Deliverable #5:  
Alpha Firmware and Breadboard Demo

1 Objective & Details

The point of this deliverable is to convince me and your classmates that your software is well on its way and that you’re system is going to work. Specifically, you will turn in working, tested code and present a functional bread-boarded (or otherwise hacked together) version, of some core function of your project.

Core Function: Your software implementation should not wait until you have your boards completed. For this deliverable, you should ideally have your entire system “working” (for some definition of working). Realistically, you’re boards probably haven’t arrived yet, and you may be sorting out other hardware issues. So, at the very least, you need to have some significant portion of your software implemented and tested using a breadboarded or otherwise hacked-together test rig.

For example, if I was building a WiFi-enabled smart refrigerator that ordered food automatically and my WiFi module wasn’t quite working yet, I might instead implement the food-missing detection and ordering logic. I might use another MSP430 to act like a WiFi module, receive SPI messages and check to see if they are well-formed. The idea is to test your system components to a degree that they are more likely to work together, once your hardware setup is finally in place and working.

When I grade your deliverable, I’m going to ask myself, “Is this team’s software well on its way to being done?” If the answer is “no,” then you haven’t done enough.

To get full credit, you must specifically do the following:

- submit working software modules (your code should include at least two different classes) and test code (both unit tests and more integrated tests) via handin, along with a text file (called testing.txt) that describes of how to run the tests and why these functions are core to your project, and
• present a rough breadboarded/hacked together demo during the class period after the due date.

**Unit Tests:** The code you submit to handin should contain both unit and integration tests. Unit tests should test a software module’s logic and function in isolation (independent of other system components). When possible, I should be able to run your unit tests easily on a single launchpad. In some cases, it will be easier to have a second launchpad verify the test outcomes — if, for example, your Serial module is being used for something else. Other tests may test a specific piece of hardware. Your description should explain how to set up each test. For tests that require special hardware, you should explain in the description why the extra hardware is necessary. I may not run all of your tests, but I should be able to easily run them by following your description.

**Integration Tests:** These tests combine modules and test aggregate functionality. Integration tests are more likely to depend on specific hardware configurations, but you should still make them as easy to run as possible. Integration testing that can be done with just a launchpad, should be done with just a launchpad.

**How many tests?** This is a tough question. As many as you need to thoroughly test your code. As a bare minimum, if you have fewer than 10 different tests, you probably aren’t being thorough enough, but some complicated components may need far more tests. Also, one test program can contain many different individual tests. When you make your tests easy to run over and over again, you increase the likelihood that you will run them.

## 2 Collaboration

This deliverable is completed as a group. Any code, algorithms, or other design elements that come from outside your group must be appropriately acknowledged in your description file. As with previous deliverables, one of you will be called on to demo your system in front of the class.

## 3 Submission Instructions

This deliverable is due by 4:00 PM, November 9\textsuperscript{th}. Absolutely no late assignments will be accepted.
Your code should be submitted via handin (handin.cs.clemson.edu) as a tar.gz or tgz archive. The archive should contain a description file (testing.txt) that lists all team member names, emails, your team’s git repository, and the information already described above. The code you submit should also be in your repository (it should probably just be a snapshot of part of your repository). Your code should be organized appropriately into subdirectories and using meaningful filenames. Your code should be readable and follow good coding style.

Your team should be ready to demonstrate your hacked up version the following class meeting (Monday, November 14th).

4 Grading

This deliverable is worth 5% of your final class grade. Your grade will be equally based on completeness (did you do what I asked), style (readability and organization), functionality (when I run your tests do they work), and of course your demo.

If your code is not submitted on time, you will get a 0%. If the randomly-chosen team member is not present (and has not arranged an exception with me) on November 14th, your team will get a 0% on the demo portion.