Each set of partners will implement their own version of a "bird" using the specifications listed in this document. Your compiled code will run on both you and your partner's "birds" for the Flock demonstration **during the final exam time, 8:30 AM, on Wednesday, June 7th in the atrium**. Remember you need to qualify your "bird" before it can participate in the Flock demonstration. If for some reason you are unable to qualify your "bird", an alternative "bird" program will be provided so that you may receive your participation points for the demonstration.

Hints:

Do NOT use dynamic memory allocations (such as malloc) on the ATmega.

You should use enum types for default values and constants to simplify future modifications.

Try to minimize the number of divides and mod used in your code. Calculate the needed values only once and store the results. A memory read is a lot faster than a divide and/or mod.

NOTE: Not all of the bird implementations need to be the same. The implementations must only meet the specifications outlined. Differences in bird behavior will not be penalized as long as the behavior is within the specifications contained in this document. In fact, it is encouraged that each group's implementation be slightly different as long as they meet the specifications.

Implementation Details:

The Atrium Group ID will be 0x7A. Each mote will use the same group ID, and Node ID (TOS_LOCAL_ADDRESS) will be the number on your mote's label. Assume the label is a hexadecimal number. (e.g. 11, A1, A5, 55)

Send all your packets to the TOS_BCAST_ADDR.

The first item in the data/payload section of a flock message is the address of the sender. There are two designators for the sender address in the message specifications: "Node0" and "TransmittingNodeNum". The "Node0" designator is used to signify that a packet should only be processed if it was sent from the root node (address == 0). The "TransmittingNodeNum" designator signifies that packets should be accepted from any source.

In a 'SangSong' message the second item in the data/payload section is simply the sequence number of the packet that is being sent. Start the "SequenceNum" at 1 and increment each time you send this type of message.

TOS_Msg.strength is automatically updated with the packet strength during reception and is part of the TOS message struct.

For the purposes of the final project assume that Timer.start() takes milliseconds. If we give you a time of 1 second assume if you set a timer to fire in 1000ms it will fire 1 second later. For the flock to be successful we all must use the same assumptions.

The variable "`transmitPower`" in adjust globals packet should update the transmit power of your transmitter. You can set the transmission power level using the function CC1000Control.SetRFPower(int) in component CC1000ControlM.

The following commands will be useful for generating random numbers:

/* Initialize the seed from the ID of the node */
async command result_t Random.init()

/* Return the next 16 bit random number */
async command uint16_t Random.rand()

When filling the message packets use the same "endian" as the ATmega.

Flock Algorithm:

Within a single "bird", the flock algorithm that you will implement is as follows:

A)	WAIT STATE (Silent while waiting) Wait to receive a packet of type *AdjustGlobals* When entering WAIT STATE, turn Tri-Color LED off and turn on the Red LED located on the corner of SoundBoard1. Tri-Color LED may be adjusted when in WAIT STATE by a command packet. If you receive a StopNWait packet when you are already in the WAIT STATE you should treat it as you just re-entered wait state. (Basically reset back to known state) Ignore SangSong Packets Stop any active timers IF(*AdjustGlobals){* Turn off the Red LED located on the corner of SoundBoard1 *Go to CLEAR STATE* *}* IF(*Command Packet*) { Perform command. Change LED and possibly sing a song. Stay in WAIT STATE, unless health status changes to DEAD }
B)	CLEAR STATE -- **Enter this state on reset** clear all historical data – call the reset function on the SongStatistics module clear all counter variables Pick a number between 0 and 99. If that number is less than chanceBornInfected, pick an infection

	length between minInfectionLength and maxInfectionLength and set the health state to INFECTED and the tri-color LED to RED. Set the mutationCount to a random number between 0 and 255, and set mutationAmount to 3. Otherwise, set the health state to HEALTHY, and the tri-color LED to GREEN. Go to LISTEN STATE
C)	SING STATE Choose a song using the algorithm listed below Send the song to the Yamaha chip If your health state is HEALTHY or IMMUNE, for every closest bird (there may be more than one if they all have the maximum RSSI), if the bird is infected (and its virus "signature" is not within your immunity range if IMMUNE), compute the chance that you'll become infected. Do this by generating a random number between 0 and 999 and comparing it with the infectiousness global times 100 - ISS, the immune system strength. If the randomly-generated number is less than the infectiousness global times ISS, compute an infection length between minInfectionLength and maxInfectionLength, set a timer for that length of time, and set your health state to INFECTED. Change the tri-color LED to RED. Adjust the Immune System Strength as detailed in the Bird Flu section. Change the Yamaha's volume so that it scales with ISS, the Immune System Strength. An ISS of 50 uses the default volume. For every 10 point change in ISS, increase or decrease both volume registers by 1. While singing your bird should continue to listen and collect information about what the neighbors are singing. After finished singing send a "SangSong" message. Go to LISTEN STATE IF(*Command Packet*) { Perform command. Change LED, possibly sing a song, and possibly change health states. Go to LISTEN STATE (or DEAD STATE if new health state is DEAD) }
D)	LISTEN STATE Set listen timer for random time between [minListen, maxListen] milliseconds Listen and collect information about what the neighbors are singing When listen timer goes off go to SING STATE IF(*Command Packet*) { Perform command. Change LED, possibly sing a song, and possibly change health states. Re-enter LISTEN STATE (or DEAD STATE if new health state is DEAD) }
E)	STARTLED STATE Send startled bird song to Yamaha chip. While singing your bird should continue to listen and collect information about what the neighbors

	are singing. Maintain the same color of the Tri-Color LED and turn on the Red LED located on the corner of SoundBoard1. Send Startled packet (remember to decrement HopCount) when finished singing the startled song. After finished singing, delay 10 seconds, turn off Red LED in the corner of SoundBoard1. IF(*Command Packet*) { Perform command. Change LED, possibly sing a song, and possibly change health states. Go to LISTEN STATE (or DEAD STATE if new health state is DEAD) } Go to LISTEN STATE
F)	DEAD STATE If any song is currently playing, stop it, and play the startle song. Do not send a startle packet! Randomly pick a value between minRebirthDelay and maxRebirthDelay, and wait for that amount of time. When that amount of time has elapsed, randomly pick one of the closest two birds, based on signal strength. Copy the health state from that bird over to yours, including the immunity range. You should select the song most recently played by the other bird of the two closest birds and go to the SING STATE to sing it. IF(*Command Packet*) { Perform command. Change LED, possibly sing a song, and possibly change health states. Stay in DEAD STATE, unless health state changes }

There are 5 types of Active Messages your program must handle; they are as follows:

AM #	Flock Message / Packet
42	SangSong - The "I sang song" message; a message from some other birdie indicating what song it sang, and its health status. You also send this packet after singing. `uint16_t transmittingNodeNum` local # of node singing `uint16_t sequenceNum` start at 1, increment each time you send this packet `uint16_t songNum` song# that was sung `uint8_t healthState` Health state of the bird `uint8_t immuneSystemStrength` Immune System Strength `uint8_t mutationCount` The “signature” of the virus `uint8_t mutationAmount` How much the virus can mutate in a single propagation `uint8_t minImmuneRange` Minimum virus “signature” that the bird is immune to `uint8_t maxImmuneRange` Maximum virus “signature” that the bird is immune to
50	AdjustGlobals - A message from Node 0 containing global parameters for all birds. **This message should be processed in any state.**

	uint16_t Node0 uint16_t minListen default 2000 millisec. uint16_t maxListen default 15000 millisec. uint16_t Threshold default 600 uint16_t minInfectionLength default 15000 millisec. uint16_t maxInfectionLength default 20000 millisec. uint16_t minImmuneLength default 15000 millisec. uint16_t maxImmuneLength default 60000 millisec. uint16_t minRebirthDelay default 15000 millisec. uint16_t maxRebirthDelay default 45000 millisec. uint16_t minPointValue Default 100 uint8_t Repetition default 3 uint8_t Silence default 10 uint8_t transmitPower default 0x0F uint8_t startledHopCount default 3 uint8_t infectiousness default 70 uint8_t chanceRecovery default 50 uint8_t chanceBornInfected default 10
51	StopNWait - A message from Node 0 telling you to stop and go to the WAIT STATE. **This message should be processed in any state.** uint16_t Node0 On receipt, go to WAIT STATE uint16_t startAddr Beginning of Address Range uint16_t endAddr End of Address Range
52	Command Packet - A message from Node 0 telling you to adjust your light and possibly play a song. **NOTES:** **1) This message should be processed in any state.** **2) When finished processing the command the bird should remain in the same state it was in before the command packet was received (e.g. WAIT, LISTEN, or DEAD).** **3) Refer to Command Packet description below** uint16_t Node0 uint16_t startAddr Beginning of Address Range uint16_t endAddr End of Address Range uint16_t LED LED state uint16_t song Song to Play uint8_t healthState New health state
60	Startled - a message from some other birdie indicating that they have been startled. Stop what you are doing and sing your startled message. After the startled message you should go to the LISTEN_STATE. Do NOT send a SangSong packet for the startle song. Instead send the startled message. Remember to decrement the HopCount when you send the message. On reception, If (HopCount > 0) process message Else ignore message uint16_t transmittingNodeNum local # of startled node uint16_t hopCount Number of hops remaining. If

	uint16_t startledSeqNum — startled by radio message. Decrement value before sending. A randomly generated number that is stored to check to see if you have been startled by this startle packet before
61	Identification – After the light sensor has detected the identification sequence it sends this packet to let the control software know the nodeID that has been identified. – Your node should ignore any packet of type 61. uint16_t transmittingNodeNum local # of indentified node

The Bird Flu:

In addition to the regular state of the bird, each bird has a health state that's either HEALTHY (but can be infected), INFECTED, IMMUNE, or DEAD. The tri-color LED should indicate the state of the bird, and the healthState fields should be set as follows:

State	LED Color	healthState
HEALTHY	Green	1
INFECTED	Red	2
IMMUNE	Blue	3
DEAD	Off	4

When a bird becomes infected, the infection length should be computed by randomly picking a value between minInfectionLength and maxInfectionLength. A timer should be set for the picked length. When the timer fires, a random number between 0 and 99 should be picked. If this number is below the chanceRecovery global, then the health state should be set to IMMUNE, and a timer should be set for a random length of time between minImmuneLength and maxImmuneLength to set the state back to HEALTHY. Otherwise, the health state should be set to DEAD, and the bird should enter the DEAD state.

A variable called ISS tracks the immune system strength, and ranges from 0 to 100 and is initialized to 50. While the bird is HEALTHY or IMMUNE, the ISS should gradually increase. When determining the health state (i.e. right before you sing a song), increase ISS by a random amount between 0 and 3 when HEALTHY or IMMUNE, and decrease ISS by a random amount between 0 and 3 when INFECTED. When then bird is INFECTED, the ISS should gradually decrease.

When you become infected, take the infecting packet's mutationCount and add a random number between –mutationAmount and mutationAmount. Wrap the mutationCount around if the result is below 0 or greater than 255. Use the mutationAmount in the packet. Keep the resulting mutationCount and send it when you sing a song. You should also remember the mutationAmount and propagate that value unchanged.

When you become immune, pick a random number n between 0 and 9. Set you minImmunityRange to mutationCount – n, and your maxImmunityRange to mutationCount + n.

Song Decision Algorithm:

The course staff has provided a "SongStatistics" module that keeps statistical data on the songs sang by the Flock. The module also utilizes a do-not sing list to ensure the flock won't be stuck singing the same song. The overall flock uses random numbers to produce an emergent behavior. You will need to implement the following song selection algorithm:

```
y = rand() % Silence
silenceThreshold = computeSilenceThreshold();

if (y < silenceThreshold)
      Silence… Don't sing a song-- go back to LISTEN STATE.
else {
      sum = the sum of all song point values
      x = rand() % sum;
      runningTotal = 0;
      for all songs {
            runningTotal += song point value;
            if (runningTotal >= x)
                  pick this song and break out of the loop
      }
}
```

When a song is picked, you must inform the SongStatisticsM module so that it can construct the blacklist.

computeSilenceThreshold() should use the light sensor ADC value to achieve the effect of singing less as the ambient light gets darker.

Description of Command Packet:

The command packet is used by the command software to instruct a single or multiple mote(s) to perform a specific action. To determine if the message was intended for your node, check if the node ID falls within the instruction's address range. (i.e. startAddr <= nodeID <= endAddr). If your node does fall within the command address range you must process the LED and sing instruction.

The LED variable contains the color value for the tri-color LED:

bit 0 controls the Blue LED in the TriColor LED
bit 1 controls the Green LED in the TriColor LED
bit 2 controls the Red LED in the TriColor LED

The command packet may also include an instruction to play a song. The song field will either contain the song number to play or 0xffff if no song should be played. If a song is requested [bird songs(0-15) or startle (16)], play that song immediately.

After singing a song triggered by a command packet, your bird should send a SangSong message with only these fields: transmittingNodeNum, sequenceNum, songNum, and healthState. The remaining fields should be filled with zeros. You may send the SangSong message immediately after you finish singing the song. When triggered by a command packet, you should send a SangSong message for all songs 0-16. In particular, **do not** send a startled message for the startled song.

If the healthState field isn't 0, you should update the health state of your bird. If the state changes to INFECTED, assume that the infection period begins when you receive the packet. If the state changes to DEAD, go to the DEAD state. Do not sing the startle song if another song is specified in the command packet. **The tri-color LED should be updated to reflect the new health state and takes precedence over the LED field.** If the state changes to HEALTHY or IMMUNE, and you're in the DEAD state, go to the LISTEN state.

If the infection timer fires while you're singing a command song, delay processing of going to IMMUNE or DEAD until the song is done.

Method to Identify Node:

To easily identify deployed nodes, your program should include two methods of identification: 1) a packet from the control software should cause the mote to identify itself, and 2) the mote should identify itself to the control software. The first method (control software to mote) is accomplished by the control software sending a command packet with the same 'start address' and 'end address', causing the specified mote to move to a uniquely identifiable state. The second method requires the mote to detect that an identification packet should be sent to the control software.

The mote should sense that it is being identified by detecting large light changes that form a pattern. A user will cover and uncover the light sensor 4 times in a row to signal that an identification packet should be sent to the control software. This means that 8 transition events need to be detected in order. After detecting the sequence you should turn on the mica2dot's red LED for 5 seconds to give feedback to the user. A sampling rate of 3 times a second should be fast enough to detect a person covering and uncovering the light sensor. To make sure random shadows do not cause the mote to falsely identify itself a transition event is defined by the 4 most significant bits of the light level changing by more than 4. In addition, your program should also have a timeout period of 2 seconds between transition events to make sure the events being detected are part of a continuous sequence. After a timeout your state machine should reset. Notice that the light sensor is used to both identify the bird and startle the bird; this means that you will startle the bird when trying to perform the identification sequence when not in the WAIT_STATE. To avoid the entire flock startling when someone is trying to identify a bird, do not send the

startle packet until after your algorithm has determined it's not an identification packet. Unlike startle, you should be able to identify the node in any state. NOTE: If, for some reason you program misses an edge with a timeout of 2 seconds, it will most likely pick up the next edge change. Therefore no points will be deducted if it takes 5-6 times to identify the mote as long as the user interface is reasonable.

Description of how the "SongStatistics" Module works:

The course staff has provided a "SongStatistics" module that will help your program pick the next song by keeping track of which songs are being sung, using a point scoring system. Points are assigned based on what the surrounding nodes are singing. The module also provides statistical feedback on your nodes point values.

When not in the WAIT STATE, your program should always be listening on the radio to the surrounding nodes to collect information about what songs the nodes are playing. From the SangSong packet (AM #42 type packets) you should pass the following information to the SongStatistics Module:

```
Uint16_t   TransmittingNodeNum
Uint16_t   songNum
Uint16_t   TOS_msg_strength
```

The SongStatistics module uses a 64-entry circular FIFO queue, and each new entry writes over the oldest entry in the queue.

The SongStatistics module determines a point value based on the number of times the song has been sang and how close in proximity the node was that sang the song(via signal strength).

The SongStatistics module utilizes the variables from the adjustGlobals packet to make sure the flock algorithm is not dominated by a single song. This means the SongStatistics module needs to keep track of the songs that have been popular so that it can force the flock to move to another song. The "Threshold" and "Repetition" values from the adjust globals packet are meant to ensure that songs will be allowed to propagate through the flock, but then eventually die off. This growth and die off is accomplished by limiting the number of song repetitions once a song's point value crosses the "Threshold". The "Repetition" count limits the number of times a song can play; thereby, allowing a strong song to propagate to a large number of nodes and then die off once a song becomes repetitive.

To accomplish the divergence the system keeps a FIFO list of 3 songs that will basically act as a do-not sing list once the song has been selected too many times (aka "Repetition" times). A song is added to the list when it has been selected with a point score greater than the threshold. The program will keep a count of the number of times the song has been chosen with a point value over the "Threshold" value. Once the song has been chosen "Repetition" times the system will assign the song a 0 point value, making it impossible to pick. Once a new song has a point value above the Threshold repetitions

times it will push the oldest song off the do-not-sing list. You will always be able to sing something, because the system only tracks the last three songs sung greater than Repetition times, which means there are thirteen songs not on that list that it can randomly choose from. The nice thing is that the do-not sing list causes the strongest songs to die off. NOTE: Both the FIFO and do-not sing list are cleared when you call the reset function.

To select a song, first issue an "updateSongSelections" command to make the "SongDecision" module recalculate the song statistics. Then wait until the "SongDecision" module indicates it has finished by signaling an "updateSongSelectionsComplete" event. Next, choose a song by calling "getSongPoints". Do NOT call "getSongPoints" before "updateSongSelectionsComplete" has been signaled. It is fine to call another "updateSongSelections" without choosing a song as long as the "updateSongSelectionsComplete" event has already been signaled from the previous call.

You'll probably want to extend the SongStatistics module to keep track of the last signal strength, health status, and song played for each bird.

Method for determining when to startle a bird:

A startle should only occur in the LISTEN or SING STATES. There are two methods to startle a bird: the light sensor, and the radio.

Use the ADC to trigger the bird being startled by detecting when there has been a change in the light level. A startled should be triggered when the 4 most significant bits of the ADC value changes by 4 or more. You should be sampling the ADC approximately 3 times a second. If a startle occurs go immediately to STARTLED_STATE and sing the startled song. After you finish singing the startled song then send a startled packet and return to the normal sing and listen. You do not send a SangSong packet for a startled, only send a startle packet. To avoid the entire flock startling when someone is trying to identify a bird, do not send the startle packet until after your algorithm is sure that the light change is not associated with a startle. In other words, play the startle song (but only if you're in the LISTEN or SING state) and hold off on the transmission of the startle packet until the timeout lets you know it's not an identification. NOTE: Only the initiator of the startled sequence (ie. Node which detected a change in light level) should set the StartledSequenceNum.

The startle detection mechanism should only detect one startle for a single movement. We want to avoid two startles being caused by the same movement. For example if someone puts their hand over the bird an edge transition will be detected as the light level is reduced (the bird becomes startled) and as the person takes his hand away the light level increases causing a second edge detection (second startle event). A person walking by a bird obscuring the light should also only cause one startle song to be played. You will need to come up with a solution to this problem so the bird is only startled once. You will need to keep the startle mechanism working the same so that the bird can still be

startled by quickly increasing or decreasing the light level. Reasonableness should be used when designing your solution. If someone leaves their hand over the bird for 1 hour or even 15 seconds it is fine to count the hand decreasing the light level as one startle and the hand moving away as a second startle. The goal is to design a solution so that a reasonable movement will only be detected once.

If you bird receives the startled packet then your bird should check to see if 1) (HopCount > 0) and 2) (StartledSeqNum != previousStartledSeqNum). Basically check if your bird has already been startled by this startled sequence so that a birdie is not continually startled by the same initial startle. If these two checks are true, then your bird should become startled. After you finish singing the startled song, decrement the HopCount, store the startledSeqNum, and send a startled packet if HopCount is > 0, then return to the normal sing and listen.

Bird Calls/Songs:

All of the song information that you will need to use in this project is in the birdsongs.h file. There are two arrays of note: a short array containing the startle song (this can be kept in static memory), and a 2-dimensional array of the regular 16 songs you have been using for the past several labs. The large array should be kept in flash memory as indicated by their array declarations.

Also included in this file is a function to play the startle song. This function will adjust the proper registers to change the tempo and the volume and load the entire song into the FIFO since the entire startle song fits into the FIFO.

A second file has been included: timbres.c. This file contains the initialization code for the timbre and timbre-allotment registers (it also includes tempo and volume register default settings). This code will replace the initialization of timbres, timbre allotment, etc. in HPLSoundBoardM.nc.