CSE 466 Exam II 15 Dec 2004

1. TinyOS Terms (20 points)

Define the following terms related to features of the TinyOS operating system and provide a basic definition as well as an example of when each feature would be used. Relate your examples to the Flock project, if possible.

a) "Provide" an interface (5 points – 3 for definition, 2 for example)

The module providing an interface implements the commands and issues the events of the interface. An example is GenericComm that provides a SendMsg interface that implements a send command and signals SendDone.

b) "Use" an interface (5 points – 3 for definition, 2 for example)

The module that uses an interface issues commands and implements the handlers for the events signaled on the interface. Your principal flock module used the GenericComm interface to issue a Send command and handles the SendDone event coming back.

c) Hardware interrupt (5 points – 3 for definition, 2 for example)

A hardware interrupt is an event at the lowest level of the TinyOS abstraction layers. For example, a timer, UART, or ADC event. It is translated by TinyOS's interrupt handlers into events at higher levels. An example is the ADC conversion of the photosensor's current value to determine if a bird was "startled".

d) Task (5 points – 3 for definition, 2 for example)

Tasks are long(er)-running computations that are queued and run in FIFO order and to completion. They can be pre-empted by interrupts and event handling but not by other tasks. You used tasks for sound generation, updating the weights of different bird calls, and determining the next song to play.

2. TinyOS Abstractions (30 points)

GenericComm uses a TinyOS parameterized interfaces for messages that are received.

a) Describe how this feature relates to the Active Message model used for TinyOS radio packets.

   Each packet has an Active Message type (AM_TYPE) that is used to route it to the handler for those types of packets. By parameterizing an interface with Active Message types, we get the nesC compiler to generate the routing logic between GenericComm and the right module to handle that packet type.

b) What if we didn't have parameterized interfaces? Every module wanting to use GenericComm would have to connect to the same interface. What would have to change to make this work? Describe any changes that would need to be made at the interface, in GenericComm, and in modules using the interface.

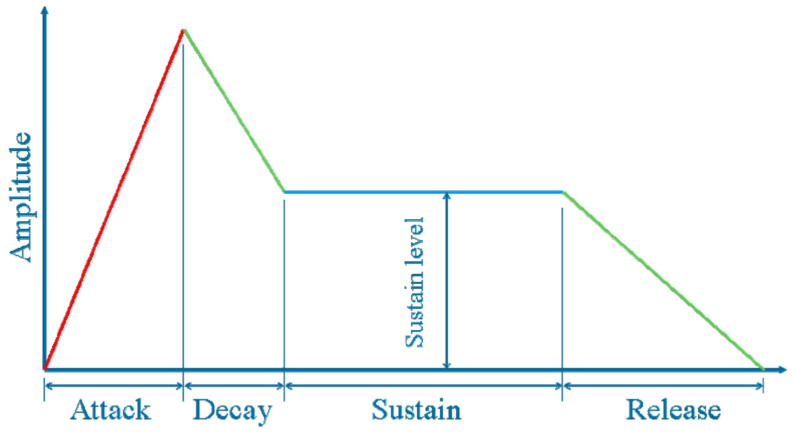
   Nothing would change in GenericComm. On receiving a packet, each module would receive every packet type and would have to check whether it was the type is could handle. When sending a packet, the module would have to ensure the Active Message packet type was part of the packet buffer. The interface would have to change so that SendDone events carried the message type so that a single SendDone event could be associated with the right packet.

c) Are there any performance implications to not having parameterized interfaces?

   Yes, since every module would receive an event for every packet received, there would be many more functions called for each packet. Most of these would simply ignore the packet. nesC optimizes this through the parameterized interface so that the filtering is done only once.

3. Sound Generation (30 points)

Describe a strategy for generating a sound waveform with an ADSR (attack-decay-sustain-release) envelope as shown below.



a) Start by describing how an audio frequency is generated. How is the frequency varied?

Sound is generated through a PWM signal controlling the speaker. Frequency is varied by varying the period of this PWM signal.

b) In the flock project, we did not vary the amplitude of our sound. How would you go about varying the amplitude using the same scheme for sound generation?

Although this may not be possible with our cheap piezo speakers, amplitude can be, in general, varied by changing the duty-cycle of the waveform driving the speaker.

c) Describe the timers you'd need, what they would be used for, and how their events would be handled.

We would need timers to generate the PWM waveform and a timer to control a finite-state machine that would take us through the ADSR curve above. This state machine would have five states: attack, decay, sustain, release, and idle. In the attack state, whenever the timer fires, we would increase the amplitude of the signal (increase the duty-cycle of the PWM waveform) at a specified rate. Similarly, in decay and release we would decrease the amplitude. In sustain, we would leave it unchanged. Each state of the state machine would have to know the rate at which to change the duty-cycle and for how long (duration of that state).

4. Radio Protocols (20 points)

One of the principal issues in saving radio power is deciding when to power-up the radio so that it can hear other packets and when to put it to sleep without missing communications. This is a synchronization problem. One way to deal with it is the method you used in constructing your flock node. Basically, you made sure to keep the radio operating for a long enough period of time that you would hear most packets from neighbors (that had just finished their bird call).

We've devised a new method. It uses a microphone to detect a special note at the end of every bird call. We'll have each song end with an inaudible note in the ultrasound range (25KHz). We'll add a microphone to our birds so that they can listen to nearby songs. If they hear sound at that high frequency, they'll then know to listen for a radio packet as soon as it ends.

Discuss the following aspects of this approach:

a) At what frequency must the ultrasound be sampled?

*We must sample faster than the Nyquist rate to be able to accurately reconstruct the waveform. This implies a sample rate greater than 2*25KHz or at least 50KHz.*

b) Assuming that our microcontroller runs at 10MHz and that it takes X instructions per *millisecond* to handle the radio stack and Y instructions for each sample of ultrasound, can we handle simultaneous radio transmission and ultrasound detection on this platform? Derive an inequality that relates X and Y assuming the sample rate for the ultrasound is as your answer for part (a).

There are 10,000,000 possible instructions in one second. X instructions are executed 1,000 times per second. Y instructions are executed 50,000 times per second. This leads to the following constraint equation:

$$10{,}000{,}000 > 1{,}000X + 50{,}000Y \qquad \text{or} \qquad 10K > X + 50Y$$

c) Typical multiple access networks use exponential backoff when a collision is detected after two or more devices try to transmit at the same time. What could we do to exploit the ultrasound output at the end of every bird call to minimize the probability of this initial collision?

By detecting the ultrasound at the end of a song, we would know that another bird was going to want to send a radio packet soon. If the module hearing the sound also wanted to send a radio packet, then it could wait a random amount of time before trying. Basically, this allows us to move the exponential backoff earlier and avoid more of the collisions that could occur when first sending a packet.