ECE195:55 Lab II Spring 2005
Building a WSN

Overview

In this lab students will build a complete WSN consisting of a small number of nodes. "Complete" means a functional, cooperating network of nodes, with functional sensors. The node hardware will be supplied and "building" here means developing and then programming the necessary algorithms into the nodes.

The purpose of the WSN is to periodically measure the light level at the node location and forward that to the base station. When a switch that is attached to a node is closed, the node must notify the base station. When a node powers up, it must automatically join the WSN. The WSN will follow a node-to-node communication model rather than node-base communication. When a node power down, the WSN must automatically reconfigure itself.

Building a robust, energy efficient WSN is a large undertaking. Thus, a collection of routines that provide OS-like services will be supplied. Also, a number of simplifying assumptions (see below) will be introduced.

Organizational

Groups consisting of 3 students will work together. Only one group at a time can have access to the equipment. Groups should schedule access to the Sensors Lab in half-day (morning/afternoon) blocks. At the start of each lab I will spend some time showing students the tools. A group can schedule more than one block. **Groups need to demonstrate the working WSN and then write a short report. One lab report per group is due by the end of the semester, Friday, May 5, e-mailed as a PDF document to me.**

Development Tools and Equipment

Node Hardware. The same hardware that was used for Lab 1 will be used for this lab, with the addition of two sensors: a light sensor, and a switch.

Programming. Programming will be done in the C programming language. Skeleton code will be provided, and the development environment for the code-compile-load development cycle will be set up. The HP InfoTech CodeVisionAVR (www.hpinfotech.ro) integrated development environment will be used.

Supplied Library. The library will provide services needed to implement the algorithm(s): send a packet to another node, ping another node to measure RSSI, respond to requests from other nodes, select communication channel, interrogate its sensors and so on. Details on the library are supplied below.

Simplifications

Coordinate System and Localization. We will assume nodes know their location in a simple Cartesian coordinate system. These coordinates will be programmed into the nodes. Node locations will be such that that very simple routing algorithms may be used (i.e., packets will not get stuck at nodes).

Sleep modes. Nodes will be alive (no sleep modes).

WSN Design

The node transceivers provide several mechanisms for constructing the communication links: communication channel, hopping sequence, *destination address*, and *masks* (XCite vernacular). This is detailed in the XCite 900 MHz documentation provided for the previous lab and on the class website (www.engineering.uiowa.edu/~ece195/2005/labNotes.html). We will use the capability to select different channels and the concept of a destination address, and not the masking and hopping sequence capabilities. The destination address will be used as the node ID, and node IDs will be unique.

Approach 1. RF channel 0 will be used for control and configuration information. RF Channel 2 will be used for communication. Upon power up, a node waits for a random time and switches to the network control channel and joins the network—discovers other nodes and their locations. Based on this information it determines its neighbors. The node then switches to the communication channel and listens for, and responds to packets sent to it. Nodes periodically switch to the network control channel and repeat the process. This enables the nodes to discover new nodes and determine which nodes have left the WSN. Nodes measure the light level at set intervals and send the packet to the base station. When the switch attached to a node is closed the node notifies the base.

Approach 2. All nodes use the same channel, Channel 0. Upon power up, a node waits for a random time and then cycles through destination addresses 0–19 and discovers other nodes and their x- and y-coordinates. Based on this information it determines nodes it will communicate with. It then switches to its destination address and listens for, and respond to packets sent to it. Nodes periodically repeat the process to discover new nodes and determine which nodes have left the WSN. Nodes measure the light level at set intervals and send the packet to the base station. When the switch attached to a node is closed the node notifies the base.

Preparation

To prepare for the lab, read through the PDF files for this lab provided on the website. Familiarize yourself with the routines used in the library. Pick one of the approaches outlined above and write pseudo C-code to implement it. Issues to consider include are the following. A node may discover that it has many neighbors (potentially all the nodes) in the network. It has to select a subset of these that it will communicate with. How will it make its selection? What routing strategy will the node implement? That is, consider it receives a request to "please forward this packet to the node at (*x*,*y*)" , and that it has a number of neighbors it can forward the packet to. Which neighbor node will it use? How will it know if the neighbor received the packet? What should be done if the packet is not received?

Report Writing

To complete the lab, document your WSN. A group should write one group report and e-mail that to me as a PDF file. How you present the information is your decision, but here are some things you may want to address. Include the C source code for each node. What was the most challenging part of building the WSN? What are the strengths and weaknesses of your WSN? What is required to add sleep modes to your WSN? Discuss the energy efficiency of your WSN, and comment on how it may be improved.

An Alternative Lab(s)

As alternative lab is to do an RSSI investigation and antenna evaluation for a cell-phone based network that is under development at IIHR. Briefly, this will consist of writing code to communicate with a cell-modem to query RSSI. Then the students will accompany an IIHR employee and visit a number of locations in an around Iowa City, measure RSSI for a few antenna configuration, and document their findings. The pros of this lab are that the code-writing part can be quite simple and that it involves field work. The cons are that it involves field work that will be time-consuming.

I am open to suggestions from students who want to propose their own lab.

Library Routines

Core Routines

Routine	Synopsis
void init(void)	Initializes MCU registers and global variables, and peripherals such as the RTC and LCD. Call once at start of program.
int getSerialNumbers(int *data);	Get the transceiver's destination address, vendor ID, and serial numbers. Always returns 1.
int setAddress(int address)	After this call, the transceiver will communicate with other transceivers with the same *address* (id), and ignore others, even though they may be using the same channel. Always returns 1.
int setChannel(int channel)	Sets the communication channel. Valid values for *channel* are 0, 2…10. Always returns 1.
int getConfig(char *packet, int ack)	Retrieves the node configuration: node address/id, coordinates, time, etc. from *packet*, and reprograms the node. The variable *ack* is ignored in the current version, and the routine always returns 1.
int putConfig(char * packet, int ntry)	Loads the node configuration: node address/id, coordinates, time, etc., into *packet* and output/transmit the packet. The proper destination address and channel must be selected prior to this routine. The variable *ntry* is ignored in this version and the routine always returns 1.
int putPacket(char *packet, int ntry)	Transmits *packet*. The proper destination address and channel must be selected prior to this routine. The user constructs *packet* prior to this routine. The variable *ntry* is ignored in this version and the routine always returns 1.
int getPacket(char *packet, int ack)	If the transceiver received a valid packet, fill *packet* and returns 1. Otherwise, returns 0. This routine will discard characters that are not part of a valid packet. The variable *ack* is ignored in the current version, and the routine always returns 1.
int getLight(void)	Measures the voltage across the photosensor. This routine is partially implemented. One of the tasks for the lab is to complete the implementation so that it returns the light level in lx.

int getSwitch(void)	Returns 0 if the switch is pressed (down) or 1 if it is open.
unsigned char read_adc(unsigned char n)	Reads ADC channel *n*. The ADC is an 8-bit, 115 KHz, ADC with a 5V reference. The return value is a number in range 0 (0V) to 255 (5V). The photosensor is connected to ADC *channe*l 0.

Support Routines

delay_ms(n)	Pauses execution for *n* milliseconds. Note that the transceiver will still receive (and buffer) characters.
void lcd_clear(void)	Clears the LCD screen
void lcd_puts(char *string)	Displays *string* on the LCD
char getchar(void)	Gets a character from the transceiver. If there is no character, waits for one.
void putchar(char c)	Writes a character *c* to the transceiver, which transmits it to the current destination address on the current channel.
printf()	This routine implements the C *stdio.h printf* routine. The output goes to the transceiver, which transmits it to the current destination address on the current channel.
sprintf()	This routine implements the C *stdio.h sprintf* routine for printing to a string.
sscanf();	This routine implements the C *stdio.h sscanf* routine for reading from a string.
int rand (void)	Generates a pseudo-random number between 0 and 32767.
void srand(int seed)	Sets the starting value seed used by the pseudo-random number generator in the rand function.
void setLed(int state)	Turns an indicator LED on (*state* = 1) or off (*stat*e = 0)
void switchToModemMode()	Switches the transceiver to modem mode.
void switchToATMode(void);	Switches the transceiver to AT/command mode.
int convertBase(char *buf, int base);	Converts the character string in *buff*, assumed to be in *base* to its internal binary representation. For example d = ConvertBase("2A",16) sets *d* to 42 decimal.

Other Routines

The main board has a Phillips PCF8563 Real-Time Clock (RTC). The clock is backed with a battery so that it retains its setting if the main board is powered down.

unsigned char rtc_get_time(unsigned char *hour, unsigned char *min, unsigned char *sec)	Returns the current time measured by the RTC. The **hour*, **min* and **sec* pointers point to the variables that must receive the values of hour, minutes and seconds. Returns the value 1 if the read values are correct. If the function returns 0 then the chip supply voltage has dropped below an acceptable value and the time values are incorrect.
void rtc_set_time(unsigned char hour, unsigned char min, unsigned char sec)	Sets the current time of the RTC.
void rtc_get_date(unsigned char *date, unsigned char *month, unsigned *year)	Returns the current date measured by the RTC. The **date*, **month* and **year* pointers must point to the variables that must receive the values of day, month and year.
void rtc_set_date(unsigned char date, unsigned char month, unsigned year)	Sets the current date of the RTC.

Globals and #defines

There the supplied main program contains a number of global variables and defines that are relevant.

rx_counter0	The number of characters in the transceiver's receive buffer. It is incremented every time the transceiver receives a character, and is decremented by *getchar*, *scanf*, etc. One can clear the buffer by setting rx_counter0 = 0.
int myaddr,myx,myy;	The node's address/id, x- and y-coordinates.

Packet Format

The WSN nodes communicate with each other using packets of ASCII characters. Packets have a two-byte preamble, the two ASCII characters "AC". A carriage return '\r', newline '\n', or NULL '\0' character is the end of packet marker. A packet can be up to 32 bytes long. The 32 includes the preamble.

The character following the preamble determines the contents of the rest of the packet.

Control Character	Function
Q	The **Q**uery request. If we receive this, call putConfig to transmit our time-stamped coordinates and sensors' values.
C	The **C**onfiguration request. If we receive this,

	another node sent us our new configuration: coordinates, destination address, and time. Retrieve and load this information by calling getConfig. This is used mostly for programming nodes during deployment.
R	The **R**oute request. If we receive this, another node is requesting we send the packet payload to another node. Respond by routing the packet to this node. We can request another node to send/forward data for us to a remote node.
A	This is an **ACK**nowledge packet. We send this in response, to for example a query request.

Following the preamble and control characters are the payload and ASCII formatted characters.

Field	# Characters	Meaning
XS	2	*x*-coordinate of the source/origin of the packet, formatted as "00"...."99"
YS	2	*y*-coordinate of the source/origin of the packet, formatted as "00"...."99"
ID	2	Node ID, typically associated with XS and YS, formatted as "00"...."99"
XF	2	*x*-coordinate of the final destination of the packet, formatted as "00"...."99"
YF	2	*y*-coordinate of the final destination of the packet, formatted as "00"...."99"
HC	2	hop count, formatted as "00"...."99"
LL	2	Light level in % of maximum lx, formatted as "00"...."99"
SW	2	Switch status, formatted as "00" or "01"
DATE	6	Date stamp when the packet was generated (at its origin) formatted as "hhmmss"
TIME	6	Time stamp when the packet was generated (at its origin), formatted as "hhmmss"
The 32nd character is '\0'		

Fields that are unknown, don't care, or undefined for a particular command, are indicated with any negative value, for example "-1". Where it makes sense, some packets may be shorter than 32 bytes. Short packets are terminated with '\r', or '\n'.

Example 1. We receive the packet

ACQ\n

Where '\n' is the ASCII newline character (ASCII 10 in decimal). This is a query request from an unknown node. Assume we are located at (*x*,*y*) = (2,3), our node address/id =12, the light level is 46%, the switch is open, but we don't know what the date or time is. Respond by generating an acknowledge packet:

`Byte #`	`012 34 56 78 90 12 34 56 78 901234 567890 1`
`Field`	`    XS YS ID XF YF HC SW LL YYDDMM HHMISS EOP`
`Packet`	`ACA □2 □3 12 -1 -1 00 00 46 -1-1-1 -1-1-1 \r`

For clarity the fields are separated, and the "□" is used to indicate a space (ASCII 32) in the packet. The actual packet will be:

```
ACA 4 312-1-1000046-1-1-1-1-1-1\0
```

This will also work:

```
ACA040312-1-1000046-1-1-1-1-1-1\0
```

Example 1. We receive the packet

ACQ1222\n

This is a query request from a node located at (*x*,*y*) = (12,22). Assume we are located at (*x*,*y*) = (4,5), our node address/id = 8, the light level is 13, the switch is open, the date (yymmdd) is 050422 and the time (hhmmss) is 145623. Respond by generating an acknowledge packet:

`Byte #`	`012 34 56 78 90 12 34 56 78 901234 567890 1`
`Field`	`    XS YS ID XF YF HC SW LL YYDDMM HHMISS EOP`
`Packet`	`ACA □4 □5 □8 12 22 00 00 13 050422 145623 \r`

Example 3. We sense a closure of the switch at date (yymmdd) 050416 and time (hhmmss) 160201, and want to send this information to the base node, located at (*x*,*y*) = (18,11). As before, we are located at (*x*,*y*) = (4,5), our node address/id = 8. The light level is 25%. Using our routing algorithm we select an appropriate neighbor node and request it to route the packet for us:

`Byte #`	`012 34 56 78 90 12 34 56 78 901234 567890 1`
`Field`	`    XS YS ID XF YF HC SW LL YYDDMM HHMISS EOP`
`Packet`	`ACR □4 □5 □8 18 11 00 01 25 050416 160201 \r`

Before we transmit the packet, we must select the destination address of the node that will route it for us.

Tasks

Add Code To Display Node ID and Time on the LCD.

This will greatly help with debugging.

Design and Implement a Strategy to Join Network

This involves cycling through possible destination addresses and discovering neighbors. Some neighbors will be closer than others. Some nodes will give better RSSI. Design a strategy to select one or two of the possible candidates. Then implement it.

Design and Implement a Strategy to Periodically Update Network

New nodes may join the network, while others may leave the network. Design and implement a method for periodically updating the network.

Design and Implement a Routing Algorithm

A node may desire, or be asked to send information to a node that it not a neighbor. It knows its own coordinates, the coordinates of its neighbors, and that of the destination. Which neighbor will it ask to relay the information? Design and implement a strategy to answer this question.

Complete Implementation *of getLight*

The photosensor is connected in series with a 50K resistor between ground and 5V. The supplied routine getLightLevel measures the voltage across the photosensor and not the light level. Using the supplied data sheet for the photosensor, this voltage must be converted to reflect the light level in lx, expressed as a percentage compared to what one would measure on a bright sunny day.

Pseudo Code

```
void main(void)
{
…
   // Initialize register, porta, and peripherals. Then print a
   // welcome message on the LCD screen.

   init();
   lcd_clear();
   lcd_putsf("Welcome...");

   // Main loop.

   while(1){

      if (isUpDateTime()) {

         // Time to rediscover our neighbors.

         joinNetwork();

      }

      if (getSwitch() == 0){

         // Make a packet to send to the base station.

         setLed(ON);
         base_x = 18;
         base_y = 19;
            …
         hc = 0;
         lux = getLight();
         sprintf(packet,"ACR%2d%2d…",base_x,base_y,…);

          // Use the routing algorithm to select a node we
          // will ask to forward our packet.

          address = findRoute(…);
          setAddress(address);
          putPacket(packet,0);
          delay_ms(1000);

          // Blink the LED a few times, and then
          // turn it off.

          …
          setLed(OFF);
       }

      if (getPacket(packet,0)){

         if (packet[0] == 'Q')  // Echo our configuration.
            putConfig(packet,0);
         if (packet[0] == ‘R’)   { // Route request

            // Construct packet, select destination
            // and send packet
                …
             putPacket(packet,0);
          }
      }
   }
}
```