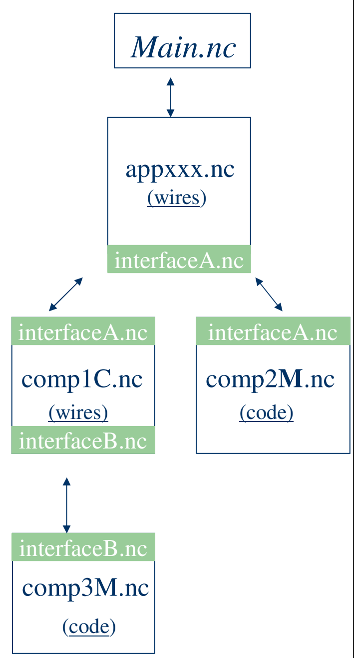
Programming TinyOS

Lesson 3

Some of the content from these slides were adapted from the Crossbow Tutorials and from the TinyOS website from Mobsys Tutorials

Basic Structure

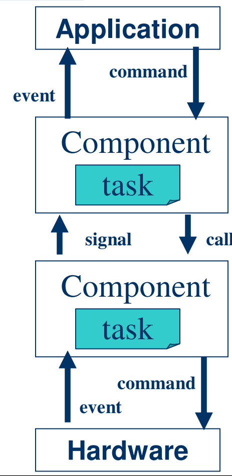
- **Interfaces** (xxx.nc)
  - Specifies functionality to outside world
  - what commands can be called
  - what events need handling
- Software Components
  - **Module** (xxxM.nc)
    - Code implementation
    - Code for **Interface** functions
  - **Configuration** (xxxC.nc)
    - Linking/wiring of components
    - When top level app, drop C from filename xxx.nc



Creating a Module

- Key is defining your interfaces
- Can use "as" to name interface

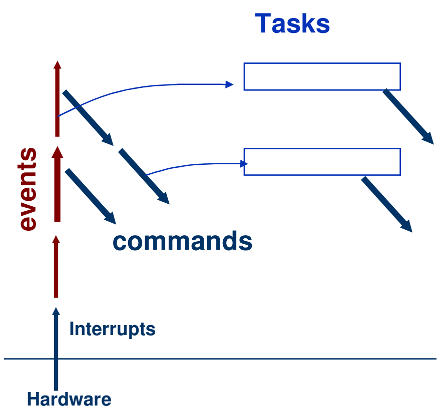
```
includes SkyRFIDdef;
module SkyRFIDM {
  provides {
    interface StdControl as Control;
  } uses {
    interface Leds;
    interface Timer as UARTSender;
    interface Timer as WriteBufferUpdate;
    interface ReceiveMsg as ReceiveWriteRFM;
    interface SendMsg as SendReadRFM;
    interface ReceiveMsg as ReceiveReadRFM;
    }
}

implementation {
  TOS_Msg sendBuffer;
  TOS_MsgPtr sendMsg;
  bool nextState;
```



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

Execution Flow



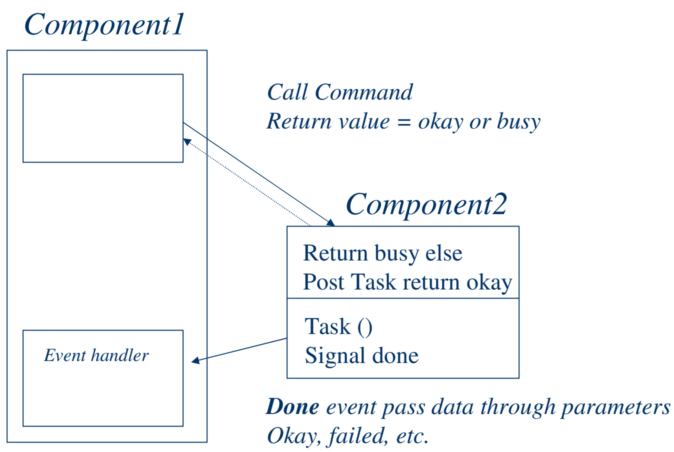
- **Events generated by interrupts preempt tasks**
- **Tasks do not preempt tasks**

FSM

- Finite State Machine Programming Style
  - Event-driven structure throughout application
- All operations are non-blocking
- Tasks extend processing outside event window
- Split Phase Operation (next slide)

> Need logical concurrency at many levels
> Meet hard timing constraints (e.g. radio)

Split Phase Operations



Concurrency Model

- Asynchronous Code (AC)
  - Any code that is reachable from an interrupt handler
- Synchronous Code (SC)
  - Any code that is ONLY reachable from a task
  - Boot sequence
- Potential race conditions
  - Asynchronous Code and Synchronous Code
  - Asynchronous Code and Asynchronous Code
  - Non-preemption eliminates data races among tasks
- nesC reports potential data races to the programmer at compile time (new with version 1.1)
- Use “atomic” statement when needed
- “Async” keyword is used to declare asynchronous code

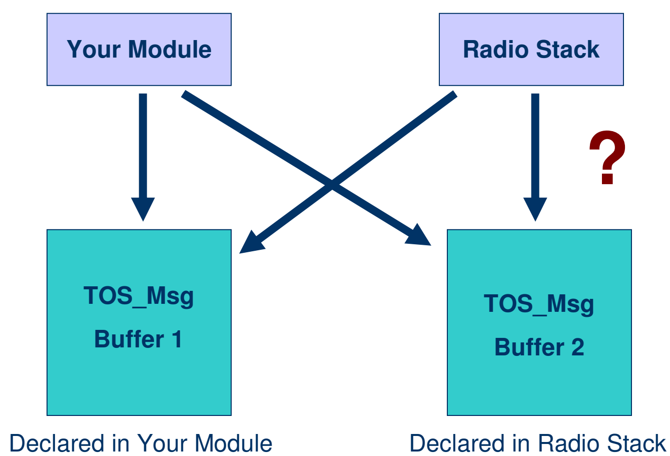
TinyOS Active Messages

- Sending
  - Declare buffer storage in a frame
  - Request Transmission
  - Handle Completion signal
  - **Buffer Management:**
    - After done event can reuse buffer

call SendMsg.send(TOS_BCAST_ADDR, 14, &data)

- Receiving
  - Declare a handler to perform action on message event
  - Active message automatically dispatched to associated handler
    - Known format
    - No run-time parsing
  - **Buffer Management:**
    - Must return free buffer to the system for the next packet reception
    - Typically the incoming buffer if processing complete

Receive Buffers



TinyOS uses Pointers

Remember TinyOS acts like a Finite State Machine

```
event result_t Timer.fired()
  {
   TOS_Msg buffer;
   buffer.data[0] = 1;

   call SendMsg.send(TOS_BCAST_ADDR, 1, &buffer);

    return SUCCESS;
  }
```

What is wrong?

File Locations

- Distribution broken into
  - apps: top-level applications
  - lib: shared application components
  - system: hardware independent system components
  - platform: hardware dependent system components
    - includes HPLs and hardware.h

Platform Folder

- Location of details of the Hardware Layer
  - Most files have the HPL prefix
- Each type of platform has its own subfolder where platform specific files are pulled from.

  (e.g. HPLUARTM, CC1000RadioC, HPLADCM)
- '.platform' file in platform directory
  - lists common platforms
  - allows compiler to pull from those platform directories second.
- 'hardware.h' is where the pins are mapped
- 'avrhardware.h" is where the macro's are defined

Pin Assignments

- Macros used to declare pins
  - `TOSH_ASSIGN_PIN(RED_LED, A, 2);`
- This gives a set of macro's that can be called
  - `TOSH_SET_RED_LED_PIN()`
  - `TOSH_CLR_RED_LED_PIN()`
  - `TOSH_MAKE_RED_LED_OUTPUT()`
  - `TOSH_MAKE_RED_LED_INPUT()`

Questions

- Open Discussion