

# ZMN2405/HP ZigBee™ Module Developer's Kit



User's Manual



## Table of Contents

|       |   |    |
|-------|---|----|
| 1.    | Introduction .....  | 5  |
| 2.    | ZigBee Networking.....                                      | 6  |
| 2.1   | Forming a Network.....                                      | 7  |
| 2.2   | Sleeping End Devices .....                                  | 8  |
| 2.3   | ZigBee Addressing.....                                      | 8  |
| 2.4   | Discovery .....   | 9  |
| 2.5   | Network Topology and Size .....                             | 9  |
| 2.6   | Static Network Addresses and Link Announcements .....       | 11 |
| 3.    | Getting Started .....                                       | 13 |
| 3.1   | Installing ZBDemo.....                                      | 13 |
| 3.2   | Installing USB Drivers .....                                | 13 |
| 3.3   | Running ZBDemo.....   | 13 |
| 3.4   | Communicating without ZBDemo.....                           | 23 |
| 4.    | The Development Board .....                                 | 24 |
| 5.    | ZBDemo .....  | 27 |
| 5.1   | Discover Radios .....                                       | 30 |
| 5.2   | Refresh Delay .....   | 30 |
| 5.3   | Switches.....   | 30 |
| 5.4   | GPIO LEDs .....   | 31 |
| 5.5   | Thermistor, Potentiometer and Analog to Digital Inputs..... | 31 |
| 5.6   | Digital to Analog Outputs .....                             | 31 |
| 5.7   | Children's Network List .....                               | 31 |
| 5.8   | Show ZigBee Tree .....                                      | 32 |
| 5.9   | Config.....   | 32 |
| 5.9.1 | Config Tab.....   | 33 |
| 5.9.2 | Config Enable Options .....                                 | 34 |
| 5.9.3 | Module I/O Tab.....   | 35 |
| 5.9.4 | Network Tab .....   | 36 |
| 5.9.5 | RF Tab .....  | 37 |
| 5.9.6 | End Device Tab.....   | 38 |
| 5.10  | Serial Comm .....   | 39 |
| 6.    | Programming the Module Firmware .....                       | 41 |
| 6.1   | In Circuit Module Programming.....                          | 43 |
| 7.    | Module & Development Board Hardware Specifications .....    | 45 |
| 7.1   | Module Pin Descriptions .....                               | 45 |
| 7.2   | Electrical Specifications .....                             | 46 |
| 7.3   | CC2430.....   | 47 |
| 7.4   | Schematics.....   | 48 |
| 8.    | Cirronet Standard Module (CSM) Profile API.....             | 51 |

|        |                                    |    |
|--------|------------------------------------|----|
| 8.1    | Module I/O Cluster (ID 0x01)       | 52 |
| 8.2    | Configuration Cluster (ID 0x02)    | 53 |
| 8.3    | Reset Cluster (ID 0x03)            | 57 |
| 8.4    | Network Cluster (ID 0x07)          | 58 |
| 8.5    | RF Cluster (ID 0x08)               | 59 |
| 8.6    | Security Cluster (ID 0x09)         | 60 |
| 8.7    | Serial Interface                   | 61 |
| 8.8    | Serial Protocol                    | 62 |
| 8.8.1  | Set Field                          | 65 |
| 8.8.2  | Set Reply                          | 65 |
| 8.8.3  | Get Field                          | 66 |
| 8.8.4  | Get Reply                          | 66 |
| 8.8.5  | Send String                        | 67 |
| 8.8.6  | Send String Reply                  | 67 |
| 8.8.7  | Receive String                     | 68 |
| 8.8.8  | Send SPI                           | 68 |
| 8.8.9  | Send SPI Reply                     | 69 |
| 8.8.10 | Get IEEE Address                   | 70 |
| 8.8.11 | Get IEEE Address Reply             | 71 |
| 8.8.12 | Get NWK Address                    | 72 |
| 8.8.13 | Get NWK Address Reply              | 73 |
| 8.8.14 | Discovery Request                  | 74 |
| 8.8.15 | Discovery Reply                    | 74 |
| 8.8.16 | Discovery End                      | 74 |
| 8.8.17 | Receive Field Event                | 75 |
| 8.8.18 | Link Announce                      | 75 |
| 8.8.19 | Device Registration                | 76 |
| 8.8.20 | Error                              | 76 |
| 8.9    | CSM Sleep Modes                    | 77 |
| 8.9.1  | Timer Sleep Mode                   | 77 |
| 8.9.2  | Interrupt Sleep Mode               | 77 |
| 8.9.3  | Configuring Sleep Mode             | 78 |
| 8.9.4  | Timer Sleep Example                | 78 |
| 8.9.5  | Interrupt Sleep Example            | 78 |
| 8.10   | Network Discovery                  | 79 |
| 8.10.1 | Device Registration Packets        | 79 |
| 8.10.2 | Discovery Command                  | 79 |
| 8.10.3 | Hierarchical Discovery             | 80 |
| 8.11   | Sample Packets                     | 81 |
| 8.11.1 | RF Channel List Example            | 81 |
| 8.11.2 | GP I/O Direction Example           | 83 |
| 8.11.3 | Microcontroller Reset              | 85 |
| 8.11.4 | UART Port/Send ASCII Data Example: | 86 |
| 8.11.5 | ADC Z Example: - Get Field         | 88 |
| 8.11.6 | ADC Z Example: - Get Reply         | 90 |
| 8.11.7 | Discovery Request                  | 91 |

|         |  |     |
|---------|--|-----|
| 8.11.8  | Discovery Reply .....                                | 92  |
| 8.11.9  | Discovery End .....                                  | 93  |
| 8.11.10 | Get IEEE Address .....                               | 94  |
| 8.11.11 | Get IEEE Address Reply .....                         | 95  |
| 9.      | Custom Profiles .....                                | 97  |
| 10.     | Layout Guidelines for ZMN2405/ZMN2405HP Module ..... | 98  |
| 10.1    | Reflow Profile for ZigBee Adapter Panel .....        | 99  |
| 11.     | WARRANTY .....                                       | 100 |

## 1. Introduction

Congratulations on your purchase of Cirronet's ZigBee Developer's Kit. The ZMN2405/HP developer's kit contains everything needed to get a two node ZigBee network up and running. Refer to Figure 1 for the contents of your kit.



**Figure 1. Developer's Kit Contents**

The modules in this kit have been designed by Cirronet Engineers and embody the accumulated expertise gained over 20 years of RF design and implementation experience. Experience gained designing and implementing some of the most difficult RF networks around including nuclear power plant monitoring, medical patient vital signs telemetry and industrial asset tracking. This manual covers both the ZMN2405DK and the ZMN2405HPDK. The only difference between the two kits is the output power of the modules included. The ZMN2405 module provides 1mW of RF power while the ZMN2405HP provides 65mW of RF power (when coupled with a 2dB dipole antenna, the ZMN2405HP provides 100mW EIRP).

The combination of Cirronet expertise and ZigBee technology creates a low-cost, versatile, yet reliable wireless data platform for applications needing periodic data, mesh networking and redundancy. With 100mW transmit power (using supplied dipole antennas, 250mW using the supplied patch antennas), the Cirronet ZMN2405HP module has the power necessary to get data through in harsh and noisy RF environments. The ZMN2405 provides 2mW with the dipole antenna and 4mW with the patch antenna, is ideal when battery current is limited.

There are various flavors of ZigBee devices that this module must support, the Coordinator, the Router and the End Device. How these devices work and what their functions are in a Zigbee network are explained in detail in Section 2, Zigbee Networking. The development board can also be configured as an End Device by downloading the End Device code load provided as part of the kit. Programming the module is explained in Section 6.

## 2. ZigBee Networking

ZigBee is a mesh networking and security stack that sits on top of an 802.15.4 MAC layer radio. 802.15.4 specifies the frequency bands, the number of channels, the spreading technique and the modulation method. ZigBee controls how data is routed between 802.15.4 physical layer radios, adding mesh and encryption capability along the way.

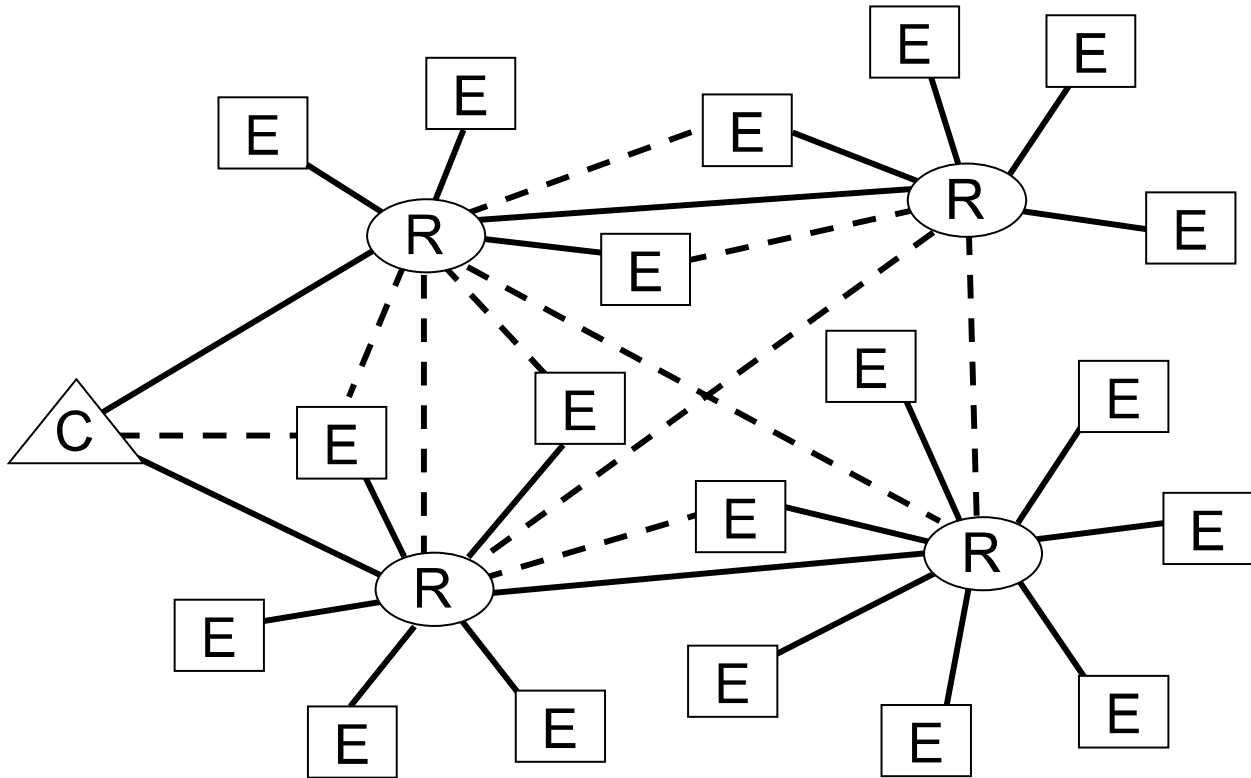
There are three types of devices specified in ZigBee: a Coordinator; a Router and an End Device. The Coordinator and Router are sometimes referred to as Full Function Devices or FFDs. The End Device is sometimes referred to as a Reduced Function Device or RFD. Since an FFD can be a Coordinator or a Router, this manual will use the Coordinator, Router and End Device names.

The Coordinator is responsible for setting the channel for the network to use, making its presence known to Routers and End Devices, assigning network addresses to Routers and End Devices and keeping the routing tables for the network that are necessary to route data from one ZigBee device to another in the same ZigBee network. Each network must have one and only one Coordinator. Without a Coordinator, a network cannot form. Typically, although it is not a requirement, the Coordinator will function as the gateway or takeout point for applications where data from the ZigBee network will be sent off the network and will be received from devices off the network.

The Router, as its name implies, is responsible for routing data from other Routers or End Devices to the Coordinator or to other Routers closer to the Coordinator. The Router can also be a data input device, either serially or through the I/O pins of the module.

The End Device can only communicate with the Coordinator or a Router. An End Device cannot communicate directly with another End Device. Communication between one End Device and another End Device must go through the Coordinator or a Router and may go through one or more Routers.

Figure 2 depicts a typical ZigBee network with two levels of Routers. The box labeled C represents the Coordinator, the boxes labeled R represent Routers and the boxes labeled E represent End Devices. As can be seen from the diagram, each End Device has multiple potential communication paths. The most likely path is a solid line with backup paths indicated with dashed lines. From this Figure it is seen how End Devices only communicate with Routers and the Coordinator while Routers can communicate with End Devices, other Routers or the Coordinator. The Coordinator can communicate with Routers and End Devices.



**Figure 2. Typical ZigBee Network**

## 2.1 Forming a Network

The Coordinator is typically the first element powered up. The Coordinator will listen on the first channel of the set of channels that has been specified for it to use in forming a network. If the Coordinator detects RF energy on that channel, it will move to the next channel in the channel set until it finds a clear channel. If a clear channel cannot be found, the Coordinator will indicate an error.

If a PAN ID has been specified, the Routers and End Devices will look for a Coordinator with the specified PAN ID. If a Coordinator is not found on a channel, or one is found, but has the wrong PAN ID, the ZigBee device will go to the next channel on its channel list until a satisfactory Coordinator is found. The Coordinator will assign each device that identifies itself to the Coordinator a 16-bit network address. This 16-bit network address is used to route data within the network. Each ZigBee device has a unique 8-byte MAC address just as in an Ethernet network. The 16-bit network address can and will change every time the Coordinator is power cycled unless the Static Network mode is enabled. (Refer to Section 2.6 for details).

The network is now formed. If the Routers and End Devices have been configured to transmit data on their own, they will begin doing as they have been told. If the Coordinator is to be used as a gateway too, it is typically necessary to let the application communicating with the gateway to know what devices are on the network. ZigBee provides a discovery command which returns from the Coordinator/gateway, the 16-bit network addresses of all devices associated in the network. The application then must request the 8-byte MAC address and optionally the "Friendly Name" of each device associated with the network. Once the devices associated with the

network have been discovered and the MAC addresses obtained, the application can address data and commands to individual devices on the network.

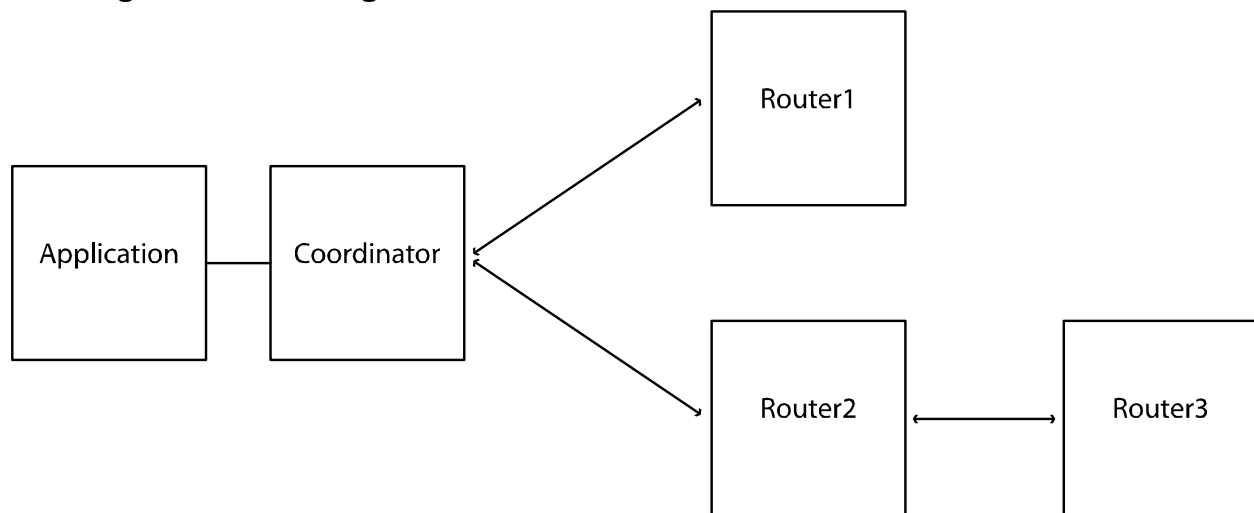
## 2.2 Sleeping End Devices

ZigBee envisions that Routers and Coordinators will always be powered and never put to sleep as they must always be awake since they do not know when another device may attempt to communicate. End devices on the other hand, are expected to send data for a brief period of time and then go to sleep for the majority of the time. When the End Device is asleep, any data addressed to it will be held by the Router (or Coordinator) with which it is associated. When the End Device wakes up, it will send a request for the Router (or Coordinator) to send any data it may be holding. End Devices do not have to go to sleep. They can be configured as mains powered and always be awake.

ZigBee provides that sleeping End Devices will do two things: 1) wake up periodically and see if their parent device is holding data for them; and 2) wake up periodically and perform some operation. It is possible to have an application poll sleeping End Devices. However, since the End Device will not respond until it wakes, the amount of time the application must wait to receive a response will be related to how often the End Device awakens.

The rest of the details of sleeping End Device modes are dependent on the application profile used. Details of the sleep modes implemented in Cirronet's CSM profile are provided in Section 8.9 of this manual.

## 2.3 ZigBee Addressing



The ZigBee standard provides for two means of addressing ZigBee devices, whether they are Coordinators, Routers or End Devices: MAC Addresses and Network Addresses. MAC addresses are just like MAC addresses in Ethernet networks. The MAC address is 8 bytes long, is unique to each device and permanently stays with the device. Network addresses are 2 bytes long and are assigned to ZigBee devices as they associate with the network. As such, the network address is not known before the network forms and can change when a ZigBee device re-associates with the network.

Much like an Ethernet device, Cirronet's ZigBee modules have their MAC addresses programmed into them at the factory as part of the manufacturing process. Cirronet's prefix of



00:30:66 is used in the MAC address and thus can be used to identify Cirronet products. While the MAC address uniquely identifies each ZigBee device - regardless of manufacturer or position in the network - it contains no routing information and thus can be used only to communicate with devices that are directly connected to a ZigBee device. That is, if a Router wants to talk to an End Device or Router to which it has a direct connection (the data does not have to be routed through another device), it can use the MAC address. However, if it wants to talk to a ZigBee device but needs to have the data routed through another ZigBee device, it must use the network address.

The network address is assigned to ZigBee devices as they associate with the network. The Coordinator always has network address 0x0000. The network addresses are assigned in such a way that the address contains routing information. As a 2-byte value, the network address can handle 65,535 potential devices in a single ZigBee network. For a ZigBee device to send data to a device on the network which is not directly connected to it, the network address must be used. Based on the address, the ZigBee Routers can determine the next ZigBee device to send the data to until it reaches the intended device.

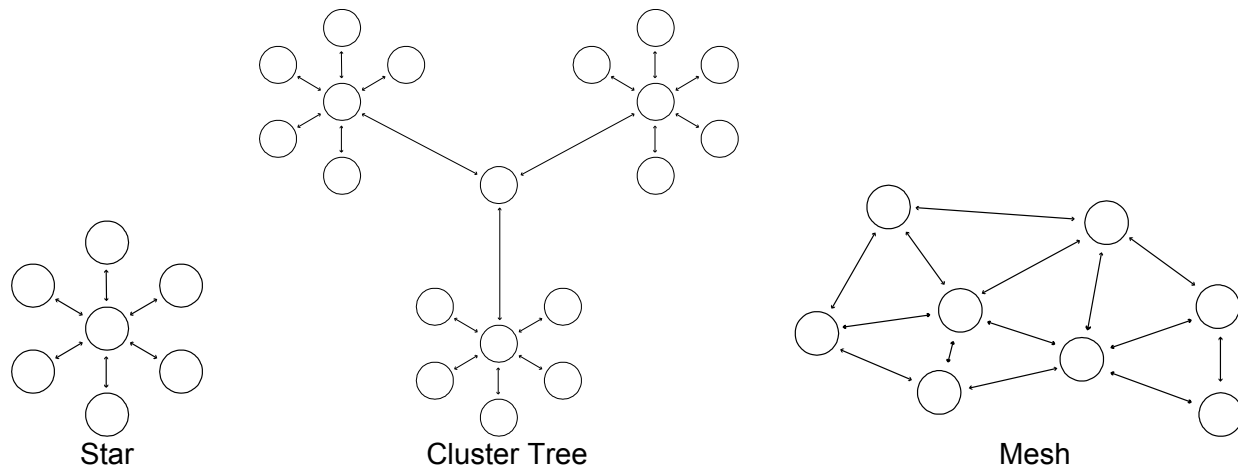
If an application needs to send data to specific devices on a network, the application must maintain a table that links device-specific MAC addresses with their assigned Network addresses. The application also must be aware that the network address of a specific device can change due to power cycling, loss of connection, etc. Network and MAC addresses can be determined through the Discovery and Get IEEE Address commands. Refer to the next section on Discovery and Section 8 for details on these commands.

## 2.4 Discovery

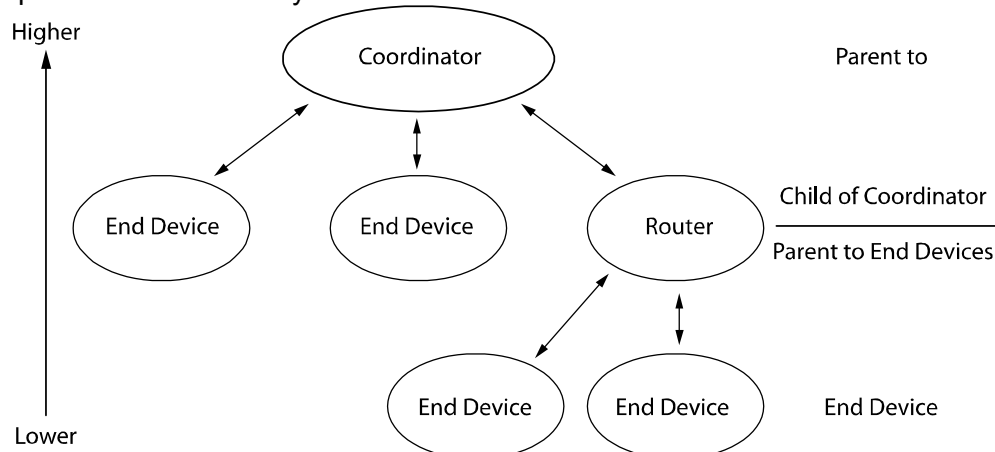
While it is possible for ZigBee devices to only communicate among other devices on the ZigBee network, typically it is a requirement that the devices on the network be known to either applications or other devices that are not part of the ZigBee network. ZigBee supports this need through a Discovery process. The Discovery process reports back all of the ZigBee devices on the network that have joined the network. All devices respond whether directly connected or not. In a network of several layers of depth, it will take a series of steps to discover all of the devices on the network. It is not necessary for sleeping End Devices to be awake to learn of their presence in the network. The parent device of the sleeping End Device will report its presence.

## 2.5 Network Topology and Size

ZigBee can support three primary network topologies: Star, Cluster Tree and Mesh. Depending on the configuration, the number of nodes the network can support will change. ZigBee uses three parameters that effectively control the network topology and the number of potential nodes. These three parameters are Max Number of Children, Max Number of Routers and Max Network Depth. These parameters are determined by the stack profile being used (this is different than the application profile) and cannot be configured after compilation of the code.



**Parents and Children.** In ZigBee parlance, devices higher and lower in the network hierarchy are referred to as Parents and Children respectively. The Coordinator in a network is the parent to all devices directly associated with the Coordinator, regardless of whether they are Routers or End Devices. Routers which are children to the Coordinator are parents to devices that directly associate with them, also regardless of whether they are Routers or End Devices. End Devices cannot be parents and are always children of either the Coordinator or a Router.



**Max Number of Children (MNC).** This parameter specifies the maximum number of devices for which a ZigBee device can act as a parent or upstream connection. For example, in a star network, MNC will determine the maximum number of devices that can associate with the Coordinator. For a cluster tree or mesh network topology, the MNC specifies the number of children the Coordinator and Routers can each have associated with them. In the only ZigBee approved stack profile (Home Control Lighting) MNC is twenty. This means that a star network can have 21 nodes in it, 1 Coordinator and 20 children. For cluster tree and mesh topologies MNC is not sufficient to determine the maximum number of network devices.

**Max Number of Routers (MNR).** This parameter specifies how many out of the MNC devices can be Routers. In the Home Control Lighting stack profile, MNR is set to 6. This means that the Coordinator can have 6 Routers directly associated with it and each Router can have 6 Routers associated directly with them. The other 14 devices directly associated with the Coordinator and Routers must be End Devices. If there are fewer than 6 routers associated with the Coordinator or a Router, the maximum number of end devices that can associate with either the Coordinator or a Router is still 14.

**Max Network Depth (MND).** This parameter specifies how many levels of Routers may be present in a ZigBee network. For a star network, while there can be Routers associated with the Coordinator, the Routers cannot have any children. For cluster tree and mesh networks multiple levels of Routers can be implemented. The Home Control Lighting stack profile set MND to 5. This means a network could be constructed with the Coordinator at the top and 5 levels of Routers beneath the Coordinator.

From these three parameters, the number of nodes that can be supported in a given network configuration can be computed. In the simple case of the star network, it is simply the Coordinator plus **MNC**. For cluster tree and mesh networks, the calculations are a little more complicated. The Coordinator can support 20 devices of which 6 can be Routers and 14 End Devices. If we assume that there are 2 Routers, and that both Routers have a full complement of 14 End Devices - then this network can support the Coordinator, two Routers and 42 End Devices. In practice, it is hard to know exactly how a network will form. While devices will attempt to associate with the highest level of the network, if they are unable to establish a connection, they will associate with a lower level of the network. So in the simple example above, if some of the 14 End Devices that are assumed to be directly associated with the Coordinator cannot establish a connection with the Coordinator but can establish a connection with one of the two Routers, then some number less than 42 End Devices, 2 Routers and the Coordinator will be supported by this network.

Because the Home Control Lighting profile is the only public stack profile, Cirronet has implemented our ZigBee modules using that stack profile and thus are constrained by the limits of 20 children per device, 6 Routers per device and 5 levels of Routers. Please contact Cirronet Tech Support if your application requires different parameter values.

## 2.6 Static Network Addresses and Link Announcements

When a ZigBee network forms, nodes are assigned their 2-byte network addresses according to how the network forms. For example, the first router that associates with the coordinator is assigned the network address `0x0001`. The second router to associate is assigned the network address `0x143E`. If the network is powered down and then powered up again, if the router originally assigned network address `0x14FE` associates with the coordinator before the router originally assigned network address `0x0001`, it will be assigned network address `0x0001`. Applications that send data over the ZigBee network must keep track of the network addresses assigned to the nodes on the network. If the network loses power, the application must detect that and rediscover the network. This can be a cumbersome process for the application especially during application development or system testing.

To alleviate the problems caused by these situations, Cirronet has implemented two features: **Static Network Addresses** and **Link Announcements**. Each is described in detail below.

### Static Network Addresses

When enabled through a configuration parameter, Static Network Addresses causes the ZigBee modules to save their network address in non-volatile memory. If power is subsequently lost to the network or just the node, the module will request its original network address when power is restored. If the entire network lost power, this will cause the network to be formed with each ZigBee device having its original network address. If the original parent is no longer available,

the ZigBee device will stop trying to re-associate with its original network address and re-associate with the network as if it had never belonged.

Static Network Addresses cannot be used in a mobile environment as it is expected and desired that nodes will associate and re-associate using different parents. In the event when Static Network Addresses are being used, if a parent is no longer available, the node device will associate and receive a new network address.

### **Link Announcements**

To prevent the application from periodically rediscovering the network again to learn the network addresses - most of which will not have changed - Cirronet has implemented Link Announcements. Link Announcements are always enabled. Any time a node associates with a network whether it is the first time or the fifth time, and regardless of with which parent device it associates, a Link Announcement message is sent to the coordinator. The Link Announcement message includes the network address of the node joining the network. The GET IEEE command can be used to determine the MAC address of the joining device.

## 3. Getting Started

Once you have identified all the components of your kit, connect either the 6dBi patch antenna directly to the RF connector on the development boards or the RF cable and 2dBi dipole antenna. It is not necessary to use the same antenna on both boards.

### 3.1 Installing ZBDemo

The next step is to copy the ZBDemo program onto a computer. The ZBDemo program is on the Software and Manuals CD in the Software directory included with the kit. Copy the file zbdemo.exe onto the hard drive of the computer and remember into which directory it was copied.

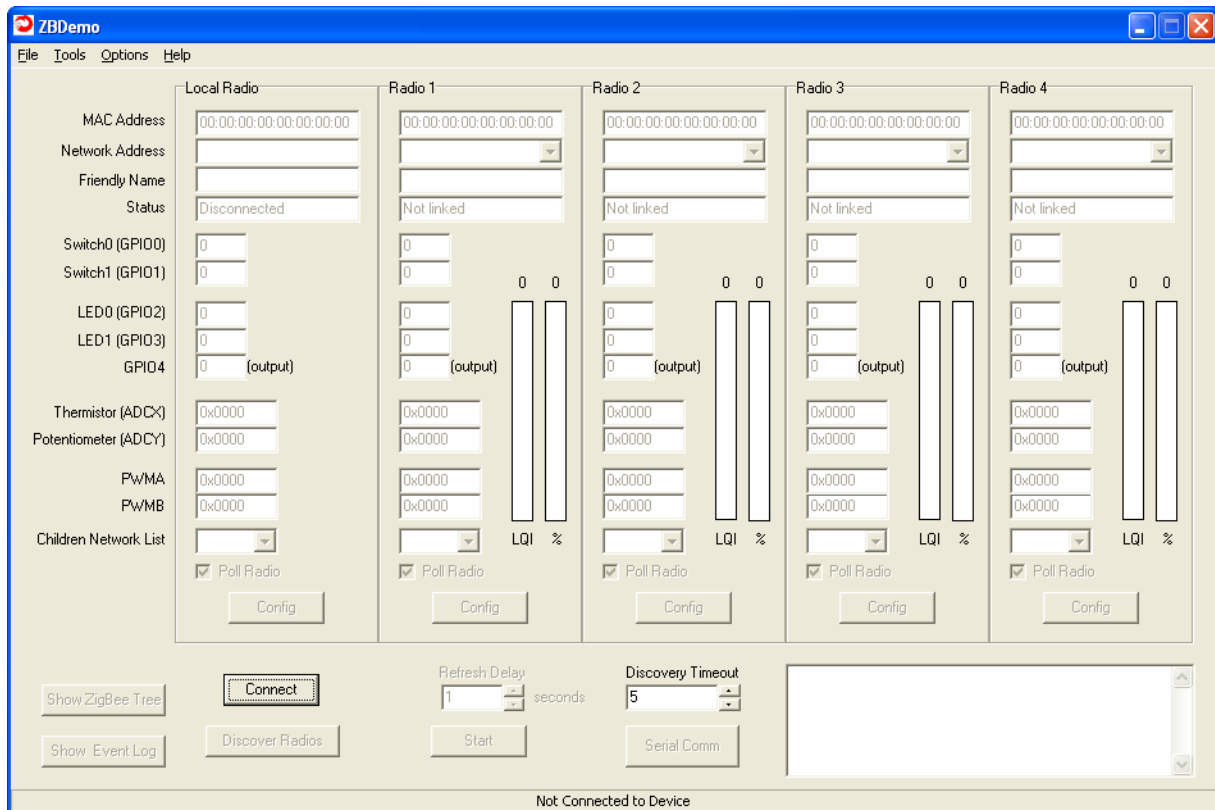
### 3.2 Installing USB Drivers

If a USB port is to be used to communicate with the development board, the drivers for the USB adapters must be installed on the PC. The USB drivers are on the Documentation and Software CD under the Drivers folder. When the computer detects the presence of the hardware, the “Add New Hardware Wizard” box will appear. Click on “Next” and follow the prompts. Refer to Section 5 for step-by-step instructions.

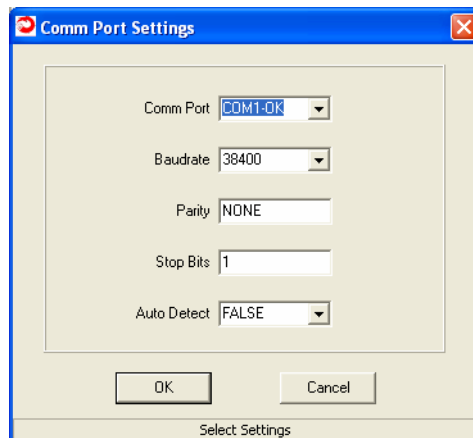
### 3.3 Running ZBDemo

One development board is setup as the Coordinator and the other development board is setup as a Router. Connect the Coordinator to either the USB or RS-232 serial port of the computer. Use either the wall-mount power supply or a 9V alkaline battery. It is not necessary to use the same power type for each radio. Apply power to the Coordinator first and verify the Power LED is on.

When the Coordinator has found that the channel is clear, the Link LED on the Coordinator will glow green. Once the Coordinator has turned on the Link LED, the Router can attempt to associate with the Coordinator. Once it has associated with the Coordinator, the Link LED will glow green. Wait until the Link LEDs on all radios are glowing green before continuing. Start the ZBDemo program by double-clicking the ZBDemo icon on the computers’ desktop.



Click the Connect button on the ZBDemo screen. A window will open showing the communications parameters to be used to communicate with the Coordinator. Select a serial port from the drop down menu. Selections that are available will be marked as “OK”, those unavailable will be marked as “N/A”. 38,400 is the Default baud rate. This window changes the PC Comm Port parameters but not the development board.

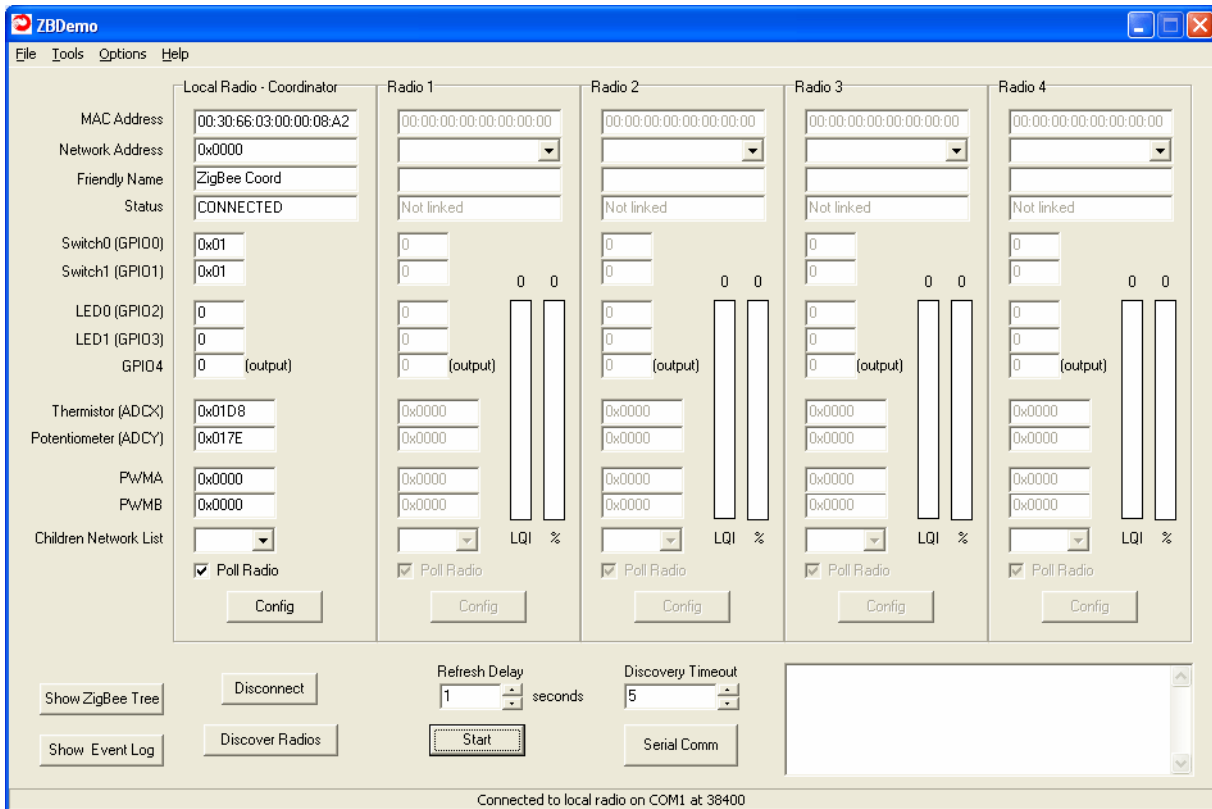


The Comm Port Settings dialog is pre-loaded with default settings. In most instances, simply clicking OK will allow ZBDemo to find the radio and load the main program window. If, however, the default settings are incorrect, the radio will not be found. This screen allows you to connect to the radio using two methods.

The first is to use the drop down menus to change the default settings for Comm Port, Baudrate, Parity and Stop Bits to the correct values. Obviously, this assumes you know which settings the radio is using. If, however, those parameters may have been changed and are not known, a second method is available.

The Auto Detect function works this way. If set to FALSE, once OK is selected, the program uses the default settings to try to communicate with the radio. If set to TRUE, the program will begin a systematic process beginning with the first valid port (COM 1 in most cases) then will cycle through each baud rate, then each parity setting, then each stop bit setting finally changing to the next available COM port and repeating the process until a radio is found.

Once the radio has been found, click OK and after a few seconds, information about the ZigBee module in the Coordinator will be displayed in the Local Radio column along with its 8-byte MAC address. The Status condition of CONNECTED indicates the computer has detected the development board as shown below.

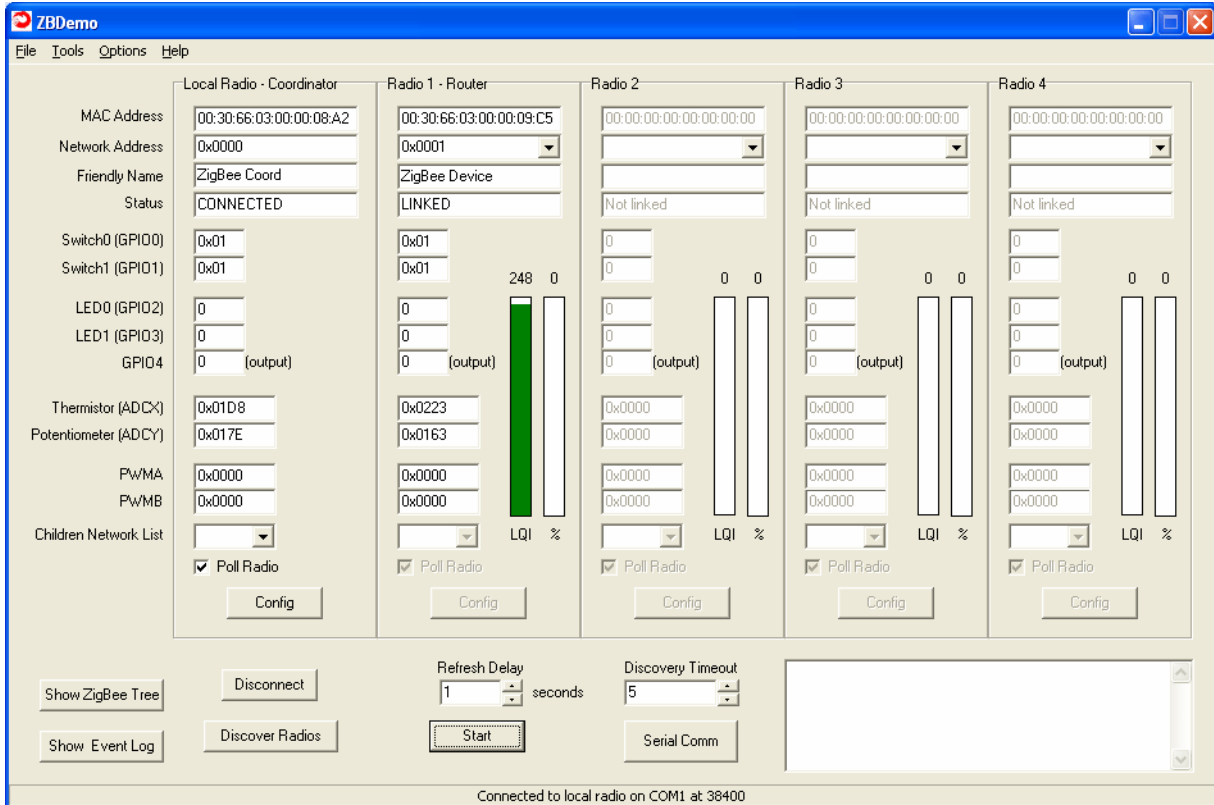


Click on the Discover Radios button. This will cause the Coordinator to request information about the Router development board.

### Router

This information will be displayed in the Radio 1 column of the ZBDemo program and the type listed as “Router” as shown below.

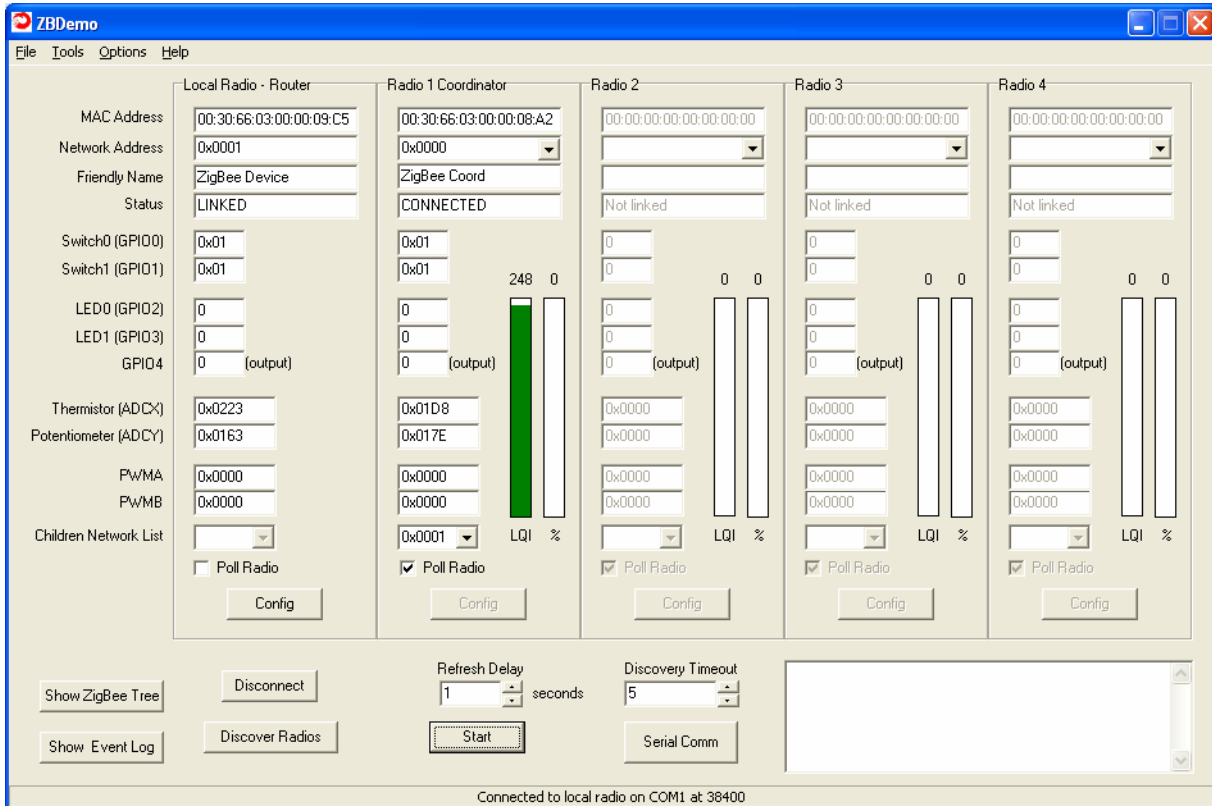
**Note: This assumes the development board is configured as a Router. If the development board has been reprogrammed as an End Device, refer to the next section.**



The Status for Radio 1 will be LINKED and the LQI (Link Quality Indicator) will show the relative strength of the link. The columns for Radios 2, 3 and 4 will be grayed out with a Status of Not Linked. Repeat the above steps on the other computer for the development board.



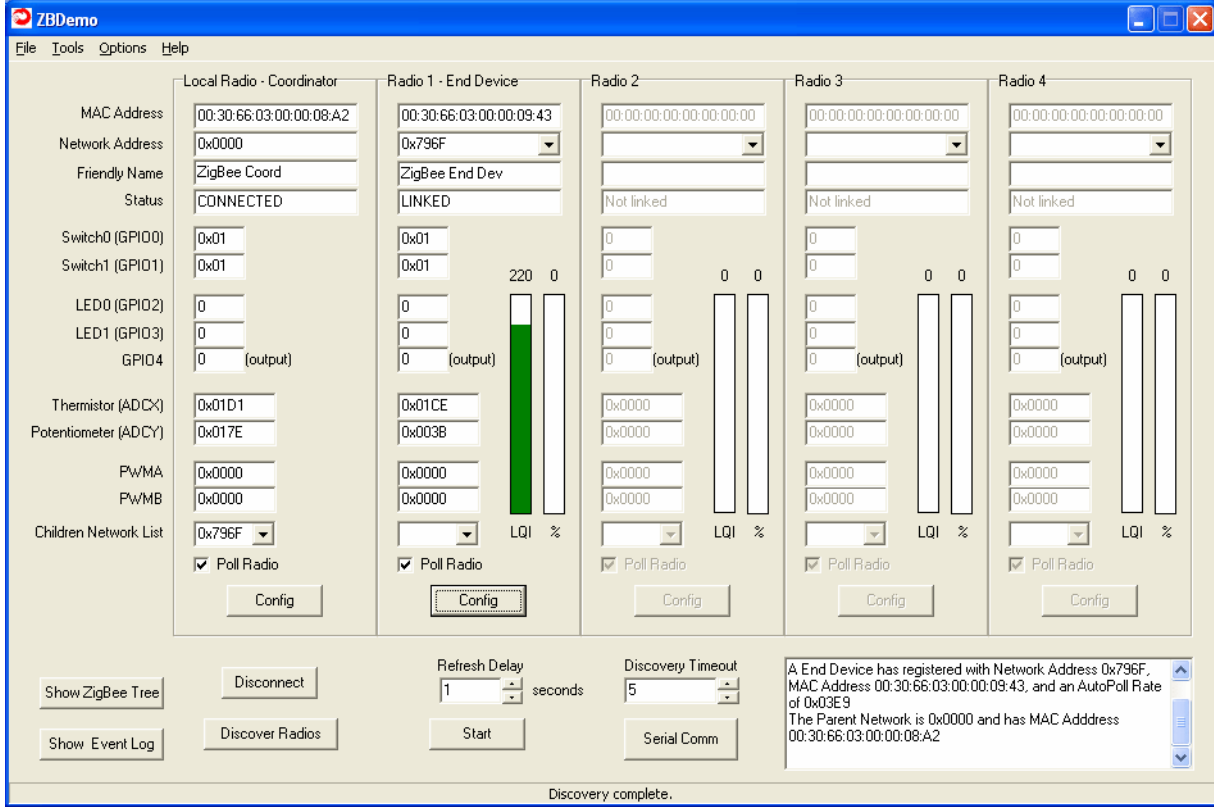
After completing the steps, the Router information will be displayed in the Local Radio column and the Coordinator information will be displayed in the Radio 1 column as shown below.



There are Poll Radio checkboxes at the bottom of each Radio column that enable periodic polling of the radio's parameters. In order to see changes made to the state-change devices on the development board, turn on continuous polling by clicking the Start button under the Refresh Delay field. A setting of 0 seconds in the Refresh Delay field will continuously poll the Radios with the Poll Radio box checked. Longer Refresh Delay settings can be set using the up and down arrows to the right of the Refresh Delay field, or the number can be highlighted and a value entered. Once the Start button is pressed, it changes to a Stop button. Clicking on the Stop button will end the polling process.

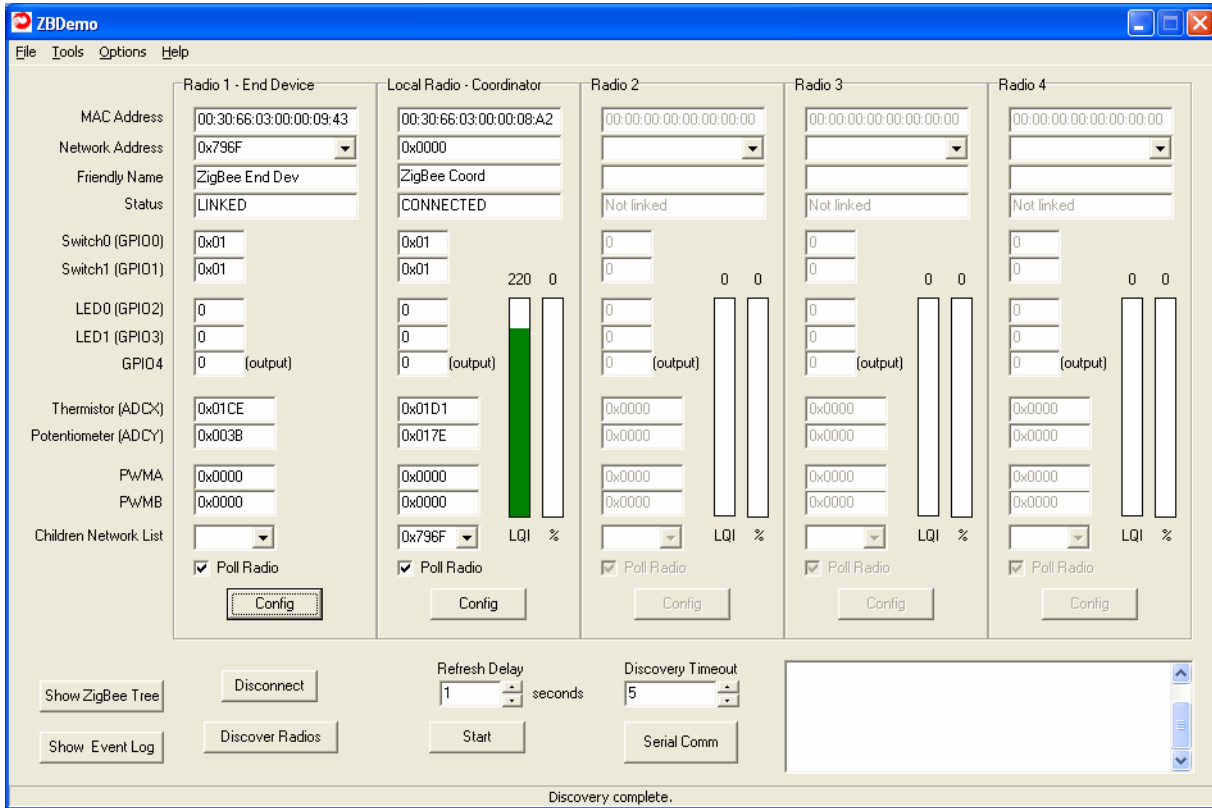
### End Device

If you have configured the development board as an End Device, this information will be displayed in the Radio 1 column of the ZBDemo program and the type listed as “End Device” as shown below.



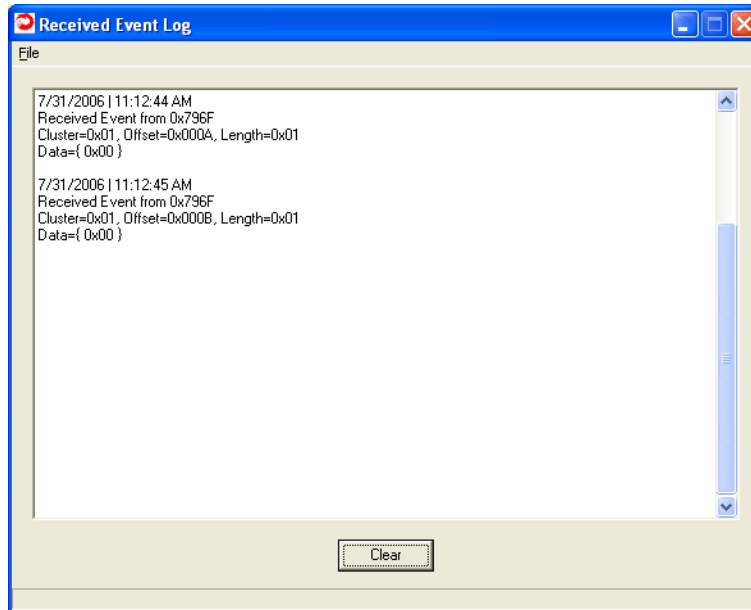
The Status for Radio 1 will be LINKED and the LQI (Link Quality Indicator) will show the relative strength of the link. The columns for Radios 2, 3 and 4 will be grayed out with a Status of Not Linked. Repeat the above steps on the other computer for the End Device.

After completing the steps, the End Device information will be displayed in the Local Radio column and the Coordinator information will be displayed in the Radio 1 column as shown below.



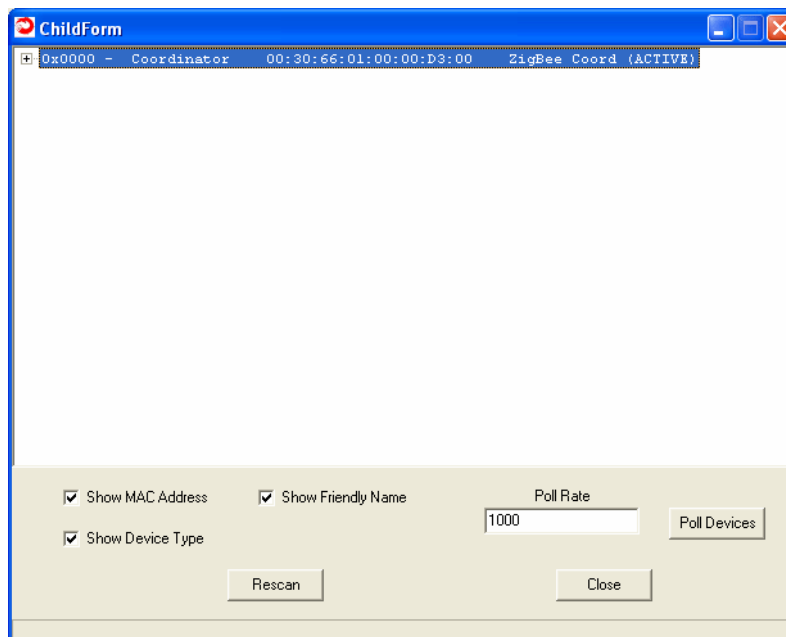
There are Poll Radio checkboxes at the bottom of each Radio column that enable periodic polling of the radio's parameters. In order to see changes made to the state-change devices on the board, turn on continuous polling by clicking the Start button under the Refresh Delay field. A setting of 0 seconds in the Refresh Delay field will continuously poll the Radios with the Poll Radio box checked. Longer Refresh Delay settings can be set using the up and down arrows to the right of the Refresh Delay field, or the number can be highlighted and a value entered. Once the Start button is pressed, it changes to a Stop button. Clicking on the Stop button will end the polling process.

In the lower left portion of the main ZBDemo window, there are two additional buttons, Show ZigBee Tree and Show Event Log. The Event Log is a chronological listing of the events that have occurred on the Zigbee network as shown below.



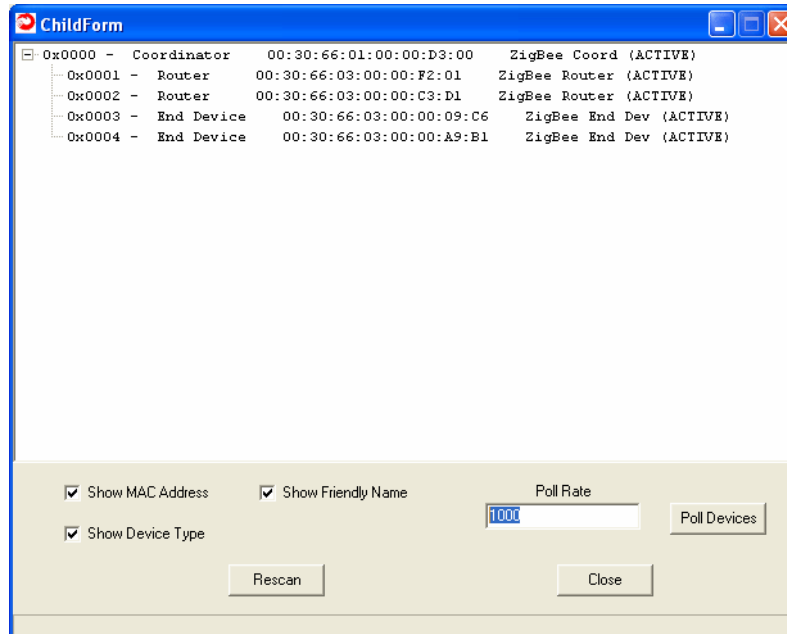
As you can see, the log shows the date and time of the event, along with the network address of device the event was received from, the Cluster, Offset and Length (in bytes) of the event and finally, the data received. The Clear button allows you to remove all recorded events from the window and have the logging start again.

The Zigbee Tree button opens the Childform window which is an expandable representation of the devices that make up the Zigbee network as shown below.



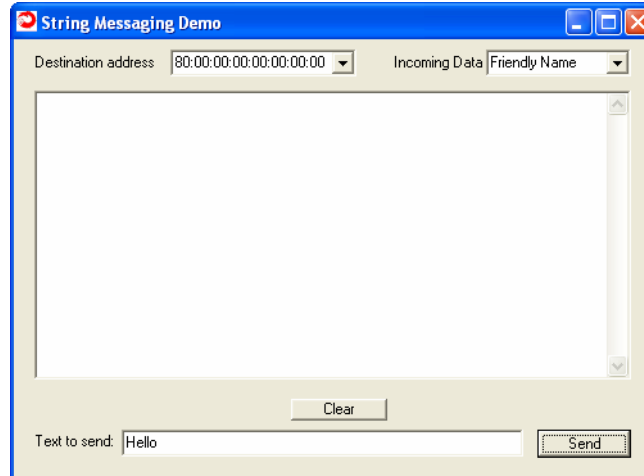
There are checkboxes at the bottom of the window (Show MAC Address, Show Friendly Name, Show Device Type) that allow you to choose the items listed for each device in the network. You can check all or not check any.

There is also a Poll Rate field that allows you to enter a polling time period, this number is in milliseconds, i.e. 1000 would equal 1 second. The Poll Devices button on this window can be used in conjunction with the main ZBDemo screen to automatically and sequentially poll ALL current network devices without having to individually select devices to load into the Radio 1 – 4 columns. This can be useful on networks where there are substantially more radios in the network than can be displayed in the four available Radio columns.



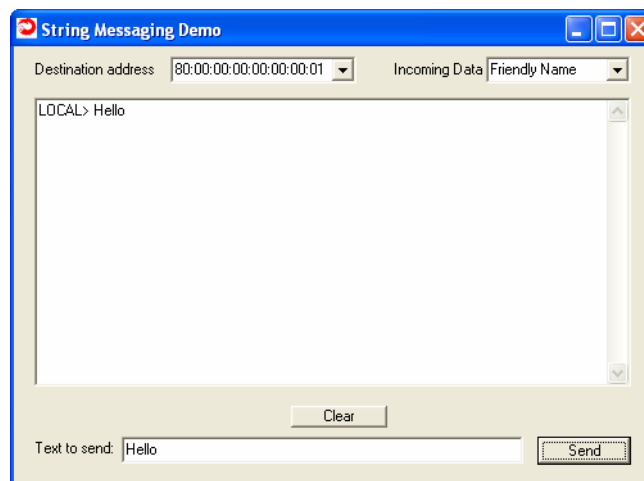
Notice there is a + sign to the left of the Coordinator line. Clicking on that sign will expand the tree so that all the network devices connected to the Coordinator will display as shown above. Double click on any line to open the configuration screens for that device. Clicking on the Rescan button will go out and scan the network for any changes, then display those changes in the Childform window.

If you want to send text messages back and forth between radios, a second serial port – either RS232 or USB – is needed on the computer or a second computer is needed. Two Zigbee USB devices cannot be attached to the same PC. This is due to the USB drivers. Click on the Serial Comm button in the lower right hand corner of the ZBDemo main screen and the String Messaging Demo box will open.

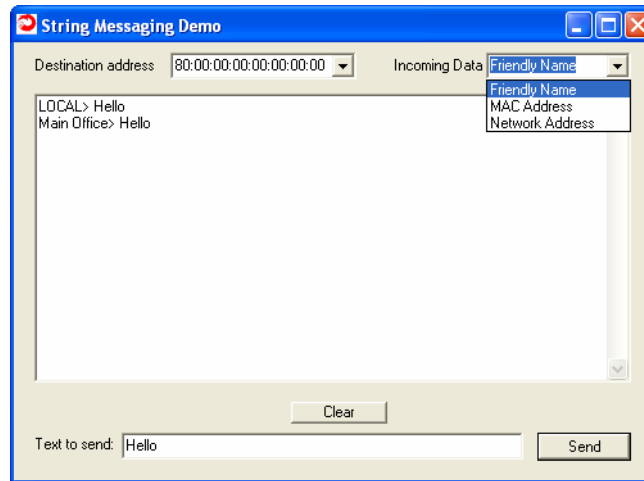


There are four fields/windows in this box: the Destination Address window top left of the box, the Incoming Data field top right of the box, the Messaging Display window in the middle of the box; and the Text to Send window at the bottom of the box. The Clear button allows you to clear the contents of the messaging display window.

To send a text string to the other radio, simply click on the Send button which will send the text entered in the Text to Send field to the other radio. Data sent by the local radio will appear in the messaging display window indicating what the Local radio transmitted as shown below.



Data received from the other radio will be displayed in the messaging window and labeled according to the Incoming Data field selection. Incoming Data can be labeled using the Friendly Name, the MAC Address or the Network Address as shown below.



The String Messaging Demo functionality works the same on the other computer connected to the Router board.

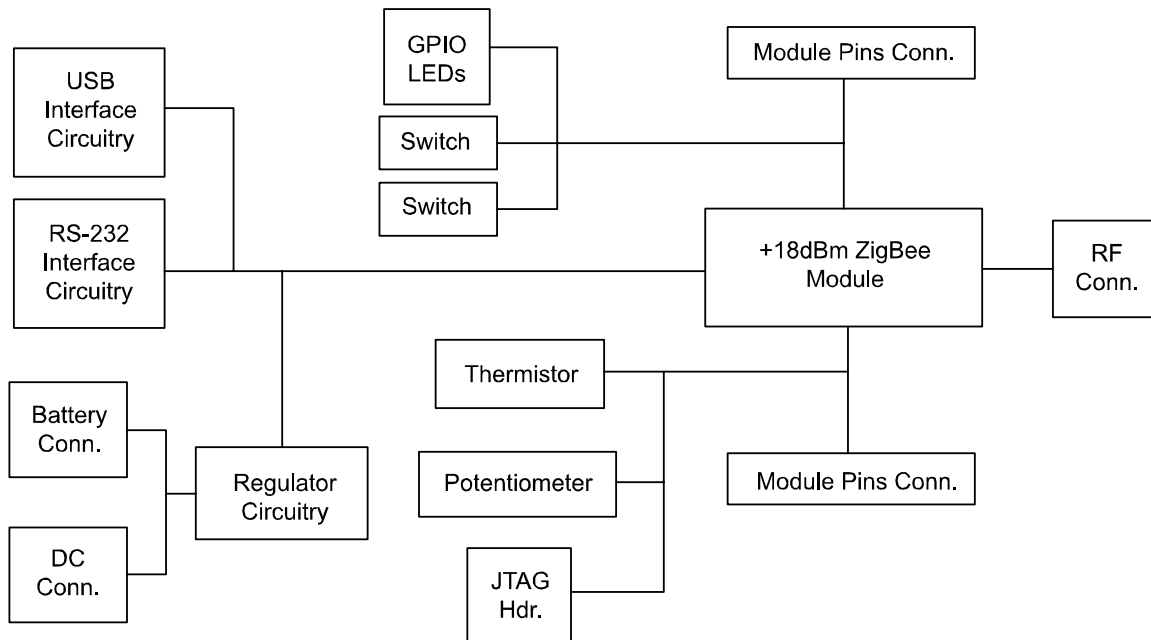
### 3.4 Communicating without ZBDemo

Packets are used to send and receive data thus unformatted data packets cannot be sent to the UART Port without first putting the module in Transparent Mode. Details on packet format and structure can be found in Section 8.8, *Serial Protocol*. For details on using Transparent Mode, see Section 8.2.

## 4. The Development Board

This section describes in detail all the features and functions of the development board that makes up the developer's kit. The development board as it comes from the factory is labeled as a Router on the bottom of the board. Additional Router or End Device development boards can be ordered from Cirronet.

A block diagram of the development board is shown below:



**Figure 3. Development Board Block Diagram**

The USB and RS-232 interface circuits convert standard level signals into 3.3V transmit and receive data signals. Only one input can be used at a time. If a USB connection is present, the RS-232 input will be electrically locked out. Note that there is no hardware flow control implemented on the RS-232 input. A standard RJ-11 connector is provided for the RS-232 connection.

The development board can run off either a 9Vdc power source or a 9V alkaline battery. The supplied wall-mount DC power supply connects to the DC power connector located next to the USB connector. The 9V alkaline battery connectors are located on the bottom of the board underneath the Power LED.

A JTAG header is provided for situations where custom profiles or module code is being developed and the module will be programmed without using the bootloader code resident in the module.

All of the ZMN2405HP module pins, except Reset, are brought out to connectors on the board. This facilitates connecting the module to other devices without having to integrate the module into another circuit. The pinouts of the connectors are listed below and are included in the board silk screen for JP1 and JP2.



| JP1           |          |            |
|---------------|----------|------------|
| Connector Pin | Signal   | Module Pin |
| 1             | Ground   | 2          |
| 2             | +3.3V    | 1          |
| 3             | PWMA     | 3          |
| 4             | PWMB     | 4          |
| 5             | GPIO0    | 5          |
| 6             | GPO11    | 6          |
| 7             | GPIO2    | 7          |
| 8             | GPIO3    | 8          |
| 9             | GPIO4    | 9          |
| 10            | GPIO5    | 10         |
| 11            | Link/TDO | 12         |
| 12            | Ground   | 2          |

| JP2           |          |            |
|---------------|----------|------------|
| Connector Pin | Signal   | Module Pin |
| 1             | Ground   | 2          |
| 2             | +5V      | NC         |
| 3             | SPI_MISO | 32         |
| 4             | SPI_MOSI | 31         |
| 5             | SPI_SCLK | 30         |
| 6             | SPI_EN   | 29         |
| 7             | ADCZ     | 27         |
| 8             | ADCY     | 26         |
| 9             | ADCX     | 25         |
| 10            | UTX      | 22         |
| 11            | URX      | 21         |
| 12            | Ground   | 2          |

Refer to the Section 7.1 for details on each module pin and its function.

The GPIO LEDs, Switches, Potentiometer and Thermistor are provided for demonstration purposes. These components can be read and written to using the ZBDemo program. The switches are momentary SPST switches and close to ground when pressed. The thermistor is used to vary an input voltage to the ADC X input with a maximum input voltage of 3.0 volts. Similarly, the potentiometer is connected in a resistor divider network to vary an input voltage to ADC Y input which can vary between ground and Vcc (~3.3V). Refer to the board schematics at the back of this manual for more details. To connect ADC X or ADC Y to off-board signals, disconnect the thermistor and potentiometer from the module inputs by removing JP3 for the potentiometer and JP4 for the thermistor.

A group of four status LEDs are provided to allow board activity to be monitored. The LED functions are provided in the table below.

| LED      | Function   |
|----------|--|
| Link     | On the Coordinator, illuminates when a clear channel has been detected and the Coordinator is ready for other devices to associate with it.<br>On Routers and End Devices, illuminates when they have associated with a Coordinator. |
| Activity | Indicates RF data activity.  |

The figure below indicates where the various components of the board are located:

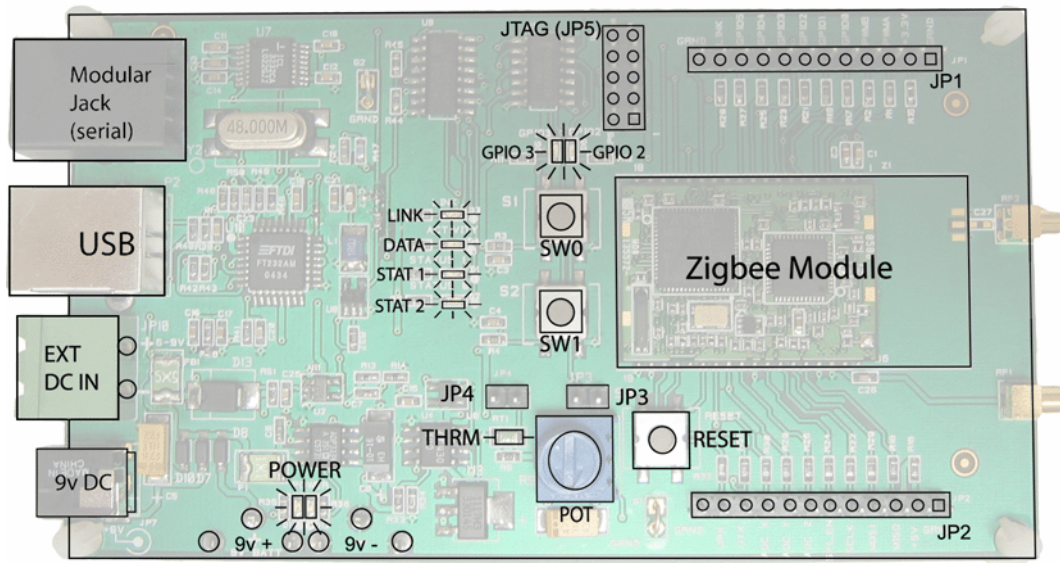
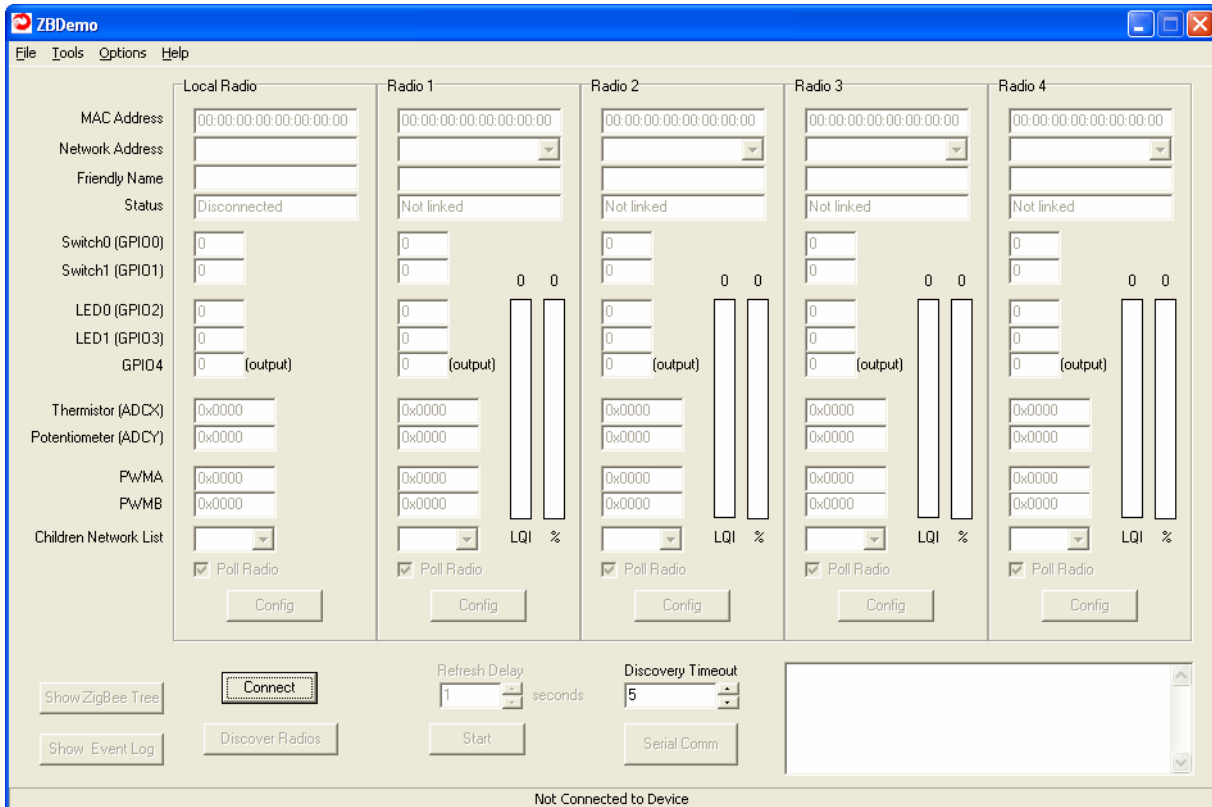


Figure 4. Development Board Component Locations

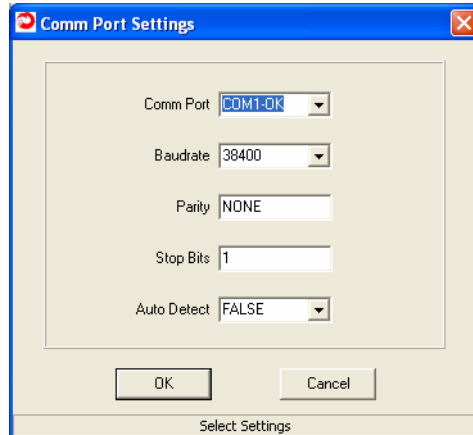
## 5. ZBDemo

This section details the features and functions of ZBDemo and provides some information on what is going on behind the GUI. ZBDemo does not need to be installed, only copied onto the computer to be connected to the developer board. The opening screen is shown below:



ZBDemo can display up to 5 devices including the Coordinator. Regardless of whether or not a radio is set up as a Coordinator or not, the information for a radio connected to the computer running ZBDemo will always be displayed in the Local Radio column. The Coordinator to which the local radio is associated will have its information displayed in one of the Radio columns. To identify the Coordinator, it is necessary to look for the MAC address on the label on the bottom of the PCB. Routers have their MAC address in the same location.

When the Connect button is clicked, the Comm Port Settings box opens.



**Note: Be sure to close all resident programs that use the Serial port such as any portable digital assistant syncing software (ex: Palm's HotSync) prior to running ZB Demo.**

The Comm Port Settings dialog is pre-loaded with default settings. In most instances, simply clicking OK will allow ZBDemo to find the radio and load the main program window. If, however, the default settings are incorrect, the radio will not be found. This screen allows you to connect to the radio using two methods.

The first is to use the drop down menus to change the default settings for Comm Port, Baudrate, Parity and Stop Bits to the correct values. Obviously, this assumes you know which settings the radio is using. The Comm Port drop down menu will display a list of Comm port selections from 1 to 16. Selections that are valid will be marked as OK. If an invalid selection is chosen, when the OK button is clicked, an error message will be displayed indicating that the selected Comm Port is not valid.

If a valid Comm port is selected, but the radio is not connected to that port, a baud rate error message will be displayed saying a radio could not be detected at the proper baudrate. If this message is received, select another Comm Port and try to connect again.

If, however, the default parameters have been changed and are not known, a second method, Auto Detect, is available.

The Auto Detect function works this way. If set to FALSE, once OK is selected, the program uses the default settings to try to communicate with the radio. If set to TRUE, the program will begin a systematic process beginning with the first valid port (COM 1 in most cases) then will cycle through each baud rate, then each parity setting, then each stop bit setting finally changing to the next available COM port and repeating the process until a radio is found.

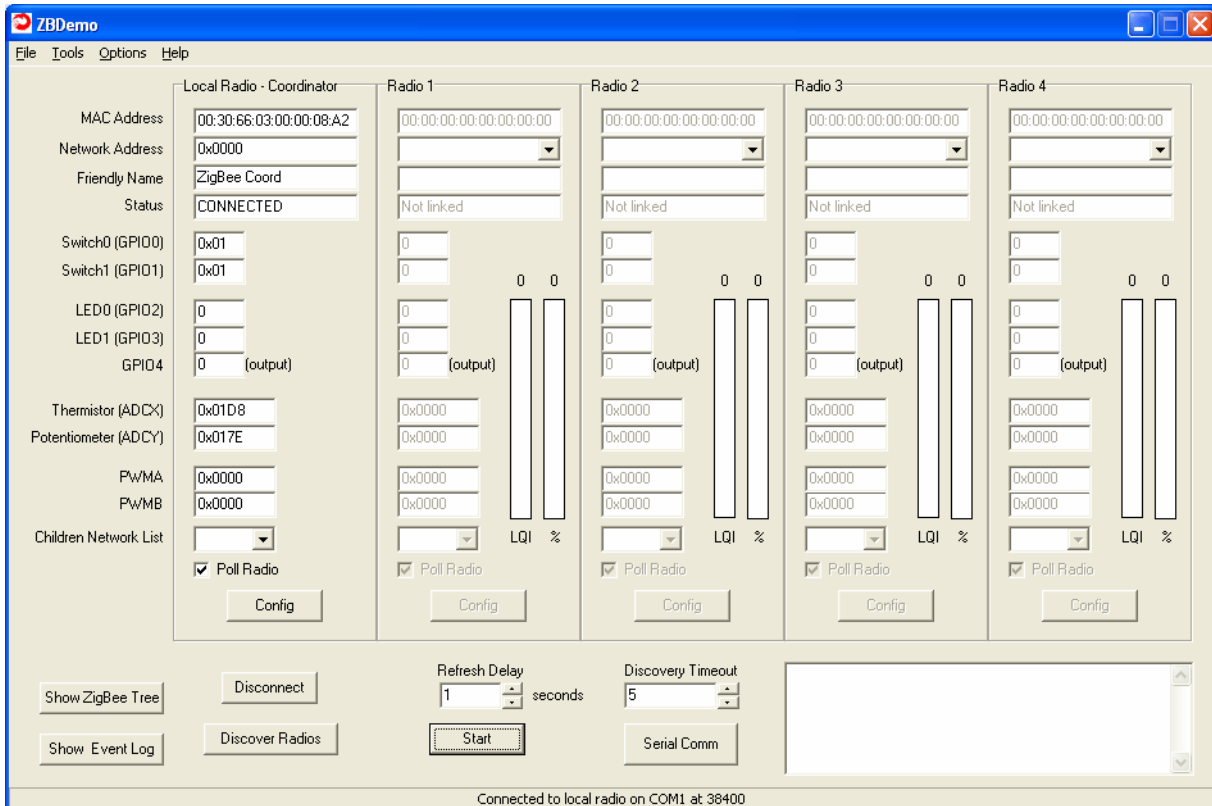
Once the radio has been found, click OK and after a few seconds, information about the ZigBee module in the radio will be displayed in the Local Radio column along with its 8-byte MAC address. The Status condition of CONNECTED indicates the computer has detected the radio as shown below.

ZBDemo works the same whether connecting to an RS-232 serial port or a USB port. On most computers, the USB Comm Ports have higher port numbers than RS-232 ports. Even on computers without an RS-232 serial port, the USB Comm Ports are numbered something other than 1 or 2. If a USB port is to be used, the drivers for the USB adapters must be installed on the PC.

The USB drivers are on the Documentation and Software CD under the Drivers folder. When the computer detects the presence of the hardware, the “Add New Hardware Wizard” box will appear. Click on “Next.” The next dialog box will ask what Windows should do. Click on the “Search for the best driver for your device” button and then click on “Next.” The next dialog box will ask where Windows should search for the driver. Put the included software CD in the computer and enter the drive letter for your CD drive. Click on the “Browse” button to the right of the window. A window will pop up displaying the contents of the CD. Double-click on the folder named “Drivers.” Click the “Next Button.”

Windows will indicate it has found the driver FTDIBUS.INF on the diskette. Click on the “Next” button to begin the driver installation. When Windows has finished the installation the messages “USB High Speed Serial Converter” and “Windows has finished installing the software that your new hardware device requires” will be displayed. Click on the “Finished” button to close the Wizard.

When ZBDemo has established a connection with a radio, the information for that radio will be displayed in the Local Radio column as shown below.



The Status of CONNECTED indicates the radio is communicating with ZBDemo and has nothing to do with whether or not the radio is part of a ZigBee network. Even when the radio is configured as a Router or End Device, the Status will indicate CONNECTED.

Each radio will have a different Network Address. As radios are discovered, they will be made available in the Network Address drop down menus. This allows you to selectively display any individual radio's parameters using the Network Address drop down menu. For situations in which more than 5 radios are being used, this feature allows you to view and/or reorganize how the parameters are displayed.

There are Poll Radio checkboxes at the bottom of each Radio column that allow the radio to be periodically polled for information when the Start button is pressed under the Refresh Delay field. A setting of 0 seconds in the Refresh Delay field will continuously poll the Radios that have Poll Radio checked. Longer Refresh Delay settings can be set using the up and down arrows to the right of the Refresh Delay field, or the number can be highlighted and a value entered. Once the Start button is pressed, it changes to a Stop button. Clicking on the Stop button will end the polling process.

## 5.1 Discover Radios

Once the ZBDemo has connected to the board, if the Link LED is illuminated, ZBDemo can discover the nodes that are part of the network. The Discover Radio button causes up to four radios that are part of the same ZigBee network as the Coordinator to be displayed at any one time and will read the status of the various components of each radio. Using the Network Address drop down menu, additional radios can be selected and their parameters displayed and/or changed. The status of each Radio will be LINKED. The MAC Address window will display the actual MAC address of each radio and the Switch, LED, Thermistor and Potentiometer windows will display the status of each component on the development board. The values that were present when the Discover Radios button was clicked will be displayed but the values will not be Refreshed until the Start button has been clicked. The Discover command can be issued from a Coordinator, Router, or End Device.

## 5.2 Refresh Delay

The Refresh Delay sets the frequency with which information from all radios, local and remote, is updated. The frequency can be adjusted in one second increments. A value of zero corresponds to a continuous updating. Since an updating of each radio requires a number of individual read and write commands, it may take a second to update each radio even if the Refresh Delay is set to zero. This time is a function of Windows and ZBDemo and not the ZigBee module.

Once the Refresh Delay has been chosen, click on the Start button to begin the updating. The status bar on the bottom left corner of the ZBDemo screen will indicate the progress being made by the program. When ZBDemo is updating the radio data, the Start button will change to a Stop button. When ZBDemo is idle, the Stop button will return to the Start button. If the Poll Radio box is unchecked for a radio, that radio will not have its information updated.

## 5.3 Switches

The Switch0 and Switch1 windows display the state of the two momentary push button switches on the development board. When the switches are open, the window will display 0x01. When

the switches are closed, the window will display 0x00. Note that ZBDemo will not update changes in either the Local Radio or the remote Radios until the Refresh is started by clicking on the Start button under the Refresh Delay window.

#### **5.4 GPIO LEDs**

The LED0 and LED1 windows allow the GPIO2 and GPIO3 module lines to be driven high or low to turn the LEDs on or off respectively. The LEDs are turned on by entering 1 in the window and turned off by entering a 0 in the window. As before, ZBDemo will not update changes in either the Local Radio or the remote Radios until the Refresh is started by clicking on the Start button under the Refresh Delay window.

#### **5.5 Thermistor, Potentiometer and Analog to Digital Inputs**

The Thermistor window displays the 10-bit ADC reading of the ADC X channel. Changes in temperature will cause the value displayed to change if ZBDemo has been put into update mode by clicking on the Start button under the Refresh Delay window. Similarly, the Potentiometer window displays the resultant 10-bit ADC value of the voltage divider created by the pot. Changing the voltage by adjusting the pot will change the value displayed in the window if ZBDemo is refreshing the data as above. The potentiometer and thermistor can be removed from the circuit by removing the headers from JP3 and JP4 respectively.

#### **5.6 Digital to Analog Outputs**

The PWMA and PWMB windows allow adjusting the duty cycle of the PWM outputs. These outputs are fed into an RC network for low pass filtering to generate an analog voltage. The allowable values are 0x0000 to 0xFFFF.

#### **5.7 Children's Network List**

Coordinators and Routers can be parents to other devices in the network. The Coordinator will always have children as long as there is at least a second device in the network. Routers may or may not have any children and End Devices cannot have children. The drop down menu will display the network address of all children associated with the radio in that column.

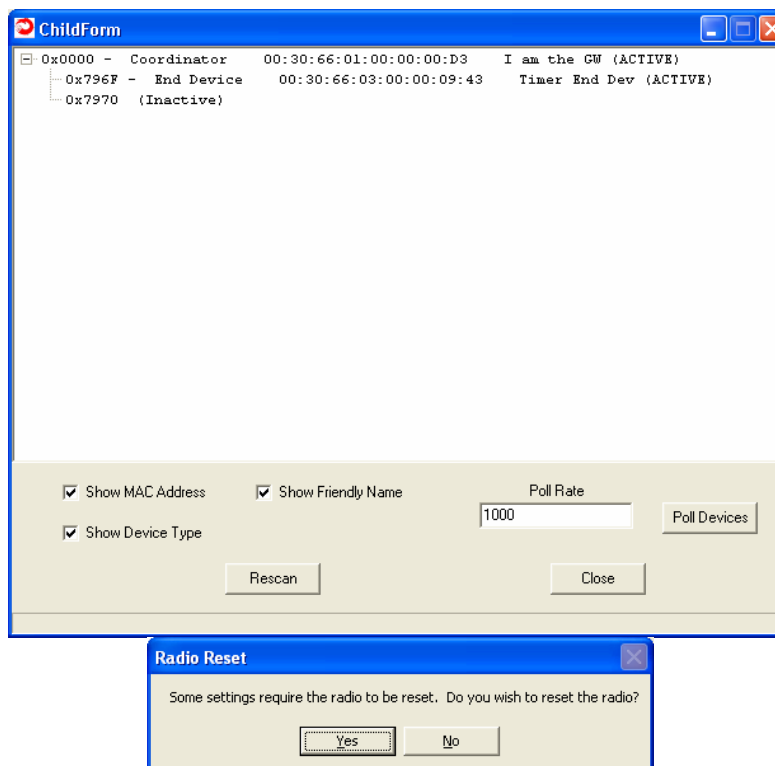
## 5.8 Show ZigBee Tree

The Show ZigBee Tree button displays a hierarchical view of the network. The Coordinator is displayed at the top. When there are other devices in the network, a plus sign will be displayed under the Coordinator. Clicking on the plus sign will expand the ZigBee Tree displaying the Routers and End Devices directly associated with the Coordinator. Any Routers with children will also have a plus sign. Clicking on the plus sign will expand the children list of that Router. This will continue until the entire network is displayed. The default is to display the MAC address of the network nodes but you can have the network addresses and friendly names displayed by checking the corresponding boxes.

## 5.9 Config

The Config button brings up the Module Clusters dialog which allows configuration parameters of each radio to be modified. There are seven possible tabs across the top of the dialog window that correspond to the Zigbee clusters; Config, Config Enable Options, Module I/O, Network, Diag, RF and End Device. However, the End Device tab will only appear if the applicable radio is configured as an End Device. It is shown in the screens below for informational purposes. A **Broadcast Mode** checkbox appears in the lower left hand side of each Cluster's tab that sends the applicable Cluster's information to all radios.

Once values have been modified, click on the **Apply** button to write the new values in the module. If a reset of the module is required for the changes to take effect, the Radio Reset dialog will be displayed as shown below. Click on Yes to reset the radio, Click on No to have the changes take affect on the next power cycle.



**Note:** See individual Cluster tables to determine if a Radio Reset is required.



### 5.9.1 Config Tab

The **Serial Baudrate** sets the communication rate for the UART port of the module. Changes take effect immediately upon clicking Apply.

The **Model Number** field shows whether the device is a Coordinator, Router or End Device.

**Friendly Name** is a 16-byte field that allows user-defined names to be assigned to modules for easier identification in the field. It allows the user to assign applicable names to each radio in the system, i.e. Main Office for the Coordinator and Pump Well 1 for the Router out on the site and so on.

**Sleep Mode** This is used on End Devices Only. Checking this box will configure the module as a battery-powered sleeping node. The default sleep mode is Timer Sleep. If Interrupt Sleep mode is desired, it can be selected under the *Config Enable Options* tab. This setting should be configured before the device joins the network. If it is changed after the device has joined the network, all devices in the network must be power cycled or reset, starting with the coordinator, to insure the network functions properly.

**Device Mode** This is currently Read Only. On read back it will be 0x00 for a Coordinator, 0x01 for a Router, and 0x02 for an End Device.

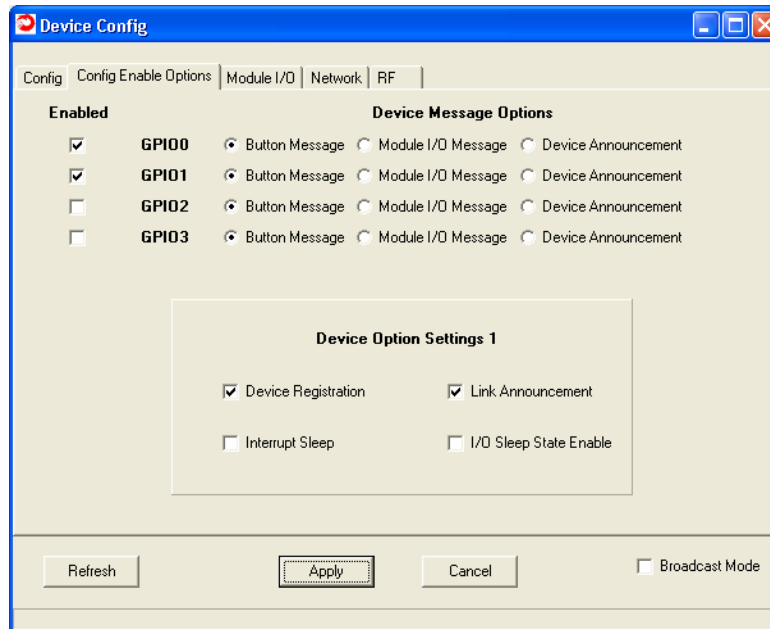
**Transparent Mode** A check in this box implements Transparent Mode and allows data without the CSM packetization to be transmitted. Any data appearing on the serial input will be transmitted to the coordinator. Data received by the device will also be output by the module without the CSM packetization. Once you enter Transparent Mode, ZBDemo will no longer work since it is expecting data to be in CSM packets. See Section 8.2 for details on protocol mode. Transparent Mode is only for a Router or End Device.

**Reporting Mode** Checking this box enables the device to send the module I/O cluster in an EVENT packet to the coordinator at an interval determined by the **Reporting Rate**. This mode works for all device types.

**Reporting Rate** This is a 32-bit value that sets the reporting interval of the reporting mode. The resolution is in 1ms increments but with the accuracy of the smaller of the Check Parent Rate and 500ms. The value may vary anywhere from 1000ms (0x000003E8) to 49.7 days (0xFFFFFFFF). If an attempt to set it less than 1000ms is made, an INVALID\_RANGE error packet will be returned.

**Firmware Version** is a read-only register that identifies the firmware revision level.

## 5.9.2 Config Enable Options



This page allows you to enable **Device Message Options** for the various GPIOs (general purpose inputs/outputs).

Putting a check in the **Enabled** box allows the device to issue various messages when GPIOs 0 through 3 are set as interruptible inputs. When the input changes from high to low, one message (defined by the radio buttons under **Device Message Options**) will be transmitted to the gateway (default is the Coordinator)

**Device Message Options** allow you to individually set what types of messages are initiated by the GPIOs 0 through 3. The options are:

- **Button Message** - This message is simply an Event message that describes the particular GPIO input state. It should always be 0b0 since the interrupts are falling edge triggered.
- **Module I/O Message** - This message is an Event containing the current data from the entire Module I/O cluster. This is the same data that is sent when the Reporting Mode is enabled.
- **Device Announcement** - This message will force the device to re-register with the gateway (coordinator) by sending a device announce packet.

**Device Options Settings** allow you to check or uncheck the following settings:

- **Device Registration** - By checking this box, the device will output a Device Registration packet for every device that joins its network. The box is checked by default.

- **Interrupt Sleep** - If the device is an End Device, checking this box when the Mains Powered box is not checked under Power Source on the *Config* tab, puts the module into Interrupt Sleep where the module can only be awakened by an interrupt on one of the GPIO lines. In this mode, the module will not perform a Check Parent operation until it is awakened by an interrupt.
- **Link Announcement** - By checking this box, the device will output a Link Announce packet when it has registered (Routers and End Devices) or formed a network (Coordinators). The Link Announce packet is output on the module's UART port. The bit is enabled by default.
- **I/O Sleep State Enable** – Checking this box causes the module to set the module GPIO pins to the state selected for each GPIO on the *End Device* tab. This feature is intended to be used for sleeping End Devices to avoid conflicts with attached circuits when the module goes to sleep. Improperly setting the I/O Sleep State of a GPIO can cause a significant increase in sleep mode current consumption due to sneak paths from the module pin through other circuits to ground or power.

### 5.9.3 Module I/O Tab

| GPIO  | GPIO Direction  | GPIO Init | GPIO Interruptible                                |
|-------|---|-----------|---|
| GPIO0 | <input checked="" type="radio"/> Input <input type="radio"/> Output | NA        | <input checked="" type="checkbox"/> Interruptible |
| GPIO1 | <input checked="" type="radio"/> Input <input type="radio"/> Output | NA        | <input checked="" type="checkbox"/> Interruptible |
| GPIO2 | <input type="radio"/> Input <input checked="" type="radio"/> Output | 0         | <input type="checkbox"/> Interruptible            |
| GPIO3 | <input type="radio"/> Input <input checked="" type="radio"/> Output | 0         | <input type="checkbox"/> Interruptible            |
| GPIO4 | <input type="radio"/> Input <input checked="" type="radio"/> Output | 0         |   |
| GPIO5 | <input type="radio"/> Input <input checked="" type="radio"/> Output | 0         |   |

DAC A Init: 0x0000      DAC B Init: 0x0000

Buttons: Refresh, Apply, Cancel, Broadcast Mode (checkbox)

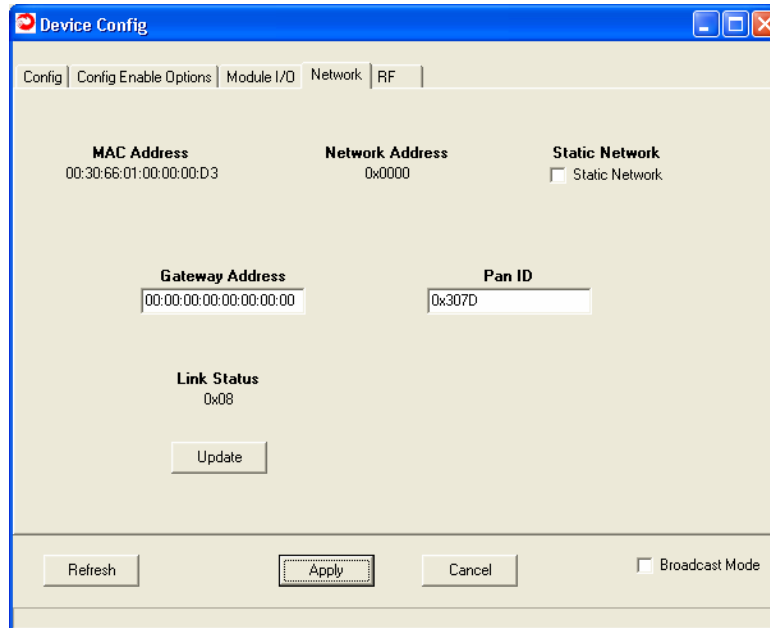
The **GPIO Direction** allows you to individually set GPIO0-5 as either inputs or outputs.

The **GPIO Init** is a non-volatile setting for the power up value of all the GPIO pins set as outputs. If a pin is set as an input this setting has no effect. Each time a reboot occurs (power cycle, command, etc.) this will be the default output level.

The **GPIO Interruptible** register allows GPIO0..GPIO3 to be used as interrupts to wake a module not configured for Mains Powered. If Interrupt Sleep Mode is enabled but no GPIO line is set as interruptible, the ZigBee module will return an error.

**DAC A Init** and **DAC B Init** initialization registers are non-volatile settings for the analog output values after a reset command or power cycling. Each time a reboot occurs this will be the default output level for either DAC A or DAC B.

## 5.9.4 Network Tab



The **MAC Address** is the unique IEEE address of the device and is set at the factory.

The **Network Address** is a read-only register that contains the network address assigned to the device by its parent. It cannot be changed by a user or pre-configured in any way. If the module is not linked, its value will be **FFFF**.

A check in the **Static Network** box allows the user to force the device to remain in the same network configuration from power-up to power-up. Uncheck the **Static Network** box and the device can join (or configure) the network in a different manner every time it restarts and thus, Network Addresses will change. Refer to Section 2.6 for details on Static Network Addresses.

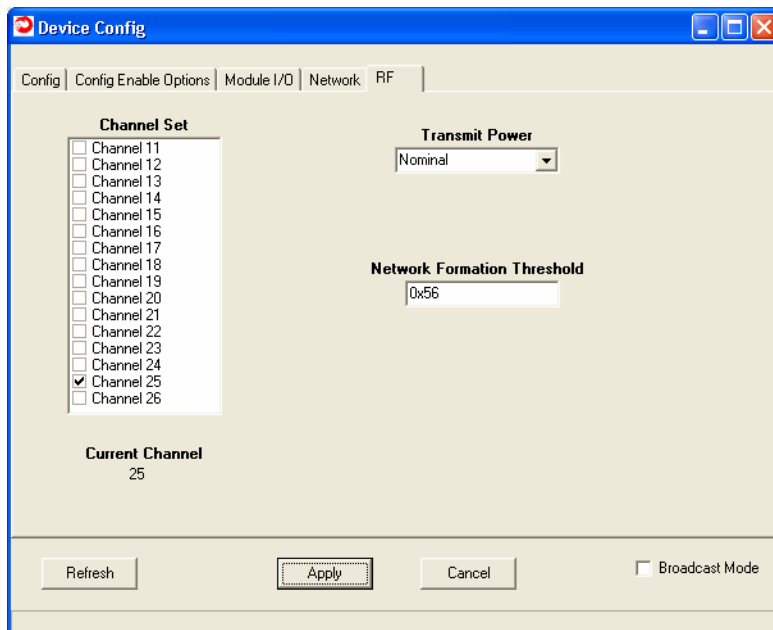
**Gateway Address** This is the destination network address for data sent to a module through its UART port in Transparent Mode. Currently, this is set for the Coordinator and cannot be changed.

A **PAN ID** is required for the Coordinator and can be any 14 bit number, i.e. **0x0000** to **0x3FFF**. In order for the Router to link to the Coordinator, its PAN ID must match the PAN ID on the Coordinator. Setting a the PAN ID of a Router to **0xFFFF** will allow a Router to connect to any Coordinator on the same RF Channel. Setting **0xFFFF** on the Coordinator causes it to pick a random value.

**Link Status** is a read-only register that provides information about the device's link status. (Update button refreshes Link Status.)

| LinkStatus Value | Description   |
|------------------|---|
| 0x01             | Device is initialized, but not connected                                  |
| 0x02             | Device is discovering PANs to join  |
| 0x03             | Device is joining a PAN   |
| 0x04             | Device has joined but is not yet authenticated by the Trust Center        |
| 0x05             | Device has been authenticated and has joined the network as an End Device |
| 0x06             | Device has been authenticated and has joined the network as a Router      |
| 0x07             | Device is starting a network  |
| 0x08             | Device has started a network as the coordinator                           |
| 0x09             | Device has been orphaned  |

### 5.9.5 RF Tab



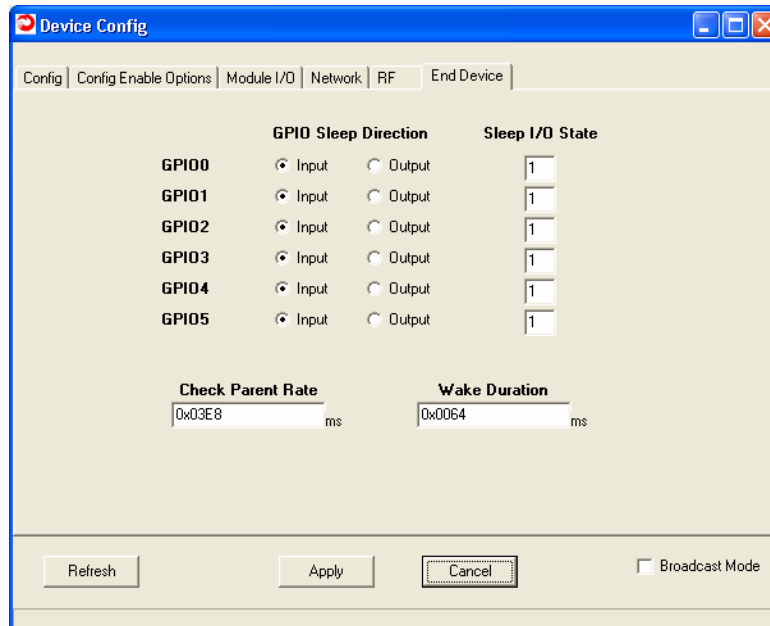
The **Channel Set** selects the RF channel(s) the module can use. Any number can be checked or unchecked as long as at least one channel is selected. If no channels are selected, when you click on Apply, an error message will display. Simply click in a box to add a check mark. Clicking on a box that already has a checkmark will remove it. The Coordinator listens on the first channel of the channel(s) that have been specified in the Channel Set field. If it detects RF energy on that channel above the Network Formation Threshold level, it moves to the next channel in the channel set list until it finds a clear channel. Once a channel has been found, text will display in the Status text window in the lower right part of the main ZBDemo window indicating that a network has been formed. If a clear channel cannot be found, the Link LED will not be lit and no text will display in the Status text window. Once a clear channel has been found, the Coordinator will listen for other devices seeking to associate. See Section 8 for a more detailed explanation.

**Transmit Power** can be set to the following levels; Nominal or -1dB, -3dB, -5dB, -7dB, -10dB, -15dB or -25dB from the nominal. Nominal transmit power for the ZMN2405HP module is +18dBm at the RF connector thus setting transmit power to -10dBm will result in +18dBm – 10dBm = +8dBm at the RF connector. Nominal level for the ZMN2405 module is 0dBm at the RF connector. The user may find it necessary to change the level if for instance, regulations require lower output power and/or when using a high gain antenna.

The **Network Formation Threshold** variable is a signed 8-bit value(0xFF = -1) in dBm which is used to vary the threshold around a default of -40dBm. Energy detected above the corresponding power level will stop the Coordinator from starting a PAN in the tested channel. This is helpful in forcing a network in an area with a lot of RF noise. The equation is:

$$\text{Threshold} = -40\text{dBm} + \text{Network Formation Threshold}$$

### 5.9.6 End Device Tab



The **GPIO Sleep Direction** allows you to control the direction of the GPIOs during a device's sleep period if the I/O Sleep State was enabled on the *Config Enable Options* tab. This enables the user to provide alternate configurations during sleep that will help minimize current consumption.

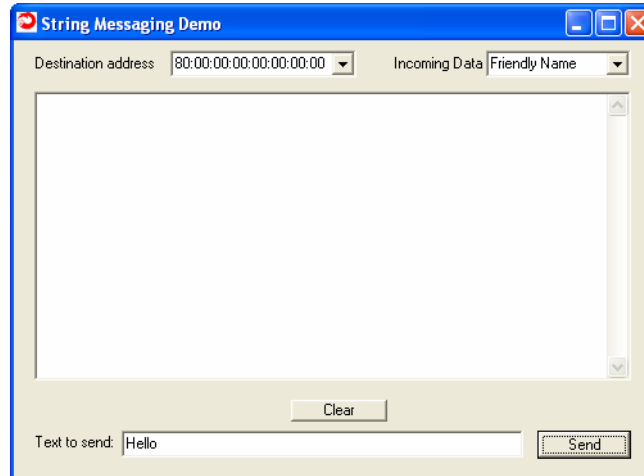
**Sleep I/O State** is used to set the output state of the GPIOs during a device's sleep period if the I/O Sleep State was enabled on the *Config Enable Options* tab. This also enables the user to provide alternate configurations during sleep that will help minimize current consumption.

**Check Parent Rate** - This setting controls the rate at which a Timer Sleep Mode End Device will awaken and ask its parent for any queued messages. Because an Interrupt Sleep End Device must request stored messages while it is awake, this setting controls that rate while the device has been externally awakened. The setting resolution is in milliseconds and 0x0000 is an invalid setting.

**Wake Duration** - This setting controls the length of time a module will remain awake when it wakes due to Check Parent timer. The default setting is 100 milliseconds. The minimum length of time is 30 milliseconds and the maximum duration is 65.5 seconds.

## 5.10 Serial Comm

Click on the Serial Com button to bring up the text messaging window as shown below.

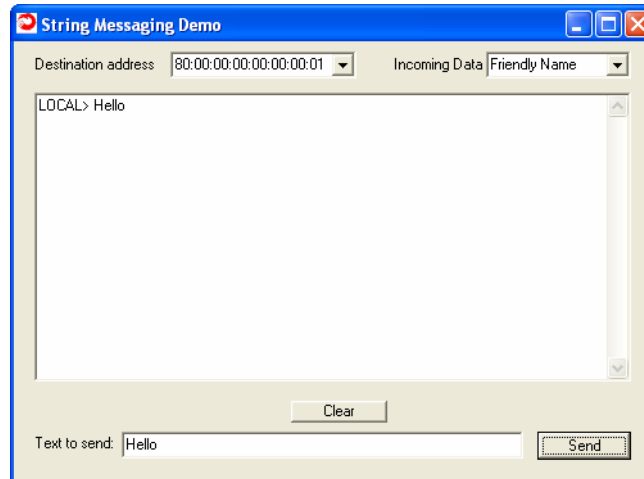


There are four fields/windows in this box: the Destination Address window top left of the box, the Incoming Data field top right of the box, the Messaging Display window in the middle of the box; and the Text to Send window at the bottom of the box. The Clear button allows you to clear the contents of the Messaging Display window.

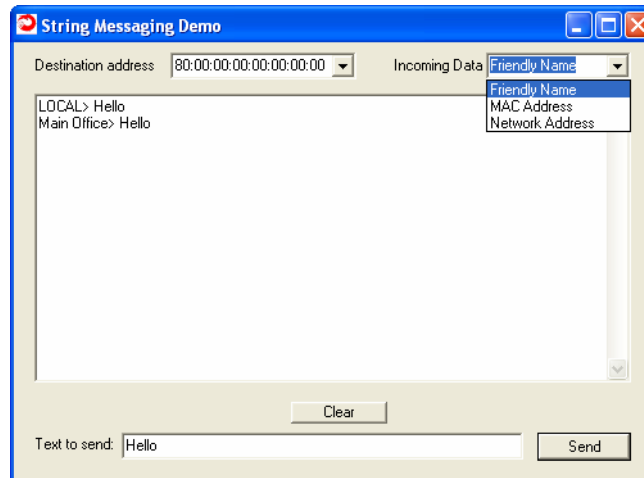
The Destination Address window is used to select the ZigBee device to which to send a text string. The drop down menu will display the Long Network Addresses of the devices that were found to be part of the network through the Discover command. The Long Network Address is the two bytes displayed in the Network Address field preceded by 80:00:00:00:00:00. Example: If the Network Address is 0x0001, the Long Network Address will be 80:00:00:00:00:00:00:01. Select the Long Network Address of the desired device.

ZigBee uses a packet format to send data. Rather than waiting for a long pause between keystrokes to determine the end of the data, ZBDemo requires the text be entered in the Text to Send window and the Send button clicked. ZBDemo can send a maximum of 68 bytes per packet thus the maximum number of characters that can be sent in a single packet is 64. When the String Messaging window is open, the radio information will not be refreshed.

When the Send button is clicked, the text in the Text to send window will appear in the chat window after a LOCAL> prompt.



When data is received by the local board from a remote ZigBee device, that string will be displayed in the chat window after a > prompt. The > prompt is labeled according to the selection designated in the Incoming Data drop down menu as shown below. In addition to Friendly Name, MAC Address or Network Address can be selected.



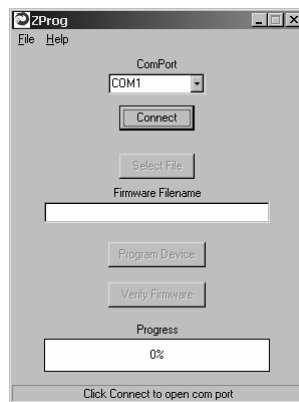


## 6. Programming the Module Firmware

The ZMN2405HP developer's kit includes three code loads which can be programmed into the ZMN2405HP module. All the modules come programmed from the factory, but Cirronet provides the capability to change the code in the module. The ZMN2405HP has bootloader code which allows the module firmware to be programmed through one of the two serial connections to the development board. The bootloader code removes the need for a JTAG programming device.

Three files are provided in the Code Loads directory of the Software and Manuals CD included with this kit. The files all start with "CSM" followed by "-C" for the coordinator, "-R" for the router, and "-E" for the end device. After the node type designation is the firmware version indicated by "vxxxxx" where xxxxx represents the version number.

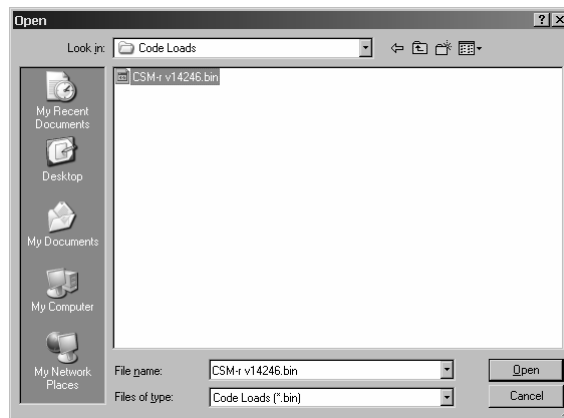
The modules are programmed using the ZProg.exe Windows utility program found in the same Code Loads directory on the CD. The program can be run from the CD or copied to your computer. When ZProg starts, the following window will appear.



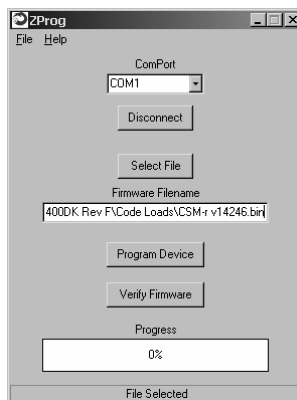
Select the serial port to which the dev kit board is connected from the drop down menu and click on Connect. If the serial port does not show up in the list, check to make sure no other program is open using that port, such as ZBDemo.

A Detecting Radio screen will appear and you will be prompted to cycle power to the dev kit. Simply unplug and reconnect the power supply cord from the dev kit board. Once the module has been detected, the Detecting Radio window will close and the Connect button on the ZProg will change to disconnect.

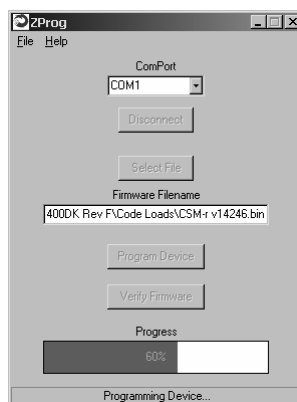
Click on the Select File button to choose the code load to use. If you know the complete path to the code load you want to use, you can type it in the Program Filename field. When you click on the Select File button the following screen will open.



Select the desired file from the list of .bin files listed. Either double-click on the file name or click once on the filename and then click on the Open button. The file select window will close and the ZProg window will now have the file you selected in the Program Filename window as shown below.



Click on the Program Device button to begin the programming process. The Progress bar will indicate the programming progress.



When the ZProg has completed programming the module, it will verify the image in the module matches the selected program file. The following window will display during the verification

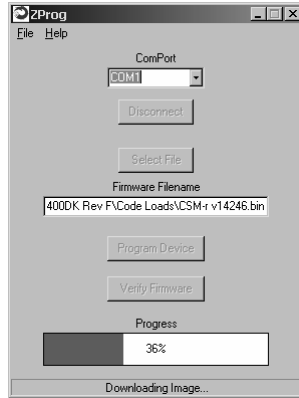


process. If the images agree, the verification window will close the success window will open.



The module is now ready for use.

ZProg can also verify the firmware that is in a module by comparing it to a program file selected. Once the file has been selected using the procedure above, click on the Verify Firmware button. The progress bar at the bottom of the ZProg screen will indicate the progress of the verification.



If the image in the module matches the selected filename, the following success window will open.



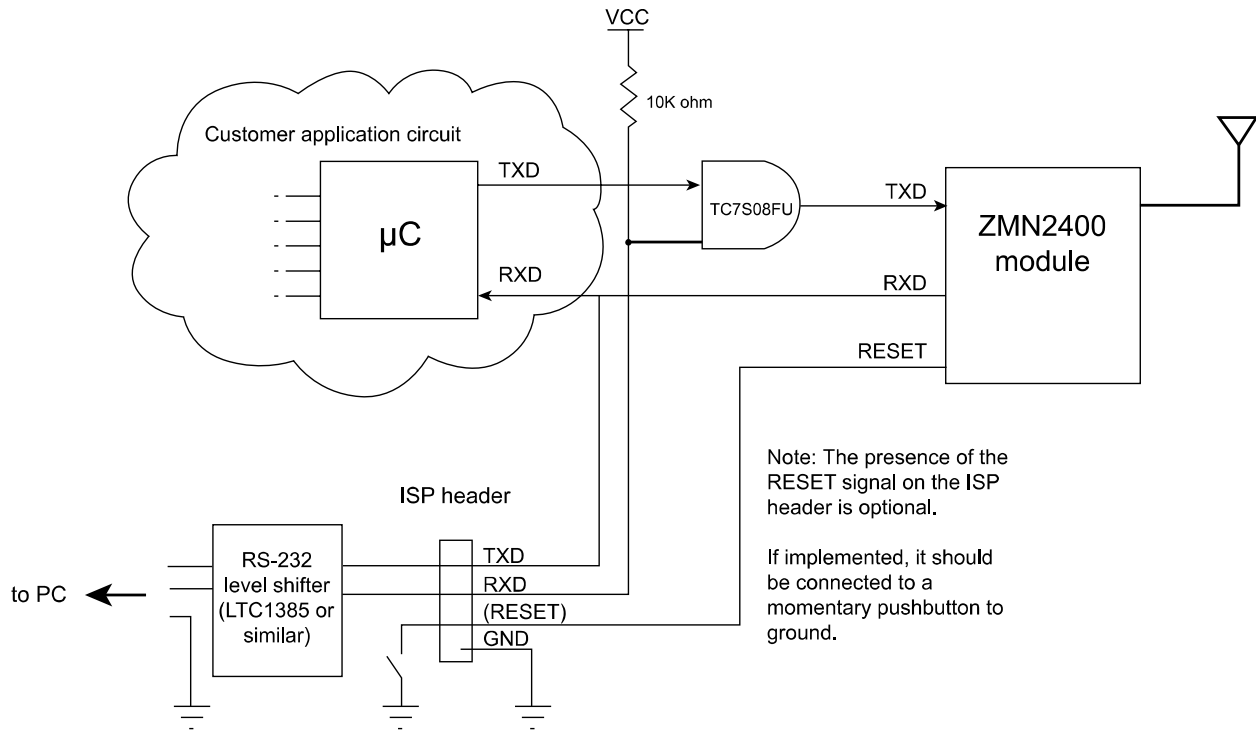
If the images do not match, a Verification Failed window will open with the message Images Differ across the top.

## 6.1 In Circuit Module Programming

Due to the early nature of ZigBee and the number of profiles currently being developed by the ZigBee Alliance, it is advisable to provide a means for reprogramming the module in the end product. If the end product has a serial port, either RS-232 or USB, nothing further is required. If the end product does not have a serial port then one must be added. The diagrams below show how one can be easily and inexpensively added.

There has been some information about over the air reprogramming of modules. While that will be the case in the future, the current size of the code and the memory resources available on the module do not allow that at this time.

Reference Circuit for Serial In-System Programming of the ZMN2405 module



## 7. Module & Development Board Hardware Specifications

### 7.1 Module Pin Descriptions

| Pin No.   | Name      | Description   |
|---|-----------|---|
| 1   | Vcc       | +3.3Vdc to +5.5Vdc  |
| 2,11,<br>17 – 20,<br>28, 33,<br>34, 36,<br>37, 39 | GND       | Power supply grounds. All ground pins must be connected to circuit ground.  |
| 3, 4  | PWMA - B  | Two pulse width modulated outputs that can be used to create an analog output with the addition of simple RC filters.                                     |
| 5 - 10  | GPIO0 - 5 | Six general purpose input/output pins. Software configurable as inputs or outputs. When configured as an output, power up state is software configurable. |
| 12  | LINK /TDO | Output status signal indicating module link status in default mode. When JTAG port is enabled, serves as JTAG Test Data Output.                           |
| 13  | /RST      | Active Low reset input. Tied to pin 24  |
| 14  | ACT/DC    | LED indicates RF data activity/JTAG data clock input. Leave disconnected if JTAG port will not be used.   |
| 15  | NC        | No Connection   |
| 16  | ADC REF   | Module's +3.3V supply, for use in ratiometric ADC readings. Load on this pin must be less than 5mA.   |
| 21  | UART_RX   | Receive data input signal of module UART. Data to be sent to the module is transmitted on this pin.   |
| 22  | UART_TX   | Transmit data output signal of module UART. Data received by the module will be transmitted to the local host on this pin.                                |
| 23  | NC        | No Connection. Leave disconnected.  |
| 24  | /RESET    | Active Low module hardware reset input. Must be held low when supply voltage is between +1.5Vdc and +2.7Vdc.  |
| 25 – 27   | ADCX – Z  | Three 10-bit Analog to Digital inputs. Inputs limited to 0Vdc to +2.5Vdc.   |
| 29  | SPI_EN    | Active Low chip enable output for SPI bus devices   |
| 30  | SPI_SCLK  | SPI port clock signal   |
| 31  | SPI_MOSI  | SPI port data output.   |
| 32  | SPI_MISO  | SPI port data input   |
| 35  | NC        | No Connection. Leave disconnected.  |
| 38  | RF        | RF signal output pin. Connect to antenna or antenna connector using a 50 ohm microstrip line.   |

## 7.2 Electrical Specifications

| Parameter                    | Min.                         | Typ. | Max.  | Units    | Condition          |
|------------------------------|------------------------------|------|-------|----------|--------------------|
| <b>Radio Characteristics</b> |                              |      |       |          |                    |
| Frequency Range              | 2.405                        |      | 2.480 | GHz      |                    |
| Spreading Method             | Direct Sequence              |      |       |          |                    |
| Modulation                   | O-QPSK                       |      |       |          |                    |
| RF Channels                  | ZMN2405 – 16; ZMN2405HP – 15 |      |       |          |                    |
| RF Data Rate                 | 250Kbps                      |      |       |          |                    |
| Channel Spacing              |                              | 5    |       | MHz      |                    |
| Spurious Emissions           |                              |      | -60   | dBm      | TX 30-1000MHz      |
| Harmonics                    |                              |      | -43   | dBm      | TX 1-12.75GHz      |
| 2 <sup>nd</sup> Harmonic     |                              |      | -43   | dBm      |                    |
| 3 <sup>rd</sup> Harmonic     |                              |      | -55   | dBm      |                    |
| Optimum load impedance       |                              | 50   |       | $\Omega$ | Using ext. antenna |
| Adjacent channel rejection   |                              | 46   |       | dB       | +5MHz              |
|                              |                              | 39   |       | dB       | -5MHz              |
| Alternate Channel Rejection  |                              | 58   |       | dB       | +10MHz             |
|                              |                              | 55   |       | dB       | -10MHz             |
| Frequency error tolerance    | -300                         |      | 300   | KHz      |                    |
| Symbol rate error tolerance  |                              |      | 120   | ppm      |                    |

### Transmit Power - Software Adjustable

|           |     |  |     |     |           |
|-----------|-----|--|-----|-----|-----------|
| ZMN2405   | -25 |  | 0   | dBm | At RF Pin |
| ZMN2405HP | -7  |  | +18 | dBm | At RF Pin |

### Receive Sensitivity

|           |  |     |  |     |                          |
|-----------|--|-----|--|-----|--------------------------|
| ZMN2405   |  | -91 |  | dBm | for 10 <sup>-5</sup> BER |
| ZMN2405HP |  | -95 |  | dBm | for 10 <sup>-5</sup> BER |

### I/O

|                          |  |
|--------------------------|--|
| Quantity and Designation | 1 — SPI Port<br>6 — General purpose I/O lines<br>3 — 10-bit ADCs<br>2 — PWMs<br>1 — UART |
|--------------------------|--|

### Environmental

|                             |     |    |    |    |         |
|-----------------------------|-----|----|----|----|---------|
| Operating temperature range | -40 | 25 | 85 | °C | Ambient |
|-----------------------------|-----|----|----|----|---------|

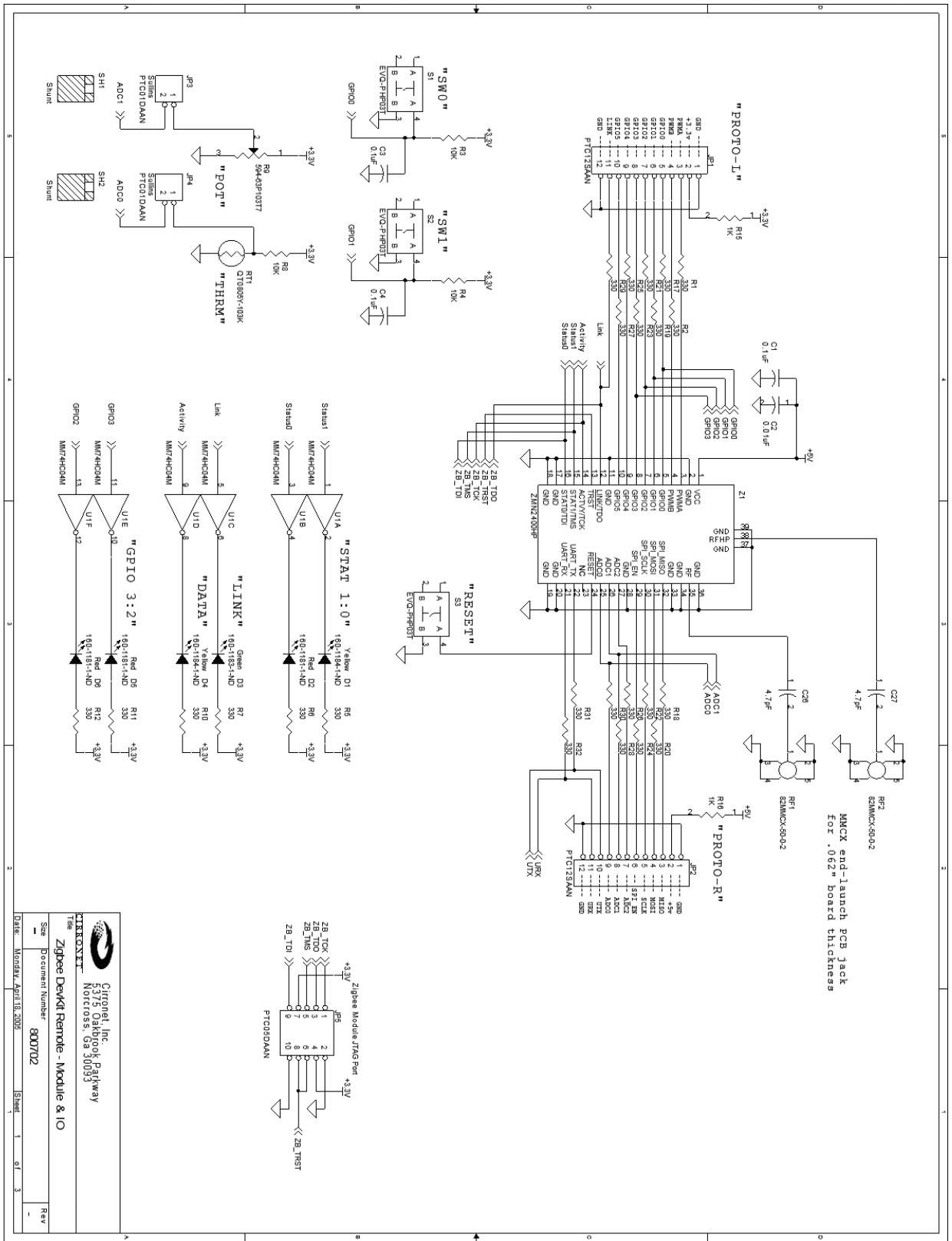
### 7.3 CC2430


The signal lines on the ZMN2405HP module are directly connected to the input pins of the CC2430. See the TI documentation below for details of those signals.

The signal lines brought out to the connectors on the development boards have 330 ohm resistors in series. The RF output on the module and the development board is designed for 50-ohm loads. Refer to the schematics on the next pages for details of the development board.

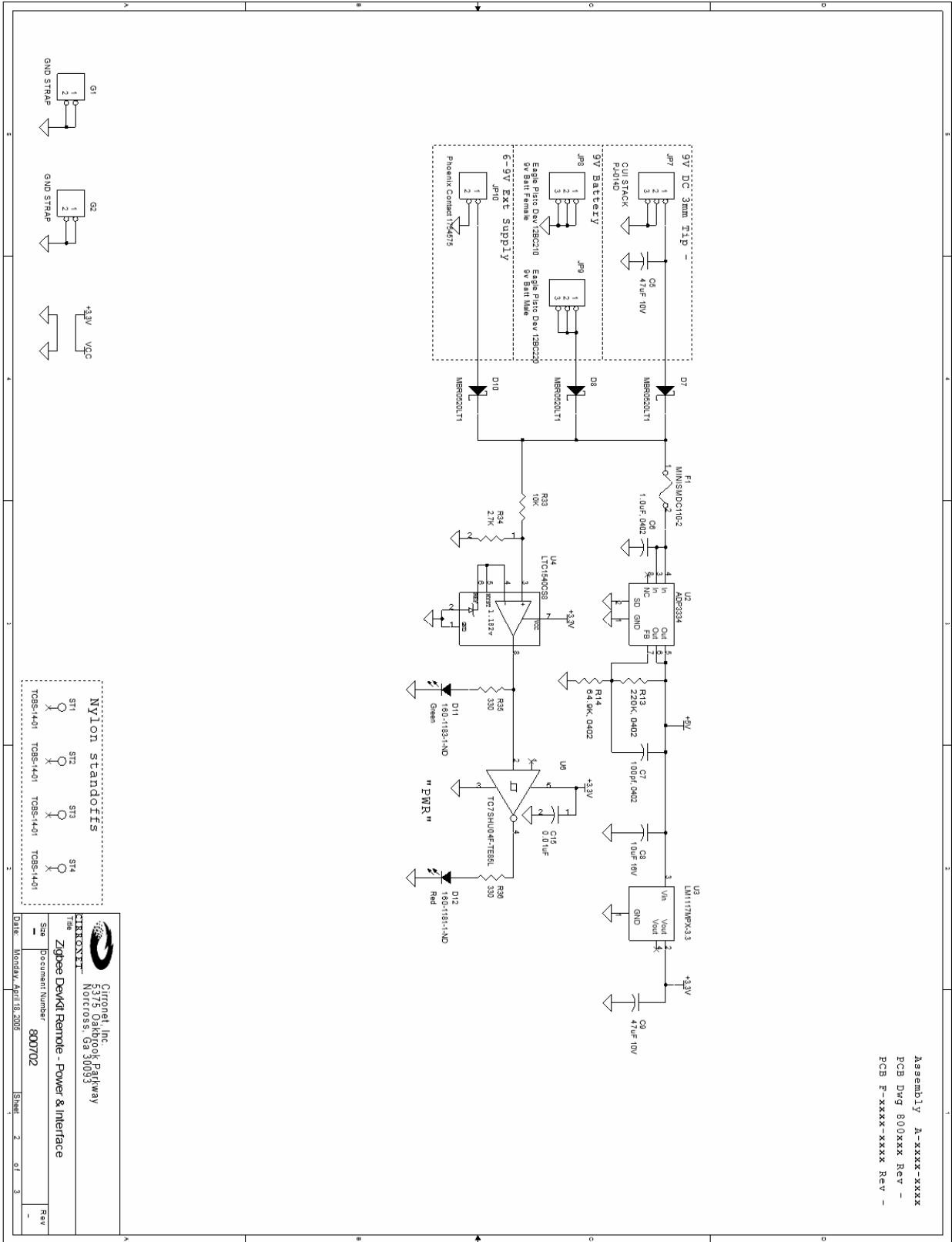
For complete details, please refer the documentation on the CC2430 available on the Texas Instruments' website, [www.ti.com](http://www.ti.com).

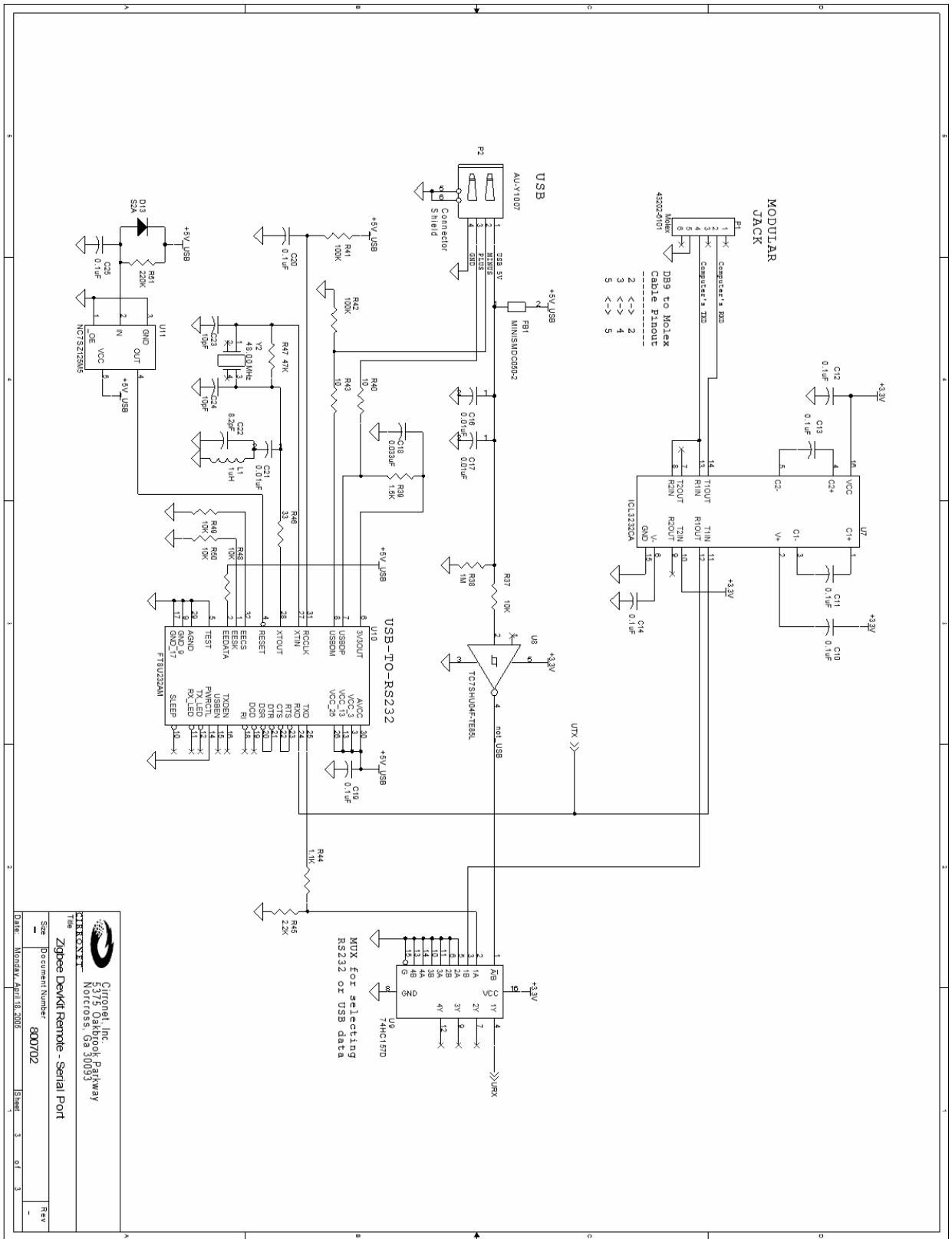
### 7.4 Schematics




**Cirronet, Inc.**  
 5375 Oakbrook Parkway  
 Norcross, Ga 30093  
**TIRES**  
**Zigbee DevKit Remote - Module & IO**  
 See Document Number **800702**  
 Date: Monday, April 18, 2006 Sheet 1 of 3








**Cirronet, Inc.**  
 5375 Oakbrook Parkway  
 Norcross, GA 30093  
**KLKROXIT™**  
 Zigbee DevKit Remote - Serial Port  
 800702  
 Date: Monday, April 18, 2005  
 Sheet 3 of 3  
 Rev -

## 8. Cirronet Standard Module (CSM) Profile API

All ZigBee modules require a firmware resident application to control the module. In ZigBee parlance, this application is called a “profile”. Schematically, the profile can be pictured as sitting on top of the ZigBee stack which in turn is sitting on top of the 802.15.4 stack. Cirronet has developed the Cirronet Standard Module (CSM) profile to provide access to all the resources on the module to off-module applications. Included on the CD that comes with ZMN2405 Developer’s Kit is the source code for ZB Demo which was written in Borland C++ CBuilder 6 for those who wish to utilize sections for their own applications. This section details the API to communicate with the CSM profile to manipulate the resources of the module.

All communications with the ZigBee module through the UART interface use a message/command protocol. Protocol commands are used to set configuration parameters, issue commands, send and receive data, report errors, etc. Thus all operations use a single packet structure with the message defined by a field within the packet. Details of the message format are provided in Section 8.2. Sample packet formats are provided in Section 8.11.

The various parameters that are used by the CSM profile are grouped together in a series of clusters. The different parameter or data locations are defined by an offset into a cluster. The first location in a cluster has an offset of zero. For the Module I/O cluster, the first location is the value of the ADC X channel. This is a 2-byte value (although only the lower 10 bits are valid) making the offset of the next value, ADC Y, two. Values are written to a cluster using the Set Field command and are read from a cluster using the Get Field command. The clusters used by the CSM profile are the Module I/O, Configuration, Reset, Network, RF, and Security clusters. Each cluster and its associated parameters and offsets are described in detail later in this section.

The module behavior is controlled by issuing Set Field commands with the desired values into the appropriate locations in the various clusters. For example, the module’s serial port baud rate is controlled by writing various values to the Serial Mode location in the Configuration cluster. Sleep Mode for an End Device is enabled through the Sleep Mode location in the same cluster.

Module I/O values are obtained by issuing a Get Field command to the appropriate location in the Module I/O cluster. The module’s GPIO lines are controlled by reading or writing to the appropriate location in the Module I/O cluster.

## 8.1 Module I/O Cluster (ID 0x01)

The ZMN2405 module has two DACs, three ADCs, six General Purpose I/O lines, a SPI port and a UART port. The Module I/O cluster defines how to manipulate the various lines and ports.

| Parameter                               | Offset | Bytes  | R/W | Reset | Description   |
|---|--------|--------|-----|-------|---|
| ADC X                                   | 0x0000 | 2      | R   | N     | On board 10-bit A to D converter channel X (Pin 25)   |
| ADC Y                                   | 0x0002 | 2      | R   | N     | On board 10-bit A to D converter channel Y (Pin 26)   |
| ADC Z                                   | 0x0004 | 2      | R   | N     | On board 10-bit A to D converter channel Z (Pin 27)   |
| DAC A                                   | 0x0006 | 2      | R/W | N     | On board 16-bit D to A converter channel A (Pin 3)  |
| DAC B                                   | 0x0008 | 2      | R/W | N     | On board 16-bit D to A converter channel B (Pin 4)  |
| GP I/O 0                                | 0x000A | 1      | R/W | N     | General Purpose I/O Line 0 (Pin 5)  |
| GP I/O 1                                | 0x000B | 1      | R/W | N     | General Purpose I/O Line 1 (Pin 6)  |
| GP I/O 2                                | 0x000C | 1      | R/W | N     | General Purpose I/O Line 2 (Pin 7)  |
| GP I/O 3                                | 0x000D | 1      | R/W | N     | General Purpose I/O Line 3 (Pin 8)  |
| GP I/O 4                                | 0x000E | 1      | R/W | N     | General Purpose I/O Line 4 (Pin 9)  |
| GP I/O 5                                | 0x000F | 1      | R/W | N     | General Purpose I/O Line 5 (Pin 10)   |
| SPI Port                                | 0x0010 | varies | R/W | N     | SPI Port data register  |
| UART Port                               | 0x0011 | varies | R/W | N     | UART Port data register – Send ASCII data   |
| GP I/O Direction                        | 0x0012 | 1      | R/W | N     | Bitmap that sets direction, input or output, of GP I/O pins. Default = 0 = input, 1 = output  |
| GP I/O Init                             | 0x0013 | 1      | R/W | Y     | The GPIO initialization register is a non-volatile setting for the value of all the GPIO output pins. Bits 0..5 correspond to GPIO0..GPIO5. If a pin is set as an input, then the corresponding bit in the register is a "don't care". The individual bit level is the corresponding bit output level.  |
| DAC A Init                              | 0x0014 | 1      | R/W | Y     | 16-bit value that defines DAC Channel A power up value. Default = 0x0000  |
| DAC B Init                              | 0x0016 | 1      | R/W | Y     | 16-bit value that defines DAC Channel B power up value. Default = 0x0000  |
| Status/GP I/O Alternate Function Enable | 0x0018 | 1      | R/W | Y     | Bitmap that selects alternate function mode for Status signals and GP I/O pins.<br>Bit 7: If 0 (Default) Pin 12 = LINK, Pin 14 = ACT, Pin 15 = STAT1, Pin 16 = STAT0<br>If 1, Pins 12, 14-16 are JTAG<br>Bit 6: Reserved<br>Bit 5: GP I/O 5 Default = 0 = I/O; 1 = Busy<br>Bits 4 – 0: GP I/O 4 – 0 Reserved  |
| GPIO Interrupts                         | 0x0019 | 1      | R/W | N     | Bitmap that allows GPIO0 – GPIO3 to be used as interrupts that send a RECEIVE_FIELD packet to the device's gateway. Each GPIO uses two bits in the register.<br>Bit 0 – 1: GPIO 0<br>Bit 2 – 3: GPIO 1<br>Bit 4 – 5: GPIO 2<br>Bit 6 – 7: GPIO 3<br>b00 – Non interruptible                      b01 – TBD<br>b10 – Falling edge                              b11 – TBD |

## 8.2 Configuration Cluster (ID 0x02)

A ZigBee network is setup or configured by writing desired parameters into the appropriate locations in the Configuration Cluster. The table below defines the parameters and their offsets for the Configuration Cluster.

| Parameter        | Offset | Bytes | R/W | Reset | Description   |
|------------------|--------|-------|-----|-------|---|
| Firmware Version | 0x0000 | 2     | R   | Y     | Allows module firmware revision to be read. The Firmware Revision is displayed as three numbers separated by periods, for example, v1.2.3. The first number (1) is designated by the first byte. The second byte is split into Upper Nybble and Lower Nybble. The Upper Nybble is the second number (2) and the third number (3) is designated by the Lower Nybble.               |
| Device Mode      | 0x0002 | 1     | R   | Y     | Indicates the mode of the module:<br>0x00 = Coordinator<br>0x01 = Router<br>0x02 – End Device   |
| Serial Mode      | 0x0003 | 2     | R/W | N     | Allows the device to communicate at different baud rates in order to match to a variety of different hosts.<br>1200 = 0x03<br>2400 = 0x04<br>4800 = 0x05<br>9600 = 0x06<br>19200 = 0x07<br>38400 = 0x08 (Default)<br>57600 = 0x09<br>115200 = 0x0B  |
| Model Number     | 0x0005 | 2     | R   | N/A   | This register identifies the Cirronet device and is read-only. The upper nybble represents the ZigBee logical device type. 0x0XXX -> Coordinator, 0x1XXX -> Router, 0x2XXX -> End Device, 0x3XXX  |
| Friendly Name    | 0x0007 | 16    | R/W | N     | User defined 16-byte field to identify devices. When writing the field for a custom application, 16 bytes must be written. If the desired Friendly Name is less than 16 bytes, the remaining bytes must be filled with a null character or whatever other character is desired to ease human readability of the name. This field can be read by the Coordinator or other Routers. |
| Sleep Mode       | 0x0017 | 1     | R/W | N     | This is used on End Devices only to enable End Device sleeping. 0x00 is disabled and is the default. 0x01 is enabled.   |

|                  |        |   |     |   |   |
|------------------|--------|---|-----|---|---|
| UART Data Mode   | 0x0018 | 1 | R/W | N | <p>This parameter selects between Transparent Mode (0x00, UART data has no packet format) and Protocol Mode (0x01, UART data uses packet format). When Transparent Mode is selected all data received on the UART port is sent to the Coordinator. To exit Transparent Mode, don't send any data for 2 seconds. Then send the following escape sequence of 8 bytes:<br/>(first) 0xED 0xAE 0xF9 0x2B 0x07 0x62 0x3C 0xED (last)</p> <p>Default = 0x01 = Protocol Mode</p>  |
| Option Settings1 | 0x0019 | 1 | R/W | N | <p>This register controls various options for the device according to the table below:</p> <p><b>Bit</b></p> <p><b>0 = Device Registration</b> - By setting this bit high, the device will output a Device Registration packet for every device that joins its network. The bit is enabled by default.</p> <p><b>1 = Link Announcement</b> - By setting this bit high, the device will output a Link Announce packet on its UART port when it has registered (Routers and End Devices) or formed a network (Coordinators). The bit is enabled by default.</p> <p><b>2 = Interrupt Sleep</b> - If the device is an End Device, setting this bit high puts the module in Interrupt Sleep mode where it can only be awakened by an interrupt. Interrupt Sleep mode turns off the Check Parent function as well as the Reporting Mode.</p> <p><b>3 = I/O Sleep State</b> - Setting this bit high causes the module to change the GPIO lines to the direction selected by SleepIODDRstate, and when selected as an output, to the level selected by SleepIOState. When selected as an input, the line is high impedance. This is beneficial if the I/O happens to be connected to low impedance devices.</p> <p><b>4 = GPIO0 Message Enable</b> - Setting this bit high causes the module to issue various messages when GPIO0 is set as an interruptible input. When the input changes from high to low, one message (defined by Option Settings 2) will be transmitted to the gateway (default is the Coordinator). The bit is enabled by default.</p> <p><b>5 = GPIO1 Message Enable</b> - Performs the same function for GPIO1. The bit is enabled by default.</p> <p><b>6 = GPIO2 Message Enable</b> - Performs the same function for GPIO2. The bit is enabled by default.</p> <p><b>7 = GPIO3 Message Enable</b> - Performs the same function for GPIO1. The bit is enabled by default.</p> <p>Default = 0xF3</p> |

|  |        |   |     |   |   |
|--|--------|---|-----|---|---|
| Message Options<br>(These are controlled by Option Setting 1)  | 0x001A | 1 | R/W | N | <p>This register sets the message options for interruptible inputs according to the table below:<br/> 0-1 = Message Options for GPIO_0.<br/> 2-3 = Message Options for GPIO_1.<br/> 4-5 = Message Options for GPIO_2.<br/> 6-7 = Message Options for GPIO_3.</p> <p><b>Note: Only one below can be selected</b><br/> 0b00: Button Message - This message is simply an Event that describes the particular GPIO input state. It should always be 0b0 since the interrupts are falling edge triggered. The mode is enabled by default.<br/> 0b01: Module I/O Message - This message is an Event containing the current data from the entire Module I/O cluster.<br/> 0b10: Device Announce Message - This message will force the device to re-register with the gateway (coordinator) by sending a device announce packet.<br/> 0b11: Reserved TBD.</p> <p>Default = 0x00</p> |
| Reporting Mode<br>(For all devices)                            | 0x001B | 1 | R/W | N | <p>When enable causes the module to report the module I/O cluster in an EVENT packet at intervals determined by the Reporting Rate below. This mode works for all device types. (0x00 = enabled, 0x01 = disabled=Default)</p>   |
| Reporting Rate<br>(For all devices)                            | 0x001C | 4 | R/W | N | <p>This is a 32 bit value that sets the reporting interval of the reporting mode. The value is set in 1ms increments and may vary anywhere from 1000ms (0x000003E8) to 49.7 days (0xFFFFFFFF). For Timer Sleep End Devices the accuracy of the interval is the smaller of the Check Parent Rate or 500 milliseconds.</p> <p>Default = 0x000003E8 = 1 second</p>   |
| Check Parent Rate<br>(Valid for End Devices Only) <sup>1</sup> | 0x0020 | 2 | R/W | N | <p>This 16 bit setting is an End Device-only setting that controls the rate at which a Timer Sleep End Device will awaken and ask its parent for any queued messages. Since an Interrupt End Device must still request stored messages while it is awake, this setting controls the rate while the device has been externally awakened. The setting resolution is in milliseconds with a default of 1 second. 0x0000 is an invalid setting.</p>   |
| Reserved   | 0x0022 | 2 | R/W | N | Reserved for future use. Do not change.   |
| Wake Duration<br>(Valid for End Devices Only) <sup>1</sup>     | 0x0024 | 2 | R/W | N | <p>This 16 bit setting controls the length of time a Timer Sleep End Device will remain awake after it has sent data. The setting resolution is in milliseconds with a default of 100 milliseconds. 0x0000 is an invalid setting.</p>   |

|   |        |   |     |   |   |
|---|--------|---|-----|---|---|
| Sleep I/O Direction<br>(Valid for End Devices Only) <sup>1</sup>    | 0x0026 | 1 | R/W | N | <p>The six LSBs (least significant bits) of this byte are used to control the direction of the GPIOs during a device's sleep period. This enables the user to provide alternate configurations during sleep that will help minimize current consumption. Bits 0..5 correspond to GPIO0..GPIO5.<br/>0 = Input, 1 = Output</p> <p>Default = 0x00</p>  |
| Sleep I/O Output Level<br>(Valid for End Devices Only) <sup>1</sup> | 0x0027 | 1 | R/W | N | <p>The six LSBs (least significant bits) of this byte are used to set the output level of the GPIOs selected as outputs by SleepIODDR when IOSleepState is enabled during a device's sleep period. This enables the user to provide alternate configurations during sleep that will help minimize current consumption. Bits 0..5 correspond to GPIO0..GPIO5.<br/>0 =Low, 1 = High</p> <p>Default = 0x3F</p> |

<sup>1</sup> These exist and can be read in both the Coordinator and Router but have no validity.

**Note: For multiple byte fields, data must be entered LSB (least significant bit) first.**



### 8.3 Reset Cluster (ID 0x03)

The Reset cluster provides a means to execute various resets of the module.

| Parameter              | Offset | Bytes | R/W | Reset | Description   |
|------------------------|--------|-------|-----|-------|---|
| Microcontroller Reset  | 0x0000 | 1     | W   | Auto  | Resets the microcontroller on the module. 0x5A is the value to write to reset the radio   |
| Reset Factory Defaults | 0x0001 | 1     | W   | Auto  | Resets all Cluster parameters to factory default values then resets the microcontroller. 0x5A is the value to write to restore factory defaults listed below. |

| Factory Defaults                                     |   |
|--|---|
| Parameter  | Default Value                                   |
| Friendly Name<br>Coordinator<br>Router<br>End Device | ZigBee Coord<br>ZigBee Router<br>ZigBee End Dev |
| Mode<br>Coordinator<br>Router<br>End Device          | 0x00<br>0x01<br>0x03                            |
| Model Number   | 0x?000 Where ? = Mode                           |
| Baud Rate  | 38400   |
| Sleep Enable   | 0x00 (Disabled)                                 |
| Protocol Mode  | 0x01 (Enabled)                                  |
| DAC A Initial Value                                  | 0x0000  |
| DAC B Initial Value                                  | 0x0000  |
| GPIO Initial Output Values                           | 0x00  |
| GPIO Direction                                       | 0xFC (All set to Outputs)                       |
| Security Code  | 0x0F0E0D0C0B0A09080706050403020100              |
| Security PIN   | 0x76543210                                      |
| Security Level                                       | 0x00  |
| Default Gateway Address                              | 0x0000000000000000                              |
| Static Network Enable                                | 0x00 (Disabled = Dynamic)                       |
| PAN ID   | 0x307D  |
| Channel Mask   | 0x00004000                                      |
| Transmit Power Attenuation                           | 0dB   |
| Network Formation Threshold                          | 0x56  |

#### 8.4 Network Cluster (ID 0x07)

| Parameter       | Offset | Bytes | R/W | Reset | Description   |
|-----------------|--------|-------|-----|-------|---|
| MAC Address     | 0x0000 | 8     | R   | N/A   | Returns factory programmed MAC address  |
| Network Address | 0x0008 | 2     | R   | N/A   | This is a read-only register that contains the network address assigned to the device by its parent.  |
| Gateway Address | 0x000A | 8     | R/W | Y     | Specifies 8-byte ZigBee MAC address of gateway to be used after power up. This allows bindings to survive power outages.  |
| Static Network  | 0x0012 | 1     | R/W | Y     | This variable allows the user to force the device to remain in the same network configuration from power-up to power-up. Set to 0x01 when a static network is desired. 0x00 will allow the device to join (or configure) the network in a different manner every time it restarts. Refer to Section 2.6.<br><br>Default = Disabled  |
| Default PAN ID  | 0x0013 | 2     | R/W | Y     | Setting this variable to anything other than 0xFFFF will force a Router or End Device to search for a network with that specific PAN ID. All others will be rejected. A Coordinator will start a network with this variable as the PAN ID. The two high order bits are currently masked, so a PAN ID of 0x7FFF will be the same as 0x3FFF. A value of 0xFFFF causes a Coordinator to form a network with a random PAN ID and A Router or End Device will join ANY PAN ID.<br><br>Default = 0x307D   |
| Link Status     | 0x0015 | 1     | R   | n/a   | This variable is a read-only register that informs the user about the device link status. There can be intermediate status steps until the device is finally linked.<br>The values are as follows:<br>0x01 – Device is initialized but not connected<br>0x02 – Device is discovering PANs to join<br>0x03 – Device is joining a PAN<br>0x04 – Device has joined but is not yet authenticated by the Trust Center<br>0x05 – Device has been authenticated and has joined the network as an End Device<br>0x06 – Device has been authenticated and has joined the network as a Router<br>0x07 – Device is starting the network<br>0x08 – Device has started a network as the Coordinator<br>0x09 – Device has been orphaned |

## 8.5 RF Cluster (ID 0x08)

| Parameter                   | Offset | Bytes | R/W | Reset | Description  |
|-----------------------------|--------|-------|-----|-------|--|
| Channel List                | 0x0000 | 4     | R/W | Y     | <p>This variable allows the user to give the device a choice of channels for a Coordinator to form a network or a Router (and End Device) to search for a network. This is a bit-map that represents the frequency channels. At least one must be selected (set high). Obviously for devices that wish to communicate, a common channel must be selected. The allowable bit-map is 0x07FFF800. The lowest single channel is 0x00000800: 2405MHz. The highest single channel is 0x04000000: 2480MHz. Any combination of all the channels (frequencies) may be chosen, but the radio will only use one channel at any given time, i.e.: it is not a frequency hopper.</p> <p>Default = 0x00004000 = Channel 12</p> |
| Channel Used                | 0x0004 | 1     | R   | N/A   | <p>This is a read-only byte that represents the channel that the radio is currently using. The meaning of this variable is more important when there are multiple channels for the device to choose.</p>   |
| TX Power                    | 0x0005 | 1     | R/W | N     | <p>The user may find it necessary to change the power level that the radio is transmitting. For instance, if regulations require lower output power, this variable can reduce the nominal module transmit power by as much as 25dB. The value specified determines the amount the nominal module power will be reduced not the actual power level. Allowable values are:</p> <p>0x00 = 0 dB<br/> 0x21 = -1 dB<br/> 0x23 = -3 dB<br/> 0x25 = -5 dB<br/> 0x27 = -7 dB<br/> 0x2A = -10 dB<br/> 0x2F = -15 dB<br/> 0x39 = -25 dB</p> <p>Default = 0dB</p>  |
| Network Formation Threshold | 0x0006 | 1     | R/W | Y     | <p>The Network Formation Threshold variable is a signed 8-bit value. Energy detected above the corresponding power level will stop the ZC from starting a PAN in the tested channel. This is helpful in forcing a network in an area with a lot of RF noise. The power level corresponding to the threshold is:</p> <p>PdBm = -45 + Threshold</p> <p>Default = 0x56 = ~ -2dBm</p>  |

## 8.6 Security Cluster (ID 0x09)

| Parameter     | Offset | Length | R/W | Description  |
|---------------|--------|--------|-----|--|
| Security Code | 0x0000 | 10     | W   | <p>TBD. The security code is used to provide network and link level security. If it is used, every device in the network must use the same code and use the same level of security. Use the incorrect code or none at all and the device will not be allowed to join the network. The code can be any 16-byte binary sequence and will be used for encrypting the data if the Security Level is appropriate.</p> <p>Default =</p>  |
| Security PIN  | 0x0010 | 4      | W   | <p>This four-byte sequence is used for device security. When set to anything other than the default, all clusters are locked and cannot be written to or read from. If an attempt is made to read or write to the device, an error packet is generated and returned with a permission error code. To unlock the device cluster write the correct 4-byte PIN value to this location. To lock them again, write any other 4-byte value to this location. In order to set or change the PIN, write an 8-byte value to this location where the first four bytes are the new PIN number and the second four bytes are the old PIN number. Remember the new PIN number because a factory reset will not reset this to the default value.</p> <p>Default = 0x76543210</p> |

## 8.7 Serial Interface

The serial interface is a standard 3-wire (RX, TX, and GND) interface used for external module communication operating with the following default parameters:

| <b>Baud Rate</b> | <b># of Bits</b> | <b># of Stop Bits</b> | <b>Parity</b> |
|------------------|------------------|-----------------------|---------------|
| 38.462Kbps       | 8                | 1                     | No            |

**Table 5. Serial Interface Parameters**

The serial interface setup will be capable of adjusting the baud rate, at least. The default may remain 38.4Kbps but max data rate will be 115Kbps. This is a variable referred to in the Data Structures.

## Serial Protocol

The module uses a serial protocol for external communication. The protocol allows for configuration, command, status and data transfer on the local device or a remote device. The figure below illustrates the protocol packet format. All fields except the Arguments field are single byte fields. Multi-byte parameters are sent LSB first.

|                      |                             |                |                           |                  |  |
|----------------------|-----------------------------|----------------|---------------------------|------------------|--|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte                    | Varies           |  |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG</b><br><b>Type</b> | <b>Arguments</b> |  |

The **SOP** or Start Off Packet field signifies the beginning of a packet, its value is 0xFD. The **Length** field is the total number of bytes in the remainder of the packet (i.e. after the **Length** packet).

Inherent in the Zigbee protocol is the possibility of multiple replies, most notably in the situation where data is returned but the ACK is not received by the sender. The sender, assuming that the data was not received, then sends the same data again. This can be done up to seven times, so in order to determine which data is redundant and which is new, the **TransID** field should be an auto-incremented, one byte number that is used to differentiate multiple replies in an interleaved command/reply application.

The **MSG Type** field determines what type of operation is to be performed or what data is being returned, i.e. Set Field, Get Field, Get NWK Address Reply, etc. The last section of the packet is the **Arguments** field and is the actual data used in performing the message function as shown below. (Refer to Table 7)

|                                    |                  |                 |                |               |               |             |
|------------------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                            | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | 1 byte        | Varies      |
| <b>MAC/NWK</b><br><b>Address**</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

**Note: All multi-byte fields are LSB (least significant byte) first.**  
**\*\* When talking to a local device, this value should be all zeros.**

The MAC/NWK address field is an 8-byte field that is used for either the 8-byte MAC Address or the 2-byte Network Address. To keep the packet format the same regardless of which address type is used, the field is always 8 bytes in length. Since the IEEE address prefix assigned to Cirronet is 00:30:66, when the MAC address type is used, the most significant byte will always be 00. Thus to indicate the use of the 2-byte network address, the MSB of the MAC/NWK field must be set to 80 with the network address in the 2 least significant bytes of the field. The other 5 bytes of the field should be set to 00.

The **ProfileID** allows for the future capability of communicating with devices running different profiles. For our purposes, this will always be the Cirronet Profile which has the identifier 0xC000.

The **Endpoint** is a logical device designator that is used to allow a single module to be treated as multiple logical devices. The CSM profile only supports a single Endpoint 0x01.

In order to accommodate different kinds of products there needs to be data that define the personality of every module when it is embedded into each kind of product. Therefore, it is necessary to define sets of data structures and functions that will be common to each wireless product. The data within them determines the specific functionality of each device. In ZigBee terminology, these structures are “Clusters” within which variables can be accessed and changed. Clusters can be thought of as groups of memory locations where configuration parameters are stored. Specific locations are specified by the **Cluster** type and an **Offset** into the cluster.

**Offset** is the index into an array of elements in the data field and can be found by referring to the particular **Cluster** table. It is a 2 byte field and as such, needs to be entered LSB first.

**Length** is the number of bytes contained in the Data section.

There is an additional parameter on replies: **LQI**. This is an indication of the quality of connection to the next radio along the routing path.

The **Cluster** variables are stored in NVRAM. There are six clusters supported in the CSM Profile: I/O, Configuration; Reset, Network, RF and Security. These clusters and the data in each are described in detail in later sections. Another advantage to using these clusters is that all Cirronet devices can be uniquely identified in a mixed vendor network. The CSM API supports the following groups of functions: network setup and discovery; module configuration; serial data communication; and I/O handling. Each functional group and their associated commands are described in detail in the following sections.

| Field     | Description   |
|-----------|---|
| SOP       | Signifies the beginning of the packet.<br>Value: 0xFD   |
| Length    | Number of bytes in the packet after the Length byte.  |
| TransID   | Used to differentiate multiple replies in an interleaved command/reply application.   |
| MSG Type  | Determines what type of operation is to be performed or what data is being returned <ul style="list-style-type: none"> <li>- 0x01: Set Field</li> <li>- 0x05: Get Field</li> <li>- 0x0A: Send String</li> <li>- 0x0C: Send SPI</li> <li>- 0x10: Get IEEE Address</li> <li>- 0x11: Get NWK Address</li> <li>- 0x64: Discovery Request</li> <li>- 0x65: Discovery End</li> <li>- 0x81: Set Reply</li> <li>- 0x85: Get Reply</li> <li>- 0x8A: Send String Reply</li> <li>- 0x8C: Send SPI Reply</li> <li>- 0x8E: Receive String</li> <li>- 0x90: Get IEEE Address Reply</li> <li>- 0x91: Get NWK Address Reply</li> <li>- 0x95: Receive Field Event</li> <li>- 0xD0: Link Announce</li> <li>- 0xE4: Discovery Reply</li> <li>- 0xF0: Device Registration</li> <li>- 0xFF: Error</li> </ul> |
| Arguments | Actual data used in performing the message function.  |



### 8.7.1 Set Field

**Message Type: Set Field 0x01**

Set Field writes the data value(s) to the specified cluster item(s). Acknowledgement is provided through a Set Reply message type.

| Arguments (Get/Set)             |                               |                             |                            |                           |                           |                              |
|---------------------------------|-------------------------------|-----------------------------|----------------------------|---------------------------|---------------------------|------------------------------|
| <b>MAC Address</b><br>(8 bytes) | <b>ProfileID</b><br>(2 bytes) | <b>Endpoint</b><br>(1 byte) | <b>Cluster</b><br>(1 byte) | <b>Offset</b><br>(2 byte) | <b>Length</b><br>(1 byte) | <b>[Data]</b><br>(LSB first) |

**MAC Address:** The destination address of the packet. If the destination is the device connected (local), then the MAC Address = 0x0000000000000000.

**ProfileID:** 0xC000

**Endpoint:** 0x01

**Cluster:** This is the cluster containing the field to be set.

**Offset:** This is the index into an array of elements in the data field. The data type is essentially encapsulated in the data itself. The number of bytes for each element is known apriori. This allows a list of elements to be read from or written to with one message.

**Length:** The number of bytes that follow.

**Data:** The data elements.

### 8.7.2 Set Reply

**Message Type: Set Reply 0x81**

Set Reply message type is sent in acknowledgement to a Set Field message type when successful. An Error Code message type is returned in the event of a Set Field failure.

| Arguments (Set Reply)       |                        |
|-----------------------------|------------------------|
| <b>Reserved</b><br>(1 byte) | <b>LQI</b><br>(1 byte) |

**Reserved:**

**LQI:** This is the Link Quality Indicator

### 8.7.3 Get Field

**Message Type:** 0x05

Get Field requests the value of a cluster item. The value is returned through a Get Reply message type.

| Arguments (Get)          |                        |                      |                     |                    |                    |
|--------------------------|------------------------|----------------------|---------------------|--------------------|--------------------|
| MAC Address<br>(8 bytes) | ProfileID<br>(2 bytes) | Endpoint<br>(1 byte) | Cluster<br>(1 byte) | Offset<br>(2 byte) | Length<br>(1 byte) |

**MAC Address:** The destination address of the packet. If the destination is the device connected (local), then the MAC Address = 0x0000000000000000.

**ProfileID:** 0xC000

**Endpoint:** 0x01

**Cluster:** This is the cluster containing the field to be set.

**Offset:** This is the index into an array of elements in the data field. The data type is essentially encapsulated in the data itself. The number of bytes for each element is known apriori. This allows a list of elements to be read from or written to with one message.

**Length:** The number of bytes that follow.

### 8.7.4 Get Reply

**Message Type:** 0x85

Get Reply message type returns the parameter value requested through a Get Field message type.

| Arguments (Get/Set Reply) |                        |                      |                     |                    |                    |                    |                 |
|---------------------------|------------------------|----------------------|---------------------|--------------------|--------------------|--------------------|-----------------|
| MAC Address<br>(8 bytes)  | ProfileID<br>(2 bytes) | Endpoint<br>(1 byte) | Cluster<br>(1 byte) | Offset<br>(2 byte) | Length<br>(1 byte) | [Data]<br>(Varies) | LQI<br>(1 byte) |

**MAC Address:** The address of the device which received the Get/Set command.

**ProfileID:** 0xC000

**Endpoint:** 0x01

**Cluster:** This is the cluster containing the field that was set.

**Offset:** Offset of data in cluster

**Length:** The number of bytes that follow.

**Data:** The string.

**LQI:** Link Quality Indicator

### 8.7.5 Send String

**Message Type:** 0x0A

Sends data string to be output of destination device's UART Receive Data signal. Successful transmission of data to destination device is acknowledged through a Send String Reply message type. Unsuccessful transmission will return an Error Code message type. Specifying the local device will cause the data string to be echoed back.

| Arguments (Send String)  |                        |                      |                     |                    |                    |                       |
|--------------------------|------------------------|----------------------|---------------------|--------------------|--------------------|-----------------------|
| MAC Address<br>(8 bytes) | ProfileID<br>(2 bytes) | Endpoint<br>(1 byte) | Cluster<br>(1 byte) | Offset<br>(2 byte) | Length<br>(1 byte) | [Data]<br>(LSB first) |

MAC Address: The destination address of the packet.  
 ProfileID: 0xC000  
 Endpoint: 0x01  
 Cluster: 0x01  
 Offset: 0x0011  
 Length: The number of bytes that follow.  
 Data: The string.

### 8.7.6 Send String Reply

**Message Type:** 0x8A

Acknowledges successful receipt and output of data string by the destination device. If the Send String was unsuccessful, an Error Code message type will be generated in place of the Send String Reply message.

| Arguments (Send String Reply) |                 |
|-------------------------------|-----------------|
| Reserved<br>(1 byte)          | LQI<br>(1 byte) |

Reserved: Reserved  
 LQI: Link Quality Indicator

### 8.7.7 Receive String

**Message Type: 0x8E**

Used to indicate data string in the Data field was sent by the device indicated in the MAC Address field using the Send String message.

| Arguments (Receive String)      |                               |                             |                            |                           |                           |                              |
|---------------------------------|-------------------------------|-----------------------------|----------------------------|---------------------------|---------------------------|------------------------------|
| <b>MAC Address</b><br>(8 bytes) | <b>ProfileID</b><br>(2 bytes) | <b>Endpoint</b><br>(1 byte) | <b>Cluster</b><br>(1 byte) | <b>Offset</b><br>(2 byte) | <b>Length</b><br>(1 byte) | <b>[Data]</b><br>(LSB first) |

MAC Address: The source address of the packet.  
 ProfileID: 0xC000  
 Endpoint: 0x01  
 Cluster: 0x01  
 Offset: 0x0011  
 Length: The number of bytes that follow.  
 Data: The string.

### 8.7.8 Send SPI

**Message Type: 0x0C**

Sends data, typically requests for data, out the SPI port of the destination device. When successful, the requested data is returned in a Send SPI Reply message type. If unsuccessful, and Error Code message in generate in its place.

| Arguments (Send SPI)            |                               |                             |                            |                           |                           |                              |
|---------------------------------|-------------------------------|-----------------------------|----------------------------|---------------------------|---------------------------|------------------------------|
| <b>MAC Address</b><br>(8 bytes) | <b>ProfileID</b><br>(2 bytes) | <b>Endpoint</b><br>(1 byte) | <b>Cluster</b><br>(1 byte) | <b>Offset</b><br>(2 byte) | <b>Length</b><br>(1 byte) | <b>[Data]</b><br>(LSB first) |

MAC Address: The destination address of the packet.  
 ProfileID: 0xC000  
 Endpoint: 0x01  
 Cluster: 0x01  
 Offset: 0x0010  
 Length: The number of bytes that follow.  
 Data: The bytes to be sent to the SPI port.

### 8.7.9 Send SPI Reply

**Message Type:** 0x8C

Returns the data requested from the SPI device connected to the destination device specified in the Send SPI command. When the local device is specified, the data will be from the SPI bus on the local module.

| Arguments (Send String)         |                               |                             |                            |                           |                           |                              |
|---------------------------------|-------------------------------|-----------------------------|----------------------------|---------------------------|---------------------------|------------------------------|
| <b>MAC Address</b><br>(8 bytes) | <b>ProfileID</b><br>(2 bytes) | <b>Endpoint</b><br>(1 byte) | <b>Cluster</b><br>(1 byte) | <b>Offset</b><br>(2 byte) | <b>Length</b><br>(1 byte) | <b>[Data]</b><br>(LSB first) |

MAC Address: The address of the device which received the Send String command.  
 ProfileID: 0xC000  
 Endpoint: 0x01  
 Cluster: 0x01  
 Offset: 0x0010  
 Length: Send SPI Reply will return the same number of bytes as were sent with the Send SPI command.  
 Data: The data elements read back from the attached SPI device.

### 8.7.10 Get IEEE Address

**Message Type:** 0x10

Requests the 8-byte unique IEEE MAC Address of the device specified by the 2-byte network address. When Request Type is 0x00, requests only the MAC address of the device specified by the network address. When Request Type is 0x01, requests the MAC address only of the device specified in the Network Address field and only the Network Addresses of devices associated with the device specified by the network address. This mode is used to request the Network Address of devices that have joined the network through a Router. To obtain the MAC Addresses of the associated devices, a Get IEEE Address command must be issued for each associated device individually.

**Note: The IEEE Address returned will not be correct if the NWK Address values specified in the Get IEEE Address and the Get IEEE Address Reply don't match. This transaction may also take up to 3 seconds to complete.**

| Arguments (Get IEEE Address) |   |                         |
|------------------------------|---|-------------------------|
| Network Address<br>(2 bytes) | Request Type<br>(1 byte)  | Start Index<br>(1 byte) |
| Network Address:             | The Network address for which the IEEE Address is being requested.  |                         |
| Request Type:                | This specifies the kind of data returned in the response: <ul style="list-style-type: none"> <li>• 0x00: Only the requested MAC Address</li> <li>• 0x01: The requested MAC Address and Network Address of Associated Devices.</li> </ul>  |                         |
| Start Index:                 | If associated devices are requested, then this is the index to the list to start returning. This is used on multiple requests to retrieve the list if it is larger than a ZigBee packet allows. Set to 0x00 if the list is not requested. |                         |

### 8.7.11 Get IEEE Address Reply

**Message Type: 0x90**

Returns the IEEE MAC address of the device(s) specified through a Get IEEE Address command. When the Get IEEE Address command uses Request Type 0x01, will return the MAC Address and Network Number of devices associated with the device specified in the Network Address field. To obtain the MAC Addresses of the associated devices, a Get IEEE Address command must be issued for each associated device individually.

**Note: The IEEE Address returned will not be correct if the NWK Address values specified in the Get IEEE Address and the Get IEEE Address Reply don't match. This transaction may also take up to 3 seconds to complete.**

| Arguments (Get IEEE Address Reply) |                               |                         |  |   |
|------------------------------------|-------------------------------|-------------------------|--|---|
| Network Address<br>(2 bytes)       | IEEE MAC Address<br>(8 bytes) | Start Index<br>(1 byte) | Number of Associated Devices<br>(1 byte) | List of Network Addresses<br>(variable) |

|                               |  |
|-------------------------------|--|
| Network Address:              | The Network address for which the IEEE Address has been requested.   |
| MAC Address:                  | The 8-byte address that corresponds to the network address.  |
| Start Index:                  | If associated devices are requested, then this is the index to the list that is being returned. This is used on multiple requests to retrieve the list if it is larger than a ZigBee packet allows.                      |
| Number of Associated Devices: | This number represents the number of devices that have associated with the device that has the accompanying MAC address. It is also the length of the list that follows. 0x00 if not requested or no associated devices. |
| List of Network Addresses:    | This is a list of 16-bit Network Addresses that is Number of Associated Devices in length. NULL if not requested or Number of Associated Devices is 0.   |

### 8.7.12 Get NWK Address

**Message Type:** 0x11

Requests the 2-byte unique ZigBee Network Address of the device specified by the 8-byte IEEE address. When Request Type is 0x00, requests only the network address of the device specified by the IEEE address. When Request Type is 0x01, requests the network address of the device specified in the IEEE Address field and the Network Addresses of devices associated with the device specified by the IEEE address. This mode is used to request the Network Address of devices that have joined the network through a Router. To obtain the MAC Addresses of the associated devices, a Get IEEE Address command must be issued for each associated device individually.

**Note:** *The MAC Address returned will not be correct if the NWK Address values specified in the Get NWK Address and the Get NWK Address Reply don't match. This transaction may also take up to 3 seconds to complete.*

| Arguments (Get NWK Address) |                          |                         |
|-----------------------------|--------------------------|-------------------------|
| IEEE Address<br>(8 bytes)   | Request Type<br>(1 byte) | Start Index<br>(1 byte) |

IEEE Address: The IEEE address for which the Network Address is being requested.

Request Type: This specifies the kind of data returned in the response:

- 0x00: Only the requested NWK Address
- 0x01: The requested NWK Address and Network Address of Associated Devices.

Start Index: If associated devices are requested, then this is the index to the list to start returning. This is used on multiple requests to retrieve the list if it is larger than a ZigBee packet allows. Set to 0x00 if the list is not requested.



### 8.7.13 Get NWK Address Reply

**Message Type:** 0x91

Returns the ZigBee network address of the device(s) specified through a Get NWK Address command. When the Get NWK Address command uses Request Type 0x01, will return the Network Address of the specified device and Network Address of any devices associated with the device specified in the Network Address field. To obtain the MAC Addresses of the associated devices, a Get IEEE Address command must be issued for each associated device individually.

**Note: The MAC Address returned will not be correct if the NWK Address values specified in the Get NWK Address and the Get NWK Address Reply don't match. This transaction may also take up to 3 seconds to complete.**

| Arguments (Get NWK Address Reply) |                               |   |   |
|-----------------------------------|-------------------------------|---|---|
| Network Address<br>(2 bytes)      | IEEE MAC Address<br>(8 bytes) | Number of Associated Devices<br>(2 bytes) | List of Network Addresses<br>(variable) |

|                               |  |
|-------------------------------|--|
| Network Address:              | The 2-byte network address that corresponds to the IEEE address.   |
| MAC Address:                  | The IEEE Address for which the Network Address has been requested.   |
| Number of Associated Devices: | This number represents the number of devices that have associated with the device that has the accompanying MAC address. It is also the length of the list that follows. 0x00 if not requested or no associated devices. |
| List of Network Addresses:    | This is a list of 2-byte Network Addresses that is Number of Associated Devices in length. NULL if not requested or Number of Associated Devices is 0.   |

### 8.7.14 Discovery Request

**Message Type:** 0x64

The following Figures and Tables all pertain to the Device Discovery process. In order to discover other devices that can communicate, a Discovery Request packet is issued. All the devices that match the profile of the Coordinator will respond with a **Discovery Reply**. This will continue until the timeout duration expires. Then a **Discovery End** packet will be returned, containing the number of replies that were received. New Discovery Request packets may not be issued until a **Discovery End** packet has been received. Once a list of network addresses has been built, a Get IEEE Address packet must be issued for each one in the list to obtain the IEEE MAC Address. Due to the amount of module memory needed for this command when used with large networks, it is recommended for use only in small networks. For larger networks, the Get NWK Address and Get IEEE Address commands should be used. See Section 8.10 for details.

| Arguments (Discovery Request) |  |                      |                     |
|-------------------------------|--|----------------------|---------------------|
|                               | ProfileID<br>(2 bytes)   | Endpoint<br>(1 byte) | Timeout<br>(1 byte) |
| ProfileID:                    | 0xC000   |                      |                     |
| Endpoint:                     | 0x01   |                      |                     |
| Timeout:                      | The amount of time, in seconds, to wait for discovery replies. 0x00 – 0x3C |                      |                     |

### 8.7.15 Discovery Reply

**Message Type:** 0xE4

Generated for each device that is Discovered, i.e., that matches the clusters of the Coordinator within the timeout period specified in the Discover command.

| Arguments (Discovery Reply) |  |
|-----------------------------|--|
|                             | Network Address<br>(2 bytes)   |
| Network Address:            | This is the 16-bit address assigned by the ZigBee Coordinator. It is used for indirect addressing. |

### 8.7.16 Discovery End

**Message Type:** 0x65

Indicates the end of the timeout period specified in the Discover command.

| Arguments (Discovery End) |   |
|---------------------------|---|
|                           | Number of Replies<br>(2 bytes)                                    |
| Number of Replies:        | Returns the number of discovery replies that the module received. |

### 8.7.17 Receive Field Event

**Message Type: 0x95**

Used to indicate messages generated by field devices and not in response to a Coordinator initiated message if the Router/End Device is in Transparent Mode.

| Arguments (Receive Field Event) |                        |                      |                     |                    |                    |                       |
|---------------------------------|------------------------|----------------------|---------------------|--------------------|--------------------|-----------------------|
| MAC Address<br>(8 bytes)        | ProfileID<br>(2 bytes) | Endpoint<br>(1 byte) | Cluster<br>(1 byte) | Offset<br>(2 byte) | Length<br>(1 byte) | [Data]<br>(LSB first) |

MAC Address: The address of the device which received the Get/Set command.  
 ProfileID: 0xC000  
 Endpoint: 0x01  
 Cluster: This is the cluster containing the field that was set.  
 Offset: This is the index into an array of elements in the data field. .  
 Length: The number of bytes that follow.  
 Data: The data elements.

### 8.7.18 Link Announce

**Message Type: 0xD0**

This is an unsolicited event that happens whenever a device associates with the network. The local device will output this message to indicate that it has either started a network (Coordinator) or associated with another device (Router or end-device). Only the values in italics will be used during LinkAnnounce. When LinkStatus (Cluster 0x07, Offset 0x0015) is queried, the status will be one any one of the enumerated values below.

| Arguments (Error)      |                 |
|------------------------|-----------------|
| LinkStatus<br>(1 byte) | LQI<br>(1 byte) |

Link Status: Description of values returned

- 0x01: Device is initialized but not connected
- 0x02: Device is discovering PANs to join
- 0x03: Device is joining PAN
- 0x04: *Device has joined but is not yet authenticated by the Trust Center*
- 0x05: *Device has been authenticated and has joined the network as an End Device*
- 0x06: *Device has been authenticated and has joined the network as a Router*
- 0x07: Device is starting a network
- 0x08: *Device has started a network as the Coordinator*
- 0x09: Device has been orphaned

LQI: Indication of the quality of connection to the next radio along the routing path

### 8.7.19 Device Registration

**Message Type:**     **0xF0**

This is an unsolicited event that happens whenever a device associates with the network. It allows the gateway (default is Coordinator) to be aware of any changes in network address for a given device. It also includes information about the type of device being registered. This allows a gateway device to understand each device type communicating with it.

| Arguments (Receive Field Event) |                                 |                                |
|---------------------------------|---------------------------------|--------------------------------|
| Network Address<br>(2 byte)     | IEEE Address<br>(8 bytes)       | Model Number<br>(2 bytes)      |
| Parent NWK #<br>(2 byte)        | Parent MAC Address<br>(8 bytes) | Check Parent Rate<br>(2 bytes) |

### 8.7.20 Error

**Message Type:**     **0xFF**

Returns Error Code when needed.

| Arguments (Error)      |                 |
|------------------------|-----------------|
| Error Code<br>(1 byte) | LQI<br>(1 byte) |

**Error Code:**     Code that designates the type of error that occurred.

- 0x01: Timeout on last Get/Set
- 0x02: Bad Packet Type
- 0x03: Unknown Error
- 0x04: Unsupported Command
- 0x05: Invalid Endpoint, Cluster, FOffset or FLength
- 0x06: Data Value out of Range
- 0x07: Not Enough Memory
- 0x08: Permission Error

**LQI:**             Indication of the quality of connection to the next radio along the routing path

## 8.8 CSM Sleep Modes

The CSM profile provides two End Device sleep modes: 1) Timer Sleep; and 2) Interrupt Sleep. The details of each mode are described below.

### 8.8.1 Timer Sleep Mode

In Timer Sleep, the module will wake up periodically to check if its parent has data for it and, optionally, wake up and send the contents of its Module IO cluster. The rate at which the module wakes to check for data from its parent is set by the Check Parent Interval parameter. The default value is once a second and can range from 1 millisecond to 65.5 seconds. Setting this parameter to a value less than 50 milliseconds will result in a higher current consumption than if the End Device was awake all of the time as the module will be continually sending requests for data to its parent.

If Reporting Mode has been enabled, then the module will wake every Reporting Interval amount of time and send the values in the Module I/O cluster to the coordinator. Note that this feature is available even if the End Device is in mains powered mode and on routers as well. This interval can range from 1 second to 49.7 days with an accuracy of the smaller of the Check Parent Interval or 500 milliseconds. The module will remain awake for the length of time specified in the Wake Duration parameter which can range from 1 millisecond to 65.5 seconds.

Any of GPIO0 – 3 enabled as an interrupt can be used to wake the module in Timer Sleep mode and more than one interrupt can be enabled at the same time. The module will remain awake as long as the interrupt is true. In addition to waking the module, the interrupt can cause one of three messages to be sent to the coordinator. The messages available are the state of the interrupt that caused the module to wake up (always zero since the module wakes on falling edge interrupts), the contents of the Module I/O cluster and a Device Announcement. The default is to send a message and the default message is the interrupt signal state message.

Data sent to the module on the UART port while the module is sleeping will be lost. If the module wakes in the middle of a transmission of data on the UART port, only that portion of the data that is sent when the module is awake will be received by the module.

### 8.8.2 Interrupt Sleep Mode

In Interrupt Sleep, the module only wakes up when an enabled interrupt is triggered. The module will not check its parent for data until it has been awakened by an external interrupt nor will it obey the Reporting Interval. While the module is being kept awake by the interrupt line, it will check its parent for data based on the Check Parent Interval parameter but the timer does not start until the module wakes up. Thus, if the Check Parent Interval is set for 50 milliseconds, the module will first check its parent for data 50 milliseconds after it is awakened by an interrupt. It will continue to check its parent for data every 50 milliseconds as long as the module is awake. The message functions described in the preceding section are still functional in Interrupt Sleep mode.

If data is being sent to the module through the UART port the interrupt must be held low (true) until all data has been sent to the module. Incoming data on the module's UART port will not keep the module awake.

### 8.8.3 Configuring Sleep Mode

When an end device joins a network, it informs the coordinator as to whether it is configured as mains powered or battery powered. The coordinator stores this information and does not change it even if the power source mode of the end device is changed. Therefore end devices should already be configured for the appropriate power source setting before being introduced into the network. If the power source mode setting must be changed after a device has joined the network, it is necessary to reset the coordinator in order to have the change registered in the coordinator.

Because the network handles data requests differently for sleeping end devices than it does for mains powered devices, an incorrect setting in the coordinator will cause problems in the network.

### 8.8.4 Timer Sleep Example

An application has a need to receive ADC values from battery-operated devices every 15 minutes. It is simpler for the application program to have the nodes wake-up on their own and report their data than it is for the application program to poll each device.

To implement this behavior in an End Device several parameters in the Configuration cluster must be set. (See Section 8.2) The End Device must have *Sleep Mode* enabled as well as the *Reporting Mode* enabled. Set the *Reporting Rate* to 15 minutes by writing `0x000DBBA0` (15 minutes = 900 000 milliseconds). Since the application is not polling the nodes, the *Check Parent* time can be set to the maximum of 65.5 seconds to minimize power consumption. (ZigBee does not allow this function to be disabled in Timer Sleep Mode.) This is accomplished by writing `0xFFFF` to the *Check Parent Rate* location in the Configuration cluster.

Every 15 minutes each node will send an *Event* packet to the coordinator with a data payload of the first 16 bytes of the Module I/O cluster. The value of ADC X will be the first two bytes, ADC Y the third and fourth bytes, and ADC Z the fifth and sixth bytes. If it is desired to have the module stay awake long enough to receive some application program reply to the data it sends (any ZigBee retries and acknowledgments are automatic and transparent to the application) the appropriate value needs to be written to the *Wake Duration* parameter in the Configuration cluster.

### 8.8.5 Interrupt Sleep Example

In this example it is desired to have battery-powered nodes sleep until a switch is tripped. When the switch is tripped, the values in the Module I/O cluster need to be sent to the Coordinator. To implement this type of module behavior, it is necessary to enable *Sleep Mode* and *Interrupt Sleep*. GPIO1 is selected to be used as the interrupt and thus must be enabled as an interrupt through bits 2 and 3 of the *GPIO Interrupts* in the Module I/O cluster. If no other GPIOs are to be enabled as interrupts, `0x0c` should be written to the *GPIO Interrupts* location. To have the first 16 bytes of the *Module I/O* cluster sent to the Coordinator when GPIO1 is driven low, the *GPIO1 Message Enable* must be enabled by setting bit 5 of the *Option Settings 1* of the *Configuration* cluster to 1. Since this location also controls enabling *Interrupt Sleep*, both functions can be enabled in a single step. Since *Options Settings 1* controls multiple settings, care must be taken to make sure other settings are not inadvertently changed. One way to do this is to first read the *Options Settings 1* value and then OR'ing that value with `0x24` and then writing the resulting byte back to the *Options Settings 1* location. *Sleep Mode* is enabled by writing `0x01` to that location in the *Configuration* cluster.

To select the *Module I/O* cluster information to be sent (instead of the GPIO1 status or a Registration Packet) `0x04` should be written to the *Message Options* location in the *Configuration* cluster. Similar to the above, if other GPIO lines are to be used as interrupts, care must be taken not to inadvertently overwrite the bits associated with them in the *Message Options* location.

## 8.9 Network Discovery

While not outside knowledge of the network is required for a ZigBee network to form, it is necessary for an application program to know the network address assigned to each device in order to address data. There are three ways to accomplish this task, each with its pluses and minuses. Each method is described below.

### 8.9.1 Device Registration Packets

When a device joins the network, the Coordinator will send a Device Registration message out its UART. The Device Registration message includes the MAC address of the device joining and the MAC address and Network Address of its parent. If it has joined directly with the Coordinator, the MAC Address and Network Address of its parent will be those of the Coordinator. The application program will need to build a table based on these Device Registration messages keeping in mind that devices can join the network at any time.

In order to learn the Network Address of the device that is joining the network, the Get NWK Address command must be issued using the MAC address of that device. The application program must add the Network Address in the table since ZigBee uses Network Addresses to send data to specific devices.

In addition, the registration packet contains the Model Number of the device and the Check Parent Rate for the device. The Model Number indicates whether it is a Coordinator, a Router, an End Device, or a Sleeping End Device. If it is a Sleeping End Device then the Check Parent Rate indicates how often the Sleeping End Device is waking and checking its parent for data. This is useful to the application program to determine how long it should wait for a response from that device before deciding the device did not answer.

While this approach has the benefit of the application program merely reacting to data received from the Coordinator, the weakness is that if the application program misses a registration packet, it will have no knowledge of that device's presence in the network.

### 8.9.2 Discovery Command

When the Discovery Command is issued a message is sent to all nodes on the network asking them to identify themselves. All the devices that receive this message respond with their 2-byte Network Address. Using this method, an application would need to build a table of Network Addresses and then use that table to request the IEEE Address of each device. This will return the MAC address associated with the specified Network Address. When a Reply Type of 1 is specified in the Get IEEE command, the data returned will include the number of devices associated with that device and the Network Addresses of each associated device. It will then be necessary to add those Network Addresses to the table and issue a Get IEEE command for each of them.

This approach has the application program issuing a single command to discover all of the devices. The weakness is that with large networks the number of devices responding will overwhelm the memory on the coordinator causing some responses to be lost.

### 8.9.3 Hierarchical Discovery

The third method is the most manual but probably the most reliable. This method consists of using the Get IEEE Address command and “walking” down the network. Regardless of on which node this process is performed, start by issuing this command for the Network Address of the Coordinator which is always 0x0000. Issue the command with Reply Type 1 to get the Network Addresses of all of the devices directly associated with the Coordinator. Keep issuing this command using the Network Addresses received from previous command replies until the entire network is discovered.

The strength of this method is the high reliability – the application program is in control. The weakness is the manual nature of the approach.



## 8.10 Sample Packets

### 8.10.1 RF Channel List Example

The RF Channel List uses a 32 bit number in the data field using the Offset field value 0x0000 of the RF Cluster, 0x08. The format of the number is shown in the diagram below.

|          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |        |        |        |        |        |        |        |        |        |        |        |
|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Reserved | Reserved | Reserved | Reserved | Reserved | 2480MHz | 2475MHz | 2470MHz | 2465MHz | 2460MHz | 2455MHz | 2450MHz | 2445MHz | 2440MHz | 2435MHz | 2430MHz | 2425MHz | 2420MHz | 2415MHz | 2410MHz | 2405MHz | 922MHz | 920MHz | 918MHz | 916MHz | 914MHz | 912MHz | 910MHz | 908MHz | 906MHz | 902MHz | 869MHz |
| 31       | 30       | 29       | 28       | 27       | 26      | 25      | 24      | 23      | 22      | 21      | 20      | 19      | 18      | 17      | 16      | 15      | 14      | 13      | 12      | 11      | 10     | 9      | 8      | 7      | 6      | 5      | 4      | 3      | 2      | 1      | 0      |
| 0        | 0        | 0        | 0        | 0        | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 1       | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |        |
|          |          |          |          | 7        |         |         |         | F       |         |         |         | F       |         |         |         | F       |         |         |         | 8       |        |        |        | 0      |        |        |        | 0      |        |        |        |
|          |          |          |          |          |         |         |         |         |         |         |         | Hex     |         |         |         |         |         |         |         |         |        |        |        |        |        |        |        |        |        |        |        |

The example above enables all 2.4GHz channels for possible use by entering 0x07FFF800 into the data field. As an example, to turn on channels 2415MHz, 2425MHz, 2430MHz, 2440MHz, 2450MHz, and 2465MHz, you would setup the following diagram.

|          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |        |        |        |        |        |        |        |        |        |        |        |
|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Reserved | Reserved | Reserved | Reserved | Reserved | 2480MHz | 2475MHz | 2470MHz | 2465MHz | 2460MHz | 2455MHz | 2450MHz | 2445MHz | 2440MHz | 2435MHz | 2430MHz | 2425MHz | 2420MHz | 2415MHz | 2410MHz | 2405MHz | 922MHz | 920MHz | 918MHz | 916MHz | 914MHz | 912MHz | 910MHz | 908MHz | 906MHz | 902MHz | 869MHz |
| 31       | 30       | 29       | 28       | 27       | 26      | 25      | 24      | 23      | 22      | 21      | 20      | 19      | 18      | 17      | 16      | 15      | 14      | 13      | 12      | 11      | 10     | 9      | 8      | 7      | 6      | 5      | 4      | 3      | 2      | 1      | 0      |
| 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 1       | 0       | 0       | 1       | 0       | 1       | 0       | 1       | 1       | 0       | 1       | 0       | 0       | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |        |
|          |          |          |          | 0        |         |         |         | 9       |         |         |         | 5       |         |         |         | A       |         |         |         | 0       |        |        |        | 0      |        |        |        | 0      |        |        |        |
|          |          |          |          |          |         |         |         |         |         |         |         | Hex     |         |         |         |         |         |         |         |         |        |        |        |        |        |        |        |        |        |        |        |

To turn on the channels above, enter 0x0095A000 (remember to enter the LSB first) into the data field using Offset value 0x0000.

Let's look at how the packet to enable the channels in the above example would be constructed. Here is our packet format.

|                   |                          |                |                 |                  |
|-------------------|--------------------------|----------------|-----------------|------------------|
| 1 byte            | 1 byte                   | 1 byte         | 1 byte          | Varies           |
| <b>SOP (0xFD)</b> | <b>Length (in bytes)</b> | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD. **Length** must wait until the whole packet is constructed before it can be determined. Since this is our first packet, our **TransID** will be 0x00. Because we are going to set the RF Channel List, our **MSG Type** is going to use the Set Field command or 0x01 so our packet so far looks like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | TBD           | 00             | 01              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

Let's go onto the Argument field which is made up of the fields below.

|                        |                  |                 |                |               |               |             |
|------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           | Varies      |
| <b>MAC/NWK Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Because our MAC/NWK Address field is a multi-byte field, the data must be entered LSB first. In this example, we will use a network address . Since the local device is a Coordinator; its network address is 00 80. That makes our first field, 00 00 00 00 00 00 00 80. Remember, when using a network address, set the MSB to 0x80. As we found out earlier, our **ProfileID** is 0xc000 and our **Endpoint** for current Cirronet products is 0x01. Looking at our **Cluster** tables we find that the RF Channel List is in the RF Cluster which is 0x08 and looking down the table to RF Channel List we see that our **Offset** is 0x0000. Its **Length** is 4 bytes or 0x04 so from our example above, if we want to enable channels 2415MHz, 2425MHz, 2430MHz, 2440MHz, 2450MHz, and 2465MHz, the data field would need to contain 0x0095A000. Loading these values into our packet structure for the **Arguments** field looks like this.

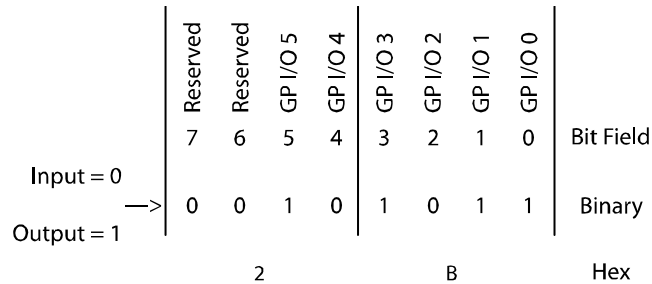
|                         |                  |                 |                |               |               |             |
|-------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 00 00 00 00 00 00 00 80 | 00 c0            | 01              | 08             | 00 00         | 04            | 00 0A 95 00 |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Now we can count up the number of bytes for our initial **Length** field which would be 4 bytes of **Data**, 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **Profile ID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the **TransID** for a total of 21 bytes or 0x15 which we would put in the **Length** field as shown below and the entire packet will look like this.

|                         |                  |                 |                |                 |               |             |
|-------------------------|------------------|-----------------|----------------|-----------------|---------------|-------------|
|                         | FD               | 15              | 00             | 01              |               |             |
|                         | <b>SOP</b>       | <b>Length</b>   | <b>TransID</b> | <b>MSG Type</b> |               |             |
| 00 00 00 00 00 00 00 80 | 00 c0            | 01              | 08             | 00 00           | 04            | 00 0A 95 00 |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b>   | <b>Length</b> | <b>Data</b> |

### 8.10.2 GP I/O Direction Example

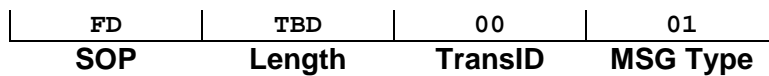
How do you set GP I/O 0 as an Output, GP I/O 1 as an Output, GP I/O 2 as an Input, GP I/O 3 as an Output, GP I/O 4 as an Input, and GP I/O 5 as an Output? The example below demonstrates the procedure for setting up the bit mapping field and the hexadecimal value that results.



Here is the packet format.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | Varies           |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD. **Length** must wait until the whole packet is constructed before it can be determined. For this example, **TransID** will be 0x00. Because we are going to set the GP I/O Direction, our **MSG Type** is going to use the Set Field command or 0x01 so the packet so far looks like this.



Let's go onto the Argument field which is made up of the fields below.

|                        |                  |                 |                |               |               |             |
|------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           | Varies      |
| <b>MAC/NWK Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Because our MAC/NWK Address field is a multi-byte field, the data must be entered LSB first. In this example, we will use a network address . Since the local device is a Coordinator; its network address is 00 80. That makes our first field, 00 00 00 00 00 00 80. Remember, when using a network address, set the MSB to 0x80. As we found out earlier, our **ProfileID** is 0xc000 and our **Endpoint** for current Cirronet products is 0x01. Looking at our **Cluster** tables we find that the GP I/O Direction is in the Module I/O Cluster which is 0x01 and looking down the table to GP I/O Direction we see that the **Offset** is 0x0012. Its **Length** is 1 byte or 0x01 so from our example above, if we want to set GP I/O 0 as an Output, GP I/O 1 as an Output, GP I/O 2 as an Input, GP I/O 3 as an Output, GP I/O 4 as an Input, and GP I/O 5 as an Output, the data field would need to contain 0x002A.

Loading these values into our packet structure for the **Arguments** field looks like this.

|                            |                  |                 |                |               |               |             |
|----------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 00 00 00 00 00 00 00 80    | 00 C0            | 01              | 01             | 12 00         | 01            | 2A          |
| <b>MAC/NWK<br/>Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Now we can count up the number of bytes for our initial **Length** field which would be 1 byte of **Data**, 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **ProfileID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the **TransID** for a total of 18 bytes or 0x12 which we would put in the **Length** field as shown below and the entire packet will look like this.

|                            |                  |                 |                |                 |               |             |
|----------------------------|------------------|-----------------|----------------|-----------------|---------------|-------------|
|                            | FD               | 12              | 00             | 01              |               |             |
|                            | <b>SOP</b>       | <b>Length</b>   | <b>TransID</b> | <b>MSG Type</b> |               |             |
| 00 00 00 00 00 00 00 80    | 00 C0            | 01              | 01             | 12 00           | 01            | 2A          |
| <b>MAC/NWK<br/>Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b>   | <b>Length</b> | <b>Data</b> |

### 8.10.3 Microcontroller Reset

In order to reset the microcontroller in the module, a value of 0x5A must be written to the radio. The example below demonstrates the procedure for using the Reset Cluster to accomplish the task. Here is the packet format.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | Varies           |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD. **Length** must wait until the whole packet is constructed before it can be determined. For this example, **TransID** will be 0x00. Because we are going to reset the microcontroller, our **MSG Type** is going to use the Set Field command or 0x01 so the packet so far looks like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | TBD           | 00             | 01              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

Let's go onto the Argument field which is made up of the fields below.

|                        |                  |                 |                |               |               |             |
|------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           | Varies      |
| <b>MAC/NWK Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Because our MAC/NWK Address field is a multi-byte field, the data must be entered LSB first. In this example, we will use a network address . Since the local device is a Coordinator; its network address is 00 00. That makes our first field, 00 00 00 00 00 00 00 80. Remember, when using a network address, set the MSB to 0x80. As we found out earlier, our **Endpoint** for current Cirronet products is 0x01 and our **ProfileID** is 0xc000. Looking at our **Cluster** tables we find that the Microcontroller Reset is in the Reset Cluster which is 0x03 and looking down the table to Microcontroller Reset we see that the **Offset** is 0x0000 and its **Length** is 1 byte or 0x01. The data field would need to contain 0x5A. Loading these values into our packet structure for the **Arguments** field looks like this.

|                         |                  |                 |                |               |               |             |
|-------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 00 00 00 00 00 00 00 80 | 00 C0            | 01              | 03             | 00 00         | 01            | 5A          |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

Now we can count up the number of bytes for our initial **Length** field which would be 1 byte of **Data**, 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **ProfileID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the **TransID** for a total of 18 bytes or 0x12 which we would put in the **Length** field as shown below and the entire packet will look like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | 12            | 00             | 01              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

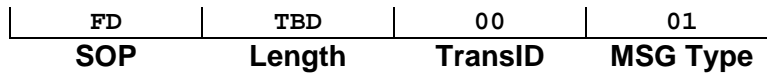
|                         |                  |                 |                |               |               |             |
|-------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 00 00 00 00 00 00 00 80 | 00 C0            | 01              | 03             | 00 00         | 01            | 5A          |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

### 8.10.4 UART Port/Send ASCII Data Example:

Let's look at constructing a packet to set the I/O operation to send ASCII data, a parameter labeled "UART Port". We'll start with our packet format model as shown below.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | Varies           |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD and as in our previous example, **Length** must wait until the whole packet is constructed. We will assume this is our first packet again which will make our **TransID** 0x00. Because we are going to set the UART Port, our **MSG Type** is going to use the Set Field command or 0x01 so our packet so far looks like this.



The **Argument** field is made up of the fields below.

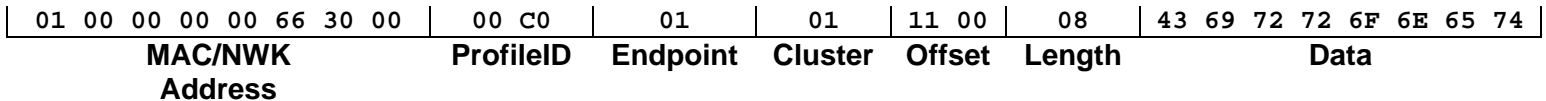
|                        |                  |                 |                |               |               |             |
|------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           | Varies      |
| <b>MAC/NWK Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

For this example, we do not want to enable mesh networking and use MAC addresses instead of Network addresses. Since the MAC addresses for all Cirronet devices start with 00:30:66, let's make the MAC address 00:30:66:00:00:00:00:01. That makes our first field, 01 00 00 00 00 66 30 00.

As we found out earlier, our ProfileID is 0xc000 and our **Endpoint** for the CSM Profile is 0x01. Looking at our **Cluster** tables we find that the UART Port is in the Module I/O Cluster which is designated by 0x01. Looking down the table to UART Port we see that our **Offset** is 0x0011.

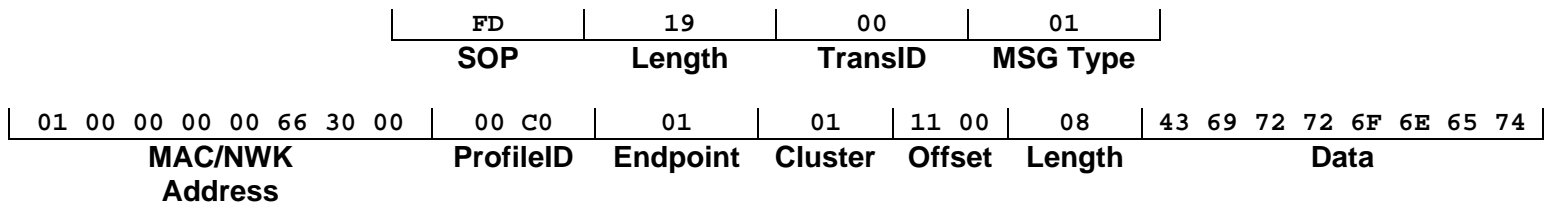
The **Length** depends on the Data field so let's say we want to send the word "Cirronet". The hexadecimal value of "C" is 0x43, "i" is 0x69, "r" is 0x72 (repeated twice), "o" is 0x6F, "n" is 0x6E, "e" is 0x65 and "t" is 0x74. Even though this is a multi-byte field, it is a "string" and strings are the exception to the rule, i.e. the data is entered exactly as it is sent. So the data field would need to contain 0x436972726F6E6574. Since our data consists of 8 bytes, our Length field will be 0x08.

Loading these values into our packet structure for the **Arguments** field looks like this.



Now we can count up the number of bytes for our initial **Length** field which would be 8 bytes of **Data**, 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **ProfileID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the

**TransID** for a total of 25 bytes or 0x19 which we would put in the **Length** field as shown below and the entire packet will look like this.



### 8.10.5 ADC Z Example: - Get Field

One more example we can do is to construct a packet to retrieve the on board 10-bit A to D converter value for channel 2. We'll start with our packet format model as shown below.

|                      |                             |                |                           |                  |  |
|----------------------|-----------------------------|----------------|---------------------------|------------------|--|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte                    | Varies           |  |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG</b><br><b>Type</b> | <b>Arguments</b> |  |

**SOP** is 0xFD and as in our previous example and the value in the **Length** field must wait until the whole packet is constructed. For simplicity's sake, we will again assume this is our first packet and make our **TransID** 0x00. Because we are requesting a value to be returned, our **MSG Type** is going to use the Get Field command or 0x05, so our packet so far looks like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | TBD           | 00             | 05              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

The **Argument** field is made up of the fields below. As you can see, there is a **Length** field but no **Data** field because we are requesting a value of a certain length be returned.

|                                  |                  |                 |                |               |               |
|----------------------------------|------------------|-----------------|----------------|---------------|---------------|
| 8 bytes                          | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           |
| <b>MAC/NWK</b><br><b>Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> |

For this example, we again do not want to use mesh networking and will use MAC addresses instead of Network addresses. Remember, MAC addresses can only be used for the local device or for a device directory associated with the local device. For devices that are not directly associated with the local device network addresses must be used. Since the MAC addresses for all Cirronet devices start with 00:30:66, let's make the MAC address 00:30:66:00:00:00:00:02. Putting LSB first, that makes our first field, 02 00 00 00 00 66 30 00.

As we found out earlier, our ProfileID is 0xc000 and our **Endpoint** for current Cirronet Zigbee products is 0x01. Looking at our **Cluster** tables we find that the ADC Z parameter is in the Module I/O Cluster which is designated by 0x01. Looking down the table to ADC Z, we see that our **Offset** is 0x0004 and since this is a multi-byte field the LSB must go first so we would enter 0x0400.

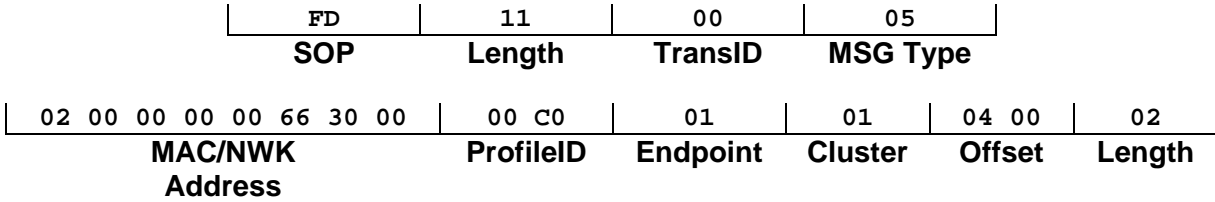
The **Length** field needs to say how many bytes are being requested, so by looking at the description of ADC Z parameter, we see that the ADC is a 10-bit device. The returned **Data** packet will require two bytes to hold the value which makes the **Length** field requested 0x02.

Loading these values into our packet structure, the **Arguments** field looks like this.

|                                  |                  |                 |                |               |               |
|----------------------------------|------------------|-----------------|----------------|---------------|---------------|
| 02 00 00 00 00 66 30 00          | 00 c0            | 01              | 01             | 04 00         | 02            |
| <b>MAC/NWK</b><br><b>Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> |



Now we can count up the number of bytes for our initial **Length** field which would be 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **ProfileID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the **TransID** for a total of 17 bytes or 0x11 which we would put in the **Length** field and the entire packet will then look like this.



### 8.10.6 ADC Z Example: - Get Reply

Now that we've sent a request, the Get Reply message type returns the parameter value requested by the Get Field message. For reference, here is our packet format model again.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | Varies           |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD and since this is a Reply, **Length** will have the appropriate value when received. Because this is in response to the Get Field request, our **TransID** must match the request so its value needs to be 0x00. We are returning a value and our **MSG Type** is going to use the Get Reply command (0x85) so our packet now looks like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | TBD           | 00             | 85              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

The **Argument** field is made up of the fields below. Notice now that we will be receiving the **Data** field with the value being returned.

|                        |                  |                 |                |               |               |             |
|------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|
| 8 bytes                | 2 bytes          | 1 byte          | 1 byte         | 2 bytes       | TBD           | Varies      |
| <b>MAC/NWK Address</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> |

The first six fields, **MAC/NWK Address**, **Profile ID**, **Endpoint**, **Cluster**, **Offset** and **Length** will be identical, but the **Data** field will contain the value of the A to D Converter.

For the purpose of this example, we will assign the value that ADC Z returns to 0x02c9 and our LQI is FC. The **Arguments** field will then look like this.

|                         |                  |                 |                |               |               |             |            |
|-------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|------------|
| 02 00 00 00 00 66 30 00 | 00 C0            | 01              | 01             | 04 00         | 02            | C9 02       | FC         |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> | <b>LQI</b> |

Now we can count up the number of bytes for our initial **Length** field which would be 1 byte for LQI, 2 bytes for **Data**, 1 byte for **Length**, 2 bytes for **Offset**, 1 byte for **Cluster**, 1 byte for **Endpoint**, 2 bytes for **ProfileID**, 8 bytes for the **MAC/NWK Address**, 1 byte for the **MSG Type**, and 1 byte for the **TransID** for a total of 20 bytes or 0x13 which agrees with the value received.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | 14            | 00             | 85              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

|                         |                  |                 |                |               |               |             |            |
|-------------------------|------------------|-----------------|----------------|---------------|---------------|-------------|------------|
| 02 00 00 00 00 66 30 00 | 00 C0            | 01              | 01             | 04 00         | 02            | C9 02       | FC         |
| <b>MAC/NWK Address</b>  | <b>ProfileID</b> | <b>Endpoint</b> | <b>Cluster</b> | <b>Offset</b> | <b>Length</b> | <b>Data</b> | <b>LQI</b> |

### 8.10.7 Discovery Request

We'll start with our packet model as shown below.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | 6 bytes          |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD and as in our previous example, **Length** will always be 6 bytes because of the fixed length of the **Arguments**. We will assume this is our first packet which will make our **TransID** 0x00. We are sending a Discover Request which is listed in the Serial Protocol table. Reading from the table, we see that our **MSG Type** is 0x64 and our packet will look like the one below.

|            |               |                |                 |                  |
|------------|---------------|----------------|-----------------|------------------|
| FD         | 06            | 00             | 64              | Varies           |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

The Discovery Request message **Argument** field is made up of the items below.

|                  |                 |                |
|------------------|-----------------|----------------|
| 2 bytes          | 1 byte          | 1 byte         |
| <b>ProfileID</b> | <b>Endpoint</b> | <b>Timeout</b> |

Cirronet's 2 byte CSM **ProfileID** is 0xc000 (multi-byte fields must be entered LSB first) and the **Endpoint** for Cirronet's CSM Profile is 0x01. Only devices with matching Profile ID and Endpoint will respond. The **Timeout** period is the amount of time, in seconds (converted to hexadecimal) to wait for discovery replies and can be any number from 0 – 60 secs. (0x00 – 0x3c) Let's choose a **Timeout** of 3 seconds or 0x03. Now our Argument field looks like this.

|                  |                 |                |
|------------------|-----------------|----------------|
| 00 C0            | 01              | 03             |
| <b>ProfileID</b> | <b>Endpoint</b> | <b>Timeout</b> |

Putting the entire packet together looks like this.

|            |               |                |                 |                  |                 |                |
|------------|---------------|----------------|-----------------|------------------|-----------------|----------------|
| FD         | 06            | 00             | 64              | 00 C0            | 01              | 03             |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> | <b>ProfileID</b> | <b>Endpoint</b> | <b>Timeout</b> |

When a device receives a Discovery Request, it responds with its assigned Zigbee network address.

### 8.10.8 Discovery Reply

Once a device that matches the profile of the device transmitting the Discovery Request, it will respond with a Discovery Reply. Let's see what that will look like. We'll start with our packet model as shown below.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | 4 bytes          |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD, **Length** will be 4 bytes. Since this is a response to the Discovery Request packet, the **TransID** will match the request which in this example is 0x00. Since this is a Discovery Reply, from the Serial Protocol table, the **MSG Type** is going to be 0xE4 and our packet will look like this.

|            |               |                |                 |
|------------|---------------|----------------|-----------------|
| FD         | 04            | 00             | E4              |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

The Discovery Reply message **Argument** field is a 2-byte network address as shown in the example below. (Don't forget the reply is LSB first in multi-byte fields!) Thus the entire packet looks like this.

|            |               |                |                 |                        |
|------------|---------------|----------------|-----------------|------------------------|
| FD         | 04            | 00             | E4              | 01 00                  |
| <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> | <b>Network Address</b> |

**Note: Additional Zigbee commands cannot be issued until a Discovery End packet has been received.**

### 8.10.9 Discovery End

The device sending out the Discovery Request will get responses like the one above but each response will have a unique Network Address. Once the **Timeout** value has been reached, and a list of network addresses has been built from the replying devices, a Discovery End packet with a **MSG Type** of  $\times 065$ , is sent down the Serial port of the device transmitting the Discovery Request. It will contain the **Number of Replies** it received and because it is a 2 byte field, LSB will be first. For our example, let's say it only found one device, the packet will look like the example below.

|     |        |         |          |                   |
|-----|--------|---------|----------|-------------------|
| FD  | 04     | 00      | 65       | 01 00             |
| SOP | Length | TransID | MSG Type | Number of Replies |

### 8.10.10 Get IEEE Address

The Get IEEE Address command can be issued for each device that responded to the Discovery Command to obtain the IEEE MAC Addresses. Let's look at what that packet will look like.

|     |        |         |          |
|-----|--------|---------|----------|
| FD  | TBD    | 01      | 10       |
| SOP | Length | TransID | MSG Type |

As you can see, **Trans ID** is 0x01 and we've entered the **MSG Type** for a Get IEEE Address request which is 0x10. We still need to determine the **Length** value so let's take a look at the Arguments portion of the packet.

|                        |                     |                    |
|------------------------|---------------------|--------------------|
| 2 bytes                | 1 byte              | 1 byte             |
| <b>Network Address</b> | <b>Request Type</b> | <b>Start Index</b> |

The 2 byte **Network Address** lets a device know when it is being communicated with and must be entered LSB first. **Request Type** can have two values, 0x00 requests only the MAC address of the device specified by the network address, whereas **Request Type** 0x01, requests the MAC address of the device specified in the **Network Address** field and the network addresses of devices that may be associated with the device specified by the **Network Address**, i.e. those directly connected to it. This mode is used to request the **Network Address** of devices that have joined the network through a Router. To obtain the MAC Addresses of the associated devices, a Get IEEE Address command must be issued for each associated device individually.

The **Start Index** field is used only if associated devices are requested. Then this is the index to the list to start returning. This is used on multiple requests to retrieve the list if it is larger than a ZigBee packet allows. Set this value to 0x00 if the list is not requested.

Using our **Network Address** from the previous example, 0x0001, a **Request Type** of 0x00, and a **Start Index** field of 0x00, here is what the Arguments part of the packet would be for a request only of the MAC address of the device specified in the Network Address field.

|                        |                     |                    |
|------------------------|---------------------|--------------------|
| 00 01                  | 00                  | 00                 |
| <b>Network Address</b> | <b>Request Type</b> | <b>Start Index</b> |

Adding it all up, our **Length** value becomes 6 bytes or 0x06 so here is what our Get IEEE Address packet will look like.

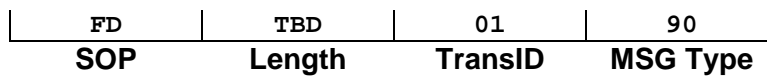
|     |        |         |          |                 |              |             |
|-----|--------|---------|----------|-----------------|--------------|-------------|
| FD  | 06     | 01      | 10       | 01 00           | 00           | 00          |
| SOP | Length | TransID | MSG Type | Network Address | Request Type | Start Index |

### 8.10.11 Get IEEE Address Reply

The Get IEEE Address is followed by a Get IEEE Address Reply and follows the same model as our other packets.

|                      |                             |                |                 |                  |
|----------------------|-----------------------------|----------------|-----------------|------------------|
| 1 byte               | 1 byte                      | 1 byte         | 1 byte          | Varies           |
| <b>SOP</b><br>(0xFD) | <b>Length</b><br>(in bytes) | <b>TransID</b> | <b>MSG Type</b> | <b>Arguments</b> |

**SOP** is 0xFD and the **Length** will be calculated by the module and will be included in the reply. The **TransID** will match the Get IEEE Address packet **TransID** which was 0x01 and Get IEEE Address Reply has a the **MSG Type** of 0x90 so the first part of our packet will look like this.



The Arguments portion of the packet will contain the **Network Address**, the **IEEE MAC Address**, the **Start Index**, the **Number of Associated Devices** and the **List of Network Addresses**. Here is the format showing the number of bytes in each field.

|                        |                         |                    |                                     |                                |
|------------------------|-------------------------|--------------------|-------------------------------------|--------------------------------|
| 2 bytes                | 8 bytes                 | 1 byte             | 1 byte                              | Varies                         |
| <b>Network Address</b> | <b>IEEE MAC Address</b> | <b>Start Index</b> | <b>Number of Associated Devices</b> | <b>List of Network Address</b> |

Since it is not known how many devices are associated with the device specified, the Start Index for the initial command should be zero. Once the number of associated devices becomes known, additional Get IEEE commands can be issued with incrementing Start Indexes until all associated devices have been identified.

The **Network Address** is of the device for which the IEEE Address has been requested. The **IEEE MAC Address** the full, 8-byte address being returned from the device associated with the network address. If associated devices were requested, then the **Start Index** is the index to the list which was specified in the Get IEEE command. This is used on multiple requests to retrieve the list if it is larger than a ZigBee packet allows. The **Number of Associated Devices** represents the number of devices that have associated with the device that has the accompanying MAC address. It is also the length of the list that follows. 0x00 if not requested or no associated devices.

The **List of Network Addresses** is a list of 16-bit network addresses that is the **Number of Associated Devices** in length. This value is 0x00 if not requested or if the **Number of Associated Devices** is 0.

Using our **Network Address** from the previous example, 0x0001 and making up an **IEEE MAC Address** to be returned by the device, 00:30:66:00:00:00:00:02, then using our **Start Index** from our previous example along with the fact that we were only looking for the device itself and not all the associated devices, our Arguments portion looks like this.

|                        |                         |                    |                                     |                                |
|------------------------|-------------------------|--------------------|-------------------------------------|--------------------------------|
| 01 00                  | 02 00 00 00 00 66 30 00 | 00                 | 00                                  | 00                             |
| <b>Network Address</b> | <b>IEEE MAC Address</b> | <b>Start Index</b> | <b>Number of Associated Devices</b> | <b>List of Network Address</b> |

Now let's add up our Length data, 1 byte for List of **Network Addresses**, 1 byte for **Number of Associated Devices**, 1 byte for **Start Index**, 8 bytes for the **IEEE MAC Address**, 2 bytes for the **Network Address**, 1 byte for **MSG Type** and 1 byte for **TransID** for a total of 15 bytes which agrees with the Length value returned of `0x0F`. The entire packet will look like this.

|  |            |               |                |                 |
|--|------------|---------------|----------------|-----------------|
|  | FD         | 0F            | 01             | 90              |
|  | <b>SOP</b> | <b>Length</b> | <b>TransID</b> | <b>MSG Type</b> |

|                        |                         |                    |                                     |                                |
|------------------------|-------------------------|--------------------|-------------------------------------|--------------------------------|
| 01 00                  | 02 00 00 00 00 66 30 00 | 00                 | 00                                  | 00                             |
| <b>Network Address</b> | <b>IEEE MAC Address</b> | <b>Start Index</b> | <b>Number of Associated Devices</b> | <b>List of Network Address</b> |



## 9. Custom Profiles

Custom profiles may be developed and downloaded to the ZMN2405HP module, either through the serial port or through the JTAG port. To develop a custom profile, development tools must be obtained and licensed from TI/Chipcon/Figure 8 Wireless. Before embarking on this effort, it is strongly recommended that familiarity is obtained with the development tools. Also, when loading custom code, care must be taken to avoid overwriting the bootloader area if you want to continue to be able to use the Cirronet bootloader.

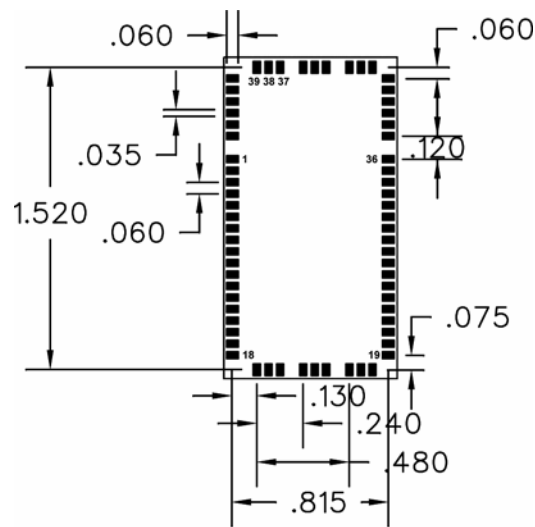
The table below shows the connection between the ZMN2405 pins and the CC2430.

| Pin No.                               | Signal   | CC2430 Pin                        |
|---------------------------------------|----------|-----------------------------------|
| 1                                     | Vcc      |                                   |
| 2,11,17-20, 28, 33,<br>34, 36, 37, 39 | GND      |                                   |
| 3                                     | PWMA     | 5 thru 10kohm<br>w/0.1uF to gnd   |
| 4                                     | PWMB     | 6 thru 10kohm<br>w/0.1uF to gnd   |
| 5                                     | GPIO0    | 18                                |
| 6                                     | GPIO1    | 8                                 |
| 7                                     | GPIO2    | 6                                 |
| 8                                     | GPIO3    | 9                                 |
| 9                                     | GPIO4    | 3                                 |
| 10                                    | GPIO5    | 4                                 |
| 11                                    | GND      |                                   |
| 12                                    | LINK/DD  | 46                                |
| 13                                    | /RESET   | 10                                |
| 14                                    | ACT/DC   | 45                                |
| 15                                    | NC       |                                   |
| 16                                    | ADC REF  |                                   |
| 21                                    | UART_RX  | 1                                 |
| 22                                    | UART_TX  | 2                                 |
| 23                                    | NC       |                                   |
| 24                                    | /RESET   | 10                                |
| 25                                    | ADCX     | 11 thru 100kohm<br>w/0.1uF to gnd |
| 26                                    | ADCY     | 12 thru 100kohm<br>w/0.1uF to gnd |
| 27                                    | ADCZ     | 17 thru 100kohm<br>w/0.1uF to gnd |
| 29                                    | SPI_EN   | 15                                |
| 30                                    | SPI_CLK  | 16                                |
| 31                                    | SPI_MOSI | 14                                |
| 32                                    | SPI_MISO | 13                                |
| 35                                    | NC       |                                   |
| 38                                    | RF       |                                   |

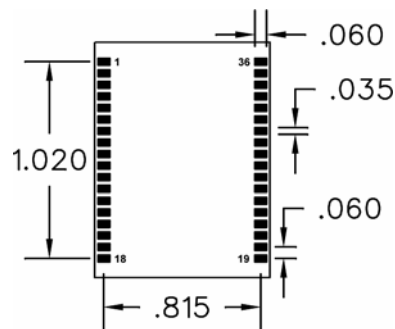
## 10. Layout Guidelines for ZMN2405/ZMN2405HP Module

- Place either bare PWB material or copper ground plane underneath the module. No traces should be run underneath the module.
- The connection between the RF output from the module (pin 38) and the customer's antenna/connector should be made with 50 ohm microstrip. A 10pF 0402 capacitor must be placed in series with this microstrip line for DC blocking purposes.
- Good grounding of the module is paramount. Place ground as close as possible to the support board ground pads.
- Follow the temperature profile shown on the next page for proper placement of the module onto your support board.
- Keep all interconnects between the support board circuitry and the module pins as short as possible.
- Place at least 1 bypass capacitor (0.1uF minimum) as close as possible to the power supply pin (pin 1) off the module.
- Pin 35 is the RF output on the ZMN2405 but is a no-connect on the ZMN2405HP.

*Note: All unlabeled pads (pins) are not connected electrically and are for mechanical support.*

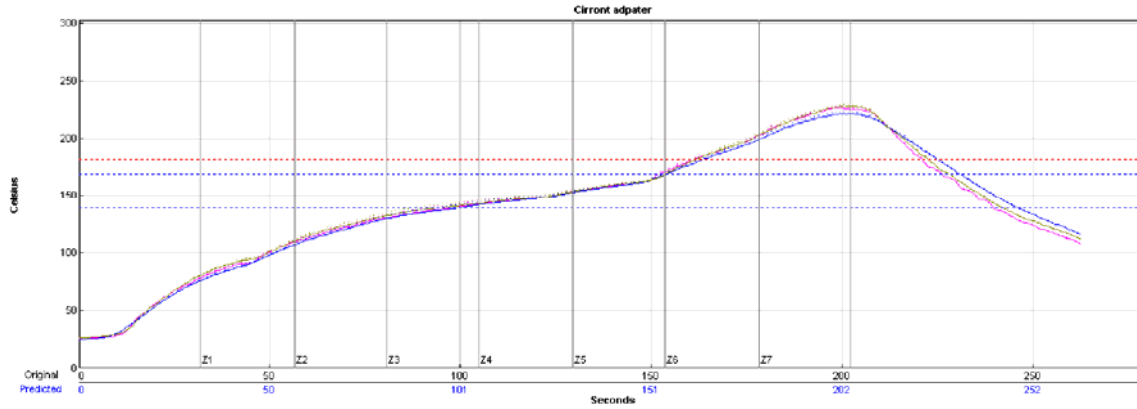


ZMN2405HP



ZMN2405

### 10.1 Reflow Profile for ZigBee Adapter Panel



| TCs   | Max Rising Slope | Max Falling Slope | Soak Time 140-170C | Reflow Time/183C | Peak Temp |      |      |     |       |      |
|-------|------------------|-------------------|--------------------|------------------|-----------|------|------|-----|-------|------|
| 2     | 1.6              | 12%               | -2.0               | 48%              | 52.8      | -86% | 59.5 | 12% | 227.1 | 21%  |
| 3     | 1.5              | 4%                | -1.8               | 60%              | 53.6      | -82% | 60.9 | 20% | 222.2 | -28% |
| 4     | 1.6              | 15%               | -2.0               | 51%              | 59.2      | -54% | 60.4 | 16% | 229.1 | 41%  |
| Delta | 0.05             |                   | 0.24               |                  | 6.45      |      | 1.39 |     | 6.89  |      |
| P.2   | 1.6              | 18%               | -2.1               | 47%              | 60.6      | -47% | 61.5 | 23% | 228.9 | 39%  |
| P.3   | 1.5              | 9%                | -1.8               | 59%              | 60.5      | -47% | 63.4 | 34% | 224.0 | -10% |
| P.4   | 1.6              | 22%               | -2.0               | 50%              | 66.3      | -18% | 62.2 | 27% | 230.8 | 58%  |
| Delta | 0.06             |                   | 0.24               |                  | 5.79      |      | 1.95 |     | 6.78  |      |

## 11. WARRANTY

Seller warrants solely to Buyer that the goods delivered hereunder shall be free from defects in materials and workmanship, when given normal, proper and intended usage, for twelve (12) months from the date of delivery to Buyer. Seller agrees to repair or replace at its option and without cost to Buyer all defective goods sold hereunder, provided that Buyer has given Seller written notice of such warranty claim within such warranty period. All goods returned to Seller for repair or replacement must be sent freight prepaid to Seller's plant, provided that Buyer first obtain from Seller a Return Goods Authorization before any such return. Seller shall have no obligation to make repairs or replacements which are required by normal wear and tear, or which result, in whole or in part, from catastrophe, fault or negligence of Buyer, or from improper or unauthorized use of the goods, or use of the goods in a manner for which they are not designed, or by causes external to the goods such as, but not limited to, power failure. No suit or action shall be brought against Seller more than twelve (12) months after the related cause of action has occurred. Buyer has not relied and shall not rely on any oral representation regarding the goods sold hereunder, and any oral representation shall not bind Seller and shall not be a part of any warranty.

**THE PROVISIONS OF THE FOREGOING WARRANTY ARE IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL (INCLUDING ANY WARRANTY OR MERCHANT ABILITY OR FITNESS FOR A PARTICULAR PURPOSE). SELLER'S LIABILITY ARISING OUT OF THE MANUFACTURE, SALE OR SUPPLYING OF THE GOODS OR THEIR USE OR DISPOSITION, WHETHER BASED UPON WARRANTY, CONTRACT, TORT OR OTHERWISE, SHALL NOT EXCEED THE ACTUAL PURCHASE PRICE PAID BY BUYER FOR THE GOODS. IN NO EVENT SHALL SELLER BE LIABLE TO BUYER OR ANY OTHER PERSON OR ENTITY FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS, LOSS OF DATA OR LOSS OF USE DAMAGES ARISING OUT OF THE MANUFACTURE, SALE OR SUPPLYING OF THE GOODS. THE FOREGOING WARRANTY EXTENDS TO BUYER ONLY AND SHALL NOT BE APPLICABLE TO ANY OTHER PERSON OR ENTITY INCLUDING, WITHOUT LIMITATION, CUSTOMERS OF BUYERS.**