Development Plan Deliverables Report
ECE 5770
Microcomputer Interfacing

Cody Herndon

May 4, 2015

Forward

The Remote Rover Training System is intended to be a lightweight, inexpensive training platform for scientists and engineers to experiment with various aspects of real-time operating systems, and system implementation and integration. From hereon in, this project will simply be referred to as "the Rover".
All code for this project, as well as pertinent documentation, can be found in the project's github account.
Sections 1.1 Determine the Process Model to be Followed, 1.2 Requirements, and 1.3 Create Tests and Hardware/Software Design were all completed in earlier exercises which were submitted to canvas throughout the semester. These assignments may be referred to throughout the document, particularly in section 1.5 and section 1.10 where software implementation and full system testing are to take place.

1.4 Review Hardware

This exercise was completed in class as a peer review was conducted between my classmates and I.

1.5 Implement/Test Hardware

Hardware was constructed using the Tiva-C development kit[6], ESP8266 WiFi board[4], and the NEO-6 GPS module[7].
Further discussion of the construction of the hardware is discussed in Section 1.7, Implement Software, as the hardware interconnection was rather simple, the hardware implementation was driven by the software's development.

1. Hardware subsections

The hardware implementation was divided into 5 major subsections:

a. Tiva-C

b. ESP8266

c. Chassis

d. Power Supply

e. NEO-6

2. Test each subsection and record results

Each subsection of the hardware was implemented and then tested with software as described in Section 1.7.

3. Fix any flaws that you uncover

Each subsection of the hardware did not require any major revisions.

4. Redraw your corrected Schematics

Minor revisions to the schematic are outlined in the following figures.

Figure 1: Initial schematic draft

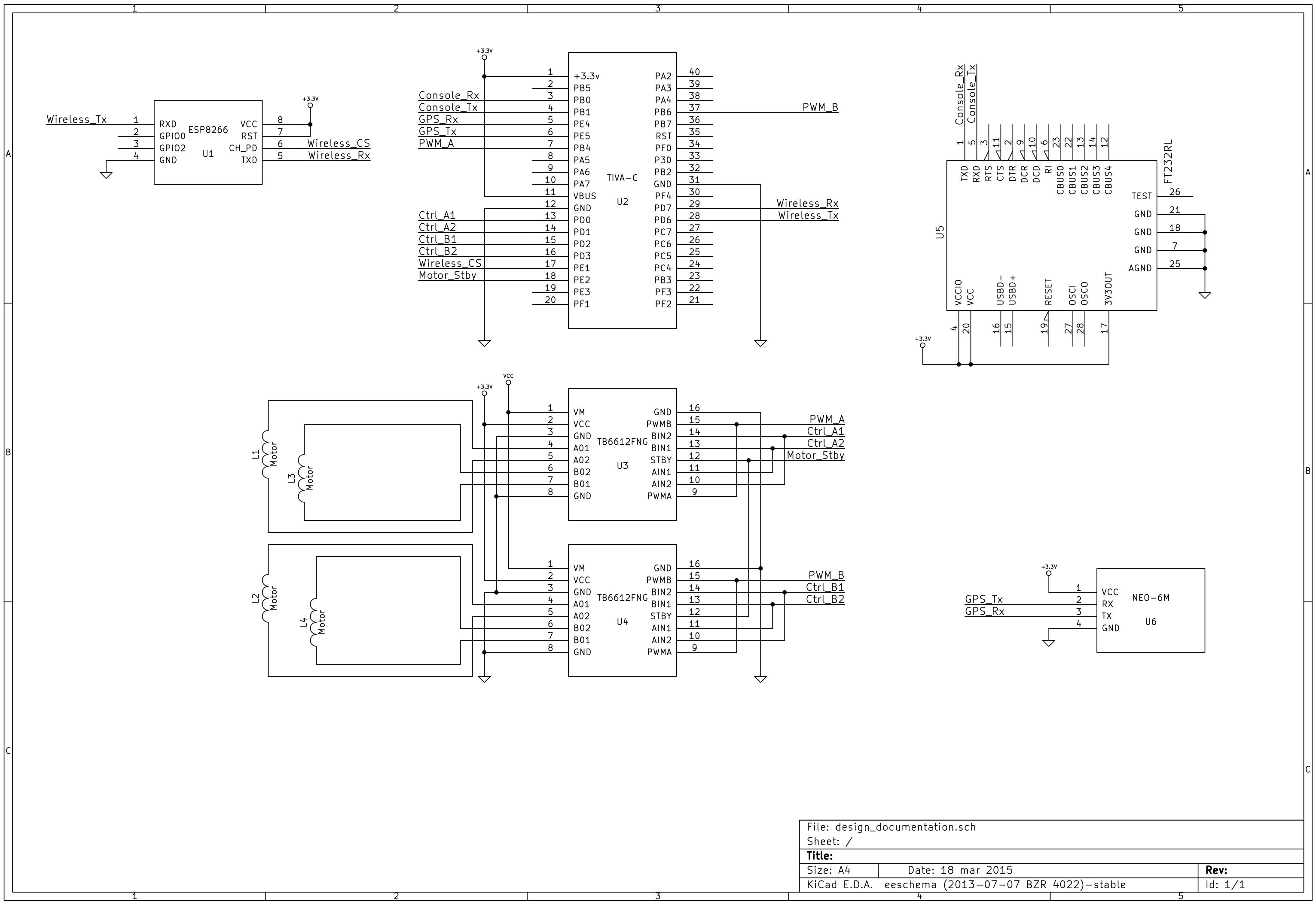
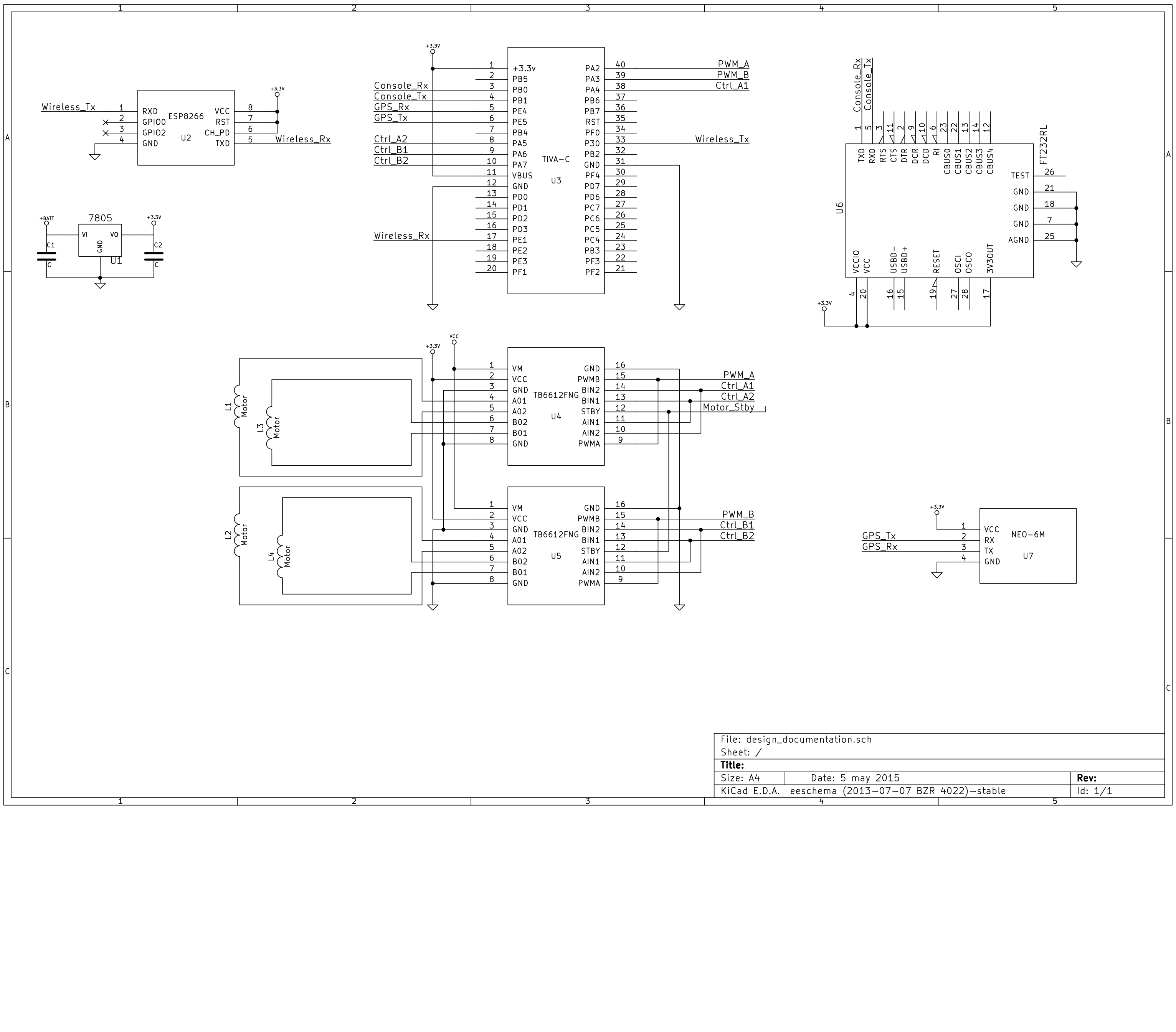


Figure 2: Revised schematic draft.



5. Measure Powers

State	Power Consumption (W)
Start-up	0.375
Idle	0.3
Motion	7.2

1.6 Review Software Design

1. Review your software design with a classmate.

This exercise was done in class. The classmate found the design to be satisfactory.

2. Make any adjustments as necessary.

No major adjustments were made during the course of this software design review.

1.7 Implement Software

Cycle 1

Cycle 1 focused on implementing the core kernel code and the core system architecture for the Micro-Kernel real-time system.

Design

The system design for the core kernel was that of a Micro kernel[1]. The micro kernel architecture was selected due to its inherit simplicity and compactness. It was entirely possible that this software system could be pushing the limits of the Tiva-C's code size and computational speed limits, so a high-speed micro kernel was selected for the core. The Real-time scheduling portion of the system was designed to be far more complicated than the final implementation, but the system was finally implemented using a simple RM or RR schedule with tasks being registered upon the initialization of the system.

Develop

Development of the kernel was implemented using strict c, as was the remainder of the software, using the Modular Programming in C[2] tutorial as a guide. Final implementation can be found at the github site.

Test

The kernel was implemented in such a manner as to be tested on an x86 desktop, and thus is as platform agnostic as possible. The program can be made on any platform with the required dependencies, namely the cunit test suite and gcc. Unit testing confirmed the functionality of the system.

Cycle 2

Cycle 2 focused on porting the kernel to the Tiva-C development board and is thus implemented using TI's development software and the *arm-none-eabi-gcc* compiler. This cycle completed with the successful execution of the kernel code on the Tiva-C board.

Design

the system design is unchanged from the previous cycle, but uses an LED blink function embedded in the kernel code to indicate successful code execution. Further confirmation was performed with the *arm-none-eabi-gdb* program.

Develop

Porting the system was completed relatively quickly. The biggest dilemma in this system port was working with the gcc cross-compilers and makefile. Results of the implementation for this section can be found in the github site.

Test

The system was implemented to run on the Tiva-C board, and thus was more difficult to debug. The system was tested and confirmed working with a hardware pin indicator, blinky LED, and gdb tracing.

Cycle 3

Cycle 3 focused on implementing UART hardware drivers to communicate with the workstation computer to ensure easier debugging, and later interface to the WiFi and GPS peripherals.

Design

The implementation of the UART driver was completed in a similar manner to the implementation of the functions used to debug the kernel. UART driver API for the Tiva-C was described in the documentation accompanying the software libraries for the device[5].

Develop

Development proceeded smoothly, as the API given in [5] was clear and concise. This development cycle completed with the successful echo of the debug input from the PC to the Tiva-C and back to the PC. The results of the development process can be found in the github account.

Test

Testing for this cycle concluded when the UART could successfully and easily echo input from the console out to the computer without dropping characters or requiring represses.

Cycle 4

Cycle 4 focused on implementing the WiFi driver and controller functionality. The WiFi driver functionality was to detect a reception of a newline and forward the results to the controller, thus abstracting the driver implementation away from the controller. The controller was to then implement a simple state machine, detect certain keywords to trigger a state change, and send corresponding responses to the WiFi module and main controller or motor controller.

Design

The controllers and drivers implemented here were intended to be modeled on the dummy function from Cycle 1. The drivers were implemented in this fashion to avoid a deadlock or starvation of the other tasks.

Develop

Development proceeded with some hitches. The Kernel needed increased capabilities in order to facilitate string parsing for the state machine to function properly. The system is otherwise largely unchanged. The results of this development cycle can be see in the github account.

Test

Testing for this cycle concluded when the system was capable of communicating with the ESP8266 and parsing commands from its interface.

Cycle 5

Cycle 5 focused on implementing the motor drivers to enable movement with the Multi-Chassis kit[3]. The Power and chassis hardware subsystems were implemented here.

Design

Design again focused on the simple task developed for Cycle 1. The motor driver receives messages from the WiFi controller and then commands the Chassis subsystem to move based on commands received from the ESP8266.

Develop

Development went smoothly for this cycle, as the motor driver turned out to be comparatively simple to implement. See the github for the result of the implementation.

Test

Testing concluded when the Motor driver was able to take commands from the WiFi controller and translate those commands successfully into motion on the Multi-Chassis kit.

1.8 Review Software Implementation

1. Review your software with a classmate.

Software was reviewed at the conclusion of each cycle with the members of the group who were not responsible for the development of software.

2. Make any adjustments necessary.

One patch was required for the WiFi controller system as outlined in the github following Cycle 5.

1.9 Test Implementations of Software Modules

Various implementations of the software were tested and debugged during the development cycles outlined above. Refer to the github for patch notes and development notes.

1.10 Test Full System to Requirements

Some tests were not able to be performed, as the system was not completed to the full specifications outlined in the Customer Requirements Documents. Thus, the following table outlines the successes and failures of the system to date.

Item	Requirement	Test	Met
2.1	50 lbs weight limit	Place the rover on a scale such that all its weight is supported on the scale. The scale should report no more than 50 lbs.	Y
2.2	dimension limits	Measure 3 feet in each dimension, the System should not project beyond the measurements in that dimension.	Y
2.3	accessibility requirements	Observe that if the System encloses its electronics that the electrical system can be accessed through a single access panel	Y
2.4	drive requirements	Command the System to travel forward 10 ft on carpet of 1/2 inch thickness.	Y
3.1.1	battery capacity	Observe that the manufacturer rates the installed battery at the required capacity.	Y
3.1.2	battery rechargeability	Observe that the manufacturer denotes that the installed battery can be recharged.	N
3.1.3	idle endurance	Activate the System and observe that it is capable of operating for at least one hour.	Y
3.1.4	active endurance	Activate the System and transmit a command to traverse forward. Observe that the System is capable of continuing to travel for 10 minutes.	Y
3.2.1	remote control	Observe that the System is capable of receiving commands and sending data without being physically connected to the base station.	Y
3.2.2	additional hardware	Observe that the base station is not attached to hardware that is not commercial off the shelf to communicate with the System.	Y
3.2.3	command range	Place the System 50 feet from the base station. Send a command from the base station to the System, observe that the System responds.	Y
3.2.4	response range	Place the System 50 feet from the base station. Observe that the System responds.	Y
3.2.5	update range	Place the System 50 feet from the base station. Attempt to update the System's software. Observe that the System is properly updated.	N
3.3.1	estimation accuracy	Activate the System and command it to travel forward 10 ft. Observe that the recorded distance is within 1 ft of the distance traveled.	N

3.3.2	battery power	Activate the System. Observe that the system reports an estimate of remaining battery power to within 10% of the manufacturer provided model of the battery's power.	N
3.3.3	motor power	Place the System in such a way as the motion system is able to travel freely. Activate the System and command it to travel forward. Observe that the power dissipated by the motors as measured by a test equipment is within 10% of the dissipation reported by the System.	N
4.1.1	speed limit	Activate the System on a flat, smooth surface and command it to travel forward 10 seconds. Measure the distance traveled and ensure that the system travels at no more than 3 mph.	Y
4.1.2	rotation speed	Activate the System on a flat, smooth surface and command it to rotate 360 degrees. Record the amount of time required to rotate, ensure that it is no less than 6 degrees per second.	Y

Conclusion

The final system developed is very nearly the system described in the proposal document. Unfortunately, time and resource constraints have caused the project to fall behind schedule. A few more development cycles to fully implement the GPS controller and extend the functionality of the WiFi controller to be able to POST data to a website would have fulfilled all the outlined requirements. Furthermore, a simple change in battery source from 1.5v AA batteries to a rechargeable Li-Ion or NICD battery pack would have fulfilled requirement 3.1.2. A final demonstration of the system in action can be found on YouTube.

References

[1] collaborative. Microkernel. http://en.wikipedia.org/wiki/Microkernel, May 2015.

[2] John R. Hayes. Modular programming in c. http://www.embedded.com/design/prototyping-and-development/4023876/Modular-Programming-in-C, November 2001.

[3] Sparkfun.com. Multi-chassis - 4wd kit (basic). https://www.sparkfun.com/products/12089, November 2013.

[4] Sparkfun.com. Wifi module - esp8266. https://www.sparkfun.com/products/13252, April 2015.

[5] TI. Sw-tm4c-utils-ug. pages "1–188", October 2013.

[6] TI. Ti launchpad. http://www.ti.com/ww/en/launchpad/community.html, May 2015.

[7] U-blox. Neo-6 data sheet. https://www.u-blox.com/images/downloads/Product_Docs/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf, May 2011.