**Memo**

**Senior Design**

ENG EC 463

To: Professor Pisano

From: Brian Kane, Katherine Murphy, Ben Paolillo, Jiwon Song, Todd Sukolsky

Team: 14: Bikeputer (Re.Cycle)

Date:

Subject: First Deliverable Testing Plan

1. For the first deliverable, we demonstrated the functionality of an LCD Touch Screen with an Arduino, the functionality of the Beaglebone and the power management circuit (on a demo board). For the LCD screen, we showed that we can navigate between nine different screens and maintain data corresponding to the current screen position; additionally, we were able to calibrate the screen whenever the user needs to do so. This component is significant because the screen is the main user interface and will eventually display the data supplied by the user, GPS, and various sensors.

The BeagleBone is the main processing unit on the board and is responsible for communication with all of the modules. In the first deliverable test we needed to demonstrate gaining root access to the processor, the ability to connect and stream data via USB and transfer UART strings via wire.

The Power Management system is described as the control block of all power. The system should bring in a voltage from a battery, in this case a LiPo 2-cell, and run an ADC conversion. The output of this ADC converter will trigger the main regulator to provide power to the rest of the board or go send the uC into a sleep state along with the rest of the board. The regulator should be shut off when the voltage is below 7.43 volts for the battery used in our first deliverable testing.

2.0 LCD Screen:

2.1 Equipment and Setup:

The touchscreen LCD is set in a breadboard with a soldered on header, then wired to the Arduino board. The board includes a boot loader, which programs the Arduino with our most updated module, generating the graphics and user interface environment. For accuracy feedback, the board is connected to the USB port of a computer, then the Serial Monitor is executed to show UART sent from the Arduino. This data, in this demonstration, consists of the coordinates of the users touch (optionally) as well as the configuration parameters, when in configuration mode.

2.2 Process and Data acquisition:

During calibration of the screen ADC readings were taken from the screen and used to store the users’ x and y coordinates, based on which part of the screen the user is touching. If the current ADC result was a value smaller than the currently stored minimum or greater than the currently stored maximum, the current value will be tentatively stored as a replacement for the calibration parameters. In the end of the process the user was prompted on whether to save these new settings, or restore the calibration to its state prior to calibration. To test the calibrating, a simple “Touch the Dot” test was executed allowing the user to attempt to touch a small dot on the screen; following the touch the screen/uC showed the results of how far off the user touch was from the dot. This information conveys the success of the ADC measurements during calibration. Our test showed that when calibrating correctly touching the dot on the screen will yield an accuracy above 95%. However, when calibrating incorrectly, meaning the user did not run his/her finger across the perimeter, but moved across the center or created wavy lines, the accuracy on the dot test was below 95%.

The other testing procedure was to navigate between screens arranged in a 3x3 square (a home screen in the middle and N, S, E, W and intermediate directions on the sides). The user should be able to tap a button on the bottom right hand corner of the screen to move between the boxes. After a successful calibration, we successfully showed correct navigation through the screens. If a calibration was faulty, it proved to be very difficult to move from screen to screen as expected.

2.3 Conclusions:

As mentioned previously, when running a correct/true calibration test where the user legitimately tries to trace around the edge of the screen all functionality of the LCD is there. The user is able to move through navigation screens with greater than 95% accuracy. In the first deliverable testing we showed this, and also showed a faulty calibration and how it leads to a difficult process of moving screens, indicating the test was successful.

3.0 Beaglebone and Power Management:

3.1 Equipment Setup:

Beaglebone is connected to a Linux, Windows or Mac environment via USB and Ethernet. The USB allows connectivity on the local subnet and the Ethernet is used to stream the shared wireless or LAN connection to the bone. In Linux the “ssh” command is implemented to connect to the board, always at local IP 192.168.7.2. In Windows a client is used such as “Putty”. Attached to the bone, on the UART2 pins, was a development board Todd previously produced for a different project. The board contains the power management system minus the backup watch battery for the Real-Time Clock. Also attached was a mini USB -> USB cable that is used to stream NMEA strings from the onboard GPS. This GPS is a test unit for the NMEA string parser (currently under development in Python). Also attached to the Bone was a small circuit made to toggle an LED on and off and, with a python script, blink. The circuit consists of one led and a 220 ohm resistor. The anode of the LED was tied to one end of the resister, the other end tied to ground. The second terminal of the resister was tied to the GPIO pin that will output 3.3V.

The power management system was implemented on the development board and a schematic of the module on the board is provided in this packet. It used an ATMEGA168A, 8-bit, microcontroller to take ADC readings, print UART strings, and communicate with other modules on the board. The UART lines are fed to the header pins 0 -> AVR\_RXD and 1 -> AVR\_TXD. The ground pin for the connection was located at pin 17 -> GND.

3.3 Measurements and Data Collection:

Step one was to root connect to the bone using the command “ssh [root@192.168.7.2](mailto:root@192.168.7.2)”. The password for the root user is null. The first test executed was compilation of a C++ script to show functionality of the Angstrom Software Distribution. The program outputted a string to the terminal “Hello from Beagle Bone! So nice to speak to you today.”

Proceeding the C++ execution was proof of live stream through the onboard USB cable. The stream was live NMEA strings from the GPS on the development board. To execute, the “minicom” command was used which is native to the Linux operating system. The baud rate for the GPS is 9600 so the command for this is “minicom –b 9600 –D /dev/ttyUSB0”. The “-D /dev/ttyUSB0” string portion tells minicom to listen over the USB port for data. The output of this command was strings of the format “$GPxxx x,x,x,x,x,x,x,x,x,x,x,x”. There are numerous types of strings. “$GPGGA” strings contain location (longitude, latitude) data. These strings were blank in this case because of our null location. The terminal received the stream of strings and displayed them in the terminal correctly, in this case with blank data fields for current location and almanac data since no antenna was connected and no reception was attainable if connected.

After this demonstration the GPIO functionality of the bone was demonstrated by toggling an LED on/off given input in the terminal. After the toggle was shown, a python script was executed to blink the LED at 1Hz.

The last test of the bone was to implement its UART2 and UART1 pins. This was also a test of the battery management system. First a python script was run to set up the correct UART ports. This was viewed in the terminal and then executed. Following this a three line connection was made to the development board for UART\_TXD, UART\_RXD, and GND. Once plugged in live UART data will stream, using minicom once again, using the command “minicom –b 115200 –D /dev/ttyOx” where “x” is either 1 or 2, depending on which UART pins we were currently testing. The baud rate for the UART data is 115200. During testing the battery voltages should stream in the terminal indicating a successful transmission. After the successful transmission was completed, the minimum threshold voltage for the ADC conversion was changed to show the functionality of the regulator enable lines. After changing the threshold, the microcontroller should take an ADC reading, find a voltage lower than the prescribed minimum, and shut off the main regulator on the board. This will trigger the onboard LED’s on the development board to shut off. The microcontroller had a voltage reading of 8.202V, which was sent to the terminal. When measuring the voltage in a multimeter the voltage found was 8.207V.

3.4 Assessment and Conclusions:

The first test of the bone is judged on the simple process of gaining root access to the board and running the C++ program with the correct output to the terminal. The compilation was successful and the terminal output was correct, indicating a successful test of this feature.

The USB stream test is considered a success if the NMEA strings appear in the minicom terminal. If the strings are not present, then streaming is not taking place and there is an internal error. Instantly after the minicom command was executed strings started streaming into the terminal indicating no internal errors from the GPS output to the USB input and a successful test.

The final test can be deemed successful based on two collections: The first requirement is the ATMEGA168A needed to transmit the UART strings to the bone correctly on both UART1 and UART2 ports. The second requirement is the ADC reading from the microcontroller had to be within .075 volts of the actual voltage measured by a multimeter. With a nominal voltage of 7.4V, an error of .075 volts represents ~1% error worst case. Comparing 8.202V (experimental from strings) to 8.207V (actual) yields a percent error of .06%. This part of the test proves successful because the difference between actual and experimental, .005V, is less than the maximum allowed of .075V.