Development Kit For the PIC® MCU

Exercise Book

Wireless - ZMD Edition

August 2009

Custom Computer Services, Inc.
Brookfield, Wisconsin, USA
262-522-6500

Copyright © 2009 Custom Computer Services, Inc.
All rights reserved worldwide. No part of this work may be reproduced or copied in any form by any means-electronic, graphic or mechanical, including photocopying, recording, taping or information retrieval systems-without written permission.
PIC® and PICmicro® are registered trademarks of Microchip Technology Inc. in the USA and in other countries.
ZigBee™ is a trademark of the ZigBee™ Alliance, Inc.

Recognized Microchip Third-Party Tool Provider

Custom Computer Services, Inc. proudly supports the Microchip brand with highly optimized C compilers and embedded software development tools.

1 UNPACKING AND INSTALLATION

Inventory

- ❑ Use of this kit requires a PC with Windows 95, 98, ME, NT, 2000 or XP. The PC must have a spare 9-Pin Serial or USB port, a CD-ROM drive and 75 MB of disk space.
- ❑ The diagram on the following page shows each component in the Wireless - ZMD Edition kit. Ensure every item is present.

Software

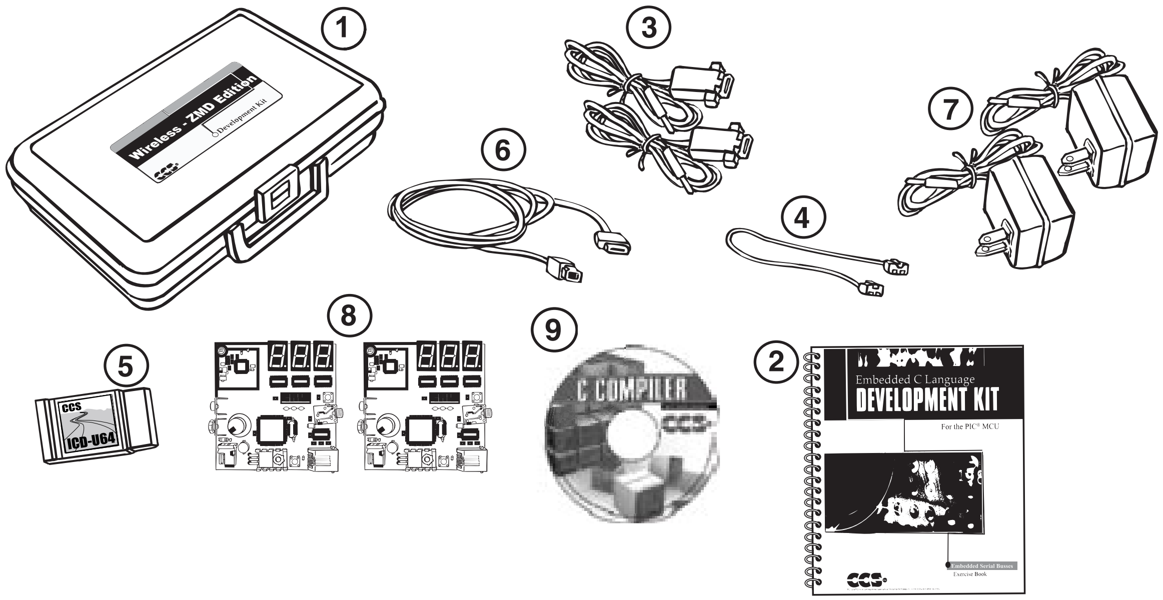
- ❑ Insert the CD into the computer and wait for the installation program to start. If your computer is not set up to auto-run CDs, then select **Start>Run** and enter **D:\SETUP1.EXE** where D: is the drive letter for your CD drive.
- ❑ Click on **Install** and use the default settings for all subsequent prompts by clicking NEXT, OK, CONTINUE…as required.
- ❑ Identify a directory to be used for the programs in this booklet. The install program will have created an empty directory **c:\program files\picc\projects** that may be used for this purpose.
- ❑ Select the compiler icon on the desktop. In the PCW IDE, click **Help>About** and verify a version number is shown for the IDE and PCH to ensure the software was installed properly. Exit the software.

Hardware

- ❑ Connect the PC to the ICD(5) using the USB cable.(1) Connect the prototyping board (8) to the ICD using the modular cable. Plug in the DC adaptor (7) to the power socket and plug it into the prototyping board (8). The first time the ICD-U is connected to the PC, Windows will detect new hardware. Install the USB driver from the CD or website using the new hardware wizard. The driver needs to be installed properly before the device can be used.
- ❑ The LED should be red(2) on the ICD-U to indicate the unit is connected properly.
- ❑ Run the Programmer Control Software by clicking on the CCSLOAD icon on the desktop. Use CCSLOAD Help File for assistance.
- ❑ The software will auto-detect the programmer and target board and the LED should be illuminated green. If any errors are detected, go to Diagnostic tab. If all tests pass, the hardware is installed properly.
- ❑ Disconnect the hardware until you are ready for Chapter 3. Always disconnect the power to the Prototyping board before connecting/disconnecting the ICD or changing the jumper wires to the Prototyping board.

(1) ICD-S40 can also be used in place of ICD-U. Connect it to an available serial port on the PC using the 9 pin serial cable. There is no driver required for S40.

(2) ICD-U40 units will be dimly illuminated green and may blink while connecting.



1. Carrying case
2. Exercise booklet
3. Two Serial PC to Prototype board cable
4. Modular cable (ICD to Prototyping board)
5. ICD unit for programming and debugging
6. Serial (or USB) PC to ICD cable
7. Two AC Adaptors (9VDC)
8. Two Prototyping boards with a PIC18LF452 processor chip,
   with ZMD wireless radio modules (installed on prototyping boards)
   *Antenna for boards is separate for shiping, install stripped end into 1-pin header
9. CD-ROM of C compiler (optional)

2 USING THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Editor

- ❑ Open the PCWH IDE. If any files are open, click **File>Close All**
- ❑ Click **File>Open>Source File**. Select the file: **c:\program files\picc\examples\ex_stwt.c**
- ❑ Scroll down to the bottom of this file. Notice the editor shows comments, preprocessor directives and C keywords in different colors.
- ❑ Move the cursor over the **Set_timer0** and click. Press the F1 key. Notice a Help file description for set_timer0 appears. The cursor may be placed on any keyword or built-in function and F1 will find help for the item.
- ❑ Review the editor special functions by clicking on **Edit**. The IDE allows various standard cut, paste and copy functions.
- ❑ Review the editor option settings by clicking on **Options>Editor Properties**. The IDE allows selection of the tab size, editor colors, fonts, and many more. Click on **Options>Toolbar** to select which icons appear on the toolbars.

Compiler

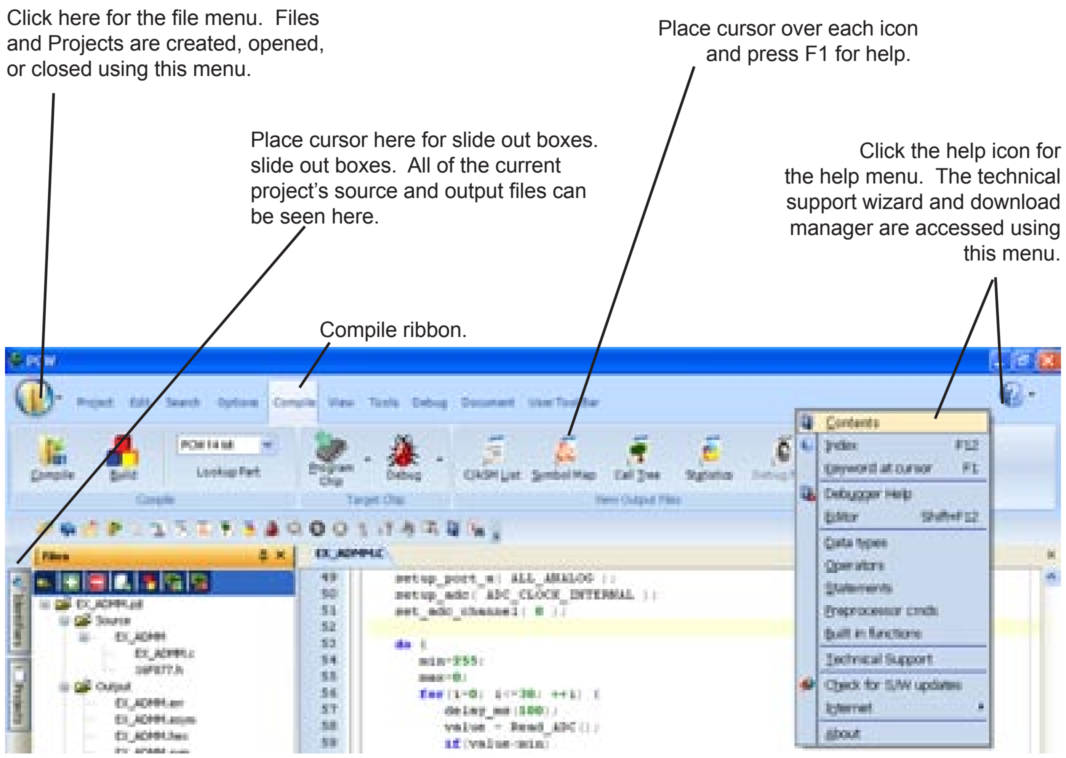
- ❑ Use the drop-down box under Compile to select the compiler. CCS offers different compilers for each family of Microchip parts. All the exercises in this booklet are for the PIC18LF452 chip, a 16-bit opcode part. Make sure Microchip **PCH 16 bit** is selected in the white box under the **Compiler** tab.
- ❑ The main program compiled is always shown in the bottom of the IDE. If this is not the file you want to compile, then click on the tab of the file you want to compile. Right click into editor and select **Make file project**.
- ❑ Click **Options>Project Options>Include Files…** and review the list of directories the compiler uses to search for included files. The install program should have put two directories in this list: devices and drivers.
- ❑ Normally the file formats need not be changed and global defines are not used in these exercises. To review these settings, click **Options>Project Options>Output Files** and **Options>Project Options>Global Defines**.
- ❑ Click the compile icon to compile. Notice the compilation box shows the files created and the amount of ROM and RAM used by this program. Press any key to remove the compilation box.

Viewer

- ❏ Click **Compile>Symbol Map**. This file shows how the RAM in the microcontroller is used. Identifiers that start with @ are compiler generated variables. Notice some locations are used by more than one item. This is because those variables are not active at the same time.
- ❏ Click **Compile>C/ASM List.** This file shows the original C code and the assembly code generated for the C. Scroll down to the line:

```
        int_count=INTS_PER_SECOND;
```

- ❏ Notice there are two assembly instructions generated. The first loads 4C into the W register. INTS_PER_SECOND is #defined in the file to 76. 4C hex is 76 decimal. The second instruction moves W into memory. Switch to the Symbol Map to find the memory location where int_count is located.
- ❏ Click **View>Data Sheet**, then **View.** This brings up the Microchip data sheet for the microprocessor being used in the current project.


Click here for the file menu. Files and Projects are created, opened, or closed using this menu.
Place cursor over each icon and press F1 for help.
Place cursor here for slide out boxes. slide out boxes. All of the current project's source and output files can be seen here.
Click the help icon for the help menu. The technical support wizard and download manager are accessed using this menu.
Compile ribbon.
PCW
Project Edit Search Options Compile View Tools Debug Document User Toolbar
Compile Build PCM 14 bit Lookup Part Program Chip Debug C/ASM List Symbol Map Call Tree Statistics
Compile Target Chip View Output Files
Files
EX_ADMM.pjt
Source
EX_ADMM
EX_ADMM.c
16F877.h
Output
EX_ADMM.err
EX_ADMM.asm
EX_ADMM.hex
EX_ADMM.c
Contents
Index F12
Keyword at cursor F1
Debugger Help
Editor Shift+F12
Data types
Operators
Statements
Preprocessor cmds
Built in functions
Technical Support
Check for S/W updates
Internet
About

```
   setup_port_a( ALL_ANALOG );
   setup_adc( ADC_CLOCK_INTERNAL );
   set_adc_channel( 0 );

   do {
      min=255;
      max=0;
      for(i=0; i<=30; ++i) {
         delay_ms(100);
         value = Read_ADC();
         if(value<min)
```

3 COMPILING AND RUNNING A PROGRAM

- ❑ Open the PCWH IDE. If any files are open, click **File>Close All**
- ❑ Click **File>New>Source File** and enter the filename **EX3.C**
- ❑ Type in the following program and **Compile.**

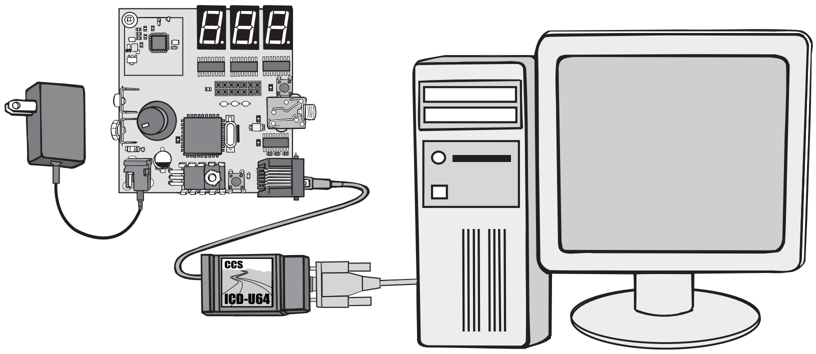
```
#include <18f452.h>
#device ICD=TRUE
#fuses  HS,NOLVP,NOWDT
#use delay (clock=10000000)

#define GREEN_LED PIN_A5

void main () {
       while (TRUE) {
              output_low (GREEN_LED);
              delay_ms (1000);
              output_high (GREEN_LED);
              delay_ms (1000);
       }
}
```

NOTES

- The first four lines of this program define the basic hardware environment. The chip being used is the PIC18LF452, running at 10MHz with the ICD debugger.
- The #define is used to enhance readability by referring to GREEN_LED in the program instead of PIN_A5.
- The “while (TRUE)” is a simple way to create a loop that never stops.
- Note that the “output_low” turns the LED on because the other end of the LED is +5V. This is done because the chip can tolerate more current when a pin is low than when it is high.
- The “delay_ms(1000)” is a one second delay (1000 milliseconds).



- ❑ Connect the ICD to the Prototyping board using the modular cable, and connect the ICD to the PC. Power up the Prototyping board.
- ❑ Click **Debug>Enable Debugger** and wait for the program to load.
- ❑ If you are using the ICD-U40 and the debugger cannot communicate to the ICD unit go to the debug configure tab and make sure ICD-USB from the list box is selected.
- ❑ Click the green go icon:
- ❑ Expect the debugger window status block to turn yellow indicating the program is running.
- ❑ The green LED on the Prototyping board should be flashing. One second on and one second off.
- ❑ The program can be stopped by clicking on the stop icon:

FURTHER STUDY

A *Modify the program to light the A5 LED for 5 seconds, then the B5 for 1 second and the B4 for 5 seconds.*

B *Add to the program a #define macro called "delay_seconds" so the delay_ms(1000) can be replaced with : delay_seconds(1); and delay_ms(5000) can be: delay_seconds(5);.*

Note: *Name these new programs EX3A.c and EX3B.c and follow the same naming convention throughout this booklet.*

4 ZIGBEE™ OVERVIEW

- ZigBee™ is a specified protocol for creating a wireless personal area network (WPAN). The two primary goals of the ZigBee™ organization was to create a WPAN that can run using the least amount of power and the least amount of resources. Low power and low complexity make ZigBee™ a very attractive wireless networking solution for embedded systems.
- The physical and MAC layer used by ZigBee™ is IEEE 802.15.4, and will be discussed in the next chapter. 802.15.4 operates in two frequency bands, 900MHz or 2.4GHz.
- There are three different types of ZigBee™ devices: full function devices (FFD) coordinators and reduced function devices (RFD). A FFD can route information between two nodes, or create entirely new networks. A coordinator is an FFD that has taken the task of creating and administrating the network. An RFD is the simplest form of device, and can only talk to an FFD – it cannot create a network or relay between two nodes.
- ZigBee™ coordinators can create three different kind of networks; Star, Cluster Tree or Mesh networks (figure 4.1):

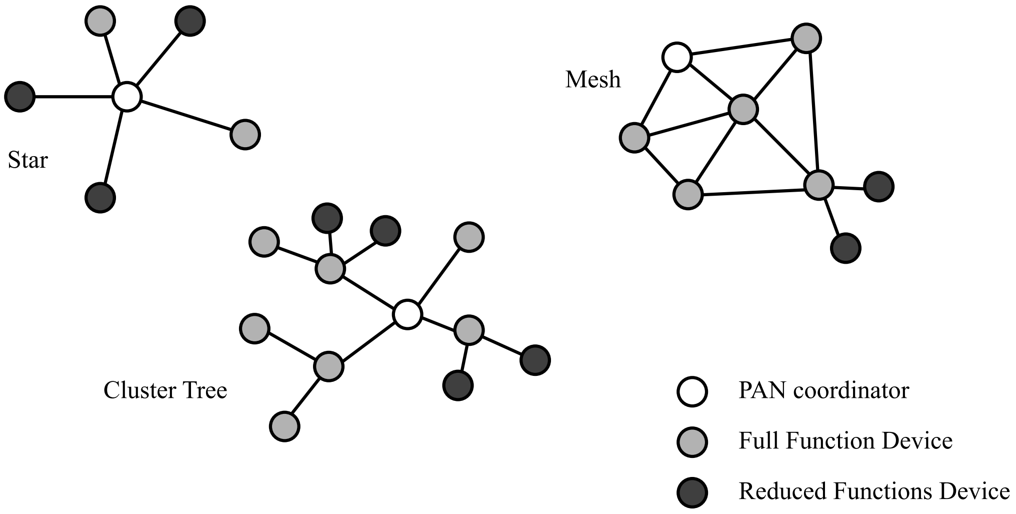


Figure 4.1

- ❑ The ability for a node to be a coordinator, router or end node is dependent on each unit’s capabilities. A RFD can only be an end node. Mesh and Cluster Tree networks provide for more stability and longer range than Star networks; sacrificing network/node complexity and larger power requirements.

5 802.15.4 OVERVIEW

- ❑ The physical/MAC layer of ZigBee™ is 802.15.4, which is a wireless standard aimed for low power consumption. Low power is achieved in two ways: low data rates and using a beacon to synchronize when nodes can go to sleep.
- ❑ The current 802.15.4 specification uses three frequency bands:

Frequency Band (MHz)	# Channels	Region	Modulation	Max Data Throughput (Kbit/s)
868.0 – 868.6	1	Europe	BPSK	20
902 – 928	10	North America, Japan	BPSK	40
2400 – 2483.5	16	Worldwide	O-QPSK	250

Figure 5.1 – 802.15.4 Frequency Bands

- ❑ 802.15.4 also uses 'Carrier Sense, Multiple Access' (CSMA) to prevent two nodes from attempting to talk at the same time. Beacons and acknowledge messages do not use CSMA.
- ❑ Here is an overview of the data sent by the 802.15.4 physical layer:

Bytes	4	1	1	x
	Preamble	SOF	Length	MAC Sublayer (MPDU)

Figure 5.2 – 802.15.4 PHY Layer

- ❑ The preamble and start-of-field marker are used to distinguish a start of 802.15.4. Length is the size of the MPDU.
- ❑ Here is an overview of the data sent in the MAC Sublayer (MPDU):

Bytes	2	1	4 to 20	n	2
	Frame Control	Seq Num	Address Field	Data Payload (MSDU)	FCS

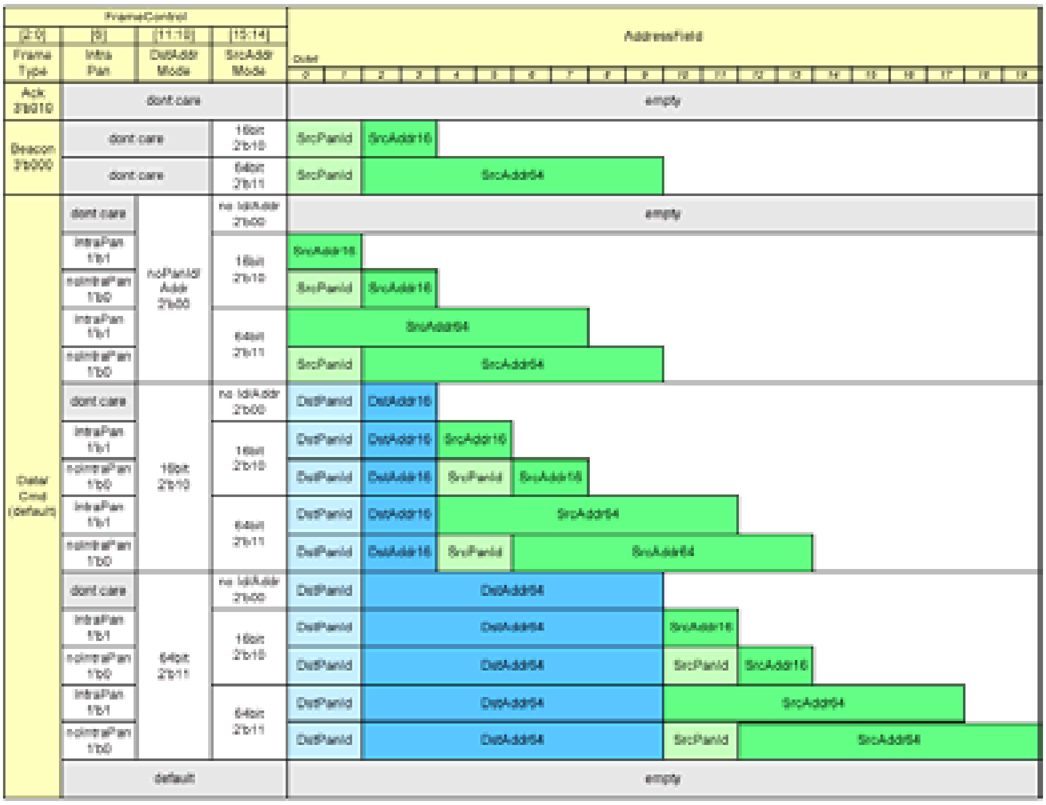
Figure 5.3 – 802.15.4 MAC Layer

- ❑ Addressing and frame control is specified by the 2 Frame Control bytes:

Bit	Description
0:2	Frame Type (0b000=Beacon, 0b001=Data, 0b010 = ACK, 0b011=Command)
3	Security Enabled
4	Frame Pending
5	Acknowledge Requested
6	IntraPan
7:9	RESERVED
10:11	Destination Addressing Mode (0b00=NONE, 0b10=16bit, 0b11=64bit)
12:13	RESERVED
14:15	Source Addressing Mode (0b00=NONE, 0b10=16bit, 0b11=64bit)

Figure 5.4 – 802.15.4 MAC Frame Control

- The size and content of the address field of the MAC datagram is dependent on the Frame Type and address mode specified in the Frame Control word. Here is a mapping of all possible addressing modes, courtesy of the ZMD44102 datasheet:



FrameControl				AddressField			
[2:0]	[6]	[11:10]	[15:14]				
Frame Type	Intra Pan	DstAddr Mode	SrcAddr Mode	Octet			
Ack 3'b010	dont care			empty			
Beacon 3'b000	dont care		16bit 2'b10	SrcPanId	SrcAddr16		
	dont care		64bit 2'b11	SrcPanId	SrcAddr64		
Data/ Cmd (default)	dont care	noPanId/ Addr 2'b00	no Id/Addr 2'b00	empty			
	intraPan 1'b1		16bit 2'b10	SrcAddr16			
	nointraPan 1'b0			SrcPanId	SrcAddr16		
	intraPan 1'b1		64bit 2'b11	SrcAddr64			
	nointraPan 1'b0			SrcPanId	SrcAddr64		
	dont care	16bit 2'b10	no Id/Addr 2'b00	DstPanId	DstAddr16		
	intraPan 1'b1		16bit 2'b10	DstPanId	DstAddr16	SrcAddr16	
	nointraPan 1'b0			DstPanId	DstAddr16	SrcPanId	SrcAddr16
	intraPan 1'b1		64bit 2'b11	DstPanId	DstAddr16	SrcAddr64	
	nointraPan 1'b0			DstPanId	DstAddr16	SrcPanId	SrcAddr64
	dont care	64bit 2'b11	no Id/Addr 2'b00	DstPanId	DstAddr64		
	intraPan 1'b1		16bit 2'b10	DstPanId	DstAddr64	SrcAddr16	
	nointraPan 1'b0			DstPanId	DstAddr64	SrcPanId	SrcAddr16
	intraPan 1'b1		64bit 2'b11	DstPanId	DstAddr64	SrcAddr64	
	nointraPan 1'b0			DstPanId	DstAddr64	SrcPanId	SrcAddr64
	default			empty			

Figure 5.5 - 802.15.4 MAC Addressing Formats

- Upon inspection of Figure 5.5, note that an ACK frame contains no addressing, a beacon frame may contain 4 to 10 bytes of addressing, and a data/cmd frame may contain 4-20 bytes of addressing.
- A beacon frame is originated by the PAN coordinator and is used to synchronize all nodes in the network. A command frame is used by frames to help create the networks shown in Figure 4.1, and will not be covered in this tutorial. Data and ACK frames will be covered extensively in the next few chapters.

6 OVERVIEW OF THE CCS WIRELESS (ZMD) DEVELOPMENT KIT

- The Wireless (ZMD) Development kit has a PIC18LF452 running at 3.3V.
- The entire board can be powered with a 9VDC power supply or with a 9V battery.
- A potentiometer is connected to RA0 (AN0), a pushbutton is connected to RA4, three LEDs are connected to RB4, RB5 and RA5 and an I/O header is connected for access to the other pins of the PIC18LF452.
- For RS232 serial I/O, an RS232 driver is connected to RC6 and RC7.
- Three 7-segment LEDs are also available, which can be controlled using RB2, RB4 and RB5 (example code will be given later for the 7-segment LEDs).
- The ZigBee™-compliant (802.15.4 PHY/MAC) radio and the ZMD44102 are being used for this tutorial book and development kit. The ZMD44102 is a one-chip-solution that provides 802.15.4 PHY/MAC support in the 800-900MHz frequency bands (see Figure 5.1). When sleeping, this radio only takes a few micro-amps, ideal for low-power or battery powered embedded systems. A good antenna can achieve distances of over 350 feet.
- This kit is intended to develop a 802.15.4 Compatable Communication Systems. Creating a ZigBee™ network will not be covered with this tutorial.

7 THE ZMD44102 RADIO

- ❑ The ZMD44102 radio is a ZigBee™-compliant, 802.15.4 PHY/MAC low-power transceiver. Even if ZigBee™ is not going to be used, it is a great low-power radio transceiver to use in any embedded project. The ZMD44102 can be interfaced using SPI or a parallel interface.
- ❑ CCS provides a library for controlling the radio, as well as several higher level MAC functions. Below is an example that generates a carrier wave on channel 6.
- ❑ Type in the following program, **Compile and Run**:

```
#include <18F452.h>
#fuses HS,NOWDT,NOLVP
#use delay(clock=10000000)

#include <ZMD44102.h>

void zmd_generate_cw(int8 channel, int8 modulated)
{
   if (modulated)
      ZPhy_SetTX(ZPHY_TX_MODE_MODULATED_CARRIER, ZPHY_TX_POWER_MAX);
   else
      ZPhy_SetTX(ZPHY_TX_MODE_CARRIER, ZPHY_TX_POWER_MAX);

   ZMac_SetTX(ZMAC_TX_MODE_DIRECT, ZMAC_TX_SLOTTED_DISABLE);
   ZPhy_SetChannel(channel);
   zmd_set_maccontrol(ZMD_mc_TxOn);
}


void main(void) {
   zmd_init();

   zmd_generate_cw(6, FALSE);

   while(TRUE);
}
```

THE ZMD44102 RADIO (CONT.)

- ❑ While this example is running, connect a spectrum analyzer to the antenna of the prototyping board. A large spike at 916MHz means the ZMD44102 is properly generating a carrier wave on channel 6:

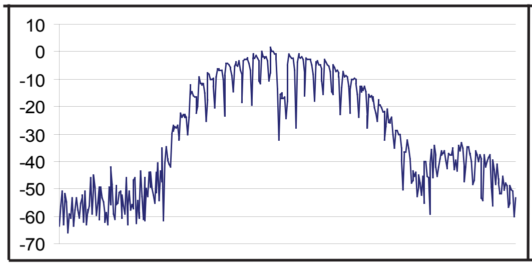


Figure 7.1 - Modulated Carrier

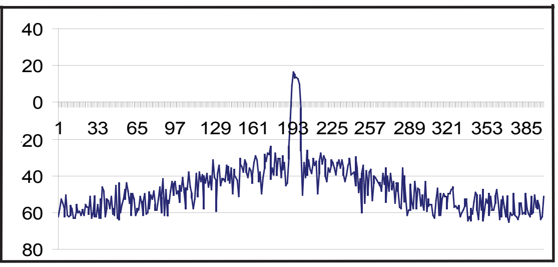


Figure 7.2 - Unmodulated Carrier

- ❑ The ZMD44102 supports the 11 channels in the 800/900MHz frequency band (see Figure 5.1). The actual frequency for each channel can be found in the following chart:

Channel	Frequency (MHz)
0	868.3
1	906
2	908
3	910
4	912
5	914
6	916
7	918
8	920
9	922
10	924

Figure 7.2 – ZMD44102 Frequencies Supported

NOTES

- If zmd_generate_cw(6, TRUE) is called, a modulated carrier wave is sent. Inspect the DSSS and BPSK modulation pattern used. If you inspect 916MHzwith a spectrum analyzer you should see Figure 7.1
- zmd_init() will initiate the ZMD interface and go into an idle state
- ZPhy_SetTX(mode, power) configures the physical layer of the ZMD44102 radio. Normally this function will not need to be called, as the default values will suffice for most applications. For possible values of power, see ZPHY_TX_POWER in ZMD44102.H.
- ZMac_SetTx(mode, slotted) configures the 802.15.4 MAC operation of the ZMD44102 radio. This function does not normally need to be called, the default values will suffice for most applications. The default mode is usually CSMA, but for generating a carrier wave it is best to turn off the CSMA mechanism. Slotted mode should only be enabled for networks, since this tutorial will not do any 802.14.4 networks, always leave slotted mode disabled.
- ZPhy_SetChannel(channel) configures the RF channel the ZMD44102 radio will operate on. You can get your current channel by using Zphy_GetChannel().
- zmd_set_maccontrol() puts the ZMD44102 into the specified operational mode. Normally the application will never call this function, instead the ZMD44102.H library will set the unit into the appropriate modes needed. See the official ZMD44102 documentation for a list of valid operational modes.

8 SENDING A SIMPLE MESSAGE

- ❑ Before continuing, copy the first 5 lines of this program and save them into a new file called **zmd.h**. Save zmd.h in the same directory as the examples of this tutorial.

- ❑ In this chapter the MAC functions in ZMD44102.C will be used to send a 16-bit message. To keep the example simple, 16-bit addressing will be used. Since this is not a network device, multi-network PAN will not be necessary.

- ❑ Looking at Figure 5.5, this addressing mode can be achieved with IntraPan=1, Destination Addressing Mode=0b10 and Source Addressing Mode=0b10. The security enabled bit of the frame control should be left clear as it will not be used. A frame will not be pending, so the frame pending bit of the frame control should be left clear. The acknowledge requested bit will be set to signify to the receiving node that we want an acknowledgement.

Bytes	2	1	2	2	2	2	1
MAC Layer	Frame Control (0x8863)	Sequence Number	PanId (0x8000)	DstAddr16 (0x0002)	SrcAddr16 (0x0001)	Data (MSDU) (0x1234)	FCS

Bytes	4	1	1	
PHY Layer	Preamble	SOF	Frame Length (14)	DATA (See MAC Layer above)

Figure 8.1 – Example MAC Packet

- ❑ With these specifications, concatenate Figures 5.2, 5.3, 5.4 and 5.5 into a complete frame (assume the PanId is 0x8000, DstAddr16 is 0x0002, SrcAddr is 0x0001 and the data is 0x1234):

- ❑ The ZMD44102 has a TX FIFO which can be used in two ways. The standard method uses the TX FIFO to hold the MSDU, and the ZMD44102 will generate the rest of the MAC header depending on the contents of several registers (such as mhrFc1Tx, mhrDstAddr16, etc). The second method requires the user to generate a MAC header and stuffs it into the TX FIFO. The transmit FIFO for the 44102 is 128 bytes.

- ❑ Once the transmit FIFO is full (and if using the first method, all the header registers are set), transmission is initiated by setting the operational control register (macControl) to TxOn (0x03). If CSMA is enabled, the ZMD44102 will first listen to the traffic channel for a short while to determine if the channel is busy. If the channel is busy it will set the macTxStatus register to denote a CSMA error and put the radio back into Idle mode. If CSMA was disabled or if there was no traffic on the channel it will transmit the frame and set the macTxStatus register to success. If an ACK was requested it will then put the radio into RX state and wait for an ACK frame for a short time. If an ACK was received in the time period, macRxStatus will be set to Ack, else it will be set to AckTimeOut.

SENDING A SIMPLE MESSAGE (CONT.)

❑ Add to **zmd.h**

```
#define BUTTON_PRESSED()    (!input(PIN_A4))
#define PIN_LED1  PIN_B4
#define PIN_LED2  PIN_B5
#define PIN_LED3  PIN_A5
#define LED_ON    output_low
#define LED_OFF   output_high

#define EXP_OUT_ENABLE  PIN_B2
#define EXP_OUT_CLOCK   PIN_B4
#define EXP_OUT_DO      PIN_B5
#define NUMBER_OF_74595 3
#include <74595.c>

const char digit_format[10]={
0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90
};

void leds_off(void) {
   LED_OFF(PIN_LED1);
   LED_OFF(PIN_LED2);
   LED_OFF(PIN_LED3);
}

void lcd_clear(void) {
   int8 digits[3]={0xFF,0xFF,0xFF};
   write_expanded_outputs(&digits[0]);
   output_low(EXP_OUT_ENABLE);
   leds_off();
}

(continued...)
```

```
(continued...)

void lcd_putd(int16 num) {
   int8 digits[3];
   if (num>999) {num=999;}

   digits[0]=num / 100;
   digits[1]=(num % 100) / 10;
   digits[2]=num % 10;

   digits[0]=digit_format[digits[0]];
   digits[1]=digit_format[digits[1]];
   digits[2]=digit_format[digits[2]];

   write_expanded_outputs(&digits[0]);
   output_low(EXP_OUT_ENABLE);

   LED_OFF(PIN_LED1);
   LED_OFF(PIN_LED2);
}

void myZMacInit(void)
{
   ZMac_Init();

   ZMac_SetMyShortAddress(MyShortAddress);
   ZMac_SetMyPanId(MyPanId);
   ZMac_UseMyShortAddress();
   ZPhy_SetChannel(MyRFChannel);
}
```

SENDING A SIMPLE MESSAGE (CONT.)

- ❑ Enter the following code into a new file called EX8.C
- ❑ Compile and run the program.

```
#define USE_READ_BUFFER
#include <USBMaster.h>
#define _bootloader
#include <VNC1L_bootloader.h>

#ORG default

void main(){
   char c;
   char filename[STRING_SIZE];

   USBMasterInit();

   printf("Application Ver 1.0");

   while(TRUE){
      if(!input(PUSH_BUTTON2)){
         fprintf(USER,"\n\rEntering bootloader...) ");
         #asm
         goto LOADER_ADDR;
         #endasm
      }
      if(kbhit(USER)){
         c=fgetc(USER);
         USBSerialTask(c);
      }
      USBMasterCallback();
   }
}
```

- ❑ This example will send a 16-bit value over the 802.15.4 MAC layer, and display the success rate at which the message was sent. It will return an error code, where a non-zero is an error. Until there is another unit acting as a receiver this example will always return a NoACK error, so the success rate displayed will always be 000%.

NOTES

- ZMac_Init() initializes the ZMD44102 radio, and prepares some of the higher level functions for sending and receiving packets. It will also set the unit's short address and PAN ID to 0xFFFF.
- ZMac_SetMyShortAddress(address) configures the receive filter on the ZMD44102 radio to accept this short (16-bit) address. It also prepares the transmit functions to use this address as the source short address.
- Although not shown, ZMac_SetMyLongAddress(*address) is also available to set the long (64-bit) address of this unit. The receive filter on the radio will accept both short and long addresses.
- According to 802.15.4, all units have a unique long address (basically a MAC address) while the short addresses are dynamically configured by the network coordinator when the node associates with the network. This tutorial will not be using a network in these examples, so statically assigning short addresses and PAN IDs is needed.
- Shorter packets have a higher chance of transmit success. This is why 802.15.4 has the ability to dynamically assign shorter addresses instead of using the entire 8-byte MAC address.
- ZMac_UseMyShortAddress() tells the MAC transmit code to use the already configured short address as the source address in future outgoing packets. ZMac_UseMyLongAddress() is also provided, this will configure the transmit code to use the long address.
- ZMac_PutPacket(*remoteNode, *data, count) will transmit count bytes of data to the remoteNode. It will return 0 if OK. It will return non-zero if error (no acknowledge, CSMA failed, or transmitter was not ready). With each ZMac_PutPacket() the 802.15.4 sequence number is incremented.
- If ZMac_PutPacket() in ZMD44102.H is inspected, notice the lower level functions that it uses, such as: ZMac_PutHeader(), ZMac_Putc(), ZMac_Putd() and ZMac_Flush(). Writing one's own ZMac_PutPacket() may be useful if implementing retries or to not increment the sequence number automatically.
- ZMac_PutPacket() will use the transmit settings that have already been configured by ZPhy_SetChannel(), ZPhy_SetTX() and ZMac_SetTX(). See the previous chapter for more information about these functions.
- remoteNode is a ZMAC_NODE_INFO struct, and contains the long address, short address and PAN ID of the node to communicate to. Although the struct can hold both the short and long address (remember, a unit can have both addresses), one must use the addressMode field to specify which address to use in the packet.

9 RECEIVING A SIMPLE MESSAGE

- ❑ The previous chapter created a unit that was transmitting messages to no one. This chapter will receive the message and display the results.
- ❑ Type the following code into a new file and save as EX9.C.
- ❑ Compile and run on the second prototype board in the kit:

```
#define MyPanId            1
#define MyShortAddress     2
#define MyRFChannel        6
#define DestShortAddress   1

#include "zmd.h"

void main(void)
{
   ZMAC_NODE_INFO remoteNode;
   int16 data, count;

   myZMacInit();

   lcd_clear();

   while(TRUE)
   {
      count=ZMac_GetPacket(&remoteNode, &data, sizeof(data), 200);
      if (count)
         lcd_putd(data);
   }
}
```

- ❑ Once running, it will display the number being transmitted by the other node on the 7 segment LED display.

NOTES

- ZMac_GetPacket(*remoteNode, *ptr, max, t) will turn on the receiver for t milliseconds and wait for an incoming packet. Once a packet is received, and the packet's destination address matches our address, remoteNode is updated with the source node and ptr is written with the packet up to max bytes. ZMac_GetPacket() will return the number of bytes in the payload, or zero if there was no packet within t milliseconds. ZMac_GetPacket() will turn off the receiver once a packet is received, regardless if t milliseconds has passed.
- If the ZMac_GetPacket() in ZMD44102.H, is inspected, note lower level API routines used, such as: ZMac_StartRX(), ZMac_GetHeader(), ZMac_Getc(). These lower level API routines will be used in the next chapter to build a simple packet sniffer. Use these lower level API routines if more control is needed.

10 PACKET SNIFFER

- ❑ A packet sniffer will view all packets transmitted and display them to the user. This is extremely helpful in debugging applications, to determine which node or unit is not behaving properly. The ZMD44102 can be put into a promiscuous mode where it will accept all packets that pass CRC. This chapter will create a simple packet sniffer application which can be used for debugging applications later.
- ❑ Add the following code to the bottom of **zmd.h**:

```
#use rs232(baud=9600, xmit=PIN_C6,rcv=PIN_C7)

void ZMac_DisplayHeader(ZMAC_HEADER *header)
{
   ZMAC_HEADER hdr;
   int8 mhr2;

   memcpy(&hdr, header, sizeof(ZMAC_HEADER));

   mhr2 = hdr.destNode.addressMode;
   mhr2 |= (hdr.sourceNode.addressMode << 4);

   printf("MHR=%X%X SQ=%X ", (int8)hdr.frameCon, mhr2, hdr.seq);

   if (hdr.frameCon.frameType == ZMAC_FCON_TYPE_DATA)
      printf("DATA ");
   else if (hdr.frameCon.frameType == ZMAC_FCON_TYPE_CMD)
      printf("CMD ");
   else if (hdr.frameCon.frameType == ZMAC_FCON_TYPE_ACK)
   {
      printf("ACK ");
      return;
   }
   else
      printf("BEACON ");

   if (hdr.destNode.addressMode)
   {
      printf("DPAN=%LX DADR=", hdr.destNode.panId.w);
      if (hdr.destNode.addressMode == ZMAC_ADDRESS_MODE_SHORT)
      {
         printf("%LX", hdr.destNode.shortAddress.w);

(continued...)
(continued...)

      }
      else
      {
         printf("%X%X%X%X%X%X%X%X",
         hdr.destNode.longAddress.b[0],
         hdr.destNode.longAddress.b[1],
         hdr.destNode.longAddress.b[2],
         hdr.destNode.longAddress.b[3],
         hdr.destNode.longAddress.b[4],
         hdr.destNode.longAddress.b[5],
         hdr.destNode.longAddress.b[6],
         hdr.destNode.longAddress.b[7]);
      }
      printf(" ");
   }
   if (hdr.sourceNode.addressMode)
   {
      if (!hdr.frameCon.intraPan)
      {
         printf("SPAN=%LX ", hdr.sourceNode.panId.w);
      }
      printf("SADR=");
      if (hdr.destNode.addressMode == ZMAC_ADDRESS_MODE_SHORT)
      {
         printf("%LX", hdr.sourceNode.shortAddress.w);
      }
      else
      {
         printf("%X%X%X%X%X%X%X%X",
         hdr.sourceNode.longAddress.b[0],
         hdr.sourceNode.longAddress.b[1],
         hdr.sourceNode.longAddress.b[2],
         hdr.sourceNode.longAddress.b[3],
         hdr.sourceNode.longAddress.b[4],
         hdr.sourceNode.longAddress.b[5],
         hdr.sourceNode.longAddress.b[6],
         hdr.sourceNode.longAddress.b[7]);
      }
      printf(" ");
   }
}
```

10 PACKET SNIFFER (CONT.)

❑ Type in the following program, save to ex10.c, compile and run on one prototyping board:

```
#define MyPanId            1
#define MyShortAddress     2
#define MyRFChannel        6

#include “zmd.h”

void main(void)
{
   int16 count=0;
   int8 len;
   ZMAC_HEADER hdr;

   myZMacInit();

   zmd_storeLQI(TRUE);
   ZMac_StartPromiscous();
   ZMac_StartListen();

   lcd_putd(0);

   while(TRUE) {
      restart_wdt();
      if (ZMac_IsRXReady())
      {
         count++;
         if (count>999) {count=0;}
         lcd_putd(count);

         len=ZMac_GetHeader(&hdr);

         printf(“\r\n * “);
         ZMac_DisplayHeader(&hdr);

         printf(“ DLEN=%U “,len);
         while(len--) {
            printf(“%X”,ZMac_Getc());
         }

         ZMac_DiscardRX();
         printf(“ LQI=%LX”, zmd_getLQI());
      }
   }
}
```

❑ While the packet sniffer is running on one board, load the transmit example from Chapter 8 on the second board. While both boards are running, inspect the serial output from the board running EX10, and note the following:

```
* MHR=6188 SQ=80 DATA DPAN=0001 DADR=0002 SADR=0001 DLEN=2 0000 LQI=043B
* MHR=6188 SQ=81 DATA DPAN=0001 DADR=0002 SADR=0001 DLEN=2 0100 LQI=0461
* MHR=6188 SQ=82 DATA DPAN=0001 DADR=0002 SADR=0001 DLEN=2 0200 LQI=03D5
* MHR=6188 SQ=83 DATA DPAN=0001 DADR=0002 SADR=0001 DLEN=2 0300 LQI=0458
* MHR=6188 SQ=84 DATA DPAN=0001 DADR=0002 SADR=0001 DLEN=2 0400 LQI=0463
```

❑ The entire packet is shown, including the 16-bit frame control value, the sequence number, destination and source address. One of the trickier aspects of 802.15.4 is that the addressing mode is dynamic, but this packet sniffer will display the proper addresses in the packet. The ZMD44102 also appends a link quality indicator the end of each packet, and it is displayed as well.

PACKET SNIFFER (CONT.)

NOTES

- This example uses the lower level MAC API, whereas the previous chapter used a canned get packet routine. This example will be beneficial to those who need more control than the canned get packet routine gives you.
- Although Source and Destination addresses are #defined at the top of the code, they are not used since the radio is put into a promiscuous mode.
- ZMac_StartListen() turns on the receiver of the ZMD44102. It will stay on indefinitely, until ZMac_StopListen() turns it off. If sending a packet, it will switch back to RX mode after the packet is completed.
- ZMac_StartPromiscous() disables the address filtering on the ZMD44102 so it will accept all packets, as long as the packet has a good CRC. ZMac_StopPromiscous() will put the unit back into normal mode.
- ZMac_IsRXReady() returns TRUE if there is a packet in the receive buffer that needs to be handled.
- ZMac_GetHeader(*header) reads the 802.15.4 header and saves it to the header pointer. It will also return the number of bytes of data left to be received (the payload). Do not call this unless ZMac_IsRxReady() returns TRUE.
- ZMac_DisplayHeader(*header) is a special routine used only in this example – it is not included in the API.
- ZMac_Getc() reads and returns the next character in the receive buffer. ZMac_Getd(*ptr, count) will read count bytes from the receive buffer and store to ptr.
- ZMac_DiscardRX() will discard any remaining data left in the receive buffer, and prepare the radio's receive buffer for the next packet. It must be called after each packet is received.
- zmd_storeLQI() configures the ZMD44102 radio to append a link quality indicator to the end of each packet. Normally this is disabled, but for packet sniffing and debugging it may be useful to turn it on so this example enables it.
- If LQI is enabled, use zmd_getLQI() to read the LQI from the end of the packet. Call this after ZMac_DiscardRX(), and only if LQI is enabled. Normally it is disabled.

11 BI-DIRECTIONAL COMMUNICATION

❑ In the previous chapters one board transmitted and the other board received. This chapter will create an application that both sends and receives data.

❑ Add the following code to the bottom of **zmd.h**:

```
ZMAC_NODE_INFO txNode;

void myDynamicZMacInit(void)
{
   ZMac_Init();

   ZMac_SetMyPanId(MyPanId);
   ZMac_UseMyShortAddress();

   txNode.panId=MyPanId;
   txNode.addressMode=ZMAC_ADDRESS_MODE_SHORT;

   if (BUTTON_PRESSED())
   {
      ZMac_SetMyShortAddress(DestShortAddress);
      txNode.shortAddress=MyShortAddress;
   }
   else
   {
      ZMac_SetMyShortAddress(MyShortAddress);
      txNode.shortAddress=DestShortAddress;
   }
   ZPhy_SetChannel(MyRFChannel);
}
```

❑ Type in the following code into a new file called EX11.C, compile and run on both prototyping boards:

```
#define MyPanId             1
#define MyShortAddress      1
#define DestShortAddress    2
#define MyRFChannel         6

#include "zmd.h"

(continued...)
```

```
(continued...)

void main(void) {
   ZMAC_HEADER header;
   int to=250;
   struct
   {
      int button;
      int analog;
   } packet;
   int8 count;

   myDynamicZMacInit();

   lcd_clear();

   setup_adc_ports(AN0);
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);

   ZMac_StartListen();

   while(TRUE)
   {
      if (ZMac_IsRxReady())
      {
         count=ZMac_GetHeader(&header);
         ZMac_Getd(&packet,2);

         lcd_putd(packet.analog);

         if (packet.button)
            LED_ON(PIN_LED3);
         else
            LED_OFF(PIN_LED3);

         ZMac_DiscardRX();
      }

      if (!to)
      {

(continued...)
(continued...)

        packet.button=BUTTON_PRESSED();
        packet.analog=read_adc();

        if (!ZMac_PutPacket(&txNode,&packet,2))
           to=250;
     }
     else
       to--;

     delay_ms(1);
   }
}
```

- ❑ In order to prevent both nodes from having the same address, myDynamicZMacInit() will load a different address if the button (A4) is held down during reset. Therefore on one of the boards, hold down the button while pressing the reset button.

- ❑ While the code is running, it will transmit the A/D conversion and button state every 250 milliseconds. If another packet is received, it will display the other node's A/D conversion on the 7-segment LED display and the current button state (lit=pressed, unlit=not pressed) on the red LED (A5).

NOTES

- The API used for receiving is the same as discussed in Chapter 10
- Using a ZMac_PutPacket() will only temporarily disable the receiver, because ZMac_StartListen() will leave it in RX mode until a ZMac_StopListen() is used.

12 WIRELESS RS-R32

❑ CCS provides a driver that can use an 802.15.4 PHY/MAC layer to create a virtual RS232 serial link between two devices. The method used is not complex, and if a standard, high bandwidth RS232 link over wireless is needed, Bluetooth is more applicable.

❑ Enter the follow code into EX12.C, compile and run on both development boards:

```
#define MyPanId                 1
#define MyShortAddress          1
#define DestShortAddress        2
#define MyRFChannel             6

#include "zmd.h"
#include <ZRS232.h>
void main(void) {
   myDynamicZMacInit();
   ZRS232Init();
   ZRS232Connect(txNode.shortAddress.w);

   lcd_clear();

   while(TRUE)
   {
      ZRS232Task();
      while (ZRS232kbhit())
      {
         putc(ZRS232getc());
      }
      while(kbhit())
      {
         ZRS232putc(getc());
      }
   }
}
```

❑ Similar to Chapter 11, this example uses myDynamicZMacInit() to configure the unit's address depending on the button state during reset. Therefore, when powering up one of the units, hold down the button (A4) to make sure it has a different address than the other unit.

❑ Connect one unit to a PC using a serial PC to prototype board cable (see item 3 on the diagram in Chapter 2), which for this example will be called Node A. Connect the other unit to another PC using a serial PC to protoype board cable, which for this example will be called Node B. On both PCs open the COMM port the serial PC to prototype board cable is connected to using a serial terminal program, the CCS SIOW or Microsoft Hyperterminal will suffice. On both PCs, configure the serial baud rate to 9600, 8 data bits, 1 stop bit and no parity.

❑ While this example is running, pressing a key on Node A's PC will result in the key being shown on Node B's PC, and vice-versa. This demonstrates creating a simple RS-232 link without wires.

NOTES

- ZRS232Init() initializes the 802.15.4 library for use. It should be called before any other ZRS232 functions.
- ZRS232Connect(address) configures the ZRS232 to send and receive packets to only this node. This library only uses short addresses. ZRS232Connect() also turns on the receiver.
- ZRS232Task() checks the 802.15.4 receive buffer for incoming characters. It also checks the transmit buffer to see if any characters need to be sent.
- If there is data in the receive buffer when ZRS232Task() is called, it will be discarded to make room for any future packets. Therefore, any data in the receive buffer must be dealt with before the next ZRS232Task().
- If there is data in the transmit buffer when ZRS232Task() is called, it will attempt to transmit the packet. If there was no collision detection and an ack was detected, then the transmit buffer is cleared to make space for new data. If there was a collision detection or no ack it will retry later, but if there are too many retries it will eventually discard the data.
- ZRS232kbhit() returns TRUE if there are any characters in the ZRS232 receive buffer.
- ZRS232getc() returns the next character in the ZRS232 receive buffer. If calling this when ZRS232kbhit() returns FALSE, the result of ZRS232getc() will be invalid.
- ZRS232putc(character) puts the character into a transmit buffer. It will return TRUE if there was space in the transmit buffer for this character.
- Although not shown, ZRS232Flush() will force the transmit the data in the transmit buffer. It will return TRUE if the data was sent successfully (no collision and an ACK was received). It will retry many times, and the process can take several hundred milliseconds. This is useful if one wants to send lots of data in between ZRS232Task()s.

Other Development Tools

EMULATORS

The ICD used in this booklet uses two I/O pins on the chip to communicate with a small debug program in the chip. This is a basic debug tool that takes up some of the chip's resources (I/O pins and memory). An emulator replaces the chip with a special connector that connects to a unit that emulates the chip. The debugging works in a simulator manner except that the chip has all of its normal resources, the debugger runs faster and there are more debug features. For example an emulator typically will allow any number of breakpoints. Some of the emulators can break on an external event like some signal on the target board changing. Some emulators can break on an external event like some that were executed before a breakpoint was reached. Emulators cost between $500 and $3000 depending on the chips they cover and the features.

DEVICE PROGRAMMERS

The ICD can be used to program FLASH chips as was done in these exercises. A stand alone device programmer may be used to program all the chips. These programmers will use the .HEX file output from the compiler to do the programming. Many standard EEPROM programmers do know how to program the Microchip parts. There are a large number of Microchip only device programmers in the $100-$200 price range. Note that some chips can be programmed once (OTP) and some parts need to be erased under a UV light before they can be re-programmed (Windowed). CCS offers the Mach X which is a stand-alone programmer and can be used as an in-circuit debugger.

PROTOTYPING BOARDS

There are a large number of Prototyping boards available from a number of sources. Some have an ICD interface and others simply have a socket for a chip that is externally programmed. Some boards have some advanced functionality on the board to help design complex software. For example, CCS has a Prototyping board with a full 56K modem on board and a TCP/IP stack chip ready to run internet applications such as an e-mail sending program or a mini web server. Another Prototyping board from CCS has a USB interface chip, making it easy to start developing USB application programs.

SIMULATORS

A simulator is a program that runs on the PC and pretends to be a microcontroller chip. A simulator offers all the normal debug capability such as single stepping and looking at variables, however there is no interaction with real hardware. This works well if you want to test a math function but not so good if you want to test an interface to another chip. With the availability of low cost tools, such as the ICD in this kit, there is less interest in simulators. Microchip offers a free simulator that can be downloaded from their web site. Some other vendors offer simulators as a part of their development packages.

References

Figure 5.5 - ZMD44102 Datasheet
http://www.zmd.de/zigbee.php?content=zig&product=zmd44102.

On The Web

Comprehensive list of PIC® MCU Development tools and information	www.mcuspace.com
Microchip Home Page	www.microchip.com
CCS Compiler/Tools Home Page	www.ccsinfo.com
CCS Compiler/Tools Software Update Page	www.ccsinfo.com click: Support → Downloads
C Compiler User Message Exchange	www.ccsinfo.com/forum
Device Datasheets List	www.ccsinfo.com click: Support → Device Datasheets
C Compiler Technical Support	support@ccsinfo.com

CCS Programmer Control Software

The CCSLOAD software will work for all the CCS device programmers and replaces the older ICD.EXE and MACHX.EXE software. The CCSLOAD software is stand-alone and does not require any other software on the PC. CCSLOAD supports ICD-Sxx, ICD-Uxx, Mach X, Load-n-Go, and PRIME8.

Powerful Command Line Options in Windows and Linux

· Specify operational settings at the execution level
· Set-up software to perform, tasks like save, set target Vdd
· Preset with operational or control settings for user

Easy to use Production Interface

· Simply point, click and program
· Additions to HEX file organization include associating comments or a graphic image to a file to better ensure proper file selection for programming
· Hands-Free mode auto programs each time a new target is connected to the programmer
· PC audio cues indicate success and fail

Extensive Diagnostics

· Each target pin connection can be individually tested
· Programming and debugging is tested with known good programs
· Various PC driver tests to identify specific driver installation problems

Enhanced Security Options

· Erase chips that failed programming
· Verify protected code cannot be read after programming
· File wide CRC checking

Automatic Serial Numbering Options

· Program memory or Data EEPROM
· Incremented, from a file list or by user prompt
· Binary, ASCII string or UNICODE string

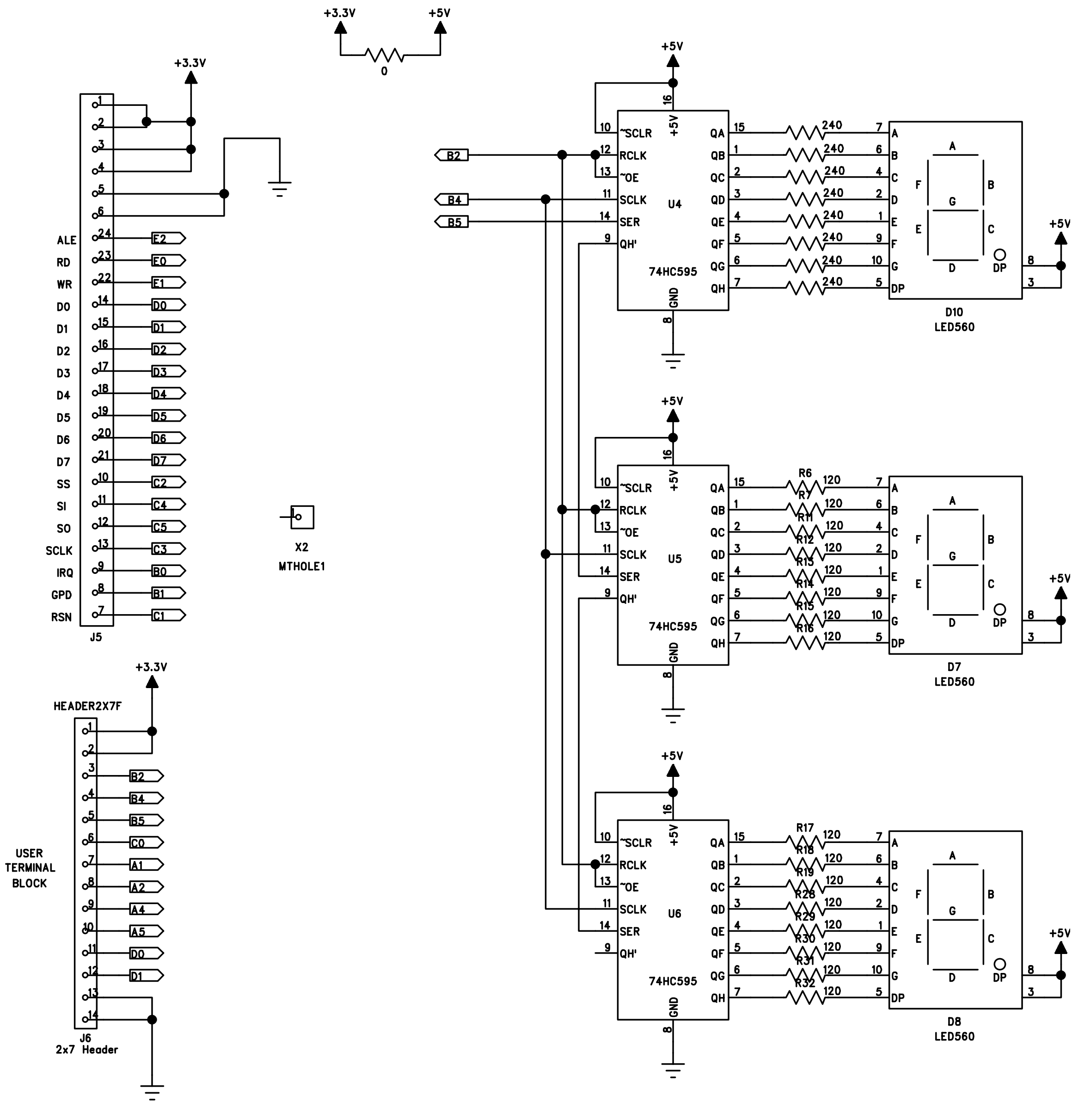
CCS IDE owners can use the CCSLOAD program with:

· MPLAB®ICD 2/ICD 3
· MPLAB®REAL ICE™
· All CCS programmers and debuggers

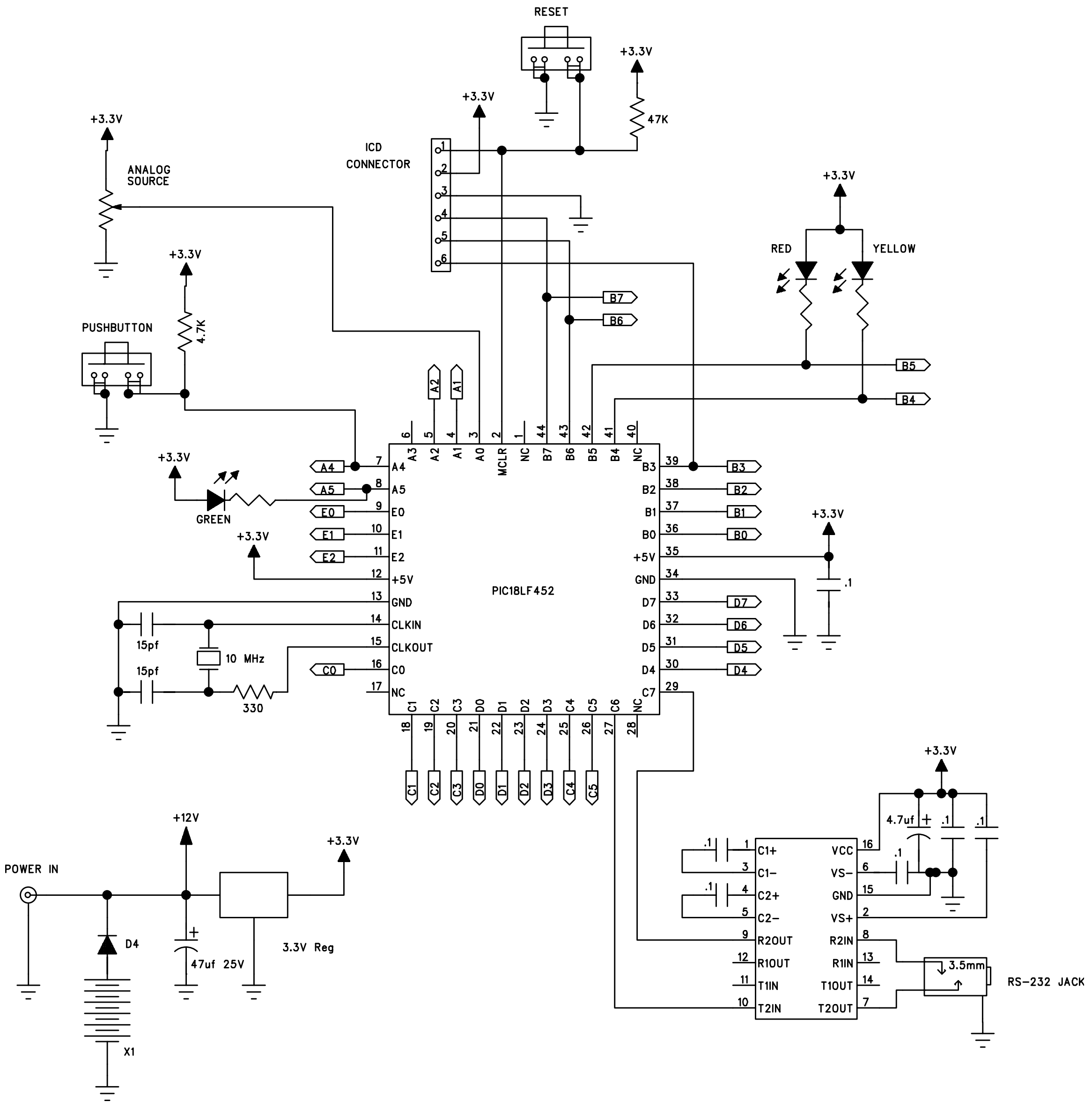
How to Get Started:

Step 1: *Connect Programmer to PC and target board. Software will auto-detect the programmer and device.*
Step 2: *Select Hex File for target board.*
Step 3: *Select Test Target. Status bar will show current progress of the operation.*
Step 4: *Click "Write to Chip" to program the device.*

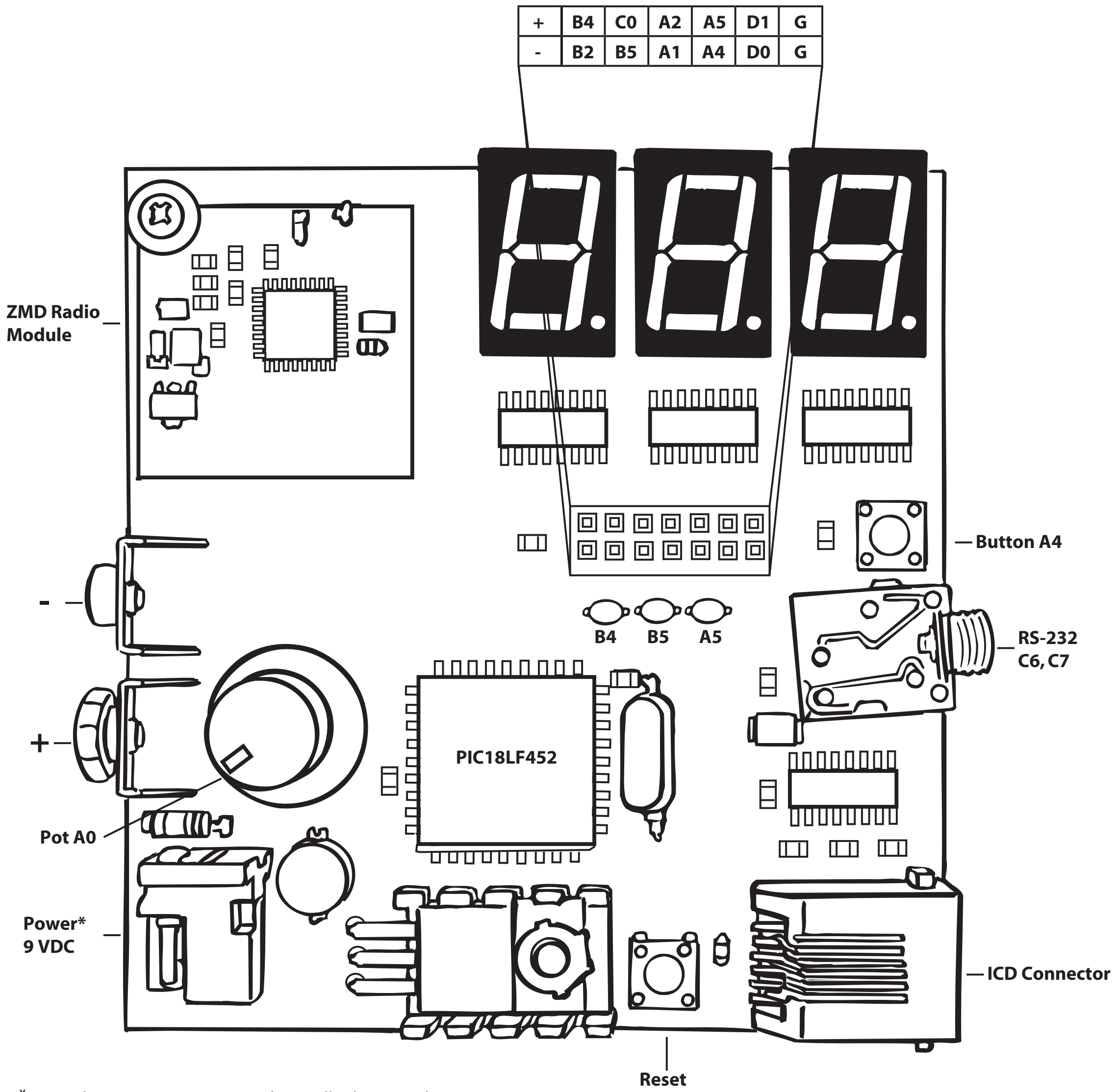
Use the Diagnostics tab for troubleshooting or the ccsload.chm help file for additional assistance.



+3.3V
+5V
0
J5
ALE
RD
WR
D0
D1
D2
D3
D4
D5
D6
D7
SS
SI
SO
SCLK
IRQ
GPD
RSN
E2
E0
E1
C2
C4
C5
C3
B0
B1
C1
X2
MTHOLE1
HEADER2X7F
USER TERMINAL BLOCK
J6
2x7 Header
B2
B4
B5
C0
A1
A2
A4
A5
U4
U5
U6
74HC595
~SCLR
RCLK
~OE
SCLK
SER
QH'
QA
QB
QC
QD
QE
QF
QG
QH
GND
240
120
R6
R7
R11
R12
R13
R14
R15
R16
R17
R18
R19
R28
R29
R30
R31
R32
D10
D7
D8
LED560
DP



RESET
+3.3V
47K
ICD CONNECTOR
ANALOG SOURCE
PUSHBUTTON
4.7K
RED
YELLOW
B7
B6
B5
B4
A2
A1
A4
A5
E0
E1
E2
GREEN
PIC18LF452
MCLR
NC
+5V
GND
CLKIN
CLKOUT
C0
B3
B2
B1
B0
D7
D6
D5
D4
C7
C1
C2
C3
D0
D1
D2
D3
C4
C5
C6
15pf
10 MHz
330
.1
4.7uf
C1+
C1−
C2+
C2−
R2OUT
R1OUT
T1IN
T2IN
VCC
VS−
VS+
R2IN
R1IN
T1OUT
T2OUT
3.5mm
RS-232 JACK
POWER IN
+12V
D4
47uf 25V
3.3V Reg
X1



*Use only one power source, either wall adapter or battery.