

**Wireless Sensor Networks for  
Monitoring Cracks in Structures:  
Source Code and Configuration  
Files**

Mathew P. Kotowsky



## Contents

Chapter 1. MICA2-Based Wireless ACM Version 1 Software: Modification of SenseLightToLog	1
1. MDA300Logger Directory	2
2. java Directory	27
Chapter 2. MICA2-Based Wireless ACM Version 2 Software: Modification of XMDA300	37
1. xbow/apps/XMesh/XMDA300 directory	38
2. xbow/tos/sensorboards/mda300 directory	53
3. xbow/tos/sensorboards/mda300 directory	64
Chapter 3. MICA2-Based Wireless ACM Version 3 Software: Modification Version 2 XMDA300: <i>Shake 'n Wake</i>	77
1. MICA2 Software	78
1.1. xbow/tos/XLib directory	78
1.2. xbow/apps/XMesh/XMDA300 directory	81
2. UC-7420 Software	97
2.1. xbow/beta/tools/src/xcmd directory	97
2.2. xbow/beta/tools/src/xcmd/apps directory	107
2.3. xbow/beta/tools/src/xlisten/boards directory	113
2.4. xbow/beta/tools/src/xlisten directory	120
Chapter 4. ēKo mote-based wireless ACPS software	133
1. CPA crack propagation pattern: cpa.xml	134
2. CPC crack propagation pattern: cpc.xml	137



## CHAPTER 1

# **MICA2-Based Wireless ACM Version 1 Software: Modification of SenseLightToLog**

This appendix contains all software used to program the MICA2 motes for Version 1 of the MICA2-based wireless ACM system. This appendix is organized by software directory and only the modified files are included.

## 1. MDA300Logger Directory

MDA300Logger is a software application designed to take readings from a Crossbow MDA300 sensor board and store them on the EEPROM memory on a Crossbow Mica 2 mote until such time as they are uploaded over the radio by a PC running the BcastInject java application. Its modular and command-processing structure are based on SenseLightToLog, and therefore has all the same dependencies as that application, with the addition of the MDA300-specific dependencies that are specified in each file. MDA300Logger also uses a slightly modified version of the same driver used for XSensorMDA300.

MDA300Logger requires at least one 'remote' mote to have the MDA300Logger application installed on it, one 'base' mote on a programming board to have TOSBase installed on it, and one PC connected to the programming board (either directly or via a network) running the modified version of BcastInject in order to start and stop the test and upload data.

```
COMPONENT=MDA300Logger  
SENSORBOARD=mda300
```

```
PFLAGS= -I../.. /tos/interfaces \  
        -I../.. /tos/system \  
        -I../.. /tos/lib \  
        -I../.. /tos/sensorboards/$(SENSORBOARD) \  
        -I../.. /tos/platform/mica2 \
```

```
PROGRAMMER_EXTRA_FLAGS = -v=2
```

```
include ../MakeXbowlocal  
include ${TOSROOT}/apps/Makeules
```

#### 4. MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*
 * "Copyright (c) 2000–2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002–2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL–LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/*
 * MDA300Logger.nc is based on SenseLightToLog.nc written by
 * David Culler and Su Ping – Intel Research Berkeley Lab
 * Date: 7/11/2002
 *
 * Modifications by:
 * Hasan Ozer and Mat Kotowsky – Northwestern University
 * November, 2004
 */

#include sensorboardApp;

configuration MDA300Logger {
    provides interface Sensing;
}

implementation {
```

```
components Main, MDA300LoggerM, SimpleCmd, LedsC, SamplerC;
components GenericComm as Comm, TimerC, Logger;
components HPLPowerManagementM;

Main.StdControl->MDA300LoggerM;

MDA300LoggerM.Leds -> LedsC;
MDA300LoggerM.Timer -> TimerC.Timer[unique("Timer")];
MDA300LoggerM.LoggerWrite -> Logger.LoggerWrite;
MDA300LoggerM.PowerEnable -> HPLPowerManagementM.Enable;

MDA300LoggerM.Comm -> Comm;
MDA300LoggerM.Logger -> Logger;

Sensing = MDA300LoggerM.Sensing;

// Sampler Communication
MDA300LoggerM.SamplerControl -> SamplerC.SamplerControl;
MDA300LoggerM.Sample -> SamplerC.Sample;

// support for plug and play
MDA300LoggerM.PlugPlay -> SamplerC.PlugPlay;
}
```

## 6. MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*
 * "Copyright (c) 2000–2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002–2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL–LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/**
 * The MDA300Logger module is designed to take readings from a Crossbow
 * MDA300 sensor board and store them in the flash memory of a Mica2
 * mote until such time as the user downloads the data over the wireless
 * radio using a java application on a PC.
 */

includes sensorboard;

module MDA300LoggerM {
  provides interface StdControl;
  provides interface Sensing;

  uses {
    interface StdControl as Comm;
    interface StdControl as Logger;
    interface Leds;
    interface Timer as Timer;
  }
}
```

```

interface LoggerWrite;
interface ProcessCmd as CmdExecute;

//Sampler Communication
interface StdControl as SamplerControl;
interface Sample;

interface PowerManagement;
//support for plug and play
command result_t PlugPlay();
command result_t PowerEnable();
}
}
implementation
{
#define ANALOG_SAMPLING_TIME    10
#define MISC_SAMPLING_TIME      15
#define READY_TEMPERATURE 0x01
#define READY_HUMIDITY    0x02
#define READY_BATTERY    0x04
#define READY_DATA      0x08
#define LINE_READY      0x0F

#define TIME_POS    0
#define SECS_POS    2
#define SAMPLE_POS  4
#define BATT_POS    6
#define SCRATCHLPOS 7

#define TEMP_POS    4
#define HUMID_POS   5
#define DATA_POS   6

#define RADIO_TICK_INTERVAL 30

// declare module static variables here
uint16_t currentRow1[8]; // current working row 1 of log
uint16_t currentRow2[8]; // current working row 2 of log
uint16_t nullDataRow[8];
uint8_t lineStatus;
short busy = 0;
unsigned int nsamples; //samples left
unsigned int total_samples; //total samples
int8_t record[25];
uint32_t time_sec;
uint32_t interval_sec;
unsigned int ticks = 0;
uint32_t secs = 0;
int radioCounter = RADIO_TICK_INTERVAL;
int rowWaiting = 0;

/**
 * Initialize the application.

```

## 8. MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
* @return Initialization status.
**/
command result_t StdControl.init() {
    call PowerEnable();
    call SamplerControl.init();
    call Logger.init();
    call Comm.init();
    return SUCCESS;
}

/**
 * Insert a row that indicates the beginning of the test
 **/

task void writeNullTask() {
    call LoggerWrite.append((uint8_t*) &nullDataRow);
}

/**
 * Start the application.
 * @return Start status.
 **/
command result_t StdControl.start() {
    int i;
    call Leds.redOn();
    for (i=0; i<8; i++)
        nullDataRow[i] = 0xFFFF;

    lineStatus=0;
    return rcombine (call Logger.start(), call Comm.start());
}

/**
 * Stop the application.
 * @return Stop status.
 **/
command result_t StdControl.stop() {
    call Logger.stop();
    return call Comm.stop();
}

/**
 * This command belongs to the <code>Sensing</code> interface.
 * It starts the timer to generate periodic events.
 *
 * @return Always returns <code>SUCCESS</code>
 **/
command result_t Sensing.start(unsigned long samples,
                               unsigned long interval,
                               unsigned long time) {
    if(samples == 0)
    {
        return SUCCESS;
    }
}
```

```

}

call Leds.redOff();
ticks = 0;
secs = 0;
call LoggerWrite.resetPointer();
nsamples = samples;
total_samples = samples;
time_sec = (uint32_t) time;
interval_sec = (uint32_t) interval;
if(interval_sec >= 30) // timer will fire every 30 seconds
    call Timer.start(TIMER_REPEAT, 30*1024);

else // interval_sec < 30, timer will fire once a second
    call Timer.start(TIMER_REPEAT, 1024);

atomic {
    nullDataRow[TIME_POS] = time_sec & 0xFFFF;
    nullDataRow[TIME_POS + 1] = (time_sec >> 16) & 0xFFFF;
}

post writeNullTask();
call SamplerControl.start();
call SamplerControl.stop();
return SUCCESS;
}

/**
 * This command belongs to the <code>Sensing</code> interface.
 * It turns everything off when the test is stopped
 *
 * @return Always returns <code>SUCCESS</code>
 */
command result_t Sensing.stop() {
    call Timer.stop();
    signal Sensing.done();
    return SUCCESS;
}

/**
 * Event handler to the <code>Timer.fired</code> event.
 * @return Always returns <code>SUCCESS</code>
 */
event result_t Timer.fired() {
    int i;

    if (interval_sec >= 30) // ticks are 30 seconds apart
    {
        ticks += 30;
        radioCounter--;
        secs += 30;

        if(radioCounter == 0)

```

10 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```

        call Comm.stop();
    }

    else // interval_sec < 30 so ticks are 1 second apart
    {
        ticks++;
        secs++;
    }

    if (ticks < interval_sec)// we haven't accumulated enough seconds
        return SUCCESS;

    ticks = 0;

    for (i=0; i < 7; i++) // Clear current row
    {
        currentRow1[i] = 0;
        currentRow2[i] = 0;
    }

    if(busy)
    {
        return SUCCESS;
    }

    if (nsamples== 0) {
        call Timer.stop();
        signal Sensing.done();
        call Leds.redOn();
        return call Comm.start();
    }

    // Whenever ticks = interval, start sampler
    call Comm.start(); // turn the radio on while taking a sample
    call Leds.greenOn();
    radioCounter = RADIO.TICK.INTERVAL;

    call SamplerControl.start();
    if(1){
        atomic {
            record[14] = call Sample.getSample(
                0,
                TEMPERATURE,
                MISC.SAMPLING.TIME,
                SAMPLER.DEFAULT);
            record[15] = call Sample.getSample(
                0,
                HUMIDITY,
                MISC.SAMPLING.TIME,
                SAMPLER.DEFAULT);
        }
    }

```

```

record[16] = call Sample.getSample(
    0,
    BATTERY,
    MISC_SAMPLING_TIME,
    SAMPLER_DEFAULT);
    record[17] = call Sample.getSample(
    7,
    ANALOG,
    ANALOG_SAMPLING_TIME,
    AVERAGE_FOUR
    | EXCITATION_25
    | EXCITATION_33
    | EXCITATION_50
    | EXCITATION_ALWAYS_ON);
    busy = 1;
}
}

nsamples--;
return SUCCESS;
}

/**
 * Default event handler to the <code>Sensing.done</code> event.
 * @return Always returns <code>SUCCESS</code>
 */
default event result_t Sensing.done() {
return SUCCESS;
}

task void writeRow1() {
atomic{
    call LoggerWrite.append((uint8_t*) &currentRow1);
}
}

task void writeRow2() {
atomic{
    call LoggerWrite.append((uint8_t*) &currentRow2);
    rowWaiting = 0;
}
}

/**
 * Handle a single dataReady event for all MDA300 data types.
 */
event result_t Sample.dataReady(
    uint8_t channel,
    uint8_t channelType,
    uint16_t data)
{

```

## 12 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```

switch (channelType) {
  case ANALOG:
    switch (channel) {
      case 7:
        atomic {
          currentRow2[DATA_POS] = data;
          lineStatus |= READY_DATA;
        }
        break;

      default:
        break;
    } // case ANALOG (channel)
    break;

  case BATTERY:
    atomic {
      currentRow1[BATT_POS] = data;
      lineStatus |= READY_BATTERY;
    }
    break;

  case HUMIDITY:
    atomic {
      currentRow2[HUMID_POS] = data;
      lineStatus |= READY_HUMIDITY;
    }
    break;

  case TEMPERATURE:
    atomic {
      currentRow2[TEMP_POS] = data;
      lineStatus |= READY_TEMPERATURE;
    }
    break;
  default:
    break;
} // switch (channelType)

if(lineStatus == LINE_READY)
{
  lineStatus = 0;
  atomic{
    currentRow1[SAMPLE_POS] = (uint16_t) nsamples;
    currentRow1[SAMPLE_POS + 1] = (uint16_t) total_samples;
    //currentRow1[SECS_POS] = (uint16_t) secs;
    currentRow1[SECS_POS] = secs & 0xFFFF;
    currentRow2[SECS_POS] = currentRow1[SECS_POS];
    currentRow1[SECS_POS + 1] = (secs >> 16) & 0xFFFF;
    currentRow2[SECS_POS + 1] = currentRow1[SECS_POS + 1];
    currentRow1[SCRATCH_POS] = 0x7777;
  }
}

```

```

        currentRow1[TIME_POS] = time_sec & 0xFFFF;
        currentRow2[TIME_POS] = currentRow1[TIME_POS];
        currentRow1[TIME_POS + 1] = (time_sec >> 16) & 0xFFFF;
        currentRow2[TIME_POS + 1] = currentRow1[TIME_POS + 1];
    }
    post writeRow1();
    rowWaiting=2;
}
return SUCCESS;
}

/**
 * Event handler for the <code>LoggerWrite.writeDone</code> event.
 * Toggle the yellow LED if status is true.
 *
 * @return Always returns <code>SUCCESS</code>
 */
event result_t LoggerWrite.writeDone( result_t status ) {
if (rowWaiting == 2)
{
    post writeRow2();
}
if (rowWaiting == 0)
{
    busy = 0;
}
call SamplerControl.stop();
call SamplerControl.init();
    return SUCCESS;
}

/**
 * Event handler to the <code>CmdExecute.done</code> event. Do nothing.
 * @return Always returns <code>SUCCESS</code>
 */
event result_t CmdExecute.done(TOS_MsgPtr pmsg, result_t status ) {
    return SUCCESS;
}
}
}

```

14 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*
 * "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/*
 * This software was slightly modified by Hasan Ozer and Mat Kotowsky to
 * include a 3rd integer argument to the start_sensing command. It
 * remains otherwise unchanged from the original University of
 * California software. Contact: kotowsky@northwestern.edu
 */

/** Configuration for SimpleCmd module.
 *
 * Author/Contact: tinyos-help@millennium.berkeley.edu
 *
 * Description:
 *
 * SimpleCmd module demonstrates a simple command interpreter for TinyOS
 * tutorial (Lesson 8 in particular). It receives a command message from
 * its RF interface, which triggers a command interpreter task to execute
 * the command. When the command is finished executing, the component
```

```

* signals the upper layers with the received message and the status
* indicating whether the message should be further processed.
*
* As a simple version, it can only interpret the following commands:
* Led_on (1), Led_off(2), radio_quieter(3), radio_louder(4),
* start_sensing(5), and read_log(6). Start sensing commands will trigger
* the Sensing.start interface while read_log will read the EEPROM with a
* specific log line and broadcast the line over the radio when read is
* done.
*/
includes SimpleCmdMsg;

configuration SimpleCmd {
    provides interface ProcessCmd;
}
implementation {
    components Main, SimpleCmdM, MDA300Logger, Logger, TimerC,
        GenericComm as Comm, PotC, LedsC;

    Main.StdControl -> SimpleCmdM;
    SimpleCmdM.Leds -> LedsC;

    ProcessCmd = SimpleCmdM.ProcessCmd;

    SimpleCmdM.CommControl -> Comm;

    SimpleCmdM.ReceiveCmdMsg -> Comm.ReceiveMsg [AMSIMPLECMDMSG];
    SimpleCmdM.SendLogMsg -> Comm.SendMsg [AMLOGMSG];
    SimpleCmdM.LoggerRead -> Logger;
    SimpleCmdM.Pot -> PotC;
    SimpleCmdM.Sensing -> MDA300Logger.Sensing;
    SimpleCmdM.ReadTimer -> TimerC.Timer [unique("Timer")];
}

```

16 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*          tab:4
 * "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/**
 * Defines an interface for a component that senses data at a certain
 * interval and scale.
 */

interface Sensing
{
    /**
     * Start sensing data.
     * @param nsamples The number of samples to gather.
     * @param interval_sec The interval (in msec) at which to gather samples.
     */
    command result_t start(unsigned long nsamples,
                           unsigned long interval_sec,
                           unsigned long time_sec);

    command result_t stop();
}
```

```
/**  
 * Signalled when sensing has completed.  
 */  
event result_t done();  
}
```

18MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */
```

```
/*
          tab:4
 * "Copyright (c) 2000–2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002–2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL–LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */
```

```
/*
 * This software was slightly modified by Hasan Ozer and Mat Kotowsky to
 * include a 3rd integer argument to the start_sensing command. It
 * remains otherwise unchanged from the original University of
 * California software. Contact: kotowsky@northwestern.edu
 */
```

```
/*
 * Author: Robert Szewczyk, Su Ping
 *
 * $\Id$
 */
```

```
/**
 * @author Robert Szewczyk
 * @author Su Ping
 */
```

```

includes SimpleCmdMsg;

/**
 *
 * This is an enhanced version of SimpleCmd that understands the
 * START_SENSING and READLOG commands.
 */
module SimpleCmdM {
    provides {
        interface StdControl;
        interface ProcessCmd;
    }

    uses {
        interface Leds;
        interface Pot;
        interface ReceiveMsg as ReceiveCmdMsg;
        interface StdControl as CommControl;
        interface LoggerRead;
        interface SendMsg as SendLogMsg;
        interface Sensing;
        interface Timer as ReadTimer;
    }
}

implementation
{

    // declare module static variables here
    TOS_MsgPtr cur_msg; // The current command message
    TOS_Msg log_msg;    // The current log message
    bool send_pending;  // TRUE if a message send is pending
    bool eeprom_read_pending; // TRUE if an EEPROM read is pending
    TOS_Msg buf;        // Free buffer for message reception
    int reads;
    int readAttempts;

    /**
     * This task evaluates a command and executes it.
     * Signals ProcessCmd.sendDone() when the command has completed.
     * @return Return: None
     */
    task void cmdInterpret() {
    struct LogMsg *lm;
        struct SimpleCmdMsg *cmd = (struct SimpleCmdMsg *)cur_msg->data;
        result_t status = SUCCESS;

        // do local packet modifications:
        // update the hop count and packet source
        cmd->hop_count++;
        cmd->source = TOS_LOCAL_ADDRESS;

```

```

// Execute the command
switch (cmd->action) {
case LED_ON:
    call Leds.yellowOn();
    break;
case LED_OFF:
    call Leds.yellowOff();
    break;
case RADIO_QUIETER:
    call Pot.increase();
    break;
case RADIO_LOUDER:
    call Pot.decrease();
    break;
case START_SENSING:
    // Initialize the sensing component, and start reading data from it.
    lm = (struct LogMsg *) (log_msg.data);
    lm->sourceaddr = TOS_LOCAL_ADDRESS;
    if (call SendLogMsg.send(TOS_BCAST_ADDR, sizeof(struct LogMsg), &log_msg)) {
        send_pending = TRUE;
    }
    call Sensing.start(cmd->args.ss_args.nsamples,
        cmd->args.ss_args.interval,
        cmd->args.ss_args.time_sec);
    call LoggerRead.resetPointer();
    break;
case READ_LOG:
    // Check if the message is meant for us, if so issue a split phase call
    // to the logger
    // call Leds.redOff();
    // call Leds.yellowOff();
    reads = 0;
    readAttempts = 0;
    call Sensing.stop();
    if ((cmd->args.rl_args.destaddr == TOS_LOCAL_ADDRESS) &&
        (eeprom_read_pending == FALSE)) {
    if (call LoggerRead.readNext(((struct LogMsg *) log_msg.data)->log)) {
        // call Leds.yellowOn();
        eeprom_read_pending = TRUE;
    }
    call ReadTimer.start(TIMER_REPEAT, 100);
    }
    break;
default:
    status = FAIL;
}

signal ProcessCmd.done(cur_msg, status);
}

event result_t ReadTimer.fired() {
    readAttempts++;
}

```

```

    if(readAttempts > 300) {
        call ReadTimer.stop();
        call Leds.yellowOff();
        call Leds.redOn();
    }
    call Leds.yellowToggle();
    if (!(eeprom_read_pending || send_pending)) {
        if (call LoggerRead.readNext(((struct LogMsg *)log_msg.data)->log)) {
            eeprom_read_pending = TRUE;
        }
    }
    return SUCCESS;
}
/**
 * Signalled in response to the event from <code>Sensing.done</code>.
 * @return Always returns <code>SUCCESS</code>
 */
event result_t Sensing.done() {
    call Leds.yellowOff();
    return SUCCESS;
}

/**
 * Initialize the application.
 * @return Success of component initialization.
 */
command result_t StdControl.init() {
    cur_msg = &buf;
    send_pending = FALSE;
    eeprom_read_pending = FALSE;
    return rcombine(call CommControl.init(), call Leds.init());
}

/**
 * Start the application.
 * @return Always returns <code>SUCCESS</code>
 */
command result_t StdControl.start(){
    return SUCCESS;
}

/**
 * Stop the application.
 * @return Always returns <code>SUCCESS</code>
 */
command result_t StdControl.stop(){
    return SUCCESS;
}

/**
 * Signalled when the log has completed the reading,
 * and now we're ready to send out the log message.
 * @return Always returns <code>SUCCESS</code>

```

## 22 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```

    **/
event result_t LoggerRead.readDone(uint8_t * packet, result_t success) {
    // Send message only if read was successful
    struct LogMsg *lm;
    if (success && eeprom_read_pending && !send_pending) {
        lm = (struct LogMsg *) (log_msg.data);
        lm->sourceaddr = TOSLOCAL_ADDRESS;
        if (call SendLogMsg.send(TOS_BCAST_ADDR, sizeof(struct LogMsg), &log_msg)) {
// call Leds.redOn();
send_pending = TRUE;
        }
    }
    eeprom_read_pending = FALSE;
    // call Leds.yellowOff();
reads++;
if((readAttempts >= 100) || (reads >= 100)) {
    call ReadTimer.stop();
    call Leds.yellowOff();
    call Leds.redOn();
}
return SUCCESS;
}

/**
 * Post a task to process the message in 'pmsg'.
 * @return Always returns <code>SUCCESS</code>
 */
command result_t ProcessCmd.execute(TOS_MsgPtr pmsg) {
    cur_msg = pmsg;
    post cmdInterpret();
    return SUCCESS;
}

/**
 * Called upon message receive; invokes ProcessCmd.execute().
 */
event TOS_MsgPtr ReceiveCmdMsg.receive(TOS_MsgPtr pmsg){
    result_t retval;
    TOS_MsgPtr ret = cur_msg;

    retval = call ProcessCmd.execute(pmsg);
    if (retval==SUCCESS) {
        return ret;
    } else {
        return pmsg;
    }
}

/**
 * Default event handler for <code>ProcessCmd.done</code>.
 * @return The value of 'status'.
 */

```

```
default event result_t ProcessCmd.done(TOS_MsgPtr pmsg, result_t status) {
    return status;
}

/**
 * Reset send_pending flag to FALSE in response to
 * <code>SendLogMsg.sendDone</code>.
 * @return The value of 'status'.
 */
event result_t SendLogMsg.sendDone(TOS_MsgPtr pmsg, result_t status) {
    // call Leds.redOff();
    send_pending = FALSE;
    return status;
}

} // end of implementation
```

## 24 MICA2-BASED WIRELESS ACM VERSION 1 SOFTWARE: MODIFICATION OF SENSELIGHTTOLOG

```
/* sensorboard.h - hardware specific definitions for the MDA300
*/

// crossbow sensor board id
#define SENSOR_BOARD_ID 0x81 //MDA300 sensor board id

#define NUMMSG1.BYTES (28) // bytes 2-29
#define NUMMSG2.BYTES (8) // bytes 2-9
#define NUMMSG3.BYTES (13) // bytes 2-13

// format is:
// byte 1 & 2: ADC reading in big-endian format

enum {
    Sample.Packet = 6,
};

enum {
    RADIO.TEST,
    UART.TEST
};
```

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*          tab:4
 * "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/*
 * File Name: SimpleCmd.h
 *
 * Description:
 * This header file defines the AMSIMPLECMDMSG and AMLOGMSG message
 * types for the SimpleCmd and MDA300Logger applications.
 */

/*
 * This software was slightly modified by Hasan Ozer and Mat Kotowsky to
 * include a 3rd integer argument to the start_sensing command. It
 * remains otherwise unchanged from the original University of
 * California software. Contact: kotowsky@northwestern.edu
 */

enum {
    AMSIMPLECMDMSG = 8,
    AMLOGMSG=9
```

```

};

enum {
    LED_ON = 1,
    LED_OFF = 2,
    RADIO_QUIETER = 3,
    RADIO_LOUDER = 4,
    START_SENSING = 5,
    READLOG = 6
};

typedef struct {
    int nsamples;
    uint32_t interval;
    uint32_t time_sec;
} start_sense_args;

typedef struct {
    uint16_t destaddr;
} read_log_args;

// SimpleCmd message structure
typedef struct SimpleCmdMsg {
    int8_t seqno;
    int8_t action;
    uint16_t source;
    uint8_t hop_count;
    union {
        start_sense_args ss_args;
        read_log_args rl_args;
        uint8_t untyped_args[0];
    } args;
} SimpleCmdMsg;

// Log message structure
typedef struct LogMsg {
    uint16_t sourceaddr;
    uint8_t log[16];
} LogMsg;

```

## 2. java Directory

```
/*
 * This software is largely based on software written by the University of
 * California. As per the implied license agreement, the original copyright
 * notice is below.
 */

/*
 * "Copyright (c) 2000–2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002–2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL–LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/**
 *
 * @author <a href="mailto:szewczyk@sourceforge.net">Robert Szewczyk</a>
 */

/*
 * Modified by Hasan Ozer (h-ozero@northwestern.edu) and Mat Kotowsky
 * (kotowsky@northwestern.edu) to be used with the MDA300Logger project.
 */

package net.tinyos.tools;

import net.tinyos.util.*;
import java.io.*;
```

```

import java.util.Properties;
import java.util.Date;
import net.tinyos.message.*;

public class BcastInject implements MessageListener {
    static Properties p = new Properties();
    public static final byte LED_ON = 1;
    public static final byte LED_OFF = 2;
    public static final byte RADIO_LOUDER = 3;
    public static final byte RADIO_QUIETER = 4;
    public static final byte START_SENSING = 5;
    public static final byte READ_LOG = 6;

    public boolean read_log_done = false;
    public boolean start_sensing_done = false;
    public boolean isRead = false;

    public static final short TOS_BCAST_ADDR = (short) 0xffff;

    public static String outputFilename;

    public static double t;

    public static void usage() {
        System.err.println("Usage: java net.tinyos.tools.BcastInject"+
            " <command> [arguments]");
        System.err.println("\twhere <command> and [arguments] can be one of the following:");
        System.err.println("\t\tled_on");
        System.err.println("\t\tled_off");
        System.err.println("\t\ttradio_louder");
        System.err.println("\t\ttradio_quieter");
        System.err.println("\t\ttstart_sensing [nsamples interval_sec time_sec]");
        System.err.println("\t\ttread_log [dest_address]");
    }

    public static void startSensingUsage() {
        System.err.println("Usage: java net.tinyos.tools.BcastInject"
            + " start_sensing [num_samples interval_sec time_sec]");
    }

    public static void readLogUsage() {
        System.err.println("Usage: java net.tinyos.tools.BcastInject" +
            " read_log [dest_address] [data_file]");
    }

    public static byte restoreSequenceNo() {
        try {
            FileInputStream fis = new FileInputStream("bcast.properties");
            p.load(fis);
            byte i = (byte) Integer.parseInt(p.getProperty("sequenceNo", "1"));
            fis.close();
            return i;
        } catch (IOException e) {

```

```

        p.setProperty("sequenceNo", "1");
        return 1;
    }
}

public static void saveSequenceNo(int i) {
    try {
        FileOutputStream fos = new FileOutputStream("bcast.properties");
        p.setProperty("sequenceNo", Integer.toString(i));
        p.store(fos, "#Properties for BcastInject\n");
    } catch (IOException e) {
        System.err.println("Exception while saving sequence number" + e);
        e.printStackTrace();
    }
}

public static void main(String[] argv) throws IOException {
    String cmd;
    byte sequenceNo = 0;
    boolean read_log = false;
    boolean start_sensing = false;

    if (argv.length < 1) {
        usage();
        System.exit(-1);
    }

    cmd = argv[0];

    if (cmd.equals("start_sensing") && argv.length != 4) {
        startSensingUsage();
        System.exit(-1);
    } else if (cmd.equals("read_log") && argv.length != 3) {
        readLogUsage();
        System.exit(-1);
    }

    SimpleCmdMsg packet = new SimpleCmdMsg();

    sequenceNo = restoreSequenceNo();
    packet.set_seqno(sequenceNo);
    packet.set_hop_count((short)0);
    packet.set_source(0);

    if (cmd.equals("led_on")) {
        packet.set_action(LED.ON);
    } else if (cmd.equals("led_off")) {
        packet.set_action(LED.OFF);
    } else if (cmd.equals("radio_louder")) {
        packet.set_action(RADIO.LOUDER);
    } else if (cmd.equals("radio_quieter")) {
        packet.set_action(RADIO.QUIETER);
    }
}

```

```

    } else if (cmd.equals(" start_sensing")) {
        packet.set_action(START_SENSING);
        short nsamples = (short)Integer.parseInt(argv[1]);
        long interval_sec = (long)Integer.parseInt(argv[2]);
        int time_sec = (int)Integer.parseInt(argv[3]);
        packet.set_args_ss_args_nsamples(nsamples);
        packet.set_args_ss_args_interval(interval_sec);
        packet.set_args_ss_args_time(time_sec);
        start_sensing = true;
    } else if (cmd.equals(" read_log")) {
        read_log = true;
        packet.set_action(READLOG);
        short address = (short)Integer.parseInt(argv[1]);
        outputFilename = argv[2];
        packet.set_args_rl_args_destaddr(address);
    } else {
        usage();
        System.exit(-1);
    }
}

try {
    System.err.print(" Sending payload: ");

    for (int i = 0; i < packet.dataLength(); i++) {
        System.err.print(Integer.toHexString(packet.dataGet()[i] & 0xff)+ " ");
    }

    System.err.println();

    MoteIF mote = new MoteIF(PrintStreamMessenger.err);

    // Need to wait for messages to come back
    BcastInject bc = null;

    if (read_log || start_sensing) {
        bc = new BcastInject();
        mote.registerListener(new LogMsg(), bc);
    }

    mote.send(TOS_BCAST_ADDR, packet);

    if (start_sensing) {
        p.setProperty("putHeader", "true");
        synchronized (bc) {
            if (bc.start_sensing_done == false) {
                System.err.println("Waiting for response to start_sensing...");
                bc.wait(10000);
            }

            System.err.println("Done waiting for acks.");
        }
    }
}

```

```

if (read_log) {
    System.err.println("Output file is " + outputFilename);
    synchronized (bc) {
        bc.isRead = true;
        if (bc.read_log_done == false) {
            System.err.println("Waiting for response to read_log...");
            bc.wait(10000);
        }

        System.err.println("Done waiting for data.");
        p.setProperty("putHeader", "true");
    }
}

saveSequenceNo(sequenceNo+1);
System.exit(0);

} catch(Exception e) {
    e.printStackTrace();
}
}

public void messageReceived(int dest_addr, Message m) {
    FileOutputStream fos = null;
    PrintWriter fileOut = null;
    FileOutputStream los = null;
    PrintWriter logOut = null;
    try
    {
        los = new FileOutputStream("BcastInject.log", true);
        logOut = new PrintWriter(los);
        logOut.println(new Date());
        logOut.println("-----");
    }
    catch (IOException e)
    {
        System.err.println("Error opening log file " + e);
        e.printStackTrace();
    }

    if(isRead)
    {
        try
        {
            fos = new FileOutputStream(outputFilename, true); // append
            fileOut = new PrintWriter(fos);
        }

        catch (IOException e )
        {
            System.err.println("Error while opening output file: " + e);
            e.printStackTrace();
        }
    }
}

```

```

    }

    long timestamp = 0;
    String [] dataRow = new String [8];
    String rawString = "";

    LogMsg lm = (LogMsg) m;

    for (int i = 0; i < 16; i++) {
        rawString +=
            (lm.getElement_log(i) < 16? "0" : "")
            + Long.toHexString(lm.getElement_log(i) & 0xff)
            + " ";
    }

    if(isRead)
    {
        if(p.getProperty("putHeader", "true").equals("true"))
        {
            fileOut.println(new Date());
            fileOut.println("-----");
        }
        p.setProperty("putHeader", "false");
        fileOut.println(rawString);
        fileOut.flush();

        try
        {
            fileOut.close();
            fos.close();
        }

        catch (IOException e)
        {
            System.err.println("Error closing the file: " + e);
            e.printStackTrace();
        }
    }

    if(!isRead)
    {
        System.err.println("Mote " + lm.getSourceaddr() + " has started.");
    }

    else if((lm.getElement_log(5) & lm.getElement_log(7)) == 0xFF)
        // two "random" positions are FF
    {
        int i = 0; // position of timestamp
        timestamp=0;
        timestamp |= lm.getElement_log(i+3) & 0xFF;
        timestamp <<= 8;
        timestamp |= lm.getElement_log(i+2) & 0xFF;
        timestamp <<= 8;
    }

```

```

timestamp |= lm.getElement_log(i+1) & 0xFF;
timestamp <<= 8;
timestamp |= lm.getElement_log(i) & 0xFF;
logOut.println("New test start at " + timestamp + "!");
}

else
{

int lineType;
if ((lm.getElement_log(14) == 0x77) & (lm.getElement_log(15) == 0x77))
    lineType=1;
else
    lineType=2;

//System.err.println("Received log message: "+lm);
logOut.println("Raw log message; " + lm);
logOut.println("reading a line of type " + lineType);

for (int i = 0; i < lm.numElements_log(); i+=2) {
    int intVal = 0;
    intVal |= lm.getElement_log(i+1) & 0xFF;
    intVal <<= 8;
    intVal |= lm.getElement_log(i) & 0xFF;

    if((i == 8) && (lineType == 2)) // temperature
    {
        t=((intVal * .98 - 3840)/100);
        logOut.println("Temperature: " + t);
        dataRow[1] = Double.toString(t);
    }

    else if((i==12) && (lineType==1)) // battery voltage
    {
        logOut.println("Battery Voltage: " + intVal*10);
        dataRow[2] = Double.toString(intVal*10);
    }

    else if((i==10) && (lineType == 2)) // humidity
    {
        if (t == 0) t = 25; // use this as a default value
        double h; // temporary humidity value
        h= 0.0405 * intVal - 4 - intVal * intVal*0.0000028;
        h= (t-25) * (0.01 + 0.00128 * intVal) + h;
        if (h > 100) h = 100;
        logOut.println("Humidity: " + h);
        dataRow[3] = Double.toString(h);
    }

    else if((i==12) && (lineType == 2)) // data
    {
        logOut.println("Data: " + (12500 * (intVal/2048.0 -1)));
    }
}

```

```

        dataRow[4] = Double.toString((12.500 * (intVal/2048.0 - 1)));
    }

    else if(i==4) // seconds into test
    {
        int secs = 0;
        secs |= lm.getElement.log(i+3) & 0xFF;
        secs <<= 8;
        secs |= lm.getElement.log(i+2) & 0xFF;
        secs <<= 8;
        secs |= lm.getElement.log(i+1) & 0xFF;
        secs <<= 8;
        secs |= lm.getElement.log(i) & 0xFF;
        logOut.println("seconds into test: " + intVal);
        dataRow[0] = Integer.toString(intVal);
    }

    else if((i==8) && (lineType == 1)) // samples remaining
    {
        logOut.println(intVal + " samples");
        dataRow[6] = Integer.toString(intVal);
    }

    else if((i==10) && (lineType == 1)) // samples
    {
        logOut.println(intVal + " samples total");
        dataRow[5] = Integer.toString(intVal);
    }

    else if(i==0) // timestamp
    {
        timestamp=0;
        timestamp |= lm.getElement.log(i+3) & 0xFF;
        timestamp <<= 8;
        timestamp |= lm.getElement.log(i+2) & 0xFF;
        timestamp <<= 8;
        timestamp |= lm.getElement.log(i+1) & 0xFF;
        timestamp <<= 8;
        timestamp |= lm.getElement.log(i) & 0xFF;
        logOut.println("Time: " + timestamp);
        dataRow[7] = Long.toString((long) timestamp);
    }
} // end for
logOut.println("----END----");
logOut.println();
logOut.flush();
try
{
    {
        los.close();
        logOut.close();
    }
}
catch (IOException e)
{

```

```
        System.err.println("Error closing log file " + e);
        e.printStackTrace();
    }
} // end linetype else

/*
synchronized (this) {
    read_log_done = true;
    this.notifyAll();
}
*/
}
}
```



## CHAPTER 2

### **MICA2-Based Wireless ACM Version 2 Software: Modification of XMDA300**

This appendix contains all software used to program the MICA2 motes for Version 2 of the MICA2-based wireless ACM system. This appendix is organized by software directory and only the modified files are included.

## 1. xbow/apps/XMesh/XMDA300 directory

```
/*          tab:4
 * IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.
By
 * downloading, copying, installing or using the software you agree to
 * this license. If you do not agree to this license, do not download,
 * install, copy or use the software.
 *
 * Intel Open Source License
 *
 * Copyright (c) 2002 Intel Corporation
 * All rights reserved.
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * Neither the name of the Intel Corporation nor the names of its
 * contributors may be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INTEL OR ITS
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * @author Leah Fera, Martin Turon, Jaidev Prabhu
 *
 * $Id: XMDA300M.nc,v 1.4 2005/01/11 05:21:35 husq Exp $
 */
```

```
/*
 * *****
 *
 * - Tests the MDA300 general prototyping card
 *   (see Crossbow MTS Series User Manual)
 * - Read and control all MDA300 signals:
 *   ADC0, ADC1, ADC2, ADC3,...ADC11 inputs, DIO 0-5,
 *   counter, battery, humidity, temp
 *
 * -----
 * Output results through mica2 uart and radio.
```

```

* Use xlisten.exe program to view data from either port:
* uart: mount mica2 on mib510 with MDA300
*       (must be connected or now data is read)
*       connect serial cable to PC
*       run xlisten.exe at 57600 baud
* radio: run mica2 with MDA300,
*         run another mica2 with TOSBASE
*         run xlisten.exe at 56K baud
* LED: the led will be green if the MDA300 is connected to the mica2 and
*       the program is running (and sending out packets). Otherwise it is red.
*-----
* Data packet structure:
*
* PACKET #1 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 1
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : analog adc data Ch.0
* msg->data[6,7] : analog adc data Ch.1
* msg->data[8,9] : analog adc data Ch.2
* msg->data[10,11] : analog adc data Ch.3
* msg->data[12,13] : analog adc data Ch.4
* msg->data[14,15] : analog adc data Ch.5
* msg->data[16,17] : analog adc data Ch.6
*
* PACKET #2 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 2
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : analog adc data Ch.7
* msg->data[6,7] : analog adc data Ch.8
* msg->data[8,9] : analog adc data Ch.9
* msg->data[10,11] : analog adc data Ch.10
* msg->data[12,13] : analog adc data Ch.11
* msg->data[14,15] : analog adc data Ch.12
* msg->data[16,17] : analog adc data Ch.13
*
*
* PACKET #3 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 3
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : digital data Ch.0
* msg->data[6,7] : digital data Ch.1
* msg->data[8,9] : digital data Ch.2
* msg->data[10,11] : digital data Ch.3
* msg->data[12,13] : digital data Ch.4

```

## 20 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
* msg->data[14,15] : digital data Ch.5
*
* PACKET #4 (of 4)
* -----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 4
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : batt
* msg->data[6,7] : hum
* msg->data[8,9] : temp
* msg->data[10,11] : counter
* msg->data[14] : msg4_status (debug)
*
*****/

// include sensorboard.h definitions from tos/mda300 directory
#include "appFeatures.h"
includes XCommand;

includes sensorboard;
module XMDA300M
{
    provides interface StdControl;

    uses {
    interface Leds;

    interface Send;
    interface RouteControl;
#ifdef XMESHSYNC
    interface Receive as DownTree;
#endif
    interface XCommand;

    //Sampler Communication
    interface StdControl as SamplerControl;
    interface Sample;

    //Timer
    interface Timer;

    //relays
    interface Relay as relay_normally_closed;
    interface Relay as relay_normally_open;

    //support for plug and play
    command result_t PlugPlay();

#if FEATURE_UART_SEND
```

```

    interface SendMsg as SendUART;
    command result_t PowerMgrEnable();
    command result_t PowerMgrDisable();
#endif
    }
}

implementation
{
#define ANALOG_SAMPLING_TIME    90
#define DIGITAL_SAMPLING_TIME  100
#define MISC_SAMPLING_TIME      110
#define MPKRATE    30720 // fire twice per minute
#define MPK_TICKS  36 // 18 minutes between samples
#define MPK_QUICK_MESH_COUNT 60 // MPK take 60 samples at a high speed
    // before lowering rate

#define ANALOG_SEND_FLAG  1
#define DIGITAL_SEND_FLAG 1
#define MISC_SEND_FLAG    1
#define ERR_SEND_FLAG     1

#define PACKET_FULL 0x1C1 // MPK Temp, Hum, Batt, ADC0
#define 0
#define PACKET_FULL 0x1FF
#endif

#define MSG_LEN 29 // excludes TOS header, but includes xbow header

    enum {
    PENDING = 0,
    NO_MSG = 1
    };

    enum {
    MDA300_PACKET1 = 1,
    MDA300_PACKET2 = 2,
    MDA300_PACKET3 = 3,
    MDA300_PACKET4 = 4,
    MDA300_ERR_PACKET = 0xf8
    };

/*****
    enum {
    SENSOR_ID = 0,
    PACKET_ID,
    NODE_ID,
    RESERVED,
    DATA_START
    } XPacketDataEnum;
*****/
/* Messages Buffers */

```

22 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```

uint32_t timer_rate;
bool sleeping; // application command state
bool sending_packet;
uint16_t seqno;
XDataMsg *tmppack;

TOS_Msg packet;
TOS_Msg msg_send_buffer;
TOS_MsgPtr msg_ptr;

int packetCount = 0; // MPK
int timerTicks = 0; // MPK
int maxTicks = 2; // MPK Broadcast every minute
// until quckMeshCount > MPK_QUICK_MESH_COUNT
int quickMeshCount = 0; // MPK

uint16_t msg_status, pkt_full;
char test;

int8_t record[25];

static void initialize()
{
    atomic
    {
        sleeping = FALSE;
        sending_packet = FALSE;
        //timer_rate = XSENSOR_SAMPLE_RATE; // ORIGINAL LINE
        timer_rate = MPK_RATE; // MPK
    }
}

/*****
* Initialize the component. Initialize Leds
*
*****/
command result_t StdControl.init() {

    uint8_t i;

    call Leds.init();

    atomic {
        msg_ptr = &msg_send_buffer;
        //sending_packet = FALSE;
    }
    msg_status = 0;
    pkt_full = PACKET_FULL;

    MAKE_BAT_MONITOR_OUTPUT(); // enable voltage ref power pin as output
    MAKE_ADC_INPUT(); // enable ADC7 as input

```

```

// usart1 is also connected to external serial flash
// set usart1 lines to correct state
// TOSH_MAKE_FLASH_SELECT_OUTPUT();
TOSH_MAKE_FLASH_OUT_OUTPUT();           //tx output
TOSH_MAKE_FLASH_CLK_OUTPUT();          //usart clk
// TOSH_SET_FLASH_SELECT_PIN();

    call SamplerControl.init();
    initialize();
    return SUCCESS;

//return rcombine(call SamplerControl.init(), call CommControl.init());
}

/*****
* Start the component. Start the clock. Setup timer and sampling
*
*****/
command result_t StdControl.start() {

    call SamplerControl.start();

    if(call PlugPlay())
    {

        call Timer.start(TIMER_REPEAT, timer_rate);

        //channel parameteres are irrelevant

        record[14] = call Sample.getSample(0,
TEMPERATURE, MISC_SAMPLING_TIME, SAMPLER_DEFAULT);

        record[15] = call Sample.getSample(0,
HUMIDITY, MISC_SAMPLING_TIME, SAMPLER_DEFAULT);

        record[16] = call Sample.getSample(0,
BATTERY, MISC_SAMPLING_TIME, SAMPLER_DEFAULT);

        //start sampling channels. Channels 7-10 with averaging since
//they are more percise.
//hannels 3-6 make active excitation
/* MPK
        record[0] = call Sample.getSample(0,
ANALOG, ANALOG_SAMPLING_TIME, SAMPLER_DEFAULT );

        record[1] = call Sample.getSample(1,

```

## 24 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT );

    record [2] = call Sample.getSample(2,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);
MPK */

        /* MPK
    record [3] = call Sample.getSample(3,
ANALOG,ANALOG.SAMPLING.TIME,
SAMPLER.DEFAULT | EXCITATION.33 | DELAY.BEFORE.MEASUREMENT);

    record [4] = call Sample.getSample(4,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);

    record [5] = call Sample.getSample(5,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);

    record [6] = call Sample.getSample(6,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);

MPK */

    record [7] = call Sample.getSample(7,
ANALOG,ANALOG.SAMPLING.TIME,
AVERAGEFOUR | EXCITATION.25 | DELAY.BEFORE.MEASUREMENT );

/* MPK
    record [8] = call Sample.getSample(8,
ANALOG,ANALOG.SAMPLING.TIME,AVERAGEFOUR | EXCITATION.25);

    record [9] = call Sample.getSample(9,
ANALOG,ANALOG.SAMPLING.TIME,AVERAGEFOUR | EXCITATION.25);
MPK */

/* MPK
    record [10] = call Sample.getSample(10,
ANALOG,ANALOG.SAMPLING.TIME,AVERAGEFOUR | EXCITATION.25);

    record [11] = call Sample.getSample(11,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);

    record [12] = call Sample.getSample(12,
ANALOG,ANALOG.SAMPLING.TIME,SAMPLER.DEFAULT);

    record [13] = call Sample.getSample(13,
ANALOG,ANALOG.SAMPLING.TIME,
SAMPLER.DEFAULT | EXCITATION.50 | EXCITATION.ALWAYS.ON);
MPK */

/* // MPK
//digital chennels as accumulative counter
```

```

    record[17] = call Sample.getSample(0,
DIGITAL,DIGITAL_SAMPLING_TIME,
RESET_ZERO_AFTER_READ | FALLING_EDGE);

    record[18] = call Sample.getSample(1,
DIGITAL,DIGITAL_SAMPLING_TIME,RISING_EDGE | EVENT);

    record[19] = call Sample.getSample(2,
DIGITAL,DIGITAL_SAMPLING_TIME,SAMPLER_DEFAULT | EVENT);

    record[20] = call Sample.getSample(3,
DIGITAL,DIGITAL_SAMPLING_TIME,FALLING_EDGE);

    record[21] = call Sample.getSample(4,
DIGITAL,DIGITAL_SAMPLING_TIME,RISING_EDGE);

    record[22] = call Sample.getSample(5,
DIGITAL,DIGITAL_SAMPLING_TIME,RISING_EDGE | EEPROM_TOTALIZER);

    //counter channels for frequency measurement, will reset to zero.

    record[23] = call Sample.getSample(0,
COUNTER,MISC_SAMPLING_TIME,
RESET_ZERO_AFTER_READ | RISING_EDGE);
    call Leds.greenOn();
MPK */
}

else {
    call Leds.redOn();
}

return SUCCESS;

}

/*****
* Stop the component.
*
*****/

    command result_t StdControl.stop() {

call SamplerControl.stop();

return SUCCESS;

}

/*****

```

## 26 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```

* Task to transmit radio message
* NOTE that data payload was already copied from the corresponding UART packet
*****/
task void send_radio_msg()
{
    uint8_t i;
    uint16_t len;
    XDataMsg *data;
    if(sending_packet)
        return;
    atomic sending_packet=TRUE;
// Fill the given data buffer.
    data = (XDataMsg*) call Send.getBuffer(msg_ptr, &len);
    tmppack=(XDataMsg *)packet.data;
    for (i = 0; i <= sizeof(XDataMsg)-1; i++)
        ((uint8_t*)data)[i] = ((uint8_t*)tmppack)[i];

    data->xmeshHeader.packet_id = 6;
    data->xmeshHeader.board_id = SENSOR_BOARD_ID;
    data->xmeshHeader.node_id = TOS_LOCAL_ADDRESS;
    data->xmeshHeader.parent = call RouteControl.getParent();

#if FEATURE_UART_SEND
    if (TOS_LOCAL_ADDRESS != 0) {
        call Leds.yellowOn();
        call PowerMgrDisable();
        TOSH_uwait(1000);
        if (call SendUART.send(TOS_UART_ADDR,
sizeof(XDataMsg),msg_ptr) != SUCCESS)
        {
            atomic sending_packet = FALSE;
            call Leds.greenToggle();
            call PowerMgrEnable();
        }
    }
    else
#endif
    {
        // Send the RF packet!
        call Leds.yellowOn();
        if (call Send.send(msg_ptr, sizeof(XDataMsg)) != SUCCESS) {
            atomic sending_packet = FALSE;
            call Leds.yellowOn();
            call Leds.greenOff();
        }
    }
}
}

```

```

#if FEATURE_UART_SEND
/**
 * Handle completion of sent UART packet.
 *
 * @author    Martin Turon
 * @version   2004/7/21      mturon      Initial revision
 */
event result_t SendUART.sendDone(TOS_MsgPtr msg, result_t success)
{
    //      if (msg->addr == TOS_UART_ADDR) {
    atomic msg_ptr = msg;
    msg_ptr->addr = TOS_BCAST_ADDR;

    if (call Send.send(msg_ptr, sizeof(XDataMsg)) != SUCCESS) {
    atomic sending_packet = FALSE;
    call Leds.yellowOff();
    }

    if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
    call PowerMgrEnable();

    //}
    return SUCCESS;
}
#endif

/**
 * Handle completion of sent RF packet.
 *
 * @author    Martin Turon
 * @version   2004/5/27      mturon      Initial revision
 */
event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
{
    atomic {
    msg_ptr = msg;
    sending_packet = FALSE;
    }
    call Leds.yellowOff();
}

#if FEATURE_UART_SEND
    if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
    call PowerMgrEnable();
#endif

    return SUCCESS;
}

/**
 * Handle a single dataReady event for all MDA300 data types.
 *

```

```

* @author    Leah Fera , Martin Turon
*
* @version   2004/3/17          leahfera    Intial revision
* @n        2004/4/1          mturon      Improved state machine
*/
event result_t
Sample.dataReady(uint8_t channel , uint8_t channelType , uint16_t data)
{
    uint8_t i;

    switch (channelType) {
case ANALOG:
    switch (channel) {
// MSG 1 : first part of analog channels (0-6)
// case 0: // Original line MPK
case 7: // MPK just cram ADC7 into ADC0's spot
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc0 =data ;
        atomic {msg_status|=0x01;}
        break;

case 1:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc1 =data ;
        atomic {msg_status|=0x02;}
        break;

case 2:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc2 =data ;
        atomic {msg_status|=0x04;}
        break;

default:
        break;
    } // case ANALOG (channel)
    break;

case DIGITAL:
    switch (channel) {
case 0:
        tmppack=(XDataMsg *)packet.data;

        tmppack->xData.datap6.dig0=data;
        atomic {msg_status|=0x08;}
        break;

case 1:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.dig1=data;
        atomic {msg_status|=0x10;}
    }
}

```

```

        break;

    case 2:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.dig2=data;
        atomic {msg_status|=0x20;}
        break;

    default:
        break;
    } // case DIGITAL (channel)
    break;

case BATTERY:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.vref =data ;
    atomic {msg_status|=0x40;}
    break;

case HUMIDITY:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.humid =data ;
    atomic {msg_status|=0x80;}
    break;

case TEMPERATURE:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.humtemp =data ;
    atomic {msg_status|=0x100;}
    break;

    default:
        break;

    } // switch (channelType)

    if (sending_packet)
        return SUCCESS;

if (msg_status == pkt_full) {
    msg_status = 0;
packetCount++; // MPK
if(packetCount >= 3) // MPK
{ // MPK
    call SamplerControl.stop(); // MPK
    packetCount = 0; // MPK

    // get rid of timer ticks accumulated
    // while sampling MPK
    atomic {timerTicks = 0;} // MPK
} // MPK

```

### 30 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```

    if(packetCount > 1) // MPK first ADC reading always is bad
        post send_radio_msg();
    }

    return SUCCESS;
}

```

```

/*****
 * Timer Fired -
 *
 *****/

```

```

    event result_t Timer.fired() {
/*      MPK
        if (sending_packet)
            return SUCCESS;          //don't overrun buffers
        if (test != 0) {
            test=0;
            call relay_normally_closed.toggle();
            call Leds.greenOn();
        }
        else {
            test=1;
            call relay_normally_open.toggle();
            call Leds.greenOn();
        }
    }
}

```

MPK \*/

```

// MPK if enough ticks, turn on the MDA300 and do a new
// MPK round of getSample calls
atomic{timerTicks ++;} // MPK

```

```

if(quickMeshCount > MPK_QUICK_MESH_COUNT)// MPK
{
    // MPK
    atomic{maxTicks = MPK_TICKS;} // MPK
}
// MPK
// MPK

```

```

// call RouteControl.manualUpdate(); // MPK

```

```

if ((timerTicks < maxTicks) && (maxTicks > 0)) // MPK
    return SUCCESS; // MPK

```

```

atomic {timerTicks = 0;} // MPK
atomic{quickMeshCount++;} // MPK
call SamplerControl.init(); // MPK Is this necessary?
call SamplerControl.start(); // MPK

```

```

if(call PlugPlay()) // MPK
{
    // MPK
    record[14] = call Sample.getSample(0,
TEMPERATURE,MISC_SAMPLING_TIME,SAMPLER_DEFAULT); // MPK
    record[15] = call Sample.getSample(0,

```

```

HUMIDITY,MISC.SAMPLING.TIME,SAMPLER.DEFAULT); // MPK
    record[16] = call Sample.getSample(0,
BATTERY,MISC.SAMPLING.TIME,SAMPLER.DEFAULT); // MPK
    record[7] = call Sample.getSample(7,
ANALOG,ANALOG.SAMPLING.TIME,
AVERAGEFOUR | EXCITATION_25 | DELAY_BEFORE_MEASUREMENT
); // MPK
}

```

```

return SUCCESS;

```

```

}

```

```

/**

```

```

* Handles all broadcast command messages sent over network.
*
* NOTE: Bcast messages will not be received if seq_no is not properly
*       set in first two bytes of data payload. Also, payload is
*       the remaining data after the required seq_no.
*

```

```

* @version 2004/10/5 mturon Initial version
*/

```

```

event result_t XCommand.received(XCommandOp *opcode) {

```

```

    switch (opcode->cmd) {
case XCOMMAND.SETRATE:
    // Change the data collection rate.
    /* MPK
    timer_rate = opcode->param.newrate;
    call Timer.stop();
    call Timer.start(TIMER_REPEAT, timer_rate);
MPK */
    atomic{maxTicks = opcode->param.newrate;} // MPK
    atomic{timerTicks = 0;} // MPK
    break;

```

```

case XCOMMAND.SLEEP:
    // Stop collecting data, and go to sleep.
    sleeping = TRUE;
    call Timer.stop();
    call Leds.set(0);
    break;

```

```

case XCOMMAND.WAKEUP:
    // Wake up from sleep state.
    if (sleeping) {
    initialize();
    call Timer.start(TIMER_REPEAT, timer_rate);
    sleeping = FALSE;
    }
    break;

```

```

case XCOMMAND.RESET:

```

## 32 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
        // Reset the mote now.
        break;

    case XCOMMANDACTUATE:
        break;

    default:
        break;
    }

    return SUCCESS;
}

#ifdef XMESHSYNC
task void SendPing() {
    XDataMsg *pReading;
    uint16_t Len;

    if ((pReading = (XDataMsg *) call Send.getBuffer(msg_ptr, &Len)) {
        pReading->xmeshHeader.parent = call RouteControl.getParent();
        if ((call Send.send(msg_ptr, sizeof(XDataMsg)) != SUCCESS)
atomic sending_packet = FALSE;
    }

}

    event TOS_MsgPtr DownTree.receive(TOS_MsgPtr pMsg,
void* payload, uint16_t payloadLen) {

        if (!sending_packet) {
            call Leds.yellowToggle();
            atomic sending_packet = TRUE;
            post SendPing(); // pMsg->XXX;
        }
        return pMsg;
    }
#endif
}
```

**2. xbow/tos/sensorboards/mda300 directory**

```
/*
 *
 * Copyright (c) 2003 The Regents of the University of California. All
 * rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * - Neither the name of the University nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ‘‘AS IS’’
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
 * Authors:   Mohammad Rahimi mhr@cens.ucla.edu
 * History:   created 08/14/2003
 * update at 11/14/2003
 *
 *
 * driver for ADS7828EB on mda300ca
 *
 */
```

```
module IBADCM
{
  provides {
    interface StdControl;
    interface ADConvert[uint8_t port];
    interface SetParam[uint8_t port];
    interface Power as EXCITATION25;
    interface Power as EXCITATION33;
    interface Power as EXCITATION50;
  }
  uses interface I2CPacket;
  uses interface Leds;
  uses interface StdControl as I2CPacketControl;
```

### 34 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
    uses interface Timer as PowerStabalizingTimer;
    uses interface StdControl as SwitchControl;
    uses interface Switch;
}
implementation
{
    enum {IDLE, PICK_CHANNEL, GET_SAMPLE, CONTINUE_SAMPLE , START_CONVERSION_PROCESS};

#define VOLTAGE_STABLE_TIME 50
//Time it takes for the supply voltage to be stable enough

#define MAX_ANALOG_CHANNELS 13
#define MAX_CHANNELS MAX_ANALOG_CHANNELS + 1
//The last channel is not an analog channel
//but we keep it only for the sake of exciation.

    /*Note:we do not do anything inside async part so all parts are synchronous and
    there is no synchronization hazard.Now ADC runs in the round-robin fashin so it
    is fair.*/

    char state;          /* current state of the i2c request */
    uint16_t value;      /* value of the incoming ADC reading */
    uint8_t chan;
    uint8_t param[MAX_CHANNELS]; /*we reserve last param for excitation of digital channels*/
    uint16_t adc_bitmap;
    int8_t conversionNumber;
//Note "condition" should be a global variable.Since It is passed by address to I2CPacketM.nc
//should be valid even out of the scope of that function since I2CPacketM.nc uses it by its
    uint8_t condition;  // set the condition command byte.

    //set of bitwise functions
#define testbit(var, bit) ((var) & (1 <<(bit)))          //if zero then return zero and if one n
#define setbit(var, bit) ((var) |= (1 << (bit)))
#define clrbit(var, bit) ((var) &= ~(1 << (bit)))

    //The excitation circuits
#define FIVE_VOLT_ON() TOSH_SET_PW5_PIN()
#define FIVE_VOLT_OFF() TOSH_CLR_PW5_PIN()

#define THREE_VOLT_ON() TOSH_SET_PW3_PIN()
#define THREE_VOLT_OFF() TOSH_CLR_PW3_PIN()

#define TURN_VOLTAGE_BUFFER_ON() TOSH_SET_PW2_PIN()
#define TURN_VOLTAGE_BUFFER_OFF() TOSH_CLR_PW2_PIN()

#define VOLTAGE_BOOSTER_ON() TOSH_CLR_PW1_PIN()
#define VOLTAGE_BOOSTER_OFF() TOSH_SET_PW1_PIN()

    //The instrumentation amplifier
#define TURN_AMPLIFIERS_ON() TOSH_SET_PW6_PIN()
#define TURN_AMPLIFIERS_OFF() TOSH_CLR_PW6_PIN()
```

```

/*declaration of function convert*/
result_t convert();

void setExcitation()
{
    VOLTAGEBOOSTER_ON();
    if (param[chan] & EXCITATION_25 ) TURN_VOLTAGEBUFFER_ON();
    if (param[chan] & EXCITATION_33 )
    {
        THREE_VOLT_ON();
    }
    if (param[chan] & EXCITATION_50)
    {
        FIVE_VOLT_ON();
    }
}

void resetExcitation()
{
    uint8_t i;
    uint8_t flag25=0,flag33=0,flag50=0;
    for (i=0 ; i < MAX_CHANNELS ; i++)
    {
        if (param[i] & EXCITATION_ALWAYS_ON)
        {
            if (param[i] & EXCITATION_25) flag25=1;
            if (param[i] & EXCITATION_33) flag33=1;
            if (param[i] & EXCITATION_50) flag50=1;
        }
    }
    if (flag25==0) TURN_VOLTAGEBUFFER_OFF();
    if (flag33==0) THREE_VOLT_OFF();
    if (flag50==0) FIVE_VOLT_OFF();
    if (!flag25 && !flag33 && !flag50)
    {
        VOLTAGEBOOSTER_OFF();
    }
}

command void EXCITATION25.on()
{
    param[MAX_CHANNELS - 1] |= EXCITATION_25;
    param[MAX_CHANNELS - 1] |= EXCITATION_ALWAYS_ON;
    VOLTAGEBOOSTER_ON();
    TURN_VOLTAGEBUFFER_ON();
}

command void EXCITATION25.off()
{
    param[MAX_CHANNELS - 1] &= !EXCITATION_25;
    if (state == IDLE) resetExcitation();
}
//otherwise the fuction will be called at the end of conversion

```

```

    }
    command void EXCITATION33.on()
    {
        param[MAX_CHANNELS - 1] |= EXCITATION_33;
        param[MAX_CHANNELS - 1] |= EXCITATION_ALWAYS_ON;
        VOLTAGE_BOOSTER_ON();
        THREE_VOLT_ON();
    }
    command void EXCITATION33.off()
    {
        param[MAX_CHANNELS - 1] &= !EXCITATION_33;
        if(state == IDLE) resetExcitation();
//otherwise the fuction will be called at the end of conversion
    }
    command void EXCITATION50.on()
    {
        param[MAX_CHANNELS - 1] |= EXCITATION_50;
        param[MAX_CHANNELS - 1] |= EXCITATION_ALWAYS_ON;
        VOLTAGE_BOOSTER_ON();
        FIVE_VOLT_ON();
    }
    command void EXCITATION50.off()
    {
        param[MAX_CHANNELS-1] &= !EXCITATION_50;
        if(state == IDLE) resetExcitation();
//otherwise the fuction will be called at the end of conversion
    }

    void setNumberOfConversions()
    {
        conversionNumber = 1;
        if(param[chan] & AVERAGEFOUR ) conversionNumber = 4;
        if(param[chan] & AVERAGEEIGHT ) conversionNumber = 8;
        if(param[chan] & AVERAGESIXTEEN) conversionNumber = 16;
        return;
    }

    command result_t StdControl.init() {
        int i;
        atomic{
            state = IDLE;
            adc_bitmap=0;
            for(i=0; i < MAX_CHANNELS ; i++) param[i]=0x00;
        }
        call I2CPacketControl.init();
        call SwitchControl.init();
        TOSHMAKEPW2.OUTPUT();
        TOSHMAKEPW4.OUTPUT();
        TOSHMAKEPW5.OUTPUT();
        TOSHMAKEPW6.OUTPUT();
        TURN_AMPLIFIERS.OFF();
        VOLTAGE_BOOSTER.OFF();
        FIVE_VOLT.OFF();
    }

```

```

THREEVOLT.OFF();
TURN_VOLTAGEBUFFER.OFF();
return SUCCESS;
}

command result_t StdControl.start() {
    call SwitchControl.start();
    return SUCCESS;
}

command result_t StdControl.stop() {
    uint8_t ADCaddress; // MPK
    ADCaddress=8; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=12; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=9; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=13; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=10; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=14; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=11; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=15; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=0; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=1; // MPK
    ADCaddress=(ADCaddress << 4) & 0xf0; // MPK
    ADCaddress=ADCaddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCaddress), 0x03); // MPK
    ADCaddress=2; // MPK

```

### 38 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```

    ADCAddress=(ADCAddress << 4) & 0xf0; // MPK
    ADCAddress=ADCAddress | 0x03; // MPK
    call I2CPacket.writePacket(1, (char*)&ADCAddress), 0x03); // MPK

    TURN_AMPLIFIERS_OFF(); // MPK
    VOLTAGE_BOOSTER_OFF(); // MPK
    FIVE_VOLT_OFF(); // MPK
    THREE_VOLT_OFF(); // MPK
    TURN_VOLTAGE_BUFFER_OFF(); // MPK

    call SwitchControl.stop();
    return SUCCESS;
}

command result_t SetParam.setParam[uint8_t id](uint8_t mode){
    param[id]=mode;
    return SUCCESS;
}

default event result_t ADConvert.dataReady[uint8_t id](uint16_t data) {
    return SUCCESS;
}

task void adc_get_data()
{
    uint8_t myIndex;
    uint8_t count;
    uint16_t my_bitmap;
    if(state != IDLE) return; //That means the component is busy in a
        //conversion process.When conversion done
        // either successfull or fail it is
        // gauranteed that this task will be
        // posted so we can safely return.
    value=0;
    state=START_CONVERSION_PROCESS;
    atomic { my_bitmap = adc_bitmap; }
    //it gaurantees a round robin fair scheduling of ADC conversions.
    count=0;
    myIndex=chan+1;
    if(myIndex > MAX_ANALOG_CHNNELS) myIndex=0;
    while(!testbit(my_bitmap, myIndex))
    {
        myIndex++;
        if(myIndex > MAX_ANALOG_CHNNELS) myIndex=0;
        count++;
        if(count > MAX_ANALOG_CHNNELS) {state=IDLE; return; } //no one waiting for conversion
    }

    chan=myIndex;

    setExcitation();
}

```

```

setNumberOfConversions();
//if among the instrumentation channels we set the MUX
if(chan == 7 || chan==8 || chan==9 || chan==10)
{
    char muxChannel;
    TURN_AMPLIFIERS_ON();
    switch (chan) {
    default: // should never happen
    case 7:
        muxChannel = MUX_CHANNELSEVEN;
        break;
    case 8:
        muxChannel = MUX_CHANNELEIGHT;
        break;
    case 9:
        muxChannel = MUX_CHANNELNINE;
        break;
    case 10:
        muxChannel = MUX_CHANNELTEN;
        break;
    };
    if ((call Switch.setAll(muxChannel)) == FAIL)
    {
        // Can not select channel
        state = IDLE;
        TURN_AMPLIFIERS_OFF();
        post_adc_get_data();
        resetExcitation();
    }
}
else {
    //If the conversions happens fast there is no need to
    //wait for settling of the power supply,
    //note that power supply should be set ON by user using the excitation command
    if(param[chan] & DELAY_BEFORE_MEASUREMENT) {
        call PowerStabalizingTimer.start(TIMER_ONE_SHOT, VOLTAGE_STABLE_TIME);
    }
    else {
        convert();
    }
}
}

result_t convert() {
    if (state == START_CONVERSION_PROCESS || state == CONTINUE_SAMPLE)
    {
        state = PICK_CHANNEL;
        // figure out which channel is to be set
        switch (chan) {
        default: // should never happen
        case 0:
            condition = 8;
            break;

```

00 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
    case 1:
        condition = 12;
        break;
    case 2:
        condition = 9;
        break;
    case 3:
        condition = 13;
        break;
    case 4:
        condition = 10;
        break;
    case 5:
        condition = 14;
        break;
    case 6:
        condition = 11;
        break;
    case 7:
    case 8:
    case 9:
    case 10:
        //these channels all use ADC channel 7 and multiplex it.
        condition = 15;
        break;
    case 11:
        condition = 0;
        break;
    case 12:
        condition = 1;
        break;
    case 13:
        condition = 2;
        break;
    }
}
// shift the channel and single-ended input bits over
condition = (condition << 4) & 0xf0;
condition = condition | 0x0f;
//tell the ADC to start converting
if ((call I2CPacket.writePacket(1, (char*)&condition), 0x03)) == FAIL)
{
    state = IDLE;
    post adc_get_data();
    resetExcitation();
    return FALSE;
}
return SUCCESS;
TURN_AMPLIFIERS_OFF();
}

// get a single reading from id we
command result_t ADConvert.getData[uint8_t id]() {
```

```

if(id>13) return FAIL; //should never happen unless wiring is wrong.
atomic {
    setbit(adc_bitmap, id);
}
post adc_get_data();
return SUCCESS;
}

//Setting the MUX has been done.
event result_t Switch.setAllDone(bool r)
{
    if(!r) {
        state=IDLE;
        TURN_AMPLIFIERS_OFF();
        post adc_get_data();
        resetExcitation();
        return FAIL;
    }

    //If the conversions happens fast there is no need to wait for settling of
    //the power supply,note that power supply should be set ON by user using
    //the excitation command

    if(param[chan] & DELAY_BEFORE_MEASUREMENT) {
        call PowerStabalizingTimer.start(TIMER_ONE_SHOT, VOLTAGE_STABLE_TIME);
        return SUCCESS;
    }
    else {
        return convert();
    }
    return SUCCESS;
}

event result_t PowerStabalizingTimer.fired() {
    return convert();
}

/* not yet implemented */
command result_t ADConvert.getContinuousData[uint8_t id]() {
    return FAIL;
}

event result_t I2CPacket.readPacketDone(char length, char* data) {
    if (length != 2)
    {
        state = IDLE;
        TURN_AMPLIFIERS_OFF();
        atomic { clrbit(adc_bitmap, chan); }
        post adc_get_data();
        signal ADConvert.dataReady[chan](ADC_ERROR);
        resetExcitation();
    }
}

```

02 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```

        return FAIL;
    }

    if (state == GET_SAMPLE)
    {
        value += (data[1] & 0xff) + ((data[0] << 8) & 0x0f00);
        conversionNumber--;
        //value = (data[0] << 8) & 0x0f00;
        //value += (data[1] & 0xff);
        if (conversionNumber==0) {
            state = IDLE;
            if(param[chan] & AVERAGE_SIXTEEN)
                value = ((value+8) >>4) & 0x0fff; //the addition is for more percision
            else if(param[chan] & AVERAGE_EIGHT )
                value = ((value+4) >>3) & 0x0fff; //the addition is for more percision
            else if(param[chan] & AVERAGE_FOUR )
                value = ((value+2) >>2) & 0x0fff; //the addition is for more percision
            // else { //do nothing since no averaging}
            TURN_AMPLIFIERS_OFF();
            atomic { clrbit(adc_bitmap,chan); }
            post adc_get_data();
            signal ADConvert.dataReady[chan](value);
            resetExcitation();
        }
        else {
            state = CONTINUE_SAMPLE;
            convert();
        }
    }
    return SUCCESS;
}

event result_t I2CPacket.writePacketDone(bool result) {
    if (!result)
    {
        state = IDLE;
        TURN_AMPLIFIERS_OFF();
        atomic { clrbit(adc_bitmap,chan); }
        post adc_get_data();
        signal ADConvert.dataReady[chan](ADC_ERROR);
        resetExcitation();
        return FAIL;
    }

    if (state == PICK_CHANNEL)
    {
        state = GET_SAMPLE;
        if ((call I2CPacket.readPacket(2, 0x03)) == 0)
        {
            //reading from the bus failed
            state = IDLE;
            post adc_get_data();
            resetExcitation();
        }
    }
}

```

```
        return FAIL;
    }
}
return SUCCESS;
}

event result_t Switch.getDone(char val)
{
    return SUCCESS;
}

event result_t Switch.setDone(bool r)
{
    return SUCCESS;
}
}
```

### 3. xbow/tos/sensorboards/mda300 directory

```

/*
 *
 * Copyright (c) 2003 The Regents of the University of California. All
 * rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * - Neither the name of the University nor the names of its
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ‘‘AS IS’’
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
 * Authors:   Mohammad Rahimi mhr@cens.ucla.edu
 * History:   created 08/14/2003
 * history:   modified 11/14/2003
 *
 */

```

```

module SamplerM
{
    provides interface StdControl as SamplerControl;
    provides interface Sample;
    provides command result_t PlugPlay();

    uses {
        interface Leds;
        interface Timer as SamplerTimer;

        //analog channels
        interface StdControl as IBADCcontrol;
        interface ADConvert as ADC0;
        interface ADConvert as ADC1;
    }
}

```

```
interface ADConvert as ADC2;
interface ADConvert as ADC3;
interface ADConvert as ADC4;
interface ADConvert as ADC5;
interface ADConvert as ADC6;
interface ADConvert as ADC7;
interface ADConvert as ADC8;
interface ADConvert as ADC9;
interface ADConvert as ADC10;
interface ADConvert as ADC11;
interface ADConvert as ADC12;
interface ADConvert as ADC13;
//ADC parameters
interface SetParam as SetParam0;
interface SetParam as SetParam1;
interface SetParam as SetParam2;
interface SetParam as SetParam3;
interface SetParam as SetParam4;
interface SetParam as SetParam5;
interface SetParam as SetParam6;
interface SetParam as SetParam7;
interface SetParam as SetParam8;
interface SetParam as SetParam9;
interface SetParam as SetParam10;
interface SetParam as SetParam11;
interface SetParam as SetParam12;
interface SetParam as SetParam13;

//health channels temp,humidity,voltage
interface StdControl as BatteryControl;
//interface ADConvert as Battery;
interface ADC as Battery;
interface StdControl as TempHumControl;
interface ADConvert as Temp;
interface ADConvert as Hum;

//digital and relay channels
interface StdControl as DioControl;
interface Dio as Dio0;
interface Dio as Dio1;
interface Dio as Dio2;
interface Dio as Dio3;
interface Dio as Dio4;
interface Dio as Dio5;

//counter channels
interface StdControl as CounterControl;
interface Dio as Counter;

command result_t Plugged();
}
}
```

## 06 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
implementation
{

#define SCHEDULER_RESPONSE_TIME 150
#define TIME_SCALE 100 //this means we have resolution of 0.1 sec
#define FLAG_SET 1
#define FLAG_NOT_SET 0

    //flag for power saving
    uint8_t flag25, flag33, flag50;

    bool stopFlag; //MPK

    //main data structure 10 byte per recorder
    struct SampleRecords{
        uint8_t channel;
        uint8_t channelType;
        // uint8_t param;
        int16_t ticks_left; //used for keeping the monostable timer
        int16_t sampling_interval; //Sampling interval set by command above,
//It is in second, SampleRecord in no use if set to zero.
    }SampleRecord[MAX_SAMPLERECORD];

    //check what is the SampleRecords that are available and return one that is available
    static inline int8_t get_available_SampleRecord()
    {
        int8_t i;
        for(i=0; i<MAX_SAMPLERECORD; i++)
        if( SampleRecord[i].sampling_interval == SAMPLE_RECORD_FREE )
        return i;
        return -1; //not available SampleRecord
    }

    //find the next channel which should be serviced.
    // task
    void next_schedule(){
        int8_t i;
        int16_t min=SCHEDULER_RESPONSE_TIME; //minimum time to be called.
// we set it to 15Sec min so that if a
// new sampling request comes we reply with 15 sec delay.

        if(stopFlag) //MPK
            return; //MPK

        for(i=0; i<MAX_SAMPLERECORD; i++) //find out any one who should be serviced before next 15
        {
            if( SampleRecord[i].sampling_interval != SAMPLE_RECORD_FREE )
            {
                if(SampleRecord[i].ticks_left < min) min = SampleRecord[i].ticks_left;
            }
        }
        for(i=0; i<MAX_SAMPLERECORD; i++) //set the next time accordingly
```

```

    {
        if( SampleRecord[i].sampling_interval != SAMPLERECORD.FREE )
        {
            SampleRecord[i].ticks_left = SampleRecord[i].ticks_left - min;
        }
    }
    min=min * TIME_SCALE ; //since timer gets input in milisecond and we get command in 0.
    call SamplerTimer.start(TIMER.ONE_SHOT , min);
}

```

```

static inline void setparam_analog(uint8_t i, uint8_t param)
{
    switch(SampleRecord[i].channel){
    case 0:
        call SetParam0.setParam(param);
        break;
    case 1:
        call SetParam1.setParam(param);
        break;
    case 2:
        call SetParam2.setParam(param);
        break;
    case 3:
        call SetParam3.setParam(param);
        break;
    case 4:
        call SetParam4.setParam(param);
        break;
    case 5:
        call SetParam5.setParam(param);
        break;
    case 6:
        call SetParam6.setParam(param);
        break;
    case 7:
        call SetParam7.setParam(param);
        break;
    case 8:
        call SetParam8.setParam(param);
        break;
    case 9:
        call SetParam9.setParam(param);
        break;
    case 10:
        call SetParam10.setParam(param);
        break;
    case 11:
        call SetParam11.setParam(param);
        break;
    case 12:
        call SetParam12.setParam(param);
        break;
    case 13:

```

## 08 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
        call SetParam13.setParam(param);
        break;
    default:
    }
    return;
}

static inline void setparam_digital(int8_t i, uint8_t param)
{
    switch (SampleRecord[i].channel){
    case 0:
        call Dio0.setparam(param);
        break;
    case 1:
        call Dio1.setparam(param);
        break;
    case 2:
        call Dio2.setparam(param);
        break;
    case 3:
        call Dio3.setparam(param);
        break;
    case 4:
        call Dio4.setparam(param);
        break;
    case 5:
        call Dio5.setparam(param);
        break;
    default:
    }
    return;
}

static inline void setparam_counter(int8_t i, uint8_t param)
{
    call Counter.setparam(param);
    return;
}

void sampleRecord(uint8_t i)
{
    if (SampleRecord[i].channelType==ANALOG) {
        switch (SampleRecord[i].channel){
        case 0:
            call ADC0.getData();
            break;
        case 1:
            call ADC1.getData();
            break;
        case 2:
            call ADC2.getData();
            break;
        }
    }
}
```

```

    case 3:
        call ADC3.getData();
        break;
    case 4:
        call ADC4.getData();
        break;
    case 5:
        call ADC5.getData();
        break;
    case 6:
        call ADC6.getData();
        break;
    case 7:
        call ADC7.getData();
        break;
    case 8:
        call ADC8.getData();
        break;
    case 9:
        call ADC9.getData();
        break;
    case 10:
        call ADC10.getData();
        break;
    case 11:
        call ADC11.getData();
        break;
    case 12:
        call ADC12.getData();
        break;
    case 13:
        call ADC13.getData();
        break;
    default:
    }
    return;
}
if (SampleRecord[i].channelType==BATTERY) {
    call Battery.getData();
    return;
}

if (SampleRecord[i].channelType==TEMPERATURE || SampleRecord[i].channelType==HUMIDITY)
    if (SampleRecord[i].channelType==TEMPERATURE) call Temp.getData();
    if (SampleRecord[i].channelType==HUMIDITY) call Hum.getData();
    return;
}

if (SampleRecord[i].channelType==DIGITAL) {
    switch (SampleRecord[i].channel){
    case 0:
        call Dio0.getData();

```

## 20 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
        break;
    case 1:
        call Dio1.getData();
        break;
    case 2:
        call Dio2.getData();
        break;
    case 3:
        call Dio3.getData();
        break;
    case 4:
        call Dio4.getData();
        break;
    case 5:
        call Dio5.getData();
        break;
    default:
    }
    return;
}
if (SampleRecord[i].channelType==COUNTER) {
    call Counter.getData();
    return;
}
return;
}

command result_t SamplerControl.init() {
    int i;
    call CounterControl.init();
    call DioControl.init();
    call IBADCcontrol.init();
    call BatteryControl.init();
    call TempHumControl.init();
    for (i=0; i<MAX.SAMPLERECORD; i++){
        SampleRecord[i].sampling_interval=SAMPLE_RECORD_FREE;
        SampleRecord[i].ticks_left=0xffff;
    }
    atomic {
        flag25=FLAG_NOT_SET;
        flag33=FLAG_NOT_SET;
        flag50=FLAG_NOT_SET;
    }
    return SUCCESS;
}

command result_t SamplerControl.start() {
stopFlag = FALSE; // MPK
    call CounterControl.start();
    call DioControl.start();
    call IBADCcontrol.start();
    call BatteryControl.start();
    call TempHumControl.start();
}
```

```

    call CounterControl.start();
    //post next_schedule();
    next_schedule();
    return SUCCESS;
}

command result_t SamplerControl.stop() {
int i;          // MPK
stopFlag = TRUE; // MPK
call SamplerTimer.stop(); // MPK
atomic { // MPK
    for(i = 0; i < MAX.SAMPLERECORD; i++) // MPK
    { // MPK
        call Sample.stop(i); // MPK
    } // MPK
} // MPK

    call CounterControl.stop();
    call DioControl.stop();
    call IBADCcontrol.stop();
    call BatteryControl.stop();
    call TempHumControl.stop();
    return SUCCESS;
}

command result_t PlugPlay()
{
    return call Plugged();
}

event result_t SamplerTimer.fired() {
    uint8_t i;
    //sample anyone which is supposed to be sampled
    for (i=0;i<MAX.SAMPLERECORD;i++)
    {
        if( SampleRecord[i].sampling_interval != SAMPLE.RECORD.FREE )
        {
            if(SampleRecord[i].ticks_left == 0 )
            {
                SampleRecord[i].ticks_left = SampleRecord[i].sampling_interval;
                sampleRecord(i);
            }
        }
    }
    //now see when timer should be fired for new samples
    //post next_schedule();
if (!stopFlag) //MPK
{ //MPK
    next_schedule();
} //MPK
    return SUCCESS;
}

```

## Z2 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
command result_t Sample.set_digital_output(uint8_t channel, uint8_t state)
{
}

command int8_t Sample.getSample(uint8_t channel, uint8_t channelType, uint16_t interval, uint8_t param)
{
    int8_t i;
    i=get_available_SampleRecord();
    if(i==-1) return i;
    SampleRecord[i].channel=channel;
    SampleRecord[i].channelType=channelType;
    SampleRecord[i].ticks_left=0; //used for keeping the monostable timer
    SampleRecord[i].sampling_interval=interval; //Sampling interval set by command above.
//SampleRecord in no use if set to zero
//SampleRecord[i].param=param;
    if(SampleRecord[i].channelType == DIGITAL ) setparam_digital(i,param);
    if(SampleRecord[i].channelType == COUNTER ) setparam_counter(i,param);
    if(SampleRecord[i].channelType == ANALOG ) setparam_analog(i,param);
    return i;
}

command result_t Sample.reTask(int8_t record, uint16_t interval)
{
    if(record<0 || record>MAX.SAMPLERECORD) return FAIL;
    SampleRecord[record].sampling_interval=interval;
    return SUCCESS;
}

command result_t Sample.stop(int8_t record)
{
    if(record<0 || record>MAX.SAMPLERECORD) return FAIL;
    if(SampleRecord[record].channelType == ANALOG ) setparam_analog(record,0); // MPK
    SampleRecord[record].sampling_interval= SAMPLE.RECORD.FREE;
    return SUCCESS;
}

default event result_t Sample.dataReady(uint8_t channel, uint8_t channelType, uint16_t data)
{
    return SUCCESS;
}

event result_t ADC0.dataReady(uint16_t data) {
    if(data != ADC.ERROR) signal Sample.dataReady(0,ANALOG,data);
    return SUCCESS;
}

event result_t ADC1.dataReady(uint16_t data) {
    if(data != ADC.ERROR) signal Sample.dataReady(1,ANALOG,data);
    return SUCCESS;
}
```

```
}

event result_t ADC2.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(2,ANALOG,data);
    return SUCCESS;
}

event result_t ADC3.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(3,ANALOG,data);
    return SUCCESS;
}

event result_t ADC4.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(4,ANALOG,data);
    return SUCCESS;
}

event result_t ADC5.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(5,ANALOG,data);
    return SUCCESS;
}

event result_t ADC6.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(6,ANALOG,data);
    return SUCCESS;
}

event result_t ADC7.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(7,ANALOG,data);
    return SUCCESS;
}

event result_t ADC8.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(8,ANALOG,data);
    return SUCCESS;
}

event result_t ADC9.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(9,ANALOG,data);
    return SUCCESS;
}

event result_t ADC10.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(10,ANALOG,data);
    return SUCCESS;
}

event result_t ADC11.dataReady(uint16_t data) {
    if(data != ADC_ERROR) signal Sample.dataReady(11,ANALOG,data);
    return SUCCESS;
}

event result_t ADC12.dataReady(uint16_t data) {
```

## 24 MICA2-BASED WIRELESS ACM VERSION 2 SOFTWARE: MODIFICATION OF XMDA300

```
        if (data != ADC_ERROR) signal Sample.dataReady(12, ANALOG, data);
        return SUCCESS;
    }

    event result_t ADC13.dataReady(uint16_t data) {
        if (data != ADC_ERROR) signal Sample.dataReady(13, ANALOG, data);
        return SUCCESS;
    }

    async event result_t Battery.dataReady(uint16_t data) {
        signal Sample.dataReady(0, BATTERY, data);
        return SUCCESS;
    }

    event result_t Temp.dataReady(uint16_t data) {
        signal Sample.dataReady(0, TEMPERATURE, data); //data type problem
        return SUCCESS;
    }

    event result_t Hum.dataReady(uint16_t data) {
        signal Sample.dataReady(0, HUMIDITY, data);
        return SUCCESS;
    }

    event result_t Dio0.dataReady(uint16_t data) {
        signal Sample.dataReady(0, DIGITAL, data);
        return SUCCESS;
    }

    event result_t Dio1.dataReady(uint16_t data) {
        signal Sample.dataReady(1, DIGITAL, data);
        return SUCCESS;
    }

    event result_t Dio2.dataReady(uint16_t data) {
        signal Sample.dataReady(2, DIGITAL, data);
        return SUCCESS;
    }

    event result_t Dio3.dataReady(uint16_t data) {
        signal Sample.dataReady(3, DIGITAL, data);
        return SUCCESS;
    }

    event result_t Dio4.dataReady(uint16_t data) {
        signal Sample.dataReady(4, DIGITAL, data);
        return SUCCESS;
    }

    event result_t Dio5.dataReady(uint16_t data) {
        signal Sample.dataReady(5, DIGITAL, data);
```

```
        return SUCCESS;
    }

    event result_t Dio0.dataOverflow() {
        return SUCCESS;
    }

    event result_t Dio1.dataOverflow() {
        return SUCCESS;
    }

    event result_t Dio2.dataOverflow() {
        return SUCCESS;
    }

    event result_t Dio3.dataOverflow() {
        return SUCCESS;
    }

    event result_t Dio4.dataOverflow() {
        return SUCCESS;
    }

    event result_t Dio5.dataOverflow() {
        return SUCCESS;
    }

    event result_t Counter.dataReady(uint16_t data) {
        signal Sample.dataReady(0,COUNTER,data);
        return SUCCESS;
    }

    event result_t Counter.dataOverflow() {
        return SUCCESS;
    }
}
```



## CHAPTER 3

### **MICA2-Based Wireless ACM Version 3 Software: Modification Version 2 XMDA300: *Shake 'n Wake***

This appendix contains all software used to program the MICA2 motes for Version 3 of the MICA2-based wireless ACM system. This appendix is organized by software directory and only the modified files are included.

## 1. MICA2 Software

### 1.1. xbow/tos/XLib directory.

```

/**
 * Provides a library module for handling basic application messages for
 * controlling a wireless sensor network.
 *
 * @file      XCommand.h
 * @author    Martin Turon
 * @version   2004/10/1   mturon      Initial version
 *
 * Summary of XSensor commands:
 *   reset , sleep , wakeup
 *   set/get (rate) "heartbeat"
 *   set/get (nodeid , group)
 *   set/get (radio freq , band , power)
 *   actuate (device , state)
 *   set/get (calibration)
 *   set/get (mesh type , max resend)
 *
 * Copyright (c) 2004 Crossbow Technology, Inc. All rights reserved.
 *
 * $Id: XCommand.h,v 1.2 2005/01/27 03:36:31 husq Exp $
 */

#define INITIAL_TIMER_RATE    10000

enum {
  // Basic instructions:
  XCOMMAND_END = 0x0,
  XCOMMAND_NOP = 0x1,
  XCOMMAND_GET_SERIALID,

  // Power Management:
  XCOMMAND_RESET = 0x10,
  XCOMMAND_SLEEP,
  XCOMMAND_WAKEUP,

  // Basic update rate:
  XCOMMAND_SET_RATE = 0x20,           // Update rate
  XCOMMAND_GET_RATE,

  // MoteConfig Parameter settings:
  XCOMMAND_GET_CONFIG = 0x30,        // Return radio freq and power
  XCOMMAND_SET_NODEID,
  XCOMMAND_SET_GROUP,
  XCOMMAND_SET_RF_POWER,
  XCOMMAND_SET_RF_CHANNEL,

  // Actuation:
  XCOMMAND_ACTUATE = 0x40,

```

```

    // MPK
    XCOMMAND.SET.TICKS = 0x50, //MPK
    XCOMMAND.SET.QUICK, //MPK
} XCommandOpcode;

enum {
    XCMD_DEVICE.LED.GREEN,
    XCMD_DEVICE.LED.YELLOW,
    XCMD_DEVICE.LED.RED,
    XCMD_DEVICE.LEDS,
    XCMD_DEVICE.SOUNDER,
    XCMD_DEVICE.RELAY1,
    XCMD_DEVICE.RELAY2,
    XCMD_DEVICE.RELAY3,
    XCMD_DEVICE.POT
} XSensorSubDevice;
// added pot line to above MPK

enum {
    XCMD.STATE.OFF = 0,
    XCMD.STATE.ON = 1,
    XCMD.STATE.TOGGLE
} XSensorSubState;

typedef struct XCommandOp {
    uint16_t cmd; // XCommandOpcode

    union {
        uint32_t newrate; //!< FOR XCOMMAND.SET.RATE
        uint16_t nodeid; //!< FOR XCOMMAND.SET.NODEID
        uint8_t group; //!< FOR XCOMMAND.SET.GROUP
        uint8_t rf_power; //!< FOR XCOMMAND.SET.RF.POWER
        uint8_t rf_channel; //!< FOR XCOMMAND.SET.RF.CHANNEL
        uint8_t quick; //MPK
        uint16_t ticks; //MPK

        /** FOR XCOMMAND.ACCTUATE */
        struct {
            uint16_t device; //!< LEDS, sounder, relay, ...
            uint16_t state; //!< off, on, toggle, ...
        } actuate;
    } param;
} __attribute__((packed)) XCommandOp;

typedef struct XCommandMsg {
    //uint16_t seq-no; // +++ Required by lib/Broadcast
    uint16_t dest; // +++ Desired destination (0xFFFF for broadcast?)
    XCommandOp inst[6];
} __attribute__((packed)) XCommandMsg;

```

80 MICA2-BASED WIRELESS ACM VERSION 3 SOFTWARE: MODIFICATION VERSION 2 XMDA300: *SHAKE*

```
typedef struct XSensorHeader{
    uint8_t board_id; // mica2,mica2dot,micaz
    uint8_t packet_id; // 1: default serialid msg
    uint8_t node_id;
    uint8_t rsvd;
}__attribute__((packed)) XCmdDataHeader;

typedef struct SerialIDData {
    uint8_t id[8];
} __attribute__((packed)) SerialIDData;

typedef struct XCmdDataMsg {
    XCmdDataHeader xHeader;
    union {
//PData1 datap1;
        SerialIDData sid;
    }xData;
} __attribute__((packed)) XCmdDataMsg;

#if defined(PLATFORMMICA2)
#define MOTE_BOARD_ID 0x60 //MTS300 sensor board id
#endif
#if defined(PLATFORMMICA2DOT)
#define MOTE_BOARD_ID 0x61 //MTS300 sensor board id
#endif
#if defined(PLATFORMMICAZ)
#define MOTE_BOARD_ID 0x62 //MTS300 sensor board id
#endif

/*
enum {
    AMXCOMMANDMSG = 48,
};
*/
```

**1.2. xbow/apps/XMesh/XMDA300 directory.**

```

/*          tab:4
 * IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.
By
 * downloading, copying, installing or using the software you agree to
 * this license.  If you do not agree to this license, do not download,
 * install, copy or use the software.
 *
 * Intel Open Source License
 *
 * Copyright (c) 2002 Intel Corporation
 * All rights reserved.
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *      Neither the name of the Intel Corporation nor the names of its
 * contributors may be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE INTEL OR ITS
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * @author Leah Fera, Martin Turon, Jaidev Prabhu
 *
 * $Id: XMDA300M.nc,v 1.4 2005/01/11 05:21:35 husq Exp $
 */

/*****
 *
 *   - Tests the MDA300 general prototyping card
 *     (see Crossbow MTS Series User Manual)
 *   - Read and control all MDA300 signals:
 *     ADC0, ADC1, ADC2, ADC3,...ADC11 inputs, DIO 0-5,
 *     counter, battery, humidity, temp
 *-----
 * Output results through mica2 uart and radio.

```

## 82 MICA2-BASED WIRELESS ACM VERSION 3 SOFTWARE: MODIFICATION VERSION 2 XMDA300: *SHAKE*

```
* Use xlisten.exe program to view data from either port:
* uart: mount mica2 on mib510 with MDA300
*       (must be connected or now data is read)
*       connect serial cable to PC
*       run xlisten.exe at 57600 baud
* radio: run mica2 with MDA300,
*        run another mica2 with TOSBASE
*        run xlisten.exe at 56K baud
* LED: the led will be green if the MDA300 is connected to the mica2 and
*      the program is running (and sending out packets). Otherwise it is red.
*-----
* Data packet structure:
*
* PACKET #1 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 1
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : analog adc data Ch.0
* msg->data[6,7] : analog adc data Ch.1
* msg->data[8,9] : analog adc data Ch.2
* msg->data[10,11] : analog adc data Ch.3
* msg->data[12,13] : analog adc data Ch.4
* msg->data[14,15] : analog adc data Ch.5
* msg->data[16,17] : analog adc data Ch.6
*
* PACKET #2 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 2
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : analog adc data Ch.7
* msg->data[6,7] : analog adc data Ch.8
* msg->data[8,9] : analog adc data Ch.9
* msg->data[10,11] : analog adc data Ch.10
* msg->data[12,13] : analog adc data Ch.11
* msg->data[14,15] : analog adc data Ch.12
* msg->data[16,17] : analog adc data Ch.13
*
*
* PACKET #3 (of 4)
*-----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 3
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : digital data Ch.0
* msg->data[6,7] : digital data Ch.1
* msg->data[8,9] : digital data Ch.2
* msg->data[10,11] : digital data Ch.3
* msg->data[12,13] : digital data Ch.4
```

```

* msg->data[14,15] : digital data Ch.5
*
* PACKET #4 (of 4)
* -----
* msg->data[0] : sensor id, MDA300 = 0x81
* msg->data[1] : packet number = 4
* msg->data[2] : node id
* msg->data[3] : reserved
* msg->data[4,5] : batt
* msg->data[6,7] : hum
* msg->data[8,9] : temp
* msg->data[10,11] : counter
* msg->data[14] : msg4_status (debug)
*
*****/

// include sensorboard.h definitions from tos/mda300 directory
#include "appFeatures.h"
includes XCommand;

includes sensorboard;
module XMDA300M
{
    provides interface StdControl;

    uses {
    interface Leds;

    interface Send;
    interface RouteControl;
#ifdef XMESHSYNC
    interface Receive as DownTree;
#endif
    interface XCommand;

    //Sampler Communication
    interface StdControl as SamplerControl;
    interface Sample;

    //Timer
    interface Timer;

    //relays
    interface Relay as relay_normally_closed;
    interface Relay as relay_normally_open;

    //support for plug and play
    command result_t PlugPlay();

#if FEATURE_UART_SEND

```

```

    interface SendMsg as SendUART;
    command result_t PowerMgrEnable();
    command result_t PowerMgrDisable();
#endif
}
}

implementation
{
#define ANALOG_SAMPLING_TIME    90
#define DIGITAL_SAMPLING_TIME  100
#define MISC_SAMPLING_TIME      110
#define MPKRATE 30720 // fire twice per minute
#define MPK_TICKS 36 // 18 minutes between samples
#define MPK_QUICK_MESH_COUNT 60 // MPK take 60 samples at a
    // high speed before lowering rate

#define ANALOG_SEND_FLAG 1
#define DIGITAL_SEND_FLAG 1
#define MISC_SEND_FLAG 1
#define ERR_SEND_FLAG 1

#define PACKET_FULL 0x1C1 // MPK Temp, Hum, Batt, ADC0
#if 0
#define PACKET_FULL 0x1FF
#endif

#define MSG_LEN 29 // excludes TOS header, but includes xbow header

    enum {
    PENDING = 0,
    NO_MSG = 1
    };

    enum {
    MDA300_PACKET1 = 1,
    MDA300_PACKET2 = 2,
    MDA300_PACKET3 = 3,
    MDA300_PACKET4 = 4,
    MDA300_ERR_PACKET = 0xf8
    };

/*****
    enum {
    SENSOR_ID = 0,
    PACKET_ID,
    NODE_ID,
    RESERVED,
    DATA_START
    } XPacketDataEnum;
*****/
/* Messages Buffers */

```

```

uint32_t timer_rate;
bool sleeping; // application command state
bool sending_packet;
uint16_t seqno;
XDataMsg *tmppack;

TOS_Msg packet;
TOS_Msg msg_send_buffer;
TOS_MsgPtr msg_ptr;

int packetCount = 0; // MPK
int timerTicks = 0; // MPK
int maxTicks = 2; // MPK Broadcast every minute until
// quickMeshCount > MPK_QUICK_MESH_COUNT
int quickMeshCount = 0; // MPK

uint16_t msg_status, pkt_full;
char test;

int8_t record[25];

static void initialize()
{
    atomic
    {
        sleeping = FALSE;
        sending_packet = FALSE;
        //timer_rate = XSENSOR_SAMPLE_RATE; // ORIGINAL LINE
        timer_rate = MPK_RATE; // MPK
    }
}

/*****
* Initialize the component. Initialize Leds
*
*****/
command result_t StdControl.init() {

    uint8_t i;

    // BEGIN MPK INT CODE
        sei(); // Enable external interrupts
        EIMSK &= ~(1 << 6); // disable ext int 6
        EICRB &= ~(0x03 << 4); // make int 6 low-level triggered
        EIMSK |= (1 << 6); // turn on ext int 6
    // END MPK INT CODE

    call Leds.init();

    atomic {
        msg_ptr = &msg_send_buffer;

```

```

        //sending_packet = FALSE;
    }
    msg_status = 0;
    pkt_full = PACKET_FULL;

    MAKE_BAT_MONITOR_OUTPUT(); // enable voltage ref power pin as output
    MAKE_ADC_INPUT();         // enable ADC7 as input

// usart1 is also connected to external serial flash
// set usart1 lines to correct state
// TOSH_MAKE_FLASH_SELECT_OUTPUT();
    TOSH_MAKE_FLASH_OUT_OUTPUT();           //tx output
    TOSH_MAKE_FLASH_CLK_OUTPUT();          //usart clk
// TOSH_SET_FLASH_SELECT_PIN();

    call SamplerControl.init();
    initialize();
    return SUCCESS;

//return rcombine(call SamplerControl.init(), call CommControl.init());
}

/*****
* Start the component. Start the clock. Setup timer and sampling
*
*****/
command result_t StdControl.start() {

    call SamplerControl.start();

    if(call PlugPlay())
    {

        call Timer.start(TIMER_REPEAT, timer_rate);

        //channel parameteres are irrelevant

        record[14] = call Sample.getSample(0,
            TEMPERATURE,
            MISC_SAMPLING_TIME,
            SAMPLER_DEFAULT);

        record[15] = call Sample.getSample(0,
            HUMIDITY,
            MISC_SAMPLING_TIME,
            SAMPLER_DEFAULT);

        record[16] = call Sample.getSample(0,

```

```

        BATTERY,
        MISC_SAMPLING_TIME,
        SAMPLER_DEFAULT);

    //start sampling channels. Channels 7-10 with averaging
    //since they are more precise.channels 3-6 make active excitation
    /* MPK
    record[0] = call Sample.getSample(0,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT );

    record[1] = call Sample.getSample(1,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT );

    record[2] = call Sample.getSample(2,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT);
MPK */

        /* MPK
    record[3] = call Sample.getSample(3,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT | EXCITATION_33 | DELAY_BEFORE_MEASUREMENT);

    record[4] = call Sample.getSample(4,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT);

    record[5] = call Sample.getSample(5,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT);

    record[6] = call Sample.getSample(6,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        SAMPLER_DEFAULT);

MPK */

    record[7] = call Sample.getSample(7,
        ANALOG,
        ANALOG_SAMPLING_TIME,
        AVERAGE_FOUR | EXCITATION_25 | DELAY_BEFORE_MEASUREMENT );

    /* MPK
    record[8] = call Sample.getSample(8,

```

```

        ANALOG,
        ANALOG.SAMPLING.TIME,
        AVERAGEFOUR | EXCITATION_25);

record[9] = call Sample.getSample(9,
        ANALOG,
        ANALOG.SAMPLING.TIME,
        AVERAGEFOUR | EXCITATION_25);
MPK */

/* MPK
record[10] = call Sample.getSample(10,
        ANALOG,
        ANALOG.SAMPLING.TIME,
        AVERAGEFOUR | EXCITATION_25);

record[11] = call Sample.getSample(11,
        ANALOG,
        ANALOG.SAMPLING.TIME,
        SAMPLER.DEFAULT);

record[12] = call Sample.getSample(12,
        ANALOG,
        ANALOG.SAMPLING.TIME,
        SAMPLER.DEFAULT);

record[13] = call Sample.getSample(13,
        ANALOG,
        ANALOG.SAMPLING.TIME,
        SAMPLER.DEFAULT | EXCITATION_50 | EXCITATION_ALWAYS.ON);
MPK */

/* // MPK
//digital chennels as accumulative counter

record[17] = call Sample.getSample(0,
        DIGITAL,
        DIGITAL.SAMPLING.TIME,
        RESET_ZERO.AFTER.READ | FALLING.EDGE);

record[18] = call Sample.getSample(1,
        DIGITAL,
        DIGITAL.SAMPLING.TIME,
        RISING.EDGE | EVENT);

record[19] = call Sample.getSample(2,
        DIGITAL,
        DIGITAL.SAMPLING.TIME,
        SAMPLER.DEFAULT | EVENT);

record[20] = call Sample.getSample(3,
        DIGITAL,

```

```

        DIGITAL_SAMPLING_TIME,
        FALLING_EDGE);

    record[21] = call Sample.getSample(4,
        DIGITAL,
        DIGITAL_SAMPLING_TIME,
        RISING_EDGE);

    record[22] = call Sample.getSample(5,
        DIGITAL,
        DIGITAL_SAMPLING_TIME,
        RISING_EDGE | EEPROM_TOTALIZER);

    //counter channels for frequency measurement, will reset to zero.

    record[23] = call Sample.getSample(0,
        COUNTER,
        MISC_SAMPLING_TIME,
        RESET_ZERO_AFTER_READ | RISING_EDGE);
    call Leds.greenOn();
MPK */
}

else {
    call Leds.redOn();
}

return SUCCESS;

}

/*****
 * Stop the component.
 *
 *****/

    command result_t StdControl.stop() {

    call SamplerControl.stop();

    return SUCCESS;

}

/*****
 * Task to transmit radio message
 * NOTE that data payload was already copied from the corresponding UART packet
 *****/
    task void send_radio_msg()
    {

```

```

        uint8_t i;
        uint16_t len;
        XDataMsg *data;
// BEGIN MPK INT CODE
EIMSK |= (1 << 6); // turn on ext int 6
// END MPK INT CODE
        if(sending_packet)
            return;
        atomic sending_packet=TRUE;
// Fill the given data buffer.
        data = (XDataMsg*) call Send.getBuffer(msg_ptr, &len);
        tmppack=(XDataMsg *)packet.data;
        for (i = 0; i <= sizeof(XDataMsg)-1; i++)
            ((uint8_t*)data)[i] = ((uint8_t*)tmppack)[i];

        data->xmeshHeader.packet_id = 6;
        data->xmeshHeader.board_id = SENSOR_BOARD_ID;
        data->xmeshHeader.node_id = TOS_LOCAL_ADDRESS;
        data->xmeshHeader.parent = call RouteControl.getParent();

#if FEATURE_UART_SEND
        if (TOS_LOCAL_ADDRESS != 0) {
            call Leds.yellowOn();
            call PowerMgrDisable();
            TOSH_uwait(1000);
            if (call SendUART.send(TOS_UART_ADDR, sizeof(XDataMsg), msg_ptr) != SUCCESS)
            {
                atomic sending_packet = FALSE;
                call Leds.greenToggle();
                call PowerMgrEnable();
            }
        }
        else
#endif
        {
            // Send the RF packet!
            call Leds.yellowOn();
            if (call Send.send(msg_ptr, sizeof(XDataMsg)) != SUCCESS) {
                atomic sending_packet = FALSE;
                call Leds.yellowOn();
                call Leds.greenOff();
            }
        }
    }

// BEGIN MPK INT CODE
TOSH_SIGNAL(SIG_INTERRUPT6)
{

```

```

        EIMSK &= ~(1 << 6);    // disable ext int 6
    post send_radio_msg();
    }
// END MPK INT CODE

#if FEATURE_UART_SEND
/**
 * Handle completion of sent UART packet.
 *
 * @author    Martin Turon
 * @version   2004/7/21      mturon      Initial revision
 */
event result_t SendUART.sendDone(TOS_MsgPtr msg, result_t success)
{
    //      if (msg->addr == TOS_UART_ADDR) {
    atomic msg_ptr = msg;
    msg_ptr->addr = TOS_BCAST_ADDR;

    if (call Send.send(msg_ptr, sizeof(XDataMsg)) != SUCCESS) {
    atomic sending_packet = FALSE;
    call Leds.yellowOff();
    }

    if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
    call PowerMgrEnable();

    //}
    return SUCCESS;
}
#endif

/**
 * Handle completion of sent RF packet.
 *
 * @author    Martin Turon
 * @version   2004/5/27      mturon      Initial revision
 */
event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
{
    atomic {
    msg_ptr = msg;
    sending_packet = FALSE;
    }
    call Leds.yellowOff();

#if FEATURE_UART_SEND
    if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
    call PowerMgrEnable();
#endif

    return SUCCESS;
}

```

```

/**
 * Handle a single dataReady event for all MDA300 data types.
 *
 * @author Leah Fera, Martin Turon
 *
 * @version 2004/3/17 leahfera Intial revision
 * @n 2004/4/1 mturon Improved state machine
 */
event result_t
Sample.dataReady(uint8_t channel, uint8_t channelType, uint16_t data)
{
    uint8_t i;

    switch (channelType) {
case ANALOG:
    switch (channel) {
// MSG 1 : first part of analog channels (0-6)
// case 0: // Original line MPK
case 7: // MPK just cram ADC7 into ADC0's spot
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc0 =data ;
        atomic {msg_status|=0x01;}
        break;

case 1:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc1 =data ;
        atomic {msg_status|=0x02;}
        break;

case 2:
        tmppack=(XDataMsg *)packet.data;
        tmppack->xData.datap6.adc2 =data ;
        atomic {msg_status|=0x04;}
        break;

default:
        break;
    } // case ANALOG (channel)
    break;

case DIGITAL:
    switch (channel) {
case 0:
        tmppack=(XDataMsg *)packet.data;

        tmppack->xData.datap6.dig0=data;
        atomic {msg_status|=0x08;}
        break;

```

```

case 1:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.dig1=data;
    atomic {msg_status|=0x10;}
    break;

case 2:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.dig2=data;
    atomic {msg_status|=0x20;}
    break;

default:
    break;
} // case DIGITAL (channel)
break;

case BATTERY:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.vref =data ;
    atomic {msg_status|=0x40;}
    break;

case HUMIDITY:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.humid =data ;
    atomic {msg_status|=0x80;}
    break;

case TEMPERATURE:
    tmppack=(XDataMsg *)packet.data;
    tmppack->xData.datap6.humtemp =data ;
    atomic {msg_status|=0x100;}
    break;

default:
    break;

} // switch (channelType)

if (sending_packet)
    return SUCCESS;

if (msg_status == pkt.full) {
    msg_status = 0;
    packetCount++; // MPK
    if(packetCount >= 3) // MPK
    {
        // MPK
        call SamplerControl.stop(); // MPK
        packetCount = 0; // MPK
    }
}

```

```

        // get rid of timer ticks accumulated while sampling MPK
        atomic {timerTicks = 0;} // MPK
    } // MPK

    if(packetCount > 1) // MPK first ADC reading always is bad
        post send_radio_msg();
    }

    return SUCCESS;
}

/*****
 * Timer Fired -
 *
 *****/
event result_t Timer.fired() {
/*    MPK
    if (sending_packet)
        return SUCCESS; //don't overrun buffers
    if (test != 0) {
        test=0;
        call relay_normally_closed.toggle();
        call Leds.greenOn();
    }
    else {
        test=1;
        call relay_normally_open.toggle();
        call Leds.greenOn();
    }
}
MPK */

// MPK if enough ticks, turn on the MDA300 and do a new
// MPK round of getSample calls
atomic{timerTicks ++;} // MPK

if(quickMeshCount > MPK_QUICK_MESH_COUNT)// MPK
{ // MPK
    atomic{maxTicks = MPK_TICKS;} // MPK
} // MPK
// MPK
//call RouteControl.manualUpdate(); // MPK

if ((timerTicks < maxTicks) && (maxTicks > 0)) // MPK
    return SUCCESS; // MPK

atomic {timerTicks = 0;} // MPK
atomic{quickMeshCount++;} // MPK
call SamplerControl.init(); // MPK Is this necessary?
call SamplerControl.start(); // MPK

if(call PlugPlay()) // MPK
{ // MPK

```

```

record[14] = call Sample.getSample(0,
    TEMPERATURE,
    MISC.SAMPLING.TIME,
    SAMPLER.DEFAULT); // MPK
record[15] = call Sample.getSample(0,
    HUMIDITY,
    MISC.SAMPLING.TIME,
    SAMPLER.DEFAULT); // MPK
record[16] = call Sample.getSample(0, BATTERY, MISC.SAMPLING.TIME, SAMPLER.DEFAULT); // MPK
record[7] = call Sample.getSample(7,
    ANALOG,
    ANALOG.SAMPLING.TIME, AVERAGE.FOUR |
    EXCITATION.25 |
    DELAY.BEFORE.MEASUREMENT ); // MPK
}

return SUCCESS;

}

/**
 * Handles all broadcast command messages sent over network.
 *
 * NOTE: Bcast messages will not be received if seq_no is not properly
 *       set in first two bytes of data payload. Also, payload is
 *       the remaining data after the required seq-no.
 *
 * @version 2004/10/5 mturon Initial version
 */
event result_t XCommand.received(XCommandOp *opcode) {

    switch (opcode->cmd) {
case XCOMMAND.SET.RATE:
    // Change the data collection rate.
    /* MPK
    timer_rate = opcode->param.newrate;
    call Timer.stop();
    call Timer.start(TIMER.REPEAT, timer_rate);
MPK */
    atomic{maxTicks = opcode->param.newrate;} // MPK
    atomic{timerTicks = 0;} // MPK
    break;

case XCOMMAND.SLEEP:
    // Stop collecting data, and go to sleep.
    sleeping = TRUE;
    call Timer.stop();
    call Leds.set(0);
    break;

case XCOMMAND.WAKEUP:
    // Wake up from sleep state.
    if (sleeping) {

```

```

        initialize ();
        call Timer.start (TIMER_REPEAT, timer_rate);
        sleeping = FALSE;
    }
    break;

case XCOMMAND_RESET:
    // Reset the mote now.
    break;

case XCOMMAND_ACTUATE:
    break;

default:
    break;
}

return SUCCESS;
}

#ifdef XMESH_SYNC
task void SendPing () {
    XDataMsg *pReading;
    uint16_t Len;

    if ((pReading = (XDataMsg *) call Send.getBuffer (msg_ptr, &Len)) {
        pReading->xmeshHeader.parent = call RouteControl.getParent ();
        if ((call Send.send (msg_ptr, sizeof (XDataMsg))) != SUCCESS)
            atomic sending_packet = FALSE;
    }
}

}

event TOS_MsgPtr DownTree.receive (TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen) {

    if (!sending_packet) {
        call Leds.yellowToggle ();
        atomic sending_packet = TRUE;
        post SendPing (); // pMsg->XXX);
    }
    return pMsg;
}
#endif
}

```

## 2. UC-7420 Software

### 2.1. xbow/beta/tools/src/xcmd directory.

```

/**
 * Sends commands to the serial port to control a wireless sensor network.
 *
 * @file      xcommand.c
 * @author    Martin Turon
 * @version   2004/10/3   mturon   Initial version
 *
 * Copyright (c) 2004 Crossbow Technology, Inc. All rights reserved.
 *
 * $Id: xcommand.c,v 1.4 2004/11/11 01:00:51 mturon Exp $
 */

#include "xcommand.h"

static const char *g_version =
    "$Id: xcommand.c,v 1.4 2004/11/11 01:00:51 mturon Exp $";

/** A structure to store parsed parameter flags. */
typedef union {
    unsigned flat;

    struct {
        // output display options
        unsigned display_raw      : 1;  //!< raw TOS packets
        unsigned display_parsed  : 1;  //!< pull out sensor readings
        unsigned display_cooked  : 1;  //!< convert to engineering units
        unsigned export_parsed   : 1;  //!< output comma delimited fields
        unsigned export_cooked   : 1;  //!< output comma delimited fields
        unsigned log_parsed      : 1;  //!< log output to database
        unsigned log_cooked      : 1;  //!< log output to database
        unsigned display_time    : 1;  //!< display timestamp of packet
        unsigned display_ascii   : 1;  //!< display packet as ASCII characters
        unsigned display_rsvd    : 7;  //!< pad first word for output options

        // modes of operation
        unsigned display_help    : 1;
        unsigned display_baud    : 1;  //!< baud was set by user
        unsigned mode_debug      : 1;  //!< debug serial port
        unsigned mode_quiet      : 1;  //!< suppress headers
        unsigned mode_version    : 1;  //!< print versions of all modules
        unsigned mode_socket     : 1;  //!< connect to a serial forwarder
        unsigned mode_framing    : 2;  //!< auto=0, framed=1, unframed=2
    } bits;

    struct {
        unsigned short output;  //!< one output option required
        unsigned short mode;
    } options;
} s_params;

```

```

/** A variable to store parsed parameter flags. */
static s_params  g_params;
static int      g_ostream;  //!< Handle of output stream
static char *   g_command;

unsigned char    g_am_type = AMTYPEXCOMMAND;  //!< TOS AM type of command to send

char *          g_argument;

int            g_seq_no = 100;    //!< Broadcast sequence number
unsigned      g_frame = 0x26;    //!< Packetizer sequence number
unsigned      g_group = 0xff;
unsigned      g_dest = 0xFFFF;  //!< Destination nodeid (0xFFFF = all)

void xcommand_print_help() {
    printf(
        "\nUsage: xcommand <-[v|q]> command argument"
        "\n          <-s=device> <-b=baud> <-i=server:port>"
        "\n          <-n=nodeid> <-g=group> <-#=seq_no>"
        "\n  -? = display help [help]"
        "\n  -q = quiet mode (suppress headers)"
        "\n  -v = show version of all modules"
        "\n  -b = set the baudrate [baud=#|mica2|mica2dot]"
        "\n  -s = set serial port device [device=com1]"
        "\n  -i = internet serial forwarder [inet=host:port]"
        "\n  -n = nodeid to send command to [node=nodeid]"
        "\n  -g = group to send command over [group=groupid]"
        "\n  -# = sequence number of command [#=seq_no]"
        "\n"
        "\n  XCommand list:"
        "\n    wake, sleep, reset"
        //"\n    set_rate <interval in millisec>"
        "\n    set_rate <interval in seconds>"
        "\n    set_leds <number from 0-7>"
        "\n    set_sound <0=off|1=on>"
        "\n    red_on, red_off, red_toggle"
        "\n    green_on, green_off, green_toggle"
        "\n    yellow_on, yellow_off, yellow_toggle"
        "\n    set_quick <0=off|1=on>"
        "\n    set_ticks <integer>"
        "\n    set_pot <integer>"
        "\n\n"
    );
}

/**
 * Extracts command line options and sets flags internally.
 *
 * @param argc      Argument count
 * @param argv      Argument vector
 *
 * @author Martin Turon

```

```

*
* @version 2004/3/10 mturon Initial version
* @n 2004/3/12 mturon Added -b,-s,-q,-x
* @n 2004/8/04 mturon Added -l [ver. 1.11]
* @n 2004/8/22 mturon Added -i [ver. 1.13]
* @n 2004/9/27 mturon Added -t [ver. 1.15]
* @n 2004/9/29 mturon Added -f,-a [v 1.16]
*/
void parse_args(int argc, char **argv)
{
    // This value is set if/when the bitflag is set.
    unsigned baudrate = 0;
    char *server, *port;

    g_params.flat = 0; /* default to no params set */

    xpacket_initialize();

    while (argc) {
        if ((argv[argc]) && (*argv[argc] == '-')) {
            switch(argv[argc][1]) {
                case '?':
                    g_params.bits.display_help = 1;
                    break;

                case 'q':
                    g_params.bits.mode_quiet = 1;
                    break;

                case 'p':
                    g_params.bits.display_parsed = 1;
                    break;

                case 'r':
                    g_params.bits.display_raw = 1;
                    break;

                case 'c':
                    g_params.bits.display_cooked = 1;
                    break;

                case 'f': {
                    switch (argv[argc][2]) {
                        case '=': // specify arbitrary offset
                            g_params.bits.mode_framing = atoi(argv[argc]+3)&3;
                            break;

                        case 'a': // automatic deframing
                            g_params.bits.mode_framing = 0;
                    }
                }
                break;

                case '0':
                case 'n': // assume no framing
            }
        }
    }
}

```

```

        g_params.bits.mode_framing = 2;
    break;

    case '1':    // force framing
default:
        g_params.bits.mode_framing = 1;
        break;
    }
    break;
}

case 'b':
    if (argv[argc][2] == '=') {
        baudrate = xserial_set_baud(argv[argc]+3);
        g_params.bits.display_baud = 1;
    }
    break;

case 's':
    if (argv[argc][2] == '=') {
        xserial_set_device(argv[argc]+3);
    }
    break;

case 'a':
    if (argv[argc][2] == '=') {
        g_am_type = atoi(argv[argc]+3);
    }
    break;

case 'g':
    if (argv[argc][2] == '=') {
        g_group = atoi(argv[argc]+3);
    }
    break;

case 'n':
    if (argv[argc][2] == '=') {
        g_dest = atoi(argv[argc]+3);
    }
    break;

case '#':
    if (argv[argc][2] == '=') {
        g_seq_no = atoi(argv[argc]+3);
    }
    break;

case 't':
    g_params.bits.display_time = 1;
    break;

case 'i':

```

```

    g_params.bits.mode_socket = 1;
        if (argv[argc][2] == '=') {
server = argv[argc]+3;
port = strchr(server, ':');
if (port) {
    *port++ = '\0';
    xsocket_set_port(port);
}
        xsocket_set_server(server);
    }
    break;

    case 'v':
        g_params.bits.mode_version = 1;
        break;

    case 'd':
        g_params.bits.mode_debug = 1;
        break;
    }
} else {
if (argv[argc]) {
// processing arguments backwards, so always update command.
if (g_command) {
    g_argument = g_command;
}
g_command = argv[argc];
}
}
    argc--;
}

if (!g_params.bits.mode_quiet) {
// Summarize parameter settings
printf("xcommand Ver:%s\n", g_version);
if (g_params.bits.mode_version)    xpacket_print_versions();
printf("Using params: ");
if (g_params.bits.display_help)    printf("[help] ");
if (g_params.bits.display_baud)    printf(" [baud=0x%04x] ", baudrate);
if (g_params.bits.display_raw)     printf("[raw] ");
if (g_params.bits.display_ascii)   printf("[ascii] ");
if (g_params.bits.display_parsed)  printf("[parsed] ");
if (g_params.bits.display_cooked)  printf("[cooked] ");
if (g_params.bits.export_parsed)   printf("[export] ");
if (g_params.bits.display_time)    printf("[timed] ");
if (g_params.bits.export_cooked)   printf("[convert] ");
if (g_params.bits.log_cooked)      printf("[logging] ");
if (g_params.bits.mode_framing==1) printf("[framed] ");
if (g_params.bits.mode_framing==2) printf("[unframed] ");
if (g_params.bits.mode_socket)     printf(" [inet=%s:%u] ",
    xsocket_get_server(),
    xsocket_get_port());
if (g_params.bits.mode_debug) {

```

```

        printf("[debug - serial dump!] \n");
        xserial_port_dump();
    }
    printf("\n");
}

if (g_params.bits.display_help) {
xcommand_print_help();
    exit(0);
}

/* Default to displaying packets as raw, parsed, and cooked. */
if (g_params.options.output == 0) {
    g_params.bits.display_raw = 1;
    g_params.bits.display_parsed = 1;
    g_params.bits.display_cooked = 1;
}

/* Stream initialization */

// Set STDOUT and STDERR to be line buffered, so output is not delayed.
setlinebuf(stdout);
setlinebuf(stderr);

if (g_params.bits.mode_socket) {
    g_ostream = xsocket_port_open();
} else {
    g_ostream = xserial_port_open();
}
}

int xmain_get_verbos() {
    return !g_params.bits.mode_quiet;
}

/**
 * The main entry point for the sensor commander console application.
 *
 * @param   argc           Argument count
 * @param   argv           Argument vector
 *
 * @author   Martin Turon
 * @version  2004/10/3     mturon      Intial version
 */
int main(int argc, char **argv)
{
    int len = 0;
    unsigned char buffer[255];

    parse_args(argc, argv);

    if (!g_command) {
xcommand_print_help();

```

```

exit(2);
}

    if (!xpacket_get_app(g_am_type)) {
printf(" error: No command table for AM type: %d", g_am_type);
exit(2);
    }
    XCmdBuilder cmd_bldr = xpacket_get_builder(g_am_type, g_command);
    if (!cmd_bldr) {
printf(" error: Command not found for AM type %d: %s",
        g_am_type, g_command);
exit(2);
    }

    printf(" Sending (#%d) %s %s : ", g_seq_no, g_command, g_argument);
    len = xpacket_build_cmd(buffer, cmd_bldr, g_params.bits.mode_socket);

    xpacket_print_raw(buffer, len);
    xserial_port_write_packet(g_ostream, buffer, len);

    return 1;
}

##### User Manual Follows #####

/**
@mainpage XCommand Documentation

@section version Version
$Id: xcommand.c,v 1.4 2004/11/11 01:00:51 mturon Exp $

@section usage Usage
Usage: xcommand <-?|v|q> command argument
@n         <-s=device> <-b=baud> <-i=server:port>
@n         <-n=nodeid> <-g=group> <-#=seq.no> <-a=app AM type>
@n  -? = display help [help]
@n  -q = quiet mode (suppress headers)
@n  -v = show version of all modules
@n  -b = set the baudrate [baud=#|mica2|mica2dot]
@n  -s = set serial port device [device=com1]
@n  -i = internet serial forwarder [inet=host:port]
@n  -n = nodeid to send command to [node=nodeid]
@n  -g = group to send command over [group=groupid]
@n  -a = application or AM type of command message [app=type]
@n  -# = sequence number of command [#=seq.no]
@n
@n  XCommand list:
@n      wake, sleep, reset
@n      set_rate <interval in millisec>
@n      set_leds <number from 0-7>
@n      set_sound <0=off|1=on>
@n      red_on, red_off, red_toggle

```

### 30MICA2-BASED WIRELESS ACM VERSION 3 SOFTWARE: MODIFICATION VERSION 2 XMDA300: *SHAKE*

```
@n      green_on , green_off , green_toggle
@n      yellow_on , yellow_off , yellow_toggle
@n
```

```
@section params Parameters
```

```
@subsection help -? [help]
```

XCommand has many modes of operation that can be controlled by passing command line parameters. The current list of these command line options and a brief usage explanation is always available by passing the `-?` flag.

```
@n
```

```
@n A detail explanation of each command line option as of version 1.1 follows.
```

```
@subsection versions -v [versions]
```

Displays complete version information for all sensorboard decoding modules within `xcommand`.

```
@n $ xcmd -v
```

```
@n xcommand Ver: Id: xcommand.c,v 1.1 2004/10/07 19:33:13 mturon Exp
@n   f8: Id: cmd_XMesh.c,v 1.4 2004/10/08 00:33:20 mturon Exp
@n   30: Id: cmd_XSensor.c,v 1.2 2004/10/07 23:14:25 mturon Exp
@n   12: Id: cmd_Surge.c,v 1.1 2004/10/07 19:33:13 mturon Exp
@n   08: Id: cmd_SimpleCmd.c,v 1.1 2004/10/07 19:33:13 mturon Exp
```

```
@subsection quiet -q [quiet]
```

This flag suppresses the standard `xcommand` header which displays the version string and parameter selections.

```
@subsection baud -b=baudrate [baud]
```

This flag allows the user to set the baud rate of the serial line connection. The default baud rate is 57600 bits per second which is compatible with the Mica2. The desired baudrate must be passed as a number directly after the equals sign with no spaces inbetween, i.e. `-b=19200`. Optionally, a product name can be passed in lieu of an actual number and the proper baud will be set, i.e. `-b=mica2dot`. Valid product names are:

```
mica2          (57600 baud)
mica2dot      (19200 baud)
```

```
@subsection serial -s=port [serial]
```

This flag gives the user the ability to specify which COM port or device `xcommand` should use. The default port is `/dev/ttyS0` or the UNIX equivalent to COM1. The given port must be passed directly after the

equals sign with no spaces, i.e. `-s=com3`.

@subsection internet `-i=hostname:port [inet]`

This flag tells xcmd to connect via a serial forwarder over a TCP/IP internet socket. Specify the hostname and port to connect in the argument. The default hostname is localhost, and the default port 9001. The keyword 'mib600' can be passed as an alias to port 10002 when connecting to that hardware device. The hostname and port must be passed directly after the equals sign with no spaces with a optional colon inbetween, i.e. `-i=remote`, `-i=10.1.1.1:9000`, `-i=mymib:mib600`, `-i=:9002`, `-i=localhost:9003`, or `-i=stargate.xbow.com`.

@subsection node `-n=nodeid [node]`

This flag specifies which node to send the message to. When not passed, the broadcast address is used (0xFFFF).

@subsection group `-g=group [group]`

This flag specifies which AM group id to send the message on. Nodes typically ignore messages for a group which they have not been programmed for, so it is important to pass the correct group here.

@subsection sequence `-#=#seq_no [sequence]`

This flag defines the sequence number that will be used for sending the command packet. The TinyOS Bcast component uses the sequence number to insure that the same command doesn't cycle through the network forever. Try and increment this number systematically everytime a command message is sent.

@subsection application `-a=am_type [app]`

This flag specifies which AM TOS type to send the message on. The AM type can be passed as an integer, or as an application name. For example, Surge sends commands on `AM_TYPE=18`. To set the rate of a Surge node, one would use `'xcmd -a=18 set_rate'`. The default `am_type` will work for any application that uses the TinyOS XCommand component, such as XSensor.

@section params Commands

@subsection sleep [XCommand]

Will tell the mote to stop collecting data and go to sleep.

@subsection wake [XCommand]

Will wake up a mote and restart data aquisition from the sleep state.

@subsection set\_rate [XCommand]

The `set_rate` command will change the data aquisition duty cycle of the

mote. The first argument is the new timer interval in milliseconds.

@subsection set\_leds [XCommand]

The set\_leds command will actuate all three LEDs of a mote using the following encoding of the first argument: bit 1 = red, bit 2 = green, bit 3 = yellow. Passing a 7 for instance will turn all the LEDs on, while passing a 0 will turn them all off.

@section building Build Process

The source code for the xcommand tool is located at: /opt/tinyos-1.x/contrib/xbow/tools/src  
@@@

To build the tool, change to the xcmd source directory and run 'make'.

@@@

To get the latest version of the source, change to the xcmd source directory and run 'cvs update'.

@section setup Setup

Xcommand is a command line tool that can be run from a cygwin shell by simply typing 'xcmd'. The executable needs to be in your working path to use it. A simple way to add Xcommand to your working path is to create a soft link to it by running the following command:

```
@n$ ln -s /opt/tinyos-1.x/contrib/xbow/tools/src/xcmd /usr/local/bin/xcmd  
@@@
```

You can use Xcommand to actuate subdevices on a mote such as the LEDs, sounder, or relays. The commands can be sent to either one mote over a serial link, or a wireless network of motes. In both configurations, you need to have a MIB510 board connected via a serial cable to your PC.

@@@

For a single mote configuration, the mote must be programmed with a XSensorMXX### application and plugged into the MIB510. The mote will listen for command packets over the UART and radio whenever it has power.

@@@

For the network of motes configuration, a base station mote needs to be programmed with TOSBase and plugged into the MIB510. All other motes need to be installed with an XSensorMXX## application and put within range of the base station or a valid multi-hop peer. Take care to program all the motes to the same frequency and group id.

\*/

**2.2. xbow/beta/tools/src/xcmd/apps directory.**

```

/**
 * Handles generating and sending commands to control an XSensor application.
 *
 * @file      cmd_XSensor.c
 * @author    Martin Turon
 * @version   2004/10/5      mturon      Initial version
 *
 * Copyright (c) 2004 Crossbow Technology, Inc. All rights reserved.
 *
 * $Id: cmd_XSensor.c,v 1.5 2005/02/02 05:47:40 husq Exp $
 */

#include "../xcommand.h"

enum {
    // Basic instructions:
    XCOMMAND.END = 0x0,
    XCOMMAND.NOP = 0x1,
    XCOMMAND.GET_SERIALID,

    // Power Management:
    XCOMMAND.RESET = 0x10,
    XCOMMAND.SLEEP,
    XCOMMAND.WAKEUP,

    // Basic update rate:
    XCOMMAND.SET_RATE = 0x20,           // Update rate
    XCOMMAND.GET_RATE,

    // MoteConfig Parameter settings:
    XCOMMAND.GET_CONFIG = 0x30,        // Return radio freq and power
    XCOMMAND.SET_NODEID,
    XCOMMAND.SET_GROUP,
    XCOMMAND.SET_RF_POWER,
    XCOMMAND.SET_RF_CHANNEL,

    // Actuation:
    XCOMMAND.ACTUATE = 0x40,
} XCommandOpcode;

enum {
    XCMD_DEVICE_LED_GREEN,
    XCMD_DEVICE_LED_YELLOW,
    XCMD_DEVICE_LED_RED,
    XCMD_DEVICE_LEDS,
    XCMD_DEVICE_SOUNDER,
    XCMD_DEVICE_RELAY1,
    XCMD_DEVICE_RELAY2,
    XCMD_DEVICE_RELAY3,
    XCMD_DEVICE_POT
} XSensorSubDevice;

```

```

// added pot line to above MPK

enum {
    XCMD.STATE_OFF = 0,
    XCMD.STATE_ON = 1,
    XCMD.STATE_TOGGLE
} XSensorSubState;

typedef struct XCommandOp {
    uint16_t    cmd;           // XCommandOpcode

    union {
        uint32_t newrate;     //!< FOR XCOMMAND.SET_RATE
        uint16_t nodeid;     //!< FOR XCOMMAND.SET_NODEID
        uint8_t  group;      //!< FOR XCOMMAND.SET_GROUP
        uint8_t  rf_power;   //!< FOR XCOMMAND.SET_RF_POWER
        uint32_t rf_channel; //!< FOR XCOMMAND.SET_RF_CHANNEL

        /** FOR XCOMMAND.ACCTUATE */
        struct {
            uint16_t device;  //!< LEDES, sounder, relay, ...
            uint16_t state;  //!< off, on, toggle, ...
        } actuate;
        } param;
} __attribute__((packed)) XCommandOp;

typedef struct XCommandMsg {
    TOSMsgHeader tos;
    uint16_t     seq_no;      //!< Required by lib/Broadcast/Bcast
    uint16_t     dest;       //!< Destination nodeid (0xFFFF for all)
    XCommandOp   inst[1];
} __attribute__((packed)) XCommandMsg;

void xcmd_set_header(char * buffer)
{
    // Fill in TOS.msg header.
    XCommandMsg *msg = (XCommandMsg *)buffer;
    msg->tos.addr     = g_dest;
    msg->tos.type     = AMIYPEXCOMMAND;
    msg->tos.group    = g_group;
    msg->tos.length   = sizeof(XCommandMsg) - sizeof(TOSMsgHeader);
}

int xcmd_basic(char * buffer, int opcode)
{
    XCommandMsg *msg = (XCommandMsg *)buffer;
    xcmd_set_header(buffer);
    // Data payload
    msg->seq_no      = g_seq_no;
}

```

```

    msg->dest          = g_dest;
    msg->inst[0].cmd = opcode;
    msg->inst[0].param.newrate = 0xCCCCCC; // Fill unused in known way
    return sizeof(XCommandMsg);
}

int xcmd_actuate(char * buffer, int device, int state)
{
    XCommandMsg *msg = (XCommandMsg *)buffer;
    xcmd_set_header(buffer);
    // Data payload
    msg->seq_no      = g_seq_no;
    msg->dest        = g_dest;
    msg->inst[0].cmd = XCOMMANDACTUATE;
    msg->inst[0].param.actuate.device = device;
    msg->inst[0].param.actuate.state = state;
    return sizeof(XCommandMsg);
}

int xcmd_get_serialid(char * buffer) {return xcmd_basic(buffer, XCOMMAND.GET.SERIALID); }
int xcmd_get_config(char * buffer) {return xcmd_basic(buffer, XCOMMAND.GET.CONFIG); }
int xcmd_reset(char * buffer) { return xcmd_basic(buffer, XCOMMAND.RESET); }
}
int xcmd_sleep(char * buffer) { return xcmd_basic(buffer, XCOMMAND.SLEEP); }
}
int xcmd_wake(char * buffer) { return xcmd_basic(buffer, XCOMMAND.WAKEUP); }

int xcmd_green_off(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.GREEN, 0);
}
int xcmd_green_on(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.GREEN, 1);
}
int xcmd_green_toggle(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.GREEN, 2);
}
int xcmd_red_off(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.RED, 0);
}
int xcmd_red_on(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.RED, 1);
}
int xcmd_red_toggle(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.RED, 2);
}
int xcmd_yellow_off(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.YELLOW, 0);
}
int xcmd_yellow_on(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.YELLOW, 1);
}
int xcmd_yellow_toggle(char *buffer) {
    return xcmd_actuate(buffer, XCMD.DEVICE.LED.YELLOW, 2);
}

```

```

}

//MPK
int xcmd_set_pot(char *buffer) {
    int pot = 7;
    if (g_argument) pot = atoi(g_argument);
    return xcmd_actuate(buffer, XCMD_DEVICE_POT, pot);
}

int xcmd_set_leds(char *buffer) {
    int leds = 7; // default to all on.
    if (g_argument) leds = atoi(g_argument);
    return xcmd_actuate(buffer, XCMD_DEVICE_LEDS, leds);
}

int xcmd_set_sounder(char *buffer) {
    int sounder = 0; // default to off.
    if (g_argument) sounder = atoi(g_argument);
    return xcmd_actuate(buffer, XCMD_DEVICE_SOUNDER, sounder);
}

unsigned xcmd_set_config(char * buffer, unsigned cmd, unsigned mask)
{
    XCommandMsg *msg = (XCommandMsg *)buffer;

    unsigned arg = mask; // default to maximum value
    if (g_argument) arg = atoi(g_argument);
    arg &= mask;

    xcmd_set_header(buffer);

    // Data payload
    msg->seq_no = g_seq_no;
    msg->dest = g_dest;
    msg->inst[0].cmd = cmd;

    return arg;
}

int xcmd_set_rate(char * buffer)
{
    XCommandMsg *msg = (XCommandMsg *)buffer;
    unsigned arg = xcmd_set_config(buffer, XCOMMAND_SET_RATE, 0xFFFFFFFF);
    if (arg == 0xFFFFFFFF) arg = 5000;
    //if (arg < 100) arg = 100;
    msg->inst[0].param.newrate = arg;
    printf("\narg is %08x\n", msg->inst[0].param.newrate);
    return sizeof(XCommandMsg);
}

int xcmd_set_nodeid(char * buffer)
{

```

```

XCommandMsg *msg = (XCommandMsg *)buffer;
unsigned arg = xcmd_set_config(buffer, XCOMMAND.SET_NODEID, 0xFFFF);
msg->inst[0].param.nodeid = arg;
return sizeof(XCommandMsg);
}

int xcmd_set_group(char * buffer)
{
XCommandMsg *msg = (XCommandMsg *)buffer;
unsigned arg = xcmd_set_config(buffer, XCOMMAND.SET_GROUP, 0xFF);
msg->inst[0].param.group = arg;
return sizeof(XCommandMsg);
}

int xcmd_set_rf_power(char * buffer)
{
XCommandMsg *msg = (XCommandMsg *)buffer;
unsigned arg = xcmd_set_config(buffer, XCOMMAND.SET_RF_POWER, 0xFF);
msg->inst[0].param.rf_power = arg;
return sizeof(XCommandMsg);
}

int xcmd_set_rf_channel(char * buffer)
{
XCommandMsg *msg = (XCommandMsg *)buffer;
unsigned arg = xcmd_set_config(buffer, XCOMMAND.SET_RF_CHANNEL, 0xFF);
msg->inst[0].param.rf_channel = arg;
return sizeof(XCommandMsg);
}

extern int xmesh_cmd_light_path(char * buffer);

/** List of commands handled by XSensor applications using XCommand. */
XCmdHandler xsensor_cmd_list [] = {
{"get_serialid",          xcmd_get_serialid},
{"get_config",           xcmd_get_config},

// Power Management
{"reset",                xcmd_reset},
{"wake",                 xcmd_wake},
{"sleep",                xcmd_sleep},

// App Control
{"set_rate",             xcmd_set_rate},

// Mote Configuration
{"set_nodeid",           xcmd_set_nodeid},
{"set_group",            xcmd_set_group},
{"set_rf_power",         xcmd_set_rf_power},
{"set_rf_channel",       xcmd_set_rf_channel},

// Actuation
{"set_sound",            xcmd_set_sounder},

```

```

    {"set_leds",      xcmd_set_leds},
    {"green_on",     xcmd_green_on},
    {"green_off",    xcmd_green_off},
    {"green_toggle", xcmd_green_toggle},
    {"red_on",       xcmd_red_on},
    {"red_off",      xcmd_red_off},
    {"red_toggle",   xcmd_red_toggle},
    {"yellow_on",    xcmd_yellow_on},
    {"yellow_off",   xcmd_yellow_off},
    {"yellow_toggle", xcmd_yellow_toggle},

    //MPK
    {"set_pot",      xcmd_set_pot},

    // XMesh command here for now...
    {"light_path",   xmesh_cmd_light_path},
    {NULL, NULL}
};

/** Valid reference names for XSensor/XCommand from the command line. */
char *xsensor_app_keywords [] = {
    "do", "cmd", "xcmd", "xcommand", "XCommand",
    "sensor", "xsensor", "XSensor",
    NULL
};

XAppHandler xsensor_app_desc =
{
    AMTYPEXCOMMAND,
    "$Id: cmd_XSensor.c,v 1.5 2005/02/02 05:47:40 husq Exp $",
    xsensor_cmd_list,
    xsensor_app_keywords
};

void initialize_XSensor() {
    xpacket_add_type(&xsensor_app_desc);
}

```

**2.3. xbow/beta/tools/src/xlisten/boards directory.**

```

/**
 * Handles conversion to engineering units of mda300 packets.
 *
 * @file      mda300.c
 * @author    Martin Turon
 * @version   2004/3/23      mturon      Initial version
 *
 * Copyright (c) 2004 Crossbow Technology, Inc.  All rights reserved.
 *
 * $Id: mda300.c,v 1.29 2005/01/29 00:47:41 mturon Exp $
 */

#include <math.h>

#ifdef __arm__
#include <sys/types.h>
#endif

#include "../xsensors.h"
#include "../timestamp/timestamp.h"

/** MDA300 XSensor packet 1 — contains single analog adc channels */
typedef struct {
    uint16_t adc0;
    uint16_t adc1;
    uint16_t adc2;
    uint16_t adc3;
    uint16_t adc4;
    uint16_t adc5;
    uint16_t adc6;
} XSensorMDA300Data1;

/** MDA300 XSensor packet 2 — contains precision analog adc channels. */
typedef struct {
    uint16_t adc7;
    uint16_t adc8;
    uint16_t adc9;
    uint16_t adc10;
    uint16_t adc11;
    uint16_t adc12;
    uint16_t adc13;
} XSensorMDA300Data2;

/** MDA300 XSensor packet 3 — contains digital channels. */
typedef struct {
    uint16_t digi0;
    uint16_t digi1;
    uint16_t digi2;
    uint16_t digi3;
    uint16_t digi4;
    uint16_t digi5;

```

```

} XSensorMDA300Data3;

/** MDA300 XSensor packet 4 — contains misc other sensor data. */
typedef struct {
    uint16_t battery;
    XSensorSensirion sensirion;
    uint16_t counter;
} XSensorMDA300Data4;

/** MDA300 XSensor packet 5 — contains MultiHop packets. */
typedef struct {
    uint16_t seq_no;
    uint16_t adc0;
    uint16_t adc1;
    uint16_t adc2;
    uint16_t battery;
    XSensorSensirion sensirion;
} __attribute__((packed)) XSensorMDA300Data5;

//pp: multihop need only the packet6
typedef struct XSensorMDA300Data6 {
    uint16_t vref;
    uint16_t humid;
    uint16_t humtemp;
    uint16_t adc0;
    uint16_t adc1;
    uint16_t adc2;
    uint16_t dig0;
    uint16_t dig1;
    uint16_t dig2;
} __attribute__((packed)) XSensorMDA300Data6;
extern XPacketHandler mda300_packet_handler;

/**
 * MDA300 Specific outputs of raw readings within a XSensor packet.
 *
 * @author    Martin Turon
 *
 * @version   2004/3/23      mturon      Initial version
 */
void mda300_print_raw(XbowSensorboardPacket *packet)
{
    switch (packet->packet_id) {
        case 1: {
            XSensorMDA300Data1 *data = (XSensorMDA300Data1 *)packet->data;
            printf("mda300 id=%02x a0=%04x a1=%04x a2=%04x a3=%04x "
                "a4=%04x a5=%04x a6=%04x\n",
                packet->node_id, data->adc0, data->adc1,
                data->adc2, data->adc3, data->adc4,
                data->adc5, data->adc6);
            break;
        }
    }
}

```

```

case 2: {
    XSensorMDA300Data2 *data = (XSensorMDA300Data2 *)packet->data;
    printf("mda300 id=%02x a7=%04x a8=%04x a9=%04x a10=%04x "
           "a11=%04x a12=%04x a13=%04x\n",
           packet->node_id, data->adc7, data->adc8,
           data->adc9, data->adc10, data->adc11,
           data->adc12, data->adc13);
    break;
}

case 3: {
    XSensorMDA300Data3 *data = (XSensorMDA300Data3 *)packet->data;
    printf("mda300 id=%02x d1=%04x d2=%04x d3=%04x d4=%04x d5=%04x",
           packet->node_id, data->digi0, data->digi1,
           data->digi2, data->digi3, data->digi4, data->digi5);
    break;
}

case 4: {
    XSensorMDA300Data4 *data = (XSensorMDA300Data4 *)packet->data;
    printf("mda300 id=%02x bat=%04x hum=%04x temp=%04x cntr=%04x\n",
           packet->node_id, data->battery, data->sensirion.humidity,
           data->sensirion.thermistor, data->counter);
    break;
}

case 5: {
    XSensorMDA300Data5 *data = (XSensorMDA300Data5 *)packet->data;
    printf("mda300 id=%02x bat=%04x hum=%04x temp=%04x "
           "echo10=%04x echo20=%04x soiltemp=%04x\n",
           packet->node_id, data->battery,
           data->sensirion.humidity, data->sensirion.thermistor,
           data->adc0, data->adc1, data->adc2);
    break;
}

case 6: {
    XSensorMDA300Data6 *data = (XSensorMDA300Data6 *)packet->data;
    printf("mda300 id=%02x bat=%04x hum=%04x temp=%04x "
           "adc0=%04x adc1=%04x adc2=%04x\n"
           "dig0=%04x dig1=%04x dig2=%04x\n",
           packet->node_id, data->vref,
           data->humid, data->humtemp,
           data->adc0, data->adc1, data->adc2,
           data->dig0, data->dig1, data->dig2);
    break;
}

default:
    printf("mda300 error: unknown packet_id (%i)\n", packet->packet_id);
}
}

/** MDA300 specific display of converted readings for packet 1 */

```

```

void mda300_print_cooked_1(XbowSensorboardPacket *packet)
{
    printf("MDA300 [sensor data converted to engineering units]:\n"
        "  health:      node id=%i packet=%i\n"
        "  adc chan 0: voltage=%i mV\n"
        "  adc chan 1: voltage=%i mV\n"
        "  adc chan 2: voltage=%i mV\n"
        "  adc chan 3: voltage=%i mV\n"
        "  adc chan 4: voltage=%i mV\n"
        "  adc chan 5: voltage=%i mV\n"
        "  adc chan 6: voltage=%i mV\n\n",
        packet->node_id, packet->packet_id,
        xconvert_adc_single(packet->data[0]),
        xconvert_adc_single(packet->data[1]),
        xconvert_adc_single(packet->data[2]),
        xconvert_adc_single(packet->data[3]),
        xconvert_adc_single(packet->data[4]),
        xconvert_adc_single(packet->data[5]),
        xconvert_adc_single(packet->data[6]));
}

/** MDA300 specific display of converted readings for packet 2 */
void mda300_print_cooked_2(XbowSensorboardPacket *packet)
{
    printf("MDA300 [sensor data converted to engineering units]:\n"
        "  health:      node id=%i packet=%i\n"
        "  adc chan 7: voltage=%i uV\n"
        "  adc chan 8: voltage=%i uV\n"
        "  adc chan 9: voltage=%i uV\n"
        "  adc chan 10: voltage=%i uV\n"
        "  adc chan 11: voltage=%i mV\n"
        "  adc chan 12: voltage=%i mV\n"
        "  adc chan 13: voltage=%i mV\n\n",
        packet->node_id, packet->packet_id,
        xconvert_adc_precision(packet->data[0]),
        xconvert_adc_precision(packet->data[1]),
        xconvert_adc_precision(packet->data[2]),
        xconvert_adc_precision(packet->data[3]),
        xconvert_adc_single(packet->data[4]),
        xconvert_adc_single(packet->data[5]),
        xconvert_adc_single(packet->data[6]));
}

/** MDA300 specific display of converted readings for packet 3 */
void mda300_print_cooked_3(XbowSensorboardPacket *packet)
{
    printf("MDA300 [sensor data converted to engineering units]:\n"
        "  health:      node id=%i packet=%i\n\n",
        packet->node_id, packet->packet_id);
}

/** MDA300 specific display of converted readings for packet 4 */
void mda300_print_cooked_4(XbowSensorboardPacket *packet)

```

```

{
  XSensorMDA300Data4 *data = (XSensorMDA300Data4 *)packet->data;
  printf("MDA300 [sensor data converted to engineering units]:\n"
        "  health:      node id=%i packet=%i\n"
        "  battery voltage:  =%i mV \n"
        "  temperature:    =%0.2f C \n"
        "  humidity:      =%0.1f %% \n\n" ,
        packet->node_id , packet->packet_id ,
        xconvert_battery_mica2(data->battery) ,
        xconvert_sensirion_temp(&(data->sensirion)) ,
        xconvert_sensirion_humidity(&(data->sensirion))
  );
}

/** MDA300 specific display of converted readings for packet 5 */
void mda300_print_cooked_5(XbowSensorboardPacket *packet)
{
  XSensorMDA300Data5 *data = (XSensorMDA300Data5 *)packet->data;
  printf("MDA300 [sensor data converted to engineering units]:\n"
        "  health:      node id=%i parent=%i battery=%i mV seq_no=%i\n"
        "  echo10: Soil Moisture=%0.2f %%\n"
        "  echo20: Soil Moisture=%0.2f %%\n"
        "  soil temperature  =%0.2f F\n"
        "  temperature:    =%0.2f C \n"
        "  humidity:      =%0.1f %% \n\n" ,
        packet->node_id , packet->parent ,
        xconvert_battery_mica2(data->battery) , data->seq_no ,
        xconvert_echo10(data->adc0) ,
        xconvert_echo20(data->adc1) ,
        xconvert_spectrum_soiltemp(data->adc2) ,
        xconvert_sensirion_temp(&(data->sensirion)) ,
        xconvert_sensirion_humidity(&(data->sensirion))
  );
}

/** MDA300 specific display of converted readings for packet 5 */
void mda300_print_cooked_6(XbowSensorboardPacket *packet)
{
  XSensorMDA300Data6 *data = (XSensorMDA300Data6 *)packet->data;
  XSensorSensirion  xsensor;
  xsensor.humidity=data->humid;
  xsensor.thermistor=data->humtemp;

  char timestring [TIMESTRING_SIZE];
  Timestamp *time_now = timestamp_new();
  timestamp_get_string(time_now , timestring);
  //printf("%s" , timestring);
  timestamp_delete(time_now);

  if(xconvert_battery_mica2(data->vref) == 0)
  {
    printf("MDA300 - Cooked\n"
          "\tTime\t\t\tID\tParent\tTemp(C)\t%RH\t\n"

```

```

        "data\t%s\t%i\t%i\t%0.2f\t%0.1f\tIMPACT\n\n",
        timestring,
        packet->node_id,
        packet->parent,
        xconvert_sensirion_temp(&(xsensor)),
        xconvert_sensirion_humidity(&(xsensor))
    );
}

else
{
    printf("MDA300 - Cooked\n"
        "\tTime\t\t\tID\tParent\tTemp(C)\t%RH\tBat (mV)\tADC7(mV)\n"
        "data\t%s\t%i\t%i\t%0.2f\t%0.1f\t%i\t\t%0.3f\n\n",
        timestring,
        packet->node_id,
        packet->parent,
        xconvert_sensirion_temp(&(xsensor)),
        xconvert_sensirion_humidity(&(xsensor)),
        xconvert_battery_mica2(data->vref),
        (float)xconvert_adc_precision(data->adc0)/1000
    );
}

/*
printf("MDA300 [sensor data converted to engineering units]:\n"
    "  health:      node id=%i parent=%i battery=%i mV\n"
    "  echo10: Soil Moisture=%0.2f %%\n"
    "  echo20: Soil Moisture=%0.2f %%\n"
    "  soil temperature  =%0.2f F\n"
    "  temperature:      =%0.2f C \n"
    "  humidity:         =%0.1f %% \n\n",
        packet->node_id, packet->parent,
        xconvert_battery_mica2(data->vref),
        xconvert_echo10(data->adc0),
        xconvert_echo20(data->adc1),
        xconvert_spectrum_soiltemp(data->adc2),
        xconvert_sensirion_temp(&(xsensor)),
        xconvert_sensirion_humidity(&(xsensor))
    );
*/
}

/** MDA300 specific display of converted readings from an XSensor packet. */
void mda300_print_cooked(XbowSensorboardPacket *packet)
{
    switch (packet->packet_id) {
        case 1:
            mda300_print_cooked_1(packet);
            break;

        case 2:

```

```
        mda300_print_cooked_2(packet);
        break;

    case 3:
        mda300_print_cooked_3(packet);
        break;

    case 4:
        mda300_print_cooked_4(packet);
        break;

    case 5:
        mda300_print_cooked_5(packet);
        break;

    case 6:
        mda300_print_cooked_6(packet);
        break;

    default:
        printf("MDA300 Error: unknown packet id (%i)\n\n", packet->packet_id);
    }
}

XPacketHandler mda300_packet_handler =
{
    XTYPE_MDA300,
    "$Id: mda300.c,v 1.29 2005/01/29 00:47:41 mturon Exp $",
    mda300_print_raw,
    mda300_print_cooked,
    mda300_print_raw,
    mda300_print_cooked,
};

void mda300_initialize() {
    xpacket_add_type(&mda300_packet_handler);
}
```

## 2.4. xbow/beta/tools/src/xlisten directory.

```
# Makefile for xlisten # $Id: Makefile,v 1.24 2005/01/28 05:19:24 mturon Exp $
```

```
CC      = gcc
ARMCC   = arm-linux-gcc
LFLAGS  = -lm
CFLAGS  = -O2 -Wall -Wno-format
```

```
# Main xlisten sources
SRCS = xlisten.c xpacket.c xconvert.c
SRCS += xserial.c xsocket.c
```

```
# Add Mote Sensor board support
SRCS += boards/mica2.c boards/mica2dot.c boards/micaz.c
```

```
# Add Mote Data Acquisition board support
SRCS += boards/mda300.c
```

```
# Add Mica2 integrated sensorboards
SRCS += boards/msp410.c
```

```
# Add support for "virtual" board that XsensorTutorial
# uses during Training seminar
```

```
# Add AM types
SRCS += amtypes/health.c amtypes/surge.c
```

```
SRCS += timestamp/timestamp.c
```

```
all: xlisten
```

```
xlisten: $(SRCS)
        $(CC) $(CFLAGS) -o $@ $(SRCS) $(LFLAGS)
```

```
xlisten-arm: $(SRCS)
        $(ARMCC) -I$(INCDIR) $(CFLAGS) -o $@ $(SRCS) -L$(LIBDIR) $(LFLAGS)
```

```
clean:
        rm -f *.o boards/*.o xlisten xlisten-arm xlisten.exe
```

```

/**
 * Listens to the serial port, and outputs sensor data in human readable form.
 *
 * @file      xlisten.c
 * @author    Martin Turon
 * @version   2004/3/10   mturon   Initial version
 *
 * Copyright (c) 2004 Crossbow Technology, Inc. All rights reserved.
 *
 * $Id: xlisten.c,v 1.17 2004/11/18 04:45:10 mturon Exp $
 */

#include "xsensors.h"

static const char *g_version =
    "$Id: xlisten.c,v 1.17 2004/11/18 04:45:10 mturon Exp $";

/** A structure to store parsed parameter flags. */
typedef union {
    unsigned flat;

    struct {
        // output display options
        unsigned display_raw      : 1;  //!< raw TOS packets
        unsigned display_parsed  : 1;  //!< pull out sensor readings
        unsigned display-cooked  : 1;  //!< convert to engineering units
        unsigned export_parsed   : 1;  //!< output comma delimited fields
        unsigned export-cooked   : 1;  //!< output comma delimited fields
        unsigned log_parsed      : 1;  //!< log output to database
        unsigned log-cooked      : 1;  //!< log output to database
        unsigned display_time    : 1;  //!< display timestamp of packet
        unsigned display_ascii   : 1;  //!< display packet as ASCII characters
        unsigned display_rsvd    : 7;  //!< pad first word for output options

        // modes of operation
        unsigned display_help    : 1;
        unsigned display_baud    : 1;  //!< baud was set by user
        unsigned mode_debug      : 1;  //!< debug serial port
        unsigned mode_quiet      : 1;  //!< suppress headers
        unsigned mode_version    : 1;  //!< print versions of all modules
        unsigned mode_header     : 1;  //!< user using custom packet header
        unsigned mode_socket     : 1;  //!< connect to a serial socket
        unsigned mode_sf         : 1;  //!< connect to a serial forwarder
        unsigned mode_framing    : 2;  //!< auto=0, framed=1, unframed=2
    } bits;

    struct {
        unsigned short output;  //!< one output option required
        unsigned short mode;
    } options;
} s_params;

/** A variable to store parsed parameter flags. */

```



```

        break;

    case 'x':
        switch (argv[argc][2]) {
            case 'c': g_params.bits.export_cooked = 1; break;
            default:
            case 'r': g_params.bits.export_parsed = 1; break;
        }
        break;

    case 'f':
        switch (argv[argc][2]) {
            case '=': // specify arbitrary offset
                g_params.bits.mode_framing = atoi(argv[argc]+3)&3;
                break;

            case 'a': // automatic deframing
                g_params.bits.mode_framing = 0;
                break;

            case '0':
            case 'n': // assume no framing
                g_params.bits.mode_framing = 2;
                break;

            case '1': // force framing
                g_params.bits.mode_framing = 1;
                break;
        }
        break;

    default:
        g_params.bits.mode_framing = 1;
        break;
}

case 'w':
case 'h': {
    int offset = XPACKET_DATASTART_MULTIHOP;
    g_params.bits.mode_header = 1;
    switch (argv[argc][2]) {
        case '=': // specify arbitrary offset
            offset = atoi(argv[argc]+3);
            break;
        case '0': // direct uart (no wireless)
        case '1': // single hop offset
            offset = XPACKET_DATASTART_STANDARD;
            break;
    }
    xpacket_set_start(offset);
    break;
}

case 'l':
    g_params.bits.log_cooked = 1;
    if (argv[argc][2] == '=') {

```

```

    }
    break;

    case 'b':
        if (argv[argc][2] == '=') {
            baudrate = xserial_set_baud(argv[argc]+3);
            g_params.bits.display_baud = 1;
        }
        break;

    case 's':
        switch (argv[argc][2]) {
case '=':
            xserial_set_device(argv[argc]+3);
            break;

case 'f':
            g_params.bits.mode_sf = 1;
            g_params.bits.mode_socket = 1;
            if (argv[argc][3] == '=') {
server = argv[argc]+4;
port = strchr(server, ':');
if (port) {
    *port++ = '\0';
    xsocket_set_port(port);
}
xsocket_set_server(server);
}
            break;

        }
        break;

case 't':
            g_params.bits.display_time = 1;
            break;

            case 'i':
                g_params.bits.mode_sf = 0;
                g_params.bits.mode_socket = 1;
                if (argv[argc][2] == '=') {
server = argv[argc]+3;
port = strchr(server, ':');
if (port) {
    *port++ = '\0';
    xsocket_set_port(port);
}
                xsocket_set_server(server);
            }
            break;

            case 'v':
                g_params.bits.mode_version = 1;
                break;

```

```

        case 'd':
            g_params.bits.mode_debug = 1;
            break;
    }
}
argc--;
}

if (!g_params.bits.mode_quiet) {
    // Summarize parameter settings
    printf(" xlisten Ver:%s\n", g_version);
    if (g_params.bits.mode_version)    xpacket_print_versions();
    printf(" Using params: ");
    if (g_params.bits.display_help)    printf("[help] ");
    if (g_params.bits.display_baud)    printf("[baud=0x%04x] ", baudrate);
    if (g_params.bits.display_raw)     printf("[raw] ");
    if (g_params.bits.display_ascii)   printf("[ascii] ");
    if (g_params.bits.display_parsed)  printf("[parsed] ");
    if (g_params.bits.display_cooked)  printf("[cooked] ");
    if (g_params.bits.export_parsed)   printf("[export] ");
    if (g_params.bits.display_time)    printf("[timed] ");
    if (g_params.bits.export_cooked)   printf("[convert] ");
    if (g_params.bits.log_cooked)      printf("[logging] ");
    if (g_params.bits.mode_framing==1) printf("[framed] ");
    if (g_params.bits.mode_framing==2) printf("[unframed] ");
    if (g_params.bits.mode_header)     printf("[header=%i] ",
        xpacket_get_start());
    if (g_params.bits.mode_socket)     printf("[inet=%s:%u] ",
        xsocket_get_server(),
        xsocket_get_port());
    if (g_params.bits.mode_debug) {
        printf("[debug - serial dump!] \n");
        xserial_port_dump();
    }
    printf("\n");
}

if (g_params.bits.display_help) {
    printf(
        "\nUsage: xlisten <-?|r|p|c|x|l|d|v|q> <-l=table>"
        "\n                <-s=device> <-b=baud> <-i=server:port>"
        "\n  -? = display help [help]"
        "\n  -r = raw display of tos packets [raw]"
        "\n  -a = ascii display of tos packets [ascii]"
        "\n  -p = parse packet into raw sensor readings [parsed]"
        "\n  -c = convert data to engineering units [cooked]"
        "\n  -l = log data to database or file [logged]"
        "\n  -xr = export raw readings in csv spreadsheet format [export]"
        "\n  -xc = export cooked in csv spreadsheet format [export]"
        "\n  -d = debug serial port by dumping bytes [debug]"
        "\n  -b = set the baudrate [baud=#|mica2|mica2dot]"
        "\n  -s = set serial port device [device=com1]"
    );
}

```

```

        "\n  -i = use socket input [inet=host:port]"
        "\n  -sf = use serial forwarder input [inet=host:port]"
        "\n  -f = specify framing (0=auto|1=on|2=off)"
        "\n  -h = specify header size [header=offset]"
        "\n  -t = display time packet was received [timed]"
        "\n  -q = quiet mode (suppress headers)"
        "\n  -v = show version of all modules"
        "\n"
    );
    exit(0);
}

/* Default to displaying packets as raw, parsed, and cooked. */
if (g_params.options.output == 0) {
    g_params.bits.display_raw = 1;
    g_params.bits.display_parsed = 1;
    g_params.bits.display_cooked = 1;
}

/* Stream initialization */

// Set STDOUT and STDERR to be line buffered, so output is not delayed.
setlinebuf(stdout);
setlinebuf(stderr);

if (g_params.bits.mode_socket) {
    g_istream = xsocket_port_open();
} else {
    g_istream = xserial_port_open();
}
}

int xmain_get_verbose() {
    return !g_params.bits.mode_quiet;
}

/**
 * The main entry point for the sensor listener console application.
 *
 * @param   argc           Argument count
 * @param   argv           Argument vector
 *
 * @author   Martin Turon
 * @version  2004/3/10     mturon     Initial version
 */
int main(int argc, char **argv)
{
    int length;
    unsigned char buffer[255];

    parse_args(argc, argv);

    while (1) {

```

```

if (g_params.bits.mode_sf) {
    // Serial forwarder read
    length = xsocket_read_packet(g_istream, buffer);
} else {
    // Serial read direct, or over socket (mib600)
    length = xserial_port_read_packet(g_istream, buffer);
}

    if (length < XPACKET_MIN_SIZE)
continue;    // ignore partial packets and packetizer frame end

    if (g_params.bits.display_time)    xpacket_print_time();

    if (g_params.bits.display_raw)    xpacket_print_raw(buffer, length);

    if (g_params.bits.display_ascii)    xpacket_print_ascii(buffer, length);

if (!g_params.bits.mode_sf)
xpacket_decode(buffer, length, g_params.bits.mode_framing);

    if (g_params.bits.display_parsed)    xpacket_print_parsed(buffer);

    if (g_params.bits.export_parsed)    xpacket_export_parsed(buffer);

    if (g_params.bits.export_cooked)    xpacket_export_cooked(buffer);

    if (g_params.bits.log_cooked)    xpacket_log_cooked(buffer);

    if (g_params.bits.display_cooked)    xpacket_print_cooked(buffer);
}
}

```

```

##### User Manual Follows #####

```

```

/**

```

```

@mainpage XListen Documentation

```

```

@section version Version

```

```

$Id: xlisten.c,v 1.17 2004/11/18 04:45:10 mturon Exp $

```

```

@section usage Usage

```

```

Usage: xlisten <-?|r|p|c|x|l|d|v|q> <-b=baud> <-s=device> <-h=size>

```

```

@n

```

```

@n    -? = display help [help]

```

```

@n    -r = raw display of tos packets [raw]

```

```

@n    -p = parse packet into raw sensor readings [parsed]

```

```

@n    -x = export readings in csv spreadsheet format [export]

```

```

@n    -c = convert data to engineering units [cooked]

```

```

@n    -t = display time packet was received [timed]

```

```

@n    -a = ascii display of tos packets [ascii]

```

```

@n    -l = log data to database [logged]

```

```

@n    -d = debug serial port by dumping bytes [debug]

```

```

@n      -b = set the baudrate [baud=#|mica2|mica2dot]
@n      -s = set serial port device [device=com1]
@n      -i = use socket input [inet=host:port]
@n      -sf = use serial forwarder input [inet=host:port]
@n      -h = specify size of TOS_msg header [header=size]
@n      -v = display complete version information for all modules [version]
@n      -q = quiet mode (suppress headers)
@n
@section params Parameters

```

```

@subsection help -? [help]

```

XListen has many modes of operation that can be controlled by passing command line parameters. The current list of these command line options and a brief usage explanation is always available by passing the `-?` flag.

```

@n

```

```

@n A detail explanation of each command line option as of version 1.7 follows.

```

```

@subsection baud -b=baudrate [baud]

```

This flag allows the user to set the baud rate of the serial line connection. The default baud rate is 57600 bits per second which is compatible with the Mica2. The desired baudrate must be passed as a number directly after the equals sign with no spaces inbetween, i.e. `-b=19200`. Optionally, a product name can be passed in lieu of an actual number and the proper baud will be set, i.e. `-b=mica2dot`. Valid product names are:

```

mica2          (57600 baud)
mica2dot      (19200 baud)

```

```

@subsection serial -s=port [serial]

```

This flag gives the user the ability to specify which COM port or device xlisten should use. The default port is `/dev/ttyS0` or the UNIX equivalent to `COM1`. The given port must be passed directly after the equals sign with no spaces, i.e. `-s=com3`.

```

@subsection internet -i=hostname:port [inet]

```

This flag tells xlisten to attach to a virtual serial connection over a TCP/IP internet socket. Specify the hostname and port to connect in the argument. The default hostname is `localhost`, and the default port `9001`. The keyword `'mib600'` can be passed as an alias to port `10002` when connecting to that hardware device. The hostname and port must be passed directly after the equals sign with no spaces with a optional colon inbetween, i.e. `-i=remote`, `-i=10.1.1.1:9000`, `-i=mymib:mib600`, `-i=:9002`, `-i=localhost:9003`, or `-i=stargate.xbow.com`.

```

@subsection serial forwarder -sf=hostname:port [inet]

```

This flag tells `xlisten` to attach to a serial forwarder connection over a TCP/IP internet socket. The hostname and port arguments are the same as the `-i` flag. The `-sf` flag tells `xlisten` to specifically use the TinyOS serial forwarder protocol.

```
@subsection raw -r [raw]
```

Raw mode displays the actual TOS packets as a sequence of bytes as seen coming over the serial line. Sample output follows:

```
@n $ xlisten -r
@n xlisten Ver: Id: xlisten.c,v 1.7 2004/03/23 00:52:28 mturon Exp
@n Using params: [raw]
@n /dev/ttyS0 input stream opened
@n 7e7e000033000000c8035f61d383036100000000e4510d610000000080070000d4b5f577
@n 7e00007d1d8101060029091e09ef082209e7080b09b40800000000000000000000100
@n 7e00007d1d81020600f007de07da07d507c30647065405000000000000000000000100
```

```
@subsection parsed -p [parsed]
```

Parsed mode attempts to interpret the results of the incoming TOS packets and display information accordingly. The first stage of the parsing is to look for a valid `sensorboard_id` field, and display the part number.

The

`node_id` of the packet sender is also pulled out and displayed. Finally, raw sensor readings are extracted and displayed with some designation as to their meaning:

```
@n $ xlisten -p -b=mica2dot
@n xlisten Ver: Id: xlisten.c,v 1.7 2004/03/23 00:52:28 mturon Exp
@n Using params: [baud=0x000e] [parsed]
@n /dev/ttyS0 input stream opened
@n mda500 id=06 bat=00c1 thrm=0203 a2=019c a3=0149 a4=011d a5=012b a6=011b a7=0147
@n mda500 id=06 bat=00c2 thrm=0203 a2=019d a3=014d a4=011e a5=0131 a6=011b a7=0140
@n mda500 id=06 bat=00c2 thrm=0204 a2=0199 a3=014c a4=0125 a5=012a a6=011f a7=0147
@n mda500 id=06 bat=00c2 thrm=0204 a2=0198 a3=0148 a4=0122 a5=0131 a6=012d a7=0143
@n mda500 id=06 bat=00c2 thrm=0203 a2=019e a3=014e a4=0124 a5=012b a6=011c a7=0143
@n mda500 id=06 bat=00c2 thrm=0204 a2=019d a3=014c a4=011f a5=0135 a6=0133 a7=011d
@n mda500 id=06 bat=00c2 thrm=0205 a2=019a a3=014c a4=011e a5=0131 a6=012d a7=011c
```

```
@subsection cooked -c [cooked]
```

Cooked mode actually converts the raw sensor readings within a given packet into engineering units. Sample output follows:

```
@n $ xlisten -c -b=mica2dot
@n xlisten Ver: Id: xlisten.c,v 1.7 2004/03/23 00:52:28 mturon Exp
@n Using params: [baud=0x000e] [cooked]
@n /dev/ttyS0 input stream opened
@n MDA500 [sensor data converted to engineering units]:
@n   health:      node id=6
@n   battery:     volts=3163 mv
@n   thermistor:  resistance=10177 ohms, temperature=24.61 C
```

### 33MICA2-BASED WIRELESS ACM VERSION 3 SOFTWARE: MODIFICATION VERSION 2 XMDA300: *SHAKE*

```
@n   adc chan 2: voltage=1258 mv
@n   adc chan 3: voltage=1001 mv
@n   adc chan 4: voltage=893  mv
@n   adc chan 5: voltage=939  mv
@n   adc chan 6: voltage=875  mv
@n   adc chan 7: voltage=850  mv
```

@subsection quiet -q [quiet]

This flag suppresses the standard xlisten header which displays the version string and parameter selections.

@subsection export -x [export]

Export mode displays raw adc values as comma delimited text for use in spreadsheet and data manipulation programs. The user can pipe the output of xlisten in export mode to a file and load that file into Microsoft Excel to build charts of the information. Sample output follows:

```
@n $ xlisten -b=mica2dot -q -x
@n 51200,24323,54113,899,97,0,58368,3409
@n 6,193,518,409,328,283,296,298
@n 6,194,517,410,330,292,310,300
@n 6,194,518,409,329,286,309,288
@n 6,194,517,411,331,287,297,300
@n 6,194,516,413,335,288,301,287
```

@subsection timed -t [timed]

Displays the time at which the packet was received.

```
@n $ xlisten -t
@n [2004/09/29 10:24:29]
@n [2004/09/29 10:36:57]
```

@subsection ascii -a [ascii]

Displays the raw packet contents as ASCII characters.

@subsection logging -l [logged]

Logs incoming readings to a Postgres database. Default connection settings are: server=localhost, port=5432, user=tele, pass=tiny.

@subsection header -h=size [header]

Passing the header flag tells xlisten to use a different offset when parsing packets that are being forwarded by TOSBase. Generally this flag is not required as xlisten autodetects the header size from the AM type. When this flag is passed all xlisten will assume all incoming packets have a data payload beginning after the header size offset.

@subsection versions -v [versions]

Displays complete version information for all sensorboard decoding modules

within xlisten.

```
@n $ xlisten -v
@n xlisten Ver: Id: xlisten.c,v 1.11 2004/08/04 21:06:41 mturon Exp
@n 87: Id: mep401.c,v 1.10 2004/08/04 21:06:41 mturon Exp
@n 86: Id: mts400.c,v 1.15 2004/08/04 21:06:41 husq Exp
@n 85: Id: mts400.c,v 1.15 2004/08/04 21:06:41 mturon Exp
@n 84: Id: mts300.c,v 1.14 2004/08/04 21:06:41 husq Exp
@n 83: Id: mts300.c,v 1.14 2004/08/04 21:06:41 mturon Exp
@n 82: Id: mts101.c,v 1.5 2004/08/04 21:06:41 husq Exp
@n 81: Id: mda300.c,v 1.4 2004/08/04 17:15:22 jdprabhu Exp
@n 80: Id: mda500.c,v 1.11 2004/08/04 21:06:41 husq Exp
@n 03: Id: mep500.c,v 1.3 2004/08/04 21:06:41 mturon Exp
@n 02: Id: mts510.c,v 1.6 2004/08/04 21:06:41 husq Exp
@n 01: Id: mda500.c,v 1.11 2004/08/04 21:06:41 abroad Exp
```

@subsection debug -d [debug]

This flag puts xlisten in a mode so that it behaves exactly like the TinyOS raw listen tool (`tinys-1.x/tools/src/raw_listen.c`). All other command line options except `-b` [baud] and `-s`[serial] will be ignored. This mode is mainly used for compatibility and debugging serial port issues. Individual bytes will be displayed as soon as they are read from the serial port with no post-processing. In most cases `-r` [raw] is equivalent and preferred to using debug mode.

@subsection display Display Options

The `-r`, `-p`, and `-c` flags are considered display options. These can be passed in various combinations to display multiple views of the same packet at once. The default display mode when xlisten is invoked with no arguments is `-r`. What follows is sample output for all three display options turned on at once:

```
@n $ xlisten -b=mica2dot -r -p -c
@n xlisten Ver: Id: xlisten.c,v 1.7 2004/03/23 00:52:28 mturon Exp
@n Using params: [baud=0x000e] [raw] [parsed] [cooked]
@n /dev/ttyS0 input stream opened
@n 7e7e000033000000c8035f61d383036100000000e4510d61000000080070000d4b5f577
@n 7e00007d1d01010600c200050293014401210135012f01220100000000000000000100
@n mda500 id=06 bat=00c2 thrm=0205 a2=0193 a3=0144 a4=0121 a5=0135 a6=012f a7=0122
@n MDA500 [sensor data converted to engineering units]:
@n   health:      node id=6
@n   battery:     volts=3163 mv
@n   thermistor:  resistance=10217 ohms, temperature=24.53 C
@n   adc chan 2:  voltage=1246 mv
@n   adc chan 3:  voltage=1001 mv
@n   adc chan 4:  voltage=893 mv
@n   adc chan 5:  voltage=955 mv
@n   adc chan 6:  voltage=936 mv
@n   adc chan 7:  voltage=896 mv
```

@section building Build Process

The source code for the `xlisten` tool is located at:  
`/opt/tinyos-1.x/contrib/xbow/tools/src/xlisten.`

@n@n

To build the tool, change to the `xlisten` source directory and run `'make'`.

@n@n

To get the latest version of the source, change to the `xlisten` source directory and run `'cvs update'`.

@section setup Setup

XListen is a command line tool that can be run from a cygwin shell by simply typing `'xlisten'`. The executable needs to be in your working path to use it. A simple way to add `xlisten` to your working path is to create a soft link to it by running the following command:

```
@n$ ln -s /opt/tinyos-1.x/contrib/xbow/tools/src/xlisten /usr/local/bin/xlisten
```

@n@n

You can use `xlisten` to read sensor data from either one mote over a serial link, or a wireless network of motes. In both configurations, you need to have a MIB510 board connected via a serial cable to your PC.

@n@n

For a single mote configuration, the mote must be programmed with a `XSensorMXX###` application and plugged into the MIB510. The mote will stream packets over the UART whenever it has power.

@n@n

For the network of motes configuration, a base station mote needs to be programmed with `TOSBase` and plugged into the MIB510. All other motes need to be installed with an `XSensorMXX##` application and put within range of the base station or a valid multi-hop peer. `Xlisten` must then be run with the `-w` flag to properly parse the wireless packets. Take care to program all the motes to the same frequency and group id.

\*/

## CHAPTER 4

### **ēKo mote-based wireless ACPS software**

This appendix contains the XML files that were created for use with the CPA and CPC crack propagation patterns. They are placed on the ēKo base station in the `/usr/xbow/xserve/configxml` directory

## 1. CPA crack propagation pattern: cpa.xml

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE XServeConfig SYSTEM "../xserve_config.dtd">
<XServeConfig>
  <XFieldExtractor name="Vishay Crack Propagation Sensor (narrow spacing)" order="3">
    <XFields>
      <!-- Tos Hdr -->
      <XField name="amType" byteoffset="2" length="1" type="uint8"/>
      <XField name="group" byteoffset="3" length="1" type="uint8"/>
      <!-- XMesh Hdr -->
      <XField name="nodeId" byteoffset="7" length="2" type="uint16"
        specialtype="nodeid"/>
      <XField name="socketId" byteoffset="11" length="1" type="uint8"/>
      <!-- XSensor Hdr -->
      <XField name="boardId" byteoffset="12" length="1" type="uint8"
        specialtype="sensorboardid"/>
      <XField name="packetId" byteoffset="13" length="1" type="uint8"/>
      <XField name="ParentID" byteoffset="14" length="1" type="uint16"/>

      <!-- Data -->
      <XField name="E1ReferenceADC" byteoffset="16" length="2" type="uint16"/>
      <XField name="E1ExcitationV" byteoffset="18" length="2" type="uint16">
        <XConversion function="1.225*2*x/y" returntype="float">
          <XConvParam variablename="x" fieldname="E1ExcitationV" type="float"/>
          <XConvParam variablename="y" fieldname="E1ReferenceADC" type="float"/>
        </XConversion>
      </XField>

      <!-- potentiometer conversion - report volts -->
      <XField name="RangeV" byteoffset="20" length="2" type="uint16">
        <XConversion function="1.225*x/z" returntype="float">
          <XConvParam variablename="x" fieldname="RangeV" type="float"/>
          <XConvParam variablename="z" fieldname="E1ReferenceADC" type="float"/>
        </XConversion>
      </XField>
    </XFields>

    <XFilter>
      <!-- LOGIC: SocketID==XSensorEKo AND BoardID (SensorId) AND PacketID==0 -->
      <XCondAnd>
        <XCond name="IsEqual">
          <XFilterParam name="fieldname" value="socketId"/>
          <XFilterParam name="fieldvalue" value="0x34"/>
        </XCond>
        <XCond name="IsEqual">
          <XFilterParam name="fieldname" value="boardId"/>
          <XFilterParam name="fieldvalue" value="162"/>
        </XCond>
      </XCondAnd>
    </XFilter>
  </XFieldExtractor>
</XServeConfig>

```

```

</XCond>
<XCond name="IsEqual">
  <XFilterParam name="fieldname" value="packetId"/>
  <XFilterParam name="fieldvalue" value="0x0"/>
</XCond>
</XCondAnd>

</XFilter>

<XDataSinks>
  <XDataSink name="Generic Print Datasink">

    <XDSPParam name="printstring"
      value="Vishay Crack Propagation
      Sensor (narrow
      spacing)[%s:%s]:\n Parent:%s
      PortID:%s \n RangeV:%s V
      ExcitV:%s RefADC:%s " />
    <XDSPParam name="printfields"
      value="boardId , packetId , ParentID , nodeId , , RangeV , E1ExcitationV ,
      E1ReferenceADC" /> </XDataSink>

  <XDataSink name="Generic File Datasink">
    <XDSPParam name="rawfilename"
      value="Vishay_Crack_Propagation_Narrow_Spacing_Raw.csv"/>
    <XDSPParam name="parsedfilename"
      value="Vishay_Crack_Propagation_Narrow_Spacing_Parsed.csv"/>
    <XDSPParam name="convertedfilename"
      value="Vishay_Crack_Propagation_Narrow_Spacing_Converted.csv"/>
    <XDSPParam name="delim" value="," />
    <XDSPParam name="header" value="yes"/>
    <XDSPParam name="timestamp"
      value="%m-%d-%Y %H:%M:%S" />
    <XDSPParam name="backup" value="yes"/>
  </XDataSink>

  <XDataSink name="Sensor Log Datasink"> <XDSPParam
    name="sensorid" value="162"/>
    <XDSPParam name="tablename"
      value="Vishay_Crack_Propagation_Narrow_Spacing_sensor_results"/>
    <XDSPParam name="sensorname" value="Crack
      Propagation (Narrow Spacing)"/>
    <XDSPParam name="columninfo"
      value="fieldName = nodeId ,
      displayName = Node Id ,
      displayOrder = 1"/>
    <XDSPParam name="columninfo"
      value="fieldName =
      RangeV , displayName = RangeV ,
      displayOrder = 2 , unitName =
      Volts , unitShortName = V ,
      sensorType = Voltage ,
      sensorMinValue = 0 ,

```

```
sensorMaxValue = 6"/> <XDSPParam
name="columninfo"
value="fieldName =
E1ExcitationV , displayName =
Excitation , displayOrder =
3 , unitName = Volts ,
unitShortName = V , sensorType =
Voltage , sensorMinValue = 0 ,
sensorMaxValue = 6"/>
</XDataSink>

</XDataSinks>
</XFieldExtractor>
</XServeConfig>
```

## 2. CPC crack propagation pattern: cpc.xml

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE XServeConfig SYSTEM ".\xserve_config.dtd">
<XServeConfig>
  <XFieldExtractor name="Vishay Crack Propagation Sensor (wide spacing)" order="3">
    <XFields>
      <!-- Tos Hdr -->
      <XField name="amType" byteoffset="2" length="1" type="uint8"/>
      <XField name="group" byteoffset="3" length="1" type="uint8"/>
      <!-- XMesh Hdr -->
      <XField name="nodeId" byteoffset="7" length="2" type="uint16"
        specialtype="nodeid"/>
      <XField name="socketId" byteoffset="11" length="1" type="uint8"/>
      <!-- XSensor Hdr -->
      <XField name="boardId" byteoffset="12" length="1" type="uint8"
        specialtype="sensorboardid"/>
      <XField name="packetId" byteoffset="13" length="1" type="uint8"/>
      <XField name="ParentID" byteoffset="14" length="1" type="uint16"/>

      <!-- Data -->
      <XField name="E1ReferenceADC" byteoffset="16" length="2" type="uint16"/>
      <XField name="E1ExcitationV" byteoffset="18" length="2" type="uint16">
        <XConversion function="1.225*2*x/y" returntype="float">
          <XConvParam variablename="x" fieldname="E1ExcitationV" type="float"/>
          <XConvParam variablename="y" fieldname="E1ReferenceADC" type="float"/>
        </XConversion>
      </XField>

      <!-- potentiometer conversion - report volts -->
      <XField name="RangeV" byteoffset="20" length="2" type="uint16">
        <XConversion function="1.225*x/z" returntype="float">
          <XConvParam variablename="x" fieldname="RangeV" type="float"/>
          <XConvParam variablename="z" fieldname="E1ReferenceADC" type="float"/>
        </XConversion>
      </XField>

    </XFields>

    <XFilter>
      <!-- LOGIC: SocketID==XSensorEKO AND BoardID (SensorId) AND PacketID==0 -->
      <XCondAnd>
        <XCond name="IsEqual">
          <XFilterParam name="fieldname" value="socketId"/>
          <XFilterParam name="fieldvalue" value="0x34"/>
        </XCond>
        <XCond name="IsEqual">
          <XFilterParam name="fieldname" value="boardId"/>
          <XFilterParam name="fieldvalue" value="161"/>
        </XCond>
      </XCondAnd>
    </XFilter>
  </XFieldExtractor>
</XServeConfig>

```

```

</XCond>
<XCond name="IsEqual">
  <XFilterParam name="fieldname" value="packetId"/>
  <XFilterParam name="fieldvalue" value="0x0"/>
</XCond>
</XCondAnd>

</XFilter>

<XDataSinks>

<XDataSink name="Generic Print Datasink">
  <XDSPParam name="printstring"
    value="Vishay Crack Propagation
    Sensor (wide spacing)[%s:%s]:\n
    Parent:%s PortID:%s \n RangeV:%s
    V ExcitV:%s RefADC:%s " />
  <XDSPParam name="printfields"
    value="boardId , packetId , ParentID , nodeId , , RangeV , E1ExcitationV ,
    E1ReferenceADC"/> </XDataSink>

<XDataSink name="Generic File Datasink">
  <XDSPParam name="rawfilename"
    value="Vishay_Crack_Propagation_Wide_Spacing_Raw.csv"/>
  <XDSPParam name="parsedfilename"
    value="Vishay_Crack_Propagation_Wide_Spacing_Parsed.csv"/>
  <XDSPParam name="convertedfilename"
    value="Vishay_Crack_Propagation_Wide_Spacing_Converted.csv"/>
  <XDSPParam name="delim" value=","/>
  <XDSPParam name="header" value="yes"/>
  <XDSPParam name="timestamp"
    value="%m-%d-%Y %H:%M:%S"/>
  <XDSPParam name="backup" value="yes"/>
</XDataSink>

<XDataSink name="Sensor Log Datasink"> <XDSPParam
  name="sensorid" value="161"/>
  <XDSPParam name="tablename"
    value="Vishay_Crack_Propagation_Wide_Spacing_sensor_results"/>
  <XDSPParam name="sensorname" value="Crack
  Propagation (Wide Spacing)"/>
  <XDSPParam name="columninfo"
    value="fieldName = nodeId ,
    displayName = Node Id ,
    displayOrder = 1"/>
  <XDSPParam name="columninfo"
    value="fieldName =
    RangeV , displayName = RangeV ,
    displayOrder = 2 , unitName =
    Volts , unitShortName = V ,
    sensorType = Voltage ,
    sensorMinValue = 0 ,
    sensorMaxValue = 6"/> <XDSPParam

```

```
name="columninfo"  
value="fieldName =  
E1ExcitationV , displayName =  
Excitation , displayOrder =  
3 , unitName = Volts ,  
unitShortName = V , sensorType =  
Voltage , sensorMinValue = 0 ,  
sensorMaxValue = 6"/>  
</XDataSink>  
</XDataSinks>  
</XFieldExtractor>  
</XServeConfig>
```