WirelessUSB™ N:1 Development Kit

Technical Reference Manual

Cypress Semiconductor Corporation • 3901 North First Street • San Jose • CA 95134 • 408-943-2600

Table of Contents

1. INTRODUCTION

1.1 Audience

This document is intended to be read by a software, firmware, or hardware engineer responsible for an N:1 implementation or those interested in the detailed system architecture.

1.2 Overview

Cypress has created software and firmware to assist in the development of N:1 projects. While it may not be the exact implementation or platform for your application, you are encouraged to read this document (or use it as a reference) and look at the source files to rapidly accelerate your own development cycle.

1.3 Design Goals

The N:1 Firmware was written to demonstrate various N:1 applications using a Cypress PSoC MCU. Some of the applications supported by the N:1 DVK include:

- Wireless security sensor
- Wireless thermostat
- Wireless actuator

The N:1 hardware was design to allow easy prototyping for additional applications through the Node Board expansion header and the included Proto Board.

1.4 Definitions

Back Channel Data	User payload sent from the Hub back to the Sensor
cpb	Chips Per Bit. Each bit transmitted over the air is broken into either 64 or 32 1µs pulses over the air. These pulses are called chips.
COBS	Consistent Overhead Byte Stuffing – encoding scheme used for serial communication
DSSS	Direct Sequence Spread Spectrum – communication scheme that multiplies data bits by a pseudo-random bit pattern (PN sequence) that "spreads" the data into a large coded stream that takes the full bandwidth of the channel.
Host	An external system that communicates with the Hub via RS-232. In the N:1 Kit a PC acts as the Host.
Node	Refers to a generic board that can be programmed to be a Hub or Sensor

Hub	An N:1 Node Board acting as a Hub for all network traffic.
ISP (or ISSP)	In-circuit System Programmer or In-System Serial Programmer – Used to program PSoC Microcontrollers in-system.
ISR	Interrupt Service Routine – The MCU switches to this routine when an interrupt is pending.
LSB	Least Significant Byte
MCU	Micro Controller Unit – An integrated device that contains a CPU and some analog or digital peripheral interface functions.
MID	Manufacturing Identification – 4-byte value read from the radio that can be used as a unique identifier.
MSB	Most Significant Byte
N:1	Pronounced “N to one”. Refers to multiple devices communicating to a single device in a star configuration.
Packet	An array of bytes sent over the air between nodes. A Data Packet contains a user Payload.
Payload	The user application passes and receives a Payload to the Protocol, which in turn wraps the Payload in a Packet used for over the air transmission/reception.
RSSI	Receive Signal Strength Indicator – The radio receiver contains circuitry to measure and report the strength of incoming RF signals.
Sensor	An N:1 Node Board that collects and sends data to the Hub. The documentation uses the term Sensor to refer to Sensors or Actuators.

2. WIRELESSUSB™ N:1 PROTOCOL

2.1 Introduction

The primary purpose of WirelessUSB™ N:1 DVK is to add a multipoint-to–point (N:1) wireless protocol to the existing WirelessUSB portfolio. The N:1 DVK is capable of servicing low data rate higher density node applications beyond simple 1:1 or 2:1 wireless connectivity. The WirelessUSB N:1 Protocol is designed for reliable 2-way communication between a wireless Hub and target Sensor or Actuator applications in N:1 networks.

2.1.1 WirelessUSB N:1 Hub

The WirelessUSB N:1 Hub forms the center of a star network that can contain thousands of Sensors or Actuators. In order to accommodate extremely power-sensitive Sensors the Hub is typically assumed to be powered by an constant external power supply. The WirelessUSB N:1 Hub interfaces with a local host as shown in Figure 1: N:1 Star Network or to a remote host (via Wi-Fi, Ethernet, etc) to create a multi-star network. For simple applications, the Hub may not require connection to a host. The entire Hub/Host application logic may be self-contained in the Hub firmware.

2.1.2 WirelessUSB N:1 Sensor (Remote Device)

The WirelessUSB N:1 Protocol has been optimized for low-power Sensors that require battery life to be measured in years, not days or months. Examples of such devices are: environmental sensors, security alarms, and hotel door locks.

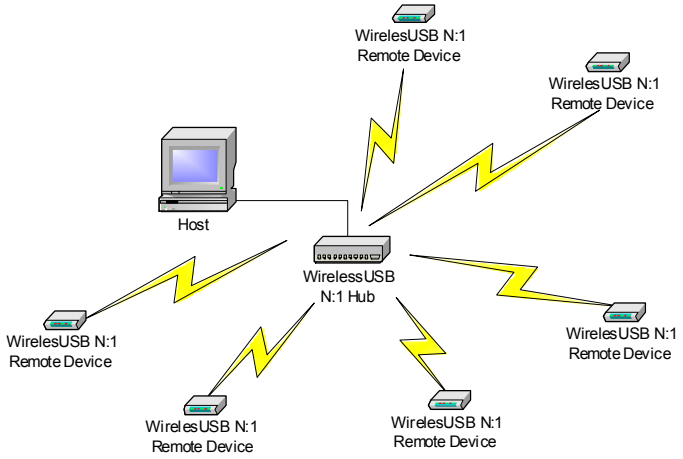


Figure 1: N:1 Star Network

2.2 WirelessUSB Radio

2.2.1 Channel Management

The WirelessUSB N:1 protocol utilizes the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) band for wireless connectivity. WirelessUSB N:1 splits the band into 79 distinct 1-Mhz channels with the first channel at 2.402 GHz. Each WirelessUSB N:1 network uses a subset* of channels spread across the 2.4 GHz frequency band in order to minimize the probability of interference from other WirelessUSB networks, while reducing the number of possible channels each Sensor must search in order to find the current channel being used by the Hub (See Section 2.7.4 for more details). Figure 2 shows an example of the spectrum divided into six channel subsets with each color representing a channel subset.

[* channel subset size is configurable from 6-13 channels per subset]

Figure 2: Channel Subsets

2.2.2 Pseudo-Noise Codes

Pseudo-Noise Codes (PN Codes) are the codes used to achieve the special matched filter characteristics of DSSS communication. The PN Codes used in WirelessUSB have minimal cross-correlation properties, meaning they are less susceptible to interference caused by overlapping transmissions on the same channel. The length of the PN Code results in different communication characteristics. Higher data rates are achieved with 32 chips per bit PN Codes, while 64chips per bit PN Codes allow longer range. The number of frequency/code pairs is large enough to comfortably accommodate hundreds of WirelessUSB devices in the same space. All devices in a network must use the same PN Code and channel in order to communicate. (See the *WirelessUSB LS Theory of Operation* document for more details.)

2.3 Error Correction

There are four methods of error correction in WirelessUSB N:1:

- Chip error correction
- Bit error correction
- Cyclic Redundancy Check (CRC)
- ACK/retransmission

2.3.1 Chip Error Correction

Chip error correction is used to ensure reliable reception of individual data bits. In the presence of interference (or near the limits of range), the transmitted PN Code will often be received with some PN Code chips corrupted. DSSS receivers use a data correlator to decode the incoming data stream. If the number of chip errors is less than the correlator error threshold, the data will be correctly received. It is therefore possible for WirelessUSB systems to successfully receive data without error on frequencies suffering from interference causing chip error rates in excess of 10 percent. Figure 3 shows a WirelessUSB 64chips/bit PN code example.

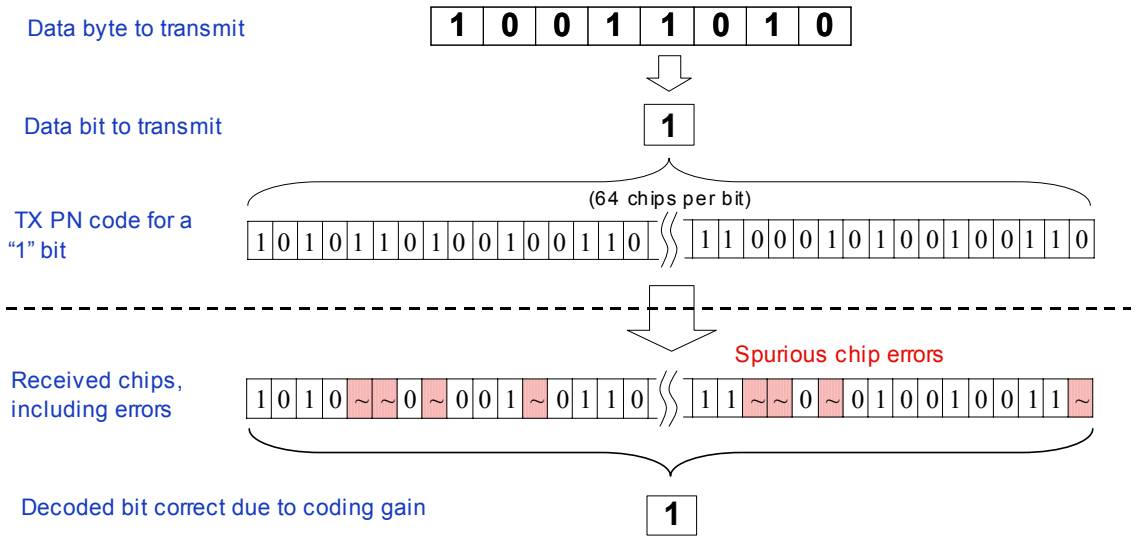


Figure 3: Chip Error Correction

2.3.2 Bit Error Correction

If the correlator threshold is exceeded, the received data bit is not corrupted; it is "erased", or in other words invalid. There is a negligible probability of data being corrupted rather than erased, because this would require interference to corrupt the majority of chips in such a way that the incoming data stream correlated with the PN Code corresponding to the opposite logic state.

Erasures are much easier to correct than errors. By XORing each byte (including the packet header and CRC) and the Network Checksum Seed (See *Section 2.4.5* for more details), and transmitting the resulting checksum, it is possible to use this checksum to correct one error in each bit position in a received packet. See below for more details.

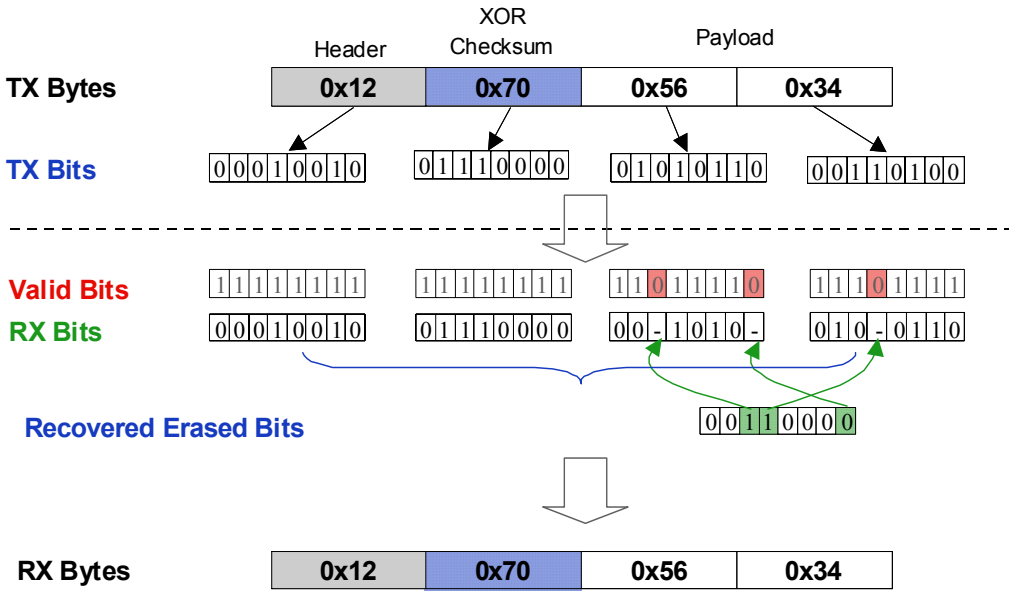


Figure 4: Bit Error Correction

2.3.3 Cyclic Redundancy Check (CRC)

By combining the bit error correction scheme above with a 16-bit CRC the N:1 protocol provides excellent error detection as well as excellent error correction. The CRC is capable of detecting errors caused by bit-shifts, dropped bits, and corrupt bits. If all bits in the packet are valid or are fixed by the bit error correction scheme, the CRC is then calculated and compared against the received CRC value. If the calculated CRC does not match the received CRC the packet is discarded. The CRC is calculated for the header and packet payload, but not on the CRC and XOR checksum fields of the packet. The CRC is optional and may be removed in order to reduce the packet length in quiet environments.

2.3.4 Packet Retransmission

The WirelessUSB N:1 Protocol employs an ACK/NAK scheme to guarantee transmission. Data Packets are retransmitted until an ACK packet is received. If a packet is corrupted during transmission and the receiver is unable to correct the errors using the error correction schemes described above, the receiver may send a NAK packet to the transmitter (or the receiver may simply not respond, which is an implied NAK). If the transmitter receives a NAK or no response from the intended receiver the transmitter will retransmit the original packet. If the Data Packet was received correctly by the receiver, but the ACK packet was lost or corrupted during transmission the transmitter will retransmit the original packet, which will be recognized as a duplicate packet by the receiver and discarded after transmitting an ACK. (See Section 2.7.5 for more details)

2.4 Network Parameters

2.4.1 Radio Manufacturing ID

Each WirelessUSB radio contains a 4-byte Manufacturing ID (MID), which has been laser fused into radio registers 0x3C - 0x3F during manufacturing. Sensors must transmit their MID to the Hub during the bind procedure as a means of device identification. The Hub also transmits its Manufacturing ID to each Sensor during the bind procedure. The Hub's Manufacturing ID is used to calculate the Network Checksum Seed and the Network CRC Seed. Applications may also use the MID as a pseudo-random seed.

2.4.2 Network Channel

During Bind Mode the Hub notifies the Sensor of the current channel being used for Data Mode; this channel is known as the current channel. Once the current channel is known by the Sensor it can calculate the other channels contained in the Network Channel sequence.

2.4.3 Network PN Code

All packets transmitted between bound devices use a single PN Code, which is determined by the Hub; this PN Code is known as the Network PN Code.

2.4.4 Device ID

During the bind procedure the Hub assigns each Sensor an 8-bit or 16-bit device ID (the length is determined by firmware implementation), which is used to uniquely identify each Sensor. All packets sent from Sensors (except for Bind Request packets) contain the Sensor's Device ID. All packets sent from the Hub contain the Device ID of the intended recipient.

2.4.5 Network Checksum Seed

The Network Checksum Seed is XORed with each byte in the header and payload (See *Section 2.3.2* for more details) when determining the checksum value for all packets sent between bound devices (Data and ACK packets) in order to reduce the possibility of packets from neighboring systems being accidentally received as valid packets. All packets sent between non-bound devices use the default checksum seed of 0x00 (Bind Request, Bind Response, and Broadcast Ping packets). The Network Checksum Seed is MID 4 of the Hub's Manufacturing ID.

2.4.6 Network CRC Seed

The Network CRC Seed is used to seed the CRC calculation for all packets sent between bound devices (Data, ACK, and Network Ping packets) in order to reduce the possibility of packets from neighboring systems being accidentally received as valid packets. All packets sent between non-bound devices use the default CRC seed of 0x00 (Bind Request, Bind Response, and Broadcast Ping packets). The Network CRC Seed is MID 3 of the Hub's Manufacturing ID.

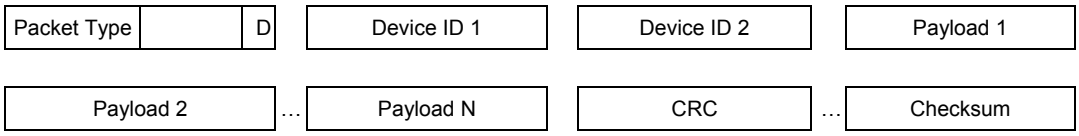
2.5 Bind Parameters

In order for all WirelessUSB N:1 devices to communicate during Bind Mode the following parameters are always used during Bind Mode:

- CRC Seed – 0x00
- Checksum Seed – 0x00
- PN Code ID – 0x00
- Base Channel – 0x00

2.6 Packet Structures

The most significant nibble of the first byte contains the packet type. Packet types 0x0 – 0x5 are defined below, packet types 0x6 – 0xF are reserved. All unused bit fields in the Packet Header are set to 0. The general WirelessUSB N:1 protocol packet format is described below:



Packet Header

> **Single Byte Device ID Bit (D**) – This is a 1-bit field specifying a 1-byte or 2-byte Device ID. If this bit is set a 1-byte Device ID will be used, otherwise the Device ID is 2 bytes. (1-byte Device ID support was not built into revision 1.0 of the N:1 Development Kit.)

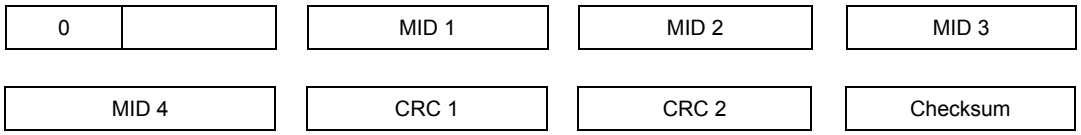
Device ID – This is a 1-2 byte field containing the Device ID assigned to this device. (See *Section 2.4.4* for more details)

Payload – This is packet type specific data as described for each packet type below.

CRC – This is an 16-bit field containing the CRC of all preceding bytes. (See *Section 2.3.3*)

Checksum – This is an 8-bit field containing the XOR checksum of all previous bytes in the packet. (See *Section 2.3.2* and *Section 2.4.5)*

2.6.1 Bind Request Packet (Sensor)

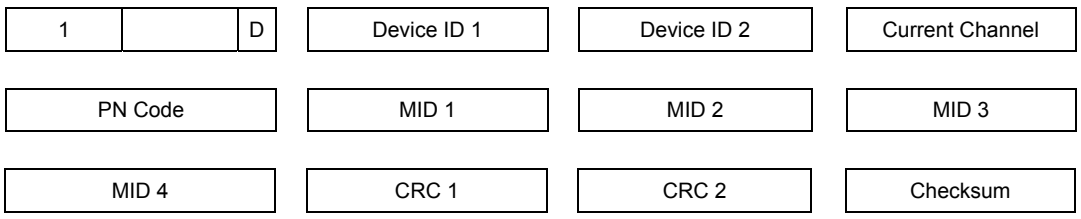


Manufacturing ID (MID 1 - MID 4) – This is the 4-byte Manufacturing ID retrieved from the WirelessUSB radio of the Sensor. MID 1 is register 0x3C, MID 2 is register 0x3D, MID 3 is register 0x3E and MID 4 is register 0x3F. (See *Section 2.4.1* for more details)

CRC – This is an 16-bit field containing the CRC of all preceding bytes. (See *Section 2.3.3*)

Checksum – This is an 8-bit field containing an XOR checksum of all previous bytes in the packet. The default checksum seed of 0x00 is used for Bind Request Packets. (See *Section 2.3.2*)

2.6.2 Bind Response Packet (Hub)



Packet Header

Single Byte Device ID Bit (D) – This is a 1-bit field specifying a 1-byte or 2-byte Device ID. If this bit is set a 1-byte Device ID will be used, otherwise the Device ID is 2 bytes.

Device ID – This is a 1-2 byte field containing the Device ID being assigned to this device. This Device ID is used in all future communications with the Hub. 0xFFFF is returned if no Device IDs are available (See *Section 2.4.4* for more details)

Current Channel –This is an 8-bit field specifying the current channel used by the Hub for Data Mode. (See *Section 2.4.2* for more details)

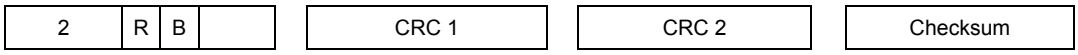
PN Code – This is an 8-bit field specifying the PN code index to be used. (See Section 2.4.3 for more details)

Manufacturing ID (MID 1 - MID 4) – This is the 4-byte Manufacturing ID retrieved from the WirelessUSB radio of the Hub. MID 1 is register 0x3C, MID 2 is register 0x3D, MID 3 is register 0x3E and MID 4 is register 0x3F.

CRC – This is an 16-bit field containing the CRC of all preceding bytes. (See *Section 2.3.3*)

Checksum – This is an 8-bit field containing the XOR checksum of all previous bytes in the packet. The default checksum seed of 0x00 is used for Bind Response Packets.

2.6.3 Ping Packet (Hub)



Packet Header

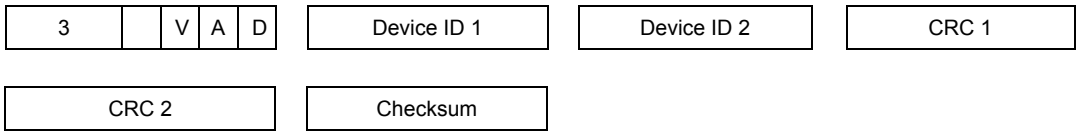
Response (R) – This is a 1-bit field specifying a Ping or Ping Response Packet (0 = Ping, 1 = Ping Response).

Broadcast (B) – This is a 1-bit field specifying a Broadcast Ping packet or a Network Ping packet. (See *Section 2.7.3* and *Section 2.7.4* for more details)

CRC – This is an 16-bit field containing the CRC of all preceding bytes.

Checksum – This is an 8-bit field containing an XOR checksum of all previous bytes in the packet. The default 0x00 checksum seed is used for Broadcast Ping Packets; the Network Checksum Seed is used for Network Ping Packets.

2.6.4 ACK Packet (Hub and Sensor)



Packet Header

Valid Device ID (V) – This is a 1-bit field specifying that the Device ID is a valid Device ID for this network. (See Section 2.7.8 for more details)

ACK Toggle (A) – This is a 1-bit field containing the value of the last received Sequence Bit from a Data packet.

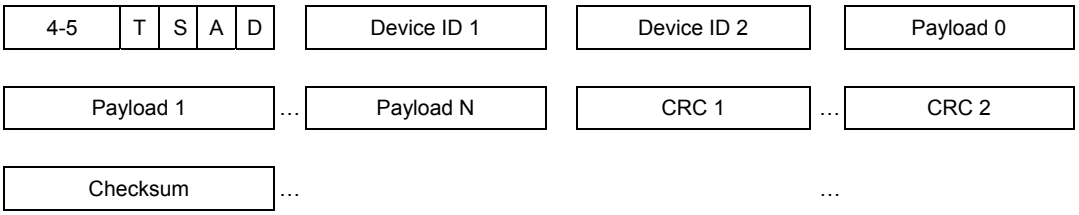
Single Byte Device ID Bit (D) – This is a 1-bit field specifying a 1-byte or 2-byte Device ID. If this bit is set a 1-byte Device ID will be used, otherwise the Device ID is 2 bytes.

Device ID – This is a 1-2 byte field containing the Device ID assigned to this device.

CRC – This is an 16-bit field containing the CRC of all preceding bytes.

Checksum – This is an 8-bit field containing an XOR checksum of all previous bytes in the packet. The Network Checksum Seed is used for ACK Packets.

2.6.5 Data Packet (Hub and Sensor)



Packet Header

Packet Type – The packet type used in Data Packets can be any type in the 4-7 range. Different Data Packet Types may be used to identify packet length, packet payload type, etc. The exact meaning of each Data Packet Type is implementation specific.

Sequence Bit (T) – This is a 1-bit field that is toggled for each new Data Packet. It is used to distinguish between new and retransmitted packets. The Sequence Bit is set to 0 after binding and after a device reset.

Synchronous Data (S) – This is a 1-bit field that is used when the packet contains synchronous data. If this bit is set the Sequence Bit is ignored; if the packet is valid the payload will be sent to the application layer regardless of the Seqeunce Bit. If this bit is not set, valid packets that contain the same Sequence Bit value as the last correctly received packet are assumed to be retransmitted packets and are discarded. Data Packets containing synchronous data are still ACKed by the Hub, but if an ACK is not received by the Sensor the Sensor is not required to retransmit the original packet. (See Section 2.7.7 for more details)

ACK Bit (A) – This is a 1-bit field containing the value of the last received Sequence Bit from a Data packet. (The ACK bit should be initialized to 1 during the bind procedure.)

Single Byte Device ID Bit (D) – This is a 1-bit field specifying a 1-byte or 2-byte Device ID. If this bit is set a 1-byte Device ID will be used, otherwise the Device ID is 2 bytes.

Device ID – This is a 1-2 byte field containing the Device ID assigned to this device.

Payload 0-N – This is byte-aligned application data.

CRC – This is an 16-bit field containing the CRC of all preceding bytes.

Checksum – This is an 8-bit field containing an XOR checksum of all previous bytes in the packet. The Network Checksum Seed is used for Data Packets.

2.7 Protocol Modes

The WirelessUSB N:1 Protocol operates in the following modes:

- Automatic Bind Mode
- Seeded Bind Mode
- Channel Selection Mode (Hubs only)
- Channel Search Mode (Sensors only)
- Data Mode

This section provides detailed descriptions of each mode, including sequence diagrams.

2.7.1 Automatic Bind Mode

2.7.1.1 Sensor (Remote Device)

Automatic Bind Mode allows the Sensor to retrieve the network parameters from the Hub (See Section 2.4 for more details). All Sensors must be bound before they can exchange data with the Hub. Upon entering Bind Mode the Sensor sets the channel and PN code to the Base Channel and PN Code reserved for Bind Mode (Base Channel 0, PN Code 0) enabling all WirelessUSB N:1 devices to effectively communicate during this procedure. The Sensor then alternately transmits Bind Requests (containing its device type and Manufacturing ID) and listens for Bind Responses (containing the network parameters) from the Hub. The Sensor transmits a defined* number of Bind Request messages on each channel. If a Bind Response is not received the Sensor moves to the next channel. If a Bind Response is received the Sensor stores the network parameters for later use and moves to Channel Search Mode. If a defined* period of time has elapsed while in Bind Mode without receiving a Bind Response, the Sensor assumes the Hub is not available and goes to

sleep. A user-initiated event may cause the Sensor to enter Bind Mode from any other mode.

[*The timeout values and number of packet transmissions are configurable.]

2.7.1.2 Hub

Upon entering Automatic Bind Mode the Hub sets the channel and PN Code to the Base Channel and PN Code reserved for Bind Mode (Base Channel 0, PN Code 0) enabling all WirelessUSB N:1 devices to effectively communicate during this procedure. The Hub listens for a defined* period of time for Bind Request on each channel before moving to the next channel in the subset. This reduces the possibility of the Hub not receiving the Bind Request from the Sensor in the event of channel interference. If the Hub receives a Bind Request from a Sensor it stores the Sensor's Manufacturing ID and sends a Bind Response packet, which contains the network parameters (See Section 2.4 for more details).

[*The timeout values and number of packet transmissions are configurable.]

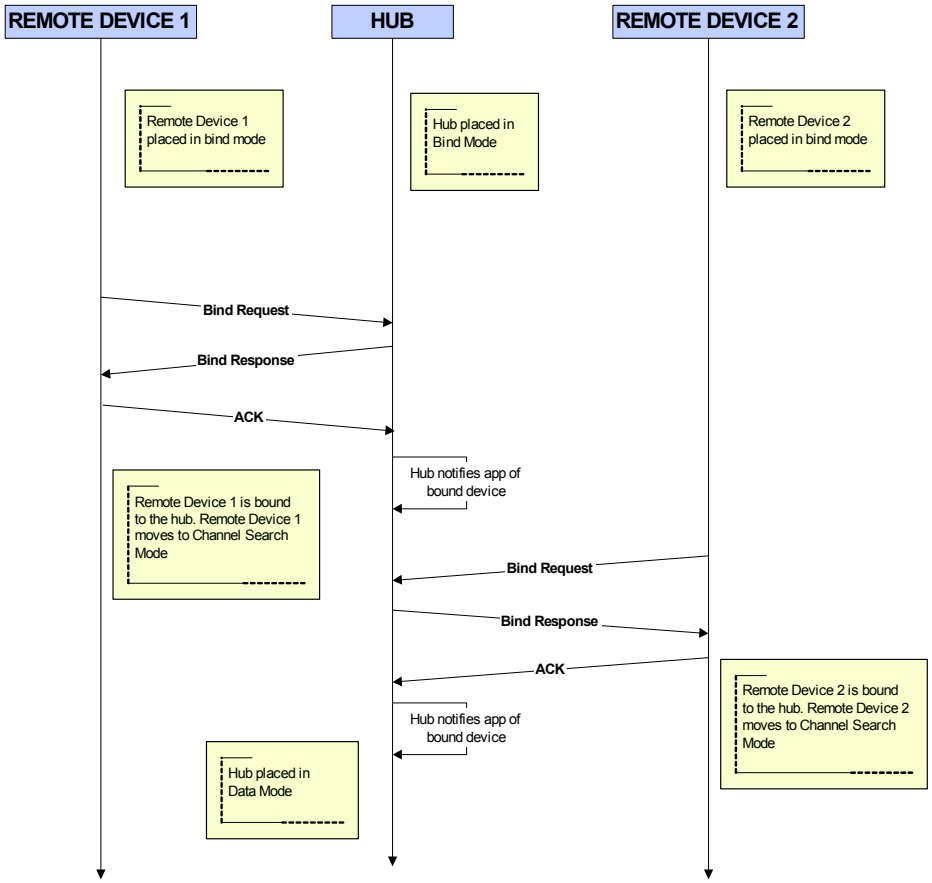


Figure 5: Bind Mode Sequence Diagram

2.7.2 Seeded Bind Mode

2.7.2.1 Sensor (Remote Device)

Sensors may be more easily bound by pre-programming them with the Network Channel and Network PN Code Index of the target Hub. Devices are able to communicate with the Hub using the Network Channel and the Network PN Code, but they have not received a Device ID or the remaining network parameters from the Hub yet. Therefore, upon initialization the Sensor sets the channel to the first channel in the Network Channel Subset and the PN Code to the Network PN Code. The Sensor then alternately transmits Bind Requests (containing its Manufacturing ID) and listens for Bind Responses (containing the network parameters) from the Hub. The Sensor transmits a defined number of Bind Request messages on each channel. If a Bind Response is not received the Sensor moves to the next channel in the Network Channel Subset. If a Bind Response is received the Sensor stores the network parameters for later use and moves to Channel Search Mode. If a defined period of time has elapsed while in Seeded Bind Mode without receiving a Bind

Response, the Sensor assumes the Hub is not available and goes to sleep.

2.7.2.2 Hub

If the Hub is configured for Seeded Binding the Hub will respond to Bind Request packets received during Data Mode as if the Bind Request packet had been received on the Bind Channel Subset and Bind PN Code during Bind Mode. If the Hub is not configured for Seeded Binding all Bind Request packets received during Data Mode will be ignored. One significant advantage of Seeded Bind mode for the Hub, is that it does not need to exit Data Mode (and therefore temporarily ignore current connected network devices) in order to bind new devices.

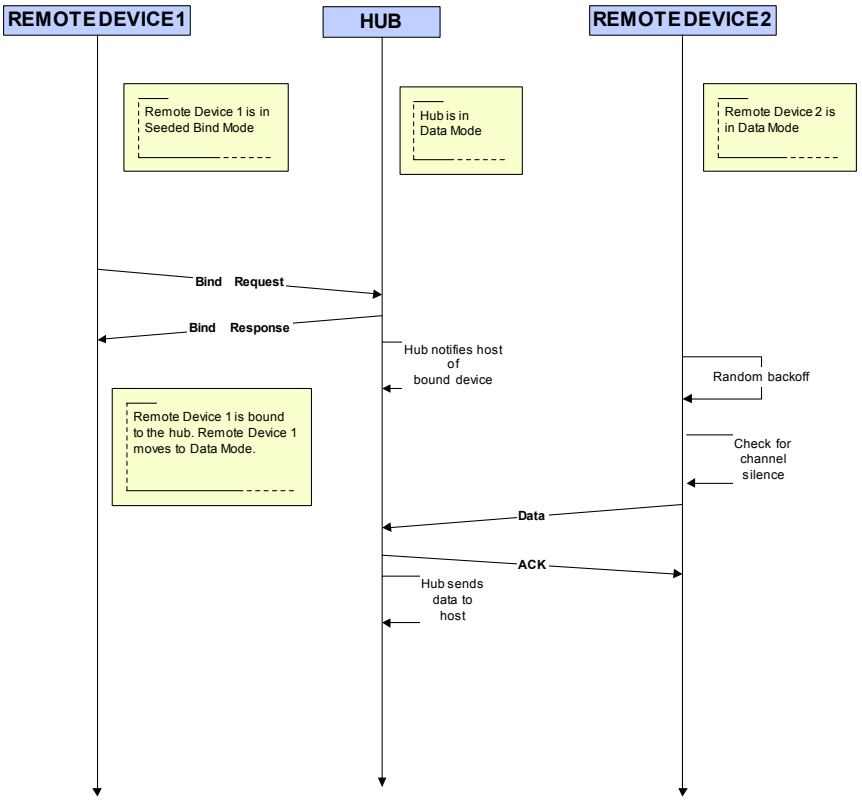


Figure 6: Seeded Bind Mode

2.7.3 Channel Selection Mode (Hub Only)

Channel Selection Mode is used by the Hub to find an available channel; channels are unavailable if they are being used by another network with the same PN code, or if there is excessive noise on the channel. The Hub first listens for activity on the selected channel by checking the Receive Signal Strength Indicator (RSSI) of the radio in order to determine if channel is being used by another wireless system. If the channel is inactive the Hub alternately transmits Broadcast Ping packets and listens

for Broadcast Ping Response packets for a defined period of time. If a Broadcast Ping Response is received, indicating that another Hub is using this channel, the Hub will select the next channel in the Network Channel Subset and repeat this procedure. The Hub will also select another channel if the RSSI is high, which indicates that the channel is being used by another wireless system. Hubs send Broadcast Ping Response packets in response to all received Broadcast Ping packets; Sensors never respond to Broadcast Ping packets. If a Ping Response is not received and the RSSI is low, the Hub assumes the channel is available, exits Channel Selection Mode, and enters Data Mode on the selected channel.

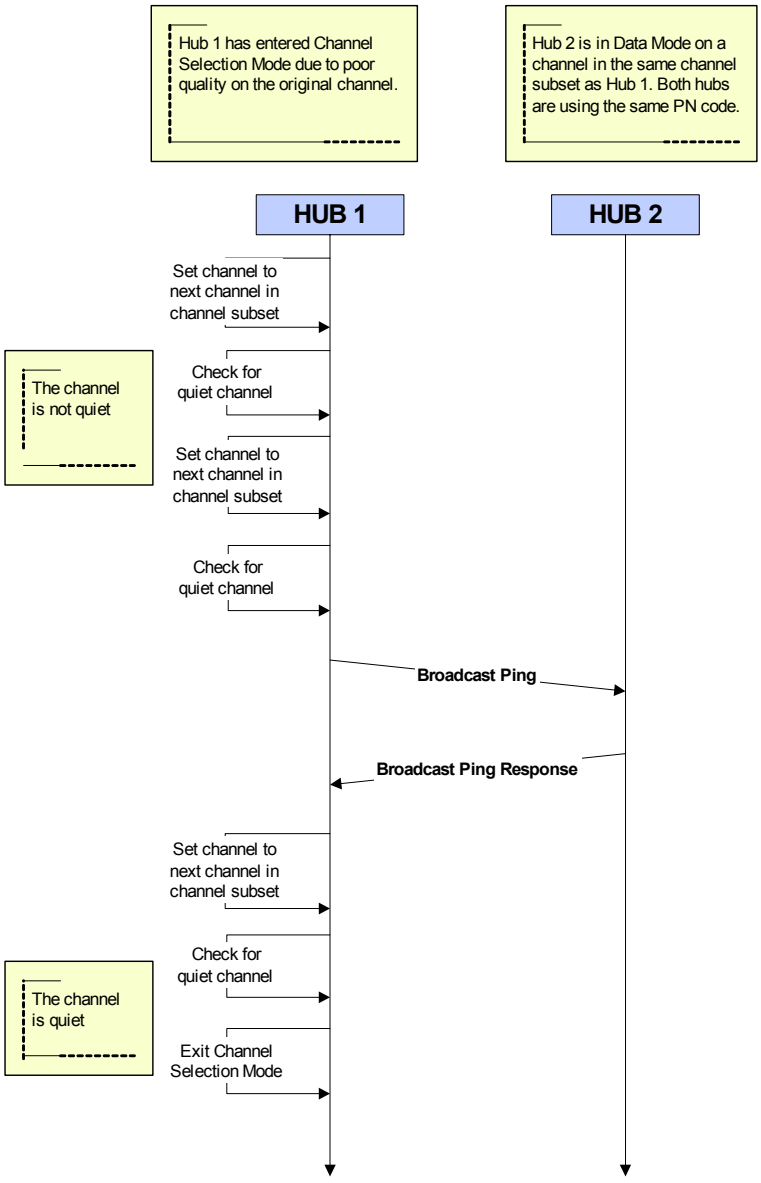


Figure 7: Channel Selection Mode

2.7.4 Channel Search Mode (Sensor Only)

Channel Search Mode is used by the Sensor to discover the current channel used by the Hub. Upon entering Channel Search Mode the Sensor selects a channel from the Network Channel Subset. The Sensor alternately transmits Data Packets containing its Device ID and listens for an ACK Packet from the Hub. If the Sensor does not receive an ACK, it selects the next channel in the Network Channel Subset and repeats the procedure. If an ACK is received the Sensor exits Channel Search Mode and enters Data Mode. If an ACK is not received on any channel in the Network Channel Subset the Sensor may choose to immediately reenter Channel Search Mode or wait (possibly sleeping) and enter Channel Search Mode later.

If the Sensor does not have data to send or the data is large, the Sensor may send Network Ping packets instead of Data Packets in Channel Search Mode. If Network Ping packets are used, the Hub will respond with Network Ping Response packets, not ACK packets. (This feature is not implemented in version 1.0 of the N:1 Development Kit.)

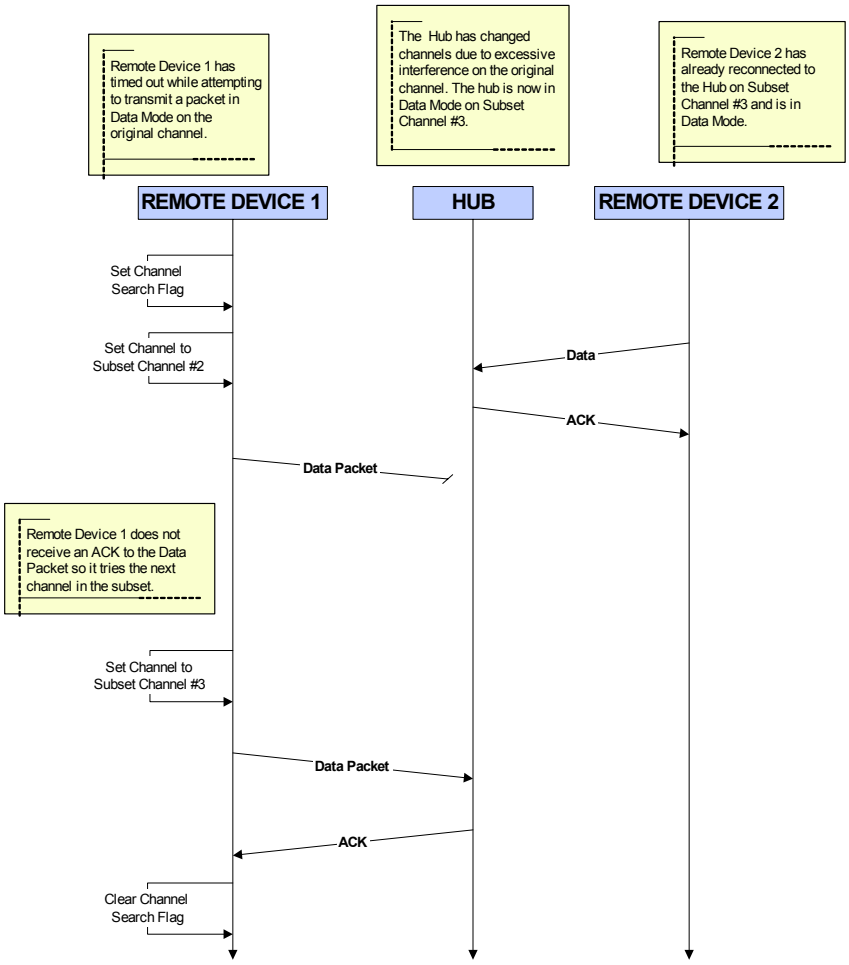


Figure 8: Channel Search Mode

2.7.5 Data Mode

2.7.5.1 Sensor

Data Mode allows application data to be transmitted between the Sensor and the Hub. When the Sensor application has data to send to the Hub the Sensor creates a Data packet and listens for a response (either an ACK or Data packet). The Sensor may also send an empty Data packet to the Hub in order to poll the Hub for data. If no response is received, the Sensor retransmits the packet. If the Sensor does not receive a response after a defined number of transmissions of the Data packet, then it assumes the channel has become unavailable due to excessive interference and moves to Channel Search Mode. If the received response is a Data packet, then the Sensor may respond immediately with an ACK or wait until the next transmitted Data packet to acknowledge the received Data packet from the Hub.

2.7.5.2 Hub

In Data Mode the Hub listens for Data packets from the Sensors. When a valid Data packet is received the Hub responds with an ACK packet or a Data packet, if there is application data to be sent back to the Sensor. During Data Mode the Hub periodically monitors the RSSI as well as the frequency of corrupted packets in order to determine the quality of the current channel. If the channel quality becomes too low the Hub will enter Channel Selection Mode to find a better channel.

2.7.6 Data Toggle

In order to guarantee data integrity a data toggle bit is used in Data Mode. Each Sensor and Hub maintains the state of their Sequence Bit, which is initialized to 0 during Bind Mode. The Sequence Bit is transmitted in the Data Packet header; the ACK packet contains a corresponding ACK bit that is the same value as the received Sequence Bit as shown in Figure 9. The Sequence Bit is toggled after an ACK is received in response to a transmitted Data Packet.

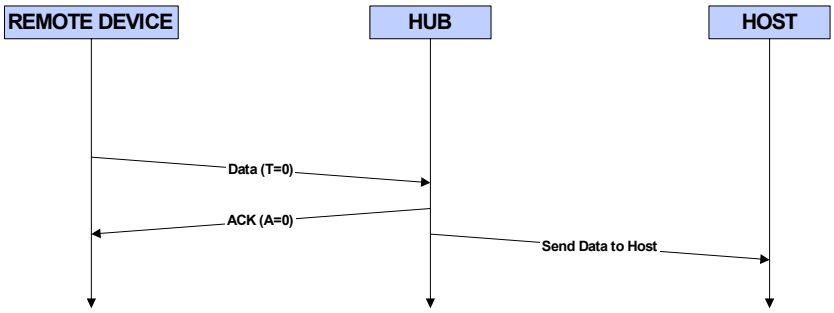


Figure 9: Data Mode: Basic Data Transfer

If the Sensor does not receive the ACK the Sensor retransmits the packet as shown in Figure 10 and Figure 11.

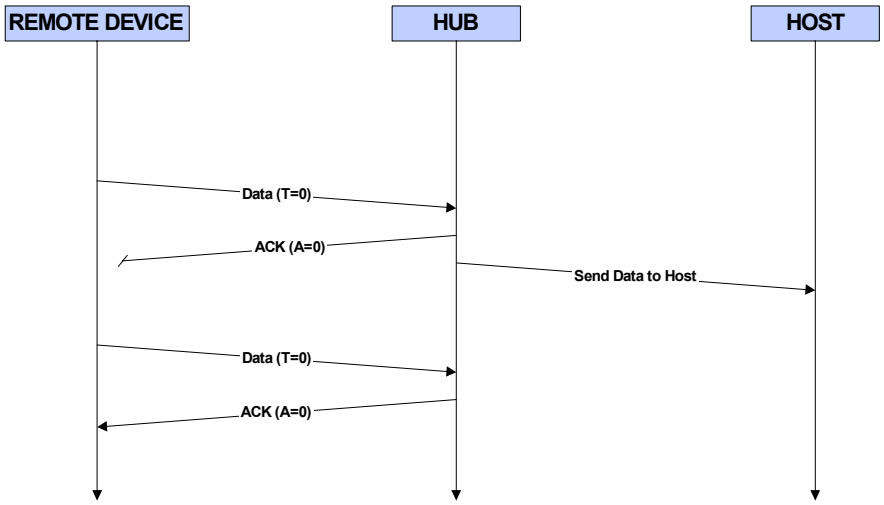


Figure 10: Data Mode: Dropped ACK

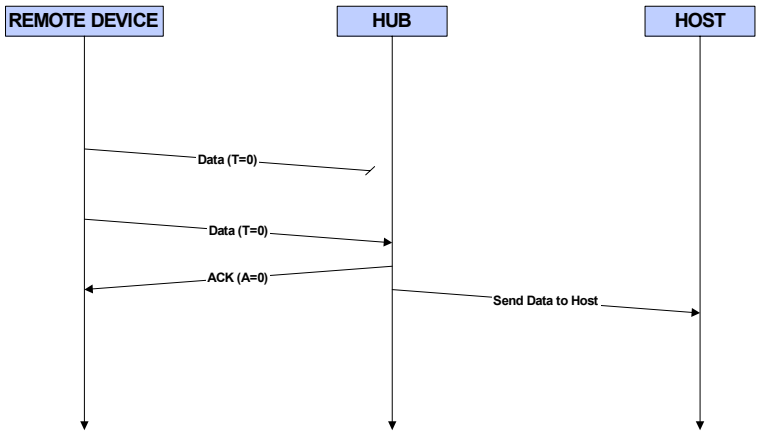


Figure 11: Data Mode: Retransmitted Packet

If the Hub has data to send to the Sensor, it responds to a received Data packet with a Data packet instead of an ACK packet as shown in Figure 12.

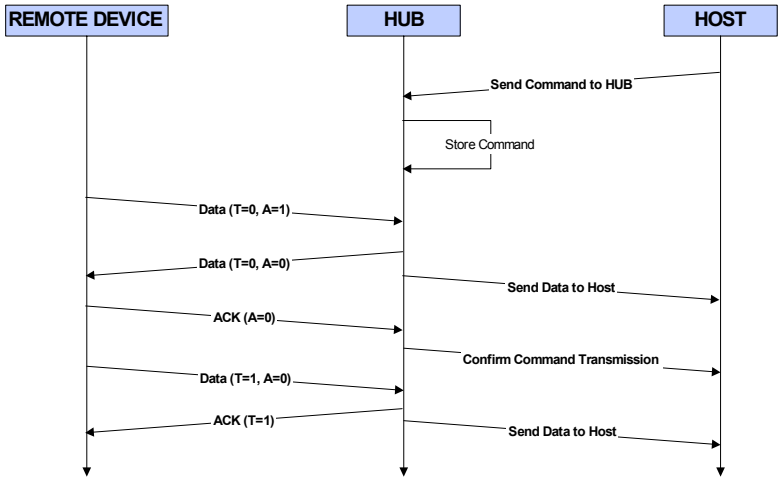


Figure 12: Data Mode: Back Channel Data

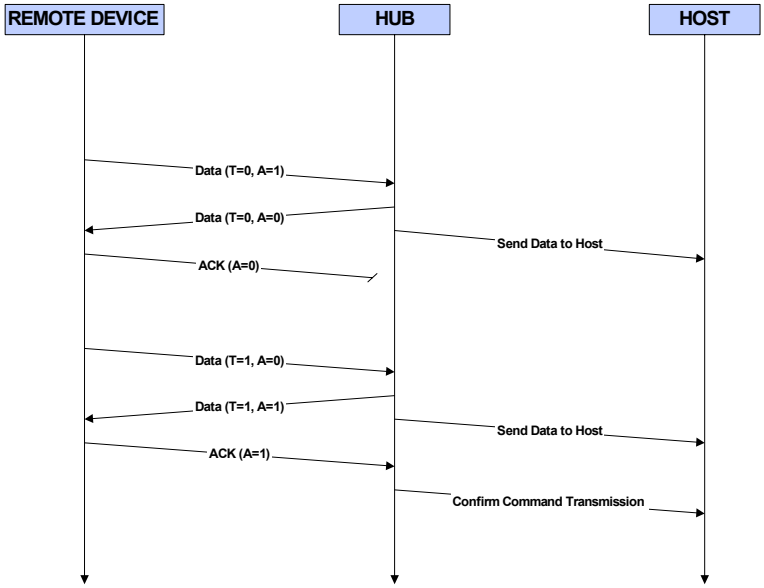


Figure 13: Data Mode: Back Channel Data with Lost ACK

2.7.7 Synchronous Data

If a Sensor has synchronous data to transmit to the Hub, such as environmental readings (temperature, humidity, etc) guaranteed packet delivery is not important, because each packet contains the current state of the Sensor. If the Synchronous Data flag is set in the header of a Data Packet the Hub will not discard the packet based on the Sequence Bit. Therefore all received Data Packets with the Synchronous Data bit set will be processed by the Hub, including duplicate packets.

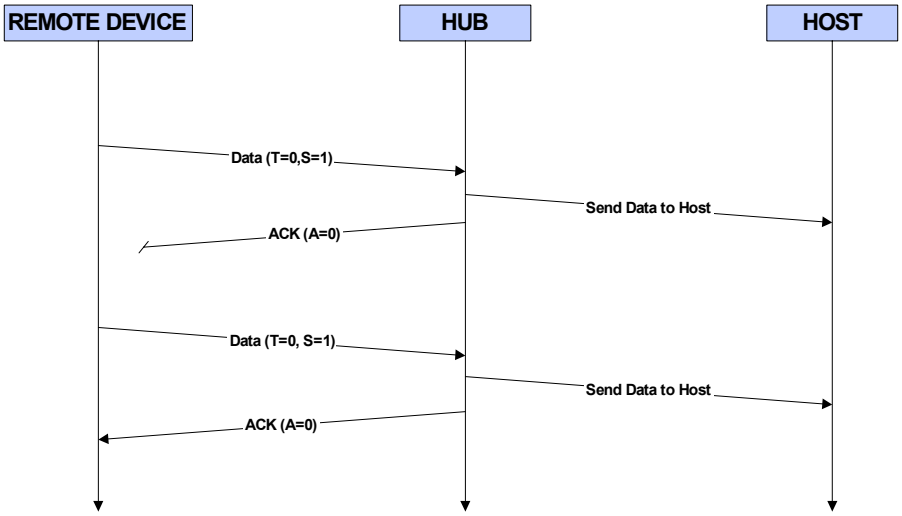


Figure 14: Data Mode with Synchronous Data

2.7.8 Invalid Device ID

If a Sensor receives an ACK packet with the Invalid Device ID Bit set the Sensor removes itself from the network and returns to an unbound state. If Seeded Bind Mode is available the Sensor may enter Seeded Bind Mode using the known Network Channel Subset and Network PN Code.

2.7.9 Sequence Bit Reset

If a Sensor receives a DATA packet with a payload length of zero the Sensor resets its Sequence Bit and ACK Toggle to zero. This allows the Hub to reset the sensor.

3. HARDWARE OVERVIEW

3.1 I/O pin assignments

The node board was designed to support several applications requiring some of the I/O pins to be used for multiple functions. In general Port0 is reserved for the analog related functions. The external crystal pins P1[0] and P1[1] (read as: Port 1 Pins 0 and 1) are shared with the ISSP programming header and LEDs. If an external crystal is needed (for more accurate timing) it may interfere with the ISSP function. Since the PSoC internal I2C hardware is used, the location of the I2C pins is somewhat limited. Please refer to the PSoC Datasheet for more information about pinout options and limitations.

When interfacing to the 3.3V radio with a 5V powered MCU, it is important to use series interface resistors on all signals being driven from the MCU to the radio.

3.2 Analog interface

The N:1 Node supports up to six analog inputs. Four of the inputs are dedicated to measuring the following:

- P0[7] Battery voltage
- P0[4] Thermistor
- P0[6] Potentiometer
- P0[5] Dip switches

P0[0], P0[1] can be used as general purpose analog inputs. In addition to the six analog inputs there are also two analog outputs which are used to output the internal A/D references, +Vref and –Vref. The Center point reference, or AGnd, is generated by the internal BandGap reference of 1.3 V. The references are 1.3 Volts on either side of AGnd or +2.6 and Ground. This arrangement was chosen in order to provide a stable reference for battery powered systems and allows for ratiometric measurements. All A/D measurements are single ended with offset correction.

Offset correction is performed prior to each analog reading. Internally AGnd is sampled by the A/D. The difference between the measured AGnd and the Ideal AGnd is used to adjust all analog reading to correct for offset and gain errors.

All of the analog hardware is referenced to –Vref. +Vref supplies power to the Thermistor and Potetiometer. While the PSoC is sleeping, -Vref is not driven in order to reduce the current drain on the battery.

More precise measurements can be taken by employing the following techniques:

- Measure the ratio of resistance (ratioMetric) in the case of thermistors.
- More elaborate gain and offset correction by feeding analog signals into a programmable gain amplifier with a gain of .9. This will allow both reference signals to be measured without causing the A/D to clip.
- Other references may provide better accuracy (Vcc/2) The band gap has an error of up to 2%.

3.2.1 Potentiometer

The potentiometer is provided as a demonstration of a generic analog input device. The potentiometer can be easily removed and replaced by almost any similar type of analog sensing device suitable for the desired application.

3.2.2 DIP Switches

The DIP switches were connected to a single analog input using a resistive ladder technique in order to reduce the number of PSoC pins utilized by the DIP switches. Depending on the application and if additional MCU pins are available, it may be more desireable to utilize conventional digital inputs for each DIP switch.

3.3 Sensor Battery Life

For most N:1 applications, the primary factor affecting battery life is the current drain while the PSoC is sleeping. The entire Sensor board consumes less than 10µA of current while sleeping, which provides a battery life of approximately 5 years. It is important to use a voltage regulator that has a very low quiesent current and ensure that the MCU is in a low power state with all relevant peripheral functions disabled. The regulator used in the N:1 DVK also has a low dropout voltage which allows more energy to be extraced from the batteries.

3.4 Prototype Expansion Header

Header J2/3 is designed for several purposes. J2 alone will allow connection to the prototype board via the 24 conductor ribbon cable. You can combine J2 and J3 as a set of eight, three pin connectors providing ground, Vcc, and signal. The PSoC can support a mixture of analog and digital input/outputs. As an example, if you wanted to connect a device that required power and outputs an analog voltage, you could connect the device across J2,Pin1/Pin2 and J3 Pin1.

3.5 Serial RAM (FRAM)

The Hub utilizes external non-volatile storage via the FRAM (Ramtron FM24CL64 64Kb (8KB) Serial 3v FRAM Memory). The FRAM was chosen for its quick write times and an almost unlimited number write/read cycles. EEPROMs have slower write times and some are specified for only 100k write cycles. The size was chosen based on several factors including cost and number of sensors supported. The current FRAM supports 512 devices (based on the default firmware configuration options). Each device has 16 bytes of storage allocated in a memory table. Two bytes contain status bits, four bytes contain the MID of the Sensor, 9 bytes are allocated for back-channel data, and one byte contains the back channel length. Because the Hub can only communicate with a Sensor when the Sensor initiates communication, there needs to be a way for the Hub to buffer and manage back channel data intended for each Sensor. The serial port latency is too great to let the Host retain this information to be retrieved on-demand. The Device ID of the sensor is used to index efficiently into the memory table.

Depending on the target application, the Hub may not require any external memory (for example, if there is no back-channel data required). The Hub may also utilize external EEPROM or other types of memory depending on the needs of the application.

3.6 Hub Power Considerations

The Hub was designed to run at 24MHz on the PSoC. This requires the PSoC be powered at 5 volts. This was done to provide the Hub as many MIPS as possible to service the 115.2k baud serial communication to the Host and SPI communication to the radio. It also demontrates how to use a 5 volt MCU with the 3.3 volt radio through the use of series interface resistors on signals being driven from the MCU to the radio.

If your application has looser requirements, the voltage can be reduced to 3.3 volts on the Hub and the PSoC run at 12MHz. Care should be taken to ensure that the worst case timing for serial communication or radio SPI communication will not interfere with any ISR's operation. This means keeping in mind that only one ISR is run at the time. If the radio ISR is running and a serial byte from the Host becomes pending in the PSoC, the serial receive ISR will be stalled until the radio ISR completes. If necessary, reduce the baud rate of serial communication through the PSoC Designer Device Editor in order to meet timing.

4. PROTOCOL/NETWORK MANAGEMENT

4.1 Network Parameters

In order for the Hub and Sensors to communicate, they must have the same set of network parameters. The network parameters consist of a Channel, PN Code Index, Checksum Seed, and CRC Seed. The network parameters are derived from the Hub's radio Manufacturing ID (MID) and can be overridden via the Host-to-Hub interface (except for Checksum Seed and CRC Seed).

4.1.1 Channel Subsets

The ISM band consists of the frequency range from 2.400-2.483 GHz and is divided into 1MHz channels. The DVK firmware uses the range 2.402-2.480 GHz which maps to channels 0-78 for a total of 79 channels. When the DVK reports that it's transmitting on channel 4, this equates to the frequency of 2.406 GHz, or channel + 2.

The 79 channels are organized into groups of possible Channel Configurations that define the spacing between channels and the number of channels in the channel sequence. A single Channel Configuration should be chosen that best fits the needs of the target application based on the expected number of devices in a single network and the number of co-located networks. For example, if a large number of co-located networks are anticipated, then the application should use a channel configuration that has a higher number of channel subsets. The following chart provides a summary of the channel usage.

Channel Configuration	# Subsets	# Channels per subset	Total Channels Used
1	6	13	78
2	7	11	77
3	8	10	72
4	9	8	72
5	10	8	70
6	11	7	77
7	12	6	72
8	13	6	78

Table 2: Channel Summary

For example, Index 2 for Channel Configuration #4 consists of the channels as follows:

2, 11, 20, 29, 38, 47, 56, 65

See the Appendix for a detailed list of channel usage.

An N:1 network will use all of the channels in the selected subset, if needed, in order to avoid noisy channels. Channel Subset 0 in all configurations is reserved for binding.

The Channel Configuration is defined in the firmware as `CHANNEL_CONFIG` in the `config.h` file. The N:1 DVK Kit is configured to use Channel Configuration #4. A Channel in the Channel Subset can be specified in the software application by selecting "Configure Network…" under the "Hub Command" menu.

If a channel subset other than the ones defined in this document is needed, it is important to try to avoid using an interval/channel spacing of 4. The radio receiver can pick up inverted signals at much lower power on the "image" frequency, which is 4 channels away from the channel you are transmitting on. This should not cause any problems since the data will be inverted and therefore the CRC will be corrupt.

4.1.2 PN Code ID

PN Code ID is short for Pseudo-random Noise Code Identification. The firmware contains one set of codes for 64kbps communication and one set for 16 kpbs communication. Cypress has created a set of PN Codes and associated a PN Code ID with them. The N:1 Kit uses a subset of these PN Codes as documented in the Appendix. The firmware uses simple array indexes to communicate which PN Code it is using; therefore, the Hub refers to a PN Code as a PN Code Index. The N:1 Software Application contains the table and translates between the array index from the Hub and translates it to a PN Code ID.

4.1.3 Device ID

A fixed 2-byte Device ID has been implemented for the N:1 DVK Kit because of its simplicity. Two other implementation methods include: single byte and dynamic. If your application does not anticipate more than 254 devices you could choose to modify the code to only support a single byte Device ID. A more code intensive option is the dynamic Device ID option. Those devices with a Device ID < 256 would save a transmission byte. In both cases, a bit in the header indicates whether the packet contains a 1-byte or 2-byte Device ID.

4.1.4 Checksum Seed and CRC Seed

Packets sent in the N:1 protocol are protected by a Checksum and CRC. In order to increase selectivity, they are seeded with values derived from the Hub's MID. This selectivity prevents two co-located Networks that happen to use the same PN code and same channel from experiencing cross-talk.

4.2 Binding Methods

Before a network has been established, the Sensors are considered to be unbound. The process of creating a network connection is called binding. During bind, a Sensor requests to join a network. In response, the Hub assigns the Sensor a Device ID and sends the Device ID along with the critical network parameters to the Sensor. The Hub stores this information in non-volatile memory and notifies the Host that a Sensor has been bound. The Sensor also stores the Device ID and network parameters in non-volatile memory. The Sensor can now communicate with Hub.

The N:1 system provides two binding methods: Automatic Bind and Seeded Bind. The method you select is dependant on the user experience and level of security required by your application.

After the network has been established, the Host and Sensor firmware read the bind parameters at startup and determines that it has been bound.

4.2.1 Automatic Bind

For Automatic Bind, the Sensor has no knowledge of the network it is to join and sends a Bind Request to the Hub on the universal Bind Channel Subset (Channel Subset 0). In order for Automatic Bind to work, the Hub must be in Bind Mode. The user can press the S1 button on the Hub to activate Bind Mode, or the Host can send a command to the Hub to enter Bind Mode. The N:1 Software Application provides a Bind toolbar button to initiate this command. In Bind Mode the Hub time-shares the bandwidth on the Bind Channel Subset and Network Channel Subset. This reduces the available connected network bandwidth to 50%; therefore, Bind Mode should be turned off when not needed.

The Sensor will only attempt an Automatic Bind if there are no bind parameters present in FLASH. Therefore, a Sensor will attempt an Automatic Bind after a Factory Reset.

4.2.2 Seeded Bind

For Seeded Bind, the Sensor knows only two of the basic network parameters: Channel Subset and PN Code Index. The Sensor will attempt to bind to the Hub via a Bind Request on the current Channel and PN Code Index to obtain the remaining network parameters: Device ID and Host MID (which is used to derive the CRC and XOR seed values). The Sensor obtains the Seeded Bind information from FLASH or from the DIP switches during startup. The N:1 Quick Start demonstrates use of Seeded Bind from FLASH mode for simple operation.

For demonstration purposes, the Sensor/Hub can be forced to perform a Seeded Bind on a subset of the pre-configured networks using the DIP switch settings as shown in Table 3. Your application is not required to use a DIP switch for Seeded Bind.

To perform a Seeded Bind, set the first three DIP switches to the desired setting based on the table below. S1 needs to be depressed when the PSoC comes out of reset in order for the firmware to perform a Seeded Bind based on the DIP switch setting. Hold down the Reset button and the S1 button together. Release the Reset button first, and then release the S1 button.

DIP [1..3]	Channel Index	PN Code Index
000	1	1
001	2	2
010	3	3
011	4	4
100	5	5
101	1	7
110	2	6
111	5	5

Table 3: Channel and PN Code Index Selection

4.2.3 Automatic vs. Seeded Bind Selection

Which binding method your application uses depends on how you feel about the following issues. The N:1 Kit uses Seeded Bind because of its simplicity and security is of lesser concern.

Automatic Bind is more secure than Seeded Bind because Sensors can join the network only when a user has access to the Hub. Because Hub intervention is required it is not as easy to establish a network as Seeded Bind.

Hub support for Seeded Bind is turned on by default in the protocol. This makes Seeded Bind easy to setup but less secure over time.

Manufacturing considerations for Seeded Bind are: Will the same FLASH based Seeded Bind parameters work for all your kits? Will manufacturing need to cycle through a set of Seeded Bind parameters? Automatic Bind doesn't have these issues because it relies on the FLASH being the same/invalid for all the Sensors.

Since Sensors in Seeded Bind use the Network Channel Subset to communicate, you don't have to dedicate 50% of your bandwidth to the task of servicing Bind.

The Host Application can disable Seeded Bind. Your application can get the benefits of Seeded Bind and the lock down of Automatic Bind by turning off Seeded Bind when the user has completed setup of the network.

5. RADIO OVERVIEW

5.1 Radio timing

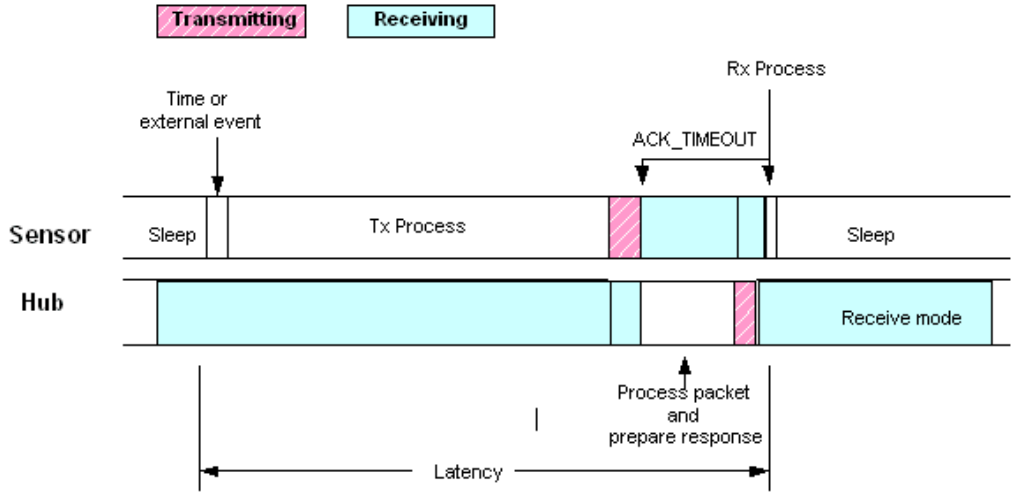


Figure 15: Normal Host to Hub transmission

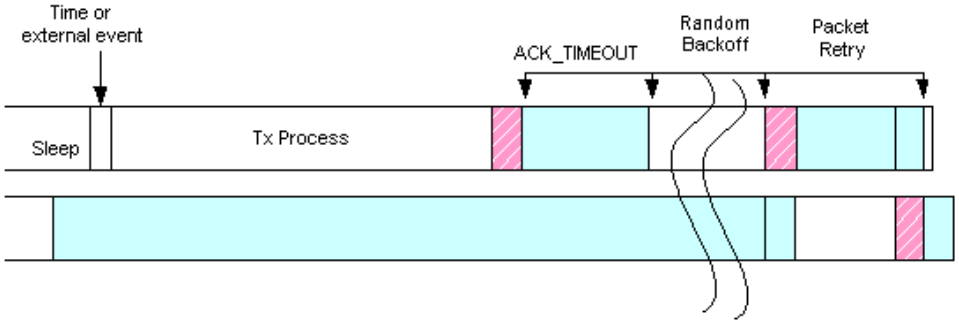


Figure 16: Packet Retry timing

5.2 Maximizing Range

Several factors can influence the range of the network. The N:1 network will operate reliably up to 50M in most environments. In some environments, the useful range can be over 100M.

Environmental conditions and antena orientation are the two primary factors that can degrade the system range.

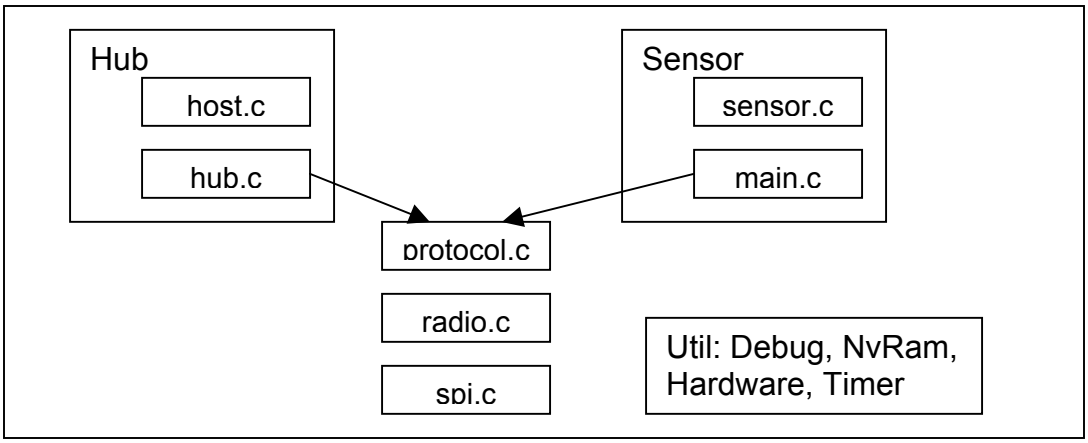
The Dynamic Network Quality test is useful for understanding the link quality of a sensor at a given range. Refer to the *N:1 DVK Users Guide*, for information about this sensor feature.

For applications that require greater range, an external power amplifier can be used.

6. FIRMWARE ARCHITECTURE

6.1 Theory of Operation

This diagram gives you a rough idea of how the files are logically structured. The files outside the Sensor and Hub are shared between the two projects. Exact copies reside in each project directory for easy intergration into PSoC Designer.



The N:1 DVK Kit demonstrates the N:1 Protocol's connection establishement and distribution of a packet, with user payload, between a Sensor and Hub. Using the N:1 Software Application you can also demonstrate back channel data from the Hub to a Sensor.

The Checksum and CRC have been placed at the end to allow the option of computing them during transmit time. The code in the Kit was written to compute them ahead of time.

To keep things simple and elegant, the N:1 v1.0 protocol uses 2-byte Device IDs exclusively. Given the low update rate of the devices an additional byte is insignificant.

The radio pins are on PSoC's Port 2 instead of Port 1 because Port 2's pins are sufficient to drive the radio. It is more important to leave Port 1 available because of its flexibility.

All of the time-critical radio and serial communication tasks are interrupt driven.

The following chart shows how much time you can starve the Radio's TX empty IRQ before disrupting the packet transfer. The number is derived by subtracting one bit-time unit from a byte. In Double Data Rate (DDR) each bit transmitted represents two bits. 32µs of preamble will be transmitted if the transmit buffer is empty at the moment it needs to start transmit.

64 Chips Per Bit: 7 bits x 64 µs = 448µs
32 Chips Per Bit - Double Data Rate: 3 x 32 DDR µs = 96µs

This means that servicing the radio cannot take longer than 96µs plus the time to initiate the TX ISR to the point where it writes the next byte in the buffer in DDR mode.

The receive buffer is either protocol or radio owned. Ownership is determined by the state of the Radio ISR. If the Radio ISR is enabled then the RX buffer is radio owned. When it is off, it is protocol owned. This also means that when transmitting, the receiver buffer is not available to the protocol.

On the Sensor the Green LED is on while the sensor is awake.

6.2 Interference Avoidance

Please refer to the Application Note "WirelessUSB LS Theory of Operation" that was included in the Kit CD-ROM for background.

There are many sources of interference including WiFi, Bluetooth, and other N:1 co-located networks. The N:1 protocol uses coding gain and interference avoidance to coexistence with these interference sources.

Three interference avoidance algorithms have been implemented in the N:1 DVK Kit: Ping, RSSI, and Bad Packet Count.

Ping: Before the Hub uses a channel it first broadcasts ping packets on the channel. If another Hub is already on the channel it will respond to the ping packet, the Hub that originated the ping will move to the next channel.

RSSI: During periods of N:1 network inactivity a reading of the ambiant noise is read. If the rolling average exceeds a threshold then the Hub will change channels.

Bad Packet Count: A count is kept for every bad packet. If a threshold is reached the Hub will change channels. When a good packet is received Bad Packet Count is decremented.

6.3 Interface Between Hub and Sensor

This explains the packet structure for the over the air interface between the Hub and Sensor. Note: Unshaded bytes refer to the user payload.

Forward channel, Sensor to Hub,

Fixed length packet [10 Bytes], Packet type 4

Byte 1	2	3	4	5	6
Header	Device ID		Battery	Potenio-meter	Temp
	MSB	LSB			

Byte 7	8	9	10
Dip Switch	CRC		XOR Checksum
	MSB	LSB	

Vairable length packet [11-17 Bytes], Packet type 5

Byte 1	2	3	4	5	6
Header	Device ID		Battery	Potenio-meter	Temp
	MSB	LSB			

Byte 7	8 .. (*n* <= *14)*	Length - 2	Length - 1	Length
Dip Switch	Data	CRC		XOR Checksum
		MSB	LSB	

Bat = unsigned 8 bit value of scaled battery voltage
Pot = unsigned 8 bit value of Pot voltage
Temp = unsigned 8 bit value of thermistor voltage divider.
Switches [Red LED state, Yellow LED state, Reed Switch, Switch S1,Dip[3:0]]
Data = Variable length user data 7 bytes max

Back channel, Hub to Sensor,

Fixed length packet [10 Bytes], Packet type 4

Byte 1	2	3	4	5	6
Header	Device ID		Interval		LED
	MSB	LSB	MSB	LSB	

Byte 7	8	9	10
Reserved	CRC		XOR

	MSB	LSB	Checksum

Vairable length packet [11-15 Bytes], Packet type 5

Byte 1	2	3	4	5	6
Header	Device ID		Interval		LED
	MSB	LSB	MSB	LSB	

Byte 7	8 .. (*n* <= *12)*	Length - 2	Length - 1	Length
Dip Switch	Data	CRC		XOR Checksum
		MSB	LSB	

IntMsb = 8 Upper bits of 16 bit interval.
IntLsb = 8 Lower bits of 16 bit interval.
Reserved/LED = bits 7-3, Reserved, bits 2-0, LED data for LED3/2/1,respectively, 1= On
Reserved
Data = Variable length user data 5 bytes max

Variable length packets are used by the serial user data applications. Refer to the users guide.

Note: Interval is defined in terms of .125 seconds and defines the time between reports from the sensor. 16 bits support a rage of .125 seconds to 136 minutes (2.27 Hours).

6.4 Interface Between Host and Hub

The protocol between the Host and Hub for the N:1 Product is defined as firmware that runs on an MCU and optionally communicates to a Host via UART or RS-232. The Host is any device with a UART or RS-232 interface that might provide further functionality based on this protocol. Using this interface is not a requirement – the Hub will function without communications with a Host.

All the commands from the Host are synchronous. The Host issues a command and then waits for a response from the Hub. Unless otherwise noted, the Host should complete a command before another command is started. Responses from the Hub do not require a response from the Host.

The Host-to-Hub serial communication will use the Consistent Overhed Byte Stuffing (COBS) algorithms for packet framing. The maximum command length will be determined at compile time. The COBS protocol does not transfer a length byte. The COBS framing byte of 0x00 determines length.

Set all undocumented bits or reserved bits to zero unless otherwise noted.

Host-To-Hub Command Summary

Command	ID	# Bytes
Get Hub Information	0x01	1
Bind	0x02	2
Delete Node	0x04	3
Send Message	0x05	5 <= n <= 14
Enumerate Devices	0x07	1
Network Configuration	0x08	3 or 7
Network Status	0x09	1
Reset	0x0A	1
Change Channel	0x0B	2

Hub-To-Host Response Summary

Command	ID	# Bytes
Hub Info Response	0x81	11
Bind Response	0x82	2
Bind Information Response	0x83	7
Delete Node Response	0x84	4
Send Message Response	0x85	4
Incoming Message	0x86	4 <= n <= 12
Enumerate Devices Response	0x87	4
Network Configure Response	0x88	2
Network Status Response	0x89	5
Reset	0x8A	2
Change Channel Response	0x8B	3
Unknown Command Response	0xFF	2

6.4.1 Host Commands to the Hub

6.4.1.1 Get Hub Information

Byte 1
Command 0x01

The Hub returns information about itself when the Host sends this command.

6.4.1.2 Bind

Byte 1	2
Command 0x02	Options

The Host turns on Bind to allow devices to join the network.

Byte: Options
Bit 0: Reserved
Bit 1: 0 = Bind Off
1 = Bind On
Bit 2: 0 = Seeded Bind Mode
1 = Disable Seeded Bind Mode
Bits 3-7: Reserved

When enabled Seeded Bind allows devices that have been pre-programmed with the PN Code Index and Channel to join the network from the Hub's connect state. The Hub does not need to be in bind mode.

6.4.1.3 Delete Node

Byte 1	2	3
Command 0x04	Device ID	
	MSB	LSB

The Delete Node command instructs the Hub to disconnect the specified Device ID (and ignore any subsequent packets). If 0xFFFF is passed in for Device ID then the Hub will delete all devices from the device table.

6.4.1.4 Send Message

Byte 1	2	3	4	5… <= 14
Command 0x05	Device ID		Options	Payload
	MSB	LSB		

The Hub uses this command to send data to a device.

Byte 4: Options

Bits 0-4: Number of transmit attempts before giving up. Value of 0 indicates infinite retries. (Not implemented in v1.0)
Bit 6: Reserved: Used by the Hub to know when there is back channel data.
Bit 7: 0 = Notify when Message Sent
1 = Inhibit Notify when Message Sent

Bytes 5 to <= 14: Payload to send to the Device ID. The 9-byte limitation is based on the amount of space available in a 16-byte page of the FRAM minus overhead.

This command will result in two responses from the Hub. The first response is immediate to acknowledge receipt of the command, and the second response will be sent after the payload is sent to the sensor.

6.4.1.5 Enumerate Devices

Byte 1
Command 0x07

When this command is issued the Hub will parse through its device table and return individual Bind Information Responses for each device in the table. When the Hub is finished it will return an Enumerate Device Response. This command is unique in the amount of traffic it will generate from the Hub is dependant on how many devices are connected. While the Hub is processing this command all wireless traffic will be NAKed.

6.4.1.6 Network Configuration

Byte 1	2	3	4	5	6	7
Command 0x08	PN Code Index	Channel	Hub MID (Optional)			
			MSB			LSB

The Host issues this command when it wants to specify the network parameters. This command will result in the Hub erasing all devices from memory. If the Hub MID is not supplied then the existing value is used.

PN Code Index and Channel cannot use the Bind PN Code Index (0x00) or the starting Bind Channel (0x00). If these values are passed in the new value will be 0x01. The upper bounds of PN Code Index and Channel are limited to the Hubs current configuration. Values passed in greater than the max value will be set to the max value. Remember: PN Code Index and PN Code ID are not one and the same – see source code in *radio.c* for translation.

6.4.1.7 Network Status

Byte 1
Command 0x09

The Hub will return its status when this command is issued.

6.4.1.8 Reset

Byte 1
Command 0x0A

When this command is issued the Hub will perform a firmware reset. Non-volatile memory will not be cleared.

6.4.1.9 Change Channel

Byte 1	2
Command 0x0B	Channel

Passing in 0xFF with this command will cause the Hub to move to the next available channel based on the channel sequence. If another Hub is on the next channel the Hub will move again to the next channel.

Advanced Debug: This command is capable of setting the radio to a specific channel. To do this correctly requires the Host have specific knowledge about the channel sequence algorithm. Therefore, it is not recommend this command be used in this way. The Hub does not verify the new channel is in the channel sequence and will set the radio to the specified channel.

6.4.1.10 Miscellaneous

The Host does not have the ability to add a sensor (via its MID) to the Hub. This would preclude customers from entering a Sensor's MID on the Host GUI. The reason is the Hub would need to scan through all its issued Device IDs to look for a MID when a sensor is bound – the amount of time required would become increasingly prohibitive in a large network. Implications: Because the Hub does not check for a preexisting MID then a sensor that is reset and rebound will be taking up two Device IDs on the Hub. Software could be written detect this condition and delete the non-used duplicate Device ID.

6.4.2 Hub Responses to Host Commands

6.4.2.1 Hub Info Response

Byte 1	2	3	4
Response 0x81	Firmware Major Ver #	Firmware Minor Ver #	Firmware Build #

Byte 5	6	7	8	9	10	11
Hub MID				Max Device IDs		Radio Version #
MSB			LSB	MSB	LSB	

The firmware version is a binary-encoded decimal number.

6.4.2.2 Bind Response

Byte 1	2
Response 0x82	Status

Byte Status: Success

6.4.2.3 Bind Information Response

Byte 1	2	3	4	5	6	7
Response 0x83	Device ID		Sensor MID			
	MSB	LSB	MSB			LSB

The Hub will return one Bind Information Response per node. If Device ID is 0xFFFF the Hub has run out of assignable Device IDs and the Sensor was not bound.

6.4.2.4 Delete Node Response

Byte 1	2	3	4
Response 0x84	Device ID		Status
	MSB	LSB	

Byte Device ID: Device ID of deleted node

Byte Status: Success, Unknown Device ID

6.4.2.5 Send Message Response

Byte 1	2	3	4
Response 0x85	Device ID		Status
	MSB	LSB	

The Hub will send two Send Message Responses. One when the wireless packet has been queued and once when the wireless packet has been sent.

Byte Status: Success (Packet Transmitted), Unknown Device ID, Send Buffer Too Large, Message Queued, Prior Send Payload Lost (Message Queued is implied)

6.4.2.6 Incoming Message

Byte 1	2	3	4 … <= 12
Response 0x86	Device ID		Payload
	MSB	LSB	

This is the received payload from Device ID specified. It is sent asynchronously (not in response to a command from the Host).

6.4.2.7 Enumerate Devices Response

Byte 1	2	3	4
Response 0x87	Status	# of Devices Enumerated	
		MSB	LSB

After all the devices have been passed up as Bind Information Responses the Hub will send this response to indicate it is done enumerating all of the connected devices.

Byte Status: Success

6.4.2.8 Network Configure Response

Byte 1	2
Response 0x88	Status

Byte Status: Success, PN Code Index Invalid, Channel Invalid

Because the status cannot be combined, it is possible to have both a PN Code Index Invalid and Channel Invalid but only receive a Channel Invalid. When an error occurs, issue the Network Status Command to find out the actual network configuration.

6.4.2.9 Network Status Response

Byte 1	2	3	4	5
Response 0x89	Current Channel	Current PN Code Index	Data Rate	Bind Mode

Byte Current Channel: The current channel the Hub is communicating on.

Byte Current PN Code Index: The current PN Code Index the Hub is communicating on. Refer to *radio.c* to convert from a PN Code Index to a PN Code ID that can be used by the user, Listener, etc.

Byte Data Rate: The wireless data rate used by the Hub (16 = 16kbps, 64 = 62.5kbps)

Byte Bind Status:

Bit 0: Reserved
Bit 1: 0 = Bind Off
1 = Bind On
Bit 2-7: Reserved

This response will be sent asynchronously if the channel changes due to interference. This Response is expected to grow as more diagnostic information is determined would be useful at the Host level.

6.4.2.10 Reset

Byte 1	2
Response 0x8A	Status

Byte Status: Success

6.4.2.11 Change Channel Response

Byte 1	2	3
Response 0x8B	Status	New Channel

Byte Status: Success, Invalid Channel

Byte New Channel: The channel the radio was set to.

6.4.2.12 Unknown Command Response

Byte 1	2
Response 0xFF	Command

The command code was not recognized as a valid code

Byte Command: The command the Hub couldn't interpret.

6.4.2.13 STATUS

Status Codes

Code	Status	Description
0x00	Success	The command was executed successfully
0x01	Undefined	
0x02	Unknown Device ID	The specified Device ID was not valid for any open connections.
0x03	Send Buffer Too Large	The specified send data buffer was too large
0x04	Prior Send Payload Lost	The specified send data buffer has been received before the prior buffer was sent. The new data has been overwritten the old.
0x05	PN Code Index Invalid	The specified PN Code Index was outside the acceptable range
0x06	Channel Invalid	The specified Channel was outside the acceptable range
0x07	Message Queued	The back channel data has been queued for transmit
0x08	Invalid Channel	The channel parameter passed in was out of range
0x09-0xFF	Undefined	

6.5 Serial Output on the Host

By default serial ouput is disable on the Host. When `DEBUG` is defined, a secondary Serial TX block on the PSoC is utilized to send debug output to a communications program. Connect a wire to the Hub Node Board's J2 pin 3 to the Serial Board's J1 pin 8. The same communication port parameters that apply to J1/UART apply here:

- 115.2kbaud
- No parity
- 8 data bits
- 1 stop bit
- No flow control

Serial output on the Hub is concise at 1 character per event. This is due to the latency introduced by writing a sequence of bytes out the serial port. For the Hub a character can be written to the serial port and then execution continues (the code is written to wait for the buffer to be empty before writing the next character). The following reference table documents each character's meaning. (Note: Some characters related to receiving a command overlap with other meanings for the character. This shouldn't be confusing because the printing of the character associated with a command occurs after the received command is dumped to the serial output.)

Char	Meaning
A	`HOST_CMD_GET_HUB_INFO` received. It can also mean an ACK was received in response to a Bind Response Packet for the just issued Device ID
B	Bad bit detected when processing packet received over the air.
b	The back channel data was sent to the Sensor. The Hub is about to send a `HUB_RSP_SEND_MSG` to the Host
C	`HOST_CMD_DELETE_NODE` received
D	`HUB_RSP_SEND_MSG` received
F	`HOST_CMD_NETWORK_CONFIG` received
G	`HOST_CMD_NETWORK_STATUS` received
H	`HUB_RSP_RESET` received
I	`HOST_CMD_CHANGE_CHANNEL` received
L	No Device IDs are available. Happens in response to a Bind.
M	Radio Receive Buffer overflow
P	A protocol packet was received by the Hub.
R	Sent response to Bind Request Packet
s	Sensor found – occurs while enumerating all the Sensors
X	When `CORRUPT_DATA` is defined will print out this character each time a packet is intentionally corrupted.

Char	Meaning
Z	Received a data packet with a 0 length payload (when a 0 length data payload is received the Hub resets its SEQN bits)
?	Unknown command from the Host. It can also mean we were in the process of receiving a packet when we were going to take an RSSI reading.
%	Command from Host had unexepected length
>	A command was received from the host; length of command; colon; command printed out. A set of two Greater Than characters, each followed by a byte, indicate the status of Sensor SEQN bits and Host SEQN bits respectively.
^	When `PRINT_HOST_TX_PACKET` is defined, the transmitted packet is printed out as each byte is written to the radio. This process is bracketed with a caret.
_	A packet was received (underscore character)
+	Instead of ACKing the Sensor we need to send a back channel packet.
:	When finding an available channel a colon is printed each time a ping packet is sent
¡	Bad packet was received – checksum and CRC failed
*	Bit inversion detected
$	The radio's receive buffer wasn't serviced fast enough and a byte was lost in the radio.

7. FIRMWARE CUSTOMIZATION

7.1 Development Environment

The firmware was development was done on Microsoft Windows PCs using Cypress's PSoC Designer version 4.1 with Service Pack 1 installed. The system requirements for this revision of PSoC Designer are Windows 98, NT4.0 (SP6), 2000, ME, and XP. At the time of this writing, PSoC Designer version 4.2 is available but does not work with the N:1 DVK Kit. Developers must uninstall 4.2 and install 4.1 to use the N:1 DVK Kit.

The firmware was written in C with the included ImageCraft C Compiler version 1.28. An ImageCraft compiler license needs to be purchased in order to compile the code.

The normal development cycle is to compile the code and download it to the PSoC on the N:1 Node Board via the ISP. For more interactive development, replace the PSoC on the N:1 Node Board with a 28-pin SSOP Foot Kit (CY3203-080) and attached a 8C27002 PSoC Pod on the 28-pin SSOP Foot. Code can then be emulated using a PSoC ICE-4000.

7.1.1 PSoC Designer 4.1 Patch Files

There are two additional files that need to be updated at this time and they can be found on the CD-ROM `\Software\PSoC Designer\PSoC Designer Patch\`. Copy the two updated files in this directory to the following locations (the default path for the PSoC program files is `C:\Program Files\Cypress MicroSystems\PSoC Designer`).

`\PSoC Designer\Data\Stdum\I2CHW\I2CHWCommon.inc`

`\PSoC Designer\Data\Stdum\I2CHW\CY8C27\Master\MasterMstr.asm`

7.2 Data Rates

The N:1 Kit defines two different data rates. 62.5 kbps (sometimes referred to as 64kbps in documentation for simplicity) and 15.625 kbps (referred to as 16kbps). The trade-off is the slower data rate is less susceptible to interference and has slightly greater range while the faster data rate has bandwidth and battery life advantages. The N:1 kit is built with 16kpbs to demonstrate maximum range and robustness.

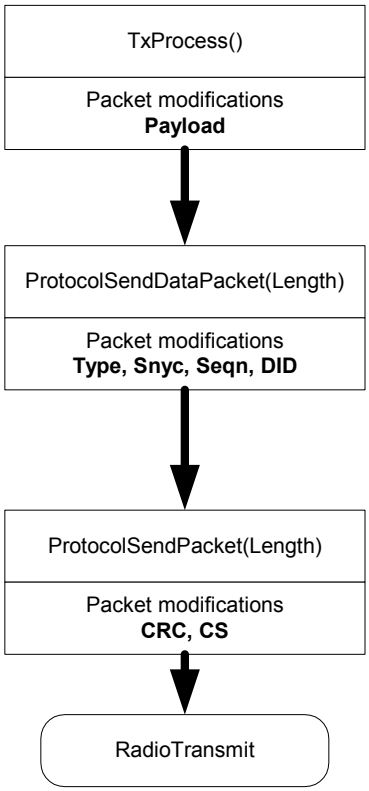
7.3 Host Baud Rate

The N:1 kit was built with a Baud Rate of 115,200 for serial communication between the Host and the Hub. This can be changed by editing the PSoC Designer Project. Change the clock used by the serial communication blocks. Flow control pins can also be implemented but were not done so in order to reduce pin count on the part.

7.4 Protocol API

All API functions are contained in the files *protocol.c* and *radio.c* The intent of these functions is to provide the user with a consistent and easy to use interface for sending and receiving data as well as managing the radio.

7.4.1 TX data flow

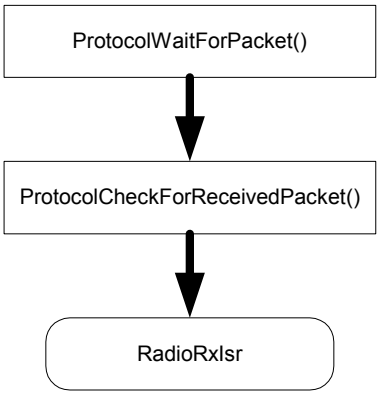


`TxProcess()` is where the application generates the data to be transmitted to the host. The N:1 kit samples all of the analog and switch inputs. This function is responsible for updating the payload portion of the `gsTxPacket` structure.

`ProtocolSendDataPacket()` is designed to transmit both fixed and variable packets. The Length parameter will be used to determine which type to use. This function will update the packet header byte and the DID fields. The packet is complete except for the CRC and CS.

`ProtocolSendPacket()` will then compute the CRC and CS for the now complete packet, and call RadioTransmit to actually transmit the packet .

7.4.2 Receive data flow:



`ProtocolWaitForPacket()` starts the receive process. This function will turn the receiver on and wait for a packet to be received or time out if no packet is received in the specified time.

`ProtocolCheckForReceivedPacket()` is called by `ProtocolWaitForPacket()` and is resposible for detecting a packet end of frame condition and verifying that the packet is error free. `ProtocolCheckForReceivedPacket()` will return `PACKET_RECEIVED` when complete and if a valid packet has been received.

`RadioRxIsr()` is resposible for process the the receive interrupts from the radio. Each interrupt will trigger this function to extract one byte of data and one byte of valid bits for the received byte. The valid byte is used in the bit correction XOR checksum.

7.4.3 Sensor Binding process

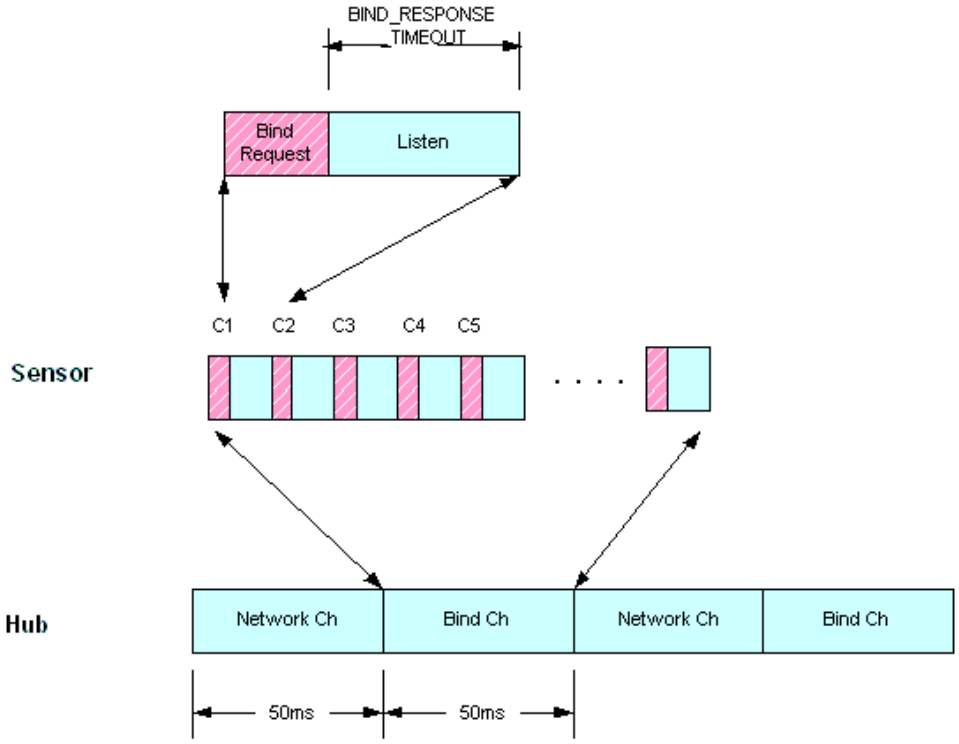
The N:1 sensor binding process consists of an application level and a protocol level. The application level provides the application specific user interface to the bind process. The protocol level is responsible for communicating with the hub to determie the bind parameters.

`SensorSetBindState()` is responsible for determining the bind state based on the FLASH bind parameters and/or dip switch state. `SensorSetBindState()` updates the `gu8ProtocolStatus` state flages `SEEDED_BIND` and `NODE_BOUND` and loads the variables ChannelIndex (CNI) and PnCodeIndex (PNI) with the correct values. For the non-bound state, CNI and PNI are set to the default bind values defined by `BIND_PN_CODE_INDEX` and `BIND_CHAN_INDEX`. These values should be set to 0/0 which are the PIN and CNI reserved for binding. `ProtocolBind()` is then called to perform the bind process.

If only the PNI/CNI FLASH parameters are valid, then only the PNI/CNI variables are updated from FLASH and the flag `SEEDED_BIND` is set. `ProtocolBind()` will then bind on the network PNI/CNI rather than PNI/CNI = 0 which is reserved for Automatic binding.

If the sensor was previously bound and the bind parameters in FLASH are valid, then the CNI/PNI will be loaded with the values in FLASH and the state flag `NODE_BOUND` will be set.

`ProtocolBind()` will attempt to bind on the channel and PN index defined by the variables ChannelIndex and PnCodeIndex. The following timing is used in the Automatic bind case when the sensor is attempting to establish a connection with the hub and exchange bind parameters. The Hub must have Bind enabled.



When the bind process is successful all of the bind parameters are stored in FLASH.

7.5 Firmware Configuration

These configuration options can be found in `config.h`. For some options, it is recommend that they be specified on the command line. Those options are noted with an asterisk (*) here and with “Note A” in the .h file. This allows the same unmodified source files be compiled for the Hub or Sensor.

7.5.1 Config.h

HUB & SENSOR* – These implicitly get tested for each build.
DATA_RATE - 16kbps or 64kbps
CHANNEL_CONFIG - Range 1-8. See Channel usage section in the users guide.
DEBUG* – Turns on and off debug
PRINT_TX_PACKET* - Prints Transmitted Packet
PRINT_RX_PACKET* - Prints Received Packet
PRINT_PACKET_CORRECTION_RESULTS* – Did I get a bad bit, etc.
PRINT_HOST_TX_PACKET* – (Hub) Prints serial packet sent to Host
SCOPE_PIN* - (Hub) Port 0 Pin 5 can be used as a debug pin
DEFAULT_XTL – 0x00 – 0x3F
PA_BIAS – 0x00 – 0x07
DEFAULT_INTERVAL [40]– (Sensor) Initial interval of sensor when connected and has not received an interval from the Hub is 5 seconds. Interval is in units of .125 seconds.
DEFAULT_NON_BOUND_INTERVAL [480]– (Sensor) Interval of sensor when communication with the Hub has been lost. This interval is currently set at 60 seconds .
ENABLE_SYSTEM_TEST* -(Sensor) Will compile in the ability to run 100 packet latency test as well as printing the latency of each packet to the debug port. NOTE: Enabling this define will also enable ENABLE_FAST_SWITCH_EVENTS.
ENABLE_ICE_SUPPORT * - (Sensor) Uses while loops in place of CPU sleep in order to work with the PSoC ICE.

7.5.2 Main.c/h (Hub and Sensor)

ENABLE_FAST_SWITCH_EVENTS. - When enabled, switch events (S1 and reed) will bypass the A/D process prior to transmitting in order to reduce the event latency. Analog data from the previous normal sample will be transmitted for these events.
RETURN_CHAR – Character appended to the end of the user serial data for packets less than the maximum size.

7.5.2.1 Protocol.h

ACK_TIMEOUT [10] – Number of mS that the Hub or sensor will wait for an ack packet before giving up.
BIND_RESPONSE_TIMEOUT [6] – Number of mS that the sensor will wait before giving up
BIND_CHECKSUM_SEED [0] – Seed value used for bind packets
BIND_CRC_SEED [0] – Seed value used for bind packets
BIND_PN_CODE_INDEX [0] – PN index used for binding
BIND_CHAN_INDEX [0] – Channel index used for binding
MAX_PACKET_SIZE [16] – Sets the limit for the total size of the data packets, Payload + protocol bytes
NUM_OF_CHANNEL_SEARCH_PINGS [10] – Number of pins the Hub will transmit while looking for a new, clear, channel.
NACK_RETRY_ATTEMPTS [7] – Number of times that the sensor will transmit a packet in the event that it doesn't receive a valid ACK from the Hub.
SEARCH_PACKET_REPEATS [3] – (Sensor) Number of times the sensor will transmit on the same channel while searching for the Hub.
SEARCH_PACKET_DELAY [20] – (Sensor) Delay in ms between transmitted packets.

NUMBER_BIND_CYCYLES [2] – (Sensor) Number of times the sensor will go through the entire channel sequence.

BIND_TX_PER_CH [2] - (Sensor) Number of bind packets transmitted on each channel.

SEEDED_BIND_CHI [1] – Channel index burned into FLASH during device programming causing the device to be factory seeded bound

SEEDED_BIND_PNI [9] - PN index burned into FLASH during device programming causing the device to be factory seeded bound

SEEDED_BIND_CRC [0X84] CRC burned into FLASH for the prior two values. The CRC must be recomputed if the seeded_bind_chi/pni are changed.

7.6 Porting considerations

Porting the N:1 FW to another processor is complicated by the use of PSoC analog and digital blocks and the uique interrupt structure of the PSoC.

7.6.1 Interrupts

Both the Sensor and Hub use C and assembly interrupt service routines. ISRs written in C use the #pragma calling format. Sensor interrupts used for purposes of awaking the processor from the sleep state don't actually provide code to service the interrupt. These interrupts are allowed to fire at the hardware level but are blocked by disabling global interrupts and not allowed to execute any code. The following table outlines all of the interrupts in use excluding any interrupts used by any PSoC user modules.

Source	C/Asm	Function	Location	Description
DCB12	C	SerialRxIsr	SensorC	Processes serial data after first serial byte has awaken the CPU via DBB11 below
		HostSerialRxIsr	Host.c	Accepts serial data from the Host PC
DCB13	C	HostSerialTxIsr	Host.C	Sends serial data to the Host PC.
DCB02	C	MsTimerTick	Timer.C	Generates background 1ms timer ticks for timing events
GPIO pin	C/Asm	IsrGpio	Radio.C, PSoCGPIOINT.asm	Processes radio interrupts when awake. When asleep allows S1 and the reed switch to awake the CPU (no ISR code is executed)
DBB01	NA	NA	NA	16-bit Sleep timer terminal count

Source	C/Asm	Function	Location	Description
DBB11	NA	NA	NA	Detects serial RX activity and generates an interrupt to awake the CPU.

7.6.2 PSoC Analog and Digital Block Usage

Sensor

In addition to the standard PSoC API blocks, the sensor configures DBB00/01/11 directly by writing to the underlying block configuration registers. DBB00/01 are reconfigured as a 16-bit sleep timer while the device is sleeping in order to improve the interval accuracy. When exiting from sleep, DBB01 is reconfigured as a DELSIG11 as indicated in the PSoC Designer Device Editor. DBB11 is initially configured as a PRS8 module. This block is then modified to generate interrupts when the cpu is sleeping based on serial RX activity on the global row input RI1[3].

DBB00	LSB of 16-bit sleep timer
DBB01	MSB of 16-bit sleep timer when sleeping, DELSIG11 timer
DCB02	Used to generate a 1ms interrupt for timing functions
DCB03	SPI master user module to communicate with the radio
DBB10	Generates pseudo random numbers
DBB11	Modified PRS block. Generates serial RX interrupts while sleeping
DCB12	Serial RX function, used for user serial data and debug
DCB13	Serial TX funciton, used for user serial data and debug
ACB00	Reference mux to enable RefH to be driven out
ACB01	Reference mux, Connects Port0 pins to A/D
ACB03	Referenc mux, enables RefLow to be driven out
ASC21	Used for the A/D

Hub

The analog blocks in the hub are configured exactly like the Sensor

DBB01	Used to generate a 1ms interrupt for timing functions
DCB02	Serial transmitter for debug only output
DCB03	SPI master user module to communicate with the radio
DBB11	Used for the A/D
DCB12	Serial reciever for the host interface
DCB13	Serial transmitter for the host interface

7.7 Files/ Functions

7.7.1 Common files

7.7.1.1 config.h

Contains the build configuration options you're most likely to modify.

7.7.1.2 cydefine.h

Contains Cypress's defines for various data types.

7.7.1.3 cywusb693_.h

Contains all the #defines for the LS Radio an is based on the datasheet. Because it only contains #defines, it is strongly recommended that you use this file if nothing else.

7.7.1.4 debug.c & .h

Routines for serial output via HyperTerm.

```
U8 GetChar()
void OutChar(U8 data)
void OutDec(U16 Value)
void OutHex(U8 Data)
void OutHex16(U16 Data)
void OutNibble(U8 Nibble)
void OutRamStr(U8 *pStr)
void OutStr(const U8 *pStr)
void PrintBanner()
void PrintLine()
void SerialDebugInit()
```

7.7.1.5 hardware.c & .h

Routines to manipulate the PSoC and N:1 Node Board.

```
U8 ConvertDipSwitch(U8 DipVoltage)
void FramDump()
void FramInit(U8 Value)
U8 FramReadByte(U16 Addr)
void FramReadBytes(U16 Addr, U8* pData, U8 Count)
void FramWriteByte(U16 Addr, U8 Value)
void FramWriteBytes(U16 Addr, U8* pData, U8 Count)
void LedHeartBeat()
void HwClearFlash()
void HwInit()
void HwOnIdle()
U8 HwReadSwitchS1(void)
U8 ReadA2DInput(A2DINPUT Input)
```

7.7.1.6 nvram.c & .h

Routines to write the PSoC's flash.

```
NVR_STATUS NvramWrite(U16 NvrBlock, void *Data, U16 Length)
NVR_STATUS NvramRead(U16 NvrBlock, void *Data, U16 Length)
```

7.7.1.7 protocol.c & .h

Contains the routines that implement the N:1 protocol.

```
void ProtocolBind()
RX_STATUS ProtocolChannelSearch(U8 PacketSize)
U8 ProtocolComputeChecksum(U8 *pData, U8 Length, U8 Seed)
U8 ProtocolComputeCrc(U8 *Buffer, U8 Length, U8 Seed)
void ProtocolFindAnAvailableChannel()
void ProtocolInit()
void ProtocolOnIdle()
BOOL ProtocolProcessProtocolPacket()
RX_STATUS ProtocolProcessSeqn(void)
void ProtocolReleaseRxPacketToRadio()
RX_STATUS ProtocolCheckForPacketFromSensor()
void ProtocolSendAckPacket(U16 DeviceId)
void ProtocolSendDataPacket(U8 PacketLength)
void ProtocolSendPacket(U8 PacketLength)
void ProtocolServiceBind()
void ProtocolStartBind()
void ProtocolStopBind()
RX_STATUS ProtocolWaitForPacket()
```

7.7.1.8 radio.c & .h

All the routines to drive the radio.

```
void IsrGpio()
void IsrInit()
void RadioDumpRegisters()
void RadioGetMid(U32* Addr)
BOOL RadioGetRssi(U8* RssiReading)
void RadioInit()
BOOL RadioIrqAsserted()
void RadioOn(U8 RxOrTx)
void RadioRxIsr()
void RadioSetChannel(U8 Channel)
void RadioSetPnCode(U8 PnCodeIndex)
void RadioSleep()
void RadioTransmit(U8 Length, U8 *pData)
void RadioTxIsr()
void RadioWakeup()
```

7.7.1.9 spi.c & .h

Routines used to access the LR Radio via the SPI interface.

U8 SpiRadioAccess(U8 Address, U8 Data)
void SpiRadioBurstAccess(U8 Address, U8 *pData, U8 Length)
void SpiRadioOn()

7.7.1.10 timer.c & .h

Contains the ISR to increment the 1 millisecond variables. Also contains blocking delay routines.

void MsTimerTick()
void TimerDelayIterations(U16 NumIterations)
void TimerDelayMsec(U16 NumMilliseconds)

7.7.1.11 version.h

File that contains the Build and Release number

7.7.2 Hub Specific files

7.7.2.1 host.c & .h (Hub Only)

Routines to implement the COBS interface between the Host and the Hub

void DecodeCobsArray(U8 *pSource, U8 Length, U8 *pDestination)
void EncodeCobsArray(U8 *pSource, U8 Length, U8 *pDestination)
U16 EnumerateAllSensors()
void HostProcessPacketFromSensor(PACKET_TYPES PacketType)
void HostProcessSerialDataFromHost()
void HostSendNetworkStatusResponse()
void HostSerialRxIsr()
void HostSerialTxIsr()
void HostTransmittedBackChannelData()
void StartTxToHost()
U8 StoreOutboundMessage()

7.7.2.2 Hub.c & .h (Hub Only)

The code that drives the rest of the system.

U8 HubDeleteSensor(U16 DeviceId)
void HubFindNextFreeDeviceId()
void HubInit()
void HubProcessPacketFromSensor()
void ResetDeviceIdTable()
BOOL SendAckOrBackChannelData()

7.7.2.3 main.c (Hub Version)

Simple main loop that calls out to the hardware, protocol, Hub, and host.

7.7.3 Sensor specific files

7.7.3.1 main.c & h

7.7.3.2 sensor.c & h

7.7.3.3 multisim.c & .h

8. SOFTWARE ARCITECTURE

8.1 Overview

This document describes the software source code modules used to communicate with the N:1 Hub board in order to send and receive data and commands. It will not cover the details of the Microsoft Foundation Class (MFC) Library. Please refer to the Microsoft Visual C++ documentation for more on MFC.

8.2 Development Environment

The following tools are required to build and develop the N:1 Software application.

Software/Firmware:

Microsoft Visual Studio.NET 2003

N:1 Firmware V01.00.01

Hardware:

Node Board 121-17800, Revision *B or *C

Serial Board 121-18200**

9. SOFTWARE CODE CLASSES

The Nto1 SW consists of the following classes:

9.1 CNto1App Class

The CNto1App class performs provides member functions for initializing the application (and each instance of it) and for running the application.

Table 1: CNto1App Methods

Method	Type	Description
CNto1App()	Public	Performs the construction
InitInstance()	Public	Performs the basic initialization
OnAppAbout()	Public	Provides the information about software version and others

9.2 CMainFrame Class

The CMainFrame class is the Visual C++ generated file that is a derived frame-window class for the application's main frame window. Other methods have been added to send, receive and process the command or data from/to the Hub.

Table 2: CMainFrame Methods

Method	Type	Description
CMainFrame()	protected	Performs the construction
OnCreate()	protected	Creates the mainframe, toolbar, status bar, …
OnClose()	protected	Handles the application closing
OnFileCaptureLog()	protected	Starts to capture the log message to file
OnFileStopLogCapture()	protected	Stops the log message capture
OnUpdateHubInfoIndicator()	protected	Updates the Hub information in the status bar
OnUpdateConnectIndicator()	protected	Updates the connection state between Hub and PC in the status bar
OnSerialEvent()	protected	Handles the serial event
OnCobsPacketsEvent()	protected	Handles the cobs packets event
SetMenuItem()	protected	Enables/disables the menu item
EnableMenuItem()	protected	Enables/disables the menu item and toolbar item
CheckToolBarButton()	protected	Check/uncheck the toolbar item
CheckMenuItem()	protected	Check/uncheck the menu item
WindowProc()	protected	Provides a Windows procedure for a CWnd object
DoHubInit()	protected	Initializes the Hub It includes Hub Reset, Get Hub Info, Get Network Info and Enumerate Devices.

Method	Type	Description
LogStatusMessage()	protected	Displays the status message in message pane.
SetCommandField()	protected	Update the command files in the Sensor Status Table with command response.
GetHubInfoCommand()	protected	Sends Get Hub Info command to the Hub
DeleteNodeCommand()	protected	Sends Delete Node command to the Hub
EnumDevicesCommand()	protected	Sends Enumerate Devices command to the Hub
NetworkConfigCommand()	protected	Sends Network Configuration command to the Hub
NetworkInfoCommand()	protected	Sends Network Status command to the Hub
ResetCommand()	protected	Sends Reset command to the Hub
ChangeChannelCommand()	protected	Sends Change Channel command to Hub
HubInfoResponse()	protected	Handel the Hub to host Hub Info Response
BindResponse()	protected	Handles the Hub to host Bind Response
BindInfoResponse()	protected	Handles the Hub to host Bind Information Response
DeleteNodeResponse()	protected	Handles the Hub to host Delete Node Response
SendMsgResponse()	protected	Handles the Hub to host Send Message Response
IncomingMsgResponse()	protected	Handles the Hub to host Incoming Message
EnumDevicesResponse()	protected	Handles the Hub to host Enumerate Devices Response
NetworkConfigureResponse()	protected	Handles the Hub to host Network Configure Response
NetworkInfoResponse()	protected	Handles the Hub to Network Status Response
ChangeChannelResponse()	protected	Handles the Hub to host Change Channel Response
ResetResponse()	protected	Handles the Hub to host Reset Response
StartTimer()	protected	Starts the timer
StopTimer()	protected	Stops the timer
GetSensorName()	protected	Gets sensor name from the sensor ID
GetSensorIDName()	protected	Gets Sensor ID & Sensor Name from the sensor ID
ClearGraphs()	protected	Clears all the graphs in graph pane
NodeIsChecked()	protected	Checks if this node is checked
ChangeTemp()	protected	Changes the temperature scale between Fahrenheit and Celsius.
CheckDuplicatedSensor()	protected	Checks and removes the duplicated and inactive sensors in the sensor table
SetSummaryFileTimer()	public	Sets the timer for the summary file
ConvertToFahrenheit()	public	Converts temperature from Celsius to Fahrenheit
GetCelsius()	public	Gets the temperature degree in Celsius from the read number
RecordToCsvFile()	public	Captures the sensor data to CSV file

Method	Type	Description
UpdateNetworkEfficiency()	public	Calculates the network efficiency and updatess the network efficiency indicator in the status bar
CountIncomingMessage()	public	Counts the incoming message number to calculate the network efficiency
DeleteNodeFromGraph()	public	Deletes the node from the graph
GetBatteryRealVolt()	public	Gets battery real voltage from the read number
GetPotRealVolt()	public	Gets poten real voltage from the read number
LogMessage()	public	Displays the raw data in the message area
ReceivedHubPacket()	public	Processes the received Hub packets
SendHubPacket()	public	Sends the packets to Hub
SendMsgCommand()	public	Formats all the sensor control to a message and sends to the Hub
GetTemperature()	public	Get the real temperature degree from the read number
UpdateAvailablePort()	public	Updates the available serial port to the serial port array
TranslatePNCodeFromIndexToID()	public	Translates the PN Code Index to PN Code ID
TranslatePNCodeFromIDToIndex()	public	Translates the PN Code ID to PN Code Index
ValidatePNCodeID()	public	Validates the PN Code ID. (Some ID don't exist in the N:1 protocol)
UpdateDuplicatedSensorFor-Increment()	public	Updates the duplicated sensor array and defers the process for deleting the node when doing the enumeration
UpdateDuplicatedSensorFor-Decrement()	public	Updates the duplicated sensor array when deleting the node
OnLButtonDownRowList()	public	Handler for mouse click in the sensor status table
OnBnClickedSendMessage()	public	Handler for clicking the Send Message button. It will organize the message of LED, sensor interval and payload and send to Hub
OnBnClickedRename()	public	Handler for clicking the Rename button. It will change the sensor name to the one in the edit box
OnTimer()	public	The framework calls this member function after each interval specified in the SetTimer member function used to install a timer
OnSummaryDialog()	public	Handler for clicking the Control Dialog command in the View menu. It invokes the Control dialog
OnTempGui()	public	Handler for clicking the Fahrenheit/Celsius button in the toolbar. It toggles data capture units between Fahrenheit and Celsius

Method	Type	Description
OnFileCaptureSensorData()	public	Handler for clicking record sensor data command in File menu. It captures the sensor data to CSV file
OnFileStopSensorDataCapture()	public	Handler for clicking stop recording command in File menu. It stops the sensor data capture
OnPreferencesSerialPort()	public	Handler for clicking Preferences->Serial Port command in File menu. It sets the serial port number and baud rate
OnPreferencesDefault-SensorInterval()	public	Handler for clicking Preferences->Default Sensor Interval command in File menu. It sets the Default Sensor Interval
OnPreferencesClearSensor-ContentsInRegistry()	public	Handler for clicking Preferences->Clear Sensor Contents in Registry command in File menu. It clears the Sensor contents such as sensor name and sensor interval in the registry
OnEditClearmessages()	public	Handler for clicking the Clear Messages command in Edit menu. It clears the message pane
OnEditCleargraph()	public	Handler for clicking the Clear Graphs command in Edit menu. It clears the temperature graph and potentiometer graph in the graph pane
OnEditClearSensorTableData()	public	Handler for clicking the Clear Sensor Table Data command in Edit menu. It clears the Sensor Table data.
OnHubConnectToHub()	public	Handler for clicking Connect to Hub command in Hub menu. It connects the host to the Hub
OnHubDisconnectFromHub()	public	Handler for clicking Disconnect from Hub command in Hub menu. It disconnects host from Hub
OnHubGetInfo()	public	Handler for clicking Get Hub Info command in Hub menu. It sends Get Hub Info command to the Hub
OnHubNetworkInfo()	public	Handler for clicking Get Network Info command in Hub menu. It sends Network Status command to the Hub
OnHubConfigurenetwork()	public	Handler for clicking Configure Network command in Hub menu. It sends Network Configuration command to the Hub
OnHubEnumerateDevices()	public	Handler for clicking Enumerate Device command in Hub menu. It sends Enumerate Devices command to the Hub
OnHubDeleteSelectedSensors()	public	Handler for clicking Delete Selected Sensor command in Hub menu. It sends Deletes Node command to the Hub
OnHubDeleteAllSensors()	public	Handler for clicking Delete All Sensors command in Hub menu. It deletes all the sensors
OnHubBind()	public	Handler for clicking Bind command in Hub menu. It toggles the bind state

Method	Type	Description
OnHubSeededbind()	public	Handler for clicking Seeded Bind command in Hub menu. It toggles the seeded bind state
OnHubChangechannel()	public	Handler for clicking Change Channel command in Hub menu. It sends Change Channel command to the Hub
OnHubReset()	public	Handler for clicking Reset Hub command in Hub menu. It sends hub Reset command to the Hub
OnGraphsTempature()	public	Handler for clicking Temperature command in the Graphs menu. It displays the temperature graph in the graph pane
OnGraphsPotentiometer()	public	Handler for clicking Potentiometer command in the Graphs menu. It displays the potentiometer graph in the graph pane
OnGraphsTemperaturesetting()	public	Handler for clicking Temperature Graph Setting command in the Graphs menu. It changes the temperature setting in the temperature graph
OnGraphsXaxisSetting()	public	Handler for clicking X Axis Setting command in the Graphs menu. It changes the X-Axis setting in the graph
OnViewHubinfo()	public	Handler for clicking Hub Info command in the View menu. It displays the Hub Information
OnViewNetworkinfo()	public	Handler for clicking Network Info command in the View menu. It displays the Network information

9.3 CDetailView Class

The CDetailView class is used to display the sensor properties and create message.

Table 3: CDetailView Methods

Method	Type	Description
CDetailView()	protected	Performs the construction
DoDataExchange()	protected	Exchanges and validates data
OnInitialUpdate()	public	Called by the framework after the view is first attached to the document, but before the view is initially displayed
OnCreate()	public	The framework calls this member function when an application requests that the Windows window be created
OnEnChangePayloadDataLength()	public	Handler for changing Payload Data Length
OnBnClickedInputAscii()	public	Handler for checking Input ASCII
OnBnClickedSendMessage()	public	Handler for clicking the Send Message button

Method	Type	Description
OnBnClickedRename()	public	Handler for clicking Rename button

9.4 CRowListView Class

The CRowListView class is used to generate and manage a table to containing low-level sensor status information for each sensor bound to the Hub.

Table 4: CRowListView Methods

Method	Type	Description
CRowListView()	protected	Performs the construction
PreCreateWindow()	public	It is called by the framework before the creation of the window
OnInitialUpdate()	public	Called by the framework after the view is first attached to the document, but before the view is initially displayed
OnLButtonDown()	protected	The framework calls this member function when the user presses the left mouse button
OnRButtonDown()	protected	The framework calls this member function when the user presses the right mouse button
OnLButtonDblClk()	protected	The framework calls this member function when the user double-clicks the left mouse button
OnKeyDown()	protected	The framework calls this member function when the user presses the keyboard
GetNodeIDByItemIndex()	protected	Gets the sensor ID from the item index
GetNodeItemAsText()	protected	Gets the node item value as text
GetNodeItemAsInt()	protected	Gets the node item value as integer
SetNodeItemAsText()	protected	Sets the node item value as text
SetNodeItemAsInt()	protected	Sets the node item value as integer
GetItemIndexByNode()	public	Gets the Item index from the node ID
AddNodeID()	public	Adds a node to the sensor status table
DeleteNodeID()	public	Deletes the node from the sensor status table
GetNumberOfNodeIDs()	public	Gets total number of nodes
GetFirstNodeID()	public	Gets the first node ID in the sensor status table
GetNextNodeID()	public	Gets the next node ID in the sensor status table
GetNumberOfSelectedNodeIDs()	public	Gets the total number of selected node
GetCurrentSelectedNodeID()	public	Gets the node ID of the currently selected node
GetFirstSelectedNodeID()	public	Gets the node ID of the first selected node
GetNextSelectedNodeID()	public	Gets the node ID of the next selected node
GetSensorName()	public	Gets the sensor name from the node ID
GetMID()	public	Gets the MID of the node
GetBattery()	public	Gets the battery voltage of the node

Method	Type	Description
GetPot()	public	Gets the potentiometer voltage of the node
GetTemp()	public	Gets the temperature of the node
GetYellowLED()	public	Gets the yellow LED state of the node
GetRedLED()	public	Gets the red LED state of the node
GetButton1()	public	Gets the button1 state of the node
GetButton2()	public	Gets the button2 state of the node
GetDIP1()	public	Gets the DIP1 state of the node
GetDIP2()	public	Gets the DIP2 state of the node
GetDIP3()	public	Gets the DIP3 state of the node
GetDIP4()	public	Gets the DIP4 state of the node
GetReceivedMessage()	public	Gets the received message of the node
GetTimeStamp()	public	Gets the time stamp of the node
SetSensorName()	public	Sets the sensor name of the node ID
SetMID()	public	Sets the MID of the node
SetBattery()	public	Sets the battery voltage of the node
SetPot()	public	Sets the potentiometer voltage of the node
SetTemp()	public	Sets the temperature of the node
SetYellowLED()	public	Sets the yellow LED status of the node
SetRedLED()	public	Sets the red LED status of the node
SetButton1()	public	Sets the button1 status of the node
SetButton2()	public	Sets the button2 status of the node
SetDIP1()	public	Sets the DIP1 status of the node
SetDIP2()	public	Sets the DIP2 status of the node
SetDIP3()	public	Sets the DIP3 status of the node
SetDIP4()	public	Sets the DIP4 status of the node
SetReceivedMessage()	public	Sets the received message of the node
SetTimeStamp()	public	Sets the time stamp of the node
SetCommand()	public	Sets the command status of the node
NodeIsChecked()	public	Verifies if the node is checked
SetLowBattery()	public	Changes to red color if the battery voltage is lower than 3.1v
CheckItem()	public	Handles the check/uncheck item to enable/disable the graph for the selected sensor

9.5 CMessageView Class

The CMessageView class is used to display the raw packet data received/sent from/to Sensors.

Table 5: CMessageView Methods

Method	Type	Description
CMessageView()	protected	Performs the construction
OnDraw()	public	It is called by the framework to render an image of the document.
UpdateVScroll()	public	Updates the vertical scroll bar
UpdateHScroll()	public	Update the horizontal scroll bar
Paint()	public	Paints message
CountLines()	public	Gets number of lines

Method	Type	Description
AddLine()	public	Adds line of message
ClearBuffer()	public	Clear all the message
OnPaint()	public	It is called by framework when Windows or an application makes a request to repaint a portion of an application's window.
OnCreate()	public	The framework calls this member function when an application requests that the window be created
OnVScroll()	public	The framework calls this member function when the user clicks the window's vertical scroll bar
OnHScroll()	public	The framework calls this member function when the user clicks the window's horizontal scroll bar
OnMouseWheel()	public	The framework calls this member function as a user rotates the mouse wheel and encounters the wheel's next notch
RenderText()	protected	Prints the text
OnUpdate()	protected	It is called by the framework after the view's document has been modified

9.6 CGraphView Class

The CGraphView class is used to graphically display the temperature and potentiometer data received from each of the sensors over time.

Table 6: CGraphView Methods

Method	Type	Description
CGraphView()	protected	Performs the construction
OnDraw()	public	It is called by the framework to render an image of the document.
UpdateGraphTimer()	protected	Updates on the graph timer
AddGraphData()	protected	Draws the graph
AddTempData()	public	Draws the temperature data
AddPotData()	public	Draws the potentiometer data
OnCreate()	public	The framework calls this member function when an application requests that the Windows window be created
OnLButtonDblClk()	public	The framework calls this member function when the user double-clicks the left mouse button
OnTimer()	public	The framework calls this member function after each interval specified in the SetTimer member function used to install a timer
OnDestroy()	public	The framework calls this member function to inform the CWnd object that it is being destroyed

9.7 CGraphSeries Class

The CGraphSeries is used to manage the graph data for one moment but different sensors.

Table 7: CGraphSeries Methods

Method	Type	Description
CGraphSeries()	public	Performs the construction
SetLabel()	public	Set the time label
GetLabel()	public	Get the time label
SetData()	public	Set the graph data
GetData()	public	Get the graph data
DeleteData()	public	Delete the graph data
GetMaxDataValue()	protected	Gets the largest data value in this series
GetNonZeroElementCount()	protected	Gets the number of data points that are not zero
GetDataTotal()	protected	Gets the sum of the data points for this series

9.8 CGraph Class

The CGraph class is used to draw the graph, X-Axis, Y-Axis, label and legend.

Table 8: CGraph Methods

Method	Type	Description
CGraph()	public	Performs the construction
SetXAxisLabel()	public	Sets the label for X-Axis
SetYAxisLabel()	public	Sets the label for Y-Axis
AppendGroup()	public	Appends a new sensor to the array
AddLegend()	public	Adds the value to the legend and assign a color and a name to this legend
RenameLegend()	public	Rename a legend
SetGraphType()	public	Sets graph type. This application only uses the line graph
SetGraphTitle()	public	Sets the title of the graph
LookupLabel()	public	Gets the group index for the given label
DrawGraph()	public	Draw the graph
GetGroup()	public	Get the group index based on the sensor name
RemoveGroup()	public	Remove the group (device)
AddSeries()	public	Add a graph series for a moment
GetSeries()	public	Get series based on the time label
GetSeriesSize()	public	Get the graph series size
RemoveSeries()	public	Remove this series
RemoveAllSeries()	public	Remove all the series
RemoveAllLegend()	public	Remove all legend
ShiftYAxis()	public	Shifts the Y-Axis
SetMaxDataValueAllowed()	public	Set the maximum allowable data value
SetMinDataValueAllowed()	public	Set the minimum allowable data value

Method	Type	Description
DrawTitle()	protected	Draws graph title; size is proportionate to width
SetupAxes()	protected	Sets the axes and origin values
DrawAxes()	protected	Draws the axis
DrawLegend()	protected	Draws the legend
DrawSeriesBar()	protected	Draws the bar graph
DrawSeriesLine()	protected	Draws the line graph
DrawSeriesPie()	protected	Draws the pie graph
GetMaxLegendLabelLength()	protected	Calculates the current max legend label length in pixels
GetMaxSeriesSize()	protected	Gets the largest number of data points in any series
GetMaxNonZeroSeriesSize()	protected	Gets the largest number of non-zero data points in any series
GetMaxDataValue()	protected	Gets the largest data value in all series
GetNonZeroSeriesCount()	protected	Gets the number about how many series are populated
WedgeEndFromDegrees()	protected	Converts degrees to x and y coords
SpinTheMessageLoop()	protected	Spins The Message Loop
RGBtoHLS()	protected	Converts color space from RGB to HLS
HLStoRGB()	protected	Converts color space from HLS to RGB
HueToRGB()	protected	Utility routine for HLS to RGB

9.9 CCobsPackets Class

The CCobsPackets class is used to encode the sending data to Cobs packet and decode the receiving data from Cobs packet. Detail information about the COBS please see “Consistent Overhead Byte Stuffing (COBS)” in IEEE April 1999 Transactions onf Networking Paper.

Table 9: CCobsPackets Methods

Method	Type	Description
CCobsPackets()	public	Performs the construction
Init()	public	Initialization
SendEvent()	public	Post the event
DataReceived()	public	Processes the received data
SendPacket()	public	Sends the packet for encoding to Cobs packet and put to the queue
GetPacket()	public	Gets the packet for both sending and receiving packet
StuffData()	protected	Encode the data to Cobs packet
UnStuffData()	protected	Decode the data from Cobs packet
NewPacket()	protected	Creates new packet

9.10 CSerial Class

The CSerial class is used for serial communications.

Table 10: CSerial Methods

Method	Type	Description
CSerial()	public	Performs the construction
CheckPort()	public	Check if particular COM-port is available
Open()	public	Open the serial communications for a particular COM port. You need to use the full devicename (i.e. "COM1") to open the port. It's possible to specify the size of the input/output queues
Close()	public	Close the serial port
Setup()	public	Setup the communication settings such as baud rate, data bits, and parity and stop bits. The default settings are applied when the device has been opened. Call this function if these settings do not apply for your application. If you prefer to use integers instead of the enumerated types then just cast the integer to the required type
SetEventChar()	public	Set/clear the event character. When this byte is being received on the serial port then the EEventRcvEv event is signaled, when the mask has been set appropriately. If the fAdjustMask flag has been set, then the event mask is automatically adjusted.
SetMask()	public	Set the event mask, which indicates what events should be monitored. The WaitEvent method can only monitor events that have been enabled. The default setting only monitors the error events and data events. An application may choose to monitor CTS. DSR, RLSD, etc as well.
WaitEvent()	public	The WaitEvent method waits for one of the events that are enabled (see SetMask).

Method	Type	Description
SetupHandshaking()	public	Setup the handshaking protocol. There are three forms of handshaking: 1) No handshaking, so data is always send even if the receiver cannot handle the data anymore. This can lead to data loss, when the sender is able to transmit data faster then the receiver can handle. 2) Hardware handshaking, where the RTS/CTS lines are used to indicate if data can be sent. This mode requires that both ports and the cable support hardware handshaking. Hardware handshaking is the most reliable and efficient form of handshaking available, but is hardware dependant. 3) Software handshaking, where the XON/XOFF characters are used to throttle the data. A major drawback of this method is that these characters cannot be used for data anymore.
SetupReadTimeouts()	public	Read operations can be blocking or non-blocking. You can use this method to setup whether to use blocking or non-blocking reads. Non-blocking reads is the default, which is required for most applications. 1) Blocking reads. It will cause the 'Read' method to block until the requested number of bytes has been read. This is useful if you know how many data you will receive 2) Non-blocking reads. It will read as many bytes into your buffer and returns almost immediately. This is often the Preferred setting.
Write()	public	Write data to the serial port. Note that we are only able to send ANSI strings, because it probably doesn't make sense to transmit Unicode strings to an application.
Read()	public	Read data from the serial port. Refer to the description of the 'SetupReadTimeouts' for an explanation about (non) blocking reads and how to use this.
Break()	public	Send a break
GetEventType()	public	Determine what caused the event to trigger

Method	Type	Description
GetError()	public	Obtain the COMM and event handle
IsOpen()	public	Check if com-port is opened
GetLastError()	public	Obtain last error status
GetCTS()	public	Obtain CTS settings
GetDSR()	public	Obtain DSR settings
GetRing()	public	Obtain RING settings
GetRLSD()	public	Obtain RLSD settings
Purge()	public	Purge all buffers
CheckRequirements()	protected	Check the requirements
CancelCommIo()	protected	Cancel IO wrapper

9.11 CSerialEx Class

The CSerialEx class is derived from the CSerial class.

Table 11: CSerialEx Methods

Method	Type	Description
CSerialEx()	public	Performs the construction
Open()	public	Open the serial communications for a particular COM port. You need to use the full device name (i.e. “COM1”) to open the port
Close()	public	Close the serial port
StartListener()	public	Start the listener thread
StopListener()	public	Stop the listener thread. Because the other thread might be busy processing data it might take a while, so you can specify a time-out
ThreadProc()	protected	Each opened COM-port uses its own specific thread, which will wait for one of the events to happen. When an event happens, then the client window is send a message informing about the event
OnEvent()	protected	Event handler

9.12 CSerialWnd Class

The CSerialWnd class is derived from the CSerialEx class.

Table 12: CSerialWnd Methods

Method	Type	Description
CSerialWnd()	public	Performs the construction

Method	Type	Description
Open()	public	Open the serial communications for a particular COM port. You need to use the full device name (i.e. “COM1”) to open the port
Close()	public	Close the serial port
OnEvent()	protected	Event handler

9.13 CAboutDlg Class

CAboutDlg class is used to provide the version information.

9.14 CActuatorOjbect Class

CActuatorOjbect class is used to save the Actuator parameters into the registry.

9.15 CAppSetting Class

CAppSetting class is used to save the temperature scale (Fahrenheit or Celsius) into the registry.

9.16 CAvgFuncRegistry Class

CAvgFuncRegistry class is used to save the average function parameters into the registry.

9.17 CCalibrationDlg Class

CCalibrationDlg class is used to select the sensors to be calibrated.

9.18 CChangeDefaultSensorIntervalDlg Class

CChangeDefaultSensorIntervalDlg class is used to input the new Default Sensor Interval value.

9.19 CDefaultSensorIntervalSetting Class

CDefaultSensorIntervalSetting class is used to save the Default Sensor Interval into the registry.

9.20 CDeltaDlg Class

CDeltaDlg class is used to select the delta calculation parameters.

9.21 CGraphSetting Class

CGraphSetting class is used to save the graph setting into the registry.

9.22 CHexEditBase Class

CHexEditBase is a Hex-Edit-Control based on the CWnd class. It implements basic behavior to edit/view data in hexadecimal view (binary).

9.23 CHubConfigDlg Class

CHubConfigDlg class is used to configure the network parameters such as channel number, PN code and Hub MID.

9.24 CHubInfoDlg Class

CHubInfoDlg class is used to display the Hub information such as firmware version, radio version, MID and maximum device ID.

9.25 CListViewEx Class

CListViewEx class is used to manage the list view table. In this application it is used to manage the sensor status table.

9.26 CMainWndPlacement Class

CMainWndPlacement class is used to retain prior window position and size.

9.27 CNetworkEfficiency Class

CNetworkEfficiency class is used to calculate the network efficiency. It is also used to keep the sensor interval.

9.28 CNetworkInfoDlg Class

CNetworkInfoDlg class is used to display the Network information such as channel, PN Code ID, data rate and bind state.

9.29 CProgressBar Class

CProgressBar class is used to display a progress bar in the status bar. In this application it is used to display the Network Efficiency.

9.30 CRegistry Class

CRegistry class is used to create or delete the subkey in the registry. It also can be used to manage the value in the registry.

9.31 CRegSettings Class

CRegSettings class is used to save the application parameter into the registry.

9.32 CSelectDlg Class

CSelectDlg class is used to select the sensor for average and modify the average name.

9.33 CSensorName Class

CSensorName class is used to save the sensor name and sensor interval into the registry.

9.34 CSerPortDlg Class

CSerPortDlg class is used to input the serial port number and baud rate.

9.35 CSerPortSetting Class

CSerPortSetting class is used to save the serial port setting into the registry.

9.36 CSubclassWnd Class

CSubclassWnd class is Generic class to hook messages on behalf of a CWnd.

Once hooked, all messages go to CSubclassWnd::WindowProc before going to the window. Specific subclasses can trap messages and do something.

9.37 CSubclassWndMap Class

The message hook map is derived from CMapPtrToPtr, which associates a pointer with another pointer. It maps an HWND to a CSubclassWnd, like the way MFC's internal maps map HWND's to CWnd's. The first CSubclassWnd attached to a window is stored in the map; all other CSubclassWnd's for that window are then chained via CSubclassWnd::m_pNext.

9.38 CSummeryl Class

CSummeryl class is used to calculate the average and delta.

9.39 CSummarySettings Class

CSummarySettings class is used to save the summary application parameter into the registry.

9.40 CTemperatureGraphSettingDlg Class

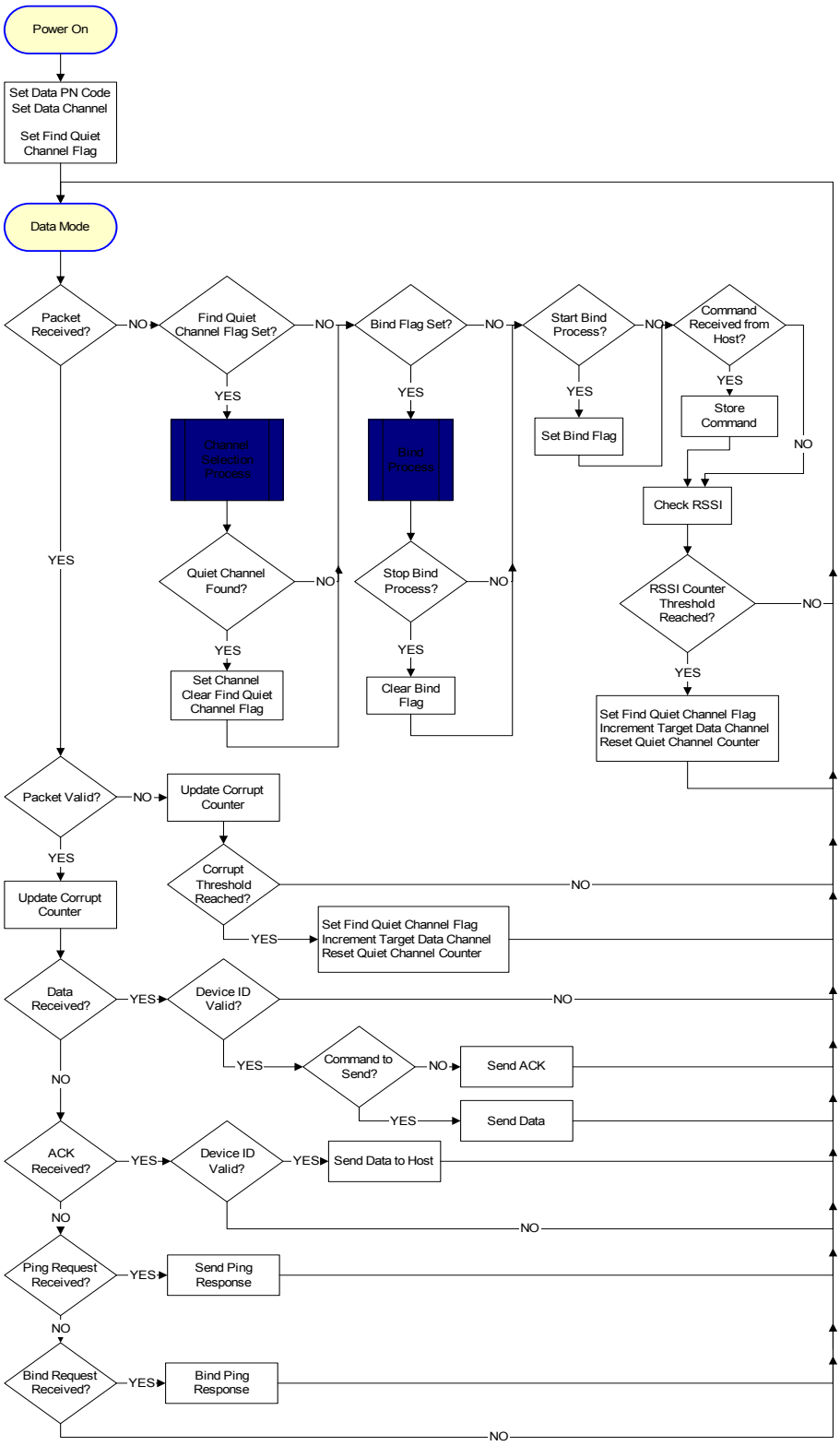
CTemperatureGraphSettingDlg class is used to input the temperature graph bound.

9.41 CXAxisSettingDlg Class

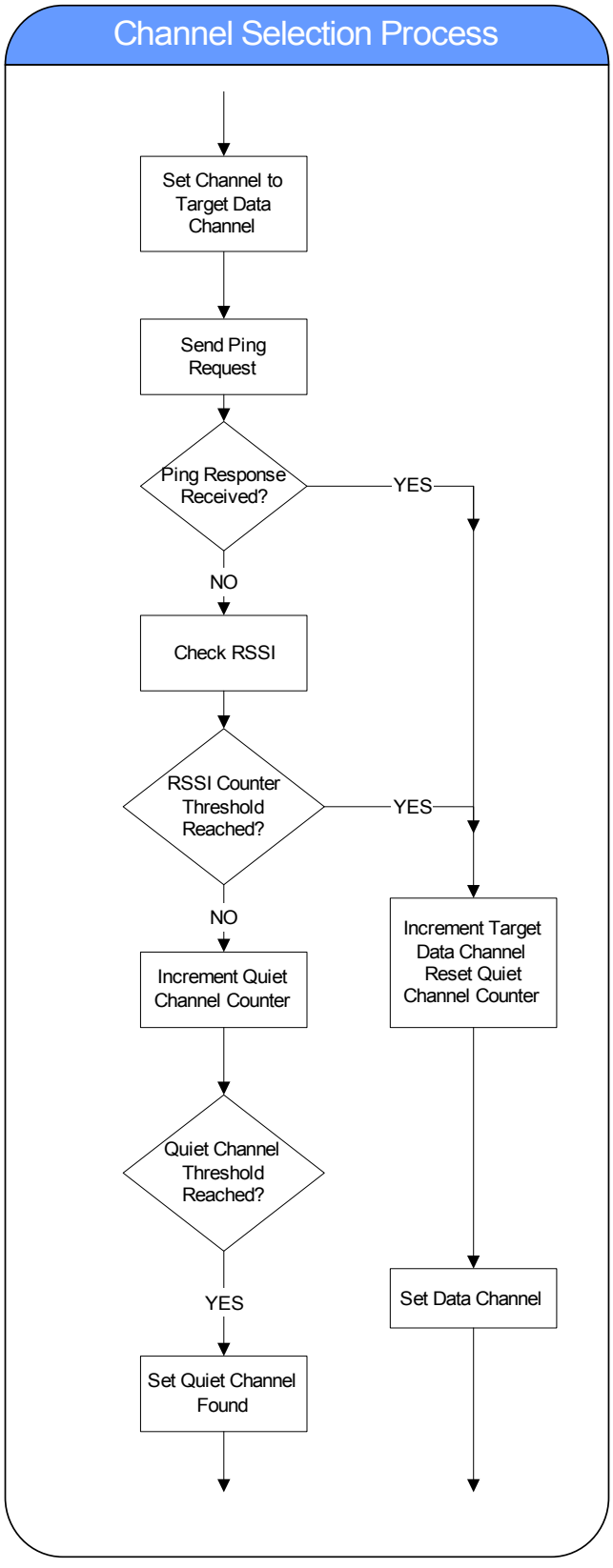
CXAxisSettingDlg class is used to input the X-Axis scale.

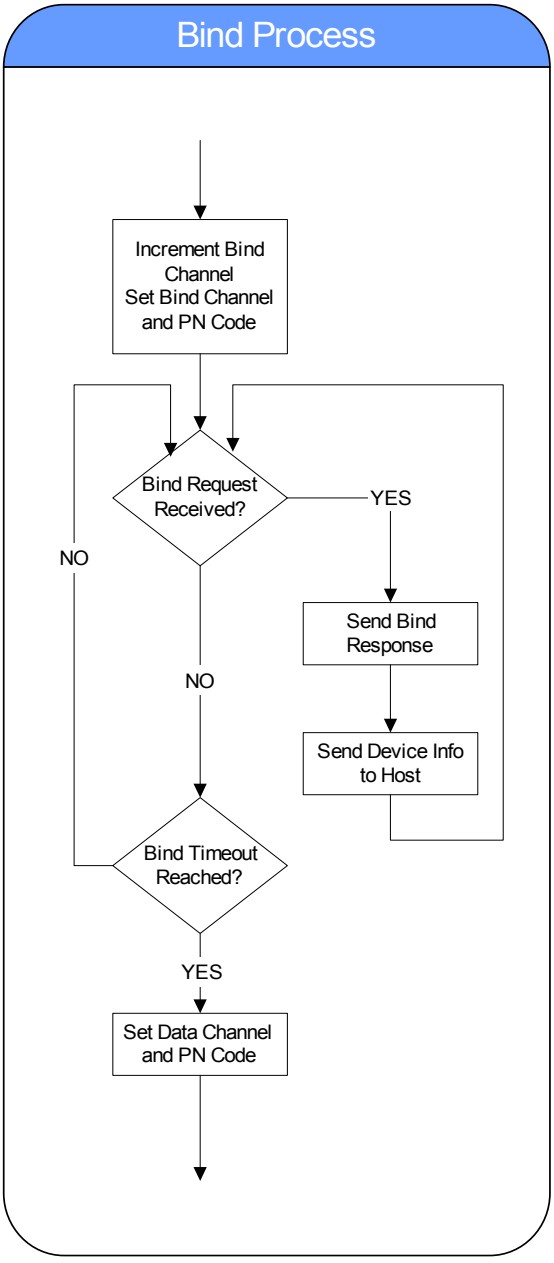
10. APPENDIX

10.1 Hub State Machine

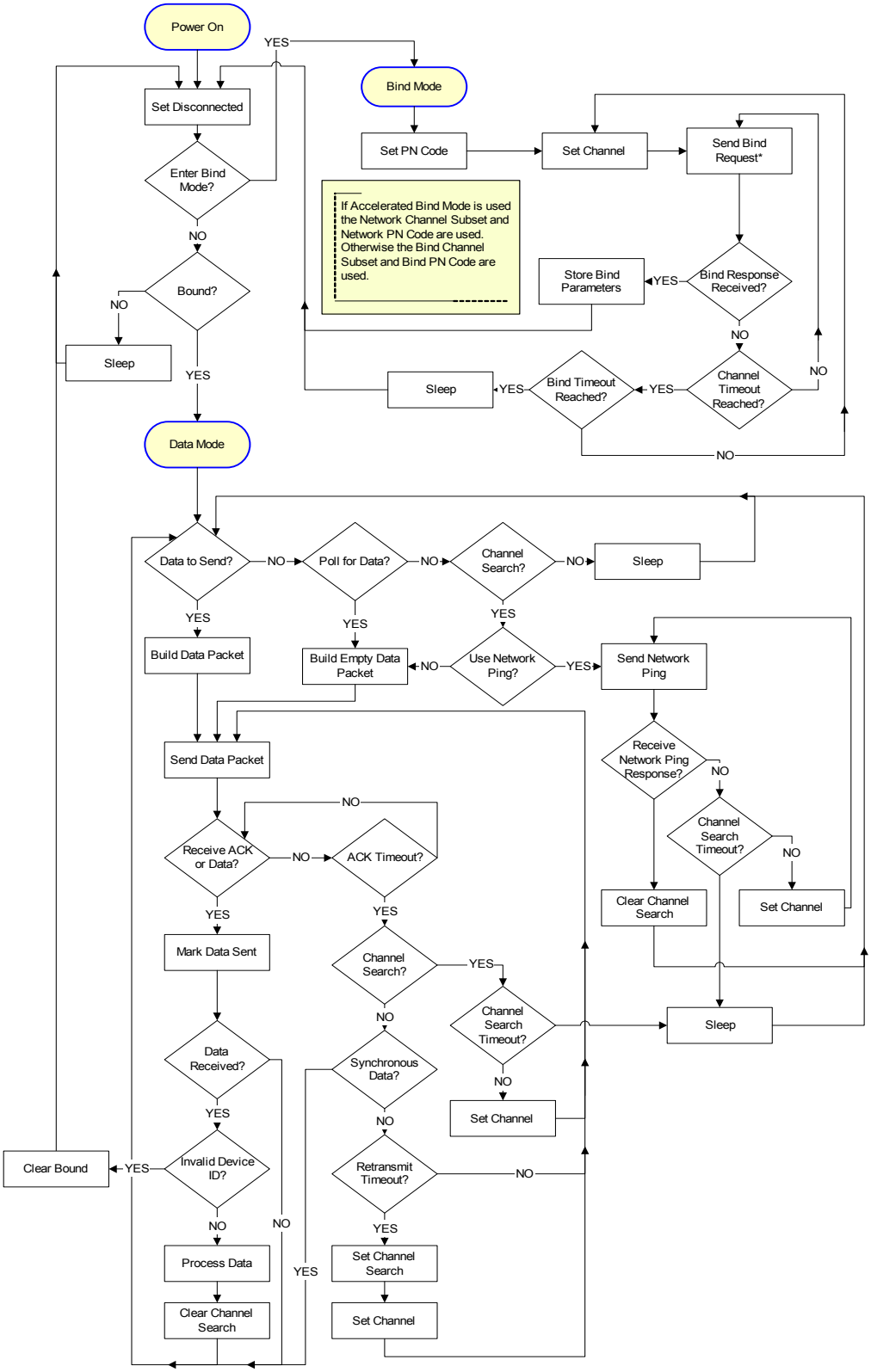


10.2 Channel selection and Bind process





Sensor State Machine



10.3 Channel Subset Table

Shaded values are reserved for binding.

The equation for deriving the next channel in the sequence is:

(Current channel + NUM_OF_SUBSETS) mod NUM_OF_CH_USED

Reserved for binding

Subset Index

Channel Configuration	1	2
Total Channels Used	78	77
Channels/subset	13	11
Channel spacing	6	7

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
60	61	62	63	64	65
66	67	68	69	70	71
72	73	74	75	76	77
0	1	2	3	4	5

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48
49	50	51	52	53	54	55
56	57	58	59	60	61	62
63	64	65	66	67	68	69
70	71	72	73	74	75	76
0	1	2	3	4	5	6

Channel Configuration	3
Total Channels Used	72
Channels/subset	9
Channel spacing	8

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
0	1	2	3	4	5	6	7

Channel Configuration	4
Total Channels Used	72
Channels/subset	8
Channel spacing	9

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
0	1	2	3	4	5	6	7	8

Channel Configuration	5
Total Channels Used	70
Channels/subset	7
Channel spacing	10

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
0	1	2	3	4	5	6	7	8	9

Channel Configuration	6
Total Channels Used	77
Channels/subset	7
Channel spacing	11

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76
0	1	2	3	4	5	6	7	8	9	10

Channel Configuration	7
Total Channels Used	72
Channels/subset	6
Channel spacing	12

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
0	1	2	3	4	5	6	7	8	9	10	11

Channel Configuration	8
Total Channels Used	78
Channels/subset	6
Channel spacing	13

0	1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38
39	40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77
0	1	2	3	4	5	6	7	8	9	10	11	12

10.4 16kbps PN Codes

```
    0x83, 0xF7, 0xA8, 0x2D, 0x7A, 0x44, 0x64, 0xD3, // PN Code ID  0 - Index  0
    0x3F, 0x2C, 0x4E, 0xAA, 0x71, 0x48, 0x7A, 0xC9, // PN Code ID  1 - Index  1
    0x17, 0xFF, 0x9E, 0x21, 0x36, 0x90, 0xC7, 0x82, // PN Code ID  2 - Index  2
//  0xA6, 0x46, 0xB5, 0x9A, 0x3A, 0x30, 0xB6, 0xAD, // PN Code ID  3 - Not Used
    0xBC, 0x5D, 0x9A, 0x5B, 0xEE, 0x7F, 0x42, 0xEB, // PN Code ID  4 - Index  3
    0x24, 0xF5, 0xDD, 0xF8, 0x7A, 0x77, 0x74, 0xE7, // PN Code ID  5 - Index  4
    0x3D, 0x70, 0x7C, 0x94, 0xDC, 0x84, 0xAD, 0x95, // PN Code ID  6 - Index  5
    0x1E, 0x6A, 0xF0, 0x37, 0x52, 0x7B, 0x11, 0xD4, // PN Code ID  7 - Index  6
    0x62, 0xF5, 0x2B, 0xAA, 0xFC, 0x33, 0xBF, 0xAF, // PN Code ID  8 - Index  7
    0x40, 0x56, 0x32, 0xD9, 0x0F, 0xD9, 0x5D, 0x97, // PN Code ID  9 - Index  8
    0x8E, 0x4A, 0xD0, 0xA9, 0xA7, 0xFF, 0x20, 0xCA, // PN Code ID 10 - Index  9
//  0x38, 0xB3, 0x31, 0xAB, 0x24, 0x78, 0xA6, 0xBD, // PN Code ID 11 - Not Used
    0x4C, 0x97, 0x9D, 0xBF, 0xB8, 0x3D, 0xB5, 0xBE, // PN Code ID 12 - Index 10
    0x0C, 0x5D, 0x24, 0x30, 0x9F, 0xCA, 0x6D, 0xBD, // PN Code ID 13 - Index 11
//  0x29, 0x29, 0x7B, 0xB5, 0xC8, 0xF4, 0x4D, 0x8A, // PN Code ID 14 - Not Used
    0x50, 0x14, 0x33, 0xDE, 0xF1, 0x78, 0x95, 0xAD, // PN Code ID 15 - Index 12
//  0xD6, 0xA6, 0xAA, 0x10, 0x96, 0xB3, 0xA6, 0xA7, // PN Code ID 16 - Not Used
//  0xF6, 0xCE, 0x0D, 0x12, 0xE3, 0x66, 0xA6, 0x94, // PN Code ID 17 - Not Used
    0x0C, 0x3C, 0xFA, 0xF9, 0xF0, 0xF2, 0x10, 0xC9, // PN Code ID 18 - Index 13
    0xF4, 0xDA, 0x06, 0xDB, 0xBF, 0x4E, 0x6F, 0xB3, // PN Code ID 19 - Index 14
//  0x93, 0x9C, 0x4E, 0x14, 0x1A, 0x39, 0xCF, 0xE6, // PN Code ID 20 - Not Used
//  0xC9, 0x2C, 0x06, 0x93, 0x86, 0xB9, 0x9E, 0xD7, // PN Code ID 21 - Not Used
    0x9E, 0x08, 0xD1, 0xAE, 0x59, 0x5E, 0xE8, 0xF0, // PN Code ID 22 - Index 15
    0xC0, 0x90, 0x8F, 0xBB, 0x7C, 0x8E, 0x2B, 0x8E, // PN Code ID 23 - Index 16
    0x80, 0x69, 0x26, 0x80, 0x08, 0xF8, 0x49, 0xE7, // PN Code ID 24 - Index 17
    0x7d, 0x2d, 0x49, 0x54, 0xd0, 0x80, 0x40, 0xC1, // PN Code ID 25 - Index 18
    0xB6, 0xF2, 0xE6, 0x1B, 0x80, 0x5A, 0x36, 0xB4, // PN Code ID 26 - Index 19
    0x42, 0xAE, 0x9C, 0x1C, 0xDA, 0x67, 0x05, 0xF6, // PN Code ID 27 - Index 20
    0x9B, 0x75, 0xF7, 0xE0, 0x14, 0x8D, 0xB5, 0x80, // PN Code ID 28 - Index 21
    0xBF, 0x54, 0x98, 0xB9, 0xB7, 0x30, 0x5A, 0x88, // PN Code ID 29 - Index 22
    0x35, 0xD1, 0xFC, 0x97, 0x23, 0xD4, 0xC9, 0x88, // PN Code ID 30 - Index 23
    0xE1, 0xD6, 0x31, 0x26, 0x5F, 0xBD, 0x40, 0x93, // PN Code ID 31 - Index 24
    0xDC, 0x68, 0x08, 0x99, 0x97, 0xAE, 0xAF, 0x8C, // PN Code ID 32 - Index 25
    0xC3, 0x0E, 0x01, 0x16, 0x0E, 0x32, 0x06, 0xBA, // PN Code ID 33 - Index 26
    0xE0, 0x83, 0x01, 0xFA, 0xAB, 0x3E, 0x8F, 0xAC, // PN Code ID 34 - Index 27
    0x5C, 0xD5, 0x9C, 0xB8, 0x46, 0x9C, 0x7D, 0x84, // PN Code ID 35 - Index 28
    0xF1, 0xC6, 0xFE, 0x5C, 0x9D, 0xA5, 0x4F, 0xB7, // PN Code ID 36 - Index 29
    0x58, 0xB5, 0xB3, 0xDD, 0x0E, 0x28, 0xF1, 0xB0, // PN Code ID 37 - Index 30
//  0x9A, 0xD6, 0x95, 0xBA, 0xA4, 0xC5, 0x32, 0x9C, // PN Code ID 38 - Not Used
    0x5F, 0x30, 0x3B, 0x56, 0x96, 0x45, 0xF4, 0xA1, // PN Code ID 39 - Index 31
    0x03, 0xBC, 0x6E, 0x8A, 0xEF, 0xBD, 0xFE, 0xF8, // PN Code ID 40 - Index 32
    0x88, 0x17, 0x13, 0x3B, 0x2D, 0xBF, 0x06, 0xD6, // PN Code ID 41 - Index 33
    0xF1, 0x94, 0x30, 0x21, 0xA1, 0x1C, 0x88, 0xA9, // PN Code ID 42 - Index 34
    0xD0, 0xD2, 0x8E, 0xBC, 0x82, 0x2F, 0xE3, 0xB4, // PN Code ID 43 - Index 35
    0x8C, 0xFA, 0x47, 0x9B, 0x83, 0xA5, 0x66, 0xD0, // PN Code ID 44 - Index 36
    0x07, 0xBD, 0x9F, 0x26, 0xC8, 0x31, 0x0F, 0xB8, // PN Code ID 45 - Index 37
//  0xD7, 0xA1, 0x54, 0xB1, 0x5E, 0x89, 0xAE, 0x86, // PN Code ID 46 - Not Used
    0xEF, 0x03, 0x95, 0x89, 0xB4, 0x71, 0x61, 0x9D, // PN Code ID 47 - Index 38
    0x40, 0xBA, 0x97, 0xD5, 0x86, 0x4F, 0xCC, 0xD1, // PN Code ID 48 - Index 39
```

10.5 64kbps PN Codes

```
    0x6A, 0xE7, 0x01, 0xEA, 0x03, 0xFD, 0x13, 0xD2, // PN Code ID 1
    0xDC, 0xC0, 0x6B, 0xB8, 0x2B, 0x09, 0xBB, 0xB2, // PN Code ID 2
    0xA3, 0x1E, 0xF2, 0xA4, 0x31, 0x32, 0x7A, 0xB3, // PN Code ID 3
```

```
0x44, 0x83, 0x3B, 0xDD, 0x14, 0xCF, 0x8E, 0xC9, // PN Code ID 4
0x35, 0x35, 0x4E, 0xC5, 0xF3, 0x52, 0x47, 0xB0, // PN Code ID 5
0x7C, 0x23, 0x8A, 0xCE, 0x45, 0x5C, 0x54, 0xD7, // PN Code ID 6
0x81, 0xAC, 0xFB, 0x83, 0x7A, 0x9A, 0x61, 0xAC, // PN Code ID 7
0x3C, 0x12, 0x5F, 0x9C, 0x39, 0x98, 0xF6, 0x8A, // PN Code ID 8
```

11. REFERENCES

Consistent Overhead Byte Stuffing (COBS) (IEEE April 1999 Transactions on Networking Paper)

WirelessUSB LR 2.4-GHz DSSS Radio SoC (Document 38-16008)

12. INDEX

I

L

M

N

P

R

S

T

U

X

Revision	Date	Changes
1.0	10/12/04	Initial Release