Forward:
All of the software is stored on Github. The repos are listed in “Documents/GitRepos”. Most, if not all, of the software files have headers that contain author, date created, date of last revision, revision history, description and notes. For more information on individual files please refer to those headers and also to commit histories on github.

The Angstrom Kernel:
The BeagleBone, a Texas Instruments device, is used as a USB hub and GPS receiver on the board. It requires a microSD be flashed with an image of a compatible linux kernel. Instructions on how to get this is at BeagleBone.org. Once an image has been downloaded it can be sourced onto a microSD if the SD is plugged into a linux computer with the “dd if=<input file> of=<output file(sd card) bs=4M”. It may take a few minutes to copy. Unfortunately, that kernel version is not compatable with our system and must be recompiled.

The system needs more UART ports than are enabled by default on the BeagleBone. To add these and strip the kernel of dead weight it was recompiled using these instructions:
http://jazprojects.wordpress.com/2012/12/23/beaglebone-adding-the-missing-uarts/
The instructions above reference this post on how to clone the repo with the linux kernel 3.2.33ps which is open source and part of a development program Robert C. Nelson has been so nice to share with all of us followers.
https://groups.google.com/forum/#!msg/beagleboard/qZ1XCpUU814/Wb7ohWzNeyIJ
The new kernel is compiled using “make menuconfig” and a version of Busy Box. Follow the instructions in the above post which guides you from start to end.

BeagleBone Scripts:
To login to the BeagleBone you must ssh into the Ethernet port and share a connetion to it. By running nmap you can find the ip address of the system and then ssh into the root account. The password is “re.cycle”. The file transfer used in development was sftp and git. Git is the preferred method as it keeps everything saved in a remote server. If you choose to implement git, make sure no passwords are stored in files that are publicly viewable otherwise the system could get hacked.

BeagleBone scripts are split into several different folders. Here is an outline of those different folders and the modules inside them.

1)Comm Scripts: The two “SendXAVR.py” programs are used to send information from the BeagleBone to the corresponding AVR controller. The “SendGAVR” is activated by the “ReceiveGAVR” which replies to requests from the GAVR. The “SendWAVR” routine is activated when the time needs to be updated on the WAVR via GPS strings. “DebugGAVR.py” was used to test the GAVR and it’s trip functionality in the absence of an LCD. It communicates to the GAVR over Serial Port 5 and can send it appropriate strings to trigger trip actions.

2)NMEA: The two scripts in this folder contain the routines for all GPS data acquisition and

parsing. “myGpsPipe.py” is reponsible for opening Serial Port 2 and streaming NMEA strings from the GPS to a “CURRENT.txt” file in the BeagleBone’s file system. The “nmeaLocation.py” script is called when a trip is going to be offloaded to an external USB stick. It is responsible for parsing those NMEA strings and pulling out date, time, latitude and longitude data.

3)setupScripts: “bicycle.service” is a file that was placed in /lib/systemd/system to be started at boot time. The service calls another script, “firstCall”, then exits. “firstCall” is responsible for starting the watchdog and basic services of the BeagleBone. Three main services it starts is “mountUSB.sh”, which is responsible for detecting whether a USB was inserted to the BeagleBone and then mounts/unmounts the device accordingingly, “gavrInterrupt.c” whcih is responsible for detecting a connection request from the GAVR, and “shutdownInterrupt.c” which is responsible for detecting the shutdown signal from the WAVR. If the gavrInterrupt is found then the ReceiveGAVR.py script is executed in the CommScripts folder. If shutdownInterrupt is pinged, the system executes the “halt” command. “pinSetup.sh” is called by firstCall to setup the GPIO lines and UART lines needed for the system. It implements the /sys/kernel/debug/omap_mux directory and changes modes. For more information on modes, go to BeagleBone.org.

ATMEL uC Develpment:

To work with the two microcontrollers on the PCB, an ATmega2560 and ATmega644PA, an ISP programmer is needed to connect to the corresponding headers (see hardware schematics) and code written for Atmel Microcontrollers. This project was developed in Atmel Studio 6. To download, visit Atmels website and follow the instructions on how to download. Once installed, it will set up a default directory in “Documents/Atmel Studio”. The code in SourceCode/* AVR can be copied into a new folder, then in Atmel Studio go to File->Open Project/Solution and browse to either the GAVR project or WAVR project. There is an abundance of doumentation on Atmel Studio on it’s website as well as great tutorials.

ATMEL uC Code:

The Graphics controller code is one program that implements different classes and headers. The classes involved are “myDate”, “myTime”, “odometer”, “heartMonitor”, “trip” and “myVector”. A flow diagram is in the folder containing this document titled “Program Flows”.

1)”myDate.h” and “myTime.h”: These two classes are responsible for keeping an accurate time for the user to view. It has functions that set the time based on certain conditions like user changes or non-valid times sent from the WAVR.

2) “gavrUart.h”, “eepromSaveRead.h”: These two include headers contain protocols for doing excactly what their names suggests. The “gavrUart” contains implementations for how the GAVR communicates over UART to the WAVR and BeagleBone while “eepromSaveRead” deals with all read and write protocols to the controllers EEPROM.

3)”stdtypes.h” and “ATmega2560.h” contain definitions of bit assignments and variable types used in GAVR_reCycle.cpp and included modules

4)”GAVR_reCycle.cpp” uses all of the include headers and is the main program for the GAVR.

The Watchdog controller code is also one program that implements the same date and time classes that the GAVR utilizes.
1)”myTime.h” and “myDate.h”: The same two classes used in the GAVR. Responsible for keeping accurate date and time.
2)”wavrUart.h”, “eepromSaveRead.h”: The “wavrUart” file contains all implementations of WAVR communication protocols to and from the WAVR over UART. “eepromSaveRead” contains implemenations of writing and reading from the EEPROM. It is important to note that the WAVR and GAVR files are based off one another, but are NOT identical.
3)”stdtypes.h” and “ATmegaXX4PA.h”: “stdtypes.h” is the same file used in the GAVR. “ATmegaXX4PA.h” contains declarations for any mega microncontroller with the trailing letters XX4PA, in our case the 644PA.
4)“WAVR.cpp” is the main program file and is used to monitor battery conditions/power and keep a RTC for the system.