

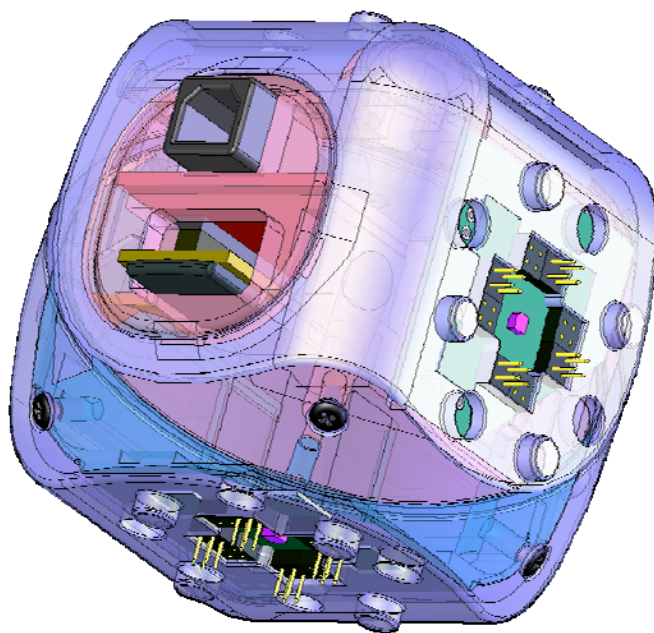
CORNELL COMPUTATIONAL SYNTHESIS LAB

Molecubes Controller

Semester Final Report

Phelps Williams

12/15/2007



Abstract

The Molecubes project aims to develop an open modular robotic framework envisioned as an expandable, robust, and low cost alternative to a variety of specialized robots with fixed body structure and functions. The project aims at popularizing the modular robotics research among robotics researchers, hobbyists, and enthusiasts by making all designs available to the public and bringing down the hardware cost. The Molecubes controller enables sequencing of commands used for physical execution of simulated movements.

Goals

This segment of the Molecubes project specifically aims to design a controller capable of executing command sequences. These command sequences are specified in a standard format defined as a portion of this project. The output of the Molecube simulator is stored in this format and is then interpreted by the controller. Provided the physical Molecube topology is configured in parallel with the simulation, the specified sequence can be executed and compared.

Background

Modular robots have been originally envisioned as a universal, robust, and low cost alternative to a variety of specialized robots with fixed body structure and functions. Unfortunately, even after two decades of study in this field, most researchers agree that these advantages are yet to be fully realized. A serious problem that hinders progress in this area is the prohibitively high cost of fabrication and operation of known modular robotic systems. This limits the modular robotic community to only few specialized labs at select universities. Another progress hindering problem is this prohibitively high expertise demands in a variety of engineering fields and computer science.

The completed Molecube modular robotic system will consist of an array of specialized modules such as a gripper, wheel, battery, structural monolithic modules, swiveling actuators, and the controllers that coordinate the actions of each. The controller has direct communication with all the Molecubes present in a robot configuration or cluster. The controller also is capable of a link to a host computer which will be able to select between three primary modes of operation. The first mode is command sequence execution, the second, sensor data collection, and the third direct Molecube control. In the final mode, commands can be sent directly to specific cubes.

Controller Design

General

Before delving into the realm of elements directly related to the objective of the project, it is important to note a few accomplishments related to project framework and documentation.

Specifically, SourceForge¹ has been used to establish an online code repository for project engineering files such as board designs and software. The services provided by SourceForge remove concerns related to distribution of generated materials. Furthermore, the project wiki² has been maintained with a variety of design and implementation notes relevant to a third party interested in building their own Molecubes.

Controller Software Design

The controller software performs three primary tasks, simulated sequence execution, cluster state sensing enabling determination of the physical state of the robot, and finally real time control. In the final mode the controller serves as a bridge between software on a host computer and the Molecubes. Physical state determination is briefly discussed in this report though much remains to be defined in this area. The primary area of development over the course of this semester has been in the simulated sequence execution segment which is a key element in the success of this project.

Before working on the controller development directly, effort was placed in defining the standard protocol to be used between cubes and between the controller and the host computer. With these standards in place, performance requirements could be flushed out further. First to be completed was the Molecube Hardware Protocol specification. This protocol provides a packet format, hardware interface, and command / response framework (Appendix A & B).

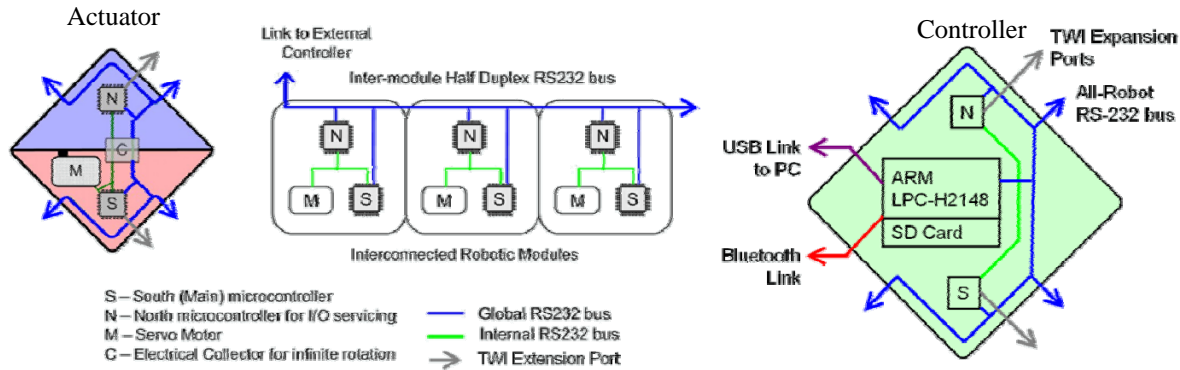


Figure 1 Communications diagram for the controller and actuator Molecubes

Dynamic addressing was another framework challenge which was approached early in order to avoid fundamental conflicts later in the design. This algorithm also allows for orientation sensing and therefore allows for the generation of an entire Molecube topology representation. A simple breadth first search is used to expand outward from the root of the search tree (the controller). Addresses are assigned as each node in the tree is discovered along the modified

¹ <http://sourceforge.net/projects/molecubes>

² <http://molecubes.org/>

linear RS232 bus shown in Figure 1. Once this tree has been established, a specially designed data structure is used to represent this tree between the controller and the host computer.

The file describing a particular configuration of cubes consists of a series of connectivity record entries as shown below. Two entries exist for each connection between two cubes, one from the perspective of each cube. This allows for errors to be detected as inconsistencies between entries.

Connectivity Record Entry					
Address	Half	Board	Orientation	Actuator Joint Angle	Null
8 bits	1 bit	2 bits	2 bits	10 bits	1 bit
24 bits					
Address	0x01 through 0xFD				
Half	Master = 1, Slave = 0				
Board	Main = 00, Right = 01, Left = 10				
Orientation	Local pin location connected to pin 1 of neighbor				
Actuator Joint Angle	Range 0 - 1023 representing angle 0 - 359				

The organization of the Master, Left and Right faces of a particular cube relate to the circuit boards which are identified similarly. Additionally, the orientation pin numbering scheme is rotated between faces in an attempt to avoid confusion. A drawing of the cube face / pin numbering can be seen in Figure 2.

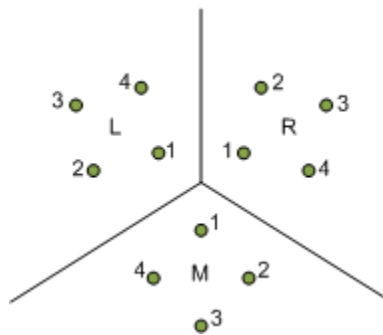


Figure 2 Cube Half Top View with Pin Numbering

Additionally, the state of this tree (represented by the details shown in the connectivity record entry above) can be updated upon request allowing the computer to maintain a nearly real time representation of the Molecube cluster. This addressing technique was inspired from the third version of the stochastic modular robotic system.

The data structure describing a time based sequence of movements has also been established (Appendix C). This data structure will be transferred from the host computer to the controller.

Three transfer techniques are envisioned, directly via USB connection, exchanged via a shared memory device, and wirelessly via a Bluetooth connection. The PC will then have the capability to play, pause or jump to a particular point in a sequence. There are some outstanding questions whether the format used is optimal for the controller. There is more flexibility in building a populated data structure on the host computer due to nonexistent memory constraints. This will be an area of further definition in the next semester.

The Molecube sequence file (*.msf) is generated by the PhysX simulation running on the host computer. This file maintains a simple header describing global parameters about the sequence such as length and the Molecube topology it is associated with. Following this header, commands compliant with the command specification document are combined with a timestamp indicating the time of execution. From this file the controller builds a memory structure consisting of timestamp ordered queues of commands. The actual runtime is quite simple, it simply needs to dequeue and execute commands when the current timestamp is less than or equal to the timestamp of the head of a particular queue. This action is continuously applied to each cube queue until no commands remain in any queues.

Revision 1 Design

The revision 1 controller software design utilized the command sequence specified by the simulation software on the host computer. The provided sequence is reformatted into sequences unique to each cube. In addition to maintaining a simple node state, the following algorithm ensures each node has its next upcoming command preloaded. This process can be observed in Figure 3.

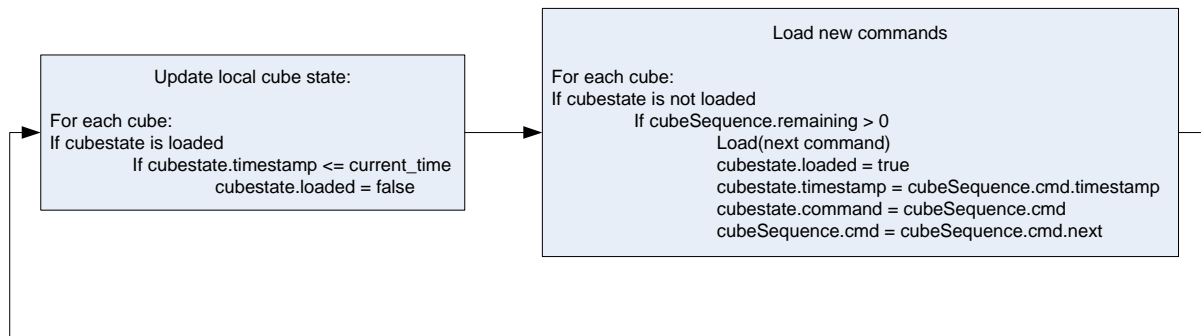


Figure 3 Runtime Algorithm Overview

A heartbeat command is globally transmitted regularly at a 10ms period. When a cube receives this heartbeat, if the timestamp matches that of the preloaded command, the command is executed. This simple approach should be robust and straight forward to implement. Many commands in rapid succession can result in a missed execution as a side effect of this approach. This must be prevented in the provided command sequence from the host computer.

This phase of development was plagued by serial communication configuration issues onboard the NXP ARM processor. Eventually after contacting NXP technical support, these issues were resolved. A side development effort during this period entailed using existing libraries to support a FAT32 file system on the memory card. This allows movement sequences and the resulting logs to be treated as files. This memory card can be plugged into a typical card reader in a computer to transfer sequences or log files. The significant amount of space available on this memory card allows the storage of a library of command sequences onboard the controller. Users can then interchange them quickly on demand.

Using the FatFS library³ the functionality to recognize the presence of a card, mount the drive, view all files and directories on the card, and read files was accomplished very quickly. Writing and appending to files caused an odd series of errors which took several weeks to resolve. This resolution is discussed in the Revision 2 Design section.

A generic packet building library has been written and successfully tested on both the Internal and External RS232 buses which are used to communicate with other connected cubes. This set of functions should provide everything necessary for broadcasting a command sequence. This library supports building the specific packet and manipulating the half duplex RS232 control circuitry at data rates up to 1Mbps.

Revision 2 Design

Development of the controller runtime marked the beginning of the second design revision. The runtime was more difficult to test because both the Molecube actuator software and the controller runtime software were developed in parallel. To mitigate this risk, the controller software was developed independently of the controller hardware as much as possible. The general algorithm could more thoroughly be tested outside of the restricted development environment onboard the ARM processor.

The intention of building this algorithm test environment was to verify the binary file parsing software and the associated data structures. Because this entails many arbitrary length components, memory referencing and dereferencing is used throughout. A common indication of incorrect usage is a segmentation fault where the program attempts to access memory in an unintended manner. Fixing these errors is trivial but only if properly detected. In the controlled desktop environment, detection is very simple. On the controller hardware itself, this is dramatically more difficult. This approach worked very well and the errors that arose on the desktop simulation were easily corrected. When the software was then ported over to the ARM development environment the algorithm worked flawlessly.

The ability to verbosely log runtime information shows great benefit although it poses a performance hit. Additional testing is necessary to form a rudimentary cost / benefit analysis.

³ http://elm-chan.org/fsw/ff/00index_e.html

Dynamic memory allocation will also be used which results in much more flexible software design at the cost of running out of memory if it isn't handled carefully. In longer more memory intensive sequences problems are more probable. Dynamic memory allocation is most valuable when the exact number of cubes and associated commands are unknown. For example, using this approach, it is possible to have 4 cubes each with 25 commands in their own sequence. Without any software modifications, it would also be possible to support 10 cubes each with one command and a single cube with 100 commands. A static approach would place limits, likely very low, on these values such as a max of 10 cubes with no more than 10 commands each. This would place difficult restrictions on the simulator and the types of testable Molecube topologies.

The runtime on the controller combines the functionality from many earlier weeks of development. The serial communications functionality issues commands to attached modules, the file I/O support allows sequence files to be read, and the runtime simulation provides the algorithm for linking these elements together. Finally, this is all running on the custom designed controller daughterboard discussed in the Controller Hardware Design section.

Controller Hardware Design

Controller Hardware design progressed in parallel with the software design process discussed above. The overall design benefitted significantly from this approach as various hardware decisions would impact software decisions and software difficulties could many times be resolved easily in hardware. This all started with establishing a hardware development environment for the ARM microprocessor consisting of a set of hardware libraries and a tool chain of software for compilation, loading, and debugging. Keeping the objective of low cost in mind, reliable, open source solutions have been found for every part of this process. The GCC compiler⁴ supports the ARM architecture and a Windows only free application called OpenOCD⁵ allows for loading and debugging. A commercial piece of software, Rowley CrossWorks⁶ for ARM is used in loading and debugging software used in the Fab@Home project is also being used here with success.

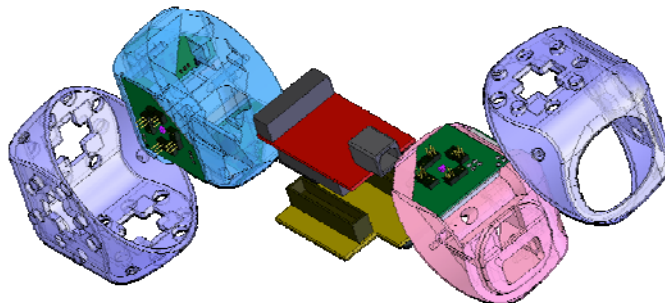


Figure 4 Exploded view of controller Molecube

⁴ <http://gcc.gnu.org/>

⁵ <http://openocd.berlios.de/web/>

⁶ <http://www.rowley.co.uk/arm/>

Revision 1 Design

The first revision of the hardware design consists of the following primary design elements

- 18V -> 5V regulator
- Internal & External Half Duplex RS232 Switching Circuit
- Olimex LPC-H2148 Interface
- Molecube N and S wiring interface
- Bootloader activation switch

It was determined at this phase in development that it would be optimal to use an off-the-shelf Olimex LPC-H2148⁷. This piece of hardware builds upon in house knowledge already present with the Fab@Home project utilizing this device. It is plugged into a daughterboard providing application specific circuitry such as voltage regulators and the half duplex RS232 switching circuit.

The dimensions of the daughterboard have been defined so the assembly fits within a volume roughly the same as the motor in a standard Molecube actuator. The structure of the cube requires minor modifications removing support for the motor, slip ring, and bearing elements. Added features are the ability to easily open the cube and an opening for external access to the USB port and memory card. In the current design, USB is the only means of communicating directly with the controller.

Within a one week period of defining requirements for the hardware, it was designed, manufactured and tested. The RS232 communications buffer works as expected supporting two channels (internal and external) at 1Mbps. The power regulator performs above and beyond its requirements as well. A layout error resulted in the two sockets for the Olimex LPC2148 being separated by 1.110" instead of 1". This did not impact testing and was corrected in revision 2.

Revision 2 Design

The revision 2 design is more compact and includes a microSD memory card and a Bluetooth wireless communication module. Both the Bluetooth module and memory card overhang the daughterboard by a certain margin. The memory card must be accessible from the outside of the cube in order to interchange the device. The Bluetooth module contains an onboard antenna which has specific clearance requirements for optimal performance.

The microSD form factor and the larger SD/MMC memory card share a compatible protocol making them interchangeable. Until this point in the revision 2 design, all development had been on the larger SD/MMC cards. The microSD card is under 500mils in width making it perfect for this space constrained application.

⁷ <http://www.olimex.com/dev/lpc-h2148.html>

After the design process worked through the above issues, the board arrived and was assembled over the course of several days. The FAT32 support issues mentioned in revision 1 were resolved by scrapping the FatFS library and starting again with a new EFSL⁸ framework. EFSL performs the same FAT filesystem support as FatFS though it is implemented in a cleaner, more reliable manner.

Actuator Hardware Modifications

Over the course of the semester, a problem in the power regulation scheme for the Molecube actuators was discovered. This problem resulted in severely exceeding the thermal specifications on the existing 10V regulator which powers the servomotor. In order to keep the regulator within specification, the input voltage range and power consumption of the motor had to be severely limited. In order to avoid this limitation, the existing design was modified to include a switching regulator intended for the loads expected. This regulator has been built and is working as designed.

Tricolor RGB LEDs were added to each of the six sides of each Molecube. These devices are controlled via an I²C bus between the ATmega164P microcontroller and a Phillips PCA9633 4 bit RGB led driver. This component has also been built and is working as designed.

Actuator Software Modifications

The Molecube actuators are now being developed utilizing AVRlib⁹ which provides significant hardware support reducing the development time necessary to take advantage of many built in components on the ATmega microcontrollers. Utilizing these libraries, the highest priority area of development is now in translating the simulated runtime sequences into actuation command sequences. Essentially implementing the various commands and responses specified in Appendix A.

Conclusions

This segment of the Molecubes project specifically aims to design a controller capable of executing command sequences. Through one semester of working on this, this has been demonstrated under basic testing conditions. Many external dependencies remain such as completion of the simulation software and actuator software. Going forward, effort will be

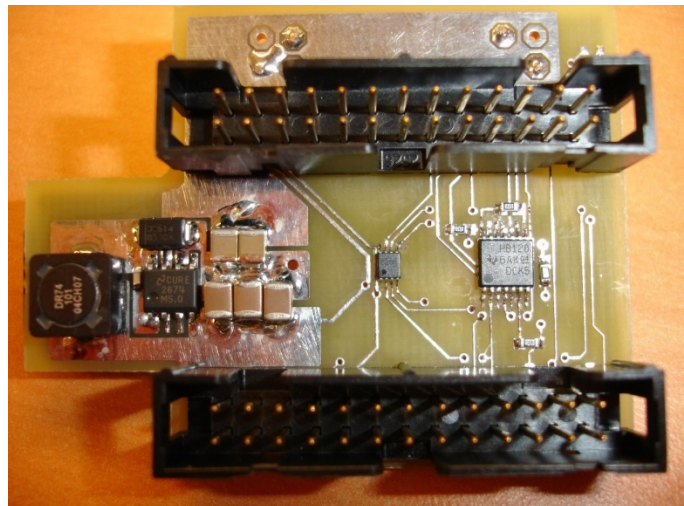


Figure 5 Revision 2 Daughter Board Top View

⁸ <http://www.efsl.be/>

⁹ <http://hubbard.engr.scu.edu/avr/avr-lib/>

focused on improving these external components allowing for focus then to be redirected back to the controller software with a framework for more complex tests to be run.

The hardware development process, with the exception of a few future work components listed below, is largely complete. Improving the runtime software in a more complex environment is the primary objective for the coming semester.

To reiterate more concisely here is an itemized list of completed project elements, elements in progress, and those planned for next semester:

- Controller Hardware Base Hardware Design – Completed
- Controller Hardware Base + Bluetooth Wireless – Spring Semester
- Controller Software Simple Sequence Execution – Completed
- Controller Software Complex Sequence Tests – Spring Semester
- FAT32 Filesystem support on SD Memory Card – Completed
- Hardware Communication Protocol (Controller <-> Actuator) – Completed
- Sequence File Specification (Host Computer -> Controller) – Completed
- Dynamic Addressing Support – In Progress
- Cube Orientation Determination Algorithm – In Progress
- USB Bootloader (Reprogram without an expensive JTAG device) – Spring Semester
- Smart Battery Pack Design – In Progress

Future Work

The following elements would ideally be completed during the spring semester. They are not on the critical path for the success of the project however. These elements are potentially interesting for exploration in the future.

Smart Battery Pack Design

The idea of a Smart Battery Pack design arose out of the idea that a cube cluster should be able to ideally operate unconstrained. Smart batteries are commonly found on laptop computers for example, the most important feature they have is a digital interface to gather battery charge state. Additional features such as charge and discharge protection, cell balancing, and temperature monitoring are common as well. After looking into the technologies required to implement such a thing, it is clear this should not be an overwhelming challenge. The battery capacity using Lithium Ion cells should be sufficient to operate a 6 cube cluster for at least 30 minutes under heavy load. The battery pack will fit into a Molecube form factor and will allow for direct monitoring of charge state. A preliminary charger design can be found in Appendix J.

Future Hardware Design Modification Ideas

Two or more external RGB LEDs and pushbuttons will allow for physical user interaction with the cube itself. A more compact controller design in general would be of great benefit due to increased packaging options. The primary limiting element in the hardware size is the Olimex LPC-H2148. The JTAG interface along with many unneeded board features are broken out. This flexibility comes at the expense of additional board real estate. The next step in improving this aspect of the design is to combine the functionality of the daughterboard (memory interface, communications, and power) with the LPC-H2148 which provides additional power regulation, the processor, and the support components necessary. Unused processor components would not be supported in the design cutting down on the design complexity in comparison with the LPC-H2148.

Appendix A: Molecube Instruction Listing

Name	Instruction	Parameter	Definition	Notes
Get Firmware Version	0x02	Instruction Parameters: 0 Bytes None	Response Parameters: 2 Bytes Firmware Version 16 bits	
Set Address	0x03	Instruction Parameters: 1 Bytes Address 8 bits	Response Parameters: 1 Bytes Address 8 bits	
Set Address with Orientation Pin†	0x04	Instruction Parameters: 1 Bytes Address 8 bits	Response Parameters: 1 Bytes Address 8 bits	Broadcast command, device which orientation pin high sets provided address and responds indicating address successfully set.
Set Address with Orientation Pin and Set Motor Master†	0x05	Instruction Parameters: 1 Bytes Address 8 bits	Response Parameters: 1 Bytes Address 8 bits	
Get Raw Angle	0x06	Instruction Parameters: 0 Bytes None	Response Parameters: 3 Bytes Global Position 10 bits Motor Position 10 bits Insensitivity Region 1 bit Filler 3 bits	
Get Measured Angle	0x07	Instruction Parameters: 0 Bytes None	Response Parameters: 2 Bytes Joint Angle 16 bits	Returns measured angle derived from information also accessible in Get Raw Angle command.
Set Orientation Pin	0x08	Instruction Parameters: 1 Bytes Side (M = 00, R = 01, L = 10) 2 bits Pin (0-3) 2 bits State 1 bit Filler (don't care) 3 bits	Response Parameters: 0 Bytes None	
Get Orientation Pins	0x09	Instruction Parameters: 0 Bytes None	Response Parameters: 3 Bytes Main (M0, M1, M2, M3) 4 bits Right (R0, R1, R2, R3) 4 bits Left (L0, L1, L2, L3) 4 bits Main DDR (M0, M1, M2, M3) 4 bits Right DDR (R0, R1, R2, R3) 4 bits Left DDR (L0, L1, L2, L3) 4 bits	Provides complete representation of current orientation pin state along with Data Direction indication (DDR) which indicates whether pin is an input or an output.
Is Orientation Pin Observed†	0x0A	Instruction Parameters: 0 Bytes None	Response Parameters: 1 Bytes Observed (boolean) 1 bit Side (M = 00, R = 01, L = 10) 2 bits Pin (0-3) 2 bits State 1 bit Filler (don't care) 2 bits	If observed is false (0) remaining bits in parameter byte are don't care
Get Motor Status	0x0B	Instruction Parameters: 0 Bytes None	Response Parameters: 10 Bytes Goal Position 16 bits Present Position 16 bits Present Speed 16 bits Present Load 16 bits Present Voltage 8 bits Present Temperature 8 bits	
Get Motor Register	0x0C	Instruction Parameters: 2 Bytes Starting Address 8 bits Number of registers to read 8 bits	Response Parameters: * Bytes Parameter 1 8 bits Parameter 2 8 bits Parameter n 8 bits	
Set Motor Register	0x0D	Instruction Parameters: * Bytes Starting Address 8 bits Parameter 1 8 bits Parameter 2 8 bits Parameter n 8 bits	Response Parameters: 0 Bytes None	
Set Motor Master	0x0E	Instruction Parameters: 0 Bytes None	Response Parameters: 0 Bytes None	The recipient of this command is expected to internally inform the other half of the molecube.
Preload Motor Profile	0x0F	Instruction Parameters: * Bytes Timestamp(msec) 32 bits Starting Address 8 bits Parameter 1 8 bits Parameter 2 8 bits Parameter n 8 bits	Response Parameters: 0 Bytes None	This allows for command preloading on the servo. Specifically utilizing the REG_WRITE(4-3-1) functionality. An ACTION (4-3-2) command is additionally preloaded and executed when a heartbeat of corresponding timestamp is received.

Preload Command	0x10	Instruction Parameters:	* Bytes	Response Parameters:	0 Bytes	
		Timestamp(msec)	32 bits			
		Instruction	8 bits	None		This is for preloading a command outside of the motor, such as a command listed here. Valid commands cannot have response parameters. An example usage of this command is Set LEDs.
		Parameter 1	8 bits			
		Parameter 2	8 bits			
		Parameter n	8 bits			
Heartbeat +	0x11	Instruction Parameters:	4 Bytes	Response Parameters:	0 Bytes	
		Timestamp	32 bits	None		Any preloaded command with matching timestamp is executed simultaneously.
Set LEDs	0x12	Instruction Parameters:	4 Bytes	Response Parameters:	0 Bytes	
		Filler (must be 0)	6 bits	None		
		Side (M = 00, R = 01, L = 10)	2 bits			
		Red	8 bits			
		Green	8 bits			
		Blue	8 bits			
Get Status	0xFE	Instruction Parameters:	0 Bytes	Response Parameters:	1 Bytes	
		None		Error Code	8 bits	Error codes have not been defined, only occurs in response packet, this instruction will replace the intended response instruction.
Notes:						
+ Valid for broadcast						
# Assumes default address upon startup of molecube module						

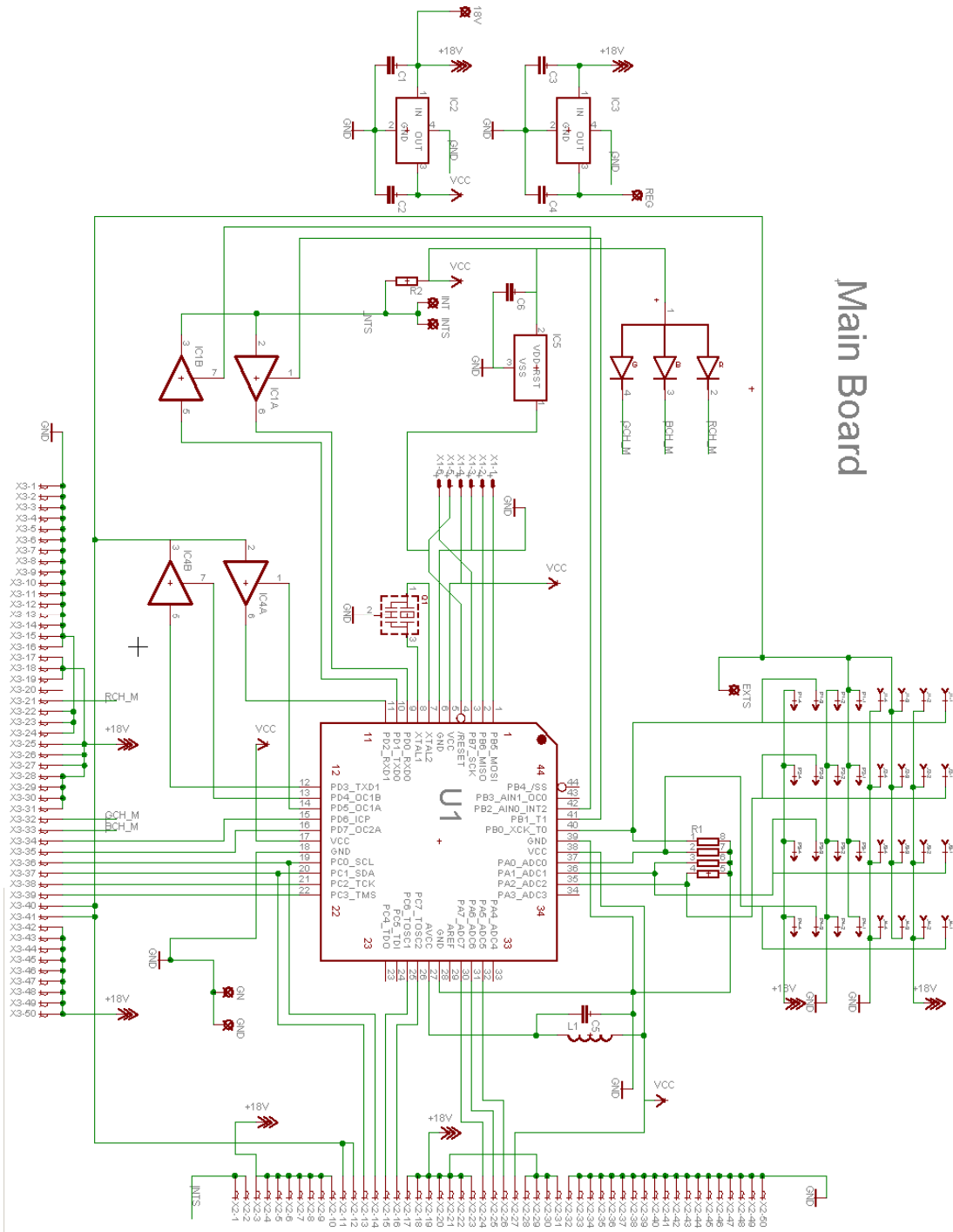
Appendix B: Packet Specification

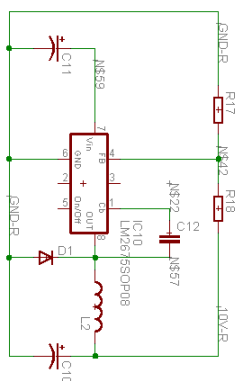
HEADER	CLASS	ID	LENGTH	INSTRUCTION	PARAM1	PARAMn	CHECKSUM
Name	Range	Description						
HEADER	0xFF	This is a single byte which identifies an incoming packet						
CLASS	0x00-0xFF	This is a single byte identifying the class of the target device.						
		Device Classes:						
		0xFF	AX-12 Servomotor					
		0xFE	Molecube North Controller					
		0xFD	Molecube South Controller					
ID	0x00-0xFD	Unique class address. There are 254 available ids for each individual class.						
		Class Broadcast Address: 0xFE						
LENGTH	0x00-0xFF	Parameter byte count. Maximum parameter count is 256 parameters per packet. It is important to note that the overall packet is as follows:						
		Overall Packet Length = 6 + Length						
		The overall packet length takes the header, class, id, length, instruction, and checksum into account.						
INSTRUCTION	0x00-0xFF	Instruction to perform						
PARAMETER0...n	0x00-0xFF	Parameters specific to each instruction						
CHECKSUM	0x00-0xFF	A simple 8bit checksum value. Calculated as follows:						
		Checksum = \wedge (ID + LENGTH + INSTRUCTION + PARAM1 + ... PARAMn)						
		Checksum value is 8bits in length and should rollover upon reaching its maxima.						

Appendix C: Molecubes Sequence File (*.msf) Specification

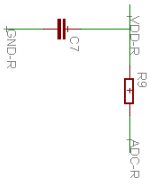
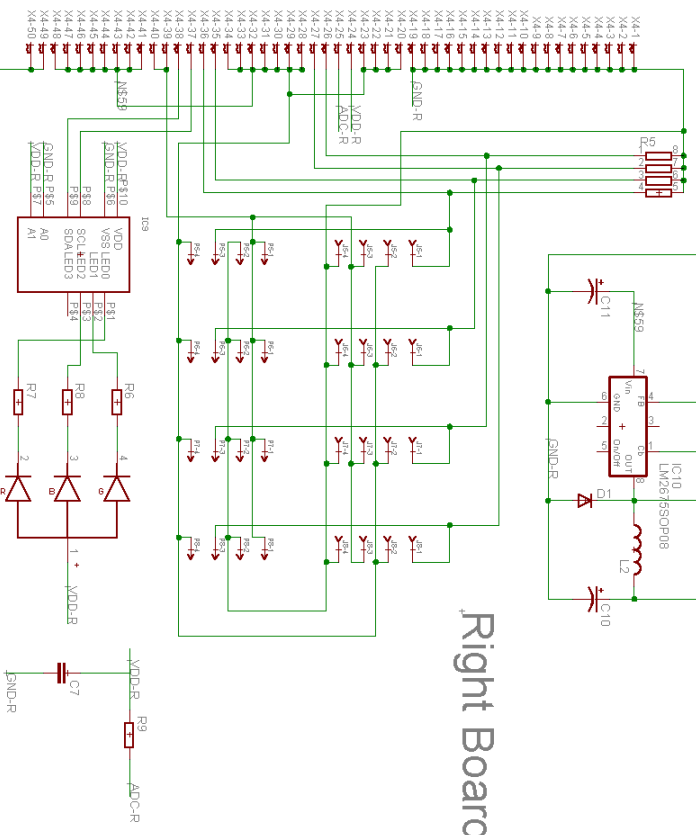
Name	Variable Type	Description	Example (base ₁₆)	Example Decoded
Sequence Name	char array size 25	Human Readable identifier of the command sequence	54 65 73 74 20 53 65 71 75 65 6e 63 65 00	Test Sequence
Topology Hash	unsigned int (32bit)	Links sequence to hardware configuration. LSB must equal number of cubes, format of 3MSB has yet to be specified and isn't	02 00 00 00	2
Command Count	unsigned int (32bit)	Total Number of commands in this particular sequence	04 00 00 00	4
Size	unsigned int (32bit)	Size of all Commands in the sequence, not including this header	34 00 00 00	52
Header Checksum	unsigned char (8bit)	Equals Name, Topology Hash, Command Count and size, summed and inverted(~)	fd	
Cmd0 Timestamp	unsigned int (32bit)	Time in milliseconds of execution	0a 00 00 00	10
Cmd0 Bus	unsigned char (8bit)	0 (Internal Bus), 1 (External Bus)	00	0 (Internal Bus)
Cmd0 Header	unsigned char (8bit)	Always 255	ff	
Cmd0 Class	unsigned char (8bit)		ff	255 is AX12 Motor
Cmd0 ID	unsigned char (8bit)	Device Address	01	
Cmd0 Length	unsigned char (8bit)		02	
Cmd0 Instruction	unsigned char (8bit)		03	
Cmd0 Param 0	unsigned char (8bit)		19	
...	...			
Cmd0 Param n	unsigned char (8bit)		00	
...	...			
Cmd N Timestamp	unsigned int (32bit)			
Cmd N Bus	unsigned char (8bit)			
Cmd N Header	unsigned char (8bit)			
Cmd N Class	unsigned char (8bit)			
Cmd N ID	unsigned char (8bit)			
Cmd N Length	unsigned char (8bit)			
Cmd N Instruction	unsigned char (8bit)			
Cmd N Param 0	unsigned char (8bit)			
...	...			
Cmd N Param n	unsigned char (8bit)			

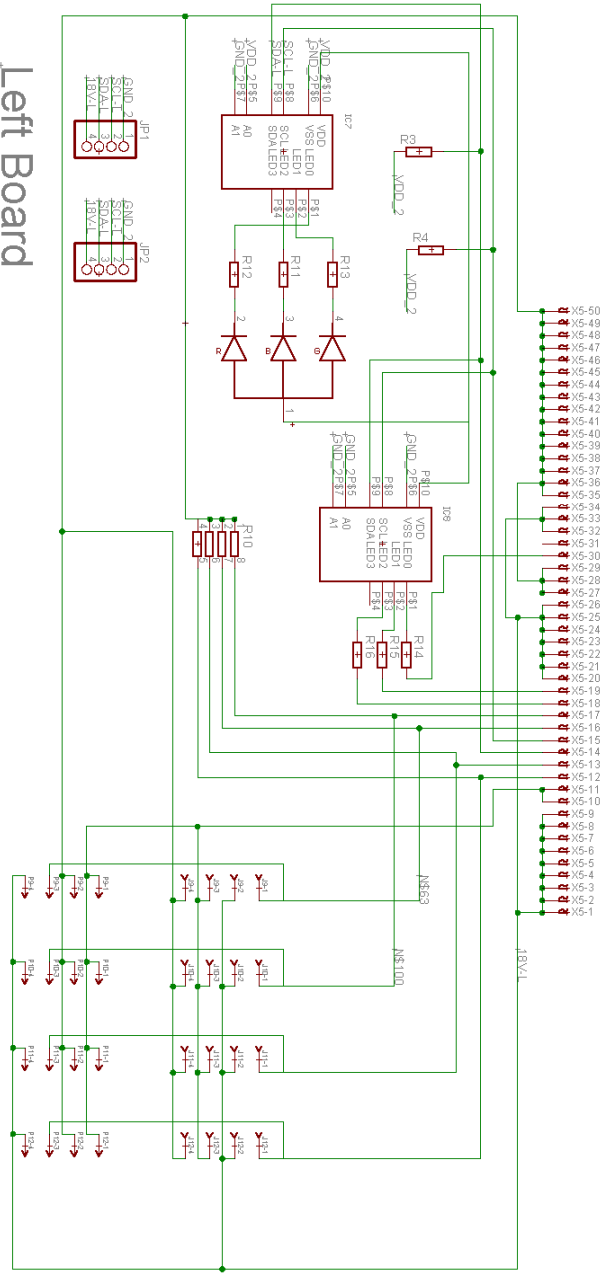
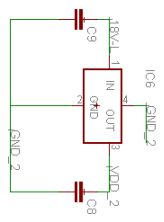
Appendix D: Actuator PCB schematic





Right Board

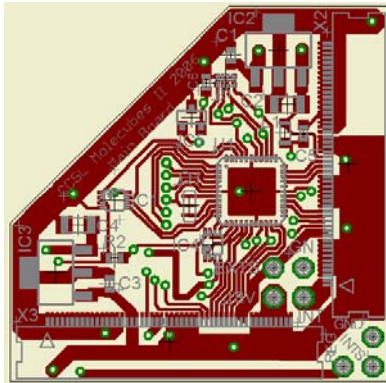




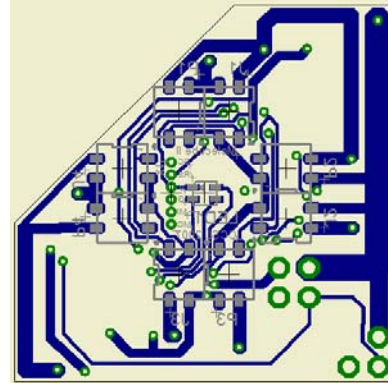
Left Board

Appendix E: Actuator PCB board design

Main Board

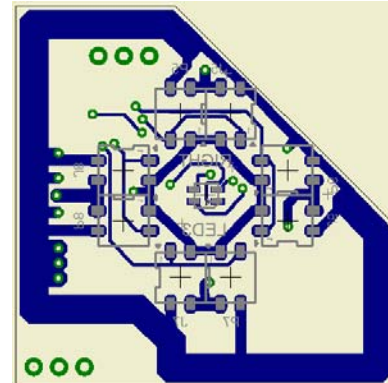
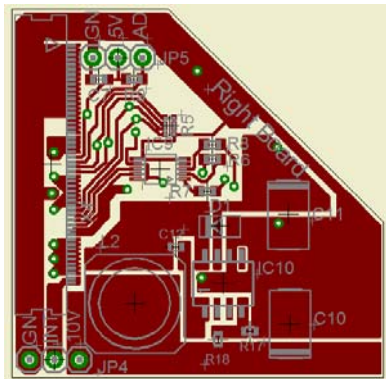


Top

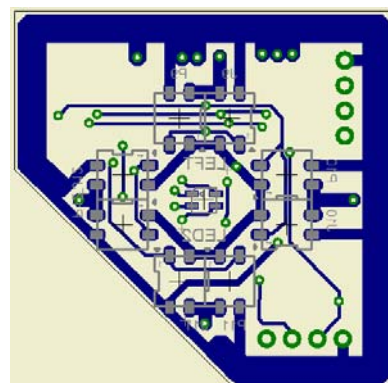
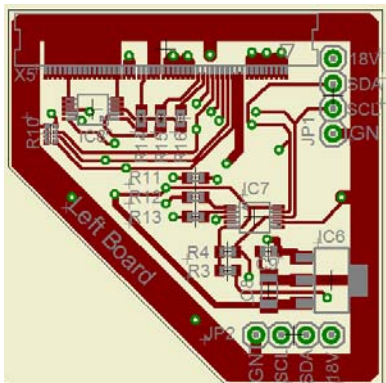


Bottom

Right Board



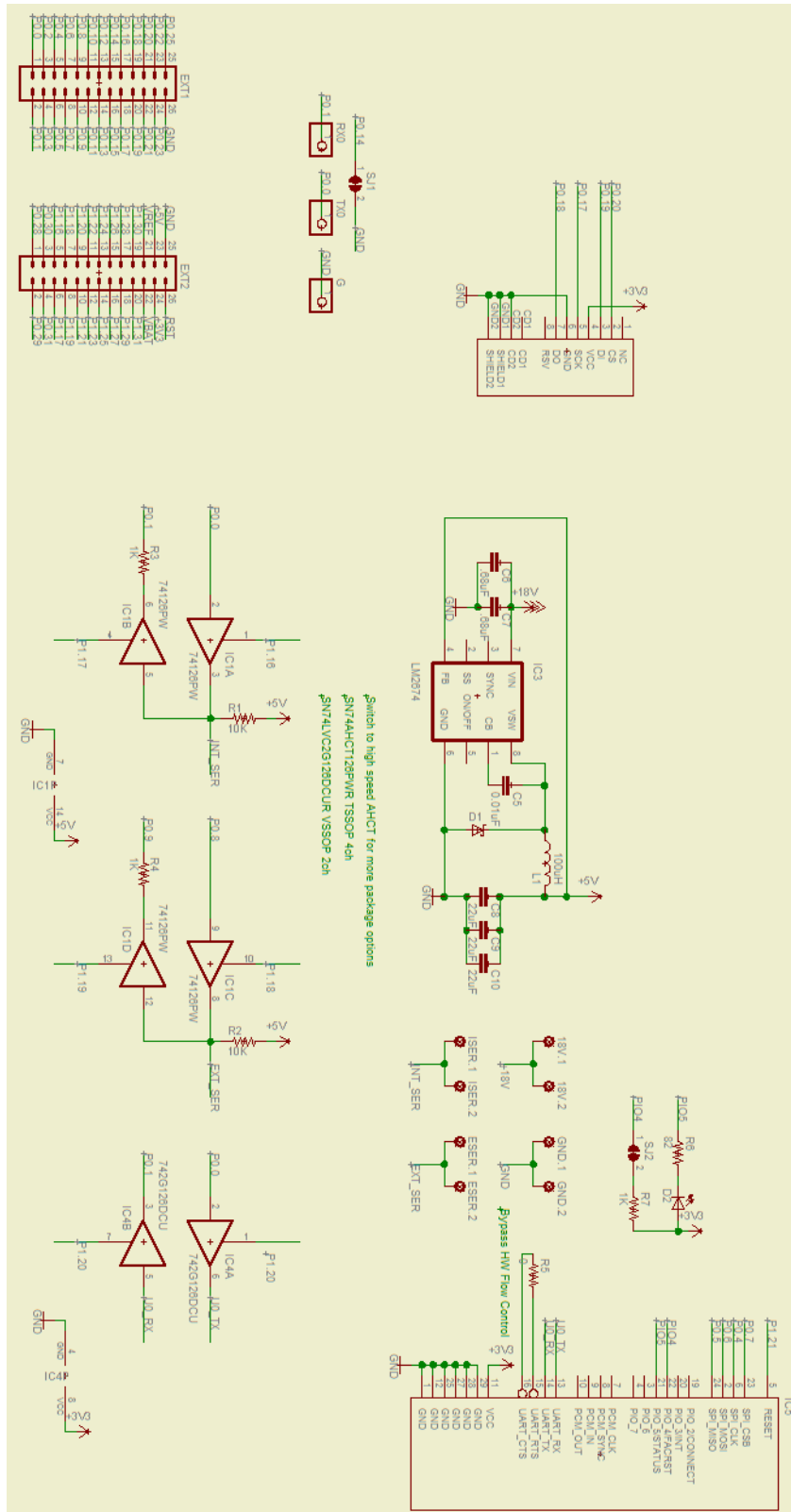
Left Board



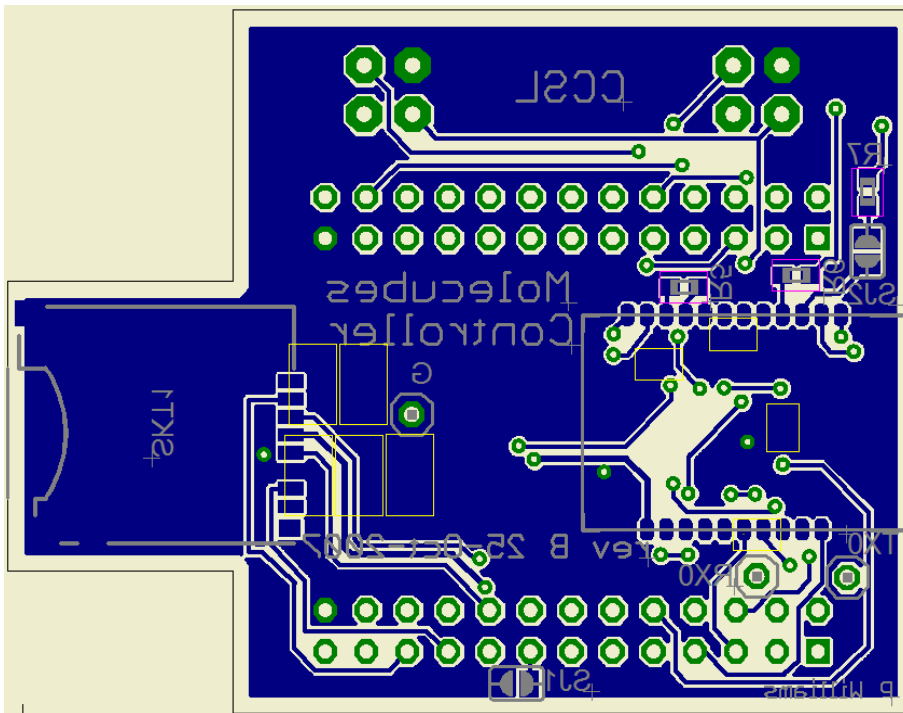
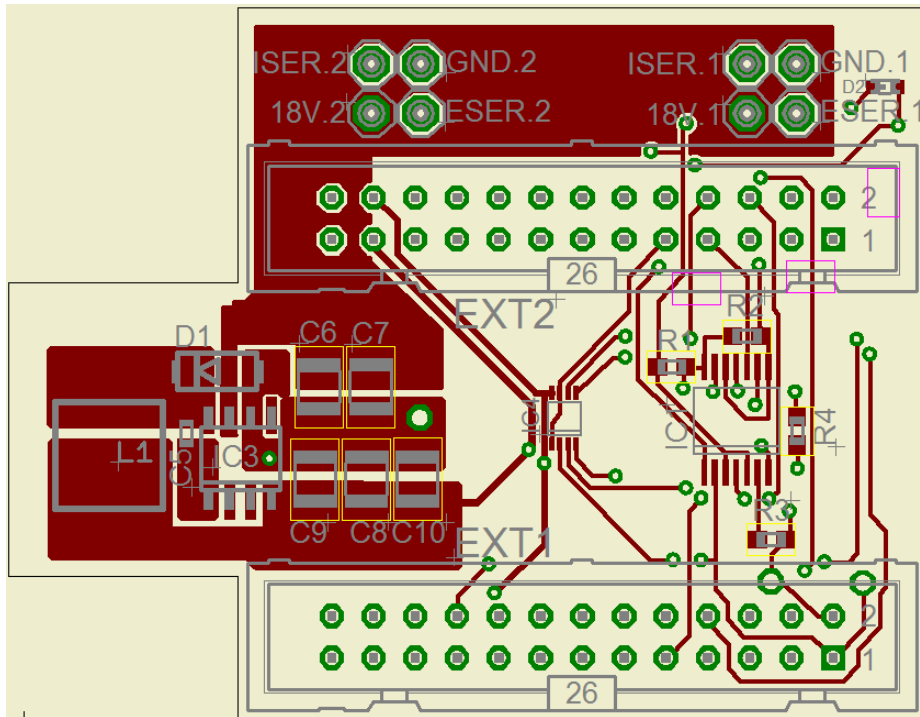
Appendix F: Actuator PCB BOM

Manufacturer Part No	Supplier Part No.	Qty	Design Reference	Description	Package	Pitch, mm	Cost \$	Total \$	200 Qty Totals	Vendor
Section 1: Parts below should be assembled on all 200 boards										
TMK107B474KA-T	587-1247-1-ND	2	C1, C9	CAP CER. 47UF 25V X5R 0603	0603		0.121	0.242	500	20.63 digikey
GRM131CF50J226ZE01L	490-1846-1-ND	2	C2, C8	CAP CERAMIC 22UF 16V X5R 1206	1206		0.94	1.88	500	161.81 digikey
C0603C104K8RACU	399-1095-1-ND	2	C5, C6	CAP 10UF 10V CERAMIC X7R 0603	0603 (1608)		0.023	0.046	500	3.94 digikey
SN74LVC2G126DCUR	296-12555-1-ND	2	IC1, IC4	IC BUS BUFF GATE DUAL 3ST	8-VSSOP	0.5	0.48	0.96	500	84 digikey
LM2940MP-5.0NOPB	LM2940MP-5.0NOPB	2	IC2, IC6	IC SV 1A LDO VREG SOT223	SOT-223-4		2.15	4.3	420	392.7 Newark
MCP120T-460U/TT	MCP120T-460U/TT-M 1	1	IC5	IC SUPRVSR 4.60V OPN DRAIN SOT23	SOT-23-3		0.48	0.48	210	80.33 digikey
PCA9633DP2-T	568-3237-1-ND	3	IC7, IC8, IC9	IC LED DRVR 4BIT I2C-10-TSSOP	10-TSSOP	0.5	1.3	3.9	610	411.75 digikey
87340-0424	WMM18688-ND	12	J1, J2, J3, J4, J5, J6, J7, J8, J9, J10, J11, J12	CONN RECEPT 4POS 2MM LOBBO SMD			1.2	14.4	2520	1,179.49 digikey
GLF2012T100K	445-3159-1-ND	1	L1	INDUCTOR 10UH 140VIA 10% 0805	0805 (2012)		0.22	0.22	210	36.28 digikey
OVSTRGBAC6	828-OVSTRGBAC6	3	LED1, LED2, LED3	LED RGB SMD 2.7 X 2.4 X 0.95			3.09	9.27	630	957.6 mouser
IRPNO22MAM-S-RC	490-1198-1-ND	12	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12	CONN HEADER 2MM DUAL SMD 4POS		2	0.7	8.4	2500	802.5 Sullins
CSTICE16M0V63-R0	Y7243CT-ND	1	Q1	RESONATOR 16.0MHZ CERAMIC	CERAMLOCK	0.95	0.87	0.87	210	117.33 digikey
EXB-28V243JX	RHM15,0KHCT-ND	3	R1, R5, R10	RES ARRAY 24K OHM 5% 4 RES SMD	0402x4	0.5	0.1	0.3	630	39.38 digikey
MCR03EZEPFX1502	RHM15,0KHCT-ND	2	R2, R21	RES 15.0K OHM 1/10W 1% 0603 SMD	0603		0.076	0.152	420	9.83 digikey
MCR03EZEPFX4701	RHM15,0KHCT-ND	2	R3, R4	RES 4.70K OHM 1/10W 1% 0603 SMD	0603		0.076	0.152	420	9.83 digikey
MCR03EZEPFX80R6	RHM80,6HCT-ND	6	R6, R8, R11, R13, R15, R16	RES 80.6 OHM 1/10W 1% 0603 SMD	0603		0.076	0.456	1260	20.1 digikey
MCR03EZEPFX1500	RHM150HCT-ND	3	R7, R12, R14	RES 150 OHM 1/10W 1% 0603 SMD	0603		0.076	0.228	630	14.74 digikey
ATMEGA324P-20MU	ATMEGA324P-20MU-M 1	1	U1	IC AVR MCU 18K 18MHZ COM	44-MLF	0.5	5.13	5.13	505	815.9 digikey
54132-5097		4	X2, X3, X4, X5	0.5 FPC ZIF HSG ASSY 50CKT EMBSTP PKG.		0.5	4.67	18.68	1500	1,512.00 MVP Micro
Section 2: Parts below should only be assembled on 60 out of 200 boards										
B45197A3107K409	495-1548-1-ND	1	C10	CAP TANT 100UF 16V 10% LOESR SMD	7343-31 (EIA)		1.05	1.05	63	57.27 digikey
B45197A6156K409	495-1580-1-ND	1	C11	CAP TANT 15UF 35V 10% LOESR SMD	7343-31 (EIA)		1	1	63	54.87 digikey
GCM188R71H103KA37I	490-4778-1-ND	1	C12	CAP CER. 01UF 50V X7R 0603	0603		0.037	0.037	70	2.59 digikey
B330A-13-F	B330A-EDICT-ND	1	D1	DIODE SCHOTTKY 30V 3A SMA	SMA	0.94	0.94	0.94	63	51.22 digikey
LM2675M-ADJ/NOPB	LM2675M-ADJ-ND	1	IC10	IC REG SIMPLE SWITCHER 8-SOIC	8-SOIC	0.5	4	4	63	191.52 digikey
DR1050-680-R	704-DR1050-680-R	1	L2	68uH 1.7A 11Tindms			1.92	1.92	63	81.27 digikey
RR0816P-8250-D-89A	RR08P8250CT-ND	1	R17	RES 825 OHM 1/16W 5% 0603 SMD	0603		0.14	0.14	70	5.28 digikey
RR0816P-5041-D-76H	RR08P6,04MCT-ND	1	R18	RES 6.04K OHM 1/16W 5% 0603 SMD	0603		0.14	0.14	70	5.28 digikey
Section 3: Parts below should not be assembled on any boards										
TMK107B474KA-T		1	C3	CAP CER. 47UF 25V X5R 0603	0603		0.121	0.121	500	20.63 digikey
GRM131CF50J226ZE01L		1	C4	CAP CERAMIC 22UF 16V X5R 1206	1206		0.94	0.94	500	161.81 digikey
LM2940MP-10		1	IC3	IC LDO REGULATOR SOT-223-4	SOT-223-4		2.15	2.15	75	120 Newark
Notes								82.504		7,420.88
1. 87340-0414 and 87340-0454 are similar to 87340-0424 and 87340-0464, except they do not have bottom locating pegs										
2. R9, C3, C4, C7, IC3, JP1, JP2, JP4, and JP5 should not be assembled on any boards										
3. C10, C11, C12, D1, IC10, L2, R17, and R18 should only be assembled on 60 out of 200 sets										

Appendix G: Controller Daughter Board Rev2 schematic



Appendix H: Controller Daughter Board Rev2 board design



Appendix I: Controller Daughter Board Rev2 BOM

Part	Description	Manufacturer P/N	Digikey P/N
EXT1	CONN HEADER 26 POS STRGHT GOLD	N2526-6002RB	MHC26K-ND
EXT2	CONN HEADER 26 POS STRGHT GOLD	N2526-6002RB	MHC26K-ND
R1	RES 10K OHM 1/10W 5% 0603 SMD	ERJ-3GEYJ103V	P10KGCT-ND
R2	RES 10K OHM 1/10W 5% 0603 SMD	ERJ-3GEYJ103V	P10KGCT-ND
R3	RES 1.0K OHM 1/10W 5% 0603 SMD	RC0603JR-071KL	311-1.0KGRCT-ND
R4	RES 1.0K OHM 1/10W 5% 0603 SMD	RC0603JR-071KL	311-1.0KGRCT-ND
R5	RES 0.0 OHM 1/10W 5% 0603 SMD	MCR03EZPJ000	RHM0.0GCT-ND
R6	RES 82 OHM 1/10W 5% 0603 SMD	MCR03EZPJ820	RHM82GCT-ND
R7	RES 1.0K OHM 1/10W 5% 0603 SMD	RC0603JR-071KL	311-1.0KGRCT-ND
C1	DELETED		
C2	DELETED		
C3	DELETED		
C4	DELETED		
C5	CAP CERM .01UF 10% 50V X7R 0603	06035C103KAT2A	478-1227-1-ND
C6	CAP CER .68UF 50V 10% X7R 1210	GRM32NR71H684KA01L	490-1860-1-ND
C7	CAP CER .68UF 50V 10% X7R 1210	GRM32NR71H684KA01L	490-1860-1-ND
C8	CAP CER 22UF 16V X5R 20% 1210	C3225X5R1C226M	445-1436-1-ND
C9	CAP CER 22UF 16V X5R 20% 1210	C3225X5R1C226M	445-1436-1-ND
C10	CAP CER 22UF 16V X5R 20% 1210	C3225X5R1C226M	445-1436-1-ND
L1	INDUCTOR SHIELD PWR 100UH SMD	DR74-101-R	513-1225-1-ND
D1	DIODE SCHOTTKY 30V 3A SMA	B330A-13-F	B330A-FDICT-ND
D2	LED 470NM BLUE CLEAR 0603 SMD	SML-E12BC7TT86	511-1589-1-ND
IC1	IC QUAD BUS BUF GATE 3ST 14TSSOP	SN74AHCT126PWR	296-4660-1-ND
IC2	DELETED		
IC3	LM2674M-5.0	LM2674M-5.0/NOPB	LM2674M-5.0-ND
IC4	IC BUS BUFF GATE DUAL 3ST 8VSSOP	SN74LVC2G126DCUR	296-12555-1-ND
IC5	MODULE BLUETOOTH W/ANT CLASS1	RN-41	WRL-08497 (Sparkfun)
SKT1	CONN TRANSFLSH R/A PUSH-PUSH SMD	500873-0801	PRT-00127 (Sparkfun)

Appendix J: Preliminary Battery Charger Schematic

