Progress Report 4

Stuart, Walt, Dan

Next Goals and Deliverables

- ASCII to bitstream converter method - Dan (maybe done by EOD)
- Bitstream to ASCII converter method - Stuart (maybe done by EOD)
- Clean up code, write documentation - Walt
- Make one class to read and write simultaneously (may require threading/simultaneos execution)-Walt
- Begin work on error detection - Dan
- Begin working on network layer design - Group
- Testing ASCII to bitstream and bitstream to ASCII - Group
- Thorough testing of the manchester encoding/decoding - Walt/Stuart

Previous Goals and Deliverables

- Debug program for receiving manchester encoding data (DONE) - Stuart/Walt
- Develop standard packet expectations - group (DONE for now – will revist with network layer with error correcting)
- Create character encoding/decoding to allow text transfer - (WIP) - Dan

  Dan has made progress on this, but we needed to fix the physical link before doing this.
- develop functionality for repairing broken packets using error detection and bit replacement. (ON HOLD/DONE FOR NOW) - Dan

  Dan read a lot about this, and we have a good sense of where to start on error-detection. However, we decided to move on to transmitting English messages before we implement this.

Discussion

- We spent much of our class time trying to debug our read rate, only to realize there was nothing wrong with it. We realized that our bits were inverted due to the input/output nature of the NIC device, and some of our errors were being caused by an existing NIC state.
- We have a working program(s) for receiving manchester encoded data. We can send data at rates up to 3,000 bits per second. This is a good start, and we are happy that we have reached the upper bound of what the class belives is possible for these devices. However, we will have to add error checking to our packets. Once we start to focus on the network layer, we will also need to come up with a handshake for the devices to connect, and will need to add automatic timing resets to the listening function so the clocks don't get out of sync.
- We use the callback method from the PiGPIO library to check for edges, then using two "start bits" to synchronize the clocks. Once we have calculated the READRATE, we use that plus a 25% buffer to read the bits, skipping the clock bits.
- One feature we have yet to implement is the ability to calculate the READRATE on the fly. We can do this by checking the tick count during the transmission, and adjusting the READRATE accordingly. This should allow us to listen at all times, without having to worry about the clocks getting out of sync.