Recap: OFDM

- Basic idea: use multiple subcarriers to reduce symbol rate per carrier
- Problem: inter-carrier-interference (ICI)
  - Idea: orthogonal subcarriers to avoid ICI, chose each subcarrier frequency so that an integral number of cycles in a symbol period
- Problem: implementation complexity
  - Idea: iFFT as an efficient algorithm for modulation on multiple subcarriers
- Problem: ISI
  - Idea: cyclic prefix (guard interval)

Recap: Summary of Wireless PHY

```
<table>
<thead>
<tr>
<th>Transmitter</th>
<th>Scrambler</th>
<th>Interleaving</th>
<th>Convolutional Encoder</th>
<th>QAM Modulator</th>
<th>Symbol Wave</th>
<th>Demodulator</th>
<th>Descrambler</th>
<th>Interleaving</th>
<th>Viterbi Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver</td>
<td>Scrambler</td>
<td>Interleaving</td>
<td>Demodulator</td>
<td>Viterbi Decoder</td>
<td>Descrambler</td>
<td>Interleaving</td>
<td>Convolutional Encoder</td>
<td>Interleaving</td>
<td>Symbol Wave</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Recap: Mobile Software Development Framework: TinyOS, J2ME

- A free and open source component based operating system and platform targeting wireless sensor networks (WSNs)
- Some design features
  - Problem: small footprint
    - Idea: TinyOS. Generate customized OS + application. Support one app at a time but flexible reprogramming

Recap: GNU Radio

- A software development toolkit that provides signal processing blocks to implement software-defined radio systems.
- Some key ideas
  - Hybrid software system (Python/C++)
  - Composite pattern to build a hierarchy of blocks
  - Internal scheduler to orchestrate data flows among blocks
  - Define gr_block as a reusable base so that defining a new block is typically simple: overwrite general_work and forecast

Recap: TinyOS

- Admin.
  - Homework 2 posted
  - Start to think about project

1. Mobile Software Development Framework: TinyOS, J2ME

   10/2/2012

   Y. Richard Yang

2. Recap: OFDM

   - Basic idea: use multiple subcarriers to reduce symbol rate per carrier
   - Problem: inter-carrier-interference (ICI)
     - Idea: orthogonal subcarriers to avoid ICI, chose each subcarrier frequency so that an integral number of cycles in a symbol period
   - Problem: implementation complexity
     - Idea: iFFT as an efficient algorithm for modulation on multiple subcarriers
   - Problem: ISI
     - Idea: cyclic prefix (guard interval)

3. Recap: Summary of Wireless PHY

   ![PHY operations of IEEE 802.11a/b/g transceiver.](Image)

4. Recap: GNURadio

   GNU Radio

   - A software development toolkit that provides signal processing blocks to implement software-defined radio systems.
   - Some key ideas
     - Hybrid software system (Python/C++)
     - Composite pattern to build a hierarchy of blocks
     - Internal scheduler to orchestrate data flows among blocks
     - Define gr_block as a reusable base so that defining a new block is typically simple: overwrite general_work and forecast

5. Recap: TinyOS

   - Admin.
     - Homework 2 posted
     - Start to think about project

6. Recap: TinyOS

   - Admin.
     - Homework 2 posted
     - Start to think about project
TinyOS: Software Concepts

- A TinyOS consists of one or more components/modules linked together:
  - software components motivated by hardware component
- Each component specifies:
  - it provides some interfaces
    - allows other components to control it
  - also uses some interfaces
    - control other components

Interface

- An interface declares:
  - a set of functions called commands that provider must implement
  - another set of functions called events that the interface user must implement

Interface: Examples

StdControl.nc

```plaintext
interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

Timer.nc

```plaintext
interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event result_t fired();
}
```

ADC.nc

```plaintext
interface ADC {
    async command result_t getdata();
    async command result_t getContinuousData();
    event result_t dataReady(uint16_t data);
}
```

Example Application

- A simple TinyOS application which periodically reads the light intensity value, computes a moving average, displays it on the LED

Module: Definition

SenseTaskM.nc

```plaintext
module SenseTaskM {
    provides {
        interface StdControl;
    }
    uses {
        interface Timer;
        interface ADC;
        interface ADC as ADCControl;
        interface Leds;
    }
}
```

Module: Implementation

- Define:
  - commands and event handlers
  - frame (storage)
    - statically allocated, fixed size to know memory requirement and avoid overhead of dynamic allocation

- See SenseTaskM.nc
Explicit Linking of Components

Two types of components:

- modules: individual components
- configurations: assemble components together, connecting interfaces (objects) used by components to interfaces (objects) provided by others

  - See SenseTask.nc

TinyOS Execution Model

- Concurrency model: only two threads
  - long running tasks that can be interrupted by hardware
  - event handlers
- Tasks are posted to a FIFO queue
  - Each task is atomic with respect to other tasks
  - run to completion, but can be preempted by events
  - the task scheduler is a simple FIFO scheduler
- Tasks perform the primary computation work
  - commands and event handlers post tasks
  - call lower level commands
  - signal higher level events
  - schedule other tasks within a component

Running tinyOS Program

- make mica
- ncc -o main.exe -target=mica SenseTask.nc
- avr-objcopy --output-target=srec main.exe main.srec
- Use uisp to install

A More Complete Sample Application

- Sensor network monitoring
  - monitor temperature and light conditions
  - periodically transmit measurements to a base station
  - sensors can forward data for other sensors that are out of range of the base station
  - dynamically determine the correct routing topology for the network

Internal Component Graph
Message Send Transition

Total propagation delay up the 5 layer radio communication stack is about 80 instructions.

Timing diagram of event propagation.

Evaluation: Storage

- Scheduler only occupies 178 bytes.
- Complete application only requires 3 KB of instruction memory and 226 bytes of data (less than 50% of the 512 bytes available).
- Only processor_init, TinyOS scheduler, and C runtime are required.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Code Size</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>AIR_digitch</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>AIR_Temperature</td>
<td>78</td>
<td>32</td>
</tr>
<tr>
<td>AIR_light</td>
<td>246</td>
<td>8</td>
</tr>
<tr>
<td>AIR</td>
<td>296</td>
<td>40</td>
</tr>
<tr>
<td>C2D</td>
<td>234</td>
<td>40</td>
</tr>
<tr>
<td>RADIO_packet</td>
<td>830</td>
<td>8</td>
</tr>
<tr>
<td>BPM</td>
<td>333</td>
<td>1</td>
</tr>
<tr>
<td>Light</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>Temp</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>UART</td>
<td>359</td>
<td>1</td>
</tr>
<tr>
<td>UART_packet</td>
<td>314</td>
<td>40</td>
</tr>
<tr>
<td>I2C</td>
<td>198</td>
<td>8</td>
</tr>
<tr>
<td>Procedure_q</td>
<td>172</td>
<td>50</td>
</tr>
<tr>
<td>TinyOS scheduler</td>
<td>178</td>
<td>19</td>
</tr>
<tr>
<td>C runtime</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3450</td>
<td>226</td>
</tr>
</tbody>
</table>

Evaluation: Timing

<table>
<thead>
<tr>
<th>Operations</th>
<th>Cost (cycles)</th>
<th>Time (µs)</th>
<th>Normalized to byte copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte copy</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Post an Event</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Call a Command</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Post a task to scheduler</td>
<td>46</td>
<td>11.5</td>
<td>6</td>
</tr>
<tr>
<td>Context switch overhead</td>
<td>51</td>
<td>12.75</td>
<td>6</td>
</tr>
<tr>
<td>External (hardware cost)</td>
<td>9</td>
<td>2.25</td>
<td>1</td>
</tr>
<tr>
<td>External (software cost)</td>
<td>71</td>
<td>17.75</td>
<td>9</td>
</tr>
</tbody>
</table>

Summary: TinyOS

- Components: provide commands and require callback hooks for event-driven programming.
- Configurations: Link components.
- TinyOS: an app (configuration) at a time, linking only necessary components.
- Two threads exec: one for event, one for task.

Discussion: Compare TinyOS/GNURadio

- What are some similar software concepts?
- What are some differences?

Discussion

- Can we use GNURadio/TinyOS for writing mobile applications for mobile phones, or in other words, what are missing?
Java2 Micro Edition (J2ME)

Java Platforms

J2ME

Java2 is divided into three platforms

- J2EE (Java2 Enterprise Edition)
  - business applications
- J2SE (Java2 Standard Edition)
  - general applications
- J2ME (Java2 Micro Edition)
  - small devices such as mobile phone, PDA, car navigation

Oracle's claims on Java on mobile devices

http://www.java.com/en/about/

J2ME Basic Concepts: Versioning

To accommodate heterogeneous mobile devices, define configurations and profiles

- A configuration provides fundamental services for a broad category of devices (e.g., lang, io, util)
- A profile supports higher-level services common to a more specific class of devices or market (e.g., life cycle, GUI)
- An optional package adds specialized services that are useful on devices of many kinds, but not necessary on all of them

Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
  - TinyOS
  - J2ME

Example J2ME Configurations

- Connected Limited Device Configuration (CLDC)
  - 160 KB to 512 KB of total memory available
  - 16-bit or 32-bit processor
  - low power consumption and often operating with battery power
  - connectivity with limited bandwidth
  - examples: cell phones, certain PDAs
- Connected Device Configuration (CDC)
  - 2 MB or more memory for Java platform
  - 32-bit processor
  - high bandwidth network connection, most often using TCP/IP
  - examples: set-top boxes, certain PDAs
CLDC Available Packages

- java.lang
- java.util
- java.io
- javax.microedition.io

CLDC Classes

- Boolean, Byte, Character, Class, Integer, Long, Math, Object, Runnable, Runtime, Short, String, StringBuffer, System, Thread, Throwable
- java.lang, java.util, java.io

Example J2ME Profiles

- Mobile Information Device Profile (MIDP) - GUI, multimedia and game functionality, end-to-end security, and greater networking connectivity, mobile phones and entry level PDAs
- Foundation Profile - set of Java APIs that support resource-constrained devices without a standards-based GUI system
- Personal Profile - Full set of AWT APIs, including support for applets and Xlets, CDC + Foundation Profile + Personal Profile for high-end PDA

Mobile Phone Framework

- MIDP 2.0
- WAP (JSR-133)
- Optional Packages
- MIDP/CLDC Classes
- Native APIs

MIDP Hardware

- Memory (added to CLDC memory) - 128 KB non-volatile for MIDP components, 8 KB non-volatile for application persistent data, 32 KB volatile for KVM
- Display - screen 96x54, display depth 1-bit, pixel shape (aspect ratio) 1:1

Mobile Phone Hardware

- Input (one or more) - one-handed keyboard (ITU-T phone keypad), two-handed keyboard (QWERTY keyboard), or touch screen
- Networking - two-way, wireless, possibly intermittent, limited bandwidth
MIDP Packages

- java.io
- java.lang
- java.util
- javax.microedition.io
- javax.microedition.lcdui
- javax.microedition.rms
- javax.microedition.midlet

version 1.0 addition in version 2.0

MIDP Technology Stack

MIDlet

- GUI based
- Each MIDP has one instance of Display
  - Display.getDisplay(this) to get the manager
  - At any instance of time at most one Displayable object can be shown on the display device and interact with user
    - display.setCurrent(<Displayable object>)

- An MIDP application is called a MIDlet
  - similar to the J2SE applet

- A MIDlet moves from state to state in the lifecycle, as indicated
  - start - acquire resources and start executing
  - pause - release resources and become quiescent (wait)
  - destroy - release all resources, destroy threads, and end all activity

MIDlet

- Displayable
  - Canvas
    - GameCanvas
  - Screen
    - Alert, List, TextBox, Form

- Form can contain multiple form items for organization
  - Labels, Image Items, String Items, Text Fields, Date Fields, Gauge, Choice Groups

MIDP Visual Display

- Each MIDP has one instance of Display
  - Display.getDisplay(this) to get the manager
  - At any instance of time at most one Displayable object can be shown on the display device and interact with user
    - display.setCurrent(<Displayable object>)
**MIDP: User Interaction**

- **Displayable objects can declare commands and declare a command listener:**
  - addCommand(Command cmd)
  - addCommand_listener()

- **Command(**label**, <type>, <priority>**)
  - **Type:** BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, and STOP

---

**HelloWorldMIDlet.java**

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloWorldMIDlet extends MIDlet implements CommandListener {
    private Command exitCommand;
    private Display display;
    private TextBox t;
    public HelloWorldMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.EXIT, 2);
        t = new TextBox("CS434", "Hello World!", 256, 0);
        t.addCommand(exitCommand);
        t.setCommandListener(this);
    }
    public void startApp() { display.setCurrent(t); }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

---

**MIDP: Persistent State**

- **Record store defined in javax.microedition.rms**
- **Record store identified by name:**
  - static String[] listRecordStores();
  - recordStore = RecordStore.openRecordStore("scores", true);
  - recordId = addRecord(byte[] data, int offset, int numBytes);
  - getRecord(int recordId);

---

**Summary: J2ME**

- Scale down a popular programming environment to ease learning
- Use virtual machines to mask device heterogeneity
- Use configuration/profiling to handle device heterogeneity and avoid using lowest common denominator
- MIDlet to manage app life cycle
- Displayable to visual display, commands and provides command listener
- Introduce persistent record store

---

**Discussion on J2ME**

- What designs of J2ME do you like and hence expect newer frameworks (e.g., IOS, Android) may have too?
- What features do you think are missing in J2ME?