Lesson 1

What is TinyOS?

- A small operating system for Microcontrollers
  - Create a uniform abstraction (e.g. Device Abstraction)
- An Open-Source Development Environment
- A Component Based Architecture
- A Programming Language & Model
  - nesC Language
nesC

- nesC is an extension of C
- Built on top of avg-gcc
- “Static Language”
  - No Dynamic Memory (no malloc)
  - No Function Pointers
  - No Heap
- TinyOS evolved to nesC
- Java influence
- nesC uses the filename extension ".nc"

Programming Model

- Separation of construction and composition
- Specification of component behavior in terms of set of interfaces.
- Components are statically linked together
- Finite State Machine Programming Style
  - Non-blocking
Basic Constructs

- **Commands** — Cause action to be initiated.
- **Events** — Call back to notify action has occurred and give results.
- **Tasks** — Background computation, non-time critical
- **Modules** — Component implemented with C Code
- **Configurations** — Component implemented with Wires
- **Interfaces** — specifications of bi-directional communications for the components

Basic Concepts

- **Interfaces** (xxx.nc)
  - Specifies functionality to outside world
  - what commands can be called
  - what events need handling

- **Software Components**
  - **Module** (xxxM.nc)
    - Code implementation
    - Code for **Interface** functions
  - **Configuration** (xxxC.nc)
    - Linking/wiring of components
    - When top level app, drop C from filename xxx.nc
The Design of TinyOS

- TinyOS/nesC is designed to speed application development through code reuse.
- The number of modules per application in the TinyOS-1.x release ranges from 8 to 67, with an average of 24. ***
- The average lines of code in a module only 120***
- Advantages of eliminating monolithic programs
  - Code can be reused more easily
  - Number of errors should decrease.


Data Flow

- Tasks
- Events
- Commands
- Interrupts
- Hardware
Thread Model

- Threads of control are rooted in two places
  - Hardware Interrupts
  - Tasks
- Threads of control may pass into component through its interfaces to another component.
- Interrupt Handlers may interrupt tasks and other interrupts.
- Tasks run to completion and may NOT be interrupted by other Tasks.
- nesC scheduler executes Tasks from a FIFO queue

Commands

- **Commands** are called with “call”
  
  ```
  call Timer.start(TIMER_REPEAT, 1000);
  ```
- Cause action to be initiated
- Bounded amount of work
  - cannot block
- Acts similar to a function call
  - Execution of a command is immediate
Events

- **Events** are called with “Signal”
  ```
  signal ByteComm.txByteReady(SUCCESS);
  ```
- Normally used to notify a **component** an action has occurred
- Used to deliver data from hardware
- Lowest-level events triggered by hardware interrupts
- Bounded amount of work
  - cannot block
- Acts similar to a function call
  - Execution of a **event** is immediate

Tasks

- **Tasks** are queued with “post”
  ```
  post radioEncodeThread();
  ```
- Used for longer running operations
- Are pre-empted by **Events**
  - initiated by interrupts
- **Tasks** run to completion
- Not pre-empted by other **tasks**.
Example Tasks

- High Level
  - Calculate aggregate of sensor readings

- Low Level
  - Encode radio packet for Transmission
  - Calculate CRC

Execution Flow
Component

- Two types of components in nesC:
  - Module
  - Configuration
- A component provides and uses Interfaces

Module

- Provides Application Code
  - Contains “C” like code
- Must implement the ‘provides’ interfaces
  - Implement the “Commands” you are providing
  - Make sure to actually Signal
- Must implement the ‘uses’ interfaces
  - Implement the “Events” that need to be handled
  - Invoke commands as needed
Configuration

- A configuration is a component that "wires" other components together.
- Configurations are used to assemble other components together.
- Connects interfaces used by components to interfaces provided by others.

Interfaces

- Specifies the behavior between components.
- Bi-directional multi-function interaction channel between two components.
  - Allows a single interface to represent a complex event
    - E.g., a registration of some event, followed by a callback
    - Critical for non-blocking operation
  - Provided interfaces
    - Represent the functionality that the component provides to its user
    - "Commands" are functions to be implemented by the interface’s provider
  - Used interfaces
    - Represent the functionality that the component needs
    - "Events" are functions to be implemented by the interface’s user
Interface Example

Interface SendMsg {
    command result_t send(uint16_t address, uint8_t length,
                          TOS_MsgPtr msg);
    event result_t sendDone(TOS_MsgPtr msg, result_t success);
}

call SendRFM.send( TOS_BCAST_ADDR, sizeCurrentPkt, sendMsg);

provides {
    interface StdControl as Control;
}
uses {
    ...
    interface SendMsg as SendRFM;
    interface ReceiveMsg as ReceiveRFM;
    interface SendMsg as SendWriteRFM;
}

event result_t SendRFM.sendDone(
    TOS_MsgPtr msg, result_t success)
{
    return SUCCESS;
}

Application

- Consists of one or more components, wired together to form a runnable program
- Has a single top-level configuration that specifies the set of components in the application and how they connect one another
- Connected wire to main component to start execution
Components/Wiring

- A Directed Wire (Arrow ‘->’) Connects Components
  - Only 2 Components at a time
  - Connection is Across an Interface
  - ‘<’ is equivalent to ‘->’
- [Component that uses the interface] ‘->’ [component that provides the interface]
- An “=” can be used to equate an external specification element, whereas a link connects two internal elements.
- Unused System Components Excluded

Blink.nc Application –
A top level configuration SW component used to form an executable

What the executable does:
1. Main initializes and starts the application.
2. BlinkM initializes ClockC’s rate at 1Hz.
3. ClockC continuously signals BlinkM at a rate of 1 Hz.
4. BlinkM commands LedsC red led to toggle each time it receives a signal from ClockC.

Note: The StdControl interface is similar to state machines (init, start, stop); used extensively throughout TinyOS apps & libs
Tips

- Your friend is grep or “find in files”
- Look at example applications in the /apps directory.
- Try to keep commands and events short.
  - Avoid long loops