AN1284

Microchip Wireless (MiWi™) Application Programming Interface – MiApp

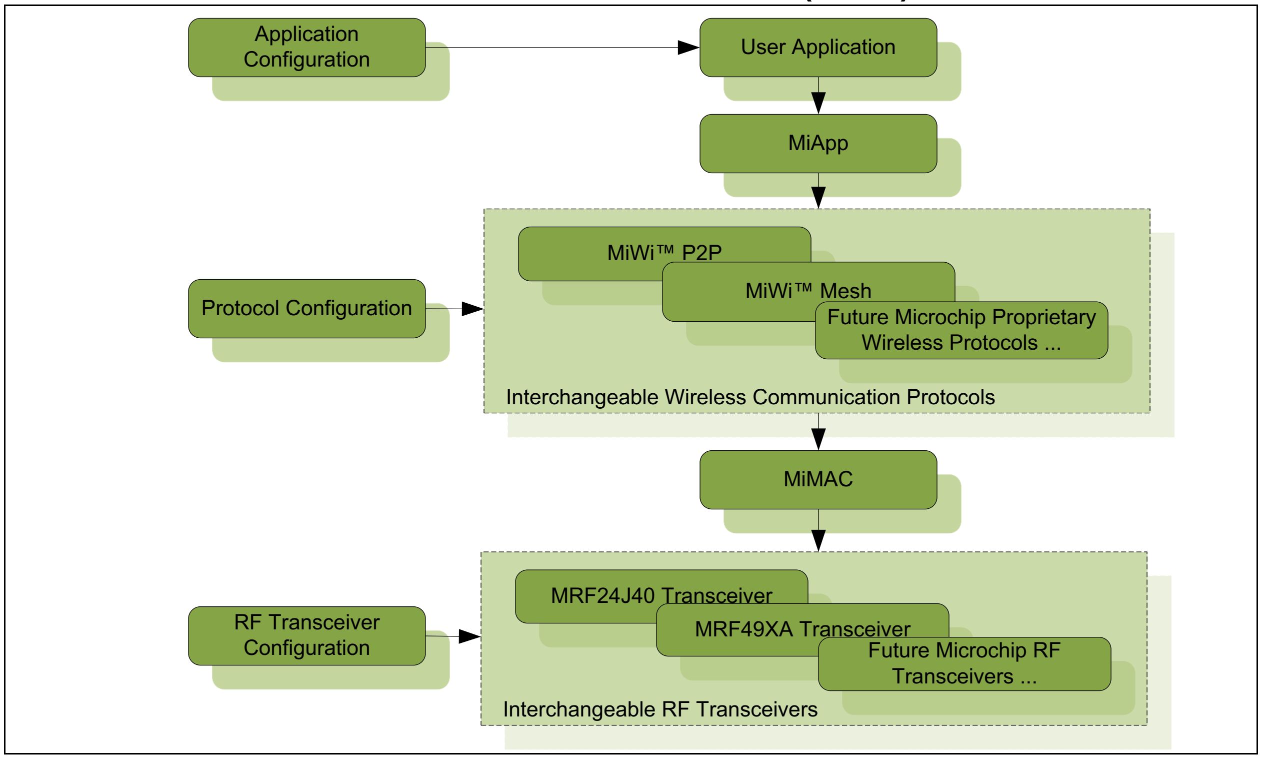
Author: Yifeng Yang
Microchip Technology Inc.

INTRODUCTION

It is not an easy task to develop a short-range, low data rate and low power wireless application. Apart from complex Radio Frequency (RF) circuit designs, the firmware development process may require the developers to understand the details of RF transceivers, as well as the different wireless communication protocols. Microchip has developed a way to handle the complex and difficult RF hardware and/or communication protocol stack software development, which allows wireless developers to focus on their own application development. This is achieved through a concise, yet powerful communication programming interface in the application layer which is called MiApp, and it is defined in this application note.

The MiApp specification defines the programming interfaces between the application layer and Microchip proprietary wireless communication protocols. The MiApp programming interface is implemented in two ways: as configuration parameters defined in the configuration file, and as a set of function calls to the Microchip proprietary wireless protocols. Complying with the MiApp specification defined in this application note, applications can use any Microchip proprietary wireless protocols. With little or no modification in the application layer, software development can be easily changed between a proprietary P2P/star topology connection protocol to a full mesh proprietary networking protocol for small or big networks, depending on the application needs.

FIGURE 1: BLOCK DIAGRAM OF MICROCHIP WIRELESS (MiWi™) STACK


Application Configuration
User Application
MiApp
MiWi™ P2P
MiWi™ Mesh
Future Microchip Proprietary Wireless Protocols ...
Protocol Configuration
Interchangeable Wireless Communication Protocols
MiMAC
MRF24J40 Transceiver
MRF49XA Transceiver
Future Microchip RF Transceivers ...
RF Transceiver Configuration
Interchangeable RF Transceivers

The MiApp specification benefits wireless application developers in multiple ways:

- Wireless application development will focus on the application itself. Complex RF or protocol considerations will be handled transparently by the MiApp programming interface.
- The MiApp specification allows maximum flexibility to choose a wireless protocol at any stage of application software development with little effort, thus greatly lowering the risk of software development. Application requirement changes in networking capabilities have little or no impact in application development.
- MiApp uses the same control interface for Microchip wireless proprietary protocols. Once you are familiar with MiApp, you can apply that knowledge to the development of another application even if it has a completely different networking capability requirement.
- By communicating to the Microchip proprietary protocols, MiApp indirectly talks to the Microchip RF transceivers through the MiMAC interface. As a result, MiApp indirectly enables the wireless application developers to switch between Microchip RF transceivers through MiMAC. This flexibility, in turn, further reduces the development risk of the wireless application project.

FEATURES

The MiApp programming interface has the following features:

- Easy to learn and use
- Powerful interface to meet most requirements from wireless applications
- Little or no extra effort to migrate the wireless application between Microchip proprietary wireless protocols
- Minimum footprint impact

CONSIDERATIONS

The MiApp specification is designed to support Microchip proprietary wireless communication protocols. Once a wireless application is implemented by the MiApp programming interface, the Microchip RF transceivers are also supported through standardization in MiMAC, the module defined in the Media Access Controller (MAC) layer.

MiMAC standardizes the interface between Microchip wireless protocols and Microchip RF transceivers. MiMAC makes Microchip RF transceivers interchangeable with little or no change in the software application code. For details of MiMAC, please refer to application note AN1283 "*Microchip Wireless (MiWi) Media Access Controller - MiMAC*".

MiMAC regulates the lower interface of the Microchip proprietary wireless protocols, while MiApp regulates the higher interface of the Microchip proprietary wireless protocols. Working together, both MiMAC and MiApp provide wireless application developers the maximum flexibility to choose the RF transceivers and wireless communication protocols at any stage of software development, thus further minimizing the risk of software development.The block diagram in Figure 1 shows the Microchip Wireless (MiWi™) stack offerings.

There are three layers of configurations for application, protocol stacks and RF transceivers. Application configuration might change between devices in the same application according to their hardware design, role in the application and network. Wireless application developers tend to do the majority of the configuration in the application layer. Protocol configurations fine-tune the behavior of the protocol stack. The majority of protocol stack configurations define the timing and routing mechanism for the chosen wireless protocol. Transceiver configurations define the frequency band, data rate and other RF related features of the RF transceiver. The default settings for the protocol and RF transceiver configurations may work with the application without any modification. The application configurations, however, usually need to be changed to fit the needs of different wireless applications.

MiApp OVERVIEW

As discussed earlier, there are two parts defined in the MiApp specification:

- Configuration parameters defined in the configuration file
- Signatures of function calls to the Microchip proprietary wireless communication protocols

The configuration file contains parameters that should be set before compilation. Generally speaking, two pieces of information are defined in the configuration file:

- Hardware Definitions: Including MCU hardware resources, peripherals definition and the RF transceiver control pins' definitions. The default hardware definitions have already been defined for several Microchip standard demo boards that support Microchip RF transceivers. In these cases, the definition of demo boards introduces all hardware definitions automatically.
- Software Definitions: These definitions control the code sections to be compiled into the firmware hex file. The software definitions include selections of Microchip proprietary wireless protocol, choice of Microchip RF transceiver and individual functionalities. Proper definitions in this category ensure the minimum firmware footprint with the intended protocol capabilities.

Application Programming Interfaces (APIs) are the function calls between the Microchip proprietary wireless communication protocols with the wireless developer's application. As a rule, the application interface must be clean, concise, easy to understand and powerful.

There are five categories of interfaces for the APIs:

- The initialization interface allows wireless application developers to properly initialize the Microchip proprietary wireless protocol that has been selected in the configuration file.
- The hand-shaking interface allows the wireless nodes to discover and get connected with their peers, or to join the network.
- Interfaces to send messages which enable application developers to transmit information from the current node to an intended audience over the air.
- Interfaces to receive messages which enable application developers to receive information over the air from other devices.
- Special functionalities which ensure the optimal operating condition for wireless nodes through environment noise control and power saving.

MiApp CONFIGURATION FILE

Of the two kinds of configurations in the MiApp configuration file, the hardware definitions depend heavily on the demo board, MCU and RF transceiver choice. Hardware definitions can be divided into following sub categories:

- I/Os on the demo board – push buttons, LEDs, serial ports, etc.
- MCU system resources – timers, interrupts, etc.
- Interconnections between MCU and RF transceiver

Hardware definitions are mainly associated with hardware selections of the wireless application system design. They depend more on the hardware than the software and vary across different designs. As a result, MiApp does not have a set of standards for those hardware definitions.

Selective compilation configurations select the features among the list of available ones. Using the selective compilation, application developers are able to configure Microchip proprietary wireless protocols to perform the desired functionality with the least possible system resources. Table 1 describes the possible selective compilation configurations, as well as the scope, value and functionalities of those selections.

TABLE 1: SOFTWARE DEFINITIONS IN CONFIGURATION FILE

Example of Definition	Functionality	Restriction

`#define PROTOCOL_MIWI`
`#define PROTOCOL_P2P`
	Selects the Microchip wireless protocol to be used in the wireless application.	Only one protocol can be defined at any one time.

`#define MRF24J40`
`#define MRF49XA`
	Selects the Microchip RF transceiver to be used in the wireless application.	Only one transceiver can be defined at any one time.

`#define TX_BUFFER_SIZE 40`	Defines the maximum size of the application payload to be transmitted, excluding all protocol headers.	There may be RF transceiver hardware restrictions on the size of buffer that can be transmitted. The hardware restriction includes all protocol headers.

TABLE 1: SOFTWARE DEFINITIONS IN CONFIGURATION FILE (CONTINUED)

Example of Definition	Functionality	Restriction
`#define RX_BUFFER_SIZE 40`	The maximum size of application payload to be received, excluding all protocol headers.	There may be RF transceiver hardware restrictions on the size of buffer that can be received. The hardware restriction includes all protocol headers.
`#define CONNECTION_SIZE 10`	The size of connection table. Determines the maximum number of devices that the node can connect to.	Depends upon available MCU RAM.
`#define ADDITIONAL_NODE_ID_SIZE 0`	Defines the size of additional information attached to the packets in the hand-shake process. Primarily used to identify the node in the application layer.	The additional node identifier plays no role in Microchip's proprietary protocols. However, it may play an important role in the application. In a simple case of light and switch, two lights may not be interested in connecting to each other, and the same applies to two switches. Using the additional node identifier enables the application to identify the role of the node in the application so that switches only connect with lights.
`#define ENABLE_PA_LNA`	Enables the RF transceiver to use an external power amplifier and/or a low noise amplifier	For RF transceivers that can control an external PA and/or LNA.
`#define ENABLE_HAND_SHAKE`	Enables Microchip's proprietary wireless protocol to establish connections with peers automatically.	Hand-shake process enables two wireless nodes to know each other. In other protocols, this process is also called "Pairing". Applications without handshake only use broadcast to exchange messages.
`#define ENABLE_SLEEP`	Enables the RF transceiver to go to sleep when idle to save power.	Sleep mode depends on the capability of the RF transceiver.
`#define ENABLE_ED_SCAN`	Enables the Microchip proprietary wireless protocol and RF transceiver to perform an energy detection scan.	The energy scan depends on the capability of the RF transceiver.
`#define ENABLE_ACTIVE_SCAN`	Enables the Microchip proprietary wireless protocol to perform an active scan to discover nodes and networks in the neighborhood.	Active Scan is used to search for existing wireless devices of the same kind in the neighborhood. Active Scan can be used to decide which device to connect to.
`#define ENABLE_SECURITY`	Enables Microchip's proprietary protocol to secure packets that are transferred.	The security engine, security mode and keys are defined in a configuration file for the RF transceiver, as security is defined as part of MiMAC.
`#define ENABLE_INDIRECT_MESSAGE`	Enables the wireless node to cache messages for sleeping devices and to deliver them once the sleeping device wakes up and asks for the messages.	Only wireless nodes that do not go to sleep can cache message for sleeping nodes. The number of messages which can be cached depends on the available MCU RAM.
`#define RFD_WAKEUP_INTERVAL 5`	Defines, in seconds, the RFD devices' wake-up time interval.	Only effective when indirect message is enabled. This definition is used for devices that are always awake to keep track of timeouts for indirect messages. The sleeping time of sleeping devices depends on the WDT setting of the host MCU.

TABLE 1: SOFTWARE DEFINITIONS IN CONFIGURATION FILE (CONTINUED)

Example of Definition	Functionality	Restriction
`#define ENABLE_BROADCAST`	Enables the wireless node to handle broadcast messages for sleeping devices.	Only wireless nodes that do not go to sleep can cache messages for sleeping nodes.
`#define ENABLE_FREQUENCY_AGILITY`	Enables Microchip's proprietary wireless protocol to perform frequency agility procedures.	N/A
`#define HARDWARE_SPI`	Enables the MCU to use the hardware SPI to communicate with the transceiver.	Defining of HARDWARE_SPI enables the MCU to use the hardware SPI to communicate with the transceiver. Otherwise, the MCU can use bit-bang to simulate SPI communication with transceiver.
`#define NWK_ROLE_COORDINATOR` `#define NWK_ROLE_END_DEVICE`	Defines the current device's role in the network.	This configuration is only used for network protocol. P2P protocol, like MiWi™ P2P, does not use this configuration.
`#define TARGET_SMALL`	Minimizes the footprint of Microchip's proprietary wireless protocols.	Some features of the Microchip proprietary wireless protocol may not be supported when minimizing the footprint of the protocol.
`#define ENABLE_NETWORK_FREEZER`	Enables the Microchip proprietary wireless protocol to store critical network parameters and to recover from power loss to the original network setting.	Requires nonvolatile memory of either MCU data EEPROM, external EEPROM or programming space. Network size and chosen wireless protocol decides the total amount of nonvolatile memory required.

MiApp FUNCTION INTERFACES

Other than the options in the configuration file, the application layer also uses function calls to communicate with the Microchip proprietary wireless protocol layer, thus controlling the transceiver indirectly to perform wireless communication. There are five categories of function calls to the protocol layers from the application layer:

- Initialization
- Hand-shaking
- Sending Messages
- Receiving Messages
- Special Functionality

The following sections describe the function interfaces in detail, as well as associated structure definitions.

Initialization

To initialize the RF transceiver and protocol stack, the application layer only needs to trigger the initialization process by calling the function *ProtocolInit*. The full function signature is:

```
BOOL MiApp_ProtocolInit(BOOL bNetworkFreezer);
```

There is only one parameter for the initialization. The input boolean decides if the network freezer feature is performed during the initialization. When the network freezer feature is performed, the old network settings that are stored in nonvolatile memory will be restored. The return value is a boolean to indicate if the operation is successful.

Other than the normal initialization process, wireless applications may need to change the transmit or receive frequency during operation. MiApp defines the following function to change the operating frequency of the RF transceiver according to the predefined channel. Each channel defines the frequency either according to the specification, or the RF transceiver settings under different operating frequency bands. The function signature is:

```
BOOL MiApp_SetChannel(BYTE Channel);
```

The only input parameter is the channel to be set. The return value indicates if the operation is successful. The possible channel numbers are from 0 to 31. Depending on the RF transceiver, frequency band and data rate, not all channels from 0 to 31 may be valid under all conditions. If the input channel is invalid under current conditions, the operating channel is not changed and the return value will be FALSE to indicate failure.

Hand Shaking

Unless hard coded in manufacturing, in most applications, the two communication endpoints need an introduction before they can unicast messages between a pair of wireless nodes. The introduction for a networking protocol is sometimes called joining the network. For the P2P protocol, this process can also be called pairing. Since this strategy does not focus on any particular topology or protocol, this process can generally be called the hand-shaking phase. Without a hand-shaking process, wireless nodes can only use broadcast, which treats every wireless node in the source radio range as the audience, to communicate with each other.

The following function calls for hand-shaking are available to the application layer:

- MiApp_StartConnection
- MiApp_SearchConnection
- MiApp_RemoveConnection
- MiApp_ConnectionMode

MiApp_StartConnection

The function call MiApp_StartConnection will enable a wireless node to start operating in different ways. There are three ways to start a PAN: start a PAN directly on a particular channel, or start a PAN after either of the two channel assessments. The full function signature is:

```
BOOL StartConnection(BYTE Mode, BYTE
ScanDuration, DWORD ChannelMap,
BYTE *DestAddr);
```

The return value of the function call indicates if the operation is successful.

The input parameter mode specifies the mode of starting the PAN. The possible modes are:

- **START_CONN_DIRECT**: Start the connection at the current channel without any channel assessment.
- **START_CONN_ENERGY_SCN**: Start the connection after an energy detection scan and the PAN start at the channel with the lowest energy.
- **START_CONN_CS_SCN**: Start the connection after a carrier sense scan and the PAN start at the channel with the lowest carrier sense detected.

For the transceivers that do not support energy detection and/or carrier sense scan, those modes are not valid and the function should start the PAN without any channel assessment if such a mode is specified in the input parameter.

The input parameter ScanDuration specifies the maximum time to perform the channel assessment. The max-and-hold method should be applied for the scan period, if multiple scans can be performed. In case the starting mode specifies no channel assessment, this input parameter will be discarded. The value of the input parameter ScanDuration complies with the definition in the IEEE 802.15.4™ specification. Its range is from 1 to 14. Equation 1 is the formula to calculate the scan duration time.

EQUATION 1: SCAN DURATION CALCULATION

$$ScanTime(us) = 960 * (2^{ScanDuration} + 1)$$

As the formula shows, a ScanDuration of 10 is roughly one second. An increase by one roughly doubles the time, while a decrease by one roughly cuts the time in half.

The input parameter ChannelMap specifies the channels to be scanned in the process. ChannelMap is defined as a 4-byte double word. It uses bit-map to represent channel 0 to channel 31. When a bit is set in the double word, it means that the corresponding channel will perform the channel assessment. For instance, if bit 0 of the input parameter ChannelMap is set, channel 0 will perform channel assessment. To perform channel assessment on all available channels, the input parameter ChannelMap will be 0xFFFFFFFF.

MiApp_SearchConnection

The function call MiApp_SearchConnection searches for and discovers the existing peer wireless nodes in the neighborhood. This procedure is also known as active scan. In some applications, this step informs the device whether it should start a PAN or choose a PAN to join. If a PAN is started, this procedure can be used to decide which PAN identifier to chose. If the device joins a PAN, this procedure is used to choose which PAN and which device to join.

The full function signature is:

```
BYTE  SearchConnection(BYTE  ScanDuration,
DWORD ChannelMap);
```

The return value of this function indicates the total number of returned PANs. The result of the return PAN will be stored in the global variable in the format of structure ACTIVE_SCAN_RESULT, which is defined as following:

```
typedef struct
{
     BYTE Channel;
     BYTE Address[];
     WORD_VAL PANID;
     BYTE RSSI;
     BYTE LQI;
     union
     {
        BYTE Val;
        struct
        {
          BYTE Role: 2;
          BYTE Sleep: 1;
          BYTE SecurityEn: 1;
          BYTE RepeatEn: 1;
          BYTE AllowJoin: 1;
          BYTE Direct: 1;
          BYTE altSrcAddr: 1;
        } bits;
     } Capability
} ACTIVE_SCAN_RESULT;
```

In this structure, element address indicates the address of the device that responded to the active scan.

Element PANID indicates the PAN identifier, if available. The PAN identifier is used to specify the network ID.

Elements RSSI and LQI indicate the strength and quality of the responding signal, respectively. This information may not be available for all RF transceivers.

Element Capability contains information regarding the capability of the device that sends back the response. It is a bitmap of capabilities, which is defined in the union. Depending upon the protocol used under the application layer, the capability information may not be available.

MiApp_RemoveConnection

The function call MiApp_RemoveConnection allows the current node to disconnect certain connections. The full function signature is:

```
void MiApp_RemoveConnection(BYTE
ConnectionIndex);
```

There is no return value for this function. The input parameter ConnectionIndex specifies the index in the connection table for the peer node to be removed. If the ConnectionIndex is 0xFF, the device will remove all connections and leave the network. In a network protocol, this also means that all the device’s children will leave the network. In case that the ConnectionIndex points to the parent node in a network protocol, the current node and all of its children must leave the network. If the connection index points to a node that is not the current node’s parent, the connection is removed and the device stays in the PAN.

MiApp_EstablishConnection

The function call MiApp_EstablishConnection will establish a connection with one or more devices. The full function signature is:

```
BYTE EstablishConnection(BYTE
ActiveScanIndex, BYTE Mode);
```

This function call will return a byte to indicate the index of the new peer node in the connection table. If the return value is 0xFF, it means the procedure to establish a connection has failed after attempting the predefined retry times. If there are multiple connections established during the procedure, the return value is the index of the connection table for one of the connections.

The parameter ActiveScanIndex is the index in the active scan result table for the node to establish connection. If the value is 0xFF, the protocol will try to establish a connection with any device. Because of this, multiple connections may be established in the process.

The parameter mode specifies the connection mode. There are two modes defined:

- **MODE_DIRECT**: This mode directly establishes a connection in the radio range. The P2P stack uses this mode to establish a connection, while a network protocol uses it to establish a connection with a parent to join the network.
- **MODE_INDIRECT**: This mode is used by a network protocol to establish a connection across the network with one or more hops. The connected devices may or may not be in the radio range of the requesting node. In the MiWi application note (AN1066), this kind of connection is also defined as a cluster socket, if the input parameter ActiveScanIndex is 0xFF.

MiApp_ConnectionMode

The function call MiApp_ConnectionMode sets the connection mode that regulates whether the current wireless node is able to accept direct connections from new devices. The full function signature is:

```
void MiApp_ConnectionMode(BYTE Mode);
```

There is no return value for this function. The input parameter “mode” indicates the mode of the operation. The possible modes of operation are:

- **ENABLE_ALL_CONN**:: This mode enables the connection under any condition. This is the default mode when the application starts.
- **ENABLE_PREV_CONN**: This mode only enables old connections. Connection requests from nodes that are already on the connection table will be allowed. Otherwise, the request will be ignored.
- **ENABLE_ACTIVE_SCAN_RSP**: This mode enables the current node to respond to any active scan request to identify itself.
- **DISABLE_ALL_CONN**: This mode disables all connection requests, including active scan.

The connection privilege decreases from ENABLE_CONN to DISABLE_ALL_CONN. Any higher privilege has all the rights for the lower one.

Sending Messages

The most important functionality of a wireless node is to communicate, or send and receive data. All protocols have reserved buffers for the data transfer, with the size equal or larger than TX_BUFFER_SIZE defined in the configuration file. Two functions are defined to manage the TX buffer in the stack:

```
void MiApp_FlushTx(void);
void MiApp_WriteData(BYTE OneByteTxData);
```

The function MiApp_FlushTx is used to reset the pointer of the transmission buffer in the stack. It has no parameter and no return value.

The function MiApp_WriteData is used to fill one byte of data to the transmission buffer in the stack. The only input parameter is the one byte of data to be filled into the transmission buffer.

Usually, MiApp_FlushTx is called first to reset the buffer pointer. Then MiApp_WriteData is called multiple times to fill the transmission buffer, one byte at a time.

After the transmission buffer is filled, the next step is to trigger the message to be transmitted by the protocol layer. There are three ways to transmit a message:

- Broadcast
- Unicast to the node by its index in the connection table
- Unicast to the node by its address, either the permanent address or the alternative network address.

Broadcasting a message targets all devices regardless of their addresses. The full function signature for a broadcast can be found below:

```
BOOL MiApp_BroadcastMessage(BOOL SecEn);
```

The return value of this function call indicates if the transmission is successful. The only input parameter, SecEn, is a boolean to specify if the payload needs to be secured.

Unicast targets a single device as a destination. There are two ways to unicast a message: the destination is represented by an index on the connection table, or the destination address is clearly given, either the permanent address or a network address.

The full function signature for unicast with an index of the connection table is:

```
BOOL MiApp_UnicastConnection(BYTE
ConnectionIndex, BOOL SecEn);
```

The return value of this function call indicates if the transmission is successful. The input parameter ConnectionIndex is the index of the destination node in the connection table. The input parameter SecEn is a boolean to indicate if the payload needs to be secured.

The full function signature for unicast with a destination address is shown below:

```
BOOL  MiApp_UnicastAddress(BYTE  *Address,
BOOL PermanentAddr, BOOL SecEn);
```

The return value of this function call indicates if the transmission is successful.

The input parameter address is the pointer that points to the destination address.

The input boolean parameter PermanentAddr indicates if the destination address is a permanent address or an alternative network address. For star or P2P topology protocol, only the permanent address is used, thus the input parameter PermanentAddr has no effect.

The input parameter SecEn indicates if the payload needs to be secured.

Receiving Messages

The other important functionality of the transceiver is to receive messages. The application layer needs to know when a message is received, the content of the message and, occasionally, how the message is received. The application layer also needs to discard the message so resources can be released and new messages can be received and processed. To work with the flow described, there are two function calls and one structure to define.

MiApp_MessageAvailable

The function call MiApp_MessageAvailable has no input parameter and returns a boolean to indicate if a new message has been received and is available for processing in the application layer. The full function signature is:

```
BOOL MessageAvailable(void);
```

DATA STRUCTURE FOR RECEIVED MESSAGES

All received messages that are forwarded to the application layer are stored in a global variable defined in the format of RECEIVED_MESSAGE as follows:

```
typedef struct
{
     union
     {
        BYTE Val;
        struct
        }
           BYTE broadcast: 1;
           BYTE ackReq: 1;
           BYTE secEn: 1;
           BYTE repeat: 1;
           BYTE command: 1;
           BYTE srcPrnt: 1;
           BYTE dstPrnt: 1;
           BYTE altSrcAddr: 1;
        } bits
     } flags;

     BYTE *SourceAddress;
     BYTE *Payload;
     BYTE PayloadSize;
     BYTE RSSI;
     BYTE LQI;
} RECEIVED_MESSAGE;
```

Depending upon the transceiver and the Microchip proprietary protocol used, not all elements in the structure are valid.

MiApp_DiscardMessage

The function call MiApp_DiscardMessage has no input parameter and returns no value. The application layer calls this function to notify the Microchip proprietary wireless protocol layer that the current packet is done processing and it is ready to process the next packet. The full function signature is:

```
void DiscardMessage(void);
```

Special Functionality

Some transceivers have special functionalities that enable the protocol stack to be more robust and adaptable to the environment.

NOISE DETECTION SCAN

The noise detection scan enables the transceiver to detect the noise level in the environment. It is valuable to start a new PAN at a quiet frequency, as well as deciding whether channel hopping is necessary and to which channel to hop.

The full function signature is:

```
BYTE MiApp_NoiseDetection(DWORD ChannelMap,
BYTE ScanDuration, BYTE DetectionMode, BYTE
*NoiseLevel);
```

The function call MiApp_NoiseDetection returns the channel with the least amount of noise. The function has four function parameters:

- ChannelMap: This input parameter defines the bitmap of channels to be scanned. For each transceiver, the supported number of channels is different; therefore, not all bitmaps in the input parameter ChannelMap are valid.
- ScanDuration: This input parameter defines the total times of noise detection on each channel. The max-and-hold mechanism is used to detect the noise level on each channel. Input parameter ScanDuration follows the IEEE 802.15.4™ specification, which was detailed earlier in this application note with a formula to calculate real time.
- DetectionMode: This input parameter defines the detection mode to be used: energy detection or carrier sense detection. Not all detection modes are supported by all RF transceivers.
- NoiseLevel: This output parameter returns the noise level on the best channel, or the channel of the return value of this function call. This output parameter enables the application layer to view the noise level on the best possible channel. The higher the NoiseLevel parameter value, the noisier the environment is.

TRANSCEIVER POWER STATE

To enable a wireless node powered by a battery, it is necessary to set the radio transceiver to a different power state, or to put it into sleep and wake it up periodically. The function call MiApp_TransceiverPowerState is defined to achieve this goal:

```
BYTE MiApp_TransceiverPowerState
(BYTE Mode);
```

The only input parameter for this function call is the operation mode. The predefined operation modes are:

- **POWER_STATE_SLEEP**: Puts the transceiver into Sleep mode.
- **POWER_STATE_WAKEUP**: Wakes up the transceiver without sending any data request.
- **POWER_STATE_WAKEUP_DR**: Wakes up the transceiver and then sends out a data request to its main associated device to ask for incoming data.

The function call MiApp_TransceiverPowerState returns a byte to indicate the status of the operation. The predefined operation status return values are:

- **SUCCESS:** Indicates that every operation is successful.
- **ERR_TRX_FAIL:** Indicates that the request to sleep or wake up the transceiver failed.
- **ERR_TXFAIL**: Indicates that the request to send out data failed. This option is only available when WAKE_DR is the operation mode.
- **ERR_RXFAIL**: Indicates that the request to receive data from the parent failed. This option is only available when WAKE_DR is the operation mode.

FREQUENCY AGILITY

The frequency agility is the capability to hop channels during operation to bypass persistent noise at certain frequency.

Not all transceivers and protocols support frequency agility. Frequency agility functions are optional for application interfaces.

There are two functions to establish frequency agility. One function is used to initiate the frequency agility procedure. The other function is used to synchronize the connection if communication is lost due to frequency agility performed at the other end of the communication.

The full function signature to initiate the frequency agility procedure is:

```
BOOL MiApp_InitChannelHopping
(DWORD ChannelMap);
```

The return value of the function call MiApp_InitChannelHopping indicates if the channel hopping operation was successful. The ChannelMap input parameter indicates available channels to move to. The ChannelMap parameter is a bitmap of possible channels. If a channel is available, the corresponding bit (nth bit for channel n) will be set; otherwise, it will be cleared.

The MiApp specification does not define when to start channel hopping. The trigger event can be continuous transmission/receiving failures or just periodically searching for the optimal frequency to operate the wireless application. It is up to the wireless application to decide when to start the channel hopping process. The MiApp specification provides the proper interface to the Microchip proprietary wireless protocols to perform these actions as dictated by the application layer.

Once the channel hopping procedure is done, it is possible that some of the wireless nodes, especially those that were in sleep when idle, do not know that the network has been moved to a different channel. It is necessary to define a function to resynchronize the connection:

```
BOOL  MiApp_ResyncConnection(BYTE  Connec-
tionIndex, DWORD ChannelMap);
```

The return value of the function MiApp_ResynConnection indicates if the resynchronization procedure is successful. There are two input parameters: ConnectionIndex and ChannelMap. ConnectionIndex is the index of the device to be synchronized in the connection table. The parameter ChannelMap is the bitmap of the possible channels to synchronized to.

CONCLUSIONS

For wireless application developers who are looking for a short range, low data rate solution, the requirements differ from point to point communication to routing messages across several hops. The MiApp specification from Microchip provides a low-cost and low-complexity solution to address nearly all those applications. It enables the wireless application developer to use Microchip's proprietary wireless protocols with little or no modification in the migration path. Working with MiMAC at the lower layer, it also indirectly enables developers to choose any existing and future RF transceivers supported by Microchip. It is highly recommended that the readers of this application note also read application note "Microchip Wireless (MiWi™) Media Access Controller - MiMAC" (AN1283) to understand the total solution available for wireless applications from Microchip. Standardization of the lower MAC layer as MiMAC and the higher application layer as MiApp offers wireless application developers maximum flexibility in the software development process.

REFERENCES

IEEE Std 802.15.4-2003, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). New York: IEEE, 2003.

Microchip Wireless (MiWi™) Media Access Control - MiMAC, Yifeng Yang, Microchip Technology Inc. 2009.

MRF24J40 Data Sheet – IEEE 802.15.4™ 2.4GHz RF Transceiver (DS39776), Microchip Technology Inc. 2008

MRF49XA Data Sheet – ISM Band Sub-GHz RF Transceiver (DS70590), Microchip Technology Inc. 2009

AN1066, MiWi™ Wireless Networking Protocol Stack (DS01066), David Flowers and Yifeng Yang, Microchip Technology Inc., 2007

AN1204, Microchip MiWi™ P2P Wireless Protocol (DS01204), Yifeng Yang, Microchip Technology Inc., 2008

REVISION HISTORY

Revision A (July 2009)

- This is the initial release of this document.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

03/26/09