Other Sensor Network Platforms: Motes, TinyOS, and NesC

Lecture #5

Previous Lecture

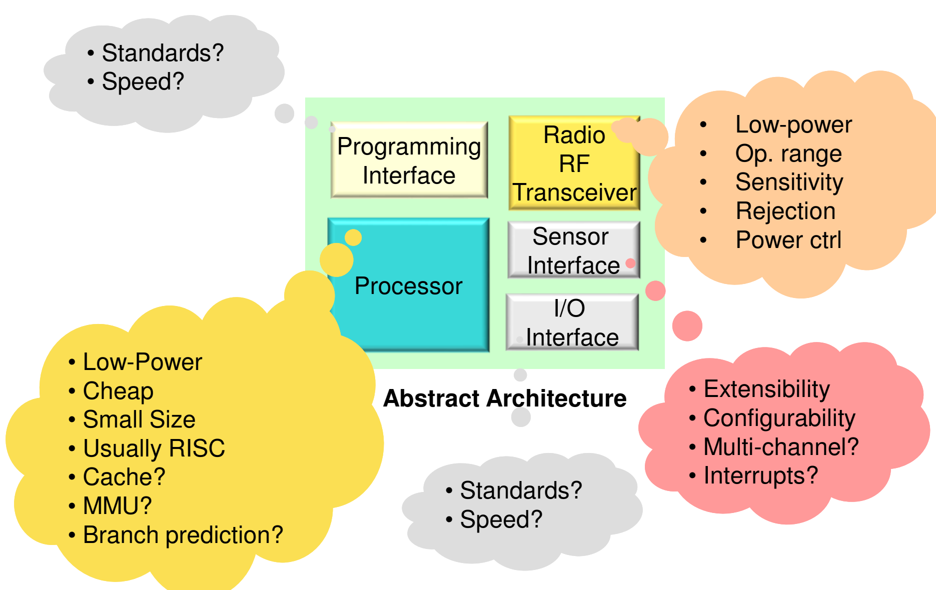
- List of potential course projects
  - Project schedule
    - Organize into groups as soon as possible
  - Project ideas
    - Decide on a project idea (First Come First Served)
    - Schedule meetings with TAs
      - Define project scope and deliverables
  - Project is responsible for 60% of the course grade

Outline of This Lecture

- Overview of other 802.15.4 platforms
- Hardware
  – Available Options
- Operating systems
  – Current flavors
  – Focus on TinyOS
- Programming abstractions
  – Possible paradigms
  – Focus on NesC

Dedicated lectures on some of these topics later

Hardware Platforms



Processor Options

- ATMEL – AVR 8-bit Microcontrollers
  - Ex: Atmega 1281, Atmega 128L etc.
- 8051 derivatives (8-bit)
  - Ex: Atmega 80251, NXP P80C31
- MSP430 (16-bit)
- 32-bit ARM family (ARM7 and ARM9)
  - Ex: NXP LPC2917, TI TMS470
- Other factors to consider:
  - Compiler support
  - Community size

Think about:
- RAM
- Flash
- Speed
- Power states
- Interfaces

Radio Transceiver Options

- 2.4 GHz
  - Chipcon CC2420, Atmel ATRF230
- Sub-1GHz
  - Chipcon CC1000, Atmel ATR2406
- Encryption support
- Protocol support – 802.15.4 Zigbee
- Advanced features:
  - Link quality indicators
  - Localization engines
  - Battery status monitoring

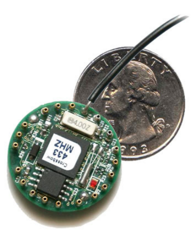
802.15.4 Vs
- Bluetooth
- 802.11
- IR

Peripheral Interfaces

- Programming and I/O interfaces
  - I²C (Inter-Integrated Circuit)
  - SPI (Serial Programming Interface)
  - UART
  - GPIO (General-Purpose Input/Output)
  - USB
  - SDIO (Secure Digital Input/Output)
  - DMA (Direct Memory Access)
- Sensor interface
  - Analog interfaces
    - Voltage ranges and digital precision
  - Digital interfaces

Mica Motes

- Developed at Berkeley
- TinyOS, Contiki, Nano-RK, SOS, Mantis
- Atmel Processors



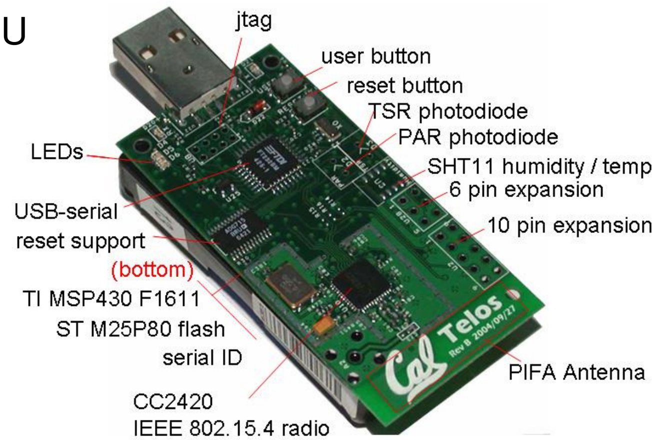
Mica Dot

Mica 2

MicaZ

Teleos

- Developed at Berkeley
- TinyOS, Contiki
- MSP430 CPU



IRIS

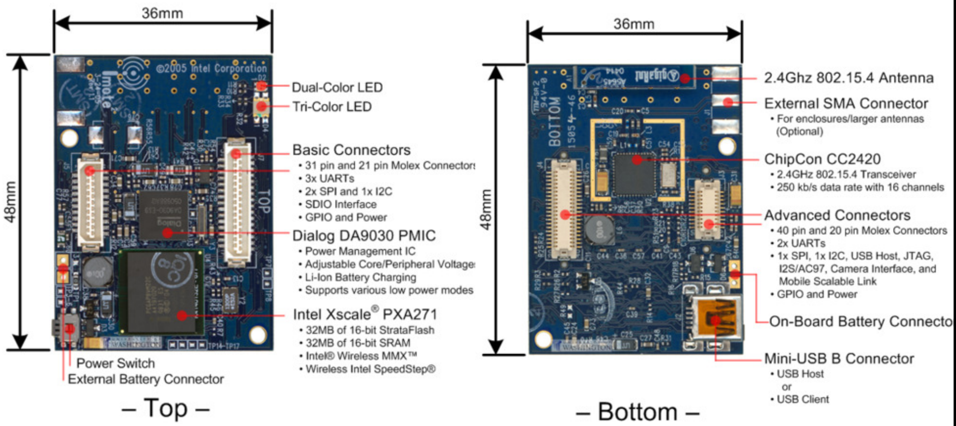
- Developed by Crossbow
- TinyOS, Contiki
- ATmega1281
- Atmel Radio

IRIS

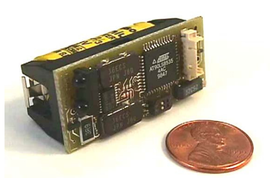
eKo (IRIS inside)

iMote 2

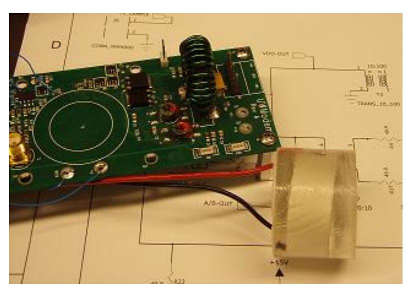
- Developed originally by Intel
- TinyOS, Linux
- PXA271 processor
- cc2420 radio



Non-RF Solutions



IrDA mote



Underwater Acoustic

Powerline Communication

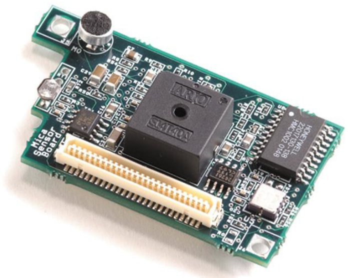
SoC Platforms

- System On a Chip (SoC)
  - CPU and radio on a single chip
  - Decrease size and cost
  - TI cc2430 / cc2431
  - Jennic, Ember
  - Atmel AT128RFA1

Sensor Networking Platforms Overiew

Node	CPU	Freq.	RAM	Radio	Freq.	Protocol	OS (API)
Mica Dot	atmel	4MHz	1kB	Rfm (ook)	416MHz	n/a	TinyOS
Mica 2	ATmega 128L	8MHz	4kB	cc1000	916MHz	n/a	TinyOS
MicaZ	ATmega 128L	8MHz	4kB	cc2420	2.4GHz	802.15.4	TinyOS, Contiki, Nano-RK
Teleos	MSP430	8MHz	10kB	cc2420	2.4GHz	802.15.4	TinyOS
Epic	MSP430	8MHz	10kB	cc2420	2.4GHz	802.15.4	TinyOS
FireFly	ATmega 1281	8MHz	8kB	cc2420	2.4GHz	802.15.4	Nano-RK
IRIS	ATmega 1281	8MHz	8kB	rf230	2.4GHz	802.15.4	TinyOS, Contiki
Imote2	pxa271	416MHz	32MB	cc2420	2.4GHz	802.15.4	TinyOS, Linux
Xbee	?	?	?	?	?	802.15.4	(Serial)
ANT	?	?	?	nRF240ap1	2.4GHz	ANT+	(Serial / I2C)

Sensor Add-ons

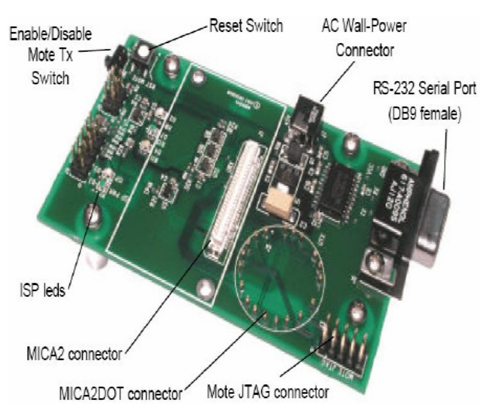


Light
Temperature
Accelerometer
Magnetometer
Speaker
Microphone

Camera
Microphone

MIT Cricket:
Ultrasound for location

Programming and I/O Interface Board



MIB510

MIB520

Outline of This Lecture

- Overview of other platforms
- Hardware
  - Available options
- Operating systems
  - Current flavors
  - Focus on TinyOS
- Programming abstractions
  - Possible paradigms
  - Focus on nesC

Operating Systems

Menu

- Nano-RK
- Tiny OS
- Mantis
- BT-nut
- SOS
- Contiki
- t-kernel
- RETOS
- ....
- ....

Operating System - Paradigms

- Multithreading
  - Pre-emptive, Co-operative, None
- Memory management
  - Software, Hardware
  - Paging ?
- Monolithic *vs* Micro-kernels
  - Loadable module support
- Event-driven *vs* Time-driven
- Tick-less *vs* Tick-based kernels
  - Energy implications

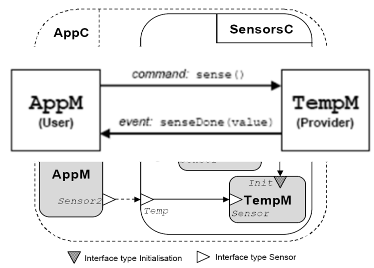
How to get a small "OS"
Do we need everything?
"make" nano-RK each time?
Can an OS reduce power?
Where do we boot from?

WSN Operating Systems

OS	Paradigm	Language	Memory	Code Space
TinyOS	Event-based	nesC	Very Low	Very Compact
Contiki	Cooperative Multitasking	C	Low	Compact
Nano-RK	Priority-based Preemptive Multitasking	C	High	Medium
SOS	Event-based loadable modules	C	Medium	Medium
Mantis	Preemptive Multitasking	C	High	Medium
LiteOS	Preemptive Multitasking, Unix flavor	C	High	Medium

TinyOS

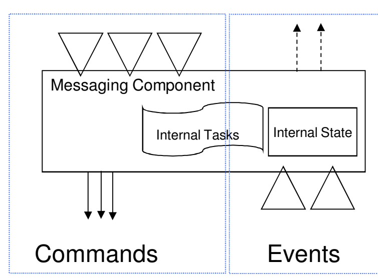
- Event-driven
- Single execution context
- Seed of thought
  - Can we "compose" a system?
- Component-based
- Requires
  - Interfaces
    - Commands and Events
  - State
    - Frame (Storage)
    - Computational Tasks



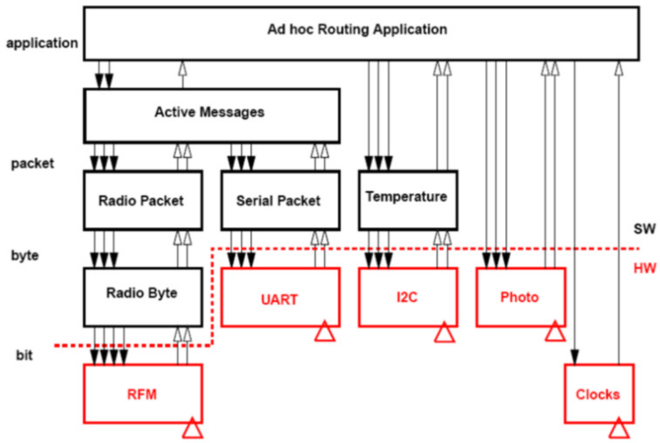
Stack? Sleep?

TinyOS Component Model

- **A component has**
  - Frame (storage)
  - Tasks (computation)
  - Interface
    - Command
    - Event
- **Frame: static storage model**
  - Compile-time memory allocation (efficiency)
- **Commands and events are function calls** (efficiency)



Application Architecture

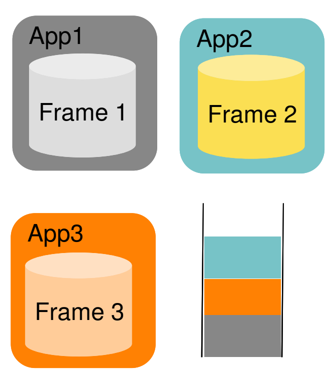


Multiple Apps?
Components?
Possibilities:
Verification?
Power-save?

Components are “Wired” together to create an application

Memory Allocation

- Static memory allocation
  - No heap (malloc)
  - No function pointers
- Global variables
  - Available on a per-frame basis
- Local variables
  - Saved on the stack
  - Declared within a method
- Recollect single shared context
- Strict memory budgeting



Commands and Events

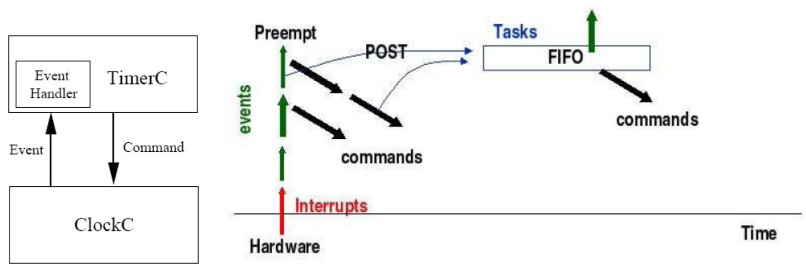
- Commands
  - deposit request parameters into the frame
  - are non-blocking
  - postpone time-consuming work by posting a task
  - can call lower-level commands
- Events
  - can call commands, signal events, post tasks
  - can***not*** be signaled by commands
  - preempt tasks, not vice-versa
  - interrupts trigger the lowest-level events
  - deposit the information into the frame

TinyOS – Thread Model

- Tasks (main thread of execution):
  - Time-flexible
  - Longer background-processing jobs
  - Atomic with respect to other tasks (single-threaded)
  - Preempted by events
- Events (interrupts):
  - Time-critical
  - Shorter duration (hand off to task if need be)
  - Interrupts task
  - Last-in first-out semantics (no priority among events)

Scheduler

- Two-level scheduling: events and tasks
- Scheduler is simple FIFO
  - a task cannot preempt another task
  - events preempt tasks (higher priority)
  - an event may preempt another event
    - ➔ post task to make event smaller and shorter



Outline of This Lecture

- Overview of other platforms
- Hardware
  - Available options
  - Detailed analysis of Mica2 Motes
- Operating systems
  - Catalog of solutions
  - Detailed study of TinyOS
- Programming abstractions
  - Possible paradigms
  - Focus on nesC

Plethora of Possibilities

- Compiled
  - Procedural
    - Vanilla C
  - Component-based
    - nesC
- Interpreted
  - Single node
    - Maté
  - Macro-programming
    - Kairos

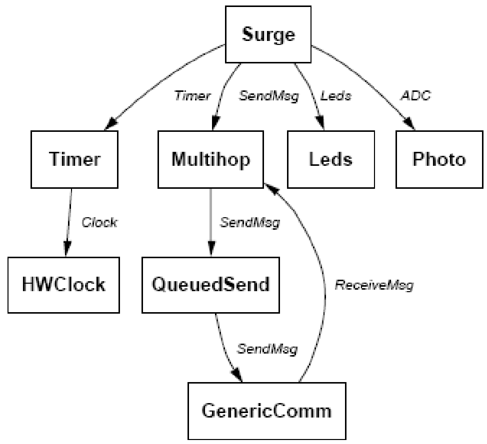
Generally talk to the OS directly

Interface with Virtual Machines (think Java)

nesC

- Networked embedded systems C
- Extension of C
- Fully static (no *malloc* or dynamic dispatch)
  - Call-graph is known at compile-time
- Reflects and supports TinyOS
- Recollect
  - Components
  - Interfaces
    - Commands
    - Events

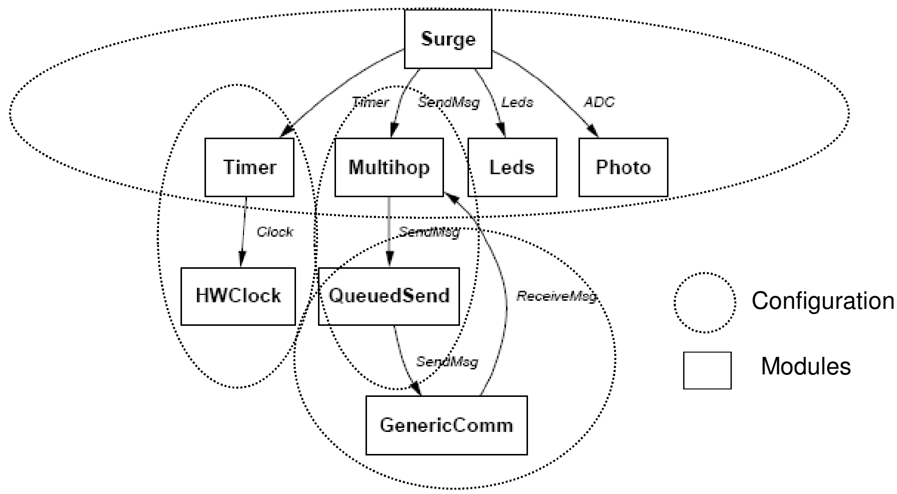
Sample Application



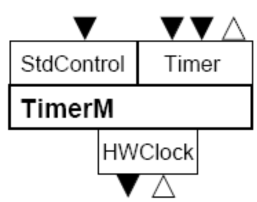
Modules and Configurations

- Modules
  - Similar to class descriptions
  - Define the interface and implementation
  - Usually denoted by files ending with "M"
- Configuration
  - Connects the classes together
  - Provides the wiring between modules
    - Matches the interfaces explicitly
  - Usually denoted by files ending with "C"

Revisiting Sample Application



Sample Module




```
module TimerM {
  provides {
    interface StdControl;
    interface Timer;
  }
  uses interface Clock as Clk;
} ...
```

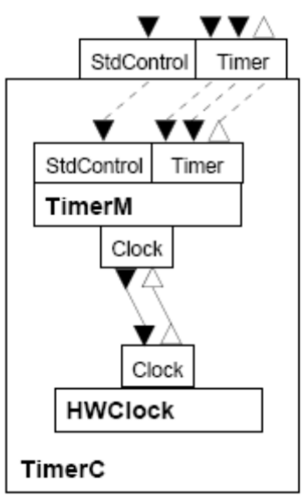
```
interface StdControl {
  command result_t init();
}

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

Module Implementation

```
module SurgeM {
  provides interface StdControl;
  uses interface ADC;
  uses interface Timer;
  uses interface Send;
}
implementation {
  uint16_t sensorReading;

  command result_t StdControl.init() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }

  event result_t Timer.fired() {
    call ADC.getData();
    return SUCCESS;
  }

  event result_t ADC.dataReady(uint16_t data) {
    sensorReading = data;
    ... send message with data in it ...
    return SUCCESS;
  }
  ...
}
```

```
interface ADC {
  command result_t getData();
  event result_t dataReady(uint16_t data);
}
```

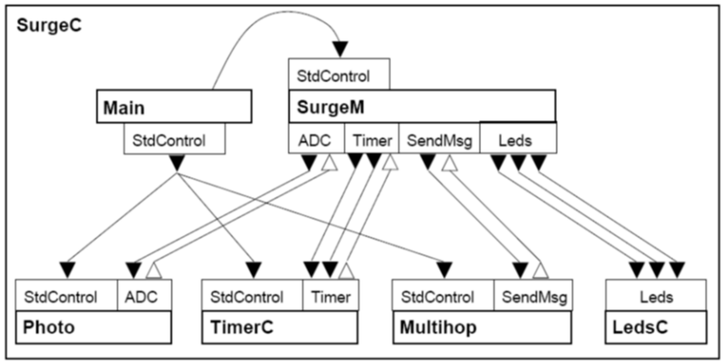

Sample Configuration




```
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }
}
implementation {
  components TimerM, HWClock;

  StdControl = TimerM.StdControl;
  Timer = TimerM.Timer;

  TimerM.Clk -> HWClock.Clock;
}
```

Provides a wiring of modules

Top-Level Application Configuration



Abstract Components

```
abstract module QueuedSend(int maxAttempts) { ... }

configuration Multihop {
  provides interface Send;
}
implementation {
  components MultihopM,
             QueuedSend(10) as newQueue, ... ;

  Send = MultihopM.Send;
  MultihopM.QueuedSendMsg -> newQueue.Send;
  ...
}
```

- Most TinyOS components are single instances
- Abstract components are rarely useful
  - Parameterized module definitions
  - Created at compile time in configurations

Notion of Tasks

```
module SurgeM { ... }
implementation {
  bool busy;
  norace uint16_t sensorReading;

  event result_t Timer.fired() {
    bool localBusy;
    atomic {
      localBusy = busy;
      busy = TRUE;
    }
    if (!localBusy)
      call ADC.getData();
    return SUCCESS;
  }

  task void sendData() { // send sensorReading
    adcPacket.data = sensorReading;
    call Send.send(&adcPacket, sizeof adcPacket.data);
    return SUCCESS;
  }

  event result_t ADC.dataReady(uint16_t data) {
    sensorReading = data;
    post sendData();
    return SUCCESS;
  }
  ...
}
```


Concurrency and Atomicity

- Synchronous code
  - Only reachable from tasks
- Asynchronous code
  - Code reachable from at least one interrupt handler
- Synchronous code is atomic w.r.t. itself
- Atomicity primitives
  - Atomic
    - Uses interrupt enable/disable for atomicity
  - *norace*
    - To explicitly specify that there is no race condition
- Things executing due to hardware interrupts
  - Specified using the async keyword

Hardware Modules as Components

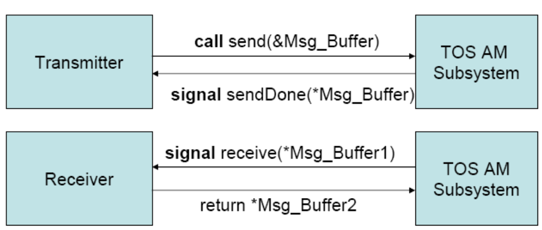
Each sensor has a component that provides one or more ADC interfaces

- MTS400/420:
  - components in <tos>\tos\sensorboards\micawb
- Include in Makefile: SENSORBOARD=micawb

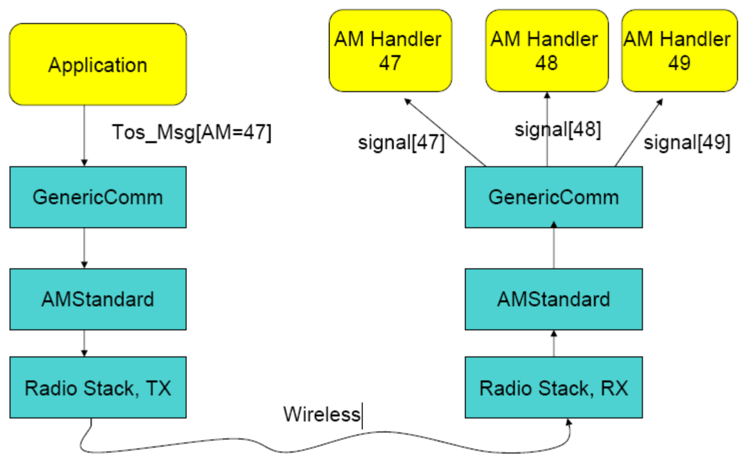
```
includes ADC;
// Define the user names for the ports
includes sensorboard;
interface ADC {
   async command result_t getData();
   async command result_t getContinuousData();
   async event result_t dataReady(uint16_t data);
}
```


Active Messages

- Used to communicate data across nodes
- Contains a HANDLER_ID field
- Receiver has “event” wired to HANDLER_ID
- Different nodes have different “event” handlers
- Solution to dynamic dispatching



Active Messages Illustrated



Summary

- Hardware Platforms
  - Firefly, Mica Z, TeleOS, IRIS, IMote, Jennic, cc2431,….
- Operating systems
  - Nano-RK, TinyOS, Mantis, Contiki, SOS,…..
- Programming paradigms
  - Vanilla C, nesC, mate, kairos,…..
- Coming soon
  - Detailed lecture on sensor operating systems
  - Discussion of possible programming abstractions

References

- http://www.lclark.edu/~jmache/jm/worldcomp06.pdf
- http://www.tinyos.net/papers/tos.pdf
- http://nescc.sourceforge.net/papers/nesc-pldi-2003.pdf
- http://www.tinyos.net/papers/lctes05.pdf
- http://portal.acm.org/citation.cfm?id=1160178
- http://retos.yonsei.ac.kr
- https://projects.nesl.ucla.edu/public/sos-2x/doc/
- http://portal.acm.org/citation.cfm?id=635508.605407
- http://www.springerlink.com/content/4669lu8eclg9u9vy/