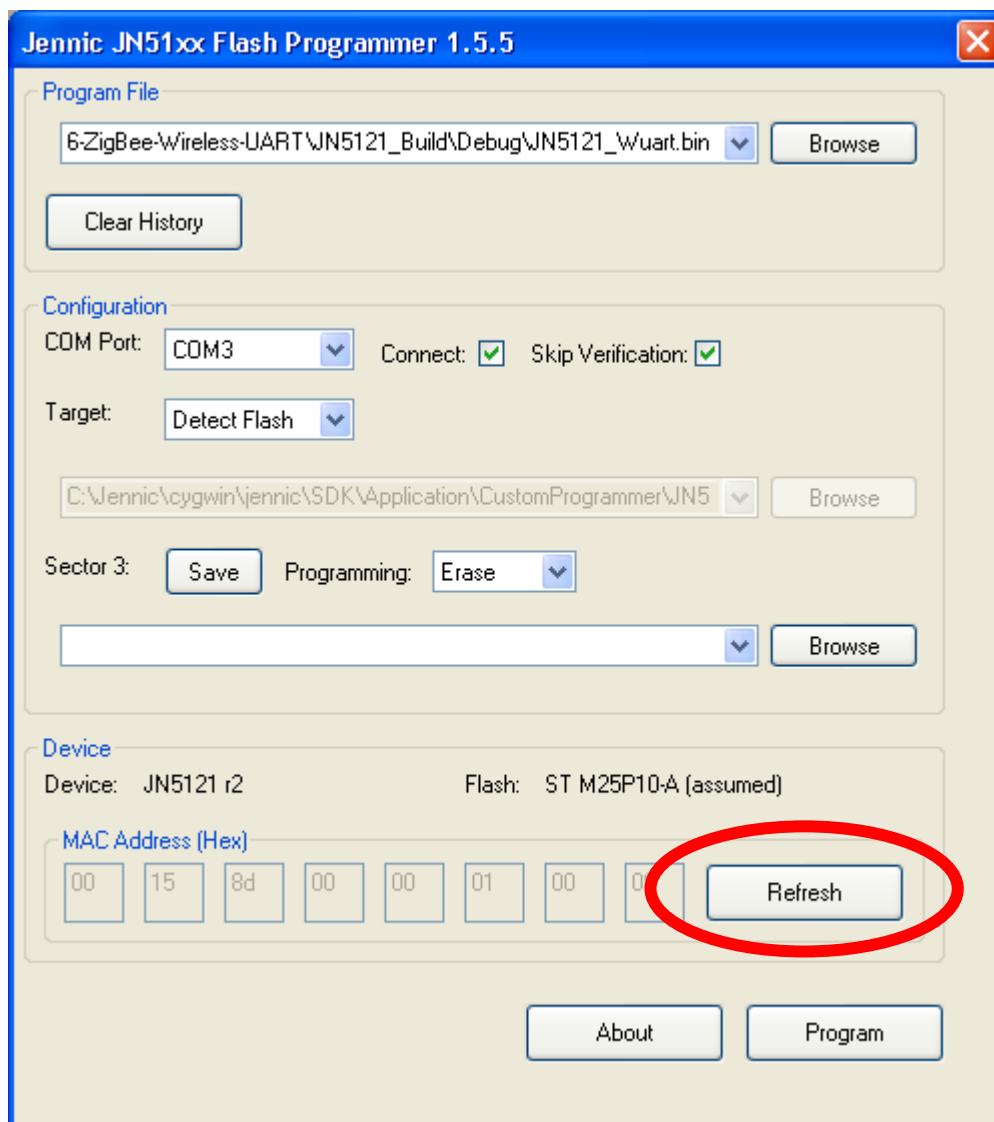
6. Odłączyć kabel lub zwolnić port i zresetować moduł.

## 1.2. Podłączanie i zwalnianie portu szeregowego

Pole **Connect** zaznaczone poniżej służy do podłączania urządzenia zewnętrznego do portu szeregowego. Jeśli program Jennic Flash Programmer ma pozostać uruchomiony i jednocześnie inna aplikacja (np. Hyper Terminal) musi połączyć się z modułem ZigBee, to pole powinno być odznaczone. Aby móc załadować program to modułu należy ponownie zaznaczyć pole **Connect**.



Jeśli programator nie będzie mógł wykryć urządzenia na wskazanym porcie szeregowym wyświetli ostrzeżenie i poprosi o wprowadzenie układu w tryb programowania. Po tym można sprawdzić, czy układ jest widoczny naciskając przycisk **Refresh**, tak jak pokazano na poniższym rysunku. Po wykryciu urządzenia zostaną wyświetlone informacje o numerze MAC urządzenia, typu układu kontrolera i pamięci.



### 1.3. Pomijanie weryfikacji

Domyślnie po zaprogramowaniu Flash programmer porównuje zawartość pamięci z danymi znajdującymi się w pliku **.bit**. Aby przyspieszyć proces programowania można pominąć proces weryfikacji zaznaczając pole **Skip Verification**.



**Jennic JN51xx Flash Programmer 1.5.5**

**Program File**

6-ZigBee-Wireless-UART\JN5121\_Build\Debug\JN5121\_wuart.bin

**Configuration**

COM Port: COM3  Connect: ☒ Skip Verification: ☒

Target: Detect Flash

C:\Jennic\cygwin\jennic\SDK\Application\CustomProgrammer\JN5

Sector 3:  Programming: Erase

**Device**

Device: JN5121 r2 Flash: ST M25P10-A (assumed)

**MAC Address (Hex)**

00 15 8d 00 00 01 00 08

## 2. Badanie jakości transmisji w obecności zakłóceń

Wymagania:

- Moduł bez wyświetlacza LCD z układem dużej mocy. Moduł ten będzie pełnił rolę elementu generującego zakłócenia. Należy go zaprogramować plikiem **JMLTU.bin**.
- Moduł z wyświetlaczem LCD z układem standardowej mocy i anteną dołączoną do złącza SMA. Moduł ten będzie służył do analizy poziomu zakłóceń i należy go zaprogramować plikiem **Surv.bin**.
- Moduł z wyświetlaczem LCD z układem standardowej mocy i anteną dołączoną do złącza SMA. Moduł ten będzie wyświetlał informacje o jakości transmisji i należy go zaprogramować plikiem **PERMaster.bin**.
- Moduł bez wyświetlacza LCD z układem z anteną ceramiczną. Moduł ten będzie źródłem danych do wyznaczenia jakości transmisji i należy go zaprogramować plikiem **PERSlave.bin**.

Po zaprogramowaniu modułu **JMLU** należy zamknąć program **Jennic Flash Programmer** lub zamknąć połączenie z portem. Następnie należy uruchomić program **Hyper Terminal** z następującymi parametrami portu szeregowego: szybkość transmisji 38400 b/s, 8 bitów danych, brak bitu parzystości, brak kontroli przepływu. Po zresetowaniu modułu **JMLU** na konsoli terminal pojawi się następujący tekst.

```
*****
*   Jennic Module Lab Test Utility V2.16   *
*****
```

- a) Standard Module
- b) Standard Module (Boost Mode - JN513x Only)
- c) High Power Module
- d) Telec Low Power Module
- e) Telec High Power Module

Please choose an option >

Należy nacisnąć klawisz “c” na klawiaturze. Pojawi się kolejne menu:

```
*****
*   Select CPU Clock Frequency             *
*****
```

- a) 16MHz
- b) 32MHz JN513x Only)

Please choose an option >





Należy nacisnąć klawisz “a” na klawiaturze. Pojawi się kolejne menu:

```
*****
*      Jennic Module Lab Test Utility      *
*****
```

- a) TX Power Test (CW)
- b) TX Power Test (Modulated)
- c) Receive Test
- d) Packet Error Rate Test
- e) Oscillator Frequency Test
- f) Current Measurement Test
- g) RF Power Measurement
- h) Trigger Packet Test
- i) Receive Packets Test
- j) Transmit Packets Test
- k) Connectionless Packet Error Rate Test

Please choose an option >

Należy nacisnąć klawisz “b” na klawiaturze. Pojawi się kolejne menu:

```
*****
*                      Tx Power Test      *
*****
```

- a) Output Continuous
- b) Output Continuous with Fine Power Adjust Mode
- c) 50% Duty Cycle
- d) 30% Duty Cycle
- e) 1% Duty Cycle
- f) Variable Freq/Duty Cycle
- g) Variable Freq/Duty Cycle with Fine Power Adjust Mode
- x) Return to main menu

Please choose an option >

Należy nacisnąć klawisz “a” na klawiaturze. Pojawi się następujący tekst:

```
*****
*      Power Test In Progress      *
*****
* Key          Function            *
*              *                  *
* +            Increment Channel   *
* -            Decrement Channel   *
* <            Reduce output power *
* >            Increase output power *
* x            Return to main menu *
*              *                  *
*****
```

```
Channel      18      ( 2.440 GHz )
Power Level   5
```

Ustawić kanał 20 za pomocą klawiszy “+” oraz “-”. Wykonać pomiary dla poziomów mocy „5”, „4”, „3”, „2”, „1” oraz „0”.





Włączyć moduł **Surv**. Z modułu **Surv** odczytać poziom zakłóceń w kanale 18. Włączyć moduł **PERMaster** i ustawić kanał 18. Włączyć moduł **PERSlave**. Z modułu **PERMaster** odczytać stopę błędów PER. Wyłączyć moduły **PERSlave** i **PERMaster**. Powtórzyć pomiary dla kanału 19, 20, 21 oraz 22.

Nacisnąć klawisz „x” na klawiaturze. Z menu wybrać „b”. Pojawi się menu:

```
*****
*                               Tx Power Test                               *
*****
```

- a) Output Continuous
- b) Output Continuous with Fine Power Adjust Mode
- c) 50% Duty Cycle
- d) 30% Duty Cycle
- e) 1% Duty Cycle
- f) Variable Freq/Duty Cycle
- g) Variable Freq/Duty Cycle with Fine Power Adjust Mode
- x) Return to main menu

Please choose an option >

Należy nacisnąć klawisz “c” na klawiaturze. Pojawi się następujący tekst:

```
*****
*           Power Test In Progress           *
*****
* Key           Function                     *
*               *                           *
* +             Increment Channel            *
* -             Decrement Channel            *
* <             Reduce output power          *
* >             Increase output power        *
* x             Return to main menu          *
*               *                           *
*****
```

```
Channel      18      (2.440 GHz)
Power Level   5
```

Ustawić kanał 20 za pomocą klawiszy “+” oraz “-”. Wykonać pomiary dla poziomów mocy „5”, „4”, „3”, „2”, „1” oraz „0”.

Włączyć moduł **Surv**. Z modułu **Surv** odczytać poziom zakłóceń w kanale 18. Włączyć moduł **PERMaster** i ustawić kanał 18. Włączyć moduł **PERSlave**. Z modułu **PERMaster** odczytać stopę błędów PER. Wyłączyć moduły **PERSlave** i **PERMaster**. Powtórzyć pomiary dla kanału 19, 20, 21 oraz 22.

Nacisnąć klawisz „x” na klawiaturze. Z menu wybrać „b”. Pojawi się menu:





```
*****
*                               Tx Power Test                               *
*****
```

- a) Output Continuous
- b) Output Continuous with Fine Power Adjust Mode
- c) 50% Duty Cycle
- d) 30% Duty Cycle
- e) 1% Duty Cycle
- f) Variable Freq/Duty Cycle
- g) Variable Freq/Duty Cycle with Fine Power Adjust Mode
- x) Return to main menu

Please choose an option >

Należy nacisnąć klawisz “d” na klawiaturze. Pojawi się następujący tekst:

```
*****
*                               Power Test In Progress                               *
*****
* Key          Function          *
*                               *
* +            Increment Channel  *
* -            Decrement Channel  *
* <            Reduce output power *
* >            Increase output power *
* x            Return to main menu *
*                               *
*****
```

```
Channel      18      ( 2.440 GHz )
Power Level   5
```

Ustawić kanał 20 za pomocą klawiszy “+” oraz “-”. Wykonać pomiary dla poziomów mocy „5”, „4”, „3”, „2”, „1” oraz „0”.

Włączyć moduł **Surv**. Z modułu **Surv** odczytać poziom zakłóceń w kanale 18. Włączyć moduł **PERMaster** i ustawić kanał 18. Włączyć moduł **PERSlave**. Z modułu **PERMaster** odczytać stopę błędów PER. Wyłączyć moduły **PERSlave** i **PERMaster**. Powtórzyć pomiary dla kanału 19, 20, 21 oraz 22.

Nacisnąć klawisz „x” na klawiaturze. Z menu wybrać „b”. Pojawi się menu:

```
*****
*                               Tx Power Test                               *
*****
```

- a) Output Continuous
- b) Output Continuous with Fine Power Adjust Mode
- c) 50% Duty Cycle
- d) 30% Duty Cycle
- e) 1% Duty Cycle
- f) Variable Freq/Duty Cycle
- g) Variable Freq/Duty Cycle with Fine Power Adjust Mode
- x) Return to main menu

Please choose an option >

Należy nacisnąć klawisz “e” na klawiaturze. Pojawi się następujący tekst:





```
*****
*      Power Test In Progress      *
*****
* Key          Function            *
*      *      *                    *
* +      Increment Channel         *
* -      Decrement Channel         *
* <      Reduce output power      *
* >      Increase output power     *
* x      Return to main menu      *
*      *      *                    *
*****
```

```
Channel      18      ( 2.440 GHz )
Power Level   5
```

Ustawić kanał 20 za pomocą klawiszy “+” oraz “-”. Wykonać pomiary dla poziomów mocy „5”, „4”, „3”, „2”, „1” oraz „0”.

Włączyć moduł **Surv**. Z modułu **Surv** odczytać poziom zakłóceń w kanale 18. Włączyć moduł **PERMaster** i ustawić kanał 18. Włączyć moduł **PERSlave**. Z modułu **PERMaster** odczytać stopę błędów PER. Wyłączyć moduły **PERSlave** i **PERMaster**. Powtórzyć pomiary dla kanału 19, 20, 21 oraz 22.

Wyłączyć moduły, zamknąć program Hyper terminal.



### 3. Badanie poboru prądu przez moduł ZigBee

Wymagania:

Moduł bez wyświetlacza LCD z układem dużej mocy. Moduł ten będzie pełnił rolę elementu generującego zakłócenia. Należy go zaprogramować plikiem **JMLTU.bin**. Moduł JMLTU powinien być zasilany z zewnętrznego źródła napięcia z włączonym szeregowo amperomierzem. Przy pomiarach należy uwzględnić prąd pobierany przez diodę LED, około 1,3mA.

Po zaprogramowaniu modułu **JMLU** należy zamknąć program **Jennic Flash Programmer** lub zamknąć połączenie z portem. Następnie należy uruchomić program **Hyper Terminal** z następującymi parametrami portu szeregowego: szybkość transmisji 38400 b/s, 8 bitów danych, brak bitu parzystości, brak kontroli przepływu. Po zresetowaniu modułu **JMLU** na konsoli terminal pojawi się następujący tekst.

```
*****
*      Jennic Module Lab Test Utility V2.16      *
*****
```

- a) Standard Module
- b) Standard Module (Boost Mode - JN513x Only)
- c) High Power Module
- d) Telec Low Power Module
- e) Telec High Power Module

Please choose an option >

Należy nacisnąć klawisz “c” na klawiaturze. Pojawi się kolejne menu:

```
*****
*      Select CPU Clock Frequency      *
*****
```

- a) 16MHz
- b) 32MHz JN513x Only)

Please choose an option >

Należy nacisnąć klawisz “a” na klawiaturze. Pojawi się kolejne menu:



```
*****  
*      Jennic Module Lab Test Utility      *  
*****
```

- a) TX Power Test (CW)
- b) TX Power Test (Modulated)
- c) Receive Test
- d) Packet Error Rate Test
- e) Oscillator Frequency Test
- f) Current Measurement Test
- g) RF Power Measurement
- h) Trigger Packet Test
- i) Receive Packets Test
- j) Transmit Packets Test
- k) Connectionless Packet Error Rate Test

Please choose an option >

Należy nacisnąć klawisz “f” na klawiaturze. Pojawi się kolejne menu:

```
*****  
*      Current Measurement Options        *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >

Nacisnąć klawisz „a” w celu wprowadzenia kontrolera w tryb głębokiego uśpienia (deep sleep mode). Jest to tryb maksymalnej oszczędności energii i można z niego wyjść jedynie poprzez reset układu.





```
*****
*      Current Measurement Options      *
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Putting device into deep sleep mode.  
Note, you must reset the device to  
exit this state

Po pomiarze prądu zresetować układ i ponownie wybrać polecenia menu naciskając klawisze „c”, „a”, „f”. następnie nacisnąć klawisz „b” i wprowadzić układ tryb uśpienia bez podtrzymania zawartości pamięci. W tym trybie praca procesora jest zatrzymana, działa jednak wewnętrzny licznik, który może wybudzić układ. Stan wyjść układu również nie ulega zmianom.

```
*****
*      Current Measurement Options      *
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Putting device into sleep mode without memory retention for 60 seconds..

Po pomiarze prądu zresetować układ i ponownie wybrać polecenia menu naciskając klawisze „c”, „a”, „f”. następnie nacisnąć klawisz „c” i wprowadzić układ tryb uśpienia z podtrzymaniem zawartości pamięci. Podobnie, jak w poprzednim trybie praca procesora jest zatrzymana, działa jednak wewnętrzny licznik, który może wybudzić układ. Stan wyjść układu również nie ulega zmianom.





```
*****  
*      Current Measurement Options      *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Putting device into sleep mode with memory retention for 60 seconds..

Po upływie jednej minuty kontroler wyjdzie z trybu uśpienia na klawiaturze nacisnąć klawisze „c”, „a”, „f”. następnie nacisnąć klawisz „d” i wprowadzić procesor w tryb uśpienia (doze). Pozostałe elementy kontrolera są zasilone i mogą działać. Tryb ten jest często używany w sytuacjach, kiedy układ nie musi wykonywać żadnych czynności, gdyż czas wyjścia z trybu „doze” jest znacznie krótszy niż z trybu „sleep”. W czasie pomiaru nadajnik i odbiornik pozostają wyłączone.

```
*****  
*      Current Measurement Options      *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Putting device into doze mode for 60 seconds..

Dokonać pomiaru prądu. Po upływie jednej minuty kontroler wyjdzie z trybu uśpienia na klawiaturze nacisnąć klawisze „c”, „a”, „f”. następnie nacisnąć klawisz „e”. W tym trybie procesor działa normalnie, wyłączony jest nadajnik i odbiornik.







```
*****  
*      Current Measurement Options      *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Micro running, press any key to exit..

Dokonać pomiaru prądu. Po naciśnięciu dowolnego klawisza program wraca do menu pomiaru prądu. Naciśnąć klawisz „f”. Ten tryb różni się od poprzedniego włączeniem sprzętowych akceleratorów wykorzystywanych przez protokół komunikacyjny.

```
*****  
*      Current Measurement Options      *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Micro running with protocol, press any key to exit..

Dokonać pomiaru prądu. Zresetować układ w celu przywrócenia go do standardowych ustawień i nacisnąć klawisze „c”, „a”, „f”. Następnie nacisnąć klawisz „g”. W tym trybie kontroler wysyła dane z maksymalną mocą (+10dBm).





```
*****
*      Current Measurement Options      *
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Micro running with protocol + TX, press any key to exit..

Dokonać pomiaru prądu. Zresetować układ w celu przywrócenia go do standardowych ustawień i nacisnąć klawisze „a” (tryb pracy modułu bez wykorzystania dodatkowego wzmacniacza), „a”, „f”. Następnie nacisnąć klawisz „g”. W tym trybie kontroler wysyła dane z maksymalną mocą (0dBm).

```
*****
*      Current Measurement Options      *
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Micro running with protocol + TX, press any key to exit..

Dokonać pomiaru prądu. Zresetować układ w celu przywrócenia go do standardowych ustawień i nacisnąć klawisze „c”, „a”, „f”. Następnie nacisnąć klawisz „h”. W tym trybie nadajnik jest wyłączony, odbiornik pracuje z maksymalnym wzmocnieniem.





```
*****  
*      Current Measurement Options      *  
*****
```

- a) Deep sleep mode
- b) Sleep mode without memory retention
- c) Sleep mode with memory retention
- d) Doze mode
- e) microcontroller running
- f) microcontroller + protocol
- g) microcontroller + protocol + TX
- h) microcontroller + protocol + RX
- x) Return to main menu

Please choose an option >  
Micro running with protocol + RX, press any key to exit..

Dokonać pomiaru prądu, wyłączyć układ, zamknąć program Hyper Terminal.





## 4. Badanie szybkości transmisji

Wymagania: 4 moduły małej mocy z antenami ceramicznymi lub antenami dookólnymi. Jeden moduł zaprogramować plikiem **KoordMSG\_xx.bin**, gdzie xx oznacza numer kanału. Pozostałe moduły zaprogramować plikiem **RouterMSG\_xx.bin**.

Po zaprogramowaniu modułu **KoordMSG** należy zamknąć program **Jennic Flash Programmer** lub zamknąć połączenie z portem, natomiast należy pozostawić połączenie modułu z komputerem za pomocą kabla USB. Następnie należy uruchomić program **Hyper Terminal** z następującymi parametrami portu szeregowego: szybkość transmisji 38400 b/s, 8 bitów danych, brak bitu parzystości, brak kontroli przepływu. Po zresetowaniu modułu **KoordMSG** na konsoli terminal pojawi się następujący tekst.

```
*****  
*      Rodzaj transmisji      *  
*****
```

- a) Transmisja ciągła bez potwierdzeń (ramki MSG)
- b) Transmisja z przerwami (10) bez potwierdzeń (ramki MSG)
- c) Transmisja z przerwami (20) bez potwierdzeń (ramki MSG)
- d) Transmisja z przerwami (40) bez potwierdzeń (ramki MSG)
- e) Transmisja z przerwami (60) bez potwierdzeń (ramki MSG)
- f) Transmisja z przerwami (80) bez potwierdzeń (ramki MSG)
- g) Transmisja z przerwami (100) bez potwierdzeń (ramki MSG)
- h) Transmisja z potwierdzeniami (ramki MSG)
- i) Transmisja kontrolowana przez koordynatora (ramki MSG)

Wybierz opcje >  
Nr kanału 21, PAN ID 96

Opcja a) oznacza, że routery będą wysyłać ramki z maksymalną szybkością procesora. Opcje b)-g) oznaczają, że routery będą wysyłać ramki w odstępach czasowych określonych w nawiasach. Wartości te należy traktować orientacyjnie. Opcja h) oznacza, że koordynator będzie potwierdzać odbiór każdej ramki, a router wyśle kolejną ramkę dopiero po otrzymaniu potwierdzenia. Opcja i) oznacza, że routery wysyłają ramki na żądanie koordynatora. Koordynator wysyła żądania do kolejnych routerów po wcześniejszym otrzymaniu prawidłowej ramki.

Po dokonaniu wyboru rodzaju transmisji należy wybrać długość wysyłanych ramek:

```
*****  
*      Długość ramek      *  
*****
```

- a) 5 bajt
- b) 10 bajtów
- c) 20 bajtów
- d) 40 bajtów
- e) 60 bajtów
- f) 80 bajtów

Wybierz opcje >





Następnie należy wybrać moc nadajników:

```
*****
*      Moc nadawania      *
*****
```

- a) -30dBm (-7dBm)
- b) -24dBm (-1dBm)
- c) -18dBm ( 5dBm)
- d) -12dBm (11dBm)
- e) - 6dBm (15dBm)
- f) 0dBm (18dBm)

Wybierz opcje >

W następnej kolejności trzeba wybrać liczbę routerów w sieci:

```
*****
*      Liczba routerow    *
*****
```

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >2

W tym momencie należy włączyć wybraną liczbę routerów. Zostaną wyświetlone ich adresy oraz informacja o rozpoczęciu testu. Test trwa 10 sekund.

```
*****
*      Liczba routerow    *
*****
```

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >2

Nowy wezel nr 1 Adres: 1

Nowy wezel nr 2 Adres: 5182

Test rozpoczęty

Po zakończeniu testu dla każdego routera zostanie wyświetlona liczba wysłanych i odebranych ramek oraz średnia szybkość transmisji. W czasie testu powinny zapalić się diody D2 na modułach routerów, a po jego zakończeniu diody powinny zgasnąć. Po zakończeniu testu test można powtórzyć lub rozpocząć nowy test. Przed rozpoczęciem nowego testu należy wyłączyć wszystkie routery.





\*\*\*\*\*

\*      Liczba routerow      \*

\*\*\*\*\*

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >2

Nowy wezel nr 1 Adres: 1

Nowy wezel nr 2 Adres: 5182

Test rozpoczety

Test zakonczony

Wezel 1: wyslanych ramek 63744: odebranych ramek 1436, Szybkosc 5744b/s

Wezel 5182: wyslanych ramek 63897: odebranych ramek 1482, Szybkosc 5928b/s

\*\*\*\*\*

r) powtorz test

n) nowy test

Wybierzopcje >

Przebieg ćwiczenia:

1. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
2. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczony dookoła koordynatora.
3. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 3 routery umieszczony dookoła koordynatora.
4. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 10 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
5. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 10 bajtów, moc -12 dBm, 2 routery umieszczony dookoła koordynatora.
6. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 20 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
7. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 20 bajtów, moc -12 dBm, 2 routery umieszczony dookoła koordynatora.
8. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
9. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczony dookoła koordynatora.
10. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery – jeden umieszczony blisko koordynatora, a drugi w odległości ok. 30cm od koordynatora.
11. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
12. Transmisja ciągła bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczony dookoła koordynatora.
13. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.





14. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
15. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
16. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
17. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
18. Transmisja ciągła z przerwami (10) bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
19. Transmisja ciągła z przerwami (40) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
20. Transmisja ciągła z przerwami (40) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
21. Transmisja ciągła z przerwami (40) bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
22. Transmisja ciągła z przerwami (40) bez potwierdzeń (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
23. Transmisja ciągła z przerwami (100) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
24. Transmisja ciągła z przerwami (100) bez potwierdzeń (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
25. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
26. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora\
27. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
28. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
29. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
30. Transmisja z potwierdzeniami (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
31. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
32. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora\
33. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
34. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
35. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
36. Transmisja kontrolowana przez koordynatora (ramki MSG), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora





Po wykonaniu powyższych ćwiczeń należy zamknąć program HyperTerminal, a następnie jeden moduł zaprogramować plikiem **KoordKVP\_xx.bin**, gdzie xx oznacza numer kanału. Pozostałe moduły zaprogramować plikiem **RouterKVP\_xx.bin**.

Po zaprogramowaniu modułu **KoordKVP** należy zamknąć program **Jennic Flash Programmer** lub zamknąć połączenie z portem, natomiast należy pozostawić połączenie modułu z komputerem za pomocą kabla USB. Następnie należy uruchomić program **Hyper Terminal** z następującymi parametrami portu szeregowego: szybkość transmisji 38400 b/s, 8 bitów danych, brak bitu parzystości, brak kontroli przepływu. Po zresetowaniu modułu **KoordKVP** na konsoli terminal pojawi się następujący tekst.

```
*****
*      Rodzaj transmisji      *
*****
```

- a) Transmisja ciągła bez potwierdzeń (ramki KVP)
- b) Transmisja ciągła z potwierdzeniami (ramki KVP)
- c) Transmisja kontrolowana przez koordynatora (ramki KVP)

Wybierz opcje >

Nr kanału 12, PAN ID 12

Opcja a) oznacza, że routery będą wysyłać ramki z maksymalną szybkością procesora. Opcja b) oznacza, że koordynator będzie potwierdzać odbiór każdej ramki, a router wyśle kolejną ramkę dopiero po otrzymaniu potwierdzenia. Opcja c) oznacza, że routery wysyłają ramki na żądanie koordynatora. Koordynator wysyła żądania do kolejnych routerów po wcześniejszym otrzymaniu prawidłowej ramki.

Po dokonaniu wyboru rodzaju transmisji należy wybrać długość wysyłanych ramek:

```
*****
*      Długość ramek      *
*****
```

- a) 5 bajt
- b) 10 bajtów
- c) 20 bajtów
- d) 40 bajtów
- e) 60 bajtów
- f) 80 bajtów

Wybierz opcje >

Następnie należy wybrać moc nadajników:

```
*****
*      Moc nadawania      *
*****
```

- a) -30dBm (-7dBm)
- b) -24dBm (-1dBm)
- c) -18dBm (5dBm)
- d) -12dBm (11dBm)
- e) -6dBm (15dBm)
- f) 0dBm (18dBm)

Wybierz opcje >







W następnej kolejności trzeba wybrać liczbę routerów w sieci:

```
*****
*      Liczba routerow      *
*****
```

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >

W tym momencie należy włączyć wybraną liczbę routerów. Zostaną wyświetlone ich adresy oraz informacja o rozpoczęciu testu. Test trwa 10 sekund.

```
*****
*      Liczba routerow      *
*****
```

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >1

Nowy wezel nr 1 Adres: 1

Test rozpoczęty

Po zakończeniu testu dla każdego routera zostanie wyświetlona liczba wysłanych i odebranych ramek oraz średnia szybkość transmisji. W czasie testu powinny zapalić się diody D2 na modułach routerów, a po jego zakończeniu diody powinny zgasnąć.

Po zakończeniu testu test można powtórzyć lub rozpocząć nowy test. Przed rozpoczęciem nowego testu należy wyłączyć wszystkie routery.

```
*****
*      Liczba routerow      *
*****
```

- 1) 1 router
- 2) 2 routery
- 3) 3 routery

Wybierz opcje >1

Nowy wezel nr 1 Adres: 1

Test rozpoczęty

Test zakończony

Wezel 1: wyslanych ramek 55169: odebranych ramek 2801, Szybkosc 11204b/s

```
*****
```

- r) powtorz test
- n) nowy test

Wybierz opcje >





### Przebieg ćwiczenia:

1. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
2. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
3. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 3 routery umieszczone dookoła koordynatora.
4. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 20 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
5. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 20 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
6. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
7. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
8. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 2 routery – jeden umieszczony blisko koordynatora, a drugi w odległości ok. 30cm od koordynatora.
9. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora.
10. Transmisja ciągła bez potwierdzeń (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora.
11. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
12. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora\
13. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
14. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
15. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
16. Transmisja z potwierdzeniami (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
17. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
18. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 5 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora\
19. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
20. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 40 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora
21. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 1 router umieszczony blisko koordynatora
22. Transmisja kontrolowana przez koordynatora (ramki KVP), ramki o długości 80 bajtów, moc -12 dBm, 2 routery umieszczone dookoła koordynatora





## 5. Badanie zasięgów

Wymagania:

- Moduł z wyświetlaczem LCD z układem standardowej mocy i anteną dołączoną do złącza SMA. Moduł ten będzie wyświetlał informacje o jakości transmisji i należy go zaprogramować plikiem **PERMaster.bin**.
- Moduł bez wyświetlacza LCD z układem standardowej mocy i anteną dołączoną do złącza SMA. Moduł ten będzie źródłem danych do wyznaczenia jakości transmisji i należy go zaprogramować plikiem **PERSlave.bin**.

Na początku należy wykonać pomiary z układami wyposażonymi w anteny ceramiczne, a następnie powtórzyć je z układami posiadającymi anteny dookólne. Pomiary zasięgu należy wykonać przy następujących poziomach mocy:

- a) 47 (0dBm)
- b) 37 (-20dBm)
- c) 27 (-40dBm)
- d) 17 (-60dBm)
- e) 7 (-80dBm)

Należy wybrać tryb pracy z potwierdzeniami (ACK). Oszacować zasięg na podstawie wskazań pakietowej stopy błędu (PER):

- a)  $PER > 0$ , transmisja możliwa
- b) Brak transmisji



## 6. Najprostszy program

### 6.1. Program koordynatora

```

/*****
***      Include files      ***
*****/
#include <jendefs.h>
#include <LedControl.h>
#include <AppHardwareApi.h>
#include <Utilities.h>
#include <JZ_Api.h>
#include <gdb.h>
#include "..\..\..\Chip\Common\Include\Printf.h"

#include "WSN_Profile.h"

/*****
***      Macro Definitions      ***
*****/
/* Timing values */
#define APP_TICK_PERIOD_ms          500

/*****
***      Type Definitions      ***
*****/

/*****
***      Local Function Prototypes      ***
*****/
PRIVATE void vInit(void);
PRIVATE void vToggleLed(void *pvMsg, uint8 u8Dummy);
PRIVATE void vTxSerialDataFrame(uint16 ul6NodeId,
                                uint16 ul6Humidity,
                                uint16 ul6Temperature,
                                uint16 ul6BattVoltage);

/*****
***      Exported Variables      ***
*****/

/*****
***      Local Variables      ***
*****/
PRIVATE bool_t bNwkStarted = FALSE;
PRIVATE bool_t bAppTimerStarted = FALSE;

/*****
*
* NAME: AppColdStart
*
* DESCRIPTION:
* Entry point for application from boot loader. Initialises system and runs
* main loop.
*****/
```



```
*
* RETURNS:
* Never returns.
*
*****/
PUBLIC void AppColdStart(void)
{
    /* Debug hooks: include these regardless of whether debugging or not */
    HAL_GDB_INIT();
    HAL_BREAKPOINT();

    /* Set network information */
    JZS_sConfig.u32Channel = WSN_CHANNEL;
    JZS_sConfig.ul6PanId   = WSN_PAN_ID;

    /* General initialisation */
    vInit();

    /* No return from the above function call */
}

/*****
*
* NAME: AppWarmStart
*
* DESCRIPTION:
* Entry point for application from boot loader. Simply jumps to AppColdStart
* as, in this instance, application will never warm start.
*
* RETURNS:
* Never returns.
*
*****/
PUBLIC void AppWarmStart(void)
{
    AppColdStart();
}

/*****
***      Local Functions      ***
/*****
/*****
*
* NAME: vInit
*
* DESCRIPTION:
* Initialises Zigbee stack and hardware. Final action is to start BOS, from
* which there is no return. Subsequent application actions occur in the
* functions defined above.
*
* RETURNS:
* No return from this function
*
*****/
PRIVATE void vInit(void)
{
    /* Initialise Zigbee stack */
    JZS_u32InitSystem(TRUE);

    /* Set DIO for LEDs */
    vLedInitFfd();

    vLedControl(0,0);
```





```
vLedControl(1,0);
vLedControl(2,0);
vLedControl(3,0);

/* Intialise serial comms unless debug mode*/
#ifdef GDB
    vUART_printInit();

#endif

/* Start BOS */
(void)bBosRun(TRUE);

/* No return from the above function call */
}

/*****
 *
 * NAME: vTxSerialDataFrame
 *
 * DESCRIPTION:
 * Transmits node data (address and sensor readings) to host via serial port.
 *
 * PARAMETERS: Name          RW  Usage
 *              ul6NodeId     R   Short address of node that generated the data
 *              ul6Humidity    R   Reading from humidity sensor (%)
 *              ul6Temperature R   Reading from temperature sensor (degrees C)
 *              ul6BattVoltage R   ADC reading of supply voltage (mv)
 *
 *****/
PRIVATE void vTxSerialDataFrame(uint16 ul6NodeId,
                                uint16 ul6Humidity,
                                uint16 ul6Temperature,
                                uint16 ul6BattVoltage)
{
    #ifndef GDB
        vPrintf("\n\r\n\rAddress = %x", ul6NodeId);
        vPrintf("\n\r\n\rHumidity = %d", ul6Humidity);
        vPrintf("\n\r\n\rTemperature = %d", ul6Temperature);
        vPrintf("\n\r\n\rVoltage = %d", ul6BattVoltage);
    #endif
}

/*****
 *
 * NAME: vToggleLed
 *
 * DESCRIPTION:
 * Gets called by a BOS timer. Toggles LED1 to indicate we are alive.
 *
 *****/
PRIVATE void vToggleLed(void *pvMsg, uint8 u8Dummy)
{
    uint8 u8Msg;
    uint8 u8TimerId;
    static bool_t bToggle;

    if (bToggle)
    {
        vLedControl(0,0);
    }
    else
    {
        vLedControl(0,1);
    }
}
```





```
    }
    bToggle = !bToggle;

    (void)bBosCreateTimer(vToggleLed, &u8Msg, 0, (APP_TICK_PERIOD_ms / 10),
&u8TimerId);
}

/*****
***          Functions called by the stack          ***
*****/

/*****
*
* NAME: JZA_vAppEventHandler
*
* DESCRIPTION:
* Called regularly by the task scheduler. This function reads the hardware
* event queue and processes the events therein. It is important that this
* function exits after a relatively short time so that the other tasks are
* not adversely affected.
*
*****/
void JZA_vAppEventHandler(void)
{
    uint8 u8Msg;
    uint8 u8TimerId;

    if (!bAppTimerStarted)
    {
        if (bNwkStarted)
        {
            bAppTimerStarted = TRUE;
            (void)bBosCreateTimer(vToggleLed, &u8Msg, 0, (APP_TICK_PERIOD_ms / 10),
&u8TimerId);
        }
    }
}

/*****
*
* NAME: JZA_vPeripheralEvent
*
* DESCRIPTION:
* Called when a hardware event causes an interrupt. This function is called
* from within the interrupt context so should be brief. In this case, the
* information is placed on a simple FIFO queue to be processed later.
*
* PARAMETERS: Name      RW  Usage
*              u32Device  R   Peripheral generating interrupt
*              u32ItemBitmap R   Bitmap of interrupt sources within peripheral
*
*****/
PUBLIC void JZA_vPeripheralEvent(uint32 u32Device, uint32 u32ItemBitmap)
{
}

/*****
*
* NAME: JZA_vAppDefineTasks
*
* DESCRIPTION:
* Called by Zigbee stack during initialisation to allow the application to
* initialise any tasks that it requires. This application requires none.
*
*****/
```





```
* RETURNS:
* void
*
***** /
PUBLIC void JZA_vAppDefineTasks(void)
{
}

/*****
*
* NAME: JZA_boAppStart
*
* DESCRIPTION:
* Called by Zigbee stack during initialisation. Sets up the profile
* information and starts the networking activity
*
* RETURNS:
* TRUE
*
***** /
PUBLIC bool_t JZA_boAppStart(void)
{
    JZS_vStartStack();

    return TRUE;
}

/*****
*
* NAME: JZA_eAfKvpObject
*
* DESCRIPTION:
* Called when a KVP transaction has been received with a matching endpoint.
*
* PARAMETERS:
*
*      Name          RW  Usage
*      afSrcAddr      R   Address of sender device
*      u8DstEndpoint  R   Endpoint at receiver
*      pu8ClusterId   R   Pointer to cluster ID
*      eCommandTypeId R   KVP command type
*      u16AttributeId R   KVP attribute ID
*      pu8AfdLength   R   Pointer to length of data
*      pu8Afd         R   Data array
*
* RETURNS:
* AF_ERROR_CODE
*
***** /
PUBLIC bool_t JZA_bAfKvpObject(APS_Addrmode_e eAddrMode,
                               uint16 u16AddrSrc,
                               uint8 u8SrcEP,
                               uint8 u8LQI,
                               uint8 u8DstEP,
                               uint8 u8ClusterId,
                               uint8 *pu8ClusterIDRsp,
                               AF_Transaction_s *puTransactionInd,
                               AF_Transaction_s *puTransactionRsp)
{
    return KVP_SUCCESS;
}

/*****
*
* NAME: JZA_vAfKvpResponse
```







```
*
* DESCRIPTION:
* Called after a KVP transaction with acknowledgement request, when the
* acknowledgement arrives. In this application no action is taken as no
* KVP transaction acknowledgements are expected.
*
* PARAMETERS:      Name                      RW  Usage
*                   srcAddressMod            R   Address of sender device
*                   transactionSequenceNum    R   KVP transaction number
*                   commandTypeIdentifier    R   KVP command type
*                   dstEndPoint              R   Endpoint at receiver
*                   clusterID               R   Cluster ID
*                   attributeIdentifier       R   KVP attribute ID
*                   errorCode                R   Result code
*                   afduLength              R   Length of payload data
*                   pAfd�                  R   Payload data array
*
*****/
PUBLIC void JZA_vAfKvpResponse(APS_Addrmode_e eAddrMode,
                              uint16 u16AddrSrc,
                              uint8 u8SrcEP,
                              uint8 u8LQI,
                              uint8 u8DstEP,
                              uint8 u8ClusterID,
                              AF_Transaction_s *puTransactionInd)
{
}

/*****
*
* NAME: JZA_pu8AfMsgObject
*
* DESCRIPTION:
* Called when a MSG transaction has been received with a matching endpoint.
*
* PARAMETERS:      Name                      RW  Usage
*                   afSrcAddr                R   Address of sender device
*                   dstEndPoint              R   Endpoint at receiver
*                   clusterID               R   Pointer to cluster ID
*                   afduLength              R   Pointer to length of data
*                   pAfd�                  R   Data array
*
* RETURNS:
* NULL
*
*****/
PUBLIC bool_t JZA_bAfMsgObject(APS_Addrmode_e eAddrMode,
                              uint16 u16AddrSrc,
                              uint8 u8SrcEP,
                              uint8 u8LQI,
                              uint8 u8DstEP,
                              uint8 u8ClusterID,
                              uint8 *pu8ClusterIDRsp,
                              AF_Transaction_s *puTransactionInd,
                              AF_Transaction_s *puTransactionRsp)
{
    uint16 u16Humidity;
    uint16 u16BattVoltage;
    uint16 u16Temperature;

    if ((eAddrMode == APS_ADDRMODE_SHORT) && (u8DstEP == WSN_DATA_SINK_ENDPOINT))
    {
        if(u8ClusterID == WSN_CID_SENSOR_READINGS)
        {

```





```

        ul6BattVoltage = puTransactionInd->uFrame.sMsg.au8TransactionData[1];
        ul6BattVoltage = ul6BattVoltage << 8;
        ul6BattVoltage |= puTransactionInd->uFrame.sMsg.au8TransactionData[0];

        ul6Temperature = puTransactionInd->uFrame.sMsg.au8TransactionData[3];
        ul6Temperature = ul6Temperature << 8;
        ul6Temperature |= puTransactionInd->uFrame.sMsg.au8TransactionData[2];

        ul6Humidity = puTransactionInd->uFrame.sMsg.au8TransactionData[5];
        ul6Humidity = ul6Humidity << 8;
        ul6Humidity |= puTransactionInd->uFrame.sMsg.au8TransactionData[4];

        vTxSerialDataFrame(ul6AddrSrc, ul6Humidity, ul6Temperature,
        ul6BattVoltage);
    }
}
return 0;
}

/*****
 *
 * NAME: JZA_vZdpResponse
 *
 * DESCRIPTION:
 * Called when a ZDP response frame has been received. In this application no
 * action is taken as no ZDP responses are anticipated.
 *
 * PARAMETERS:      Name          RW  Usage
 *                   u8Type        R   ZDP response type
 *                   pu8Payload    R   Payload buffer
 *                   u8PayloadLen  R   Length of payload
 *****/
PUBLIC void JZA_vZdpResponse(uint8  u8Type,
                           uint8  u8LQI,
                           uint8 *pu8Payload,
                           uint8  u8PayloadLen)
{
}

/*****
 *
 * NAME: JZA_vStackEvent
 *
 * DESCRIPTION:
 * Called by Zigbee stack to pass an event up to the application.
 *
 * RETURNS:
 * TRUE
 *
 *****/
PUBLIC void JZA_vStackEvent(teJZS_EventIdentifier eEventId,
                           tuJZS_StackEvent *puStackEvent)
{
    if (eEventId == JZS_EVENT_NWK_STARTED)
    {

        // load the simple descriptor now that the network has started
        uint8 u8InputClusterCnt      = 1;
        uint8 au8InputClusterList[] = {WSN_CID_SENSOR_READINGS};
        uint8 u8OutputClusterCnt    = 0;
        uint8 au8OutputClusterList[] = {};

        (void)afmeAddSimpleDesc(WSN_DATA_SINK_ENDPOINT,

```





```
WSN_PROFILE_ID,  
0x0000,  
0x00,  
0x00,  
u8InputClusterCnt,  
au8InputClusterList,  
u8OutputClusterCnt,  
au8OutputClusterList);  
  
    bNwkStarted = TRUE;  
}  
  
/*****  
END OF FILE  
*****/
```

## 6.2. Program routera

```
/*****  
Include files  
*****/  
#include <jendefs.h>  
#include <ALSdriver.h>  
#include <HTSdriver.h>  
#include <LedControl.h>  
#include <AppHardwareApi.h>  
#include <JZ_Api.h>  
#include <gdb.h>  
  
#include "WSN_Profile.h"  
  
/*****  
Macro Definitions  
*****/  
/* Timing values */  
#define APP_TICK_PERIOD_ms          100  
#define APP_TICK_PERIOD             (APP_TICK_PERIOD_ms * 32)  
  
#define APP_DATA_SEND_PERIOD_ms     1000  
#define APP_DATA_SEND_PERIOD        (APP_DATA_SEND_PERIOD_ms / APP_TICK_PERIOD_ms)  
  
/*****  
Type Definitions  
*****/  
  
/* Battery reading state definitions */  
typedef enum  
{  
    E_STATE_READ_BATT_VOLT_IDLE,  
    E_STATE_READ_BATT_VOLTS_ADC_CONVERTING,  
    E_STATE_READ_BATT_VOLTS_COMPLETE,  
    E_STATE_READ_BATT_VOLTS_READY  
}teStateReadBattVolt;  
  
/* Temperature/Humidity Sensor - reading state definitions */  
typedef enum  
{  
    E_STATE_READ_TEMP_HUMID_IDLE,  
    E_STATE_READ_HUMID_RUNNING,  
    E_STATE_READ_TEMP_HUMID_COMPLETE,
```





```
E_STATE_READ_TEMP_START,  
E_STATE_READ_TEMP_HUMID_RUNNING,  
E_STATE_READ_TEMP_COMPLETE,  
E_STATE_READ_TEMP_HUMID_READY  
}teStateReadTempHumidity;  
  
/* Battery measurement data */  
typedef struct  
{  
    uint16_t          teStateReadBattVolt      eState;  
    uint16_t          u16Reading;  
}tsBattSensor;  
  
/* Temp/Humidity measurement data */  
typedef struct  
{  
    uint16_t          u16TempReading;  
    uint16_t          u16HumidReading;  
    uint16_t          teStateReadTempHumidity eState;  
}tsTempHumiditySensor;  
  
/***** Local Function Prototypes *****/  
PRIVATE void vInit(void);  
PRIVATE void vSendData(void);  
PRIVATE void vInitSensors(void);  
PRIVATE void vReadTempHumidity(void);  
PRIVATE void vReadBatteryVoltage(void);  
PRIVATE void vAppTick(void *pvMsg, uint8_t u8Param);  
  
/***** Local Variables *****/  
PRIVATE uint8_t u8AppTicks = 0;  
PRIVATE tsBattSensor sBattSensor;  
PRIVATE tsTempHumiditySensor sTempHumiditySensor;  
PRIVATE bool_t bAppTimerStarted = FALSE;  
PRIVATE bool_t bNwkJoined = FALSE;  
  
/*****  
 *  
 * NAME: AppColdStart  
 *  
 * DESCRIPTION:  
 * Entry point for application. Initialises system, starts scan then  
 * processes interrupts.  
 *  
 * RETURNS:  
 * void, never returns  
 *  
 *****/  
PUBLIC void AppColdStart(void)  
{  
    /* Debug hooks: include these regardless of whether debugging or not */  
    HAL_GDB_INIT();  
    HAL_BREAKPOINT();  
  
    /* General initialisation: reset hardware */  
    JZS_sConfig.u32Channel = WSN_CHANNEL;  
    JZS_sConfig.u16PanId = WSN_PAN_ID;  
  
    /* General initialisation: reset hardware */  
    vInit();  
}
```





```
    /* No return from the above function call */
}

/*****
 *
 * NAME: AppWarmStart
 *
 * DESCRIPTION:
 * Entry point for application from boot loader. Simply jumps to AppColdStart
 * as, in this instance, application will never warm start.
 *
 * RETURNS:
 * Never returns.
 *
 *****/
PUBLIC void AppWarmStart(void)
{
    AppColdStart();
}

/*****
 **** Local Functions ****
 *****/
/*****
 *
 * NAME: vInit
 *
 * DESCRIPTION:
 * Initialises Zigbee stack and hardware. Final action is to start BOS, from
 * which there is no return. Subsequent application actions occur in the
 * functions defined above.
 *
 * RETURNS:
 * No return from this function
 *
 *****/
PRIVATE void vInit(void)
{
    /* Initialise Zigbee stack */
    JZS_u32InitSystem(TRUE);

    /* Set DIO for LEDs */
    vLedInitRfd();
    vLedControl(0,0);
    vLedControl(1,0);

    /* Set sensors */
    vInitSensors();

    /* Start BOS */
    (void)bBosRun(TRUE);

    /* No return from the above function call */
}

/*****
 *
 * NAME: vInitSensors
 *
 * DESCRIPTION:
 * Initialise the temperature/humidity sensor and set the ADC to measure the
 * supply voltage.
 *****/
```





```
*
*****/
PRIVATE void vInitSensors(void)
{
    /* Initialise temp/humidity sensor interface */
    vHTSreset();
    sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_IDLE;

    /* Initialise ADC for internal battery voltage measurement */
    sBattSensor.eState = E_STATE_READ_BATT_VOLT_IDLE;

    vAHI_ApConfigure(E_AHI_AP_REGULATOR_ENABLE,
                    E_AHI_AP_INT_DISABLE,
                    E_AHI_AP_SAMPLE_2,
                    E_AHI_AP_CLOCKDIV_2MHZ,
                    E_AHI_AP_INTREF);

    /* Wait until the analogue peripheral regulator has come up before setting
       the ADC. */
    while(!bAHI_APRegulatorEnabled());

    vAHI_AdcEnable(E_AHI_ADC_CONVERT_DISABLE,
                  E_AHI_AP_INPUT_RANGE_2,
                  E_AHI_ADC_SRC_VOLT);
}

/*****
*
* NAME: vAppTick
*
* DESCRIPTION:
*
* Called by a BOS timer expiry. Reads sensor data and if complete transmits
* to coordinator.
*
*****/
PRIVATE void vAppTick(void *pvMsg, uint8 u8Param)
{
    uint8 u8Msg;
    uint8 u8TimerId;
    static bool_t bToggle;

    /* Read sensor data */
    vReadTempHumidity();
    vReadBatteryVoltage();

    if (u8AppTicks++ > APP_DATA_SEND_PERIOD)
    {
        /* If sensor reads are complete */
        if ((sBattSensor.eState == E_STATE_READ_BATT_VOLTS_READY) &&
            (sTempHumiditySensor.eState == E_STATE_READ_TEMP_HUMID_READY))
        {
            /* Toggle LED1 to show we are alive */
            if (bToggle)
            {
                vLedControl(0,0);
            }
            else
            {
                vLedControl(0,1);
            }
            bToggle = !bToggle;

            u8AppTicks = 0;
        }
    }
}
```





```
/* Transmit data to coordinator */
    vSendData();

    sBattSensor.eState = E_STATE_READ_BATT_VOLT_IDLE;
    sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_IDLE;
}
}
(void)bBosCreateTimer(vAppTick, &u8Msg, 0, (APP_TICK_PERIOD_ms / 10),
&u8TimerId);
}

/*****
 *
 * NAME: vReadBatteryVoltage
 *
 * DESCRIPTION:
 *
 * Uses ADC to read supply voltage. Measurement is performed using a state
 * machine to ensure that it never blocks.
 *
 *****/
PRIVATE void vReadBatteryVoltage(void)
{
    uint16 u16AdcReading;

    switch(sBattSensor.eState)
    {
        case E_STATE_READ_BATT_VOLT_IDLE:
            vAHI_AdcStartSample();
            sBattSensor.eState = E_STATE_READ_BATT_VOLTS_ADC_CONVERTING;
            break;

        case E_STATE_READ_BATT_VOLTS_ADC_CONVERTING:
            if (!bAHI_AdcPoll())
            {
                sBattSensor.eState = E_STATE_READ_BATT_VOLTS_COMPLETE;
            }
            break;

        case E_STATE_READ_BATT_VOLTS_COMPLETE:

            u16AdcReading = u16AHI_AdcRead();

            /* Input range is 0 to 2.4V. ADC has full scale range of 12 bits.
               Therefore a 1 bit change represents a voltage of approx 586uV
            */

            sBattSensor.u16Reading = ((uint32)((uint32)(u16AdcReading * 586) +
                ((uint32)(u16AdcReading * 586) >> 1))) /
                1000;

            sBattSensor.eState = E_STATE_READ_BATT_VOLTS_READY;
            break;

        case E_STATE_READ_BATT_VOLTS_READY:
            break;

        default:
            break;
    }
}

/*****
 *
 */
```





```
* NAME: vReadTempHumidity
*
* DESCRIPTION:
*
* Read temperature/humidity sensor. Reading is performed using a state machine
* to ensure that it never blocks.
*
*****/
PRIVATE void vReadTempHumidity(void)
{
    switch(sTempHumiditySensor.eState)
    {
        case E_STATE_READ_TEMP_HUMID_IDLE:
            vHTSstartReadHumidity();
            sTempHumiditySensor.eState = E_STATE_READ_HUMID_RUNNING;
            break;

        case E_STATE_READ_HUMID_RUNNING:
            if ((u32AHI_DioReadInput() & HTS_DATA_DIO_BIT_MASK) == 0)
            {
                sTempHumiditySensor.eState =
E_STATE_READ_TEMP_HUMID_COMPLETE;
            }
            break;

        case E_STATE_READ_TEMP_HUMID_COMPLETE:
            sTempHumiditySensor.ul6HumidReading =
ul6HTSreadHumidityResult();
            sTempHumiditySensor.eState = E_STATE_READ_TEMP_START;
            break;

        case E_STATE_READ_TEMP_START:
            vHTSstartReadTemp();
            sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_RUNNING;
            break;

        case E_STATE_READ_TEMP_HUMID_RUNNING:
            if ((u32AHI_DioReadInput() & HTS_DATA_DIO_BIT_MASK) == 0)
            {
                sTempHumiditySensor.eState = E_STATE_READ_TEMP_COMPLETE;
            }
            break;

        case E_STATE_READ_TEMP_COMPLETE:
            sTempHumiditySensor.ul6TempReading = ul6HTSreadTempResult();
            sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_READY;
            break;

        case E_STATE_READ_TEMP_HUMID_READY:
            break;

        default:
            break;
    }
}

/*****
*
* NAME: vSendData
*
* DESCRIPTION:
*
* Transmit sensor data to coordinator.
*
*****/
```







```
***** /
PRIVATE void vSendData(void)
{
    AF_Transaction_s asTransaction[1];

    asTransaction[0].u8SequenceNum = u8AfGetTransactionSequence(TRUE);
    asTransaction[0].uFrame.sMsg.u8TransactionDataLen = 6;

    asTransaction[0].uFrame.sMsg.au8TransactionData[0] = sBattSensor.ul6Reading;
    asTransaction[0].uFrame.sMsg.au8TransactionData[1] = sBattSensor.ul6Reading
>> 8;
    asTransaction[0].uFrame.sMsg.au8TransactionData[2] =
sTempHumiditySensor.ul6TempReading;
    asTransaction[0].uFrame.sMsg.au8TransactionData[3] =
sTempHumiditySensor.ul6TempReading >> 8;
    asTransaction[0].uFrame.sMsg.au8TransactionData[4] =
sTempHumiditySensor.ul6HumidReading;
    asTransaction[0].uFrame.sMsg.au8TransactionData[5] =
sTempHumiditySensor.ul6HumidReading >> 8;

    (void)afdeDataRequest(APS_ADDRMODE_SHORT,      /* Address type */
                          0x0000,                  /* Destination address */
                          WSN_DATA_SINK_ENDPOINT, /* destination endpoint */
                          WSN_DATA_SOURCE_ENDPOINT, /* Source endpoint */
                          WSN_PROFILE_ID,          /* Profile ID */
                          WSN_CID_SENSOR_READINGS, /* Cluster ID */
                          AF_MSG,                  /* Frame type */
                          1,                        /* Transactions */
                          asTransaction,           /* Transaction data */
                          APS_TXOPTION_NONE,       /* Transmit options */
                          SUPPRESS_ROUTE_DISCOVERY, /* Route discovery mode */
                          0);                       /* Radius count */
}

/***** Functions called by the stack *****/
/*****

/*****
*
* NAME: JZA_vZdpResponse
*
* DESCRIPTION:
* Called when a ZDP response frame has been received. In this application no
* action is taken as no ZDP responses are anticipated.
*
* PARAMETERS:      Name          RW  Usage
*                  u8Type         R   ZDP response type
*                  pu8Payload     R   Payload buffer
*                  u8PayloadLen   R   Length of payload
*
***** /
PUBLIC void JZA_vZdpResponse(uint8 u8Type,
                             uint8 u8LQI,
                             uint8 *pu8Payload,
                             uint8 u8PayloadLen)

{
}

/*****
*
* NAME: JZA_pu8AfMsgObject
*
*****
```



```
* DESCRIPTION:
* Called when a MSG transaction has been received with a matching endpoint.
* In this application no action is taken as no MSG transactions are expected.
*
* PARAMETERS:      Name          RW  Usage
*                   afSrcAddr      R   Address of sender device
*                   dstEndPoint    R   Endpoint at receiver
*                   clusterID      R   Pointer to cluster ID
*                   afduLength     R   Pointer to length of data
*                   pAfd           R   Data array
*
* RETURNS:
* NULL
*
***** /
PUBLIC bool_t JZA_bAfMsgObject(APS_Addrmode_e eAddrMode,
                               uint16 u16AddrSrc,
                               uint8 u8SrcEP,
                               uint8 u8LQI,
                               uint8 u8DstEP,
                               uint8 u8ClusterID,
                               uint8 *pu8ClusterIDRsp,
                               AF_Transaction_s *puTransactionInd,
                               AF_Transaction_s *puTransactionRsp)
{
    return 0;
}

/*****
*
* NAME: JZA_vAfKvpResponse
*
* DESCRIPTION:
* Called after a KVP transaction with acknowledgement request, when the
* acknowledgement arrives. In this application no action is taken as no
* KVP transaction acknowledgements are expected.
*
* PARAMETERS:      Name          RW  Usage
*                   srcAddressMod  R   Address of sender device
*                   transactionSequenceNum R   KVP transaction number
*                   commandTypeIdentifier R   KVP command type
*                   dstEndPoint    R   Endpoint at receiver
*                   clusterID      R   Cluster ID
*                   attributeIdentifier R   KVP attribute ID
*                   errorCode       R   Result code
*                   afduLength     R   Length of payload data
*                   pAfd           R   Payload data array
*
***** /
PUBLIC void JZA_vAfKvpResponse(APS_Addrmode_e eAddrMode,
                               uint16 u16AddrSrc,
                               uint8 u8SrcEP,
                               uint8 u8LQI,
                               uint8 u8DstEP,
                               uint8 u8ClusterID,
                               AF_Transaction_s *puTransactionInd)
{
}

/*****
*
* NAME: JZA_eAfKvpObject
*
* DESCRIPTION:
```





```
* Called when a KVP transaction has been received with a matching endpoint.
*
* PARAMETERS:      Name          RW  Usage
*                  afSrcAddr      R   Address of sender device
*                  u8DstEndpoint  R   Endpoint at receiver
*                  pu8ClusterId   R   Pointer to cluster ID
*                  eCommandTypeId R   KVP command type
*                  u16AttributeId  R   KVP attribute ID
*                  pu8AfdLength   R   Pointer to length of data
*                  pu8Afd         R   Data array
*
* RETURNS:
* AF_ERROR_CODE
*
***** /
PUBLIC bool_t JZA_bAfKvpObject(APS_Addrmode_e eAddrMode,
                               uint16 u16AddrSrc,
                               uint8 u8SrcEP,
                               uint8 u8LQI,
                               uint8 u8DstEP,
                               uint8 u8ClusterId,
                               uint8 *pu8ClusterIDRsp,
                               AF_Transaction_s *puTransactionInd,
                               AF_Transaction_s *puTransactionRsp)
{
    return KVP_SUCCESS;
}

/ *****
*
* NAME: JZA_vAppDefineTasks
*
* DESCRIPTION:
* Called by Zigbee stack during initialisation to allow the application to
* initialise any tasks that it requires. This application requires none.
*
* RETURNS:
* void
*
***** /
PUBLIC void JZA_vAppDefineTasks(void)
{
}

/ *****
*
* NAME: JZA_vPeripheralEvent
*
* DESCRIPTION:
* Called when a hardware event causes an interrupt. This function is called
* from within the interrupt context so should be brief. In this case, the
* information is placed on a simple FIFO queue to be processed later.
*
* PARAMETERS: Name          RW  Usage
*              u32Device      R   Peripheral generating interrupt
*              u32ItemBitmap  R   Bitmap of interrupt sources within peripheral
*
***** /
PUBLIC void JZA_vPeripheralEvent(uint32 u32Device, uint32 u32ItemBitmap)
{
}

/ *****
*
```





```
* NAME: JZA_vAppEventHandler
*
* DESCRIPTION:
* Called regularly by the task scheduler. This function reads the hardware
* event queue and processes the events therein. It is important that this
* function exits after a relatively short time so that the other tasks are
* not adversely affected.
*
*****/
PUBLIC void JZA_vAppEventHandler(void)
{
    uint8 u8Msg;
    uint8 u8TimerId;

    if (!bAppTimerStarted)
    {
        if (bNwkJoined)
        {
            bAppTimerStarted = TRUE;
            (void)bBosCreateTimer(vAppTick, &u8Msg, 0, (APP_TICK_PERIOD_ms / 10),
&u8TimerId);
        }
    }
}

/*****
*
* NAME: JZA_boAppStart
*
* DESCRIPTION:
* Called by Zigbee stack during initialisation.
*
* RETURNS:
* TRUE
*
*****/
PUBLIC bool_t JZA_boAppStart(void)
{
    JZS_vStartStack();
    return TRUE;
}

/*****
*
* NAME: JZA_vStackEvent
*
* DESCRIPTION:
* Called by Zigbee stack to pass an event up to the application.
*
* RETURNS:
* TRUE
*
*****/
PUBLIC void JZA_vStackEvent(teJZS_EventIdentifier eEventId,
                            tuJZS_StackEvent *puStackEvent)
{
    if (eEventId == JZS_EVENT_NWK_JOINED_AS_ROUTER)
    {
        bNwkJoined = TRUE;
    }
}

/*****
***      END OF FILE      ***
*****/
```





---

Instrukcja współfinansowana przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

/ \*\*\*\*\* /

