**Nrf8001** Key points about the nRF8001(possible include the figure from the data sheet showing connections to the uC)

-Spi communication

-Two way hand shake

-Application command interface

-power up and load firmware generated by the nRF go Studio. This firmware is dependent on the application of the nRF8001.

**Tiny OS** in summary:

TODO

**Original Mission**

Add BLE to a tinyos mote via the active message interface and measure the power consumption compared to the original radio.

**What was actually done:**

Integrate the Nordic BLE stack into tinyos to allow motes to talk over blue tooth, this could not be a substitute for the original radio as was envisioned because Bluetooth is a connection based protocol. Tiny os was built on broadcasting openly or to specific nodes but a connection did not have to be established ahead of time in either case. So rather than attempt to connect the BLE stack into the active message interface and compare power consumption the project focused adding BLE functionality to tiny os as an expanded feature and not a substitute for the original radio.

A new platform was constructed, micable, which was an adaptation of the micaz platform to incorporate the new blue tooth chip. At the time the micaz was selected because it was believed that this was the only mote platform that made SPI lines externally available via the MIB…… expansion board. It was later discovered that the only SPI line on the expansion board was SCLK, so the SPI was bit banged using the GeneralIO interface connected at the platform level the necessary IO lines.

The Nordic stack consisted of 2 main parts(high and low level platform independent software drivers) and 2 parts that were used for data encoding and queuing. Each part of the nRF8001 stack provided by Nordic became a tiny OS interface and a top level driver configuration was used to wire them together and ultimately to the lowest platform dependent layer.

-lib\_aci – This was the top level provided by the nRF Bluetooth stack. It was used to determine what commands would be sent to the hal\_aci\_tl interface.

-hal\_aci\_tl- This module was the lowest platform independent level of the driver. It was told what command to send by lib\_aci and determined how the commands would be sent. It used pins set at the platform level to bit bang spi commands out that were placed in the tx queue and add incoming message to the rx queue.

-acilib – this module encoded standard messages defined by the application command interface.

-aci\_queue- This module was used to perform operations of the incoming and outgoing queues to that application command interface.

**The test application:**

A UART test application was adapted from the Arduino library to run in Tiny OS which allowed the mote to successfully send messages back and forth between an Iphone and the micaz mote. The mote is the client and the iphone is the server in this configuration. Once the server has initiated a connection it can send information over BLE that the mote will print to the terminal via a serial port.

**Results**

I will be generating these tonight…

**Future work:**

The fact that TinyOS is an event driven operating system was not leveraged by the Bluetooth stack because it was adapted from what would traditionally run sequentially on a generic microcontroller. First, in the future it would be nice to adapt the stack to take advantage of all of the intricacies of TInyOS. The stack developed used only commands to call down into the stack and polling commands instead of event for moving information up the stack. Next, the use case of Bluetooth Low energy in embedded sensor networks needs to be analyzed to determine how to best encapsulate the new stack so it can be easily wired into future applications and platforms. Lastly, it is also worth mentioning the next generation of Bluetooth low energy will incorporate support for mesh networks. This lends, combined with the work completed above, could open up too opportunities of research.