LiteOS User's Guide

Version 2.1
Last updated: Oct 2 2011

This guide illustrates how to get started as well as an In-depth description of the LiteOS operating system.

Download location: www.liteos.net

Contents

Preface

LiteOS provides a UNIX-like environment for sensor networks, networked embedded devices, and cyber physical systems. It provides a thread-based run-time execution environment for applications. The goal of this User's Guide is to get you familiarized with its environment.

For updates, check:

www.liteos.net

References

The following are additional documentation, available on www.liteos.net

- LiteOS Kernel Developer's Guide

Installation of LiteOS

This section provides a step-by-step guide on how to install LiteOS on your computer.

Following are the hardware requirements for constructing the working environment of LiteOS (updated):

- **MicaZ or IRIS nodes with Atmega128 or Atmega1281 processor, at least three nodes are required, one as base station, the other two as experiment nodes, four or more suggested.**
- **MIB510 Programming Board or MIB520 Programming board (at least one of them)**
- **PC installed with Microsoft Windows Vista or 7 (XP is no longer recommended due to driver complications. If you use XP, then you have to install additional drivers according to this guide. Note that if you use Mac or Linux, you may have to search for drivers for yourself, and the AVR Studio does not currently support these two operating systems)**
- **Serial or USB port (at least one port on the PC)**
- **USB hub (for expanding the number of USB ports)**
- **AVR Jtag MkII debugger (strongly recommended, part of this guide will assume that you have AVR Jtag MKII debugger available)**
- **CC2420DK (for debugging purposes, strongly recommended)**

Installing LiteOS on Windows machine

The first step to configure your PC is to install required third-party software. The following software is needed:

�Java JDK 1.6 or later
�Cygwin (must installed to use the interactive shell)

You may skip this section if your PC is already installed with JDK and Cygwin.

We next explain how to install various third party software packages and configure them.

Installing JDK

If you have installed JDK 6 or later on your computer, you may skip this section.

Otherwise, click
http://www.oracle.com/technetwork/java/javase/downloads/index.html

to download the latest version of JDK. As of the time this document is being written, this version is JDK 7.

Click to download Java SDK, and install it on your computer, following its instructions.

It is strongly suggested that you choose the x86 (32-bit) of Java to install the serial port support later.

Now follow the steps in the LiteOS download page on liteos.net, especially step 4, to install the java serial port support.

Installing Cygwin

Download Cygwin setup from

http://www.cygwin.com/

Install Cygwin following its instructions. Note that it may take a while to download all Cygwin packages from the Internet, so choose an Internet mirror for Cygwin that is near your location.

Check that Cygwin and Java have been correctly configured

Open the Cygwin window, and type the following:

```
$ java –version

java version "1.6.0_07"

Java(TM) SE Runtime Environment (build 1.6.0_07-b06)

Java HotSpot(TM) Client VM (build 10.0-b23, mixed mode, sharing)
```

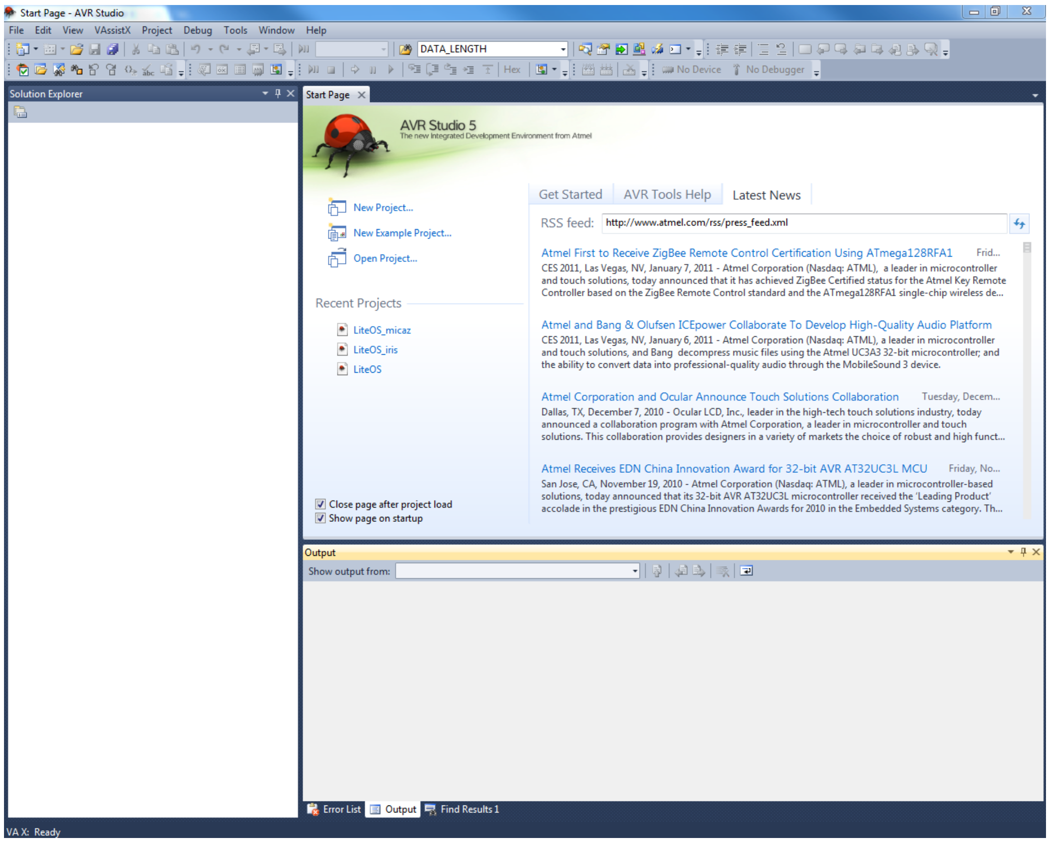
If you do not see a Java version that matches your configuration, then you probably have not installed JDK on your machine correctly, or have not set up the environment variables correctly.

Install the AVR Studio 5.0

Download the latest version of AVR studio 5.0, from the following link:

http://www.atmel.com/microsite/avr_studio_5/default.asp?source=redirect

Follow the instructions on installation of AVR Studio 5.0. Finally you should see some screenshot as following:



RSS feed: http://www.atmel.com/rss/press_feed.xml

Atmel First to Receive ZigBee Remote Control Certification Using ATmega128RFA1 Frid...
CES 2011, Las Vegas, NV, January 7, 2011 - Atmel Corporation (Nasdaq: ATML), a leader in microcontroller and touch solutions, today announced that it has achieved ZigBee Certified status for the Atmel Key Remote Controller based on the ZigBee Remote Control standard and the ATmega128RFA1 single-chip wireless de...

Atmel and Bang & Olufsen ICEpower Collaborate To Develop High-Quality Audio Platform
CES 2011, Las Vegas, NV, January 6, 2011 - Atmel Corporation (Nasdaq: ATML), a leader in microcontroller and touch solutions, and Bang decompress music files using the Atmel UC3A3 32-bit microcontroller; and the ability to convert data into professional-quality audio through the MobileSound 3 device.

Atmel Corporation and Ocular Announce Touch Solutions Collaboration Tuesday, Decem...
Dallas, TX, December 7, 2010 - Ocular LCD, Inc., leader in the high-tech touch solutions industry, today announced a collaboration program with Atmel Corporation, a leader in microcontroller and touch solutions. This collaboration provides designers in a variety of markets the choice of robust and high funct...

Atmel Receives EDN China Innovation Award for 32-bit AVR AT32UC3L MCU Friday, No...
San Jose, CA, November 19, 2010 - Atmel Corporation (Nasdaq: ATML), a leader in microcontroller-based solutions, today announced that its 32-bit AVR AT32UC3L microcontroller received the 'Leading Product' accolade in the prestigious EDN China Innovation Awards for 2010 in the Embedded Systems category. Th...

This means that you have successfully installed the AVR Studio on your PC.

Get familiar with your equipment

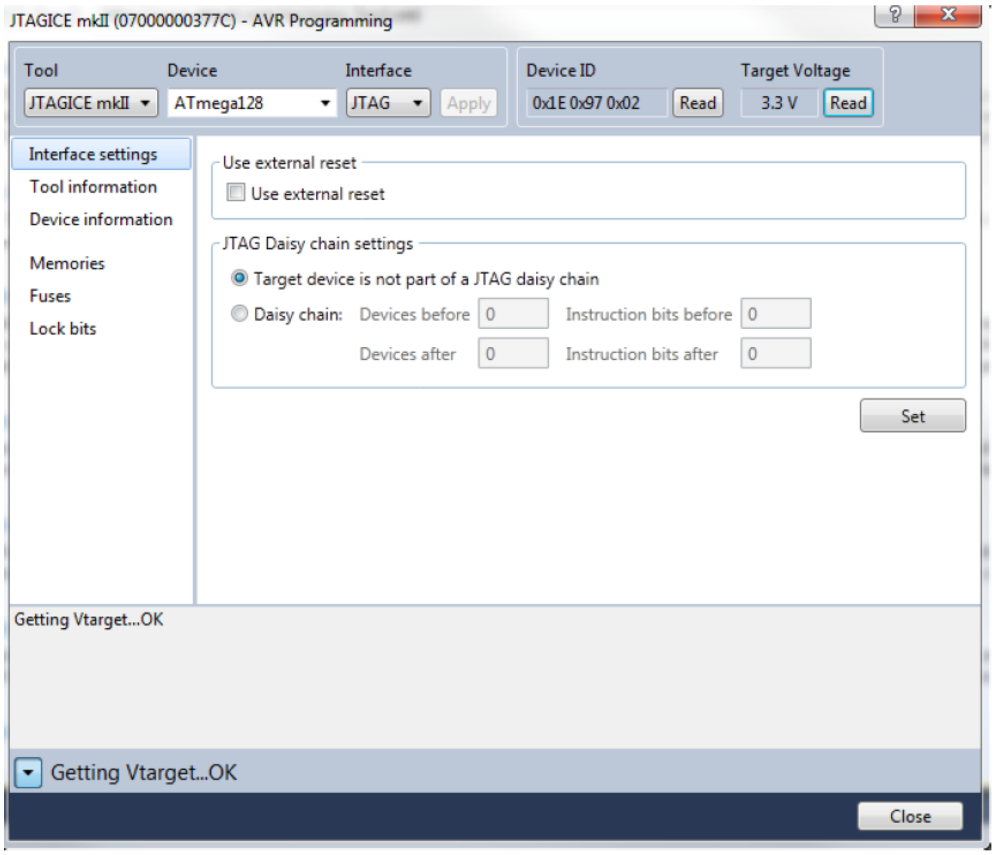
The following steps assume that you have the equipment available for your experiments:

One MIB520 board (or MIB510), several MicaZ or IRIS nodes, one JTAG mkII debugger, and cables for connecting these tools.

Now open AVR Studio, and go to Tools->AVR Tools Firmware Upgrade to upgrade the driver for JTAG to the latest version.

Now connect the MicaZ node to the MIB520/MIB510 programmer, and connect JTAG to MIB520/MIB510 programmer. There is a JTAG pin on MIB520/MIB510. After you have successfully connected things, then perform the following task:

Open AVR Studio to verify all connections are done correctly. Open AVR Studio, go to Tools->AVR Programming, and see the following figure:


JTAGICE mkII (07000000377C) - AVR Programming

Tool: JTAGICE mkII
Device: ATmega128
Interface: JTAG
Device ID: 0x1E 0x97 0x02
Target Voltage: 3.3 V

Use external reset: [ ] Use external reset

JTAG Daisy chain settings: [✓ Target device is not part of a JTAG daisy chain] Daisy chain: Devices before 0, Instruction bits before 0, Devices after 0, Instruction bits after 0

```
Getting Vtarget...OK
```

Select the tool and device as above, and click Apply to gray it out, and then click "Read" and "Read" on the right, you should see something similar to the Device ID above and the target voltage.

Note when you connect MIB520 to PC, it will ask for drivers. In that case, sometimes Windows 7 successfully finds driver for it. If not, here is a useful link containing a driver: http://digital.ni.com/public.nsf/allkb/FD238ED75B22740B

86257315004E35FE

A bare-bone Blink example

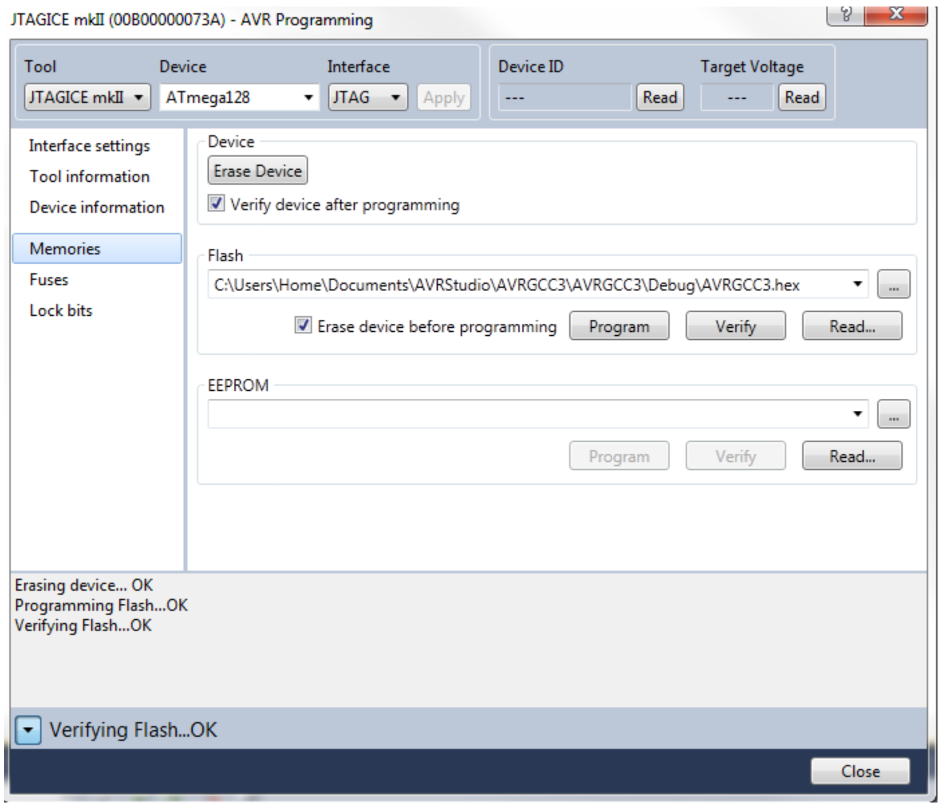
Read the AVR Studio User Guide by clicking Help->View Help. Figure out how to use AVR Studio, and various functionalities of the AVR Studio. If you are familiar with Microsoft Visual Studio, you will find AVR Studio easy to use, given that it is mostly based on Visual Studio.

Then write the following program into AVR Studio, and compile it successfully. Note that it is recommended if you choose to compile with –O1 or –O2 level of optimization in AVR-GCC.

NOTE: Never compile with –O0 in AVR environment for the LiteOS project.

```
#include <avr/io.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#define F_CPU 8000000UL // 8 MHz
#include <util/delay.h>
int main(void)
{
DDRA = 0xFF;
while(1)
{
PORTA = 0xFF;
_delay_ms(1000);
PORTA = 0x000;
_delay_ms(1000);
}
return 0; //It never reaches here!
}
```

Then upload the generated hex file to the MicaZ or IRIS node using JTAG. You should be able to see something like this in the uploading box. Note that here, my created project is located in my Documents directory. Your created project will probably be located somewhere else.


JTAGICE mkII (00B00000073A) - AVR Programming

Tool: JTAGICE mkII
Device: ATmega128
Interface: JTAG
Device ID: ---
Target Voltage: ---

Device
- [✓] Verify device after programming

Flash: C:\Users\Home\Documents\AVRStudio\AVRGCC3\AVRGCC3\Debug\AVRGCC3.hex
- [✓] Erase device before programming

EEPROM

```
Erasing device... OK
Programming Flash...OK
Verifying Flash...OK
```

Verifying Flash...OK

Now you should see that the MicaZ or IRIS node blinks periodically. This means that the AVR Studio and the JTAG have been installed successfully.

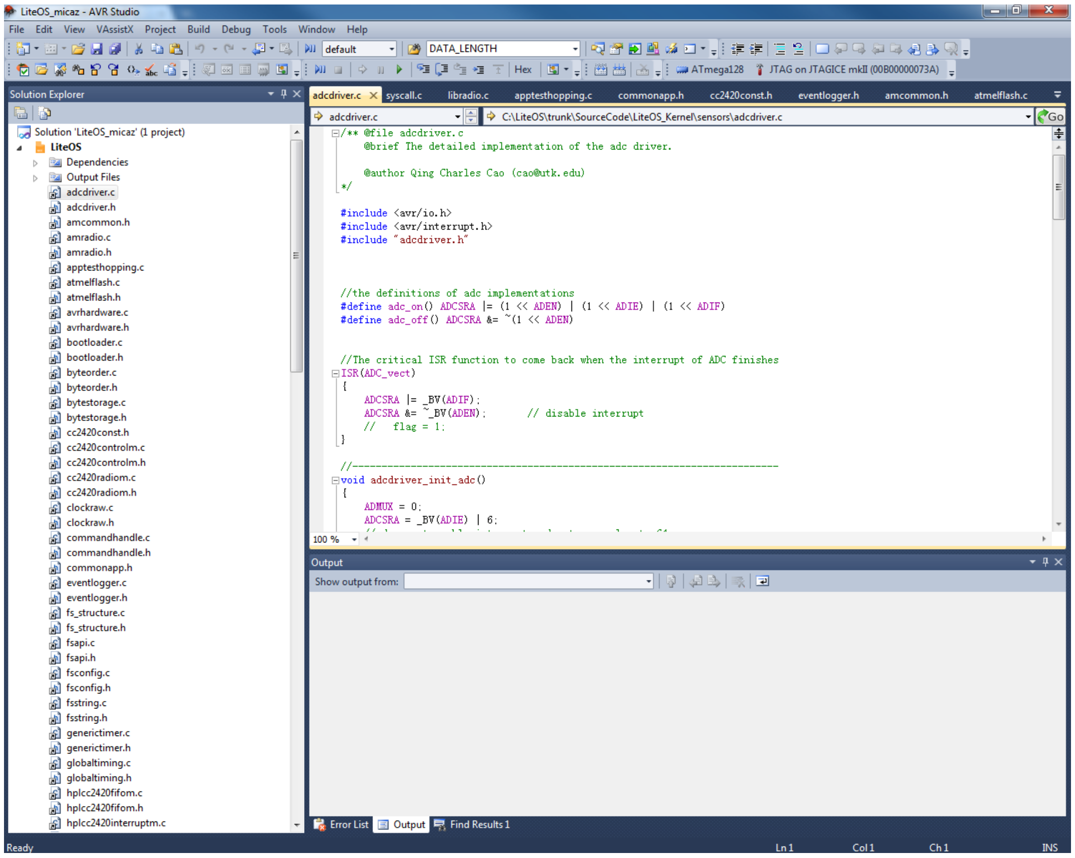
Install the LiteOS 2.1

LiteOS 2.1 is closely integrated with AVR Studio 5.0. To open

the LiteOS project, which is pre-built in AVR Studio, copy the LiteOS root folder to the C drive (required to correctly use AVR Studio), and open the file:
C:\LiteOS\trunk\LiteOSNewProject\LiteOS_micaz.avrsln

In case you are using IRIS motes, you should open the file LiteOS_iris.avrsln for you needs.

Then you should see something like following in AVR Studio:



```
/** @file adcdriver.c
    @brief The detailed implementation of the adc driver.

    @author Qing Charles Cao (cao@utk.edu)
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "adcdriver.h"



//the definitions of adc implementations
#define adc_on() ADCSRA |= (1 << ADEN) | (1 << ADIE) | (1 << ADIF)
#define adc_off() ADCSRA &= ~(1 << ADEN)


//The critical ISR function to come back when the interrupt of ADC finishes
ISR(ADC_vect)
{
    ADCSRA |= _BV(ADIF);
    ADCSRA &= ~_BV(ADEN);       // disable interrupt
    //   flag = 1;
}

//------------------------------------------------------------------------
void adcdriver_init_adc()
{
    ADMUX = 0;
    ADCSRA = _BV(ADIE) | 6;
```

Now you should be able to build LiteOS successfully, and download the code either through JTAG or through USB port. This part is detailed in the next section.

Understanding and Using LiteOS System

Installation LiteOS Operating System on the Motes

So far, you have successfully installed LiteOS on your PC. Congratulations! Next you want to install the kernel to the motes. You may already have several MicaZ or Iris motes at hand, and you have connected the programming board to your PC. The guide below illustrates how to install the LiteOS components on the motes.

Specifically, in the LiteOS environment, the model followed by LiteOS is simple: a base station node (which contains an application image as base) is connected to the PC. It receives your commands from the PC and translates them into message formats that can be understood by other nodes. These other nodes are installed with what we call the LiteOS kernel. Therefore, there are two images you will need to install to the nodes, the base station image, and the kernel image.

There are two ways to install a compiled image to the motes, a process that some refer to as "flashing". The first approach is through the AVR Studio environment. Refer to the previous section on how this is done. The second way is through command-line utilities such as uisp and avrdude. Specifically, we use avrdude in this guide. If your Cygwin environment does not have avrdude installed, you should download avrdude from its official website and install it.

Note: Only avrdude supports later hardware such as IRIS. If you only use MicaZ, you may also use uisp command, which is less reliable than the avrdude command.

Now start the Cygwin you have installed earlier. Note that if you are using Windows Vista or 7, you have to start Cygwin as follows: right click it, and select "Run as Administrator". This is because by default, the UAC mechanism of Windows Vista or 7 prevents Cygwin to perform certain actions such as creating new files and directories.

Install the base

Under the directory **binary**, LiteOS provides a binary file, called Base_micaz.hex, that allows you to create a base node. Note that you could not use AVR Studio to program a base node for compatibility reasons. Therefore, you need to follow the following steps when you create the base node.

First, before getting started, find out which port you are using on your PC. Open the device manager, under port (COMs and LPT) tab, and find out which port your PC is using to communicate with the programming board. Note that this setup is machine-specific. Normally, you will find that MIB510 connects to COM1 of your PC for the serial port. In case you use a serial-to-USB adaptor, you may find that another port is being used. In case you use MIB520, you will find a pair of ports, say, COM5 and COM6, are used. In our illustration below, we shall assume that the serial port used is COM5 and COM6, given that we are assuming a MIB520 to carry out the installation.

Assuming that you have two ports, COM5 and COM6, for the MIB520 device. Use the following command to flash the base image into the mote:

```
avrdude –p m128 –c mib510 –PCOM5 –V –U flash:w:Base_micaz.hex
```

Note that even you are using MIB520 here, you need to specify the command as mib510 in avrdude to avoid errors.

Note that if you are using IRIS motes, you should use m1281 instead of m128 here.

Install the kernel

Compared to Version 1.0 of LiteOS, the way to install the kernel is greatly simplified: After you compile the kernel in AVR Studio 5, copy the LiteOS.hex file generated in the default folder into the current folder, assuming that you are using the COM6 to download code, simply use the following command to install the kernel:

```
avrdude –p m128 –c mib510 –PCOM6 –V –U flash:w:LiteOS.hex
```

You may also use the AVR Studio to directly flash the kernel image.

Using LiteShell

After you have installed the kernel and the base, the next step is to start the LiteOS shell. Follow the procedure below to achieve this goal.

First navigate to the JavaTools/classes subdirectory. Start the shell as follows. In this case we assume we are using COM5 and COM6 to connect to the mote. **Note that, when you use MIB520, which shows up as a pair of communication ports, like COM5 and COM6, you must use the higher number of port in this command.**

```
$java tools.sf.SerialForwarder -comm serial@COM6:57600 &
```

This will start serial forwarder, and run it in the background.

Then you type in:

```
$java tools.terminal.terminal
```

Note that, in this case, the terminal program will connect to the serial forwarder to communicate. If the serial forwarder is not started, you will encounter an error like following:

```
$Error on sf@localhost:9001: java.net.ConnectException: Connection refused: connnect
```

Note:

> You may encounter a Java exception that says it cannot find the main class. If so, make sure that your current working directory is under the classes directory. You do not need to set any extra environment variables to invoke the terminal.

Under the hood: Explore the LiteOS Structure

In this section, we describe the system file structure for LiteOS. Once you have downloaded the LiteOS system and extracted the contents to the default directory (which is assumed to be C:\LiteOS), you will find that there is a directory named trunk inside. Inside this trunk directory, there are multiple sub-directories. We next describe the contents of each directory in more details. All our references to individual files in this directory are relative to the LiteOS\trunk directory.

SourceCode directory: This directory contains the source code for both the LiteOS kernel and the base station. Due to development environment issues, the base station source code cannot be compiled under AVR Studio. Instead, you may simply use the Base_micaz.hex and Base_iris.hex under the Binary directory to create base station nodes.

Tools directory: This directory contains all the tools available for LiteOS. Of particular interest is the tools located in JavaTools directory, which contains the detailed implementation on the java-based tools such as the PC side of the shell. In our development, we use Eclipse to develop tools under this directory.

LiteOSProjects directory: This directory contains the AVR Studio projects for both MicaZ nodes and IRIS nodes.

Documentation directory: This directory contains all the manuals available for the LiteOS project.

LiteOS Compilation Options

In this section, we describe the compilation options for LiteOS. Specifically, whenever you compile the LiteOS operating system, you are supplying some, but not all, of the following compilations options to the node:

```
-DPLATFORM_AVR
-DRADIO_CC2420
-DRADIO_RF230
-DPRINT_SOURCE_ENABLED
-DFILE_SYS_RANGE=32
-DMAX_FILE_TABLE_SIZE=2
-DNUM_BREAKPOINTS=0
-DMAX_MSG_LENGTH=32
-DLITE_MAX_THREADS=3
-DF_CPU=8000000UL
-DCOMMON_SHARE_SCHEDULING
-DAVR_STACK_PREPARE_LENGTH=34
-DBOOTLOADERSIZE=0
-DTRACE_MEMORY_CONTENTS
-DFORMATFILESYSTEM
-DTRACE_ENABLE
-DTRACE_MEMORY_CONTENTS
-DVER_DEBUG
-DTEST_PRINTING
-DPLATFORM_CPU_MEASURE
-DENERGY_SHARE_SCHEDULING
-DENERGYSAVINGMODE
-DENERGY_INSTRUMENTATION
```

-DVER_DEBUG
-DBASE_MODE
-Os
-combine
-Wall
-c
-gdwarf-2
-fsigned
-char
-v
-mmcu=atmega128
-mmcu=atmega1281

The following describes the information on each of these compilation options.

-DPLATFORM_AVR
This option specifies that the target platform is AVR. Currently LiteOS only supports AVR-based platforms, but there are plans to extend LiteOS to additional platforms in the future.

-DRADIO_CC2420
This option specifies that the target radio chip is CC2420. Specifically, if you are compiling towards MicaZ, you should use –DRADIO_CC2420.

–DRADIO_RF230
If you are compiling towards IRIS, you should use –DRADIO_RF230.

-DPRINT_SOURCE_ENABLED
This option is for debugging purposes only, and can be safely disabled in production mode. This option specifies that when a node transmits debugging information via the serial port, it prints the source node id that sends this message. This is useful when multiple MIB520 boards are connected to the PC simultaneously for debugging purposes.

-DFILE_SYS_RANGE=32
This option specifies the number of file system inode blocks. Specifically the smaller this number, the fewer bytes of memory the file system will consume in the EEPROM on the sensor node.

-DMAX_FILE_TABLE_SIZE=2
This option specifies the maximum number of opened file handles globally. This parameter should be set larger if more files need to be opened at the same time.

-DNUM_BREAKPOINTS=0
This option specifies the number of breakpoints that will be supported at runtime during interactive use with the end user. These breakpoints are for debugging purpose only. Please note that they are software breakpoints, which means that they are separate from the breakpoints that are inherently supported by the AVR Studio environment.

-DMAX_MSG_LENGTH=64
This option specifies the largest length of messages to be received by the applications. Setting it to be 64 is sufficient for most application needs.

-DLITE_MAX_THREADS=3
This option specifies the maximum number of concurrent threads to be active in the kernel. Setting it to be 3 means that at most three active threads will be active at the same time. If you are using IRIS motes, you can set a much larger number given that IRIS has more memory available.

-DF_CPU=8000000UL
This option specifies the CPU frequency of the node. On MicaZ or IRIS, this defaults to 8MHz.

-DCOMMON_SHARE_SCHEDULING
This option specifies the scheduling policy. Currently, the common_share_scheduling is the default scheduling policy. The user may also implement their own scheduling policies.

-DAVR_STACK_PREPARE_LENGTH=34
This option specifies the stack length to be used by the user. By default this should be 34 under any optimization levels. If there is no optimization, this value should be set to be 36. However, this is strongly not recommended.

-DBOOTLOADERSIZE=0
This is the size of the bootloader. By default it is set to be 0. If bootloader is used, it should be set as 512.

-DTRACE_MEMORY_CONTENTS
This is an option for debugging only. If the node will report the memory contents periodically, then it should be set to be true. Otherwise, it should not be set.

-DFORMATFILESYSTEM
Whether the node should format the file system at the beginning of system reboot. By default, this is not set, because most applications do not require using the file system. But if the application needs to read/write files, then this option has to be set.

-DTRACE_ENABLE
Whether the tracing functionality is turned on or not. If it is turned on, the system will record many types of events for debugging purposes.

-DENERGYSAVINGMODE
Turn on the energy saving mode. By default it is turned off.

-DENERGY_INSTRUMENTATION
Turn on the energy instrumentation mode. By default it is turned off.

-DVER_DEBUG
Turn on additional debugging statements in the project.

-DTEST_PRINTING
Test the printing when the code initializes.

-DPLATFORM_CPU_MEASURE
Measure the CPU usage during the execution of programs.

-DENERGY_SHARE_SCHEDULING
Schedule threads based on their energy consumption.

-DVER_DEBUG
Debug mode. Prints additional parameter values.

-DBASE_MODE
Base mode, especially in use for iris motes.

-Os -combine -Wall -c -gdwarf-2 -fsigned-char -v -mmcu=atmega128(1)
These are the settings common to AVR GCC. Refer to the AVR GCC documentation on the details of these options.

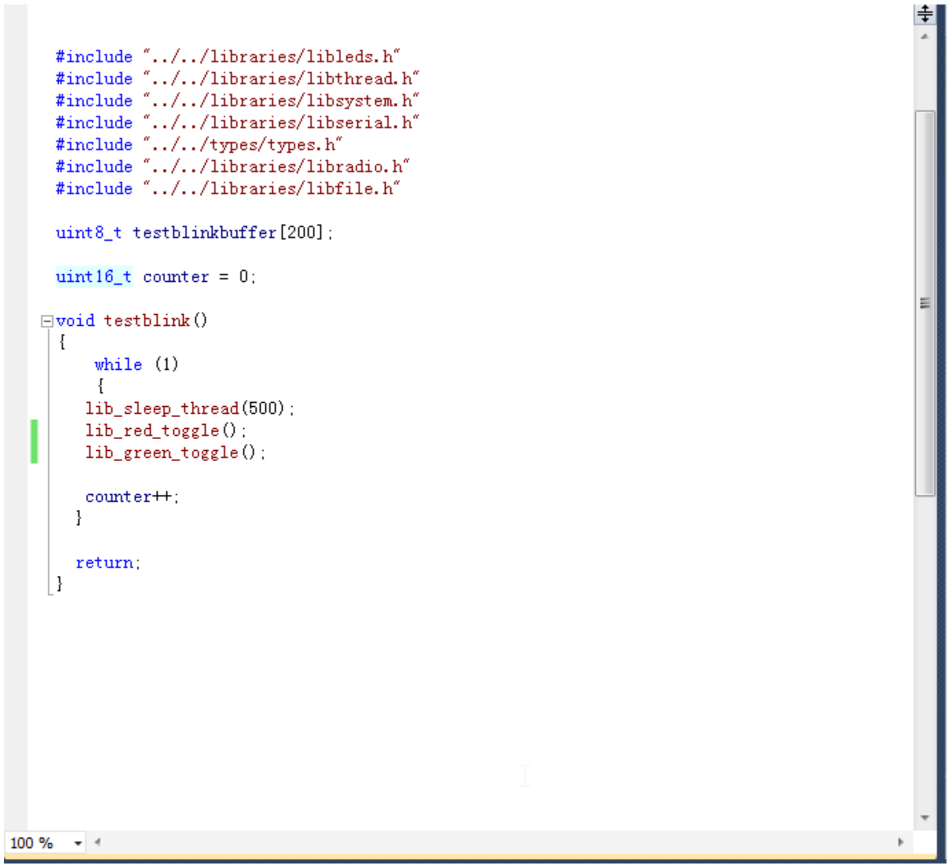
Application Development

Your first application under the LiteOS framework

Now that you have installed the LiteOS kernel and the base station on a bunch of nodes, you may wonder how to write and deploy your own applications. This brief guide shows you how to run a typically application, such as Blink, inside the LiteOS environment.

Different from LiteOS 1.0, which requires you to develop applications separately with the kernel and compile them as two steps, in LiteOS 2, we are compiling everything in one single step. The advantages of doing this is that as a developer, you no longer need to manually copy the applications from the PC to the node wirelessly. In fact, this process is time-consuming and error-prone. LiteOS 2 therefore removes this step to streamline the process of development and debugging.

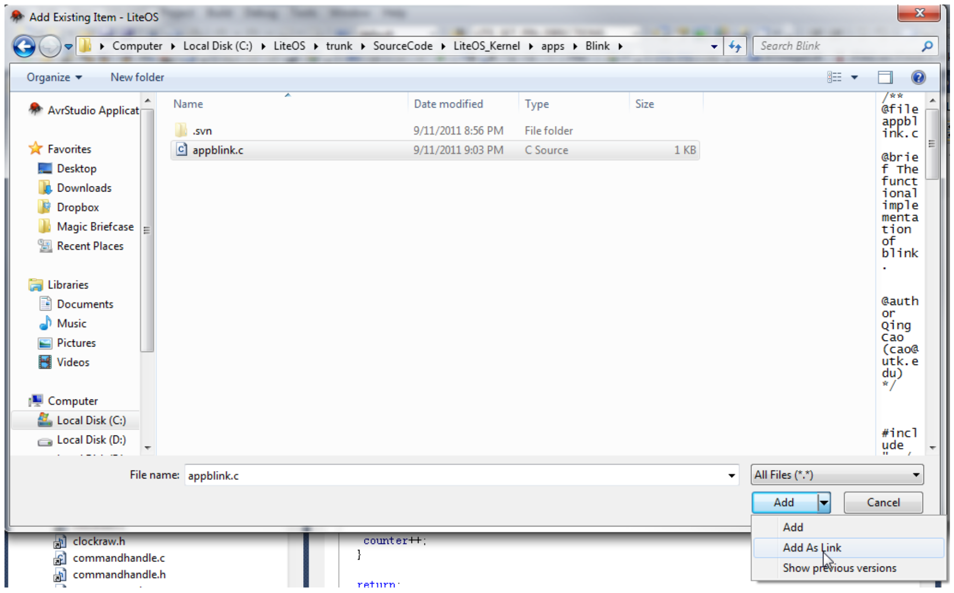
There are three steps to create link to an application in LiteOS 2. First, you write an application with a specific entry function. For example, if you want to write a blink application, you can write something like the following:



```
#include "../../libraries/libleds.h"
#include "../../libraries/libthread.h"
#include "../../libraries/libsystem.h"
#include "../../libraries/libserial.h"
#include "../../types/types.h"
#include "../../libraries/libradio.h"
#include "../../libraries/libfile.h"

uint8_t testblinkbuffer[200];

uint16_t counter = 0;

void testblink()
{
    while (1)
    {
   lib_sleep_thread(500);
   lib_red_toggle();
   lib_green_toggle();

   counter++;
  }

  return;
}
```

Note that in this screenshot, there is a buffer which is uint8_t testblinkbuffer[200]. This is needed as the thread will have a separate stack compared to the main function. Therefore, this buffer needs to be supplied by the user application. Second, note that there is an entry function testblink() in the user application. This function is going to serve as the entry function for the thread. Finally, note that this function toggles the red and green LED in the function body. It mainly consists of a loop that periodically toggles so that the LED will blink accordingly.

After you have created this application in a separate file, e.g., appblink.c, in the Apps directory, the next step is to add this file to the LiteOS project in AVR Studio. Note that you should always add the file by link, which means that you are not creating a separate copy of the file in the AVR Studio. See the following screenshot for this process.



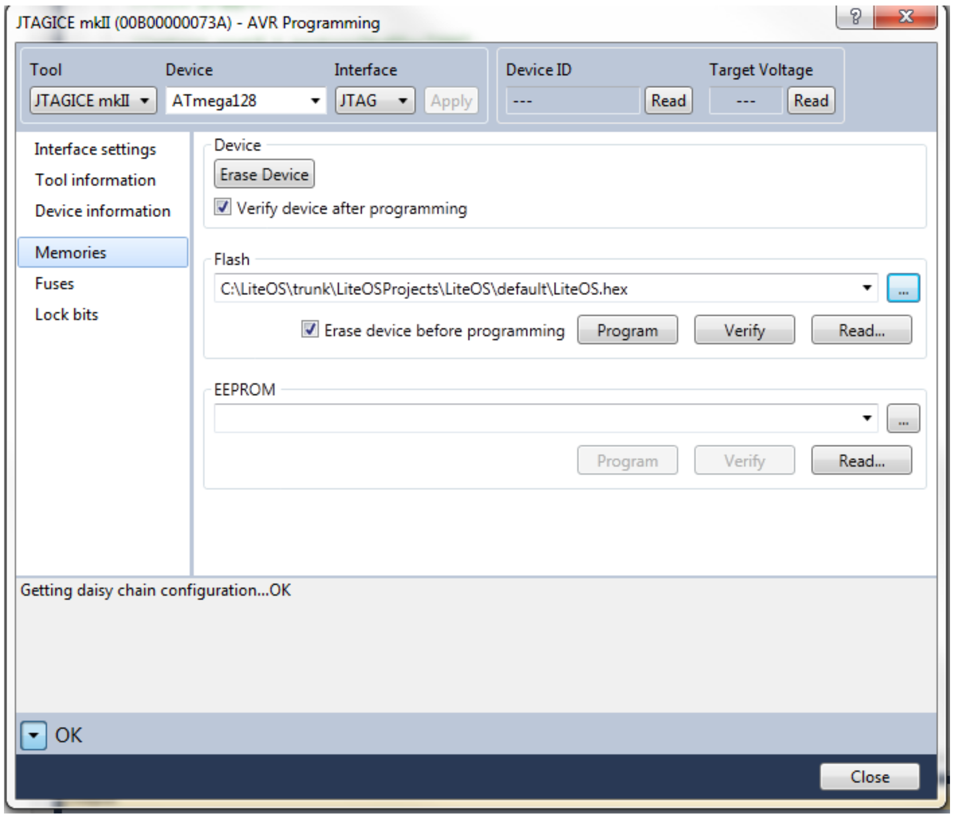
Finally, the last step is to add a reference to the application code in the file commonapp.h to enable the files to be compiled successfully. In this example, you will see something like this in the commonapp.h file:

```
extern uint8_t testblinkbuffer[200];
void testblink();
```

After these three steps, you should be able to compile the LiteOS kernel together with the application successfully.

Deploy and execute your application

After you have developed your application, the next step is to deploy and execute the application on the mote. To do this, simply open the menu at Tools->AVR Programming to open the dialog as shown below.



C:\LiteOS\trunk\LiteOSProjects\LiteOS\default\LiteOS.hex

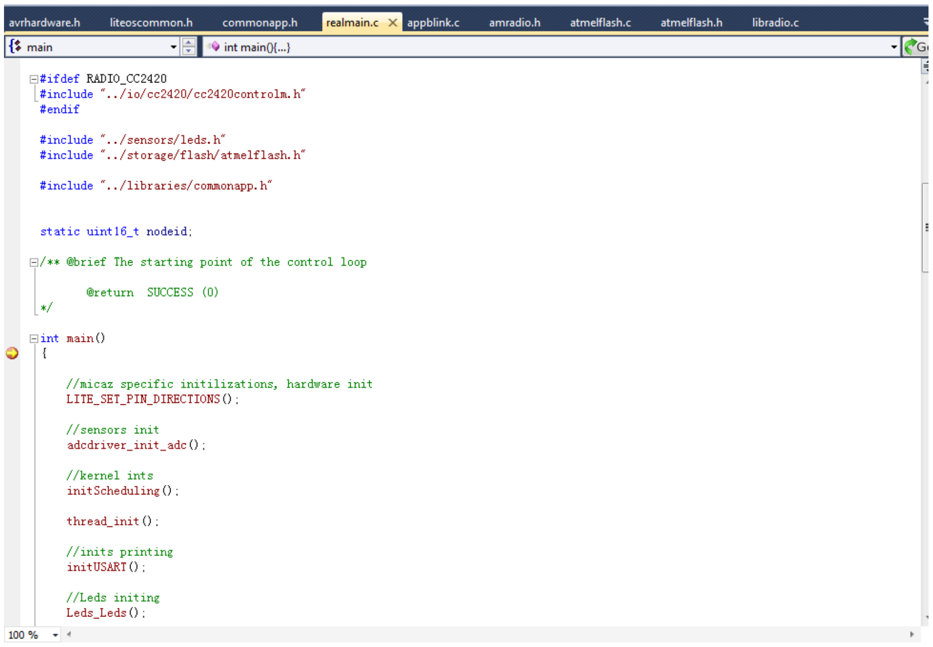
Getting daisy chain configuration...OK

Then click the Program button to flash the mote with the new LiteOS image. This time you should see the Blink application is working correctly.

Debugging your application

One great advantage of using AVR Studio is that it provides very powerful debugging capabilities based on the JTAG hardware device. Specifically, you may follow the following steps to start debugging the application.

In the menus, click the Debug-> Start Debugging and Break to start the debugging process. After the debugging is started, you should be able to see the screenshot as following:



```
#ifdef RADIO_CC2420
#include "../io/cc2420/cc2420controlm.h"
#endif

#include "../sensors/leds.h"
#include "../storage/flash/atmelflash.h"

#include "../libraries/commonapp.h"


static uint16_t nodeid;

/** @brief The starting point of the control loop

      @return  SUCCESS (0)
*/

int main()
{

    //micaz specific initilizations, hardware init
    LITE_SET_PIN_DIRECTIONS();

    //sensors init
    adcdriver_init_adc();

    //kernel ints
    initScheduling();

    thread_init();

    //inits printing
    initUSART();

    //Leds initing
    Leds_Leds();
```

This screen shows that the debugging has been successfully started and the breakpoint is now at the first line of the code.

Use LiteShell Commands

Now that you have installed the LiteOS kernel and the base station on a bunch of nodes, you may wonder how to operate a sensor network using LiteShell. The command details of LiteShell are explained in this section.

Suppose we here installed LiteOS on several nodes, we can then use LiteShell commands to operate them.

List current directory information

LiteOS offers a logical view of the sensor network in which they appear to be "mounted" to the root of the PC's directory tree. Hence, the user should expect to see a set of directories, one for each sensor sub-network. Within each, the user should expect to see a set of subdirectories, one for each node of the given sensor network. Finally, within a node, the user should expect to see the local file system.

The example below illustrates the use of common Unix commands to list directory information on the nodes. In most cases, we use the ls command. This command has the option of –l, which lists detailed information. Ls always displays the current up-to-date information.

Caution:

If you turn on formatting the file system option, every time a mote is turned on, it takes about 1-2 minutes to finish the boot-up process when it formats the file system, among other tasks. Please be patient during this phase. Typing ls during this time will get no results, but this is normal.

```
$ java tools.terminal.terminal
Welcome to use LiteOS shell.

$ls -l
The returned has 3 packets.

Name    Type    Size    Protection
sensornet1  network   --    rwxrwxrwx
Time elapes 547

$cd sensornet1
cd command successful
Time elapes 0

$ls -l
The returned has 3 packets.

Name    Type    Size    Protection
nodeC  noderoot   --    rwxrwxrwx
nodeB  noderoot   --    rwxrwxrwx
nodeA  noderoot   --    rwxrwxrwx

Time elapes 500

$cd nodeA
cd command successful
Time elapes 0

$ls -l
The returned has 1 packets. Name
            Type    Size    Protection
Time elapes 516

$cd de
No such subdirectory exists. Probably you have not used the ls command to list
uch directory. Currently cd only works for directories that are listed using ls
Time elapes 0
```

Create and delete files and directories

The next example illustrates creating and deleting files and directories. The commands involved are ls, cd, mkdir, touch, and rm.

```
$ls
```

```
The returned has 3 packets.
nodeC
nodeB
nodeA
Time elapes 500

$cd nodeA
cd command successful
Time elapes 0

$ls
The returned has 0 packets.
Time elapes 500

$mkdir test1
Make dir complete
Time elapes 265

$touch test2
creating file complete
Time elapes 266

$ls -l
The returned has 3 packets. Name
          Type   Size   Protection
test1  directory   --    rwxrwxrwx
test2  file   0    rwxrwxrwx
Time elapes 532

$rm test1
Time elapes 32

$ls -l
The returned has 1 packets.
Name    Type   Size   Protection
test2  file   0    rwxrwxrwx
Time elapes 500
```

Copy data between the PC and a node

By using the cp command, you may also copy data from and to the PC. The current copy status is displayed in real time to help locate network problems.

Remember that a wireless sensor network is assumed to be organized into one or more sub-networks, each of which has a unique network name. Each sub-network, in turn, has multiple nodes, each again with a different name. The local file system on each node is organized hierarchically. To address a file in a sensor network, therefore, the user simply provides a complete location, or a relative location to the current working directory.

To address a file on your PC, you are required to start the file location with **/.** For example, if a file is located at c:/Data/mydata.txt, then it can be written as /c/Data/mydata.txt in LiteShell.

Caution:

> LiteShell currently does not support navigating in your PC directories. Nor does it support copy file between nodes. To do the former, use Windows Explorer. To do the later, please use the PC as the relay.

In the following example, we copy a file called testcp.zip from the location c:/Temp/testcp.zip to the current working directory of /sn01/nodeC (the root directory of nodeC). To achieve this, do the following:

```
cp /c/Temp/testcp.zip testcp.zip
```

You can again copy it back to your PC location c:/Temp/testcpback.zip by:

```
cp testcp.zip /c/Temp/testcpback.zip
```

The copy operation in LiteShell is reliable. Hence, you can be certain that when you copy large files, not one byte of it will be corrupted by the underlying unreliable radio communication.

```
$ls -l
The returned has 1 packets. Name
         Type   Size   Protection
test2  file   0    rwxrwxrwx
Time elapes 515

$cp /c/temp/test.zip test.zip
The reply has 1 packets.
Now trying to send 0 30
Now trying to send sync
Now reply is good on sync
Now trying to send 30 60
Now trying to send sync
Now reply is good on sync
Now trying to send 60 90
Now trying to send sync
Now reply is good on sync
Now trying to send 90 106
Now trying to send sync
Now reply is good on sync
cp succeed
Copy finished
Time elapes 10375

$ls -l
The returned has 2 packets.

Name    Type   Size   Protection
test.zip  file   5254    rwxrwxrwx
test2  file   0     rwxrwxrwx
Time elapes 500

$rm test.zip
Time elapes 47

$ls -l
The returned has 1 packets. Name
           Type   Size   Protection
test2  file   0     rwxrwxrwx
Time elapes 500

$cp /c/Temp/test.zip . The
reply has 1 packets. Now
trying to send 0 30
Now trying to send sync
Sync reply shows has got 29
Now trying to send 29 59
Now trying to send sync
No sync reply
Now trying to send 29 59
Now trying to send sync
Now reply is good on sync
Now trying to send 59 89
Now trying to send sync
Now reply is good on sync
Now trying to send 89 106
Now trying to send sync
Now reply is good on sync
cp succeed
Copy finished
Time elapes 13625
```

```
$ls -l
The returned has 2 packets. Name
            Type    Size    Protection
test.zip  file   5254     rwxrwxrwx
test2  file   0     rwxrwxrwx
Time elapes 516
```

Other File System Command Examples

This part illustrates several other command examples for controlling a sensor network. Commands cd, ls, search, ps, pwd, and du are illustrated. Please refer to Appendix I for a list of commands that are supported.

```
$search test
/sensornet1/nodeA/test.zip
/sensornet1/nodeA/test2
/sensornet1/nodeA/test3
/sensornet1/nodeA/test3/test4
/sensornet1/nodeA/test3/test4/testfile
Time elapes 516

$cd test3
cd command successful
Time elapes 0

$ls
The returned has 1 packets.
test4
Time elapes 500

$cd test4
cd command successful
Time elapes 16

$ls
The returned has 1 packets.
testfile
Time elapes 500

$pwd
/sensornet1/nodeA/test3/test4
Time elapes 0

$du
```

```
The reply has 1 packets.
Node nodeA remains 87% of EEPROM and 98% of Flash

Time elapes 47

$ps
The reply has 1 packets.
sysshell   Active
Time elapes 15
```

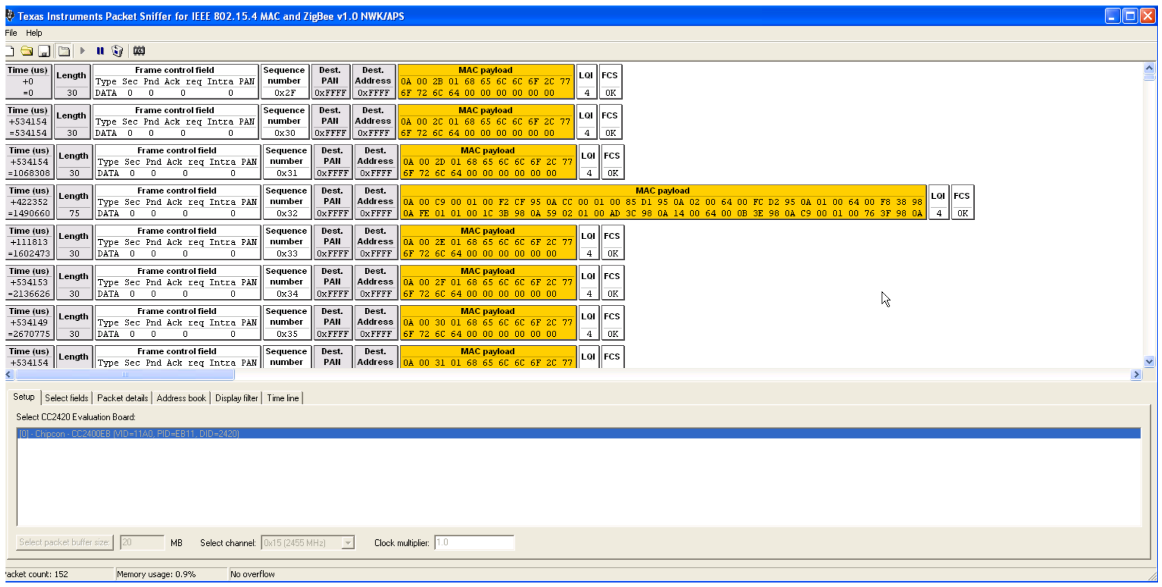
Advanced Features of LiteOS

LiteOS provides two functions that are unique for debugging and analyzing code behavior. In this section, we describe these two features, how to turn them on, and how to analyze the resulting data.

Using the Event Logging Functions in LiteOS

The event logging tool is turned on by setting the TRACE_ENABLE option in the compiler options. When enabled, it will periodically report data to the base station containing the events that have occurred in the kernel. Specifically, you will obtain something like the following in the packet sniffer (we use the CC2420 packet sniffer as an example).

Once TRACE_ENABLE is turned on, there is a line of code, initTrace(4000) in the realmain.c file that specifies the reporting frequency in milliseconds. In this example, we are setting the reporting frequency to be 4 seconds. If you turn on the packet sniffer, you will periodically see some output as shown in the following figure:


Texas Instruments Packet Sniffer for IEEE 802.15.4 MAC and ZigBee v1.0 NWK/APS
File
Help
Setup
Select fields
Packet details
Address book
Display filter
Time line

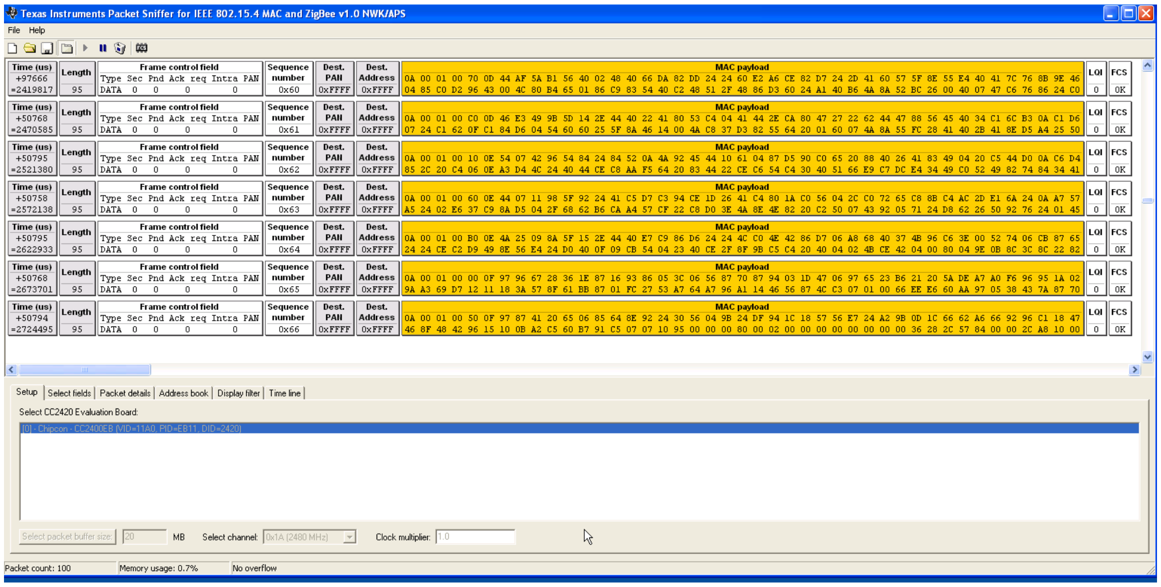
```
Time (us)  Length  Frame control field                Sequence  Dest.   Dest.    MAC payload                                                                                          LQI  FCS
                   Type Sec Pnd Ack req Intra PAN     number    PAN     Address
+0         30      DATA  0   0      0        0        0x2F      0xFFFF  0xFFFF   0A 00 2B 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=0                                                                               6F 72 6C 64 00 00 00 00 00 00
+534154    30      DATA  0   0      0        0        0x30      0xFFFF  0xFFFF   0A 00 2C 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=534154                                                                          6F 72 6C 64 00 00 00 00 00 00
+534154    30      DATA  0   0      0        0        0x31      0xFFFF  0xFFFF   0A 00 2D 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=1068308                                                                         6F 72 6C 64 00 00 00 00 00 00
+422352    75      DATA  0   0      0        0        0x32      0xFFFF  0xFFFF   0A 00 C9 00 01 00 F2 CF 95 0A CC 00 01 00 85 D1 95 0A 02 00 64 00 FC D2 95 0A 01 00 64 00 F8 38 98   4    OK
=1490660                                                                         0A FE 01 01 00 1C 3B 98 0A 59 02 01 00 AD 3C 98 0A 14 00 64 00 0B 3E 98 0A C9 00 01 00 76 3F 98 0A
+111813    30      DATA  0   0      0        0        0x33      0xFFFF  0xFFFF   0A 00 2E 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=1602473                                                                         6F 72 6C 64 00 00 00 00 00 00
+534153    30      DATA  0   0      0        0        0x34      0xFFFF  0xFFFF   0A 00 2F 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=2136626                                                                         6F 72 6C 64 00 00 00 00 00 00
+534149    30      DATA  0   0      0        0        0x35      0xFFFF  0xFFFF   0A 00 30 01 68 65 6C 6C 6F 2C 77                                                                     4    OK
=2670775                                                                         6F 72 6C 64 00 00 00 00 00 00
+534154                                                                          0A 00 31 01 68 65 6C 6C 6F 2C 77
```

```
Select CC2420 Evaluation Board:
[0] - Chipcon - CC2400EB (VID=11A0, PID=EB11, DID=2420)

Select packet buffer size: 20 MB    Select channel: 0x15 (2455 MHz)    Clock multiplier: 1.0

acket count: 152    Memory usage: 0.9%    No overflow
```

As observed, the long packet in the middle is the packet that contains the events.

Reporting Memory Contents in LiteOS

The memory logger tool is similar to use, except that it requires the base channel, the reporting channel, and the frequency in its parameter settings. Once configured well, it will also obtain some output like the following:


Texas Instruments Packet Sniffer for IEEE 802.15.4 MAC and ZigBee v1.0 NWK/APS
File
Help
Setup
Select fields
Packet details
Address book
Display filter
Time line

```
Time (us)  Length  Frame control field                Sequence  Dest.   Dest.    MAC payload                                                                                                                          LQI  FCS
                   Type Sec Pnd Ack req Intra PAN     number    PAN     Address
+97666     95      DATA  0   0       0         0      0x60      0xFFFF  0xFFFF   0A 00 01 00 70 0D 44 AF 5A B1 56 40 02 48 40 66 DA 82 DD 24 24 60 E2 A6 CE 82 D7 24 2D 41 60 57 5F 8E 55 E4 40 41 7C 76 8B 9E 46
=2419817                                                                         04 85 C0 D2 96 43 00 4C 80 B4 65 01 86 C9 83 54 40 C2 48 51 2F 48 86 D3 60 24 A1 40 B6 4A 8A 52 BC 26 00 40 07 47 C6 76 86 24 C0  0    OK

+50768     95      DATA  0   0       0         0      0x61      0xFFFF  0xFFFF   0A 00 01 00 C0 0D 46 E3 49 9B 5D 14 2E 44 40 22 41 80 53 C4 04 41 44 2E CA 80 47 27 22 62 44 47 88 56 45 40 34 C1 6C B3 0A C1 D6
=2470585                                                                         07 24 C1 62 0F C1 84 D6 04 54 60 60 25 5F 8A 46 14 00 4A C8 37 D3 82 55 64 20 01 60 07 4A 8A 55 FC 28 41 40 2B 41 8E D5 A4 25 50  0    OK

+50795     95      DATA  0   0       0         0      0x62      0xFFFF  0xFFFF   0A 00 01 00 10 0E 54 07 42 96 54 84 24 84 52 0A 4A 92 45 44 10 61 04 87 D5 90 C0 65 20 88 40 26 41 83 49 04 20 C5 44 D0 0A C6 D4
=2521380                                                                         85 2C 20 C4 06 0E A3 D4 4C 24 40 44 CE C8 AA F5 64 20 83 44 22 CE C6 54 C4 30 40 51 66 E9 C7 DC E4 34 49 C0 52 49 82 74 84 34 41  0    OK

+50758     95      DATA  0   0       0         0      0x63      0xFFFF  0xFFFF   0A 00 01 00 60 0E 44 07 11 98 5F 92 24 41 C5 D7 C3 94 CE 1D 26 41 C4 80 1A C0 56 04 2C C0 72 65 C8 8B C4 AC 2D E1 6A 24 0A A7 57
=2572138                                                                         A5 24 02 E6 37 C9 8A D5 04 2F 68 62 B6 CA A4 57 CF 22 C8 D0 3E 4A 8E 4E 82 20 C2 50 07 43 92 05 71 24 D8 62 26 50 92 76 24 01 45  0    OK

+50795     95      DATA  0   0       0         0      0x64      0xFFFF  0xFFFF   0A 00 01 00 B0 0E 4A 25 09 8A 5F 15 2E 44 40 E7 C9 86 D6 24 24 4C C0 4E 42 86 D7 06 A8 68 40 37 4B 96 C6 3E 00 52 74 06 CB 87 65
=2622933                                                                         24 24 CE C2 D9 49 8E 56 E4 24 D0 40 0F 09 CB 54 04 23 40 CE 2F 8F 9B C5 C4 20 40 04 02 4B CE 42 04 00 80 04 9E 0B 8C 3C 8C 22 82  0    OK

+50768     95      DATA  0   0       0         0      0x65      0xFFFF  0xFFFF   0A 00 01 00 00 0F 97 96 67 28 36 1E 87 16 93 86 05 3C 06 56 87 70 87 94 03 1D 47 06 97 65 23 B6 21 20 5A DE A7 A0 F6 96 95 1A 02
=2673701                                                                         9A A3 69 D7 12 11 18 3A 57 8F 61 BB 87 01 FC 27 53 A7 64 A7 96 A1 14 46 56 87 4C C3 07 01 00 66 EE E6 60 AA 97 05 38 43 7A 87 70  0    OK

+50794     95      DATA  0   0       0         0      0x66      0xFFFF  0xFFFF   0A 00 01 00 50 0F 97 87 41 20 65 06 85 64 8E 92 24 30 56 04 9B 24 DF 94 1C 18 57 56 E7 24 A2 9B 0D 1C 66 62 A6 66 92 96 C1 18 47
=2724495                                                                         46 8F 48 42 96 15 10 0B A2 C5 60 B7 91 C5 07 07 10 95 00 00 00 80 00 02 00 00 00 00 00 00 00 00 36 28 2C 57 84 00 00 2C A8 10 00  0    OK
```

```
Select CC2420 Evaluation Board:
[0] - Chipcon - CC2400EB (VID=11A0, PID=EB11, DID=2420)

Select packet buffer size: 20 MB    Select channel: 0x1A (2480 MHz)    Clock multiplier: 1.0

Packet count: 100    Memory usage: 0.7%    No overflow
```

After this, a typical memory analysis tool can be used to reproduce the memory contents in the kernel.

-

Appendix

Appendix I: A List of LiteShell Commands supported

Command Name	Command Semantics
ls	List directory content
cd	Enter a directory
cp	Copy files
rm	Delete files or directories
mkdir	Create a new directory
touch	Create a new file
pwd	Display current directory location
du	Display remaining file system space
ps	Display running threads
kill	Kill a thread
man	Show the help of commands
search	Search particular files using names
format	Format the file system
setbasemode	Set the base mode to be promiscuous mode or not
rbb	Reset the base station
rbn	Reset the entire sensor network within one hop
history	Print the list of previous commands
setchannel	Change the channel used by the base station
memory	Display memory allocation information of current threads

Troubleshooting Tips

- Check FAQ on the website for updated answers to common problems.
- If things went wrong, reboot the node.
- The communication between base and node could be severely degraded by all kinds of interference. If one command does not get a reply, try it again.
- Because of unreliable radio, multiple ls –l commands may return different results.
- Connect all cables and battery securely.
- Report bugs in mailing lists so that they can be solved in future releases.
- It takes 1-2 minutes to boot up a node and format itself. Please be patient during this procedure.
- Some environments have strange problems with file permissions when extracting from the tar.gz tar ball. Change file permission to 777 if you meet such problems, i.e., type the following command: chmod 777 *.