Current State of the Project:
The entire system is integrated and functional minus the desired LCD screen. Attaching the peripherals to the main module will enable the entire system at which point heart rate sensing, speed sensing, gps data and battery charging (when pedalling) will commence. The limitations with the LCD screen revolve on getting the 8080 programming method to work. The GAVR is responsible for interface with the LCD, however the Crystal Fontz screens (see vendor information) have proved tedious and unsuccessful. For ECE day we are falling back onto the LCD screen we tested at the first deliverable in the Fall. The screen is quite small, around 2.4” diagonal, however we are able to interface with it both on a breadboard and with the GAVR on our custom PCB. This represents the only unfinished and non-fully integrated part of the system.

Future Improvements:
FIrst and foremost the GAVR should be eliminated from the picture and all LCD functionality should be ported to the BeagleBone, eliminated at least $30 of cost to the system as well as greatly simplifying the interfaces. The BeagleBone has two forty pin headers that are designed to attach to pre made “Capes”, one of which is an LCD screen. The BealgeBone has kernel modules that can communicate with LCD screens and these should be included and uses. See BeagleBone.org for more information about the kernel modules and add-on capes.
The BeagleBone kernel could also be updated to allow for quicker boot time and better file management/usb support/lcd support. The current version of Angstrom is larger than required and a minimal Debian or other system should be used. One option is BuildRoot, however certain things need to be changed in the kernel modification that we are unclear on how to do. See “BeagleBone Kernel and OS” for more information on what was done.

If the GAVR is not replaced, the RevB should have flash storage that the GAVR can communicate with and store trip data that way instead of the EEPROM since the Atmel EEPROM is guaranteed only for 100,000 read/writes. Flash offers a smart solution that doesn’t require power in off-mode.

An improvement that will trickle down from only implementing the BeagleBone is a smaller PCB and therefore a smaller case. If the PCB is smaller the screen can be the largest denominator for the case which will significantly decrease the size. Currently, the case is a bit bulky and multi-level. To improve the case the levels should be broken down to one to create a shallower module. The batteries can then be moved to the same level and keep, or be close to keeping, the same lateral and longitudinal dimensions.

The battery placement and charging implementation could also be improved upon. By turning the main power source from a two battery system to a one battery system less space would be needed in the casing and the switching circuit would be obsolete. Time of use may suffer, however the current longevity on a full charge for the system is well over the requirements.
By connecting the GPS backup 3.3V line with the WAVR 3.3V backup line the system can

eliminate another coin cell battery also reducing the size of the system. The backup battery will not last nearly as long, however the GPS line draws near 0 current since the voltage is there only to preserve the SRAM during “off-mode” operation. During “full-power” operation the GPS does not draw any current on the backup line, but just used the line to store the current location data in that SRAM.

The real time clock for the system is implemented in the WAVR. This is effective because the system if the chip is in power save is draws less than 10 microAmps. The issue arises that when all power is lost in the system the internal date and time is lost. There are Real Time Clock chips, such as the MCP1702 and DS1337 that function only to keep time and work on a backup coin cell that is soldered to the board. The clocks are guaranteed to keep time for over one year minimum making them an ideal choice. Most communicate over I2C, so using an ATmega168A can be used to synchronize the time will all devices since it has I2C, SPI and two UART ports that can be interfaced with.

BeagleBone Kernel and OS:
Currently the BeagleBone is running a custom kernel of Angstrom, a lighter version of the debian system, that TI has customized for the BeagleBone. THis software is very heavy and takes a good thirty seconds to boot up when the original kernel is installed. Also, the initial kernel does not enable SPI1 and SPI3 as well as I2C1 and I2C3, and UART4 and UART5. To modify this the kernel must be recompiled. To do this, see the Software Readme as it has links to the tutorials on how to add the correct mux settings.

One alternative that was investigated and executed was the BuildRoot OS that is compiled using a large array of make files and then make menuconfig. The result is a very small rootfs partition and lightweight OS that boots up in around 4 seconds. Although the boot time was fantastic, the omap_mux settings were not trivial to change so the idea had to be abandoned. If you don't know what the omap_mux is, google “BeagleBone omap_mux”. In short, the mux is a way to change the BeagleBone headers to different settings. Once such setting is a normal GPIO pin that is accessible through “/sys/class/gpio/” while others can be configured to work with the LCD screen (lcd_data<x>) for example, while others deal with Serial Communication and SPI by wire. These can be changed by echoing certain values to the correct pin in /sys/kernel/debug/omap_mux. The script “pinSetup.sh” has numberous examples of this and is located in the Software/Source/BeagleBone/setupScripts folder.

One important feature that was difficult to find was how to enable interrupts on the BeagleBone GPIO pins. To do this, first the pin must be set as the GPIO input. Then to set up the GPIO in /sys/class/gpio the gpio must be exported, direction set to “out”, and “rising”/”falling”/”both” must be echoed to the “edge” directory. When the edge we are looking for is triggered, a uevent is created which can be detected by the OS. To detect these events a polling method must be used, from all of my research, and is not trivial. To see how to poll an interrupt pin see Software/Source/gavrInterrupt or shutdownInterrupt. Both pins are polled by their respective monitors and on a rising edge the interrupt routine is executed.

Some important notes is that the Angstrom distribution is booted using systemd, not init.d and rc.common. The result is to add scripts on boot a service must be installed in the /lib/systemd/system directory and then enabled. For information on how to do this the Arch Linux wii page has a great tutorial on the systemd call structure since ArchLinux uses this to boot.

GPS:
Currently all GPS functionality works and is completely accurate. The NMEA strings are streamed to the BeagleBone on UART 2 and when a new trip is started the previous trip's GPS data is placed in a file relating to the trip number on the GAVR. One issue that can arise is if trip numbering is lost on the GAVR. If there is a conflict in trip number the GAVR has full control and the gps data might be lost. To fix this in future revisions, the GAVR should constantly be checking and confirming that the BeagleBone is storing the correct trip information. In parallel, the BeagleBone should have the ability to change the number of the GPS files given a start date. The file structure can be much better for the GPS, ie the date and time of creation can be cross-referenced with the GAVR trip.

Speed and HR Sensing:
Currently the Speed sensing is 100% accurate and ready to go. The only possible improvement would be how the average speed is calculated. Currently is is a weighted calculation based on the number of rotations that have passed in the current trip. The maximum number of rotations that can be recorded is 65535, a uint16_t value. Once this is hit, the number is not incremented anymore and the average speed calculation no longer becomes dynamic and using the same weighting factor. This implies that if you cycle a trip for over two years, or even multiple days, the average speed will change more rapidly than it should after a longer amount of time. One way to fix this is to implement long 32 bit calculations using a separate class, or convert the number to a double that can store a HUGE value. The issue then becomes conversions during calculations which drive up CPU use and operation time.

Heart Rate sensing is working, but not optimal. Currently peak detection is being used by sampling an ADC pin whose input comes from the Pulse Heart Rate Monitor. The absolute maximum is found, then we look for local maximums a certain distance away from the max depending on 1)last heart rate, 2)average sample reading, 3)range of adc data and 4)where the maximum peak was detected. A better implementation of the Heart Rate sensing would be to integrate with a Polar or Garmin strap heart rate monitor that sends a pulse signal wirelessly, or at the very least implement a fourier transform of the data to pick out valid human heart rates. Currently the minimum heart rate that is recognized is 20 BPM (if you have this, you should go to the hospital). The maximum heart rate is 200 (you are working very hard or are extremely out of shape and working very hard). This poses an issues that noise will destroy, or severely limit, the capabilities of the fourier transform, however it will be more accurate than the peak detection methods used currently.