CSE 466 Exam II 14 March 2008

1. Definitions (30 points)

Define the following terms related to features of the iMote2 and the Linux operating system and provide a basic definition as well as an example of when each feature was used in your lab assignments.

a) Modules and device abstractions

Modules are additions to the operating system which have a standard interface, and allow the system to write data to a device as if it was a file. They can be added to the kernel at run-time eliminating the need to compile all device drivers into the kernel ahead of time. You wrote a module for blinking LED and used one to implement the AirStick.

b) Packet buffers

Packet buffers are areas in memory used to hold incoming and outgoing packets. They allow a program sending packets to go ahead while send buffers are being emptied by the radio device driver. Similarly, incoming packets can be held in packet buffers until a program is ready to deal with them. You used packet buffers in all your radio communications.

c) Describe an interaction you had between device modules and why you think it occurred.

A common interaction occurred when using timers. If more than one module uses a timer, then they can set/reset it when the other is unaware and cause improper operation in the interrupt handlers. This may have occurred when you wrote your blink module using the same timer used by the accelerometer device driver.

d) IRQ return values

IRQs return values that indicate whether they did or did not handle an interrupt so that the interrupt handler can either continue to call other handling functions or exit. In your AirStick and blink modules you had to check to make sure that the interrupt was actually one that you wanted to handle in a particular interrupt handler.

e) I^2C (TWI) read/write register commands

You used the TWI interface to read and write values to the AirStick hardware (interacting with the ATmega microcontroller on that board) and thereby control its transmitting and sampling. The interface supports the use of registers in a slave device which can be read and written by a master device. This allows data to be communicated in both directions.

f) Byte/word alignment in the packet format

When transferring information, it is important to select the correct endianness so that the correct number is reconstructed. Little endian notation selects the lowest order byte and sends that first, big endian numbers are sent with the most significant byte first. Within a byte you can similarly have little and big-endianness in the bit order. You had to deal with this in your code for the tosmac when you were sending numbers larger than 8-bits, you had to read them with the same endianness that you were sending them. You also had to make sure that 8-bit and 16-bit values were properly distinguished.

2. Soccer Enhancement (30 points)

There are many ways to enhance the soccer game that was done for the final project. Consider the enhancements below and discuss how you would go about implementing them, what parameters might be needed to be determined, and what problems do you foresee if packets are dropped.

a) Specialized players. A special goalie player on each team would be constrained to stay near the goal but would have the power to neutralize a merged player up to a size of 4. Special forward players would be able to move twice as fast as a goalie.

One way to go about this is to split the processing between the main computer and the controller. If static player types were desired, then at the beginning of the game (or half, or whatever) each player would select a player type that they wanted and send that request to the game controller until they got an acknowledgement of their request.

During game play, a player like the goalie wouldn't need anything different, the "walls" would just need to be constrained by the game master to be the goalie box. The forwards could normalize their data to 40, instead of 20 allowing them to move faster (and allowing them to contribute more to a group's velocity). Merging would be an interesting problem for both cases, you need to be able to add the velocities of the forwards in a meaningful way, and you need to probably always have the goalie be captain (he is effectively a group of 4…that moves slowly). Dropping packets wouldn't be any bigger deal than it is now, as long as you had acknowledgement for the initial setup.

Another approach is to handle this entirely in the game master by applying appropriate multiplication factors to each player speed and neutralizing ability. Then there would be no differences in how lost packets would affect the game.

b) Field worm holes. Worm holes on the field would allow a player to move rapidly from one end of the field to the other. Of course, the entrance and exits of the wormhole would also be visible to the other team.

Here, almost all the code for the wormholes would be in the game master. By adding another action bit, it would be easy to allow the game controllers to react (and indicate that they had been teleported). The potential problem with lost packets is if the master loses packets when a player is trying to avoid the wormhole. It would be a much larger error than missing just a regular movement packet when the player is far from a wormhole because the teleporting takes the player to a far side of the field.

An important consideration is some hysteresis in entering/exiting a wormhole so that a player doesn't constantly go back and forth (in and out never being able to escape). We would want to add some delay before a player can re-traverse a wormhole in the opposite direction to allow time for it to move away from the exit.

Parameters that would need to be determined include the radius of action of the wormhole, and if there would be any way to "shut down" a wormhole by having an opposing player hang out at the exit of the wormhole.

3. AirStick (40 points)

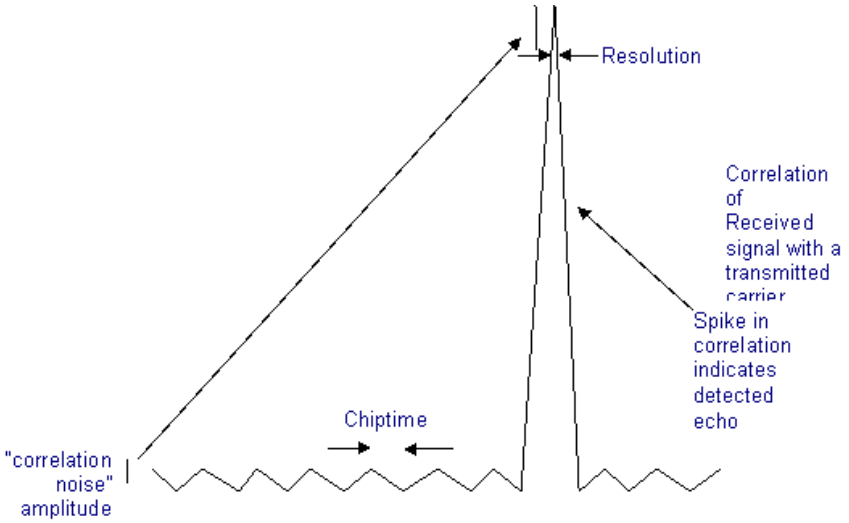
Light takes about 1ns (10^{-9} sec) to travel one foot. The waveforms we generated in our electric field sensing implementations had periods or chip-times on the order of 0.1 ms (corresponding to 10kHz frequency). Because the propagation time of the electric field was very small, the received signal was essentially synchronous with the transmitted signal: there was no noticeable phase shift or time delay between the received and transmitted signals.

Another common use of Pseudo-Noise signals (like those we generated with the LFSR) is to measure time delays in situations where time delays are meaningful. Examples include GPS, sonar range-finding, and laser range-finding. In these applications, Pseudo-Noise signals have a very clear advantage over periodic signals: the autocorrelation of a periodic signal has periodic maxima, so a peak in the correlation could indicate a reflection at many different distances. The autocorrelation of an LFSR signal has just one maximum, which identifies a reflection unambiguously.

Sound takes about 1ms (10^{-3} sec) to travel one foot. Suppose you are given a robot with a "dual" sonar rangefinder consisting of 2 ultrasound transmitters and 1 ultrasound receiver. Imagine that the transmitters are pointed in 2 slightly different directions (slightly to the left and right of the robot's heading, say). In this problem you will write code for CDMA spread-spectrum sonar range-finding. Assume you are given a 255-element array called LFSR with the values of a 255-element Maximum Length LFSR filled in (you can assume that the entries in the array are +1 or -1…if you assume differently, just specify your assumption). You can generate all the necessary LFSR sequences (for both TX and RCV) by indexing into this array.

Ignore attenuation effects. Just assume that you get equally strong echoes no matter what the distance. Also, assume that the ADC read operation is the only operation that takes any time.

a) Write code that generates the appropriate TX signals, collects (and buffers) the receive samples, does rolling correlations to detect echoes, and at the end returns 2 time delays, for the largest echo from each of the 2 TX signals. This code will be very different from your previous correlation code. To make the coding simpler, just transmit the entire Pseudo-Noise burst first, before starting to listen. When listening, you will have to maintain a buffer containing the last 255 samples you have seen. You will need to correlate this vector of samples with each of the 2 LFSR-generated carrier vectors. When there is a maximum in the correlation then that corresponds to a detected echo. Do not concern yourself with multiple echoes. Just return the time of the largest detected echo. So, for each of the 2 TXs, you will need one variable for the maximum correlation value seen on that channel so far, and one variable for the time at which the maximum correlation on that channel happened. Skeleton code is provided on the next page. Fill in the requested areas. Assume that main starts by calling tx(); and then correlate().



```
// Given: lfsr[i] for 0 <= i <= 254 is a ML LFSR sequence
// values of lfsr[i] are either +1 or -1;

int inc_headortail(int ht) { // increments & wraps head or tail pointer
  ht++;
  if (ht>254) {
    ht= 0;
  }
  return(ht);
}

int gai(int index, int h) { // gai == get_abs_index
  int abs_index;
  abs_index = (h+index) % 255;
  return (abs_index);
}

int lf1_h = 0;       // lfsr 1 head
int lf1_t = 254;     // lfsr 1 tail
int lf2_h = 10;      // Choosing lfsr 2 to be offset from 1 by 10
int lf2_t = gai(10, 254); // (10+254) % 255;

void tx() {
int i;
  for(i=0; i<255; i++) {
    output_PIN1(lfsr(gai(i,lf1_h)));
    output_PIN2(lfsr(gai(i,lf2_h)));
    READ_ADC; // Just for timing
  }
}

void correlate() {
int lf1_corr = 0;     // current correlation
int lf1_max_corr = 0; // max observed correlation
int lf1_peak_time = 0;// time of max observed correlation

int lf2_corr = 0;
int lf2_max_corr = 0;
int lf2_peak_time = 0;

int buf_h = 0;   // buf_h points to oldest sample
int buf_t = 254; // buf_t points to newest sample

int buf[255];
int i;
int t;
int T_MAX;

  // First fill buffer
  for (i=0; i<255; i++) {
    buf[i] = READ_ADC;
  }

  T_MAX = 1000; // How long are we going to wait for an echo?
```

```
  // Now start rolling correlation
  for (t=0; t<T_MAX; t++) {
    lf1_corr = 0;
    lf2_corr = 0;
    for (i=0; i<255; i++) {
      lf1_corr += lfsr[gai(i,lf1_h)] * buf[gai(i,buf_h)];
      lf2_corr += lfsr[gai(i,lf2_h)] * buf[gai(i,buf_h)];
    }
    if (lf1_corr > lf1_max_corr) {
      lf1_max_corr = lf1_corr;
      lf1_peak_time = t;
    }
    if (lf2_corr > lf2_max_corr) {
      lf2_max_corr = lf2_corr;
      lf2_peak_time = t;
    }
    buf_h = inc_headortail(buf_h); // increment head, new oldest sample
    buf_t = inc_headortail(buf_t); // increment tail, this has now
                                   // wrapped around and is pointing at
                                   // oldest sample instead of newest
    buf[buf_t] = READ_ADC; // replace stale oldest sample (tail) with a
                           // new sample, so that tail is again
                           // pointing to the newest sample
    lf1_corr = 0; // reset correlations back to 0
    lf2_corr = 0; // reset correlations back to 0
  } // t

  printf ("Chan 1: echo strength %d at time
                                     %d",lf1_max_corr,lf1_peak_time);
  printf ("Chan 2: echo strength %d at time
                                     %d",lf2_max_corr,lf2_peak_time);
}
```

b) Since you are transmitting first, and then listening, what is the closest object you can properly detect (by properly, we mean that you can hear the whole Pseudo-Noise sequence from). Assume 1 ft / ms speed of sound, length 255 LFSR, and chip time 0.1ms. What if the chip time was 0.01ms? What could you do to your code to give a reasonable minimum distance even with a 0.1ms chip time?

*Our minimum time is 255*chip-time. For the sound to make a round trip at 1ft/ms, that would mean that:*
255(.1ms)*(1ft/ms) = 2*minimum-distance*

Solving for minimum-distance yields approximately 13ft.
If the chip time was decreased to .01ms then the minimum-distance would be 1.3ft.

To improve the minimum-distance while keeping the chip-time at .1ms, we could decrease the length of the code from 255 down to something on the order of 63 or 31.

Another option is to start listening to the return signal right away rather than after 255 have been send out. That would be more sensitive to noise as we wouldn't have the benefit of the entire code.

c) What is the maximum range your sensor will detect an echo (this is determined by how long your code is willing to wait for an echo)?

Determining the maximum distance depends on the length of the timeout for the return signal. This would correspond to:
T_MAX + 255)(.1ms)*(1ft/ms) = 2*maximum-distance*

Solving for maximum-distance assuming a T_MAX equal to 1000 yields 63ft.