# Introduction (Rowshon):

# Protocol (Justin):

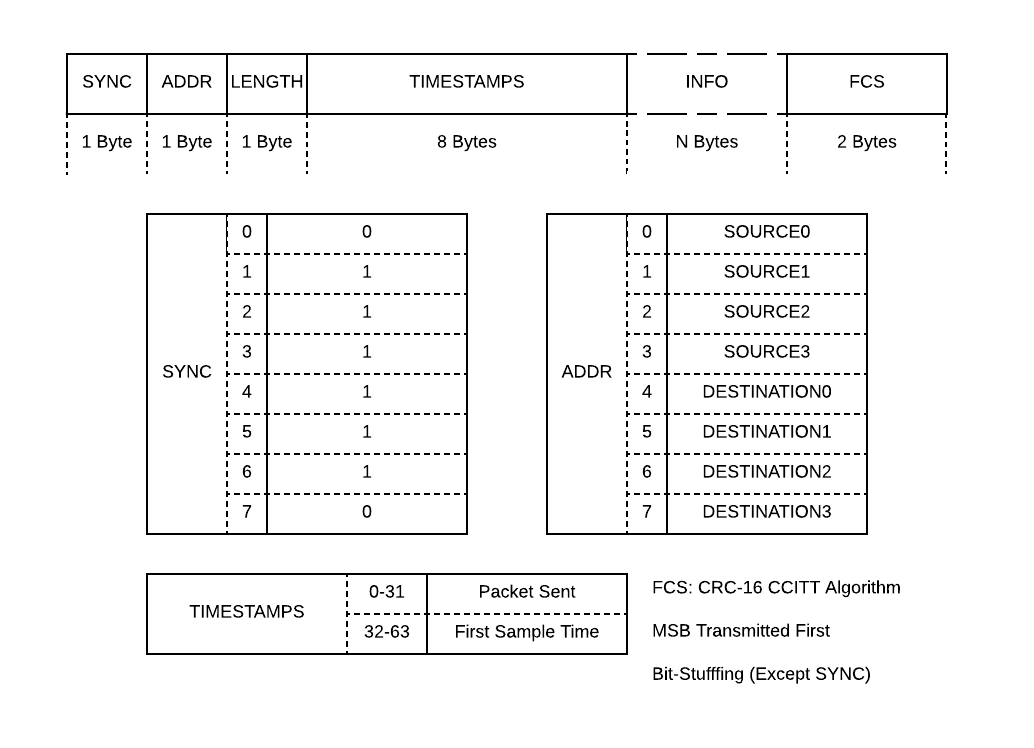
## Physical Layer (Justin)

The physical layer parameters are defined by the capabilities of the CC2500 radios. A standard radio setup was used, and no extraneous design choices were made for the physical layer. TABLE ??? shows the relevant physical layer parameters.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Data Link Layer (Justin)

A custom packet format was developed for the project. The packet uses a SYNC work for byte aligning received packets, an address field to identify the source and destination of the packet, a length field that specifies the length of the packet, a timestamp field that provides the necessary information for time synchronization, the info (or data) field, and a frame check sequence to verify that there are no errors in the packet. FIGURE ??? shows the format of the packet. All fields within the packet are transmitted most-significant bit (MSB) first.



The SYNC word is a zero, followed by six ones, and then another zero. To avoid the possibility of this pattern appearing elsewhere in the packet, the rest of the packet is bit stuffed. Bit stuffing is achieved by searching for a sequence of five ones, and then inserting a zero immediately following the fifth one.

The address word contains both the source and destination addresses. The first four bits of the address word are the address of the source node, the last four bits are the address of the destination node.

The length word contains an unsigned 8-bit char that specifies the length of the info field. This is used to tell the receiving node when to end the packet.

The timestamps field contains two 32-bit unsigned integer timestamps. The first timestamp is the time at which the packet was sent. The second timestamp is the timestamp of the first sample of data within the packet. Alternatively, the first timestamp could be the time that the previous packet was sent. This allows the timestamp to be created closer to the actual transmit time, eliminating some of the processor delay and allowing finer synchronization.

The info field can be any number of bytes, up to a maximum of 255 bytes, as defined by the 8 bit length field.

The Frame Check Sequence is a two word field that uses CRC-16 CCITT algorithm. It is calculated using the every field except the SYNC field. The receiver recalculates this field and compares it the value contained in the packet. If they are the same, then it is almost certain that there are no errors in the packet.

For the implementation presented in this paper, the RX interrupt stores the FIFO into a temporary buffer and calls a “find\_sync” function that begins to search bitwise through the buffer for the SYNC word. When the sync word is found, the function fills the next 16 bits into an RX Buffer, un-stuffing any bit stuffed zeros if necessary. The first 8 bits are read to find the source and destination addresses, the next 8 bits are read to find the length of the info field. The packet then fills the rest of the packet into the RX Buffer based upon the specified length field, while un-stuffing any bit-stuffed zeros. If necessary, multiple RX interrupts are triggered to continue filling the buffer for long packets.

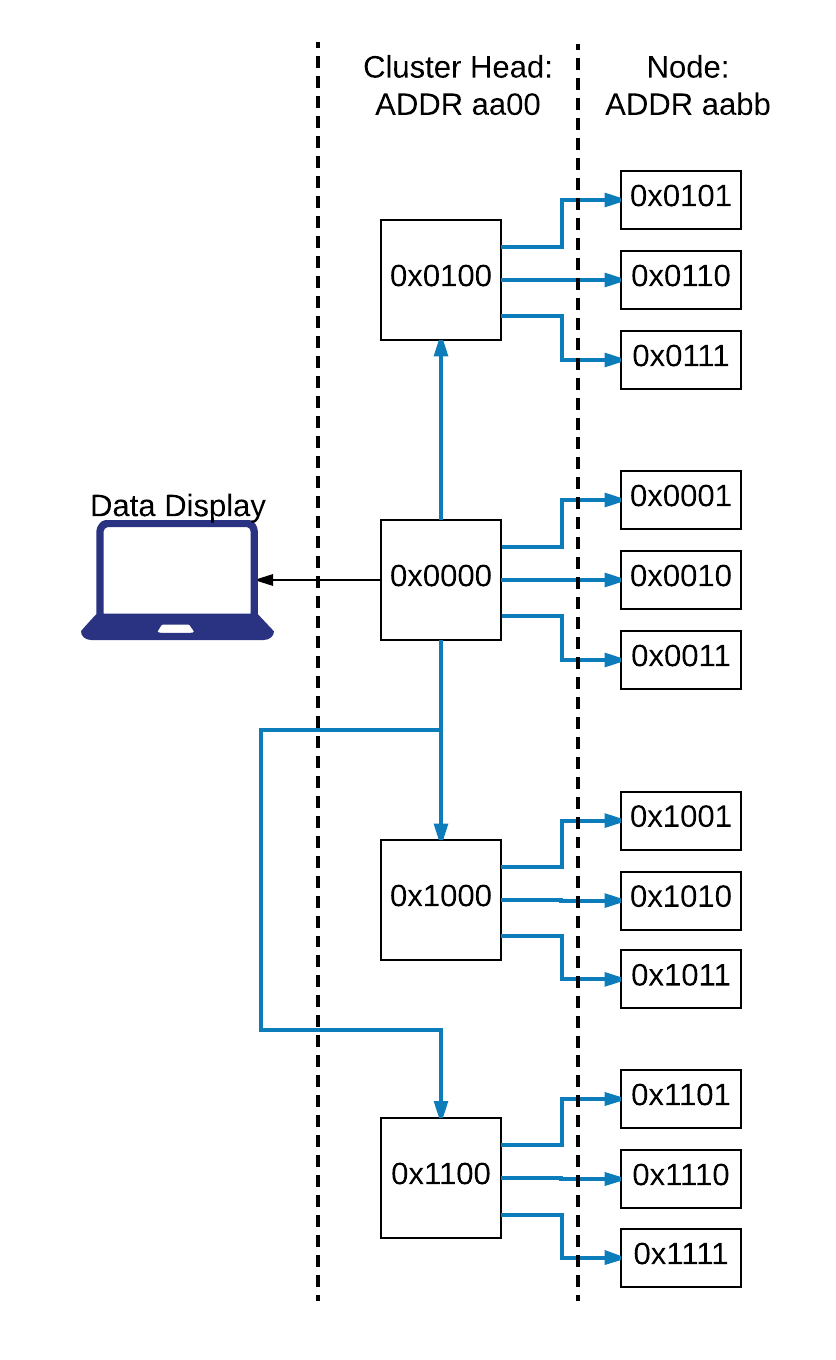
Once the whole packet has been received, a “packet received” function is called which first checks the FCS. If the FCS is correct, the function implements whatever commands are specified by the network layer protocols.

## Network Layer (Justin)

The protocol becomes project specific in the network layer. In this case, we chose to implement a topology that would be well suited to the structural health monitoring of a tower. The 4 bit address field allows a total of 16 devices in the network. We broke this into four clusters, each with one cluster head and 3 nodes. The intent is for each node to be a small collection of sensors on one leg of the tower. The cluster head would be a larger node with a larger battery to facilitate cluster-to-cluster communication.

The first two bits of the address specify the cluster to which the device belongs, the second two bits specify which node in the cluster the device is. The cluster head is always “aa00”, where “aa” is the cluster address. The rest of the nodes in the cluster are addressed “aabb”, where “bb” specifies the address of the node within the cluster.

One cluster head connects to the end user application, such as a laptop. This cluster head will be responsible for time synchronization throughout the network, data organization, and application data reporting. Since this cluster head is directly connected to the end user application, it will not use battery, and is considered to have no energy restrictions. FIGURE ??? shows the topology and addressing of the network.



### Time Synchronization

The original plan for the project had been to implement a time synchronization algorithm similar to that used by SOURCE ???. There would be two layers of synchronization: synchronization between the nodes and the cluster heads, and synchronization between the cluster heads and the base station. Each layer would use the same synchronization method. First, three timestamps are taken:

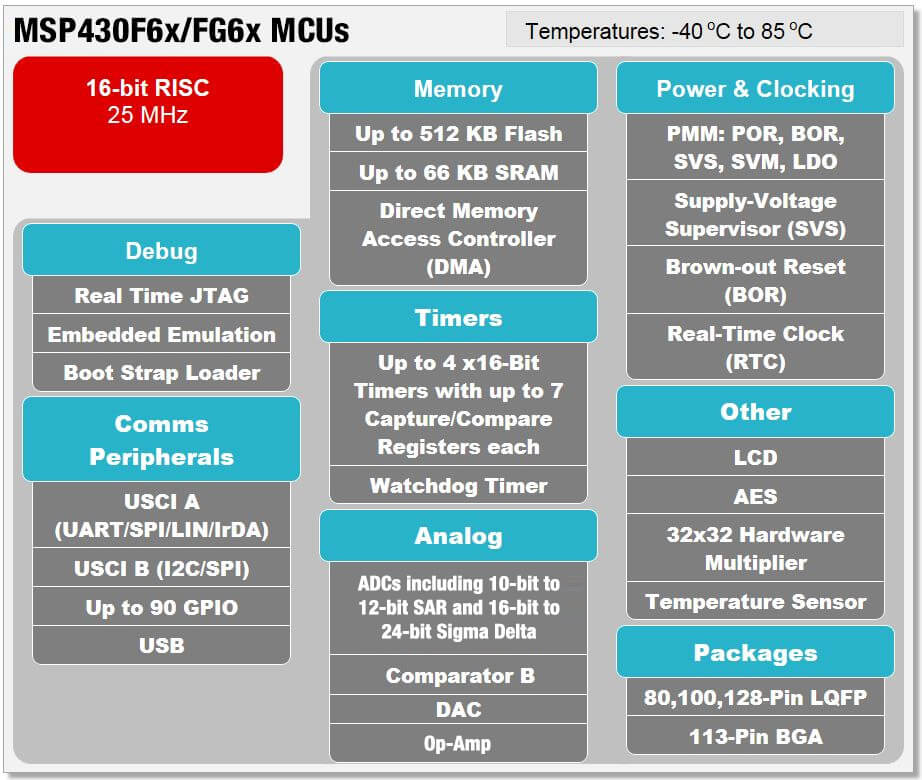
* Timestamp the first sample of every packet. Send the sample timestamp in the application header of the packet.
* Timestamp the time the packet is sent. The sent timestamp is included in the data of the next packet.
* Timestamp the time the packet is received.

The sent and received timestamps are used to reference the node clocks to the cluster head clock. The sample timestamps from the nodes are then mapped in terms of the cluster head's clock. This is a multi-step process that involves eliminating any timestamp errors resulting from lost packets and interpolating between timestamps to create a common clock [SOURCE].

Due to time constraints, this time synchronization was not actually implemented. Instead, a simpler broadcast system was implemented. The cluster head creates a synchronization broadcast, with a timestamp as it creates the packet. The nodes receive this packet and update their timestamp. The issue with this method is that it introduces considerable interrupt handling and access time delay.

# Hardware and Implementation (Jordan & Rowshon):

## MSP430 (Jordan & Rowshon)



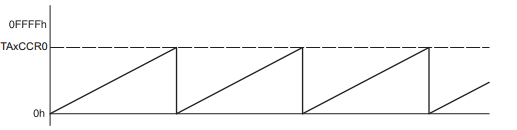
The MSP430f5438 has been selected as the projects embedder microcontroller given it contains all necessary peripherals and can carry out all necessary computations. The all available MSP functionality is shown in FIGURE, as can been seen the MSP provides an all in one microcontroller solution from its debugging capabilities which allow us to run the code line by line to track down code errors to its communication peripherals that allow for an easy interface between the user and the CC2500 radio.

### UART Setup

Using the existing serial to UART converter hardware on the MSP430 experimenter board a USCI A peripheral is set up to facilitate a UART to serial terminal connection. Although a serial connection is not a core function of the WSN system it is invaluable for designing and testing the system as it allows a real-time interface with the user/developer. Most importantly the serial connection allows for the end user of the system to obtain sensor data and health status of the WSN system. Through this interface it is possible to reset the system, carry out transmission testing, check the status of the system, check and change node/cluster head addresses, and view all received data and failed reception/transmission. Given the wide range of functionality provided it is easy to see the necessity of UART communication in the system.

### Timer?

In order to keep a virtual clock on the MSP it is necessary to take advantage of the built-in timer modules. The timer modal is used in “up mode” to use clock ticks to count up to a pre-programmed value of the capture compare register, as seen in FIGURE. Using a capture compare register in this manner the MSP can be kept in a low powered sleep state while the timer peripheral uses interrupts to increment the virtual clocks counter for the programed time interval.



## Radio (Jordan):

All nodes in NAME ??? system use the same radio, TI’s CC2500 for all wireless communication. The CC2500 is a low-cost S-band transceiver designed for low power applications and also allows for interrupt driven transmission and reception. Using an interrupt driven transmit and receive scheme is made possible by the use of the CC2500’s programmable GDO pins. All nodes use 2-FSK, bit stuffing, and an FCS redundancy check.

CC2500 radios provide three programable general digital output (GDO) pins. GDO2 is used in the SPI peripheral of the CC2500 to communicate with the MSP430 while GDO0 and GDO1 are programmed to trigger an interrupt on the MSP430 in the event of a radio reception or transmission. More specifically the interrupts occur based on the number of bits in the First in First Out (FIFO) buffer threshold. When the number of bits passes the programmed threshold the associated GDO pin will be pulled high. This allows for the MSP to then appropriately service the flagged TX/RX buffer.

Apart from the GDO pins the only avenue of communication between the MSP and CC2500 is their SPI bus. Through the SPI buss the MSP can program, read/write registers, and most importantly strobe the radio into different states. Upon receiving a strobe, the command, the radio will change to the directed state, in doing so changing its operation. When strobed into TX mode the radio will initiate transmission of whatever is populated in the TX FIFO until it runs out of things to transmit. In contrast the only way the radio can receive packets is if it has been strobed into the RX state.

# Results (Jordan)

# Other Work (Rowshon):

## SensorTag

## BLE

# Future Work (Justin)

There are three major improvements that were planned as stretch goals for the project. First, the time synchronization algorithm that was originally proposed could be implemented. Second, collision avoidance and sleep scheduling could also be added to improve functionality. This is discussed in more detail below. Lastly, sensors need to be integrated into the network.

## Collision Avoidance and Sleep Scheduling

The small size of the network makes time-division multiple access (TDMA) a good choice for collision avoidance and energy conservation. Medium access could be time-divided according to device address, eliminating contention based overhead and collisions. This would also provide a sleep and listening schedule for the nodes, reducing overhead due to idle listening.

This comes at the cost of latency and fairness. To get a full analysis of the structure the network will have to wait to hear from every node, causing significant latency. However in most SHM applications latency is not critical. The network is used mostly for analysis and monitoring. In cases where the network is used to initiate a response system, the delay of the response system (such as heaters, mechanical dampeners, clamps, alarms, etc.) is typically orders of magnitude longer than the latency.

In terms of fairness, the cluster heads will lose battery life faster than the other nodes as they will have longer listening and transmission period. This could be solved by giving the cluster heads larger batteries, rechargeable power, or implementing an cluster head election system.

# Conclusion