MiWi MESH Version 2.10 Migration Guide

Introduction

As Microchip launches an update to the MiWi Mesh Stack (please refer to the Release Notes for more information), the Stack, which was open sourced earlier, is also migrating to a Library based stack. The new library platform not only offers the same features found in earlier versions, but also supports new features, which reduce the complexities involved when creating bigger networks. This document details the differences between the old and new library based version of the stack, highlights the new features added and provides guidance on the changes that are needed in order for existing customers to migrate from older stack versions to the new library based stack.

Table of Contents

1) Background

MiWi is Microchip's proprietary Wireless Networking stack designed to support low rate personal area networks (LRPAN's).
MiWi supports three network topologies.

- Peer to Peer (P2P)
- STAR
- MESH

In older versions of the stack (prior to version 2.10) MiWi MESH networking stack was further divided into 2 categories.

- Mesh (4 hops)
- Mesh Pro (65 hops)

Both versions of Mesh discussed above have been deprecated and a new library based Mesh networking stack has been released supporting up to 100 hops. Backward compatibility with older versions are maintained in this version. All the existing MiApp API's have been maintained. Several new MiApp functions have been added in order to support some new features. Applications running the previous versions of Mesh stack can be migrated to the new library based mesh stack with little to no changes. It's believed that the migration will be simple, and that most applications will work properly with the new Mesh stack without a great deal of effort or costly software re-development. One of the goals of this new stack version is to wrap the complex Mesh networking algorithms inside a library, so that embedded developer may focus on the MiApp API's when developing his wireless application.

2) Hardware Supported

Microcontrollers: 8 bit PIC and 16 bit PIC microcontrollers.
Transceivers:

- MRF24J40 2.4 GHz transceiver.
- MRF89XA Sub GHz transceiver.

3) Quick Start Guide

Application source code changes where required because of these differences between old MiWi Mesh stack and new library based Mesh stack.

- Library based stack.
- New features added.
- Network is more dynamic and mobile. A wireless node now has the capability to essentially change its role and can be more mobile, easily switching between parents.
- Library based stack now supports the XC8 compiler and XC16.

Comparison of old version of MiWi Mesh and the new Library based Mesh Stack.

The major difference between the old and new version of Mesh stack is the addition of libraries. The two libraries supporting the new stack are

- EightBit_MiWiMeshSubGhzLib.a (8 bit -Sub GHz) , 16Bit_MiWiMeshSubGhzLib.a (16 bit -Sub GHz)
- EightBit_MiWiMesh24GhzLib.a(8 bit -2.4 GHz), 16Bit_MiWiMesh24GhzLib.a (16 bit -Sub GHz)

Previous Version of MiWi Mesh

```
miwi_demo_kit
├── headers
│   ├── framework
│   │   ├── driver
│   │   │   ├── mrf_miwi
│   │   │   │   ├── drv_mrf_miwi.h
│   │   │   │   ├── drv_mrf_miwi_24j40.h
│   │   │   │   ├── drv_mrf_miwi_89xa.h
│   │   │   │   ├── drv_mrf_miwi_crc.h
│   │   │   │   └── drv_mrf_miwi_security.h
│   │   ├── miwi
│   │   │   ├── miwi_api.h
│   │   │   ├── miwi_mesh.h
│   │   │   └── miwi_nvm.h
│   ├── system_config
│   │   ├── miwikit_pic18f46j50_24j40
│   │   └── miwikit_pic18f46j50_89xa
├── Important Files
│   ├── Makefile
├── Linker Files
├── Source Files
│   ├── framework
│   │   ├── driver
│   │   │   ├── mrf_miwi
│   │   │   │   ├── drv_mrf_miwi_24j40.c
│   │   │   │   ├── drv_mrf_miwi_89xa.c
│   │   │   │   ├── drv_mrf_miwi_crc.c
│   │   │   │   └── drv_mrf_miwi_security.c
│   │   ├── miwi
│   │   │   ├── miwi_mesh.c
│   │   │   └── miwi_nvm.c
│   ├── system_config
│   │   ├── miwikit_pic18f46j50_24j40
│   │   └── miwikit_pic18f46j50_89xa
├── Libraries
├── Loadables
```

New Version of MiWi Mesh

```
miwi_demo_kit
├── headers
│   ├── Bridge.h
│   ├── framework
│   │   ├── driver
│   │   │   ├── mrf_miwi
│   │   │   │   ├── drv_mrf_miwi_24j40.h
│   │   │   │   ├── drv_mrf_miwi_89xa.h
│   │   │   │   ├── drv_mrf_miwi_security.h
│   │   │   │   └── drv_mrf_miwi.h
│   │   ├── miwi
│   │   │   ├── miwi_api.h
│   │   │   └── miwi_nvm.h
│   ├── NetworkBridge.h
│   ├── NetworkManager.h
│   ├── system_config
│   │   ├── miwikit_pic18f46j50_24j40
│   │   └── miwikit_pic18f46j50_89xa
├── Important Files
│   ├── Makefile
├── Linker Files
├── Source Files
│   ├── Bridge.c
│   ├── framework
│   │   ├── driver
│   │   │   ├── mrf_miwi
│   │   │   │   ├── drv_mrf_miwi_24j40.c
│   │   │   │   ├── drv_mrf_miwi_89xa.c
│   │   │   │   └── drv_mrf_miwi_security.c
│   │   ├── miwi
│   │   │   └── miwi_nvm.c
│   ├── system_config
│   │   ├── miwikit_pic18f46j50_24j40
│   │   └── miwikit_pic18f46j50_89xa
├── Libraries
│   ├── EightBit_MiWiMesh24Ghz
│   ├── EightBit_MiWiMeshSubGhz
├── Loadables
```

4) New MiApp API's:

Initialization

4.1)void MiApp_SetAddressPan(uint8_t *address, uint16_t panid)

Input Parameters:

address	Short Address of the Node.
panid	Denotes the network PANID.

Return Parameters:

None

Description:

This function is used to set the device address and the PANID. The input parameters to this function are short address of the wireless node and the PANID of the network to which the device will join. The short address should be a unique within the network. The PANID value should be the same for all the nodes within the network. This function is applicable to all device types - Pan Coordinator, Coordinator, Sleeping RFD's.

Restriction:

This function applicable only for MiWi Mesh networks.

4.2)uint16_t MiApp_InitSleepRFDBuffers(uint8_t *buffer, uint16_t bufferSize, uint16_t rfdMaxDataSize);

Input Parameters:

buffer	A pointer to a byte array
bufferSize	Length of buffer
rrfdMaxDataSize	Maximum length in byte, of data to store for each Sleeping RFD

Return Parameters:

Returns number of Sleeping RFD Buffers allocated.

Description:

This function is used to initialize the indirect message buffer within a network. Indirect messages are messages cached by the Pan Coordinator for their sleeping RFD nodes. Upon waking up, Sleeping End Devices may request these cached indirect messages from the Pan Coordinator.

Restriction:

Only the PAN Coordinator within a MiWi Mesh Network may call this function.

Prerequisite:

The macro ENABLE_INDIRECT_MESSAGE should be enabled in the Pan Coordinator.

Data Requesting and Receiving Messages

4.3)void MiApp_RequestData(void)

Input Parameters:

None

Return Parameters:

None

Description:

This function is used by RFD sleeping devices to request from the Pan Coordinator, any data it may have cached for it.

Restriction:

Only RFD nodes within a MiWi Mesh network shall call this function.

Status

4.4)addr_t MiApp_GetParentAddress(void)

Input Parameters:

None

Return Parameters:

addr_t The address of the node's parent.

Description:

This function is used any device to get the address of the node's parent.

Restriction:

This function applicable only to MiWi Mesh networks.

4.5)bool MiApp_IsMemberOfNetwork(void)

Input:

None

Output:

Boolean: True - if a Coordinator or Sleeping End Device that called this function is currently a member of the network.

False – if Coordinator or Sleeping End Device that called this function is currently not part of the network.

Description:

This function returns true if Coordinator or RFD is member of the network. Always returns TRUE for a PAN Coordinator. This function helps a wireless node identify its status in a network. If a node is not actively part of a network, it may send a connection request to begin the process of joining a network.

Miscellaneous Functions

4.6)void MiApp_SetNetworkSecure(void)

Input Parameters:

None

Return Parameters:

None

Description:

This function is used to secure all the messages (data , command , status , control and network maintenance) that are transmitted within a network.

Restriction:

This function is applicable only within MiWi Mesh networks.

5) Example Code Snippets with real application use cases:

The application layer uses various function calls to communicate with the MiWi stack layer, to control the transceiver and to establish wireless communications. Developing a MiWi application generally consists of perform three groups of tasks: configuring the network, connecting wireless nodes in a particular topology, and then communicating among the devices. Communicating involves the transmission and reception of messages.

The following section provides code samples for developing a model MiWi Mesh Application.

5.1) Configurare and Connect:

```
main.c   → Device Type : Pan Coordinator

// Global Variable
DeviceType_t myDeviceType;

void main(void)
{
  // commission
  myDeviceType = DeviceCoordinator;
  MiApp_SetAddressPan(myAddr.bytes, 2byte_PANID);
  MiApp_ProtocolInit(network_freezer_disable);
  MiApp_SetChannel(channel_no);

  // To create a new network
  MiApp_StartConnection (START_CONN_DIRECT, 0, 0);

  // Initialize the buffer for sleeping RFD devices.
  MiApp_InitSleepRFDBuffers(sleepRFDData,sizeof(sleepRFDData),10);
 // Initialize a secure network
 MiApp_SetNetworkSecure(true);
```

main.c → Device Type : Coordinator

```
// Global Variable
DeviceType_t myDeviceType;

void main(void)
{
 // commission
 myDeviceType = DeviceRFD;
 MiApp_SetAddressPan(myAddr.bytes, 2byte_PANID);
 MiApp_ProtocolInit(network_freezer_disable);
 MiApp_SetChannel(channel_no);

// Join or connect to any network that exists
 MiApp_EstablishConnection (0xFF, CONN_MODE_DIRECT);

//////   -- Joining a specific network -- //////////////
 // Scan for available networks or nodes
 ActiveScanResults = MiApp_SearchConnection(ScanDuration , ChannelMap);
 // Join a specified node after getting the scan results
 MiApp_EstablishConnection(ActiveScanIndex, CONN_MODE_DIRECT);

 // Initialize the buffer for sleeping RFD devices.
 MiApp_InitSleepRFDBuffers(sleepRFDData,sizeof(sleepRFDData),10);
 // Initialize a secure network
 MiApp_SetNetworkSecure(true);
```

main.c → Device Type : RFD Sleeping Device

```
// Global Variable
DeviceType_t myDeviceType;
void main(void)
{
 // commission
 myDeviceType = DeviceSleepingRFD;    //Macro ENABLE_SLEEP should be enabled
 MiApp_SetAddressPan(myAddr.bytes, 2byte_PANID);
 MiApp_ProtocolInit(network_freezer_disable);
 MiApp_SetChannel(channel_no);

// Join or connect to any network that exists
 MiApp_EstablishConnection (0xFF, CONN_MODE_DIRECT);

//////   -- Joining a specific network -- //////////////
 // Scan for available networks or nodes
 ActiveScanResults = MiApp_SearchConnection(ScanDuration , ChannelMap);
 // Join a specified node after getting the scan results
 MiApp_EstablishConnection(ActiveScanIndex, CONN_MODE_DIRECT);
```

```
 // Initialize the buffer for sleeping RFD devices.
 MiApp_InitSleepRFDBuffers(sleepRFDData,sizeof(sleepRFDData),10);
 // Initialize a secure network
 MiApp_SetNetworkSecure(true)
```

5.2) Communicate:

5.2.1)Transmit Message:

```
while (1)
{
  // Transmission
    if (button1_pressed)
    {
      // Unicast Message
      MiApp_FlushTx();
      MiApp_WriteData(Data_Byte_1);
      MiApp_UnicastAddress(DestAddr.bytes, NULL, Sec_En)
    }
    else if (button2_pressed)
    {
      // Broadcast Message
      MiApp_FlushTx();
      // Data Type … Added for Future Support of OTA/bootloader
      MiApp_WriteData(mUserDataType);
      // Data Starts here
      MiApp_WriteData(Data_Byte_1);
      MiApp_UnicastAddress(DestAddr.bytes, NULL, Sec_En)
    }
  // Receive Logic Discussed Next …..
```

5.2.2) Receive Message:

```
  // Receive Message
    if (MiApp_MessageAvailable ())
    {
 // Ignore 1st byte … Added for Future Support of OTA/bootloader
 // First data byte starts from index 1
     received_byte1 = rxMessage.Payload[1];
         // Discard the packet once processed.
         MiApp_DiscardMessage ();
    }
 } // End of While 1
} // End of main
```

5.3) Miscellaneous Cases:

If a wireless node is inactive, for example not transmitting nor receiving any packets, the application may call a special function called *MiApp_IsMemberOfNetwork().* If the return value of this function is *TRUE*, the node is still connected to the network. However, if it is *FALSE*, the

wireless node is not part of any network, an must rejoin the network in-order to reestablish communications.

Rejoin on No Network Activity:

```
// Receive Message
  if (MiApp_IsMemberOfNetwork() == false)
  {
    // Join or connect to any network that exists
    MiApp_EstablishConnection (0xFF, CONN_MODE_DIRECT);

     //////   -- Joining a specific network -- //////////////
     // Scan for available networks or nodes
     ActiveScanResults = MiApp_SearchConnection (ScanDuration, ChannelMap);
     // Join a specified node after getting the scan results
     MiApp_EstablishConnection (ActiveScanIndex, CONN_MODE_DIRECT);
  }
```

6) Conclusion:

The new MiWi Mesh stack offers a few addition MiApp APIs, which is intended to provide a more a stable, reliable and flexible MESH network. Building your application in the current library base MESH stack adds little to no additional complexity.