LAST REVISED - 06/04/05 at 02:37pm

Each set of partners will implement their own version of the specifications listed in this document. Your compiled code will then be run on both of your birds for the Flock demonstration on June 8th from 8:30am – 10:30am. Remember you will need to qualify before you may participate in the Flock demonstration.

Hints:

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 for all groups to have slight differences as long as they meet the specifications.

Try to minimize the amount 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.

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.

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* Turn Tri-Color LED off and turn on the Red LED located on the corner of SoundBoard1 IF(*AdjustGlobals)* *Go to INITIALIZATION STATE*
B)	INITIALIZATION STATE: (bird defaults here when turned on) Initialize each of the 8 phrases that you will be singing according to the Method for Selecting Default Genes Section. Turn off all LEDs Initialize the DeathAge variable to some random value between LifeSpanMax and LifeSpanMin Set CurrentAge and BreedCounter to zero Calculate the threshold numbers for each of the 6 stages of life and store so there is no need to recalculate (see Method for Aging Birds Section) Initialize the BreedTimerLength variable to some random value between MinBreedTime and MaxBreedTime Go to CLEAR STATE
C)	CLEAR STATE clear Correlation FIFO and receive queue data (all historical data) clear all counter variables Go to SING STATE
D)	SING STATE UPDATED BREED TIMER SECTION: Set breed time to fire in BreedTimerLength milliseconds. When breed timer fires increment BreedCounter. If

	BreedCounter >= NumBreedTimersNeeded then go to BREED STATE and reset BreedCounter to 0. (NumBreedTimersNeeded comes from the bird's age, previous factor was known as BreedRate). Set age timer to repeat every AgeTimerLength(from AdjustGlobals); when age timer fires go to AGE STATE. While waiting for breed timer and age timer perform the following jobs in parallel: - Sing Process: See Sing Process Section o If(lightlevel > `NightADCThreshold`) ▪ Sing Process: send bird phrases to Yamaha chip from the current phrase list. This goes on continuously with silence in between the phrases (50% - 80% silence). As the light level decreases the amount of silence increases. Tempo and Volume should be varied with NumBreedTimersNeeded within global limits, and changed only during periods of rest. o Else ▪ Silence - Broadcast Process: After each iteration through your phrase list, broadcast a SangPhrases packet. - Listen and Calculate Process: Refer to Listen and Calculate Process section.
E)	BREED STATE (BREED STATE should be executed in parallel with SING STATE, don't stop singing to breed. The reason that BREED STATE is separated is because breeding only happens periodically.) Modify bird's genes by changing one of the phrases in the phrase list via crossover or mutation. See Bird Breeding Section for details. If breeding with "Crossover" algorithm, bit shift the correlation number by 7 and use the result (ensure the number is in the range 1 through 6) and use that to set the Tri-Color LED. If breeding with "Mutation" algorithm, set the Tri-Color LED to 7 (white light). Send AlteredGenes packet every fourth breeding cycle. (Note: each packet should the include information for the last four breed cycles). Go to SING STATE
F)	STARTLED STATE Send startled bird song to Yamaha chip. While singing your bird should continue the listen and calculate process. Turn Tri-Color LED off 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. Delay 10 seconds, turn off Red LED and Go to SING STATE

G	AGE STATE
	Increment CurrentAge.
	If(Currentage > One of the 5 thresholds)
	  Alter breed rate by changing the NumBreedTimersNeeded factor
	If(CurrentAge < DeathAge)
	  Go to SING STATE
	Else if(CurrentAge >= DeathAge)
	  Go to INITIALIZATION STATE

Some Implementation Details:

TOS_Msg.strength is automatically updated with the packet strength during reception.

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.

You can set the transmission power level using the function CC1000Control.SetRFPower(int) in component CC1000ControlM.

When filling the message packets use the same “endian” as the Atmega.

The Atrium Group ID will be 122. 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, A14)

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.

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

AM #	Flock Message / Packet

50	AdjustGlobals - A message from Node 0 containing global parameters for all birds. This packet should be accepted and processed at any time in any state.
	
	Default Values are subject to change. Non-time based values relate to number of age ticks which have occurred.
	
	`uint16_t    Node0`
	`uint16_t    TransmitPower        default 1`
	`uint16_t    StartledHopCount     default 1`
	`uint16_t    LifeSpanMin          default 30`
	`uint16_t    LifeSpanMax          default 2000`
	`uint16_t    MinTempo             default 31`
	`uint16_t    MaxTempo             default 17`
	`uint16_t    MinBreedTime         default 5000 in milliseconds`
	`uint16_t    MaxBreedTime         default 13000 in milliseconds`
	`uint16_t    AgeTimerLength       default 10000 in milliseconds`
	`uint16_t    InbreedingThreshold  default 4`
	`uint8_t     InbreedingLimit      default 50`
	`uint8_t     NightADCThreshold    default 20`
	`uint8_t     MutationPercent      default 5`
	`uint8_t     RestScaleFactor      default 2`
	`uint8_t     MinVolume            default 0x08`
	`uint8_t     MaxVolume            default 0x1F`

51	StopandListen - A message from Node 0 telling you to stop and listen. This packet should be accepted and processed at any time in any state. `uint16_t    Node0                On receipt, go to A`
42	SangPhrases - The "I sang song of these phrases" message; a message from some other birdie indicating what phrases it sang. You should send a packet of this type after singing your entire phrase list.

uint16_t	TransmittingNodeNum	local # of node singing
uint16_t	SequenceNum	start at 1, increment each time you send this packet
uint16_t	phraseDNA[8]	List of phrases that birdie just sang where phraseDNA[0] is the first phrase sang in the last sequence.
uint16_t	Tempo	
uint16_t	volume	

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 just continue the algorithm. Do NOT send a SangPhrases packet for the startle song. Instead send the startled message. Remember to decrement the HopCount when you send the message. This packet should be accepted and processed at any time in any state.

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 startled by radio message. Decrement value before sending.
uint16_t	StartledSeqNum	A randomly generated number that is stored to check to see if you have been startled by this startle packet before. Set by the bird startled by the light sensor.

43

AlteredGenes – A message containing information about the last four breed cycles. This packet should be sent out at the completion of the fourth breeding cycle.

The RSSI values should be populated from the Correlation FIFO at the time of sending this packet.

uint16_t	TransmittingNodeNum	local # of startled node
uint16_t	Matings	# of matings since birth
uint16_t	MaxStrengthNode	Node num corresponding to closest node
uint16_t	MaxStrength	RSSI value corresponding to closest node
uint16_t	MinStrengthNode	Node num corresponding to most distant node
uint16_t	MinStrength	RSSI value corresponding to most distant node
uint16_t	Mates[4]	Local # of last 4 mates where Mates[3] is the most recent mate (genetic history)
uint16_t	MatesCorrelation[4]	Correlation value of last 4 mates where MatesCorrelation[i] corresponds to the correlation value of Mates[i] song
uint8_t	numMutations	Number of mutations which have occurred since last AlteredGenes pkt sent

Sing Process:

Each bird phrase contains a variable number of notes to play and each note is stored as a `uint16_t` in a 2D array stored in program memory. The 2d array in program memory contains all notes that will be played and is indexed by phrase number. The first `uint16_t` of each phrase array contains the number of notes that phrase contains. For example phrases[phraseNum][0] contains the number of `uint16_t's` which follow and phrases[phraseNum][1], phrases[phraseNum][2], and phrases[phraseNum][3] contain the phrase's 3 notes.

Each bird should be continually singing so you should setup the Yamaha interrupt to notify you when the FIFO is beginning to get low (don't set it to one as you don't want to wait until the last instant to add more notes even if they are rests). Each phrase in the phraseDNA array should be played in order and separated by a variable number of rest notes. There are four rests notes to randomly choose from and play. The number of rests to play should be the result of the following equation which is a function of light level:

Num_rests_to_play = xor(4 most significant bits of the ADC light level) * RestScaleFactor

Once you have calculated the number of rests to play, you should randomly select on of the rest notes and add it to the Yamaha chip FIFO. Repeat this process until you have selected "Num_rests_to_play" silent notes. The silent notes are located in the rests[] in Meadowlark.h.

While the light level is below NightADCThreshold, the bird should remain silent.

Listen and Calculate Process:

Receive queue length = 2
Correlation FIFO length = 16

While listening, collect information on songs (phrase lists) being sung (AM #42) by surrounding birds. Incoming SangPhrases packets should be compared against 2 other 'waiting' packets in the receive queue to see if it from a 'closer' bird that is not already in the correlation FIFO. To determine if a bird is closer, compare the signal strength. If the signal strength is smaller than one of the other 2 waiting packets in the receive queue and there is not an existing entry in the correlation FIFO with the same bird ID, replace the largest signal strength packet in the receive queue with the new packet.

Each correlation FIFO entry should contain the following information:

```
uint16_t   TransmittingNodeNum
uint16_t   phraseDNA[8]
uint16_t   TOS_msg_strength
uint16_t   Correlation value
```

Each addition to the correlation FIFO writes over the oldest entry in the FIFO.

To calculate the Correlation Value of the song you are about to process compared to the song you are currently singing follow the following algorithm that runs the computeDistanceBetweenSongs() function on all songs, with all possible shifts, and keeps track of the lowest score the songs achieve. At the end, the "closest" song is used for the mutation/breeding.

```
computeDistanceBetweenSongs:
1. initialize an int bestscore (to infinity), or to some maximum int
2. for shifts of 0, +/- 1, +/-2, +/-3 and +/-4
     a. computeDistanceBetweenIndices ()on the song you are currently playing, and
        a shifted version of the song you are trying to obtain a Correlation Value to
        compare how close the song is.
     b. multiply the result of c computeDistanceBetweenIndices() by the (|shift| + 1)
        to force a penalty for shifts (i.e., if  we shifted a score by 4, we multiply by
        5—this ensures that we do not zero out everything with a zero )
     c. if the result for (b) is smaller than the current best score, change bestscore to
        the current bestscore and continue
```

This function runs computeDistanceBetweenIndices on any two songs (it is assumed that song1 will be the bird's current song, and song2 one of the (possibly shifted) songs the bird has heard recently. The function computes the distance between the *indices* of two phrases.

```
computeDistanceBetweenIndices (int i, int j) {
        return min {|i – j|, closestEndDistance(i)+ closestEndDistance(j)}
}
```

For closestEndDistance, assume n is the size of the array—thus, the highest index is n-1. This ensures that items at index 0 and index n-1 are exactly one apart (and that everything is at least distance 1 from one of the two ends).

```
closestEndDistance (int i ) {
  return min {| i – 0| , |n – i|}
 }
```

This algorithm will find the smallest distance between two indices into the list of n (set at 49). Since we envision the list of songs as a circular array, but want to avoid using modular arithmetic (for reasons of speed), we use the closestEndDistance function.

There are two cases which occur during the calculation of computeDistanceBetweenIndices (in the case of equality, it doesn't matter which is chosen):

1. $|i - j| <$ closestEndDistance(i) + closestEndDistance(j)

   In this case, the smallest distance between the two points is the normal straight line distance, without wrapping around.

2. closestEndDistance(i) + closestEndDistance(j) $< |i - j|$

This is the case where choosing wrapping from one end around to the other is actually a shorter distance than computing the “straight line” distance of |i – j |.

Bird Breeding Process:

Compute linear distance between my birdsong and heard songs as above.

Modify genes by:

1) If no bird exists in the correlation FIFO go to step 5.
2) Choose a bird to mate with.
   a) If InbreedCounter is less than InbreedingLimit
      i) Choose a bird with the lowest correlation value from the Correlation FIFO.
   b) Else
      i) Choose a bird with the highest correlation value from the Correlation FIFO.
      ii) Clear InbreedCounter.
3) If the correlation value of the chosen mate is less than InbreedingThreshold, increase the InbreedCounter value.
4) Compute a DNA crossover with the phraseDNA from the bird selected in step 1: (happens every time)
   a) Determine the starting position by obtaining a random number in range 0-7. (This starting position applies to both the source and destination phraseDNA).
   b) Select a random number between 1 and 5 and copy that number of phrases from the source DNA to your local DNA. Make sure to copy DNA phrases with wrap-around.
5) Perform a DNA mutation MutationPercent of the time
   a) Calculate which gene to change in the phraseDNA by computing a random number in range 0-7.
   b) Compute new gene based on a computed random number in the range of 0-48.

Method for Selecting Default Genes Section:

1. Randomly pick a phrase in the range 0-48
2. Randomly pick a number, between 1 and 4, of times to repeat phrase
3. Add the phrase select in step 1 to phraseDNA repetition times (step 2).
4. Repeat until phraseDNA full (goto step 1).

Method for determining when to startle a bird:

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 changes its value by 3 or more. You should be sampling the ADC approximately every 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, steps D through F. You do not send a SangPhrases packet for a startled. 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 10 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, steps D through F.

Method for Aging a Bird:

The birds should age over time and have a different rate of breeding based on the current age of the Bird. There will be 6 different rates for breeding 0 through 5. The length between breed timers firing is BreedTimerLength. When the Breed Timer fires, the BreedCounter should be incremented. If BreedCounter >= NumBreedTimersNeeded, then you should go to the BREED STATE. (NOTE: a NumBreedTimersNeeded value of zero means the bird is too young to begin breeding yet). When the bird reaches the DeathAge it should move to the INITIALIZATION STATE and be reborn again.

Both the DeathAge and the 5 thresholds to change the rate of breeding should be calculated during the INITIALIZATION STATE. The DeathAge is calculated by obtaining a random number between the two global parameters between LifeSpanMax and LifeSpanMin. The 5 thresholds are obtained by dividing the lifespan of the bird (Age 0 until DeathAge) into 6 equal parts. You should only calculate the Age Thresholds once to avoid performing duplicate calculations that will waste processing cycles. ~~Notice that Breed Rate peaks at 3 and goes back to 1 with age.~~ NumBreedTimersNeeded is also used to set volume and tempo within their min-max limits.

NumBreedTimersNeeded	Begin Age	End Age
0	0	DeathAge/6
4	DeathAge/6	2 * DeathAge/6
8	2 * DeathAge/6	3 * DeathAge/6
12	3 * DeathAge/6	4 * DeathAge/6
15	4 * DeathAge/6	5 * DeathAge/6
19	5 * DeathAge/6	DeathAge