MIT 12cp Project Report

Sun SPOT Study: Wireless Sensor Networks with Java

Supervisor: Dr. Uwe Roehm

-Semester 1, 2006-

Student: Yuhsiang Tsai
SID: 305259938
# Table of Contents

1. Introduction .................................................................................................................. 2
   1.1 Project Aim .............................................................................................................. 3

2. Sun SPOT Technology .................................................................................................... 4
   2.1 Hardware .................................................................................................................. 4
       2.1.1 Sun SPOT Processor Board .............................................................................. 4
       2.1.2 General Purpose Sensor Board ....................................................................... 6
       2.1.3 Test board ........................................................................................................ 9
       2.1.4 Wireless Networking Merge Sun SPOT ............................................................. 10
       2.1.5 Power consumption and battery life ................................................................. 13
   2.2 Software .................................................................................................................... 16
       2.2.1 Java Virtual Machine Squawk ......................................................................... 16
       2.2.2 Interrupt latency ............................................................................................... 18
       2.2.3 Security ............................................................................................................ 20
   2.3 Development Environment ...................................................................................... 21
       2.3.1 NetBeans .......................................................................................................... 21
       2.3.2 Sun SPOT SDK ................................................................................................. 21
       2.3.3 Sensor Board Library ....................................................................................... 21

3. Wireless Networking ....................................................................................................... 22
   3.1 Motivation .................................................................................................................. 22
   3.2 Security ..................................................................................................................... 23
   3.3 CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) ........................... 23
   3.4 IEEE 802.11 RTS/CTS ............................................................................................ 23
   3.5 Spread Spectrum ....................................................................................................... 24
   3.6 Direct-Sequence Spread-Spectrum .......................................................................... 25
   3.7 Frequency-Hopping Spread-Spectrum ..................................................................... 25

4. Sun SPOT Programming .................................................................................................. 26
   4.1 Green LED Deployment ......................................................................................... 26
   4.2 Tri-color LED Deployment ....................................................................................... 26
   4.3 Switches SW1 and SW2 Deployment ....................................................................... 27
   4.4 Light Sensor Deployment ......................................................................................... 27
   4.5 Temperature Deployment ......................................................................................... 27
   4.6 2G/6G Accelerometer Deployment ......................................................................... 29
   4.7 Radio connection ....................................................................................................... 29
   4.8 Squawk API ............................................................................................................. 31

5. Example Applications ..................................................................................................... 32
   5.1 SPOT-to-SPOT Application ..................................................................................... 32
   5.2 SPOT Application with Basestation Architecture .................................................. 35

6. Conclusions ..................................................................................................................... 38

7. References ..................................................................................................................... 39

Appendix A – Chips Power-Consumption .......................................................................... 41
Appendix B - Abbreviations ............................................................................................... 42
Appendix C - SPOT Application Code – SPOT to SPOT .................................................. 44
   Application Code for RX SPOT .................................................................................... 44
   Application Code for TX SPOT .................................................................................... 46
Appendix D - Basestation Application code ....................................................................... 50
   Basestation SPOT code ............................................................................................... 50
   Remote SPOT code ....................................................................................................... 53
1. Introduction

Sun Labs from Sun Microsystems has developed Sun SPOT (Small Programmable Object Technology) to explore the potential of wireless sensor networking (WSN). Sun SPOT is a small WSN device which integrates multiple sensors and transducers (sensors combined with actuator mechanisms) with programmable facilities and a radio. By simplifying the development of wireless transducer applications, the Sun SPOT systems introduce the potential use of wireless transducer application and implements into real-world products.

The capability of sensing just one physical phenomenon is not enough for today’s sophisticated applications. A device that provides several wireless sensors and transducers with programmable environment is limited only by imagination. In contrast, a Sun SPOT provides a small, battery-powered platform on which applications can interact with a set of wireless sensors facilities and those applications can be written entirely in Java.

Sun SPOTs run the Squawk virtual machine, which is a small Java(TM) VM almost written in Java that runs without an OS directly on small devices. Squawk implements an isolate mechanism, allowing applications to be treated as objects. Multiple isolates can run concurrently in one VM, and isolates can be migrated between different instances of the VM.

For developers who are experienced with Java program development, it is only a small learning curve to get familiar with Sun SPOT application development. This makes it possible that the Sun SPOT community can grow rapidly, as all Java developers can be easily involved.

For the actual development, Sun SPOT is able to be programmed with an Integrate Development Environment (IDE) such as Netbeans. This reduces the painfulness of debugging wireless transducer applications in high-level language though out the
development. Such an IDE simplifies the deployment of Java applications into Sun SPOT with regard to compiling and deploying.

Sun SPOT simplifies the development of wireless transducer applications because the communication of SPOTs can be achieved via wireless networking using a standard 802.11.4 radio. In addition, a set of libraries are provided to access the wireless sensor facilities.

In summary, Sun SPOT provides a programmable platform with multiple sensors, a wireless radio, and Java support. The support of a Java software development environment makes Sun SPOT a very attractive WSN platform for many potential applications in a variety of communities such as military, government officials, business, hospital and even house intelligent automation.

1.1 Project Aim

The project aim is to explore the basic functionality of Sun SPOT devices and to give an overview of the software, hardware and development environment of the SPOT technology. The focus of this project is on a prototype version of the Sun SPOT technology, called bSPOT; note that Sun plans to replace this version by the eSPOTS in the near future. Two example applications are developed; they demonstrate the use of the facilities of the Sun SPOT devices and how to communicate between SPOTs via radio connection. One application is built in a SPOT-to-SPOT architecture, while the second application demonstrates a basestation architecture.
2. Sun SPOT Technology

This section gives an overview of the Sun SPOT technology with regard to hardware, software and the development environment. The Hardware section explores the physical components of SPOT as well as the wireless networking chip CC2420 and overall energy consumption. The Software section discusses the virtual machine, Squawk, which runs and handles applications in the SPOT and the performance of SPOT interrupt latency in detail. Finally, the last subsection will introduce the actual development environment in SPOT.

2.1 Hardware

The Sun SPOT consists of three boards: a processor board, a sensor board and a test board. Each of them carries out different tasks.

Note: Different versions of Sun SPOT devices might have different hardware such as flash memory storage or battery capacity. In the following, we are discussing the bSPOT version.

2.1.1 Sun SPOT Processor Board

The processor board is the core component of the Sun SPOT devices. It contains a 32-bit ARM9 CPU, 512K memory, 2 Mb flash storage and wireless networking following 802.15.4 the standard with integrated antenna.

- CPU: 180 MHz 32 bit ARM920T core
- Memory: 512K RAM/2M Flash
- Radio: 2.4 GHz IEEE 802.15.4 radio with integrated antenna
2.1.1.1 Radio: 2.4 GHz IEEE 802.15.4 radio with integrated antenna
Sun SPOT devices support two types of protocols for wireless communication in the standard of IEEE 802.15.4. They are radio and radiogram protocols. SPOT radio communication is only capable of operating on one channel at any time, and they can not transmit and receive at the same time.

2.1.1.2 Memory: 256K RAM/2M Flash
The 2M Flash memory is organized in 8 x 8 Kb and 31 x 64 Kb sectors. Two applications take 128 Kb capacities each. 1Mb is available for data storage. The remaining memory is taken by system configuration and execution.

IEEE 64-bit address
Each Sun SPOT has its own IEEE 64-bit network address. They might have more than one address when communicating between a host and target when one Sun SPOT acts as a basestation. The address of a SPOT is assigned by user. A basestation SPOT has two addresses: one is for host and one is for radio. The address for communicating with the host is fixed to 2 according to the Sun SPOT Developer’s Guide [17].

Direct Communication
- **Radio**: The radio protocol is a socket-like peer-to-peer communication protocol that reliable, provides buffered stream-based IO between two devices.
- **Radiogram**: The radiogram protocol is a client-server protocol that provides buffered datagram-based IO between two devices.

Multi-Hop Communication
According to the Sun SPOT Developer’s Guide [17], page 15, the current version of the JDK only provides point-to-point (Single hop) communication between nodes using a choice of two protocols. However, the mesh networking (Multi-Hop) is demonstrated by a sample application provided in the SDK CD. The
application shows three nodes: a receiver, a forwarder and a sender. The forwarder is used for catching packets for the receiver and the sender. If the forwarder is missed in between, the sender and the receiver will not be able to transmit.

**Broadcast:** Broadcast is allowed in wireless networking. However, the broadcast is not reliable due to the fact that datagrams might be lost.

### 2.1.2 General Purpose Sensor Board

The sensor board integrates multiple sensors, monitoring LED and interactive switches into one board. All the facilities of this board are programmable in Java. The facilities of the sensor board are:

- One 2G/6G 3-axis accelerometer
- One temperature sensor
- One light sensor
- Two 8-bit tri-color LEDs
- 6 analog inputs
- Two momentary switches
- 5 general purpose I/O pins and 4 high current output pins

In the following, the characteristics of each sensor will be explained in detail.
2.1.2.1 2G/6G 3-axis accelerometer

The three-axis accelerometers measure acceleration in three dimensions. Accelerometers are very handy for measuring the orientation of an object relative to the earth, because gravity causes all objects to accelerate towards the earth.

The Sun SPOT SDK contains a sample of application, “Ex1-reactomatic”, that demonstrates the usage of the accelerometer by indicating the X, Y and Z readings with different colors on one tri-color LED. The range of value for each axis is 4 to 929 as I experienced manually. The default value without having any acceleration is 461 to 463 for X, Y and Z axis.

2.1.2.2 Temperature sensor

This sensor is capable of detecting the environmental temperature. The temperature value which is read from the sensor is a raw value that represents a temperature number without the standard of Celsius or Fahrenheit. The model of the temperature sensor is ADT7411 with a 10-bit temperature-to-digital converter which is capable of detecting -40 to +125 Celsius. The range of the raw value is 0 to 1023. The following mathematical algorithm can be applied to convert the raw reading into Celsius or Fahrenheit. The resolution of raw value is 0.25 Celsius degrees.

\[
\begin{align*}
\text{Temperature Positive} &= \frac{\text{rawValue}}{4} \\
\text{Temperature Negative} &= \frac{\text{rawValue} - 1024}{4}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Positive</th>
<th>Out of range</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°C</td>
<td>+120°C</td>
<td>-40°C</td>
</tr>
<tr>
<td>0</td>
<td>500</td>
<td>864</td>
</tr>
</tbody>
</table>

Reference As specified on page 19 of the ADT7411 data sheet
2.1.2.3 Light sensor
The light sensor measures the range from darkness to lightness and converts it to a raw value. The range for the raw light value is 0 to 1023. The value represents the intensity of detected light.

Raw Value Range:
Darkness 0 → 1023 Brightness

2.1.2.4 Two 8-bit tri-color LEDs
There are two 8-bit tri-color LEDs located on the sensor board. These LEDs are able to flash with a range of colors. The typical use of these two LEDs is to show the status when an application is in a particular state. Therefore, it might be easier for a developer to see the current progress of an application. The color of the LEDs is shown with red, green and blue. The range of each color for intensity is 0 to 255.

2.1.2.5 Two momentary switches
The two switches can be used to interact with an application. An application can be changed to a sub-function or other program when user-defined application has supported.
2.1.3 Test Board
The test board contains the USB cable interface to communicate with a computer. There is also a 650 mAh lithium-ion battery attached to it. A set of LEDs are located which indicate a particular status of SPOT such as transmitting or receiving with a computer while deploying or running an application.

- Interface: USB 2.0 interface with power monitoring
- Battery: 3.6V rechargeable 650 mAh lithium-ion battery
- Power switch, reset buttons and status LEDs

2.1.3.1 USB 2.0 interface with power monitoring
Sun SPOT devices can be connected to a computer via USB cable. All applications are transmitted into SPOTs via the USB cable. While the USB cable is connected, the battery is recharged.

2.1.3.2 3.6V rechargeable 650 mAh lithium-ion battery
The power supply of Sun SPOTs can be either via battery or via USB cable. One is supplied by 650 mAh lithium-ion battery which is rechargeable. The other just simply plugs USB cable to Sun SPOT.

2.1.3.3 Power switch, reset buttons and status LEDs
The power switch and reset button are located on the test board. The power switch only works when no USB cable is plugged. 4 LEDs indicate the status of the SPOT. They indicate the status of recharging, transmitting (TX), receiving (RX) and power.
2.1.4 Wireless Networking Merge Sun SPOT

Each Sun SPOT has a wireless network communication chip, a Chipcon CC2420, which is a true single-chip 2.4 GHz IEEE 802.15.4 compliant radio (RF) transceiver designed for low-power and low-voltage wireless applications. Sun SPOT provides the programmable platform to control the properties of the radio connection.

2.1.4.1 Chipcon CC2420

CC2420 includes a digital direct sequence spread spectrum (DSSS) baseband modem providing an effective data rate of 250 kbps.

The CC2420 is a low-cost, highly integrated solution for robust wireless communication in the 2.4 GHz unlicensed industrial, scientific and medical (ISM) band. It complies with worldwide regulations covered by ETSI EN 300 328 and EN 300 440 class 2 (Europe), FCC CFR47 Part 15 (US) and ARIB STD-T66 (Japan).

The hardware support of CC2420 includes packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information. These functions reduce the complexity on the host controller.

2.1.4.2 The characteristics of CC2420

- True single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver with baseband modem and MAC support
- 250 kbps effective data rate
- Low current consumption (RX: 18.8 mA, TX: 17.4 mA)
- Low supply voltage (2.1 – 3.6 V) with integrated voltage regulator
- Low supply voltage (1.6 – 2.0 V) with external voltage regulator
- Programmable output power
- No external RF switch / filter needed
- 128(RX) + 128(TX) byte data buffering
• Hardware MAC encryption (AES-128)
• Battery monitor
• Powerful and flexible development tools available

<table>
<thead>
<tr>
<th>OPERATING CONDITIONS:</th>
<th>Min</th>
<th>Typical</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Range</td>
<td>2400</td>
<td>2483.5</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>Data Rate</td>
<td>250</td>
<td></td>
<td>Kbps</td>
<td></td>
</tr>
<tr>
<td>Operating voltage</td>
<td>2.1</td>
<td>3.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Operating temperature</td>
<td>-40</td>
<td>85</td>
<td>°C</td>
<td></td>
</tr>
</tbody>
</table>

CC2420 Performance statistic table reference from [6]

2.1.4.3 Channels
The 2.4GHz ISM band supports 16 channels. The available channel range is from 11 to 26. Channel 26 is usually the most reliable because interference from 802.11 devices, mobile phones and 2.4GHz phones would affect if they are in the same channel. As channel 26 is rarely used, signals in channel 26 would not be easily interfered by other signals from 802.11 devices. The actual channel can be configured on the application-level used setChannel().

2.1.4.4 PAN
The Sun SPOT environment supports broadcasting. Broadcasting is not reliable because datagrams might be lost during transmitting. NoReplyException will not been thrown if the remote SPOT did not receive a datagram. Thus, the delivery of datagram to receiving nodes is not guaranteed. All devices which communicate via broadcasting must have same personal area network (PAN). For more information about PAN, please refer to Appendix B.
2.1.4.5 Distance Range

In a board area, the maximum of distance is in the measurement of my 100 footsteps which is approximately 50 centimeter for each footprint. Thus we can see the furthest distance is about 50 meters according to CC2420 data sheet but that can vary greatly depending on temperature, humidity and air quality. In addition, the power of the radio is adjustable by setting the properties of the radio connection in applications. (Note: The current SDK does not seem to support this, as no reply was posted to a corresponding question on SunSPOTWorld forum. However, the SDK with eSPOT should support this.)

The radio connection does not act similar to infrared. Infrared is easily influenced from interference such as any obstacles in between and the range is only limited in 10 meters.

The spreading spectrum which is deployed for wireless networking has resistance from interference and is noise-like signal by doubling the original signal. Even with a thick obstacle in between or SPOTs in closed place such as in a draw or box, SPOTs still can communicate. However, the range of SPOT would significantly reduce.

The distance that a signal can be transmitted depends on several factors. The primary hardware factors that are involved are:

- Transmitter power
- Cable losses between the transmitter and its antenna
- Antenna gain of the transmitter
- Localization of the two antennas

This refers to how far apart the antennas are and if there are obstacles between them. Antennas that can see each other without any obstacles between them are in line of sight.
• Receiving antenna gain
• Cable losses between the receiver and its antenna
• Receiver sensitivity

2.1.4.6 Port
Legal port numbers for SPOTs are supposed in the range of 1 to 254. However, the current legal range is only from 1 to 127 due to an existing bug.

2.1.5 Power consumption and battery life
The batteries built into the devices are 650 mAh LiIon batteries. As with these types of devices in general, battery life depends on how much the SPOT is used.

The VM is in control of the processor and able to put the system to sleep whenever it is appropriate. This saves developers quite a bit of complicated code to do so.

Sun SPOT are powered either via AA battery or USB cable. While a USB cable is plugged to SPOT, the battery is recharged. Sun SPOT devices provide two sleep modes: shallow sleep and deep sleep mode.

Shallow sleep is in the situation of no threads ready to run but SPOT active whereas deep sleep mode is in case of no threads ready to run and SPOT is not active.
The following table shows the energy consumption of the different parts of the bSPOT devices. This information has been compiled from the reference [21].

<table>
<thead>
<tr>
<th>Situation</th>
<th>CPU</th>
<th>Radio</th>
<th>Sensor Board</th>
<th>Thread</th>
<th>Power-Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating with Radio</td>
<td>Busy</td>
<td>on</td>
<td>No</td>
<td>Running</td>
<td>105.5mA</td>
</tr>
<tr>
<td>Operating</td>
<td>Busy</td>
<td>off</td>
<td>No</td>
<td>Running</td>
<td>80mA</td>
</tr>
<tr>
<td>Shallow sleep</td>
<td>Idle</td>
<td>off</td>
<td>No</td>
<td>Ready</td>
<td>24mA</td>
</tr>
<tr>
<td>Shallow sleep</td>
<td>Idle</td>
<td>on</td>
<td>No</td>
<td>Ready</td>
<td>44mA</td>
</tr>
<tr>
<td>Deep sleep</td>
<td>Idle</td>
<td>off</td>
<td>No</td>
<td>No</td>
<td>33uA</td>
</tr>
</tbody>
</table>

**2.1.5.1 Power Consumption statistics [21]**

The power-consumption of radio for receive mode is 18.8 mA and transmit mode is 17.4 mA as specified in CC2420 features. And the sensor board consumes roughly between 5.5 and 6.5 mA without any facilities active. (Reference from the table in Power usage from Sun SPOT SDK CD)

Deep sleep mode can last the power of SPOT for over a year. Furthermore, the LEDs in sensor board are heavy power-consuming. Interestingly, transmitting is drawn slightly lower than receiving.

The replacement of battery is only done by changing the battery board with new AA battery. Alternatively, external power supply can be built up according to http://ipod.hackaday.com/entry/1234000520028239/.
Summary of power consumption:
• Radio RX 23mA, TX 22mA when CPU busy
• Radio RX 18.8mA, TX 17.4mA when CPU Idle
• CPU busy 80mA
• Idle sensor board 5.5 to 6.5mA
• Shallow sleep 24mA
• Deep sleep 48uA

The built-in AA battery of a Sun SPOT device has 650 mAh built. As we can see the figure above when the SPOT is running an application which the radio facility is on, it consumes roughly 100 mA. The battery only supplies 650 mA per hour. Thus, the battery life lasts theoretically 6.5 hours, if the device is in constant use. However, in actual testing, the SPOT lasts for about 8 hours when the SPOT keeps in busy and radio is on.

In addition, more consumption would occur when deploying more sensors facility. Also the 8-bit tricolor LEDs are energy hungry.

The power consumption of other chips such as temperature sensor and accelerometer sensor are shown in appendix A.

(Note: according to the discussion of Sun SPOT forum, current bSPOT does not support deep sleep mode. The eSPOT may release with the deep sleep mode. Please refer to Reference [15].)
2.2 Software

2.2.1 Java Virtual Machine Squawk

Squawk is a Java virtual machine that is Connected Limited Device Configuration (CLDC), Java Platform, Micro Edition (J2ME) compliant for small devices. Explanation for CLDC and J2ME abbreviation is in appendix B. Squawk is specially designed to run on small, memory constrained devices with no operating system footprint required.

Squawk VM runs on conventional networks of desktops and servers as well as on wireless networks of small transducer, and so enables a unification of those two domains.

Characteristics of Squawk Virtual Machine

- Fully capable J2ME CLDC 1.1 Java VM with OS functionality
- VM executes directly out of flash memory
- Device drivers written in Java
- Automatic battery management

Squawk is implemented in entirely Java key components automatically translated into C and compiled for the target device. Certain space savings of Squawk enhances automatically because bytecodes are a more efficient representation than the equivalent functionality in native code.

Squawk achieves further space savings by verifying class files in advance so that a class verifier is not required inside the VM. The pre-verified classes get bundled into what are called suites for deployment. Furthermore, each suite is also an especially memory efficient representation of the classes it contains. On average, suite files are 35% the size of class files.
The following table indicates files size in Suite is significantly smaller.

<table>
<thead>
<tr>
<th>Application</th>
<th>JAR</th>
<th>Suite</th>
<th>Suite/JAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLDC</td>
<td>458,291</td>
<td>149,542</td>
<td>0.33</td>
</tr>
<tr>
<td>cubes</td>
<td>38,904</td>
<td>16,687</td>
<td>0.42</td>
</tr>
<tr>
<td>hanoi</td>
<td>1,805</td>
<td>835</td>
<td>0.46</td>
</tr>
<tr>
<td>delta blue</td>
<td>30,623</td>
<td>8,144</td>
<td>0.27</td>
</tr>
<tr>
<td>mpeg</td>
<td>100,917</td>
<td>54,888</td>
<td>0.54</td>
</tr>
<tr>
<td>manyballs</td>
<td>12,017</td>
<td>6,100</td>
<td>0.51</td>
</tr>
<tr>
<td>pong</td>
<td>17,993</td>
<td>7,567</td>
<td>0.42</td>
</tr>
<tr>
<td>spaceinvaders</td>
<td>50,854</td>
<td>25,953</td>
<td>0.51</td>
</tr>
<tr>
<td>tilepuzzle</td>
<td>18.516</td>
<td>7,438</td>
<td>0.40</td>
</tr>
<tr>
<td>wormgame</td>
<td>23,985</td>
<td>9,131</td>
<td>0.38</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>753,905</strong></td>
<td><strong>286,285</strong></td>
<td><strong>0.38</strong></td>
</tr>
</tbody>
</table>

Table with file sizes of different example programs [1], page 16.

Squawk VM is able to run multiple applications simultaneously whereas most conventional Java VMs can only run one application a time. Multiple applications are managed such that they do not share mutable state and each application is treated as an object.

Applications can be directly controlled through methods such as start(), pause() and exit(). Also, synchronization on shared immutable resources is track on a per isolate basis so that threads within one isolate are managed as a group without affecting threads in another.

The compounded memory savings deliver a truly small footprint Java VM: The current version of Squawk running on the Sun(TM) SPOT uses 80KB of RAM and 270KB of flash memory, including all the libraries for CLDC 1.1, radio, and sensor / actuator access.
2.2.2 Interrupt latency

Interrupt latency is the delay through out an interrupt creation. Interrupt latency occurs when an interrupt is requested from software or hardware.

These figures describe the relative performance of various interrupt architectures.

<table>
<thead>
<tr>
<th></th>
<th>Handled locally (current release)</th>
<th>Handled locally (with Kernel handling enabled, but not used)</th>
<th>Handled in kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>single threaded app</td>
<td>0.33</td>
<td>0.38</td>
<td>5.14</td>
</tr>
<tr>
<td>second thread that creates no garbage</td>
<td>0.52</td>
<td>0.38</td>
<td>5.14</td>
</tr>
<tr>
<td>second thread that creates garbage</td>
<td>47.08</td>
<td>66.16</td>
<td>5.14</td>
</tr>
</tbody>
</table>

All figures shown in milliseconds.

The table shows the interrupt latency in different modes [13].

The Interrupt latency occurs when handles multiple threads. There are two ways of handling threads in SPOT. They are handled locally or handled in kernel and will result differently in interrupt latency.

2.2.2.1 Handled locally

The current release handles interrupts in the same virtual machine as the application/driver. The driver sets up the interrupt controller sets up the interrupt controller. The interrupt controller is blocked and waits for the signal from Squawk VM to trigger. When an interrupt occurs, the interrupt handler sets a bit in a global data structure to record the interrupt, and disables the interrupt to avoid repeats.
When Squawk VM reschedule, the bit is detected, the event triggers the interrupt handler to resume. The interrupt handler handles the interrupt and then enables the interrupt again.

The latency is optimized if Squawk is idle. However, if it is executing bytecodes in another thread the penalty is quite small as the VM reschedules after a certain number of back branches. A delay takes place according to how close to the next reschedule the interrupt occurs. The delay is shown on handled locally column between first thread and second thread.

When garbage collection takes place, the reschedule is carried out after the garbage collection is finished. Therefore, we can see the significant delay in Handle locally above.

2.2.2.2 Handled in kernel
This architecture sets up an internal VM, kernel, within Squawk with its own state and heap. The interrupt handler runs in the kernel VM, and communicates asynchronously to the rest of the driver in the kernel VM.

The delay in handled locally will be prevented from handled in kernel because there is no need to wait for garbage collection finished before unblocking interrupt handler in the kernel VM. The advantage of this handle is shown on the figure above which the latency is more independent. However, the context switch between VMs would take time when the interrupt occurs. Therefore, the figure above shows these timing are identical.
2.2.3 Security
Security for wireless transducer application is particularly important and difficult. Sun Microsystems has recently reported highly optimized implementations of RSA that perform much better than any reported previously. Sun Labs has also developed implementations of Elliptic Curve Cryptography (ECC), a resource efficient alternative to RSA, that provide an additional order of magnitude performance improvement on 8-bit CPUs. Explanation of Elliptic Curve Cryptography is provided in appendix B abbreviations.
2.3 Development Environment
The following gives an overview of the development environment for Sun SPOT devices with regard to Netbeans integrated development environment (IDE) support, Sun SPOT SDK and sensor board library.

2.3.1 NetBeans
NetBeans is an integrated development environment for Java development. SPOT applications are available to be developed in NetBeans. NetBeans is able to provide the high-level debugging for SPOT developers. However, the low-level debugging is not supported by NetBeans.

In Squawk VM, the low-level and high-level debuggers are provided for SPOT developer. However, they are still under development. Thus, the uses of these debuggers are painful for developers. In addition, the Sun SPOT Developer’s Guide [17] suggests debugging the program by using println statements.

2.3.2 Sun SPOT SDK
Sun SPOT SDK contains the libraries for the basic hardware of SPOT devices as well as sample applications. The Squawk is also included in Sun SPOT SDK.

2.3.3 Sensor Board Library
The sensor board library contains the use of sensors, LEDs and radio in the sensor board. The library does not include in SPOT SDK. To import the library, it needs to be installed explicitly. Please refer to the Sun SPOT Developer’s Guide [17] or the Quickstart manual.
3. Wireless Networking

3.1 Motivation

The client/server architecture applied for most of network infrastructures when the Internet has come out. However, engineers realize that the architecture gradually decreased the performance due to too much workload on server side. Therefore, the Intranet has being developed to solve such performance issue.

Organization begins to establish a scale of local area network. In order to avoid influencing the operation of an organization, the LAN always should be finished once. Unfortunately, the LAN establishment needs to be precise and deliberate in all aspects such as distance between offices and future network extension. In fact, there are always some points left out when building up a massive LAN.

A huge LAN consists of a large number of cables just like spider web, complicated and out of order. In reality, the intranet problem is commonly caused by network cables. Therefore, the repair is time-consuming and not efficient due to complex cable mapping through floors and ceilings. Such as an exhibition, the network connection is required to be as flexible as possible. The wire cable network is hard to overcome such case.

In 1997, IEEE 802 has accomplished the network standard, IEEE802.11 which represents the first standard for wireless local area network (WLAN) products from an internationally recognized, independent organization.

There are two types of networking structure for WLAN. An ad-hoc network is a simple network where communications are established between multiple stations in a given coverage area without the use of an access point or server.

The client/server network uses an access point (AP) that controls the allocation of transmit time for all stations. APs own the bridge functionality and also include extra
functionalities depend on the varied brand. The client/server network allows extending the coverage of WLAN significantly by interconnecting APs.

3.2 Security
The security issue is always hard to overcome in computing from all perspectives especially in wireless networking. The access control in many wire networks are limited at physical level which trust all the users on the local network whereas in WLAN, anyone who is in the coverage of wireless network can connect access points and access to WLAN in advance.

The common solution is to encrypt packets before transmitting. The modern access points possess the facility of decryption for packets. However, the solution has been proven that is cracked easily. If all access points are available for public, anyone can have access to online anywhere within coverage.

3.3 CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance)
In wired networks, the networks can is able to detect when a collision occurs. Unlikely in wireless networks, the sending node can not detect a collision, so it attempts to avoid collision by sending a jam signal to all other stations. After sufficient time, all stations have received the jam signal. Then the sending node starts transmitting. While other stations attempt to transmit and they receive a jam signal, the transmitting will stop for a random time and retry again.

3.4 IEEE 802.11 RTS/CTS
RTS/CTS is an additional method to implement virtual carrier sensing in Carrier sense multiple access with collision avoidance according to Wikipedia. The issue of implementing CSMA/CA in 802.11 cause the hidden node problem.

RTS/CTS solves the hidden node problem and also exposed node problem. When a node requests to transmit data to recipient, the transitter send a request-to-send (RTS) packet. After the recipient revceived the packet, it sends clear-to-send(CTS) packet back to
transmitter. Then the sending node starts transmitting data. In case of sending node did not receive the CTS packet, it will send RTS packet over again. Any other node receive the CTS packet will stop sending data in a given time, which is included in both CTS and RTS packet. Any other node receiving the RTS frame but not the CTS frame is permitted to transmit to other neighboring nodes.

The RTS/CTS is designed in the condition that all node have same distance between each other. Therefore, if two nodes are concurrently sending to the same recipient in the situation of one node is closer, the recipient only receives the packet from closer node because the packet of closer node exceeds always.

3.5 Spread Spectrum
A method of transmitting a signal by spreading it over a broad range of frequencies using a compatible receiver to reassemble the signal. This provides reduced interference and can increase the number of simultaneous users within a radio frequency band. There are two types of spread spectrum widely used in radio connection; they are direct-sequence spread spectrum and frequency-hopping spread spectrum.
3.6 Direct-Sequence Spread-Spectrum

One of two types of spread-spectrum radio technology used in wireless LAN (WLAN) transmissions. To increase a data signal’s resistance to interference, the signal at the sending station is combined with a higher-rate bit sequence that spreads the user data in frequency by a factor equal to the spreading ratio.

3.7 Frequency-Hopping Spread-Spectrum

The Frequency-hopping spread-spectrum technique modulates the data signal with a narrowband carrier signal that “hops” in a predictable sequence from frequency to frequency as a function of time over a wide band of frequencies. Interference is reduced, because a narrowband interferer affects the spread-spectrum signal only if both are transmitting at the same frequency at the same time. The transmission frequencies are determined by a spreading (hopping) code. The receiver must be set to the same hopping code and must listen to the incoming signal at the proper time and frequency to receive the signal.
4. Sun SPOT Programming
The API document of Sun SPOT SDK is located at .squawk\software\doc\javadoc\spots and .squawk\software\doc\javadoc\squawk. However, the classes DemoSensorBoard and SensorBoardColouredLED, which are provided in the sensor board library sensor-board-kit.jar, are not contained in the API document. Furthermore, the Sun Spot Developer’s Guide [17] introduces the use of the sensor library.

The location of the sensing facilities on the sensor board, please refer to section 2.1.2
General Purpose Sensor Board. The following shows the programming of common facilities in SPOT in Java syntax.

4.1 Green LED Deployment
import com.sun.squawk.peripheral.ILed;
import com.sun.spot.sensorboard.DemoSensorBoard;

ILed greenLED;
greenLED = DemoSensorBoard.getGreenLed();

greenLED.setOn(); //TURN OFF LED
greenLED.setOff(); //TURN ON LED
greenLED.setOn(true); //TURN OFF LED
greenLED.setOn(false); //TURN ON LED

4.2 Tri-color LED Deployment
import com.sun.spot.sensorboard.SensorBoardColouredLED;

SensorBoardColouredLED lightLed
lightLed = SensorBoardColouredLED.getLed1(); //getLed2() is the other Led.
lightLed.setOn(); //switch it on
lightLed.setRGB(0, 0, 0); // (Red,Green,Blue) in the range of 0 to 255.
4.3 Switches SW1 and SW2 Deployment

```java
import com.sun.squawk.peripheral.ISwitch;
import com.sun.squawk.peripheral.testboard.TestBoard;

ISwitch sw1 = TestBoard.getInstance().getSwitchP1();
sw1.waitForChange(); // This will block until switch1 is pressed. Need to throw a
InterruptException

if(sw1.isClose()) // Detect the sw1 status.
    System.out.println("SW1 is now pressed.");
else if(sw1.isOpen())
    System.out.println("SW1 is now released.");
```

4.4 Light Sensor Deployment

```java
import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.squawk.peripheral.io.RangeInput;
// The getRange() return 1024 for all instances of RangeInput class

RangeInput lightSensor;
lightSensor = DemoSensorBoard.getLightSensor();

System.out.println("Light Range: "+ lightSensor.getRange());
System.out.println("Light Value: "+ lightSensor.getValue());
```

4.5 Temperature Deployment

```java
import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.squawk.peripheral.io.RangeInput;

RangeInput tempeSensor;
tempeSensor = DemoSensorBoard.getTemperatureSensor();
```
System.out.println("Temp Range: " + tempeSensor.getRange());
System.out.println("Temp Value: " + tempeSensor.getValue());

4.5.1 Algorithm Converting Raw Value to Celsius

//Reference from Sample, FridgificationApplication, from Sun SPOT SDK CD
private int convertToCelsius(int value) {
    int convertedValue;
    if (((value >> 9) & 0x000001) > 0) {
        convertedValue = (((value & 0xFF) - 512) / 4);
    } else {
        convertedValue = value / 4;
    }
    return convertedValue;
}

4.5.2 Simplified Algorithm

private int convertToCelsius(int value) {
    int convertedValue;
    if (value > 512 && value < 1024) {
        convertedValue = (value - 1024) / 4;
    } else {
        convertedValue = value / 4;
    }
    return convertedValue;
}
4.6 2G/6G Accelerometer Deployment

```java
import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.squawk.peripheral.accelerometer.Accelerometer3D;
import com.sun.squawk.peripheral.accelerometer.LIS3L02AQAccelerometer;

((LIS3L02AQAccelerometer)acc).set6GScale();
// special scale call for particular accelerometer
// or set to 2G by set2GScale()

RangeInput x = acc.getX();
RangeInput y = acc.getY();
RangeInput z = acc.getZ();
```

4.7 Radio Connection

The default IEEE address of SPOTs are -1 which are not acceptable for radio connection. Before running radio connection applications, the addresses of SPOTs need to be set as well as the port number consistency. In the following example, the address of the server is 129 whereas the client’s is 128. Both are using the port number 100. To configure the address of a SPOT, execute an ANT command, `-Ddebugclient.port=XXX`.

4.7.1 For server

// 129 is address of other SPOT
StreamConnection conn = (StreamConnection) Connector.open("radio://129:100");

DataOutputStream dos = conn.openDataOutputStream();
try {
    dos.writeUTF("Hello");
} catch(NoAckException e) {
    System.out.println("No sending to 129");
}
} finally {
    conn.close();
    dos.close();
}

4.7.2 For client

// 128 is address of other SPOT
StreamConnection conn = (StreamConnection) Connector.open("radio:///128:100");

DataInputStream dis = conn.openDataInputStream();
DataOutputStream dos = conn.openDataOutputStream();
String question = "";
try {
    question = dis.readUTF();
    System.out.println(question);
} catch (NoAckException e) {
    System.out.println("No received from 128");
}
finally {
    conn.close();
    dis.close();
    dos.close();
}
4.8 Squawk API

The Squawk API provides access of different low-level facilities of the Java VM. The following may help to understand the property of Sun SPOT device. For more information about the CLDC API, please refer to the section 7. Reference [20].

**RunTime.FreeMemory()**

Returns the available memory of 256K bytes RAM. Calling `RunTime.gc()` may increase the available memory (see below).

**RunTime.TotalMemory()**

Returns 2M bytes Flash Memory of Sun SPOT devices. This may vary depending on the hardware of a host.

**RunTime.gc()**

Runs the Java garbage collector. Calling this method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. Java Virtual Machine will automatically run garbage collection whenever it thinks appropriate.

**System.currentTimeMillis()**

The current system time, measured in milliseconds. The type of value is `long`. The initial value is 0 when a SPOT application starts. Applications initiate when `ant run` or `ant host-run` command executed or when the power of Sun SPOTs is turned to on. The `currentTimeMillis` counter will keep running until the application is reset or terminated.
5. Example Applications
Two applications have been built to demonstrate the usage of Sun SPOT devices as well as the communication between them. The first application applies the SPOT-to-SPOT architecture whereas the second one is built in a basestation architecture.

5.1 SPOT-to-SPOT Application

Hardware Requirement
- Two Sun bSPOTs
- USB cable
- A Java-capable computer

Description
The application is intended to establish a connection between two SPOTs. One SPOT sends data constantly to the other SPOT by radio connection. The sending SPOT firstly detects temperature value and light value and shows them by the red and blue light intensity of LED1. Then the sending SPOT includes light and temperature value into a datagram and sends it to the receiving SPOT. The receiving SPOT receives data and then process the data to distinguish temperature or light message. After the process, the receiving SPOT turns on LED1 and shows the same status of the sending SPOT.

When the connection is not available, the green led will flash to show the status. And if the connection is established, the green led will be off.

The application also examines the distance of the SPOT connection and what might interfere the range. According to cc2420 data sheet, the furthest range is 50 meters in a clear path. However, interference will affect the radio range. The interference can be any 802.11 devices and any 2.4 GHz wireless connection phone. To optimize the performance, the channel can be set to 26 which is rarely used in other 802.11 devices.
Experience Range
The following table shows the radio range in different situation.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Clear path outdoor</td>
<td>50 meters</td>
</tr>
<tr>
<td>b. The receiving SPOT is in a metal box with a hole on the top. Sender and receiver are placed at same height indoor.</td>
<td>5 meters</td>
</tr>
<tr>
<td>c. The receiving SPOT is packed into a static shield bag in a metal box with a hole on the top. Sender and receiver are placed at same height indoor.</td>
<td>1 meter</td>
</tr>
<tr>
<td>d. Same situation as c, but a person stands in between.</td>
<td>No available</td>
</tr>
<tr>
<td>e. Two SPOTs are placed in separate rooms with both doors closed. The two rooms are neighbored with brick walls. The distance in between is about 4 meters.</td>
<td>Available</td>
</tr>
</tbody>
</table>

The direction of SPOTs might influence the connection range. The positive radio direction is where the switches are facing. When two SPOTs are facing each other with positive direction, it will optimize the connection range. Also, the height of SPOTs might result different range distance. Especially the sending node is higher then receiving node about half meter. This SOMETIMES extends the current range. From my experience, it often has different result. Lightness indoor will not influence the range much different.
**Datagram Transmit**

The exception, `NoAckException`, is only thrown when a datagram from sending node does not reach the receiving node. Thus, the success on delivery of datagram is detectable for sending node. However, the receiving node will not throw `NoAckException` if a datagram is lost. The receiving node will block when calling `read()` method and wait for datagram from sending node. There is no way we can know the data is guaranteey sent to receiving node. Therefore, application-level acknowledgement is implemented to ensure the delivery of datagram when necessary.

**Application Code**

The application code is attached in appendix C. The RX is the program for receiving node and TX is for sending node. Each of programs needs to compile and deploys into SPOT. (Refer to the SPOT Developer Guide [17] or Quickstart manual)
5.2 SPOT Application with Basestation Architecture

A basestation architecture consists of a computer (host), a basestation SPOT and a remote SPOT. A basestation refers as a basestation SPOT communicate to a computer (host) via USB cable as well as communicating remote SPOT by wireless networks.

**Hardware Requirement**

- Two Sun bSPOTs
- USB cable
- A Java-capable computer

**Description**

The application uses the basestation architecture which a SPOT connects with a host via USB acted as a basestation. The basestation communicate with a host as well as the remote SPOT.

The remote SPOT keeps detecting current temperature and light value. After obtaining the values, it then tries to send it to the basestation SPOT. If the basestation is not in the given coverage, the remote SPOT will accumulate data into buffer and send the buffer to basestation as soon as the basestation is in the available range again.

The green LED of remote SPOT flickers on and off which means that the connection is not available. Once the connection is established, then the green LED will be off.

The basestation is only used for receiving data from remote SPOT and sends the data to the host. A graphical user interface (GUI) application runs in a host and communicates
with the basestation. The GUI contains a text area to display data and a button to clear the text area. Whenever the basestation receives data, the data will be sent to the host via USB cable and pop up the data on the GUI. In addition, the application code in basestation can not contain any use of the facilities such as LED, light or switches, otherwise the exception will occur. Thus, I presume the basestation is only used for receiving data.

The blue light of sending SPOT means the SPOT is now obtaining data and sending datagram to basestation. To activate the remote SPOT, press switch one. To stop the process, press switch two until blue light is off. To clear the buffer, press switches one and two together when the remote SPOT is not sending and obtaining data.

Development
Before a SPOT can act as a basestation, the SPOT needs to be deployed and run the sample application, Basestation, to initialize the basestation utility. Next, compile and run the application by commands, ant host-compile and ant host-run referred to Quickstart manual. The remote SPOT would just do like normal compiling, deploying and running.

Addresses
The Host and the remote SPOT each have their own IEEE 64-bits address. The basestation has two addresses. One is for serial line and one is for its radio. The serial line address is used for communicating with the host. The Host address is fixed at 1 and the serial line address in basestation is fixed at 2. The other address for basestation is adjustable used to communicate with remote SPOT. To adjust address, by ant flashconfig -Dserial.number=XXX.

The address of SPOT can be seen when deploying an application or by the statement, Spot.getInstance().getProperty(“IEEE_ADDRESS”). The example of seeing address in deploying looks like following where 128 is the address of the SPOT.
-do-run:
  [java] Devel Library
  [java] ==============================================================
  [java] Native lib Version = RXTX-2.1-7pre17
  [java] Java lib Version  = RXTX-2.1-7pre17
  [java] Waiting for target to synchronise...
  [java] (please reset Spot if you don't get a prompt)


Reserved port
According to the Sun Spot Developer's Guide [17] page 11, the host uses a radiogram connection on port 125 to configure the basestation. For this reason, applications should not use this port.

Application Code
The application code is attached in appendix D. The Basestation SPOT is the program for receiving node connected with a host. The remote SPOT is used for sending data to basestation SPOT. Each of programs needs to be compiled and deployed into SPOT. (Refer to the SPOT Developer’s Guide or Quickstart manual)
6. Conclusions

Sun SPOT devices integrate multiple sensors into one transducer. Sun SPOT devices are much more flexible than other devices. Furthermore, all the facilities are programmable in the high-level language Java. Sun lab research aims to make Sun SPOT devices to be easily programmed by any Java developers. **Hence, any software developers familiar with Java can almost straight-ahead start working with Sun SPOTs.** With this great participation in development, Sun SPOT technology will grow rapidly in stability and maturation.

However, one major drawbacks of Sun SPOT devices is that they need to be recharged regularly. As a network can consist of a large number of nodes, this becomes a huge workload in maintenance.

The Sun SPOT devices can be used in any communities such as hospital, business, military, government and home automation. I believe Sun SPOT technology will have significant impact in home automation and hospital.

As of today, home automation only operates with single interaction such as dimming lights and automated door. Sun SPOT is able to perform more complicated tasks. One example at Volkswagen where researchers created a prototype of a sensor network in which a vehicle performed a safety check of home before the owner drove off. There is a lot more waiting for researcher to discover because the flexibility of Sun SPOT technology.

Although there are many benefits associated with Sun SPOT device whereas some drawbacks exist too. However, it should be noted that Sun SPOT is still under development. The evolution of Sun SPOT technology will end up with better performance.
7. References
[1] Cristina Cifuentes. JAVATM ON WIRELESS SENSORS. Sun Labs. Feb 2006
Wireless Devices with Squawk and SpotWorld.
URL: http://www.ics.uci.edu/~lopes/bpc05/papers/smith.pdf
Last accessed: 11/05/2006
projects/old_projects/wlan/simulations/Simulations/simulations.html
Last accessed: 05/05/2006
/reference/wireless1.html . Last accessed: 08/05/2006
[5] Daniels J. Legal port numbers for Spots. SunSPOT World forum discussion. URL:
[6] 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Chipcon Products. Chip Data
sheet. URL: http://www.chipcon.com/files/CC2420_Data_Sheet_1_4.pdf
Last accessed: 09/05/2006
technologies_tech_note09186a00800e90fe.shtml . Last accessed: 11/05/2006
Direct-sequence_spread_spectrum . Last accessed: 11/05/2006
wiki/Frequency-hopping_spread_spectrum . Last accessed: 15/05/2006
[10] Carrier Sense Multiple Access With Collision Avoidance. Wikipedia article. URL:
http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance
Last accessed: 17/05/2006
URL: http://www.sunspotworld.com/forums/viewtopic.php?t=29
Last accessed: 21/05/2006


[18] *Sun SPOT Development Kit*. Sun Microsystems. 29/09/2005


Chip Data sheet.

## Appendix A – Chips Power-Consumption

### CC2420: Radio Wireless Sensor

<table>
<thead>
<tr>
<th>Power Mode</th>
<th>Typ. Current</th>
<th>Wake-up Consumption from OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>0.2 µA</td>
<td></td>
</tr>
<tr>
<td>(Voltageregulator off)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerDown</td>
<td>20 µA</td>
<td>0.3ms</td>
</tr>
<tr>
<td>(Voltageregulator on)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Idle</td>
<td>426 µA</td>
<td>0.9ms</td>
</tr>
<tr>
<td>(Crystaloscillator on)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReceiveMode</td>
<td>19.7 mA</td>
<td>1.1ms</td>
</tr>
<tr>
<td>TransmitMode @ 0dBm</td>
<td>17.4 mA</td>
<td>1.1ms</td>
</tr>
<tr>
<td>TransmitMode @ -5dBm</td>
<td>14mA</td>
<td></td>
</tr>
<tr>
<td>TransmitMode @ -10dBm</td>
<td>11mA</td>
<td></td>
</tr>
</tbody>
</table>

Source: data sheet in [6].

### ADT7411: Temperature Sensor

<table>
<thead>
<tr>
<th>Power requirements</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Dissipation</td>
<td>10</td>
<td>mW</td>
<td></td>
<td>VDD=3.3 V. Using normal mode.</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>µW</td>
<td></td>
<td>VDD=3.3 V. Using shutdown mode.</td>
</tr>
</tbody>
</table>

Source: data sheet in [23].

### LIS3L02AQ: 2G/6G Accelerometer

<table>
<thead>
<tr>
<th>Power requirements</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage(3)</td>
<td>2.4</td>
<td>3.3</td>
<td>3.6</td>
<td>V</td>
<td>V\text{DD}</td>
</tr>
<tr>
<td>Supply Current</td>
<td>0.85</td>
<td>1.5</td>
<td>mA</td>
<td></td>
<td>I\text{DD}</td>
</tr>
<tr>
<td>Supply Current in Power Down Mode</td>
<td>2</td>
<td>5</td>
<td>µA</td>
<td></td>
<td>I\text{DD}</td>
</tr>
</tbody>
</table>

Source: data sheet in [24].
Appendix B - Abbreviations

PAN

A **personal area network** (PAN) is a computer network used for communication among computer devices (including telephones and personal digital assistants) close to one person. The devices may or may not belong to the person in question. The reach of a PAN is typically a few meters.

Reference from Wikipedia, URL: http://en.wikipedia.org/wiki/Personal_area_network

ISM band

The **industrial, scientific and medical** (ISM) radio bands were originally reserved internationally for non-commercial use of RF electromagnetic fields for industrial, scientific and medical purposes.


Interrupt handler

An **interrupt handler**, also known as an interrupt service routine, is a subroutine in an operating system or device driver whose execution is triggered by the reception of an interrupt. Interrupt handlers have a multitude of functions, which vary based on the reason the interrupt was generated and the speed at which the Interrupt Handler completes its task.


J2ME

**Java Platform, Micro Edition** or **Java ME** (formerly referred to as Java 2 Platform, Micro Edition or J2ME), is a collection of Java APIs for the development of software for resource constrained devices such as PDAs, cell phones and other consumer appliances.

Reference from Wikipedia, URL: http://en.wikipedia.org/wiki/Java_ME
CLDC

The **Connected Limited Device Configuration** (CLDC) is a specification of a framework for Java ME applications targeted at devices with very limited resources such as pagers and mobile phones. The CLDC was developed under the Java Community Process as JSR 30 (CLDC 1.0) and JSR 139 (CLDC 1.1).

Reference from Wikipedia, URL:

ECC

**Elliptic curve cryptography** (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields

Reference from Wikipedia, URL:
http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

RX: Receive Mode
TX: Transmit Mode
Appendix C - SPOT Application Code – SPOT to SPOT

Note. There might be different port for different SPOTs. Use –Ddebugclient.port=XXX to config when deploying or running an application.

Application Code for RX SPOT

/ *
 * Startup.java
 *
 * Created on 5/1/2006
 *
 * This is the program which receiving data from the other SPOT.
 * The data contains with temperature and light value which is
 * detected from the sending SPOT. When the data is received,
 * then the data is processed to flash LED1.
 *
 */

package squawk.application;

import java.io.DataInputStream;
import java.io.IOException;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.squawk.peripheral.io.RangeInput;
import com.sun.spot.sensorboard.SensorBoardColouredLED;
import com.sun.squawk.peripheral.NoAckException;

public class Startup {
    
    SensorBoardColouredLED Led1;
    protected final static String LIGHT_MESSAGE = "photo: ";
    protected final static String TEMPERATURE_MESSAGE = "temp: ";
    StreamConnection conn;
    DataInputStream dis;
    int tempValue;
    int lightValue;

    public Startup() throws IOException {
        initLEDs();
        initConnection();
    }
}
readDatagram();
}

//Read datagram from the other SPOT
public void readDatagram() throws IOException{
    String message = "";
    dis = conn.openDataInputStream();
    while(true) {
        try {
            message = dis.readUTF(); // This will block util message arrives.
            System.out.println(message);
            if (message.startsWith(LIGHT_MESSAGE)) {
                processLightMsg(message);
            }
            if (message.startsWith(TEMPERATURE_MESSAGE)) {
                processTempMsg(message);
            }
        } catch (NoAckException e) {} 
    }
}

//Process temperature message from received datagram
public void processTempMsg(String message) throws IOException{
    Integer tValue =
    Integer.valueOf(message.substring(TEMPERATURE_MESSAGE.length(),
    message.length()));
    tempValue = tValue.intValue();
    setLightAndTempLed1(tempValue,lightValue);
}

//Process light message from received datagram
public void processLightMsg(String message) throws IOException{
    Integer lValue = Integer.valueOf(message.substring(LIGHT_MESSAGE.length(),
    message.length()));
    lightValue = lValue.intValue();
    setLightAndTempLed1(tempValue,lightValue);
}

//Turn on the Led with value tv,lv
public void setLightAndTempLed1(int tv,int lv) throws IOException{
    Led1.setRGB(tv,0,lv);
}

//Initialize LED1
public void initLEDs() throws IOException {
    Led1 = SensorBoardColouredLED.getLed1();
}
Led1.setOn();
Led1.setRGB(0,0,0); //Blink it out
}

//Initialize connection
public void initConnection() throws IOException {
    conn = (StreamConnection) Connector.open("radio://29:100");
}

/** Creates a new instance of Startup */
public static void main(String[] args) throws IOException {
    Startup s = new Startup();
}

---

Application Code for TX SPOT

/*
* Startup.java
*
* Created on 5/1/06
*
* The application retrieves data from SPOT temperature sensor and
* light sensor, then display the value via LED1 intensity. Blue
* represents light and Red represents temperature. Once the data
* is collected, the SPOT will also send the data to the other SPOT.
* The datagram is began with “photo:” and “temp:” to distinguish
* light message or temperature message.
*/

package squawk.application;

import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.IOException;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.spot.sensorboard.SensorBoardColouredLED;
import com.sun.squawk.peripheral.ISwitch;
import com.sun.squawk.peripheral.io.RangelInput;
import com.sun.squawk.peripheral.ILed;
import com.sun.squawk.peripheral.NoAckException;
public class Startup {

    SensorBoardColouredLED Led1;
    RangeInput lightSensor;
    RangeInput tempSensor;
    protected final static String LIGHT_MESSAGE = "photo: ";
    protected final static String TEMPERATURE_MESSAGE = "temp: ";
    ILed greenLED;
    StreamConnection conn;
    DataInputStream dis;
    DataOutputStream dos;
    int tempValue;
    int lightValue;

    public Startup() throws IOException {
        initLEDs();
        initSensors();
        initConnection();

        //Sending datagram for every half second
        try {
            while(true) {
                tempValue = getTempValue();
                lightValue = getLightValue();
                sendTempAndLight(tempValue,lightValue);
                Thread.sleep(500);
            }
        } catch(java.lang.InterruptedIOException e) {
            System.out.println("Error:" + e.toString());
        }
    }

    //Initialize Sensor
    public void initSensors() throws IOException {
        tempSensor = DemoSensorBoard.getTemperatureSensor();
        lightSensor = DemoSensorBoard.getLightSensor();
    }

    //Initialize LED1
    public void initLEDs() throws IOException {
        greenLED = DemoSensorBoard.getGreenLed();
        Led1 = SensorBoardColouredLED.getLed1();
        Led1.setOn();
        Led1.setRGB(0,0,0); //Blink it out
greenLED.setOn(true);

//Retrieve temperature value from light sensor
public int getTempValue() {
    int tv;
    try {
        tv = tempeSensor.getValue();
        System.out.println("Temp Value: "+tv);
    } catch(IOException e) {
        System.out.println("Temp Sensor Error:" + e.toString());
        return -1;
    }
    return tv;
}

//Retrieve light value from light sensor
public int getLightValue() {
    int lv;
    try {
        lv = lightSensor.getValue();
        System.out.println("Light Value: "+lv);
    } catch(IOException e) {
        System.out.println("Light Sensor Error:" + e.toString());
        return -1;
    }
    return lv;
}

//This method is to transfer received value into the range of 0 to 255.
//And send to LED1
public void setTempAndLightToLED1(int tv, int lv) throws IOException{
    //Set Temperature to LED1 blue
    if(tv == -1) {
        System.out.println("Set Temp Error: Value is "+tv+".");
        return;
    } else if(tv >= 255)
        tv = 255;
    else if(tv > 100)
        tv = (tv - 100) * 2 + 100;
    else if(tv < 100)
        tv = 100 - (100 - tv) * 2;
    //SetTemp End
//Set Temperature to LED1 red
if(lv == -1) {
    System.out.println("Set Light Error: Value is " + lv + ".");
    return;
} else if(lv >= 765)
    lv = 765 / 3;
else if (lv < 3 && lv >= 0)
    lv = 0;
else
    lv = lv / 3;

//SetLight End
tempValue = tv;
lightValue = lv;
Led1.setRGB(tv, 0, lv);
}

//Initialize connection
public void initConnection() throws IOException {
    conn = (StreamConnection) Connector.open("radio://28:100");
}

//Send temperature and light datagram to the other SPOT
public void sendTempAndLight(int tv, int lv) throws IOException {
    dos = conn.openDataOutputStream();
    greenLED.setOn();//turn off
    try {
        dos.writeUTF(this.LIGHT_MESSAGE + lv);
        dos.writeUTF(this.TEMPERATURE_MESSAGE + tv);
        dos.flush();
    }
    catch (NoAckException e) {
        //The Exception is occured when the other SPOT is not
        //Available
        System.out.println("Error: "+e.toString());
        greenLED.setOff();//turn on when no connection
    }
}

/** Creates a new instance of Test */
public static void main(String[] args) throws IOException {
    Startup s = new Startup();
}
}
Appendix D - Basestation Application code

Note. There might be different port for different SPOTs. Use
–Ddebugclient.port=XXX to config when deploying or running an application.

Basestation SPOT code

/*
 * GUI.java
 * Created on 5/20/06
 * This java class contains with pure graphical user interface.
 * A JTextArea is to show the data from remode SPOT.
 * A JButton is to clear up JTextArea.
 */

package squawk.application;

import javax.swing.*;

/**
 * @author Dion
 */
public class GUI extends javax.swing.JFrame {
    
    /** Creates new form GUI */
    public GUI() {
        initComponents();
        super.setSize(400,600);
        super.setTitle("Basestation Application");
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code ">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;
        jscrollPane1 = new javax.swing.JScrollPane();
        jtxtAMsg = new javax.swing.JTextArea();
        jbtnClear = new javax.swing.JButton("Clear");

    }
    
    // </editor-fold>
}
This class is to act as basestation communicate with both
host and the remote SPOT. When basestation SPOT received
package squawk.application;

import java.io.DataInputStream;
import java.io.IOException;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

public class Startup {
    StreamConnection conn;
    DataInputStream dis;
    GUI gui;

    public Startup() throws IOException {
        UI();
        initConnection();
        readDatagram();
    }

    // Read datagram from the other SPOT and display
    // on JTextArea of GUI instance
    public void readDatagram() throws IOException {
        String message = "";
        dis = conn.openDataInputStream();
        while (true) {
            try {
                message = dis.readUTF();
                gui.txtAMsg.append(message + "\n");
            } catch (IOException e) {}
//Create a class of GUI and initialize
public void UI(){
    gui = new GUI();
    gui.show();
}

/** Creates a new instance of Test */
public static void main(String[] args) throws IOException {
    LowPanPacketDispatcher.getInstance().initBaseStation();
    Startup s = new Startup();
}

Remote SPOT code

/*
* Startup.java
*
* Created on 05/20/06
* The application first starts to accumulate data into buffer, once the
* remote SPOT (basestation) is in the networking coverage. Then the
* buffer is sent to the remote SPOT. Otherwise, the buffer will keep
* accumulating data. To start sending data and retrieving sensor data,
* switch 1 is the activator. Switch 2 is to stop sending and collecting
* data. Switch 1 and switch 2 pressed simultaneously to clear up the buffer
* when the SPOT is not sending or collecting. *
*/

package squawk.application;

import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import com.sun.spot.sensorboard.DemoSensorBoard;
import com.sun.spot.sensorboard.SensorBoardColouredLED;
import com.sun.squawk.peripheral.ISwitch;
import com.sun.squawk.peripheral.io.RangeInput;
import com.sun.squawk.peripheral.ILed;
import com.sun.squawk.peripheral.NoAckException;
import com.sun.squawk.peripheral.spot.*;
public class Startup {

  ILed greenLED;
  SensorBoardColouredLED Led1;
  StreamConnection conn;
  DataOutputStream dos;
  int tempValue, lightValue;
  RangeInput tempSensor, lightSensor;
  private int RGBg = 0, RGBr = 0, RGBb = 0;
  StringBuffer buffer;
  ISwitch sw1, sw2;

  //DataInputStream dis; //Flash memory test instance
  //IFlashMemoryDevice mem; //Flash memory test instance
  //int firstPos, pos; //Flash memory test instance

  public Startup() throws IOException{
    initLeds();
    initSensors();
    initConnection();
    initSwitch();
    clearBuffer();

    try {
      while(true) {
        resetRGBVal();
        //press switch one and two, clear out the buffer.
        if(sw1.isClosed() && sw2.isClosed()) {
          this.clearBuffer();
          turnRGBredASec();
        // press switch one, collecting data and sending data
        } else if(sw1.isClosed() && sw2.isOpen()) {
          long startTime = System.currentTimeMillis();
          buffer.append("Start Time:"+startTime+"\n");
          while(true){
            collectDataToBuffer();
            sendBufferToSpot();
            Thread.sleep(2000);
            // switch two to terminate the collecting and sending
            if(sw2.isClosed() && sw1.isOpen()) {
              this.resetRGBVal();
              this.greenLED.setOn(true);
              this.turnOnRGB1();
              
```
break;
}
}
}
Thread.sleep(2000);
}
}
catch(InterruptedException e) {
    this.turnRGBred();
    this.turnRGBgreen();
    this.turnRGBblue();
    this.turnOnRGB1();
    System.out.println("Error:" + e.toString());
}

//initialize connection
public void initConnection() throws IOException {
    conn = (StreamConnection) Connector.open("radio://28:100");
}

//initialize Sensor
public void initSensors() throws IOException {
    tempSensor = DemoSensorBoard.getTemperatureSensor();
    lightSensor = DemoSensorBoard.getLightSensor();
}

//Initialize LED
public void initLeds() {
    greenLED = DemoSensorBoard.getGreenLed();
    Led1 = SensorBoardColouredLED.getLed1();
    Led1.setOn();
    greenLED.setOn(true);
}

//Initialize switch1 and switch2
public void initSwitch() {
    sw1 = DemoSensorBoard.getSwitch1();
    sw2 = DemoSensorBoard.getSwitch2();
}

//Turn red LED for 0.2 second. This is treated as a message
//to show the success of an operation such as clear buffer.
public void turnRGBredASec() throws InterruptedException {
    this.turnRGBred();
    this.turnOnRGB1();
Thread.sleep(200);
this.resetRGBr();
this.turnOnRGB1();
}

public void turnRGBgreen() {
    this.RGBg = 50;
}

public void turnRGBblue() {
    this.RGBb = 50;
}

public void turnRGBred() {
    this.RGBr = 50;
}

public void resetRGBg() {
    this.RGBg = 0;
}

public void resetRGBb() {
    this.RGBb = 0;
}

public void resetRGBr() {
    this.RGBr = 0;
}

//Reset all preset value
public void resetRGBVal() {
    resetRGBr();
    resetRGBg();
    resetRGBb();
}

//Retrieve value from temperature sensor
public int getTempValue() {
    int tv;
    try {
        tv = tempeSensor.getValue();
        turnRGBblue();
    } catch(IOException e) {
        System.out.println("Temp Sensor Error:" + e.toString());
        return -1;
    }
}
return tv;
}

//Retrieve value from light sensor
public int getLightValue() {
  int lv;
  try {
    lv = lightSensor.getValue();
    turnRGBblue();
  } catch(IOException e) {
    System.out.println("Light Sensor Error:" + e.toString());
    return -1;
  }
  return lv;
}

//Send buffer to remote spot (basestation)
public void sendBufferToSpot() throws IOException {
  greenLED.setOn(true);

  try {
    //Exam the connection is available
    dos = conn.openDataOutputStream();
    dos.writeUTF(""");
    dos.flush();

    //If available, append end time to the buffer.
    buffer.append("End Time:" + System.currentTimeMillis()+"n");

    dos.writeUTF(buffer.toString()+"Length:"+buffer.toString().length());
    dos.flush();

    clearBuffer();
    this.turnRGBgreen();
    this.turnOnRGB1();
  } catch(NoAckException e) {
    System.out.println("Error: " + e.toString());
    greenLED.setOn(false);
    this.resetRGBg();
  }
}

//append temperature and light value to buffer
public void collectDataToBuffer(){
  long revTime;
  int tv,lv;
this.resetRGBb();

tv = getTempValue();
lv = getLightValue();
revTime = System.currentTimeMillis();

buffer.append("After receiving sensors Time:" + revTime+"\n");
buffer.append("T:"+tv+"\n");
buffer.append("I:"+lv+"\n");

//See the content of buffer for debug purpose. 
System.out.println(buffer.toString());

turnOnRGB1();
}

//Clear up buffer
public void clearBuffer() {
    buffer = new StringBuffer();
}

//Turn LED1 on with preset value
public void turnOnRGB1() {
    Led1.setRGB(RGBr,RGBg,RGBb);
}

/** Creates a new instance of Test */
public static void main(String[] args) throws IOException {
    Startup s = new Startup();
}
}