MIWI Wireless Protocol
Team 8

- Paul Krutty – Introduction/Conclusion
- Michael Allon/Donghun Ha – MiMac Layer
- Stephen Hilton – MiApp Layer
- Micah Zastrow – Network Configuration
MIWI Protocol Stack

- MiMac – Link Layer
  - Responsible for making final delivery preparation and sending to physical layer

- MiApp – Application Layer
  - User interaction
Network Configurations

- **Star Network**
  - One central node/multiple branch nodes

- **Cluster Tree**
  - One central coordinator, multiple sub-coordinators
  - Multiple branches on each

- **Mesh**
  - Similar to tree but branch nodes can communicate with one another
MiMAC – Features

- Easy
- Flexibility
  - Simple, but strong, on security module
- Concise, but powerful, programming interface
- Minimum impact to the firmware footprint.
BLOCK DIAGRAM OF MICROCHIP WIRELESS (MiWiTM) SOLUTIONS
MiMAC – 3 Parts

- MiMAC Frame Format
- MiMAC Security Module
- MiMAC Universal Programming Interface
MiMac Frame
MiMac Header: Frame Format

- Packet type
  - Data, command, ack, other
- Broadcast
  - Unicast / Broadcast
- Security
- Repeat
- Acknowledge
- Destination Present
- Source Present
MiMac Header: Extra Control

- Ack info
- Header index
- Payload index
MiMac Header

- Sequence Number
- Destination Address
- Source Address
Payload

Header - Payload - CRC
Security

QuickTime™ and a decompressor are needed to see this picture.
Programming Interface

- Configuration
- Transmitting Packets
- Receiving Packets
- Special Functionality
MiApp – MiWi Application Programming Interface

- What is MiApp?
- Programming interface between the application layer and wireless communication protocols
- Regulates higher interface of the proprietary wireless protocols while MiMAC regulates lower interface
- Implemented through:
  1) Configuration parameters defined in configuration file.
  2) Set of function calls to Microchip proprietary wireless protocols.
Configuration Parameters

- Types of Definitions:
  - Hardware: i.e. MCU system resources, I/O's on demo board, RF transceiver control pins’ definition. In cases of Microchip demo boards, defined automatically
  - Software: i.e. Individual functionalities of wireless protocol to be compiled. Ensure proper protocol capabilities with least possible system resources.
# Software Definitions in Configuration File

<table>
<thead>
<tr>
<th>Example of Definition</th>
<th>Functionality</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define PROTOCOL_MIWI</td>
<td>Selects the Microchip wireless protocol to be used in the wireless application.</td>
<td>Only one protocol can be defined at any one time.</td>
</tr>
<tr>
<td>#define PROTOCOL_P2P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#define MRF24J40</td>
<td>Selects the Microchip RF transceiver to be used in the wireless application.</td>
<td>Only one transceiver can be defined at any one time.</td>
</tr>
<tr>
<td>#define MRF49XA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#define TX_BUFFER_SIZE 40</td>
<td>Defines the maximum size of the application payload to be transmitted, excluding all protocol headers.</td>
<td>There may be RF transceiver hardware restrictions on the size of buffer that can be transmitted. The hardware restriction includes all protocol headers.</td>
</tr>
<tr>
<td>#define CONNECTION_SIZE 10</td>
<td>The size of connection table. Determines the maximum number of devices that the node can connect to.</td>
<td>Depends upon available MCU RAM.</td>
</tr>
<tr>
<td>#define ENABLE_HAND_SHAKE</td>
<td>Enables Microchip's proprietary wireless protocol to establish connections with peers automatically.</td>
<td>Hand-shake process enables two wireless nodes to know each other. In other protocols, this process is also called “Pairing”. Applications without hand-shake only use broadcast to exchange messages.</td>
</tr>
<tr>
<td>#define ENABLE_SLEEP</td>
<td>Enables the RF transceiver to go to sleep when idle to save power.</td>
<td>Sleep mode depends on the capability of the RF transceiver.</td>
</tr>
<tr>
<td>#define ENABLE_ED_SCAN</td>
<td>Enables the Microchip proprietary wireless protocol and RF transceiver to perform an energy detection scan.</td>
<td>The energy scan depends on the capability of the RF transceiver.</td>
</tr>
<tr>
<td>#define ENABLE_ACTIVE_SCAN</td>
<td>Enables the Microchip proprietary wireless protocol to perform an active scan to discover nodes and networks in the neighborhood.</td>
<td>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.</td>
</tr>
<tr>
<td>Example of Definition</td>
<td>Functionality</td>
<td>Restriction</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>#define ENABLE_BROADCAST</td>
<td>Enables the wireless node to handle broadcast messages for sleeping devices.</td>
<td>Only wireless nodes that do not go to sleep can cache messages for sleeping nodes.</td>
</tr>
<tr>
<td>#define ENABLE_FREQUENCY_AGILITY</td>
<td>Enables Microchip's proprietary wireless protocol to perform frequency agility procedures.</td>
<td>N/A</td>
</tr>
<tr>
<td>#define HARDWARE_SPI</td>
<td>Enables the MCU to use the hardware SPI to communicate with the transceiver.</td>
<td>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.</td>
</tr>
<tr>
<td>#define NETWORK_COORDINATOR</td>
<td>Defines the current device's role in the network.</td>
<td>This configuration is only used for network protocol. P2P protocol, like MiWi™ P2P, does not use this configuration.</td>
</tr>
<tr>
<td>#define NETWORK_END_DEVICE</td>
<td>Minimizes the footprint of Microchip's proprietary wireless protocols.</td>
<td>Some features of the Microchip proprietary wireless protocol may not be supported when minimizing the footprint of the protocol.</td>
</tr>
<tr>
<td>#define TARGET_SMALL</td>
<td>Enables the Microchip proprietary wireless protocol to store critical network parameters and to recover from power loss to the original network setting.</td>
<td>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.</td>
</tr>
<tr>
<td>#define ENABLE_SECURITY</td>
<td>Enables Microchip's proprietary protocol to secure packets that are transferred.</td>
<td>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.</td>
</tr>
<tr>
<td>#define ENABLE_INDIRECT_MESSAGE</td>
<td>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.</td>
<td>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.</td>
</tr>
</tbody>
</table>
Function Interfaces

- Function calls are used to communicate with Microchip proprietary wireless protocol layer.

Five Categories:
- Initialization
- Hand-shaking
- Sending Messages
- Receiving Messages
- Special Functionalitity
Initialization

- To Initialize the RF transceiver and protocol stack:

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

- Where bNetworkFreezer allows you to restore to old network setting stored in nonvolatile memory

- To change TX/RX frequency:

```c
BOOL MiApp_SetChannel(BYTE Channel);
```
Hand Shaking – StartConnection

- Start, Search, Remove, Mode
- Modes for StartConnection:
  - Direct Connect on current channel.
  - Lowest Energy Channel
  - Channel with Lowest Carrier Sense

```c
BOOL StartConnection(BYTE Mode, BYTE ScanDuration, DWORD ChannelMap,
BYTE *DestAddr);
```
Hand Shaking - SearchConnection

- RSSI: Strength of Signal
- LQI: Quality of Signal
- Capability of Device

```c
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;
```
Hand Shaking – Remove/Establish

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

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

Modes : Direct and Indirect

- Indirect – Establishes a connection across the network with one or more hops
- Direct – P2P or connection with parent.
Hand Shaking – ConnectionMode

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

Modes:

- Connect under any condition
- Allow only previous connection
- Respond to any Active Scan request
- Disables all connection request
Sending Messages

- FlushTX resets pointer of TX buffer in stack
- WriteData fills one byte of data to TX buffer in stack

Transmission in 3 ways
- Broadcast
- Unicast by connection table index
- Unicast by address

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

```c
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;
```
Special Functionality

- **Noise Detection Scan** – Detects noise level in environment
- **Transceiver Power State** – Allows to be powered by battery, puts to sleep and wakes it up periodically
- **Frequency Agility** – Capability to hop channels during operation to bypass persistent noise on a channel.
Network Configurations

Two Functional Device Types:
- Full Function Device (FFD)
- Reduced Function Device (RFD)

<table>
<thead>
<tr>
<th>MiWi™ PROTOCOL DEVICE TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Type</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>PAN Coordinator</td>
</tr>
<tr>
<td>Coordinator</td>
</tr>
<tr>
<td>End Device</td>
</tr>
</tbody>
</table>

The PAN coordinator is the most essential:
- It starts the network
- It selects the channel and the PAN ID of the network.
- All other devices joining onto the PAN have to its instructions
Network Configurations:  
Star Network

Star Network:  
- Consists of one PAN coordinator node and one or more end devices.  
- All end devices communicate only with the PAN coordinator
Network Configurations: Cluster Tree Network

Cluster Tree Network:
- One PAN coordinator
- Multiple coordinators are allowed to join on to the network
Network Configurations: Mesh Network

Mesh Network:
• Full Function Device (FFDs) can route messages directly to other FFDs
• Message latency can be reduced and reliability is increased.
In Conclusion

- MiWi Wireless Protocol
  - Protocol Stack
    - MiMac
    - MiApp
  
- Network Configurations
  - Star
  - Cluster Tree
  - Mesh
Questions