



User Manual Reference Design

ANTDKT3

ANTDKT3UP1

ANTAP2DK1

D00001087 Rev1.4

P +1 403.932.4620 F +1 403.932.6521

Copyright Information and Usage Notice

Restricted use ANT Development Kits

ANT Development Kits are intended for use solely by design engineers for the purpose of evaluating the feasibility of ultra low-power wireless data communications applications. The user's evaluation must be limited to use of an assembled ANT Development Kit within a laboratory setting which provides for adequate shielding of RF emission which might be caused by operation of the Development Kit following assembly. The assembled Development Kit must not be operated in a residential area or any area where radio devices might be subject to harmful electrical interference. Hardware contained in the Development Kit may not be certified for use by the FCC in accordance with Part 15, or to other known standards of operation governing radio emissions. Distribution and sale of this Development Kit is intended solely for use in future development of devices which may be subject to FCC regulation, or other authorities governing radio emission. This Development Kit may not be resold by users for any purpose. Operation of the Development Kit in the development of future devices is deemed within the discretion of the user and the user shall have all responsibility for any compliance with any FCC regulation or other authority governing radio emission of such development or use. All products developed by the user must be approved by the FCC or other authority governing radio emission prior to marketing or sale of such products and user bears all responsibility for obtaining the authority's prior approval, or approval as needed from any other authority governing radio emission. If user has obtained the Development Kit for any purpose not identified above, user should return the Development Kit to Dynastream Innovations Inc. immediately. The Development Kit is an experimental device, and Dynastream makes no representation with respect to the adequacy of the Development Kit in developing ultra low-power wireless data communications applications or systems. The Development Kit and products based on the technology in the Development Kit operate on shared radio channels. Any Products using ANT technology must be designed so that a loss of communications due to radio interference or otherwise will not endanger either people or property, and will not cause the loss of valuable data. Dynastream assumes no liability for the performance of products which are designed or created using the Development Kit.

The Dynastream Innovations Inc. ANT Products described in this document are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Dynastream product could create a situation where personal injury or death may occur. If you use the Products for such unintended and unauthorized applications, you do so at your own risk and you shall indemnify and hold Dynastream and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Dynastream was negligent regarding the design or manufacture of the Product.

The information disclosed herein is the exclusive property of Dynastream Innovations Inc. and is not to be reproduced and/or distributed without the written consent of Dynastream Innovations Inc. No part of this publication may be reproduced or transmitted in any form or by any means including electronic storage, reproduction, execution or transmission without the prior written consent of Dynastream Innovations Inc. The recipient of this document by its retention and use agrees to respect the security status of the information contained herein.

This document is intended for limited circulation.

The information contained in this document is subject to change without notice and should not be construed as a commitment by Dynastream Innovations Inc. unless such commitment is expressly given in a covering document

Reference Design Disclaimer

The references designs and codes provided with the development kit may be used with ANT devices only and remain the copyrighted property of Dynastream Innovations Inc. The reference designs and codes are being provided on an "as-is" basis and as an accommodation, and therefore all warranties, representations, or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.

©2009 Dynastream Innovations Inc. All Rights Reserved.

TABLE OF CONTENTS

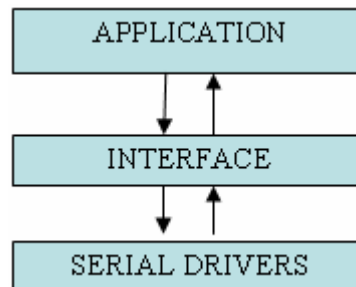
COPYRIGHT INFORMATION AND USAGE NOTICE.....	2
1 INTRODUCTION	4
2 SERIAL DRIVER IMPLEMENTATION	5
2.1 SERIAL LINES' INITIALIZATION AND SYNCHRONOUS RESET SEQUENCE	5
2.2 BIT SYNCHRONOUS COMMUNICATION	6
2.3 INTERFACE BETWEEN THE APPLICATION AND THE SERIAL DRIVERS.....	8
3 CODE EXAMPLES	10
3.1 FREQUENCY HOPPING	10
3.1.1 <i>Design Description</i>	10
3.1.2 <i>Hardware Setup</i>	11
3.2 RELAY DEVICE	11
3.2.1 <i>Design Description</i>	12
3.2.2 <i>Hardware Setup</i>	12
3.2.3 <i>Closing Remarks</i>	14
3.3 AUTO SHARED CHANNELS	14
3.3.1 <i>Design Description</i>	14
3.3.2 <i>Hardware Setup</i>	16
3.4 LED CONTROL.....	17
3.4.1 <i>Design Description</i>	17
3.4.2 <i>Hardware Setup</i>	17
APPENDIX A - ANT MODULE TO TI EZ430 INTERFACE BOARD	19

1 Introduction

These reference designs are provided to aid with the development of firmware for HOST MCU applications interfacing to the ANT protocol. Each reference design consists of firmware targeted for the TI MSP430-2013 MCU. However, the firmware has been written such that it can easily be ported to any HOST MCU. In order to run the reference designs described below, a TI eZUSB MSP430-2013 development kit, in addition to the ANT development kit, is required. The MSP430 develop board must be interfaced to an ANT module and configured for bit synchronous data communication (for details see the document *Interfacing with ANT General Purpose Chipsets and Modules*).

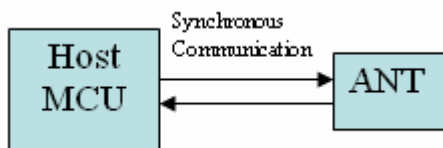
The reference design document and source codes are zipped into one package available for download from ANT developer's zone.

Each application described in the reference designs consists of three layers: application, interface and driver. The application layer communicates with the ANT interface layer and the ANT interface layer communicates directly with the serial driver. The figure below describes the flow of data in the reference designs. The interface and driver layer are common to each reference design.



2 Serial Driver Implementation

The reference designs utilize a 'Bit-Synchronous' serial interface between the ANT device and the TI MSP430 microcontroller. The reference designs implement 'Bit Bashing' based on the assumption that hardware UART is not available.



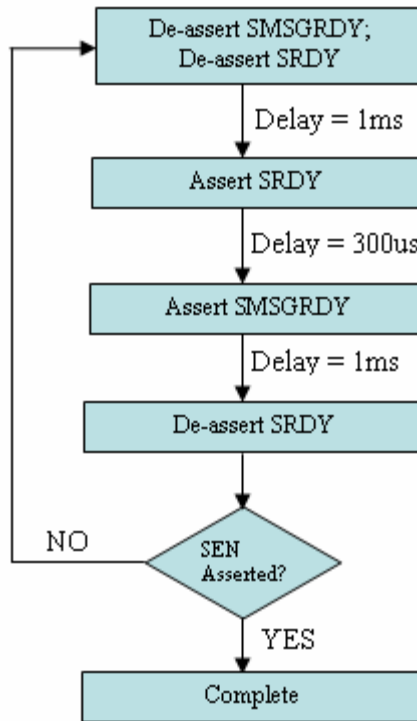
2.1 Serial Lines' Initialization and Synchronous Reset Sequence

The pins and interrupts used to communicate with the ANT device must first be initialized, including all the serial lines (SEN, SIN, SOUT, SMSGRDY, SRDY, and SCLK). Initialization is followed by performing the Synchronous Reset sequence for the ANT device. Each of the serial lines corresponds to a pin on the microcontroller's Port P1:

MCU Pin	Serial Lines	Direction w.r.t to MCU
P1.1	SMSGRDY	Output
P1.2	SOUT	Output
P1.3	SIN	input
P1.4	SCLK	Input
P1.5	SRDY	Output
P1.6	SEN	Input

Initialization of the serial lines is implemented by the function `BitSyncSerial_Init ()` and the Synchronous Reset sequence is implemented by the function `BitSyncSerial_SyncReset ()`; both functions are found in the file `BitSyncSerial.c`. The Synchronous Reset sequence comprises a particular sequence of the $\overline{\text{SMSGRDY}}$ and $\overline{\text{SRDY}}$ signals as described in the *Interfacing with ANT General Purpose Chipsets and Modules* document. It is important to perform a synchronous reset during the initialization process to ensure both the HOST and ANT serial ports are in a known state.

The following flowchart illustrates the process flow of the Synchronous Reset sequence.



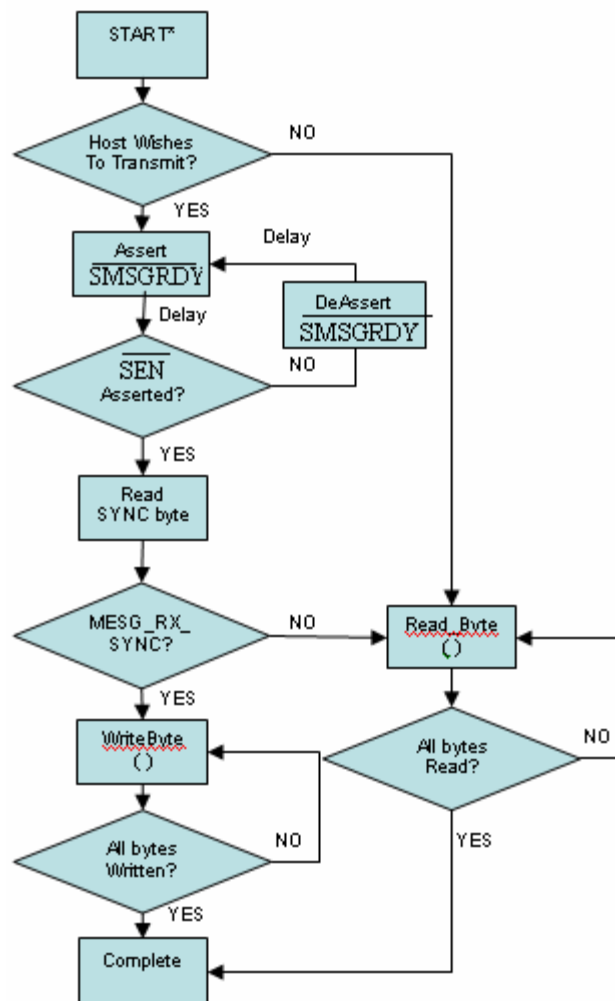
The first transaction after a synchronous reset is always from the HOST to ANT device. Therefore, the $\overline{\text{SMSGRDY}}$ signal can stay asserted following the synchronous reset because it will get de-asserted after a HOST to ANT transaction.

2.2 Bit Synchronous Communication

In the file BitSyncSerial.c, the function used to check for messages from the ANT module as well as to transmit messages to the ANT module is BitSyncSerial_Transaction().

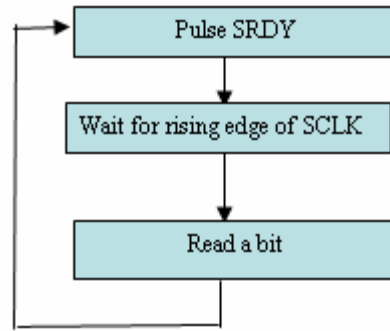
If the HOST MCU has a message to transmit, it asserts the $\overline{\text{SMSGRDY}}$ line to notify ANT and continues to pulse $\overline{\text{SMSGRDY}}$ until ANT asserts the $\overline{\text{SEN}}$ line. After ANT asserts the $\overline{\text{SEN}}$ line the HOST checks if it has received a message from ANT that it needs to process.

The first byte (MSG_RX_SYNC or MSG_TX_SYNC) received from ANT indicates to the HOST MCU if ANT is ready to receive a message or if ANT wants to transmit a message. Based on the SYNC byte, data is either transmitted using WriteByte() or received using ReadByte().



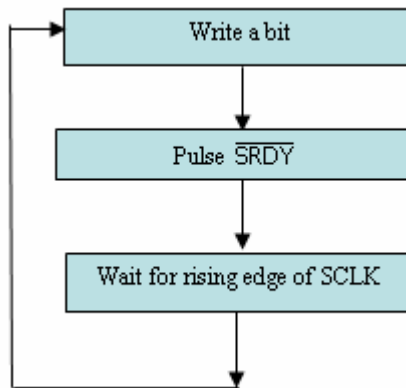
*The loop that checks if the $\overline{\text{SEN}}$ line is asserted, will not exit if the ANT module is not plugged in. Refer to the BitSyncSerial_Transaction() function in BitSyncSerial.c for this loop.

The ReadByte() function reads a single byte from the ANT module. Bit processing by the HOST MCU is done on the rising edge of the SCLK pulse. The $\overline{\text{SRDY}}$ signal is asserted to indicate that the HOST is ready to receive a bit. The HOST then waits for the rising edge of the SCLK pulse to read the bit. The rising edge of the SCLK pulse is captured by an interrupt. The interrupt is implemented by the function INT_PORT1() in each of the MainDemo.c files.



The WriteByte() function is used to write a single byte to the ANT module. After a bit is written, the SRDY line is pulsed to indicate that the bit can be read by ANT device. The HOST then waits for the rising edge of the SCLK line which confirms that the bit has been read by ANT device.

The following flowchart illustrates the data flow in the WriteByte() function.



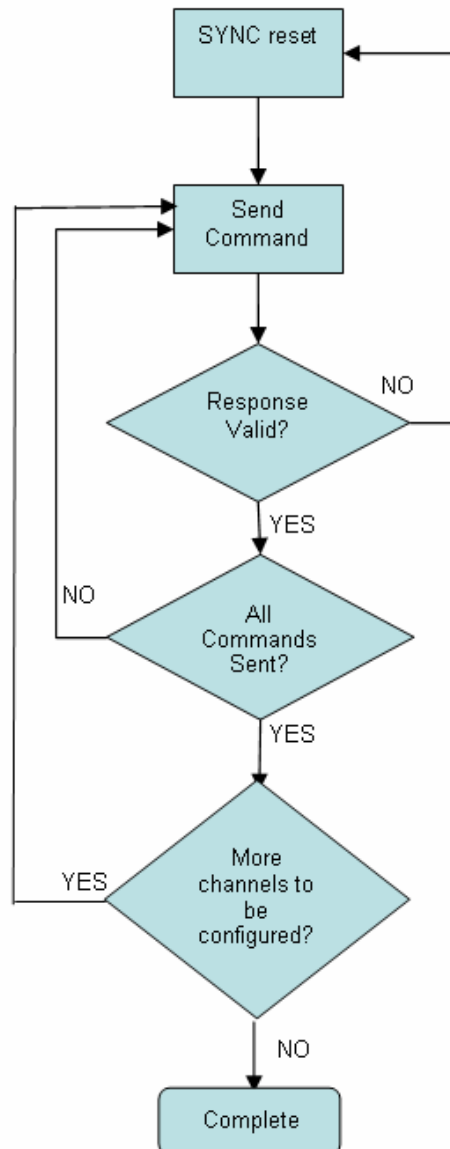
2.3 Interface between the Application and the Serial Drivers

The application layer communicates with the serial drivers through the Application Programming Interface (API) provided by ANT protocol. The ANT API comprises configuration commands that are used by the HOST MCU to configure the ANT device. The configuration entails verification of the commands and data transmitted and received. Various Channel Event codes indicate whether a command or data has been correctly received by the ANT device. These commands are listed in the ANT Message Protocol document.

The file ANTInterface.c contains the implementation of the interface between the application and the serial drivers.

The ANT device must be initialized and this is done by the ANTInterface_Init() function. This is followed by verifying if the HOST MCU wants to transmit/receive to/from the ANT device. If a message is received, a flag is set and the received message is then processed.

The ANT channel configuration is implemented by the function ANTInterface_ChannelConfig(). The data flow of this function is shown below:



The function `ANTInterface_MesgProcess()` is used to send/receive message to/from ANT. This function simply calls the function `BitSyncSerial_Transaction()` which is documented above. If a message has been received, this function will set a flag indicating this case. The application code can then query this flag using the function `ANTInterface_GetPendingEvent()`, which returns a pointer to the buffer of the received message.

3 Code Examples

The reference design code examples are all built on the same common code base as described in the sections above. They are for demonstration purposes only and may be expanded and configured to suit any specific application need. They are differentiated at the application level, so as to illustrate various uses and implementations of an ANT device. The application level code for each example is contained in a single file. The following table describes what each reference design file contains. Following is a description of the reference designs and sample implementations.

File	Description
Main_Demo1.c	Frequency Hopping (Master)
Main_Demo2.c	Relay Device
Main_Demo3.c	Auto Shared Channel (Slave)
Main_Demo4.c	Frequency Hopping (Slave)
Main_Demo5.c	LED Control

3.1 Frequency Hopping

The purpose of the Frequency Hopping reference design is to illustrate how a simple frequency hopping transmit and receive mechanism can be implemented at the application level. Frequency hopping requires that both the transmitter and the receiver switch among many frequency channels in a sequence that is known to both of them. The advantages of frequency hopping include resistance to noise and interference and increased security (frequency hopping signals are difficult to intercept).

3.1.1 Design Description

In order to realize a frequency hopping ANT network, a MASTER and a SLAVE node must be configured to transmit and receive at a prescribed set of frequencies. The example code to implement this reference design is contained in Main_Demo1.c for the MASTER and Main_Demo4.c for the SLAVE.

The MASTER and SLAVE are setup to hop through the following 4 frequencies (in MHz):

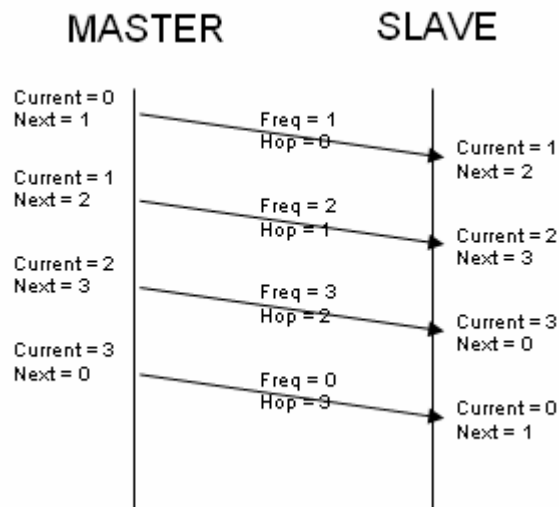
... 2466, 2469, 2472, 2475 ...

The hop order, specific frequencies and number of frequencies are arbitrary and application dependent. The sequence above was chosen for demonstration purposes only.

The MASTER communicates the frequency hopping to the SLAVE by including this information in the data payload of the broadcast message. The broadcast data packet payload for this reference design is defined as follows:

Byte[]	Description	Value
0	Message length	0x09
1	Message ID (Broadcast)	0x4E
2	Channel number	0x00
3	Hop frequency index. Frequency index of the frequency at which <i>next</i> message will be sent.	0..3
4	Reserved	0x00
5	Message type (standard temperature)	
6	A/D subtype	0xFE
7	Reserved	0x00
8	Reserved	0x00
9	Temperature LSB	0..255
10	Temperature MSB	0..255

The MASTER node actually sets its *current* transmit frequency in the data packet it is about to transmit to the SLAVE. It then hops to the *next* frequency before it sends this data to the SLAVE. The SLAVE knows that the *next* MASTER transmit frequency will be offset by 2 from the frequency reported in the data buffer (or 1 away from the current frequency). This way the MASTER and the SLAVE will always stay synchronous to each other. The MASTER controls which frequency will be next in the hop sequence.



On startup, the SLAVE node will always be configured to search at the default frequency (2466MHz) for a MASTER. Hence, the search time may be prolonged as the MASTER makes its way through the various transmit frequencies until it uses the default frequency again. Once a MASTER is found and the channel established, the SLAVE will monitor the status of the channel. If the communication link is lost, the SLAVE state machine in the application is configured such that it will cycle through the possible frequencies sequentially until the MASTER is found or a timeout condition occurs.

3.1.2 Hardware Setup

To run this reference design example, the following hardware is required:

- 2X TI eZ430-F2013 development boards
- 2X ANT modules
- 2X ANT module to EZ430 interface boards (not provided in the development kit – see APPENDIX A)

One of the ANT modules must be configured to run as the frequency hopping MASTER. To do this, build the Main_Demo1 project and load onto the processor using the IAR Workbench IDE. Similarly, configure the SLAVE frequency hopping node by building the Main_Demo4 project and loading it onto the other ANT module.

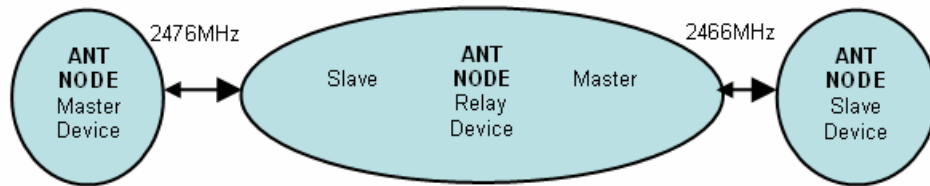
To run, reset or power cycle the MASTER and SLAVE modules. The SLAVE module LED should toggle each time a message is received from the MASTER. Alternatively, the IAR IDE may be used to run the code directly, allowing one to step through the code to look at various buffer and variable states.

3.2 Relay Device

The purpose of the Relay Device reference design is to illustrate how a simple relay device may be implemented at the application level. The purpose of a relay device is to pass data from one ANT device to another. For example, a light switch-to-relay-to-control panel application. This allows the physical range of a network to be extended over a much larger distance. The ability to relay data between devices is a critical key to mesh type network implementations.

3.2.1 Design Description

The example code that implements the Device Relay reference design is contained in the file Main_Demo2.c. The relay works by opening up two channels at two different frequencies. One channel is setup as a MASTER, the other as a SLAVE. Data received on one channel is re-packaged and copied over for transmission to the other channel. The relay is bi-directional, meaning that data can travel from the MASTER channel to the SLAVE channel and vice-versa. Although the relay is capable of handling both broadcast and acknowledge data types, it should be noted that acknowledge behavior should be handled at the application level. A positive acknowledge from the Relay does not guarantee that the message reached its destination. Hence, a transmitter state machine cannot be advanced based on positive acknowledges of a message. Burst data transactions are not supported in this implementation.



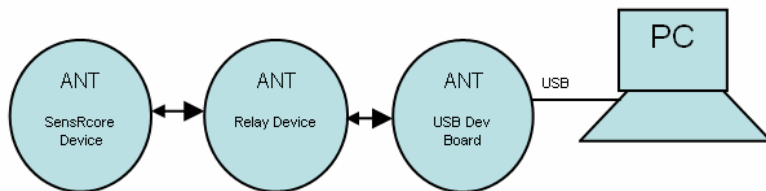
In the figure above, a MASTER device is communicating with a SLAVE device via a Relay device. If the Relay was disabled or removed from the picture, the communication path would be broken.

The table below describes the channel parameters used by the Relay Device.

Chan	Type	Dev	Type	Trans	Freq
0	MASTER	49	1	3	2466
1	SLAVE	49	1	3	2476

3.2.2 Hardware Setup

An ANT relay network can be easily implemented using components available in the ANT development kit in conjunction with the eZ430-F2013 development kit available from TI. In this example, an ANT node MASTER is configured using SensRcore™ and an ANT SLAVE is configured using *ANTware*. The Device Relay is implemented on the eZ430 development kit interfaced to an ANT module. It is assumed that the user is familiar with *ANTware* and *SensRware* as well as basic setup regarding the ANT development kit. Please see the ANT User Manual for more information. A block diagram of the network setup is depicted in the following figure.



The following hardware is required for this setup:

- 3X ANT modules, at least two ANT11TS33M5IB modules
- 1X ANT I/O interface Board
- 1X ANT Battery Board
- 1X ANT module to TI EZ430 Interface board (not provided in the development kit – see APPENDIX A)

- 1X USB interface Board
- 1X TI eZ430-F2013 development board

Configuration of the example network requires the following steps:

Step 1: Configure a SensRcore™ ANT MASTER node using *SensRware*. For this example, the SensRcore™ module will be configured to output the state of the buttons on the ANT I/O interface Board at a rate of 4Hz. To do this, the ANT module should be configured with the following parameters using *SensRware*:

Data Channel 0	
Data Type	Digital Input
ANT Channel #	0
Pin Mask/Number	15
Message Ratio	1
Message Ratio Offset	0
ANT Channel 0	
Channel Type	MASTER
Device Number	49
Device Type	1
Transmission Type	3
Message Period	4
RF Frequency	76

After configuring the SensRcore™ ANT module, attach it to the I/O interface Board and the Battery board. Set the dipswitches on the battery board:

	VCC	GND
1		X
2	X	
3		X
4		X
5		X

Step 2: Configure the Relay Device using the eZ430 development kit. Build the reference design "ANT Ref Design Demo 2" using IAR Workbench IDE. Load the executable onto the development board and execute. Make sure that an ANT module is attached to the eZ430 development board using the ANT EZ430 interface board or equivalent.

Step 3: Configure an ANT SLAVE node using *ANTware*. Plug in an ANT module to a USB interface board attached to a PC. Using the *ANTware* utility, configure the module to the following parameters:

ANT Channel 0	
Channel Type	SLAVE
Device Number	49
Device Type	1
Transmission Type	3
Message Period	4
RF Frequency	66

Step 4: Reset the system and observe the behavior. Reset the MASTER ANT node by power cycling or hitting the reset button. Reset the Relay Device by re-starting the execution of the firmware using the IAR

Workbench. Alternatively, if firmware has been loaded to flash and the eZ430 MCU is running stand-alone, power cycle the MCU to reset the device. Finally, reset the ANT SLAVE node by opening the SLAVE channel in *ANTware*.

The SLAVE channel on the relay device will quickly find the ANT MASTER channel and immediately begin copying data to its MASTER channel. The SLAVE ANT device running within *ANTware* will search and find the MASTER channel of the Relay device and begin displaying messages received in a text box. The ID of the message should be 0x40 (Digital Data). The message should appear in the *ANTware* text box as follows:

```
[40][00][00][00][00][00][0F][0F]
```

The last two bytes of the message signify the pin mask and pin state respectively. The pin mask should always be 0F. The pin state will change depending on which buttons are pressed. A pin will have a state of 1 if its button is not pressed, 0 if it is. For example, if button A is pressed, the pin state will change to 0E (first bit in byte is 0). If buttons A and B are pressed the pin state will change to 0C, and so on. Disabling the Relay will result in a loss of data and, eventually, a timeout condition in the SLAVE node.

3.2.3 Closing Remarks

A relay device can extend the physical range of a wireless network by passing data from one ANT device to another. It should be noted that although this implementation used two channels operating at difference frequencies to implement the relay, this is not a requirement for a relay device. In fact, setting up a relay with two channels operating at identical frequencies could potentially provide a mechanism where a network will continue to operate even if the relay is disabled.

3.3 Auto Shared Channels

The purpose of the Auto Shared Channel reference design is to illustrate how an auto shared channel SLAVE device can be configured on a HOST MCU. Auto shared channels are useful for situations where a MASTER device may wish to communicate to a large number of SLAVE devices without prior knowledge of the SLAVE shared address over a single channel. A slave address is assigned by the MASTER to the SLAVE dynamically. Complete details, including state machine diagrams for the auto shared channel implementation are detailed in ANT application note 7 ("ANT AN07 – Auto Shared Channel – Master Example") and will not be repeated here.

3.3.1 Design Description

The code that implements the Auto Shared Channel SLAVE may be found in the file Main_Demo3.c. The Auto Shared Channel reference design implements a state machine for handling a dynamic slave address assignment as described in application note 07.

The ANT channel in the reference design is setup with the following parameters:

Chan	Type	Dev	Type	Trans	Freq
0	SHARED SLAVE	49	1	7	2466

The message period is set to 8Hz.

The application utilizes a serial number to send to the MASTER as its unique ID. This serial number is embedded in the code as:

```
0x04 0x00 0x00 0x00
```

Please note that the unique ID (i.e. the serial number) must be changed for each slave that is added to a network. For example, if two auto shared slaves are to be setup, the reference code must be re-compiled with the hard coded serial number changed to something different to the address listed above.

The auto shared slave reference design defines the following states:

- NO_STATE
- SEARCHING_FOR_ADDRESS
- REQUESTING_ADDRESS
- WAITING_FOR_CONFIRM
- CONFIRMING_ADDRESS
- ADDRESS_SET
- OPERATING

3.3.1.1 NO_STATE

This is an illegal state that the state machine should never enter. It is used to initialize a temporary state variable.

3.3.1.2 SEARCHING_FOR_ADDRESS

This is the initial state of the state machine and is entered upon start-up or reset. The slave address at this point is set to 0xFFFF to indicate that the SLAVE is searching for an address. While in this state, the SLAVE device will listen for a "SLOT AVAILABLE" message from a MASTER. Once such a message is received, the SLAVE will send a "REQUEST_TO_ACQUIRE" message to the MASTER. This is an acknowledged data type message. Subsequently the SLAVE will immediately go into the REQUESTING_ADDRESS state.

3.3.1.3 REQUESTING_ADDRESS

In this state, the SLAVE is waiting for a positive acknowledge from the MASTER to the REQUEST_TO_ACQUIRE message. If the acknowledge is positive, the SLAVE will set its address to 0xFFFE to indicate that it is in the process of acquiring a SLAVE address from a MASTER. It will then immediately go to the WAITING_FOR_CONFIRM state. In the event that the REQUEST_TO_ACQUIRE message failed, the SLAVE state machine will return to the SEARCHING_FOR_ADDRESS state.

3.3.1.4 WAITING_FOR_CONFIRM

In this state, the SLAVE is waiting for a BUSY_ACQUIRING message from the MASTER. Once this message is received, the SLAVE will check that the unique ID set in this message does in fact match the unique ID of the SLAVE. If it does **not**, the SLAVE state machine will return to the SEARCHING_FOR_ADDRESS state. Otherwise, the SLAVE will send a CONFIRM_ACQUIRED message to the MASTER and immediately go to the CONFIRMING_ADDRESS state.

3.3.1.5 CONFIRMING_ADDRESS

In this state the SLAVE waits for a response to the CONFIRM_ACQUIRED message that it sent in the previous state. If the acknowledge is positive, the SLAVE will set its address to the assigned slave address and immediately go to the ADDRESS_SET state. Otherwise it will return to the SEARCHING_FOR_ADDRESS state.

3.3.1.6 ADDRESS_SET

In this state the slave address of the SLAVE has been set and the device is ready to receive regular data from the MASTER. In the event of a timeout situation, for example if the MASTER is disabled, the SLAVE will return to the SEARCHING_FOR_ADDRESS state.

3.3.1.7 OPERATING

This is the state at which a channel has been established between the MASTER and the SLAVE and application data has been received. It is entered once the assigning of a slave address is complete and the SLAVE begins receiving data from the MASTER.

3.3.2 Hardware Setup

The Auto Shared reference design may be tested by using the software tool supplied with application note 7. This tool can be used to set up a shared channel MASTER node to handle the communication with the shared channel SLAVE node described in this reference design. Please refer to the specified application note for details. The following hardware will be required:

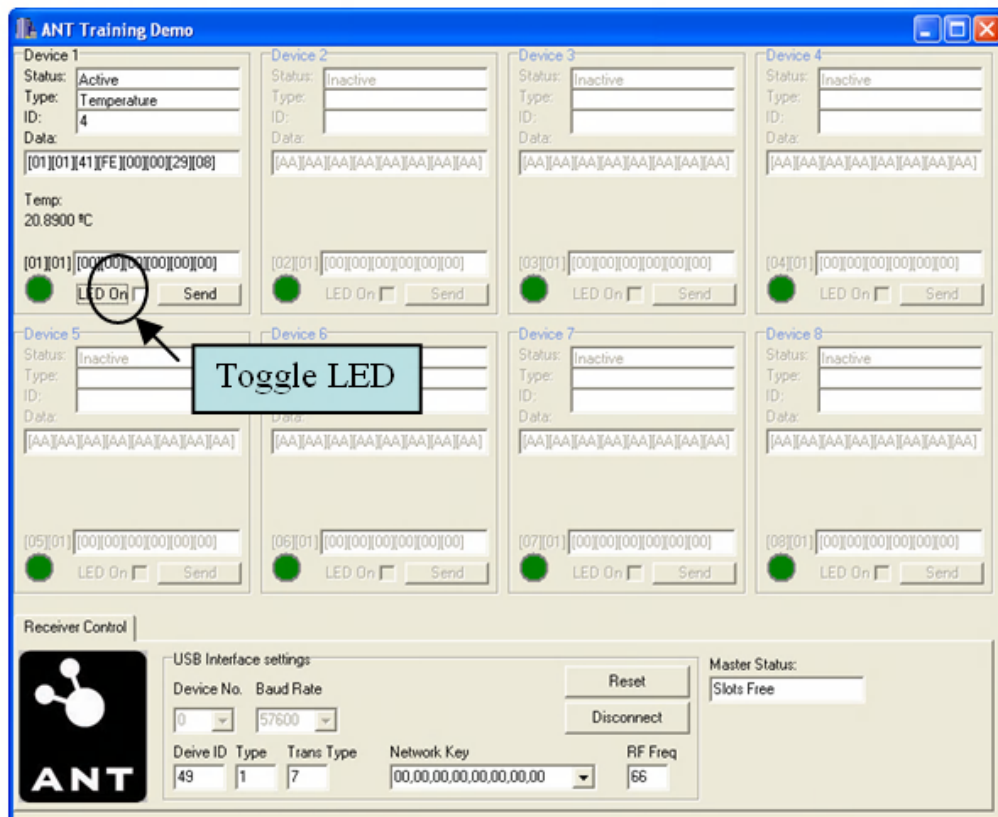
- 2X ANT modules, at least one ANT11TS33M5IB or ANTAP281M5IB modules.
- 1X ANT module to eZ430 Interface board (not provided in the development kit – see APPENDIX A)
- 1X USB interface Board
- 1X TI eZ430-F2013 development board

The following steps are required in implement this example:

Step 1: Connect ANT USB interface board with an ANT module to PC. Run "ASC" application and connect to the device using channel settings described in section 1.3.1.

Step 2: Connect eZ430 USB stick with ANT interface board and ANT11TS33M5IB / ANTAP281M5IB module to the PC. Run IAR Workbench. Build, load and execute the "ANT Ref Design Demo 3".

Step 3: The MASTER device should detect and establish an auto shared channel with the SLAVE. This will be reflected in the ANT PC utility used in step 1. To test that the connection is well established, click the LED check box in the software utility associated with the open channel. This should toggle the LED on the eZ430 development board.



3.4 LED Control

The LED Control reference design is a simple application which demonstrates how to receive and interpret ANT messages. The ANT message received is used to toggle an LED on the TI eZ430 development board. It also provides a good example of interfacing to SensRcore™ devices.

3.4.1 Design Description

The code to implement the LED Control reference design may be found in the file Main_Demo5.c. The application opens up one SLAVE channel and subsequently listens for one message – the SensRcore™ Digital Input message (0x40). Upon receiving this message, it will determine whether the state of the first IO digital input has changed on the remote ANT node. If it has, the application will toggle the state of the on-board LED. If it has not, nothing is done. Effectively, this is an edge triggered LED control; the on-board LED is toggled on the HIGH to LOW transition of a digital input on a remote ANT SensRcore™ node. The following table lists the SLAVE channel parameters utilized by the LED Control reference design. Details regarding the SensRcore™ Digital Input message may be found in the document "ANT SensRcore™ Message and Usage".

Chan	Type	Dev	Type	Trans	Freq
0	SLAVE	49	1	5	2466

3.4.2 Hardware Setup

This reference design is designed to work with a remote SensRcore™ ANT module mounted on an I/O interface Board and battery board. The following hardware is required to realize this setup:

- 2X ANT modules, at least one ANT11TS33M5IB module
- 1X ANT I/O interface Board
- 1X ANT Battery Board
- 1X ANT module to TI eZ430 Interface board (not provided in the development kit – see APPENDIX A)
- 1X USB interface Board
- 1X TI eZ430-F2013 development board

The following steps are required to implement this example:

Step 1: Program an ANT module with SensRcore™ (ANT11TS33M5IB). Mount the ANT module onto the ANT USB development board and plug into the PC. Using the *SensRware* software utility, program the following SensRcore™ setup:

Data Channel 0	
Data Type	Digital Input
ANT Channel #	0
Pin Mask/Number	1
Message Ratio	1
Message Ratio Offset	0
ANT Channel 0	
Channel Type	MASTER
Device Number	49
Device Type	1
Transmission Type	5
Message Period	4
RF Frequency	66

Remove the module from the USB board and assemble the remote ANT stack using the I/O Board and the battery board. Be sure that the dip switches on the battery board are setup as follows:

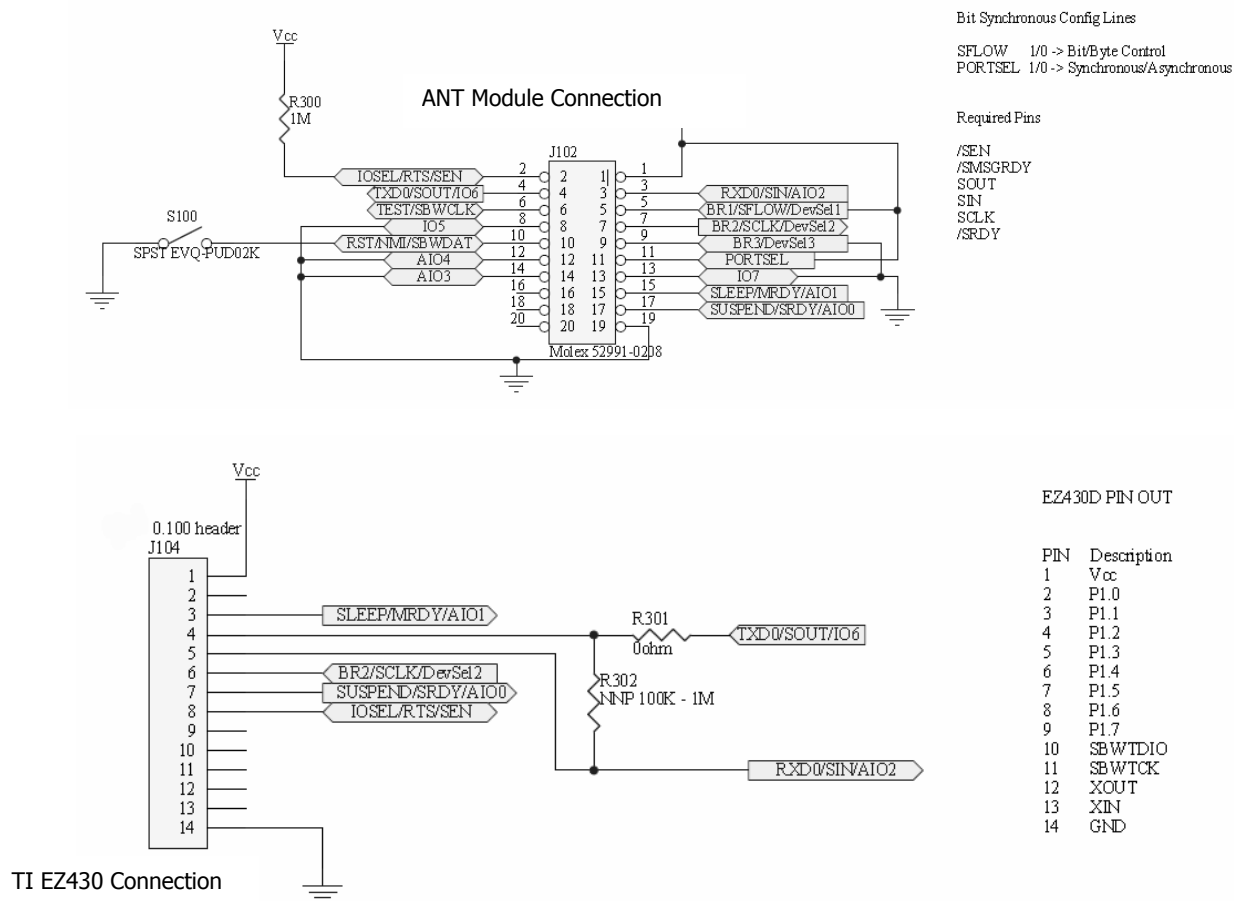
	VCC	GND
1		X
2	X	
3		X
4		X
5		X

Step 2: Configure the SLAVE device. Attach another ANT module to the eZ430 development board and plug into the PC. Build the ANT Ref Design Demo 5 application and load onto the development board.

Step 3: Run the demo by resetting both ANT nodes. Pressing the "A" button on the remote ANT module IO board should toggle the state of the LED on the eZ430 development board.

APPENDIX A - ANT module to TI EZ430 Interface board

The schematics of the ANT module to TI EZ430 interface board is provided below. Gerber files of a reference PCB layout are provided on this package downloadable from ANT developer's zone.



For more information regarding interfacing external MCU to ANT modules, please refer to the interface control document "Interfacing with ANT General Purpose Chipsets and Modules".