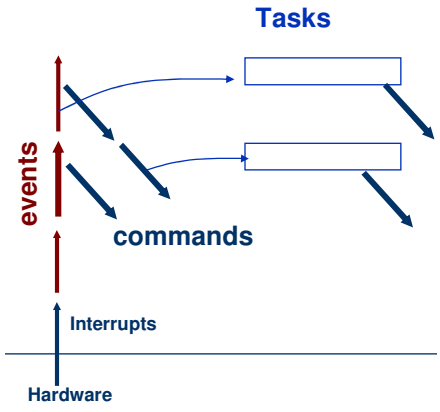
Programming TinyOS

Lesson 2

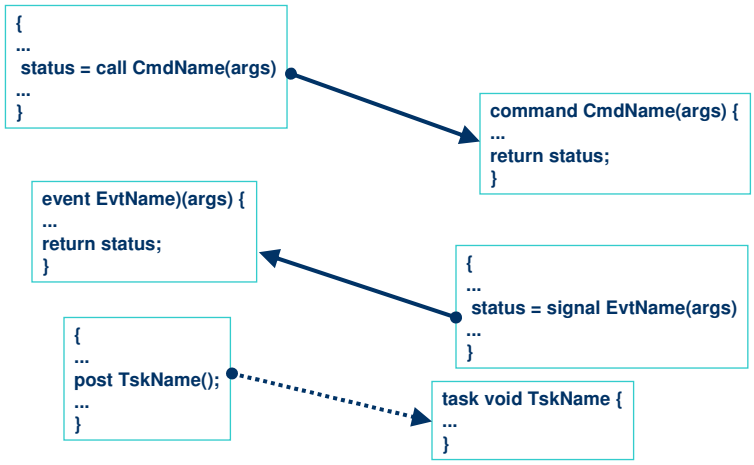
Some of the content from these slides were adapted from the Crossbow Tutorials and from the TinyOS website from Mobsys Tutorials

Execution Flow

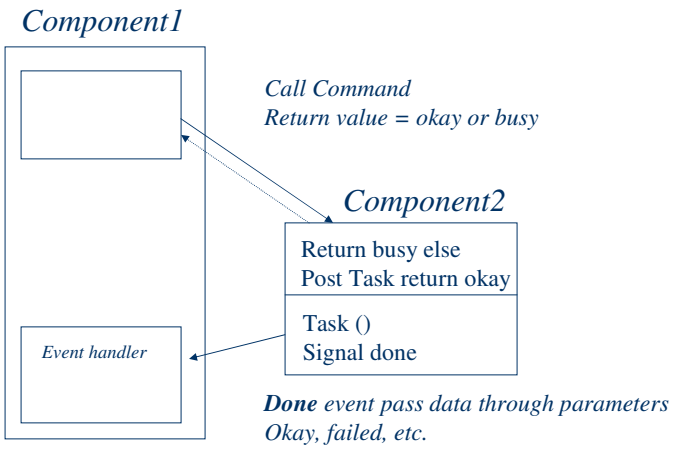


- Events generated by interrupts preempt tasks
- Tasks do not preempt tasks

Commands, Events & Tasks

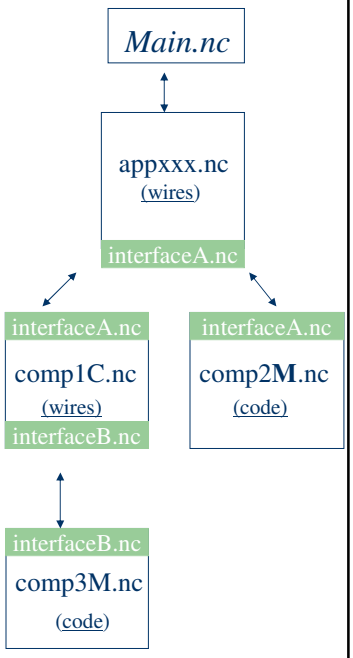


Split Phase Operations



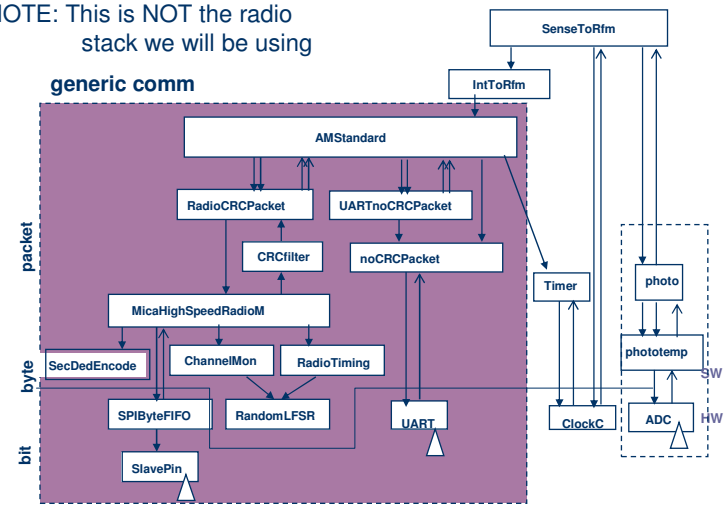
Basic Structure

- **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



The Complete Application

NOTE: This is NOT the radio stack we will be using



Interfaces

- A component specifies a set of *interfaces* by which it is connected to other components
  - provides a set of interfaces to others
  - uses a set of interfaces provided by others
- Interfaces are bi-directional
  - include commands and events
- Interface methods are the external namespace of the component

Interfaces can Fan-out

- nesC wiring allows interfaces to “Fan-out”
- A single “provides” can be wired to more than one “uses” and vice versa.
- Provide a combine function to handle result

```
implementation {
  components Main, Counter, IntToLeds, TimerC;

  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> Counter.StdControl;
  Main.StdControl -> TimerC.StdControl;
```

```
result_t ok1, ok2, ok3;
....
ok1 = call UARTControl.init();
ok2 = call RadioControl.init();
ok3 = call Leds.init();
....
return rcombine3(ok1, ok2, ok3);
```

Exercise

- Which of the following goes inside the module you are implementing if we assume you are the "user" of the interface? (e.g. using the radio stack)

  NOTE: Not all of these choices are exposed through an interface. Assume those that are not exposed are implemented in your module.

  - post TaskA();
  - call CommandB(args);
  - signal EventC(args);
  - TaskA implementation
  - CommandB implementation
  - EventC implementation

Naming Multiple Instances

- A component can have multiple instances of an interface by giving them different names

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

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

Parameterized Interface

- A parameterized interface allows a component to provide *multiple instances* of an interface that are parameterized by a value

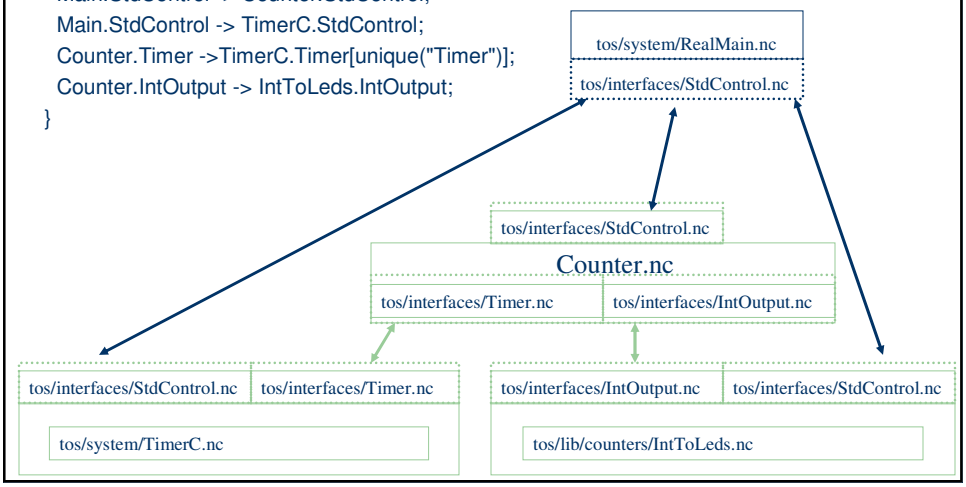
```
provides interface Timer[uint8_t id];
```

- By wiring Timer to a separate instance of the Timer interface provided by TimerC, each component can effectively get its own "private" timer.
- Use a compile-time constant function unique() to ensure your index will be unique.

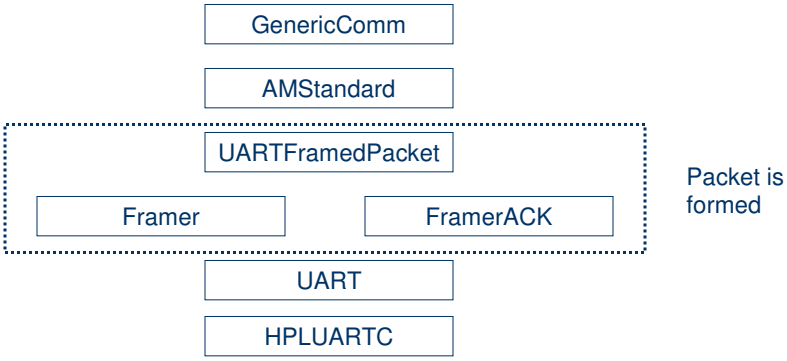
```
SenseM.Timer -> TimerC.Timer[unique("Timer")];
```

```
configuration CntToLeds {
}
implementation {
  components Main, Counter, IntToLeds, TimerC;

  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> Counter.StdControl;
  Main.StdControl -> TimerC.StdControl;
  Counter.Timer ->TimerC.Timer[unique("Timer")];
  Counter.IntOutput -> IntToLeds.IntOutput;
}
```



Exercise



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

Naming Convention

- Use mixed case with the first letter of word capitalized
- Interfaces (Xxxx.nc)
- Components
  - Configuration (XxxC.nc)
  - Module (XxxxM.nc)
- Application – Top level Component (Xxxx.nc)
- Commands, Events, & Tasks
  - First letter lowercase
  - If a command/event pair form a split-phase operation, Event should be the command with the suffix “Done” or “Complete”
  - Prefix commands with “TOSH_” that touch hardware
- Variables – First letter lowercase
- Constants – All Caps

Debugging

- Cover in more detail in later lectures
- Applications can be built to run on the PC (TOSSIM)
  - Good to debug
  - Does not perfectly simulate the hardware
- Toggle LED:
  - Can only get so much information from 1 LED.
  - Very useful to indicate:
    - Radio packet transmit/receive.
    - Timer fired.
    - Sensor activation.