Chloe, Part II

A Robotic Exploration into Digital Companions
Chloe, Part II: Movement

Melissa Kronenberger

ITGM 736: Physical Interactive Media

Fall 2013

I Design Abstract

When we use the term, 'interaction design,' we are not limited to discussing video games and web sites. In fact, interaction designers can be found in many disciplines, from other entertainment-specific fields like theme park design, to broader and more varied subjects like marketing. Interaction design is relative to information systems, the service industry, education, communication, and any other system in which an exchange and an experience both occur.

What follows is my continued exploration into the realms of physical interactive media, through the design and development of an interactive toy named *Chloe*. As a toy, Chloe takes the form of a six-legged robot, or 'hexapod.' As in an interactive entity, she is driven by an on-board embedded system.

To further the research and development of my *Agon & Alea Thesis*, Chloe will be used to gain insight into how physical toys can become the artificial companions of their owners.

At the conclusion of *Chloe, Part I*, Chloe had been assembled and it was possible to use a terminal communicate with her control board and manipulate one of her joints. In *Chloe, Part II*, I will be laying the foundation of her on board software, with the goal of writing a small control engine that will be used to control her motors in the final stages of the project, *Chloe, Part III,* where I expect to be implementing Inverse Kinematics algorithms to be controlling her legs.

II Table of Contents

I Design Abstract 2
II Table of Contents 3
III Introduction 4
IV Background 7
V Project Management 10
VI System Design 14
VII Implementation and Debugging 17
VIII Revised Proposal 18
IX Future Projects 18
X Project B Breakdown 18
XI Future Projects 20
XII Breakdown of Project A 21

III Introduction

1 Project Overview

Chloe, Part II is the second of a three-part project that deals with the construction, design, and programming of a small interactive robot. This robot will be brought to life by a small embedded system. At the conclusion of *Chloe, Part III*, the toy will use an Inverse Kinematics engine to control its legs so that it can walk in a believable fashion.

2 Project Purpose

This report details my continued experience developing with physical interactive media for the very first time. Its primary purpose is to enable me as an interactive designer. As a result, I will have the guts and intuition necessary to design for upcoming technology, or lead a team of toy developers.

Secondly, because *Chloe* already fits the mold of being an artificial character, I will be using her t further my thesis research into how players can bond with and derive emotional pleasure from interactive characters. Specifically, I will use Chloe to evaluate how bring a character into physical space and animating them can be used to create emotional pleasure.

After this project, I intend to utilize Chloe to evaluate how vocal interactions can be used in conjunction with physical movements to create emotional pleasure.

3 Report Structure

In *Chloe, Part I*, I elected to write my report up as a narrative in the first-person present tense. As the report was a personal exploration and thesis tool, I wanted to remember the tense confusion of trying to solve problems I could barely ask the questions for.

Chloe, Part II, however, has been a completely different experience. The majority of *Part II*'s development went to programming labor, an area in which I am already very confident in my skills. There were many pitfalls on this journey, but in general *Part II* has involved much less desperate confusion and the learning curve has been more gradual. A larger amount of development has taken place, but the lessons learned were smaller.

In light of this, and to practice my skill with writing up a variety of reports, I have elected to structure *Chloe, Part II* as a concise write up of my implementation and debugging.

There are a few subsections of my implementation process in which I engaged in investigative behavior. Although I will not focus on this part of the implementation process, I will include the raw 'investigation' text files I created as Appendices at the end of the paper.

4 Personal Methodology Development

In being truthful to my future self, I would like to note that progress on *Chloe, Part II* has been less emotionally gratifying than the significantly more difficult, slower-to-progress, and much more emotionally intensive *Part I*.

I write this so as to accurately represent my experience, so that my future self will understand the whole of my development process, take note of my interests, and perceive my strengths and weaknesses. Take heed future self, and understand that you work best with projects you *don't* already know how to do.

However I also write this to highlight the importance of sticking to one's guns after the initial novelty has worn off, and to impress how important it is to work through times of intense labor in order to bring meaning to those times of rapid exploration and development.

At the conclusion of *Chloe, Part I*, I did not have a walking hexapod. The majority of my exploration time had been spent digging around in a terminal or tightening screws. At the conclusion of *Part II*, I have given meaning to those explorations by teaching the robot to walk.

5 Project Motivations

I am a game designer, and I am working on creating emotionally compelling interactive characters that can converse with their players and offer entertainment and pleasure through emotional bonding.

When I first stumbled upon the body I would purchase for Chloe's construction, the complete project came to me immediately, along with the motivation to go through with it. I have always been fascinated with robots, and I suddenly found myself with the opportunity to pursue that interest while at the same time furthering my Thesis.

In fact, I would go so far as to say that I fell in love with the project instantaneously, and that completing it provides its own motivation.

In addition to this simple but powerful motivation, Chloe provides an opportunity for me to grow my personal design and development methodologies, create a toy to play with my cat, test my thesis, and, in the future, test the effect of sound-based interactions with technology. It also provides an opportunity to gain insight into physical interactive systems.

6 Objectives and Aims

Chloe, Part II is responsible for meeting several independent sets of objectives.

Firstly, *Chloe, Part II* must continue working towards fulfilling the course requirements set forth by my ITGM 736: Physical Interactive Media course. The course requirements are defined as such: students must learn to use physical input devices, develop physical interactive media prototypes, research the best software and hardware solutions for a personal project, develop a personal project using custom

software and hardware, produce short video documentation of their projects for festivals and portfolio reels, and write research reports.

In comparison to *Chloe, Part I, Part II* has entailed recording a large amount of video footage to study the movements of the robot while debugging.

Secondly, *Chloe, Part I* must work towards the individual requirement of the project as set forth in the syllabus; in this case, Project B. This includes respecting the time table set forth in the course syllabus for the project pitch, prototype, documentation, and presentation.

I have also identified a number of objectives that I intend to meet before I can move on to *Chloe, Part III.* This includes successfully driving all of Chloe's servos. It also entails conducting further work into modes of powering her both at my workstation and while on the go; my previous power solution had been insufficient to power all of her legs. My extended goals include getting Chloe to stand and to walk (poorly).

My overarching objective for *Chloe* is to give me the confidence and familiarity necessary to start other physical interactive media projects. My learning objectives are focused on gaining a broad understanding of the discipline and the various steps of the development pipeline. In this way I not only gain an entry-point to learning more about the discipline, but I also develop the understanding I will need to communicate with hardware engineers and low level programmers in the future.

At the end of *Chloe*, I should be able to visualize the process for translating almost any idea for a physical interactive media project into a design and then into a fully developed product.

On a personal level, I am eager to learn more about how power, current, amplitude, and voltage work, and how tools like batteries for RC cards or cell phone power packs can be used to supply power for small mobile devices.

7 Minimum Requirements

Chloe, Part II's minimum requirements and in fact her overall objectives have changed from the initial Project B proposal anticipated at the start of *Chloe, Part I*. Rather than focusing on implementing sound recognition, *Chloe, Part II* now focuses on conducting the labor necessary to get Chloe mobile.

The minimum requirements have been derived from taking into account objectives that were established through in-class discussions with the teacher concerning project scope, and were formalized in the project proposal:

1. Install all preexisting components of Chloe so that they can ride on board if she is disconnected from the computer (control board and BeagleBone attached to chassis)
2. Install battery packs such that Chloe can operate separated from the computer and other charging stations.
3. Software that meets the following objectives:

    1. Chloe can move all of her limbs
    2. Chloe can stand herself upright from a predefined starting position.
4. Video documentation is to be taken of Chloe's attempts at movement

8 Extended Goals

Extended goals formalized in the project proposal were:

1. Extended goal: Chloe can walk using simple, preplanned, Forward Kinematics.
2. Extended goal: Chloe can alternate between two different 'behaviors', such as movement and turning around.
3. Extended goal: behaviors can be displayed and modified in real time through a console in order to experiment with and perfect movements.
4. Extended goal: An IK library has been successfully included to software

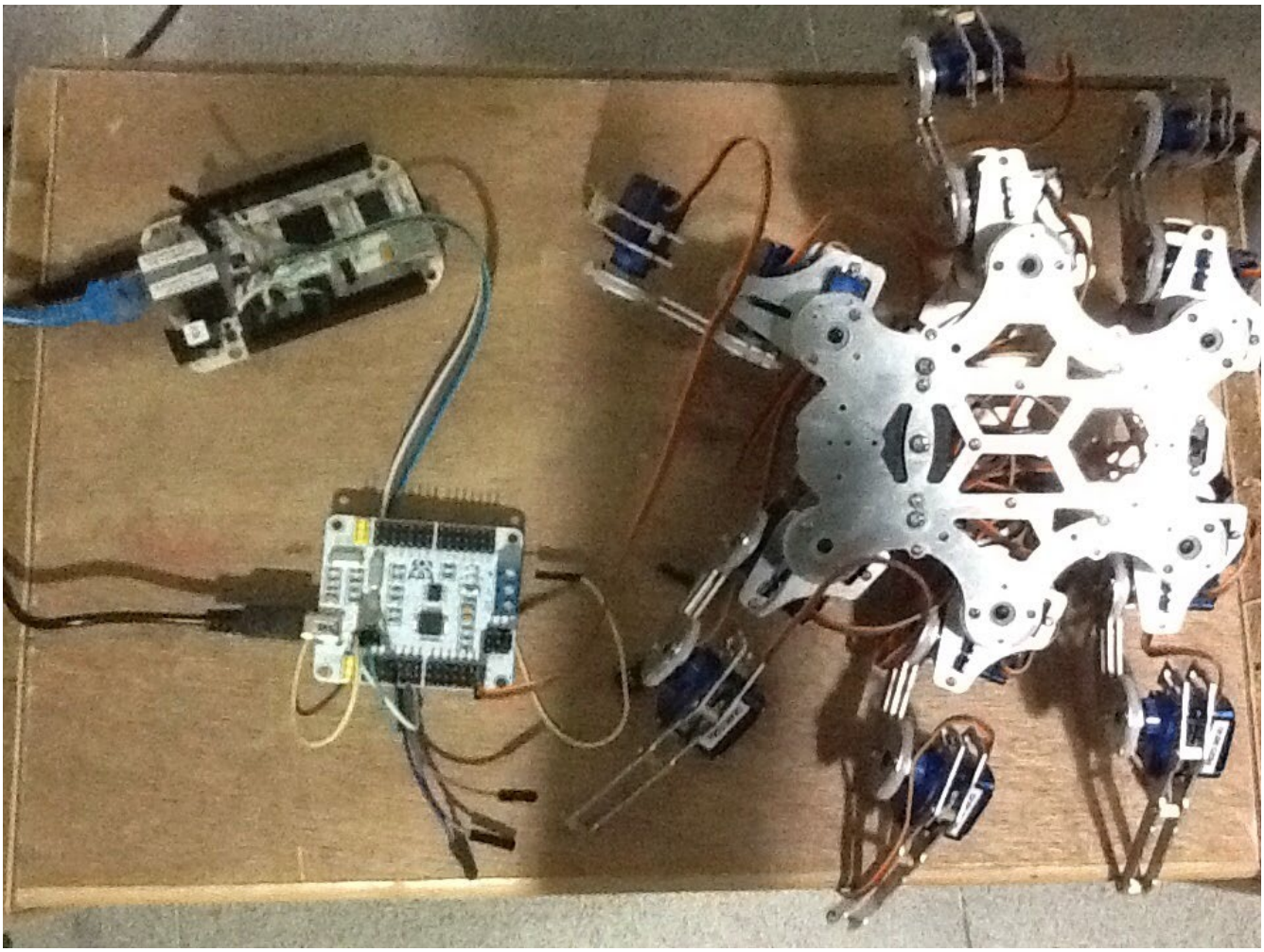
9 System Development Goals

Mid development, it became clear that it did not suffice to program Chloe without some systems analyst work. I realized that I had neglected to design the layout of her code, and that I could implement an architecture that would facilitate development goals. After examining minimum and extended goals, and looking at the time that had been spent on given activities, I made the decision to design and implement a small control engine. The design of this engine is detailed in the section “System Design,” along with the criteria that it meets.

This control system was used to meet minimum requirements 3.1 and 3.2, to meet extended goals 1,2, and 3, and to make it easier to implement goal 4 at the onset of *Chloe, Part III.*

IV Background

1 List of Core Parts



1.a BeagleBoard BeagleBone

1. MicroUSB power cord & communication
2. White, Revision A5
3. 3.3 Volt system
4. Thorough documentation, community, and guides at www.beagleboard.org
5. Embedded distribution of Ubuntu

1.b Control board for direct interface with servos

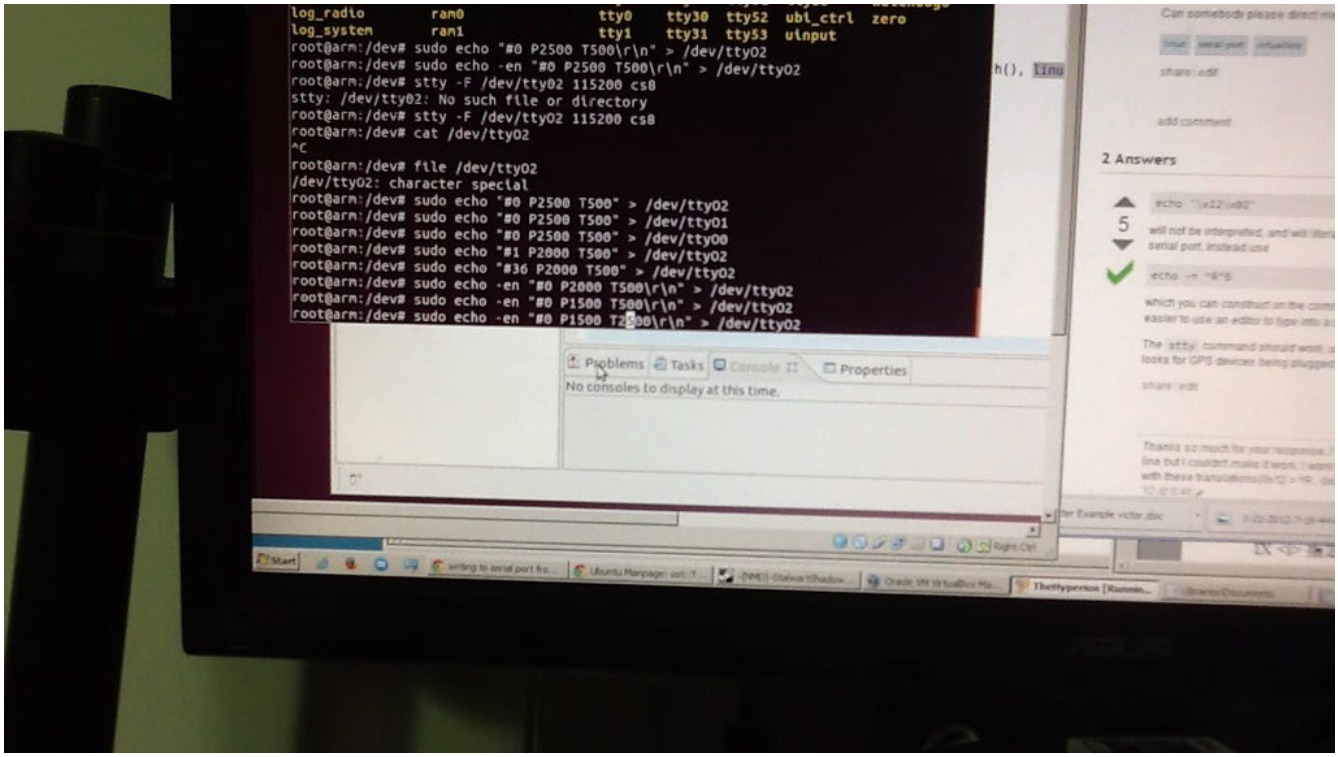
1. Printed documentation in Chinese
2. 5 Volt system
3. UART driven, capacity of 32 analog servos
4. MicroUSB power cord & communication (insufficient to power all servos)
5. 3 pins for alternative means of powering board (we will be using this)
6. Specialized for RC

1.c Chloe's Chassis and Legs

1. 18 analog servos, 3 per leg
2. 5 Volt system
3. Assembled

1.d Windows PC

1. Windows 7
2. Running Oracle Virtual Machine
3. Running Ubuntu Desktop Linux distribution for ease of communication with board
4. Eclipse IDE
5. GIT repository at Bitbucket.org for tracking changes to code and enhanced portability



2 Additional Resources from *Chloe, Part I*

Most of what I took form *Chloe, Part I* was in terms of learning and understanding. I could assemble a new chassis in a day, set up my work environment in an hour, and reconnect my boards in a minute, but the conceptual models I require in order to perform those tasks was forged over the entire length of *Chloe, Part I.*

Out of all of the information I aggregated and interpreted for *Chloe, Part I*, there are certain components that will serve as immediate launching points for development in *Chloe, Part II.* Chief among these is my knowledge of what a 'pin multiplexer' is.

Using the Linux terminal application during *Chloe, Part I*, I was able to locate and run a script that activated my UART2 Rx and Tx pins. In layman's terms, I found the tool that would allow me to stick one half of a wire into my BeagleBone, and the other half into my control board, and transmit commands that would get translated by the control board and relayed to the servos.

I also learned to identify how Linux was able to talk to devices, and that the serial device I wanted to transmit across would take the shape of a symbolic file named 'ttyO2' that would appear in a certain folder on board my Linux distribution.

I demonstrating my understanding of this by sending commands in via the terminal and observing as the leg moved.

However the 'ttyO2' logical port does not remain open when the BeagleBone is restarted. I will have to go into my kernel's files and determine how to ensure the port is activated on boot. That way, any C++ program I write will be able to talk to through the port without me hacking into a Linux console and initializing anything. This is referred to as 'pinmuxing' or configuring the pin multiplexer.

In addition, I need learn the API necessary to communicate with the port via C++. With something simple like the on board LED's of the board, this meant using commands like 'fopen' and 'fwrite.' In communicating across a serial port, I anticipate I will have to use additional functionality just to initialize the port, much less write to it.

3 List of Extended Parts

These components will only be brought into play if the minimum requirements for *Chloe, Part II* are met and exceeded.

1. 2x Cell-phone charging power pack (battery for robot)
    - Outputs 3100 milli amps (2.1 + 1 amps at 5 volts)
2. 2x Wireless charging pad & receiver
3. Assorted wires, including a Mini USB wires for connecting wireless charging receiver to power pack
4. 5 volt Power supply for testing
5. Soldering tools

V Project Management

1 Documentation as a Design Methodology

For me, documenting my design process is a vital part of my design methodology that helps me combat negative emotions like anxiety while providing me with an in-depth look at my progress and the time it takes for me to complete tasks. It is because of this thorough documentation that I was able to reconstruct my project log for the documentation phase.

In *Chloe, Part II*, my documentation was quite different from *Chloe, Part I*. This is because *Chloe, Part I* was a detailed investigative process, where *Chloe, Part II* was largely concerned with design, implementation, and debugging. My documentation for *Chloe, Part* II therefore began life as detailed designs written on a note pad early each morning as I started off my day in a Hong Kong tea shop with jammed toast and lai cha.

These design notes served as the map for my programming process, which at time could grow very hectic and disjointed as I hunted down the API for complex programs or tried to debug various errors. Since the architecture I was building had many individual components, it would have been incredibly easy to finish a debugging task only to experience a sense of disorientation and exhaustion when I attempted to remember what large-scale project I had been working on.

My design notes not only served to anchor me in a chronological implementation process, but they also kept track of which classes were responsible for what variables and what functionality, so that I did not accidentally implement anything twice or with incompatibilities. I was very happy with the speed and precision I was able to code with as a result.

After my code was written, I would commit each major update to a repository using the GIT framework and an online service called Bitbucket. GIT is traditionally seen as a tool for social coding. However, GIT also allows me unprecedented portability and revision-tracking as I move between development machines (for example, my home and school machines, or my laptop and desktop). More importantly, GIT and Bitbucket supplied me with an extremely accessible ‘journaling’ tool.

I always had an incentive to ‘commit’ and update after every major revision so that I could revert back to a ‘working’ version of my code if some new change I made broke everything, or if my computer spontaneously combusted. With each commit, I was forced to supply a message. These messages would be logged and dated, and as a result I was left with a very well-maintained, fine-grained record of all the progress I had made in implementing *Chloe, Part II*, complete with my thoughts and the exact moment I committed the change.

This, along with my design notes, allowed me to go back once or twice weekly and fill in a detailed project log concerning what I had done with my life each and every day since the project began. This is an incredibly amazing part of my design methodology, which works to counteract anxiety. Despite the pressures all students begin to feel as we near the midterms, I can safely say that I am more on top of my projects than I have ever been in the past.

Bitbucket's additional issue tracking and wiki tools, along with google calendar, provided an additional level of control over my milestones and my feelings towards my milestones that I had never previously experienced.

2 Project Schedule

Chloe, Part II was planned on a much tighter time table than *Chloe, Part I.*

- 15 October – Project B assigned
- 17 October – Due: Project Ideas
- 22 October – Due: Project Pitch
- 24 October – Due: Prototype
- 29 October – Due: Documentation
- 32 October - Due: Presentation

3 Project Log

11 October 2013 (F): Recovering after the turn in I did for Thursday's documentation.

12 October 2013 (A): I went shopping and bought a new charging solution for Chloe. I was in SSP almost all day, looking at shops, trying to find things like anti-static cases, evaluating microphones, etc. Bought components with The Captain to make wireless charger, battery packs, and microphone.

13 October 2013 (U): Pinmuxing successful! It took me all day trying various things, researching… The biggest problem was that things have changed very rapidly over a short period of time, making it difficult to find accurate information about pinmuxing that's relevant for today. The eventual solution was very quick. I've also researched some code for talking to ttyO2, and I've bookmarked a tool called 'Libserial' for interfacing with the port.

14 October 2013 (M):

- Went to an Arduino talk (by Massimo Banzi, cofounder and CEO of Arduino).
- Then I came home and worked, and worked, and man I got a lot done. At first I was stuck forever on using Libserial, which just didn't seem to be worrying. After printing some debug statements, I managed to track down what I felt the problem might be; I believe that the escape characters \r and \n simply weren't transferring properly, and I couldn't find a way to alter this.
- After some evaluation, I decided to back up and look at several of the other more 'nitty-gritty' solutions I had bypassed on the way to Libserial. It was a little ugly to look at, but I knew the termios.h solutions I found were the code on which Libserial was founded; they were the absolute lowest baseline I could work with and they were where I'd have the most control. It was slightly difficult to decide to use termios.h, because I couldn't find a concrete example for talking to a UART.
- Eventually I dived in, and now I have a working C++ program that transfers a command to Chloe and causes a motor to move.

15 October 2013 (T): In school we had a lecture on matrices and modeling the robot. I bought Soldering supplies and checked out battery solutions again now that I realize my battery pack doesn't have enough amps.

16 October 2013 (W): Worked on system analysis and programming Chloe.

- Started doing the major work I needed in order to write Chloe's code, including how I was going to represent the different servos and call them quickly and naturally, while at the same time storing all the calculations.
- Programming. Had to figure out how to do 'static pointer,s' to singletons and otherwise. Spent a lot of time asking questions, trying things, and trying to hash together the C++ syntax I needed to do what I wanted to do. For example, to let any member of an array all talk to the same output device, I needed a 'static' pointer to that output device. I also figured out how to make templates work in my Utility class.
- I wrote all the documentation due for Project B: Ideas.
- All of this together is sort of impressive given that I woke up at like 2:40 PM!

17 October 2013 (R): Presented Project B: Ideas.

18 October 2013 (F): Did a little bit of work on Chloe; had a spider crawl across my screen while I was trying to do work for my other classes. It's a sign!

19 October 2013 (A): The Captain has passed on the art of soldering to me! I have created the charger for Chloe after much labor and noxious fumes!

20 October 2013 (U): I coded nonstop, from tea shop into the home, for a definitive six hours solid, and then about an hour after that in debugging. I managed to code by listening to pop music and by virtue of having laid out everything I was making in the tea shop in pseudo-code first (without that, I never would have made it). It was very hard to do because it was an architecture, it didn't yield immediate results, there were abstract classes, and Chloe cannot yet be powered with all her legs attached so I can't even see if what I did has value. Judging by the messages that are sent to the system and printed out, however, I have done it correctly. Had a ton of big commits on this day. Ran into a problem where C++ vectors can't be initialized easily, and found a way around that using arrays.

21 October 2013 (M): I made big process with Chloe. I plugged her in, powered her successfully, and then managed to edit the program in such a way that she took her initialization pose. I accidentally broke her shoulder. I took video.

22 October 2013 (T):

- Chloe Prototype Due Thursday. Did 'Pitch' powerpoint for Chloe Part II.
- Neat terminology lecture on actuators, etc.

23 October 2013 (W):

- Replaced Chloe's bad shoulder and realized that the cause of the problem originally was that I had plugged the shoulder and wrist/tarsus into each other's slots. The old shoulder joint still offers no

resistance while unpowered, but might be less damaged than I think. Hoping to get her to stand soon.

- Trying to find a metadata browser for my iPad so I can use pictures I've taken of Chloe to reconstruct when it was I worked on her. Might take awhile, as I'm also updating my apps.
- Spent 6 hours programming and got Chloe standing. Note: I did it entirely by using ‘Minerva’ or my secretarial persona, in which I write down instructions for myself baby set by baby step until task began to self-motivate. It was awesome getting so much done.
- I did soldering in the evening till 3:00 am and totally messed it up! But learned a lot and gained experience. I was trying to join those four lines together. Twice. And I nearly killed my batteries and my board with a short circuit!

24 October 2013 (R):

- I brought Chloe in to class for everyone to see, including a visitor, and showed the video of her walking, which was very impressive.
- The professor told me that Chloe's powering system wouldn't work and gave me a lecture on current.
- Will talk with The Captain when I get home. I think the professor’s lecture might be influenced by the fact that he is used to designing with bare bones circuit components. However, The Captain’s been working with RC toys forever and has built wheeled robots before. I think there might be something the professor doesn’t know to consider, which the Captain does. It is clearly my soldering job was poor, and I’ll continue blaming that for my failure until I know otherwise.
- Looked at phone and camera batteries with the professor.
- I picked up some cables and other materials before heading home, including two cables that already had two USB heads. The Captain helped me buy them because they're not that common anymore, and I don’t speak Cantonese.

25 October 2013 (F):

- Spent time programming with Chloe to figure out how to make a non-blocking input pipeline. Ended up finding a simple enough example of the Linux poll() and its Windows counterpart that I was able to throw some code together, cross my fingers, and pray. I had been fiddling around with cin.rdbuf()->isavail() but then I read a blurb that explained that this technique wouldn’t work, as the buffer did not fill until cin tried to read from it and found it empty.
- At first I got weird results; then I realized poll wasn’t sensing input until I pressed Enter, and I realized that since poll uses a structure to submit a 'poll request' that modifying the structure could probably be used to sense commands like WASD minus the ENTER. But as I wanted to enter a whole line and use backspaces anyway, I left it as is :) It was better than my original mental solution.
- 'quit' now quits the program.
- I satoutside in the teashop today, because it was utterly full. The binding on my notebook is going.

- At the teashop I went and figured out what my console commands were all going to look like, how I was going to enter them, break them apart, and parse them, how I was going to convert enumerated types to strings and back again for ease of use and readability, how I'd input the command to read or write to filename, how I'd structure the file, and I wrote some notes about my power solution with Chloe.
- I cleaned. Lots :) Straightened up the disaster I made on Wednesday with the soldering

26 October 2013 (A): Strings lack an easy 'split' functionality in C++! Oi this language is so silly sometimes. I'm frustrated because Chloe is currently unpowered, and being unable to test her makes me grouchy. I really hate programming interfaces, so I have almost no interested in actually implementing everything I figured out.

27 October 2013 (U):

- Implemented the entire console blind XD. Haven't tested more than the list function yet. Ran into some 'game breaking' bugs in terms of being unable to convert floats to strings, only to totally power through them by using multiplication/division and integers, and string concatenation with the character '.' so that now no matter what float<->string related bugs I encounter, I have a methodology for getting through them. In the end I went with this solution due to the lack of any other people reporting a difficulty with all floats registering as 0.
- I realized there was a significant chance that either the ARM architecture or my Bone in specific handled floats in a way that the stringstream conversion method wasn't prepared to handle. I'm sure there are workarounds or methods for correction, but I'll have to approach that with a new eye later because all the key term collections I navigated to didn't yield a solution (people were much more concerned with how to format floats and trimming or inserting zeros. For now my solution works, and that's a testament to my skill as a problem solver; I didn't get bogged down trying to find an aesthetically 'right' solution, and I implemented a 'wrong' solution that did exactly everything I needed it to do.
- All that programming was exhausting, mostly because I wasn't programming a very thrilling part of Chloe. I'm glad to have it behind me though. That type of thing could really act as a blocker that would keep me from getting to the 'good' stuff. In fact, the rest still is getting in my way! But work is work, and getting unpleasant work behind me is good for me! That means I don't have to do it later when I'm feeling even less interested.

28 October 2013 (M):

- My brainstorming at the tea shop mostly dealt with trying to keep myself calm about Chloe's power situation- or maybe calm myself down. I seem to be afraid of putting her two new cables together because the last setup didn't work. I seem to be thinking very fatalistically about her power and I need to stop and just try things. After all, there's an easy way to test if the four-USB setup will hurt itself or not, and then if it doesn't hurt itself I simply have to test it. I need to power Chloe one way or another (example: by reconnecting her to the power supply) or I can't move forward at all!

VI System Design

The minimum requirements for this project require a small program that opens up a connection through the logical ttyO2 (The UART2) port and transmits instructions successfully to the control board. It will be easy to see if the project is successful based on whether or not the servos move.

However, I anticipate I will dramatically overshoot my minimum requirements, despite the narrow time allotment for doing so. In light of this, I have taken the next step to thoroughly plan out and design a 'servo control engine' responsible for driving the physical aspects of the robot. This engine, whether it is implemented in full now or in *Chloe, Part III*, will also be the means through which the IK library communicates with the robot's motors.

Given the short time window I have to implement this 'engine,' it is vital that I spend some time on this design step, so that I produce a tool that accelerates my development process instead of hinders it.

1 Need for a Control Engine

In order to facilitate all current and further development of Chloe, this project sets forth the design of a control engine that wraps the communications protocol and reduces the workload of the human programmer. Below are listed the specific problems the control engine has been designed to address.

1.a Lack of Meaningful Human Terms

The control board accepts commands in a form that is not closely linked to a human mental model of how the robot works. By abstracting the interface to work with recognizable names for servos, degrees, and seconds instead of milliseconds, the Engine takes into account the context of the robot's development and lightens the mental strain on the programmer while reducing room for error

1.b Manage Small Discrepancies

Due to small physical discrepancies in the assembly of the robot, and to differences in motor orientation from the left side of the chassis to the right, different commands must be given to seemingly similar motors in order to illicit the same desired results. Without the Engine, the burden of tracking, remembering, and utilizing these details would be left to the programmer.

1.c Batch Commands

Frequently the programmer will need to move more than one joint in an identical manner, and efficiency would be lost without the ability to condense six identical commands into one. The value of this is compounded when corrections need to be made rapidly.

1.d Editing Commands in Realtime

Due to the long compilation and upload pipeline, it is difficult to accurately test new commands or strings of commands, an important weakness when the robot is not perfectly symmetrical and it is expected numerous small tweaks will be necessary to obtain desired behavior. It is also impossible to modify commands at runtime.

1.e Encapsulate Commands in Modular 'Behaviors'

There is no sense of modularity to naked commands, which limits their readability and reusability, and reduces the ease of programming an AI to initiate them due to the lack of encapsulation and labeling.

1.f Enforcing Timing

There is no way to coordinate the timing of commands outside of the use of 'delay' which would block all other movements or computations while active.

2 Criteria for Control Engine

1. Encapsulates commands in 'Behaviors' which are named and editable entities. A primitive AI is then expected to select from the available behaviors to execute commands.
2. Wraps communication functionality to use human recognizable terms for use by the programmer, such as joint names instead of servo numbers, degrees instead of impulse size, and times in seconds instead of milliseconds.
3. Contains an interior model of the robot which accounts for small physical discrepancies
4. Permits editing of commands in realtime through a terminal.
5. Can read or write command sets to file.
6. Has an interior timer on which all updates are based, and allows commands to be triggered at certain times.
7. Can handle both forward and inverse kinematics, or at least has an anchor point for the inclusion of inverse kinematics.

3 Design of Control Engine

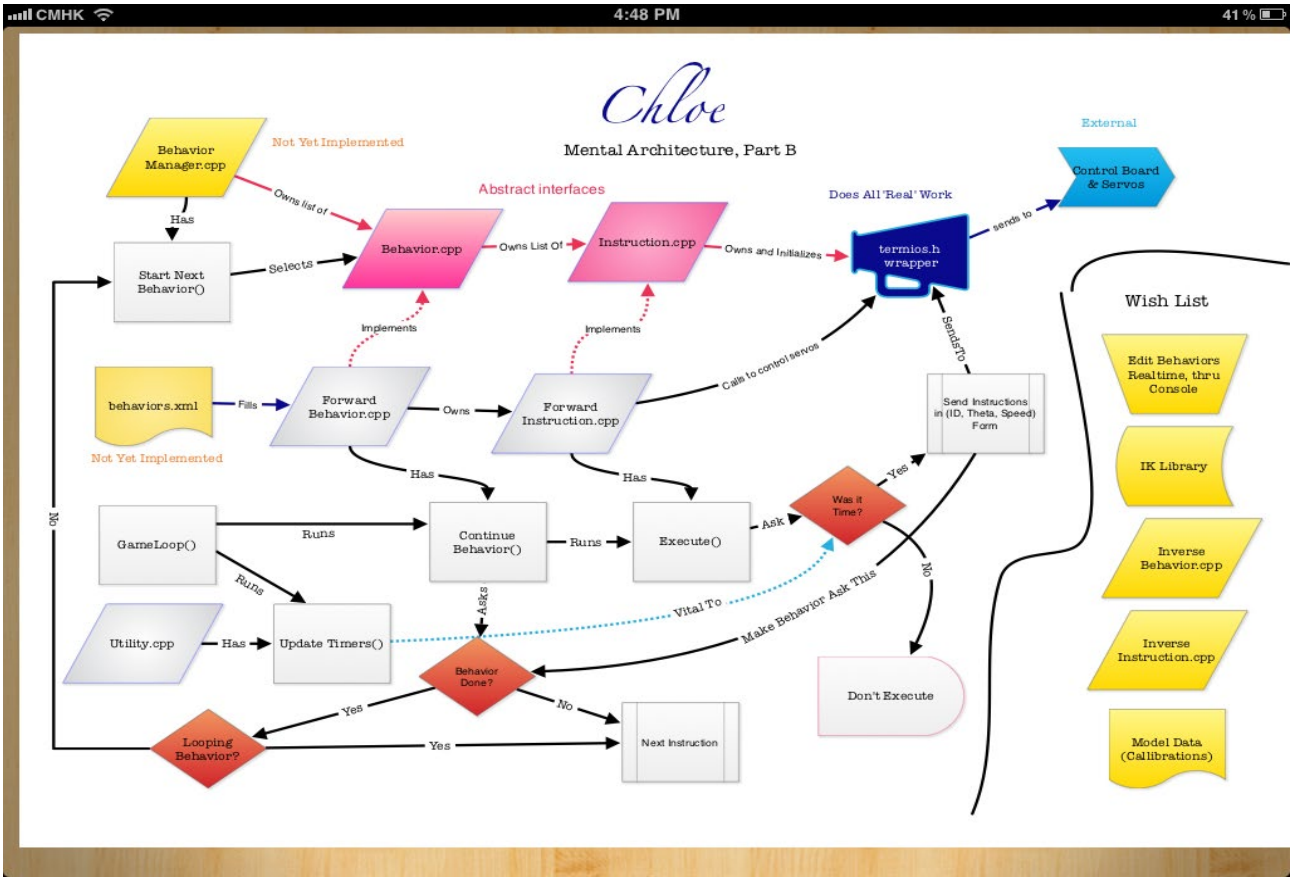
I laid out the design of the control engine using a custom diagramming strategy. The primary shapes to be aware of are parallelograms for representing classes, squares for representing functions, and diamonds for representing decisions.

The control engine is made up of the following classes:

1. ServoSpeak: Termios.h wrapper which sends commands to the control board in the format the control board will understand them in.
2. Behavior: Abstract Class that represents a list of Instructions, knows whether it is a 'looping' or continuous behavior, and has a name.
    a. Forward Behavior: This behavior houses a list of Forward Instructions and an iterator. Each iteration of the game loop, it attempts to execute the next Forward Instruction and advances its iterator. If the behavior itself is looping, the iterator returns to zero after going out of bounds.
    b. Inverse Behavior: Not currently implemented, will be used for IK engine
3. Instruction: Abstract Class that holds data in a user-recognizable format, but then sends instructions in control board format to ServoSpeak for execution.

a. Forward Instruction: This instruction contains a list of servos identified by enumerated type, a theta expressed in degrees specifying the angle to rotate the servo to, a key time to execute specified in seconds, and a duration over which to rotate that is also specified in seconds.

4. Utility: Maintains time and deltaTime, grabs input from console, provides utility functions, and hosts enumerated types to describe servos.
5. Behavior Manager: Owns, selects, and cycles through Behavior sets, and parses input to read and write to Behaviors and Instructions
6. Main: Game loops that calls Utility update time and input and then BehaviorManager to update Behaviors and Instructions.



VII Implementation and Debugging

The implementation and debugging process can be thought of as being divided into several subcatagories. These are: Pinmuxing, communications, cross-compiling, architecture, motion, power, and console.

Chloe, Part II vastly overshot most of its minimum requirements and successfully implemented many of its extended requirements. Chloe only fell behind on one dimension, where substantial work revealed new problems to solve as the project moves onward into *Part III*.

1 Pinmuxing

For clarification, the BeagleBone offers more functionality on its P8 and P9 expansion headers than it has pins for. To determine which pins have which functionality enabled, the board has a pin multiplexer to select between the different functions. Vital to *Chloe, Part II* was enabling one of the pins to function as a UART serial port, so that the BeagleBone could send instructions to its servo control board. While it was already known how to enable this port manually, the port needed to be enabled automatically at boot time so that Chloe could be detached from her Linux host machine (the PC).

Pinmuxing, or configuring the pin multiplexer, was the most difficult phase of *Chloe, Part II*. An investigative journal was kept of the numerous attempts to understand firstly what the task demanded, discover what tools were available to complete the task, and lastly to carry out the task. Complications arose due to recent changes in the philosophy of how Linux works on embedded boards.

Until recently, the BeagleBone had its serial ports pre-configured to work out of the box through its kernel. When this stopped being the case, it was not easy or straightforward to configure the pin multiplexer, requiring developers to dig through thousands of pages of documentation on various circuits (namely the ARM processors themselves) and correlate them with information in the BeagleBone documentation.

The scripts necessary for configuring the multiplexer were also difficult to decipher, as in many cases it was not clear exactly what the resources researched for this task were attempting to accomplish. Further complications arose as a result of the BeagleBone Black, which tended to dominate resulting search queries, and the use of the term 'capes,' which seemed to be used both when discussing software on board the kernel for pinmuxing, and physical board add-ons for the BeagleBone.

However, since the release of the first new kernel, additional resources had been included in subsequent kernel updates. Among these were predefined scripts for enabling the necessary serial port functionality, which could be added into a file that specified scripts to run at boot time.

The full raw text of the pinmuxing investigation is available in this report as an appendix.

2 Communications

Once the pinmuxing had been finished, it was easy to communicate with the control board via Linux prompts. Explorations in *Chloe, Part I* had highlighted the way to talk with some elements of the

BeagleBone, such as the LEDs, by writing to a symbolic 'file' in the Linux file system. However, the means of using C++ code to transmit across the port were not yet known.

Work began with a library called LibSerial, designed to facilitate communication across a serial port. This work was very difficult and resulted in the second full raw text investigation, which has been included in this report as an appendix. It led to an exploration of how to cross-compile libraries. However, while it appeared that LibSerial was eventually implemented correctly, it was unsuccessful in communicating the messages necessary to drive Chloe's servos.

It was deduced that this might be owed to something in LibSerial's initialization which, hidden to the developer, might have been preventing the correct transfer of escape sequences such as \r or \n. At this point a substantial amount of effort had been put into understand LibSerial, but this effort had revealed enough about communicating with logical ports that alternative routes for completing the communications implementation became understandable.

LibSerial and all similar libraries for Linux systems are built on the standard functionality provided in the low level termios.h include and its relevant libraries. LibSerial had been written in a more user-friendly C++ style, and termios.h took after a more low level and less human-readable C convention. LibSerial had been the tool of choice initially because it was easier to comprehend. Although it failed to yield results, it provided insight into the workings of termios.h.

The LibSerial implementation was sidelined and replaced with a termios.h implementation. Although the program initially hung, it was observed that the termios.h implementation had successfully driven one of the servos it was connected to. Further investigation showed that the 'hang' had come from a programming error elsewhere in the program, unrelated to termios.h.

The termios.h implementation was encapsulated in a class called 'ServoSpeak,' which was to be called on from the rest of the program to send messages to the servos.

3 Cross-Compiling

Although the development of Chloe's previous programs had required understanding cross-compiling, most of this was accomplished by activating pre-existing plug-ins or scripts or following short tutorials, and was shortly forgotten.

Trying to get LibSerial to work entailed learning much more about the compilers and libraries involved, and revealed that the activated plugins had likely given the Eclipse IDE access to a wealth of cross-compiled libraries for all of the standard C++ libraries. Cross-compiling by hand was very difficult, and both the discovery that it was necessary, and the actions taken to cross-compile, are recorded at the end of this report in an Appendix.

4 Architecture

Here, 'architecture' is used to mean the implementation of the classes and structures that fulfilled the encapsulation and modularity criteria of the System Design; specifically criteria 1, 2, 3 and 6. The difficulties at this stage were mostly concerned with re-learning C++ programming conventions. This included the limitations of the vector class when attempting to initialize long lists of data, how to work

with singletons and global pointers, and static, constant, and external functions, pointers, and variables.

Programming was facilitated with the use of research tools. When the syntax for a certain functionality, such as string concatenation, was required it could be quickly looked up. Debugging was straightforward and logical. When the expected syntax did not yield results, several similar attempts were made. If this continued to fail, the exact C++ conventions for achieving a certain effect were researched.

The primary difficulty at this phase was a managerial one. After a substantial amount of effort had been invested in learning how to provide a type of functionality, it might have become difficult to back track and recall exactly what larger scale task was currently in progress. To this end, the system design for the architecture was used as a task and milestone list, and dramatically relieved the mental burden of remembering the 'big picture.'

5 Motion

Chloe was able to move one limb after the communications phase of implementation. After her architecture was largely completed, efforts began to get Chloe to take an initialization pose and then stand. The trial runs, which were numerous, were recorded on video. It became clear that Chloe was suffering a power deficiency. At times it seemed her commands would not launch, that her servos would give up, or that she would move at a different pace than her commands had specified.

However, after much trial and error, Chloe finally stood up.

Immediately afterward, with an understanding of the power limitations in place, it was trivial to program Chloe to 'attempt' walking.

Chloe's power packs were added on to her chasis, along with the BeagleBone, after she accidentally stepped on the board and nearly damaged herself. It was clear that Chloe did not have the power to successfully get her legs beneathe her while so burdened, and so could not properly stand or walk.

An experiment was conducted where Chloe was successfully balanced on only three legs while unpowered, indicating that Chloe should be able to walk, and that her motors going limp due to power problems are the most likely culprits for her inability to rise. It is believed that Chloe will be able to stand when her power problems are resolved; however strapping her power packs to her belly instead of her head may also give her the boost she needs to take a standing position.

If Chloe continues to be unable to rise after her power situation is changed, hooking her up to a small cart that carries the batteries as a spidery 'abdomen' will be the most logical and aesthetically pleasing route to take.

6 Power

It was a requirement of *Chloe, Part II* to come up with a power solution for Chloe. This was necessary to power all of Chloe's legs so that she would be able to move them all, stand, and walk. In order to

meet these requirements, Chloe was initially supplemented with a power supply that allowed her to walk.

She has since been outfitted with two battery packs that should allow her to have mobile power; however this solution has not yet been successfully implemented. The first two attempts resulted in a short circuit that, thankfully, did not damage any of the equipment involved. A third attempt will be made with newly soldered cables.

These power packs are already charged by wireless charging pads, and wireless receivers mounted to the robot's chassis.

However, Chloe's power situation has become more complex since *Chloe, Part II* began. Additional information from the control board's manufacturer suggests that 6 and 7 volt power is necessary in order to properly drive the board, and Chloe continues to be plagued by the fact that she needs a large amount of current in order to properly drive her servos (and estimated six amps).

After being mounted, the weight of the power packs increased the symptoms of her power deficiency.

Full-filling this requirement will therefore most-likely become an auxiliary goal of *Chloe, Part III.* However, as it is not necessary for Chloe to be fully detached from a immobile power source in order to demonstrate that she can move around, it may also be deemed desirable to eschew this goal in favor of implementing other functionality. These decisions will be made after further evaluation of the project's progress.

7 Console

A console has been partially implemented for Chloe that will permit her Behaviors and Instructions to be displayed and edited in real-time, and then saved to and loaded from files. This will help when trying to establish an accurate model of Chloe's small physical irregularities for use with the IK system. Currently, the console is capable of listing behaviors, and will soon be capable of adding or editing them.

VIII Conclusions

Chloe, Part II was a resounding success. It took everything that had been learned and built from *Chloe, Part I* and exemplified it in a walking robot that made viewers smile. Many extended goals were met, despite the fact that the minimum requirements themselves had been set higher than initially believed was necessary.

An important understanding of power and Chloe's power condition has also been achieved as a result of this project, which I as the developer did not previously possess. Although there are still many questions to be answered about exactly how Chloe will be powered, and I am moderately disheartened that she is not yet free of her power supply, I am also eager to learn these lessons concerning power so that I will later have the foresight and resourcefulness to create other mobile toys in the future.

IX Appendix A: Pinmux Investigation

1 Description of What Follows

There was a subsection of *Chloe, Part II* that warranted the same investigative style of problem solving as the majority of *Chloe, Part I.* This was figuring out how to configure the pin multiplexer on board the BeagleBone. While trying to understand this task and solve the problems it entailed, I created a large first-person document. I am included this document unedited and in its raw form, for future reference as to exactly how I managed to resolve challenges.

2 Getting Started on October 13th

I started today from here: http://elinux.org/BeagleBoardPinMux#UART2
Based on this I kinda understand I need to do some uboot and kernel related things. I also realized that BeagleBoardPinMux is BeagleBoard specific, not BeagleBone. It still works, but I have to find the corresponding pinmux tables.

I've found some BeagleBone Pin Mux tables here:
https://github.com/bradfa/beaglebone_pinmux_tables

3 Trying to Make Sense of What I'm Reading

However I don't see the connection just yet between the tables available in both resources. Let's see if I can work through this. First of all, I know the most important pin to me is Pin 21. Pin 21 Is transmit for uart2, it's uart2_txd. Perfect.

Now what do I have to fill in to my kernel based on this tutorial? Let's see, there's a 'U-Boot' header and a 'kernel' header. I'm seeing a u-boot heading that's the same as the 'BeagleBoard Description' more or less. So based on the BeagleBone PinMux tables, I'm assuming this means the corresponding item on *my* board's table for 'U-Boot' is 'spi0_d0.'

There's an offset category on the elinux page, and on my Bone's pinmux tables I see a PINCTRL ADDR header, which resembles OFFSET most closely. For me, that will be 0x154.

Now let me look at KERNEL on the elinux page. I see something like...

OMAP35XX Pin RevC: AE3
U-BOOT: MMC_DAT7
KERNEL: AE3_3430_USB3HS_TTL_NXT

So I'm seeing basically where AE3 is coming from, and 3430 and 34XX appear to be the only two options for what comes after that. I should note that I have a 'PROC' field on my BeagleBone PinMux reference, and it says pin 21's PROC field is B17. This looks pretty similar to AE3.

3.a Looking Around Texas Instruments

I clicked on the Link to the technical reference document for the BeagleBoard, but it was a dead link. I navigated to 'Search Technical Documents" on the Texas Instruments site, and I see it asking for what product I want to look at. Here I see Product: OMAP35x Processor. That looks familiar! Let's see what the processor is on the BeagleBone itself.

Looking at this BeagleBone System Reference Manual, I see that the board currently uses either the AM3359 or AM3358 processor. I believe mine will have the AM3359 because this is a spec sheet for revision 6 and I have revision 5, and the AM3359 is the chip that's being phased out.

So back at Texas Instruments Tech Docs, I'm selecting "ARM-Based Processor Platforms" and "AM335x ARM Cortex-A8." I selected multiple kinds of documents and found the TRM (Technical Reference Manual) under User Guides.

The Technical Reference Manual is ~4734 pages long XD.

Where to go first? Obviously I don't want to *read* this. I can see Section 19 is about Universal Asynchronous Reciever/Transmitter(UART) and it has a section on Integration, 19.2, specifically 19.2.3 is a Pin List...
Aha! Table 19-9. UART Muxing Control! Whoa, I don't really understand what this means. I'm seeing things like... well let's look at the line that looks most like the one relevant to me:

TXD,	RXT,	RTSn,	CTSn,	UART

vs the other two lines

IRTX,	IRRX,	SD,	not used,	IrDA (SIR, MIR, FIR)
RCTX,	RCRX,	SD,	not used,	CIR

Okay, can I use these to build something that looks like "AE3_3430_USB3HS_TTL_NXT"? Why don't I look at USB stuff for the board and see if anything looks familiar? That will be on pg 2217, section 16.2.3.

No, I'm afraid nothing here is looking familiar... What if I check out USB3HS? No findings. TTL_NXT? Nothing. Well then I'm not sure where to look. Wait! I've an idea. Look back to the elinux page. Aha! The Offset was nabbed from the TRM, as were probably a number of other things. But the kernel itself isn't made by Texas Instruments of course!

3.b Asking Questions About the Resource

Looking up KERNEL, I see it refers to this: Pin description as defined in /arch/arm/mach-omap2/mux.c – Angstrom Kernel V2.6.290R42. Aha! I need to look on my board for this! That makes sense, let's look up on the elinux page to a 'Software' header high up near the top.

Okay, let's check out where we can get some more information, and list it down here before clicking off into the blue.

Observation A: “Details of the Beagle's expansion header can be found in the BeagleBoard System Reference Manual in Table 20 Expansion connector signals.”
Question: Is there a corresponding Bone SRM?

Observation B: Default Configuration is done in U-Boot and can be overwritten by the Linux kernel.
Question: So does that mean I only have to change things in U-Boot, or in the kernel?

Obseravtion C: Pinmux configuration can be done in the bootloader. For the beagle, see file board/ti/beagle/beagle.h in the U-Boot sources.
Question: Where is the corresponding Bone .h? This is phenomenal. It has a bunch of things that read like:

```
MUX_VAL(CP(UART1_RX),        (IEN  | PTD | DIS | M0)) /*UART1_RX*/\
MUX_VAL(CP(UART2_RX),        (IEN  | PTD | DIS | M4)) /*GPIO_147*/\
```

Which looks pretty darn important. Maybe this is on my board somewhere?

Observation D: Recent kernels place the mux configuration in individual board files, and for the Beagle this is arch/arm/mach-omap2/board-omap3beagle.c
Question: Where do I find something like this on my boad?

3.c Looking Around In the BeagleBone

Now that I have some questions and observations, let's look at my BeagleBone. I'm trying to find something that can be used for Muxing, so I did a “find -name “arch” 2>/dev/null” I got some folders under usr/bin/, and also three different folders in /lib/modules/ that all have a /kernel/arch.

Hey! I think the arch in usr/bin is an executable, lol. Hmm, I guess find gave me files instead of folders? When I run it, I get the output armv7l. Aha, going to lib/modules had more fruit. There are three 'modules' and I'm not sure which one I ought to care about. Right now I've navigated to lib/modules/3.11.1-armv70x14/kernel/arch, and there is an arm folder. Let's see if I can tackle Observation D.

Nope, not here, this just leads us to lib/modules/3.11.1-armv70x14/kernel/arch/arm/devices/wand-rfkill.ko

Backtrack. Found mach-omap2! However, the only thing under it is a mailbox_mach.ko file. Backtrack again? Darn, I didn't find anything in any of these 3 places. Part of this could be because 'omap' is the

name of the processor on the BeagleBoard, but not the name on the Bone; and yet they seem to be interrelated and I still run across the word 'Omap' a lot. In the bone module I found oprofile.ko

Let's see if I can track down a beagle.h sort of file. Or heck, even where my u-boot stuff is.

I navigated to boot/uboot and even though I don't see any pinmuxing there's definitely some things in here, like Docs and backup. Still I can't find anything I need.

3.d Checking out the SRM

I have to remember what my fundamental question is or I'll get lost and frustrated reading documentation that doesn't parse. My question is: What am I writing, and where am I writing it, to enable UART2 at runtime without running Linux commands via PinMux?

I checked out the BeagleBone SRM and I am finding P8 and P9 Mux Options Modes. Everything here is phrased differently, so I'm going to grab information about Pin 21 as it is here.

First, Connector 21. It's Pin is B17, and that's familiar. The Signal Name is given as UART2_TXD (note I don't believe anything concerning the UART has changed from revision 5 to revision 6 but it's still worth noting).

Now, looking at the Mux Options Modes 0-3, which I find at Table 12, section 16.13.3.1 in the SRM, I see that UART2_TXD's mode0 is indeed spi0_d0 as I found earlier. It's mode1 is uart2_txd. According to this, ths PROC column is the pin number on the processor, the PIN column is the pin number on the expansion header, and the MODE columns are the mdoe settings for each pin. Setting each mode to align with the mode column will give that function on the pin.

4 Perhaps There is an Easier Way!

With the Linux community constantly growing, often easier or new ways pop up. I found this article in my bookmarks of materials I'd reviewed earlier in my quest but hadn't yet understood: http://blog.pignology.net/2013/05/getting-uart2-devttyo1-working-on.html

The article was one I used to find out that there were dtbo and dts files waiting around in the Linux distribution, just waiting for me to utilize them. In June 2013 (pretty recently, which explains a lot of older, more do-it-yourself information) these files were added.

According to this, I need to add BB-UART2 to /media/BEAGLEBONE/uEnv.txt with the key capemgr.enable_partno. He writes:

This is my current uEnv.txt:
root@beaglebone:/lib/firmware# cat /media/BEAGLEBONE/uEnv.txt
optargs=quiet video=HDMI-A-1:1280x1024@60e capemgr.enable_partno=BB-SPI0DEV,BB-UART4,BB-CANBUS1

What I construe from this is that my uEnv.txt should read

capemgr.enable_partno=BB-UART2

Hey I know where to find that uEnv.txt I found it while looking under boot/uboot/ But... how do I edit a text file on board my Beagle?

4.a Editing Remotely Trial A

Why don't I just pull it off and put it back on when I'm done. Let's see... Using reference: http://hintsforums.macworld.com/archive/index.php/t-29244.html

“scp /home/me/Desktop/file.txt user@192.168.1.100:/home/remote_user/Desktop/file.txt”

I can translate that into “scp uEnv.txt tuliashuliva@TheHyperion MyDragon/uEnv.txt” right? Uhoh, system hung. Did I do something wrong? Aha, I missed the colon. I did “scp- heyyy wait a minute... Why is there a MyDragon/uEnv.txt: no such file or directory? Aha! Fixed it. There was a missing “./”

So the final command I did was on the Hyperion and was “scp ubuntu@192:/boot/uboot/uEnv.txt ./MyDragon/uEnv.txt” Tada! And walla, the file arrived safely. Let's open her up. I made a copy of the file so I could have a backup in case I broke anything.

First thing's first, it looks like # can be used as a comment at the start of a line. There is, for example, a line that's been commented out that would only apply to BeagleBone Black, and it reads:
#optargs=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN,BB-BONE-EMMC-2G
What a great showcase of what I'm going to have to change. Now I know just about right after that I can throw in my capemgr.enable_partno.

Uh oh. /boot/uboot/uEnv.txt, Permission denied. I can't contact root@ that ip address, because what happens is that it asks me for root's password. I try ubuntu's password and temppwd, but neither works. Instead I'll work from the board and sudo -s into root. Now I'll try and reverse the command...

Let's try a “scp tuliashuliva@192.168.7.1:./MyDragon/uEnv.txt uEnv.txt”
Darn, looks like it hung. Was it the ./MyDragon? The tuliashuliva@192.158.7.1? I guessed that part

I try to ssh tuliashuliva@192.168.7.1 from within root@arm and again it hung.

I'll just try SCPing it into the Beagle's home directory and then moving the file with a sudo mv.
I got a 'failed to preserve ownership for /boot/uboot/uEnv.txt': Operation not permitted.

Hmm, how to edit a remote file over ssh?

4.b Editing Remotely Trial B; And The Discovery of Infinite NULLs

Wait I have a theory! I go to my windows machine where the BeagleBone USB is actually plugged in (I don't reroute it to the Linux machine, and on the linux virtual machine I'm able to use the ethernet

connection, which is amazingly cool. I navigate to my E: drive where the Bone is located. There a uEnv.txt! I examine it in Notepad++:

NULLNULLNULLNULLNULLNULLN[...]

Well. That is not going to work. Odd, because this looks identical to what I'm seeing in boot/uboot/ on Linux, and it's just an ASCII text file. Could we view it in Linux? I steal the USB for the virtual machine.

I think I might know the problem; the programs I'm using are attempting to open with UTF-8. But wait, no, I have definitely opened this file with a text reader once it was no longer in this location. How strange...

4.c Editing Remotely Trial – Aha! It Was There All Along!

Waa! I read something where another person had tried to mv a file and had received this error. They mentioned that the files appeared to have all transferred fine. I ran a cat on my uEnv.txt and I find that the new line is there; the only thing where permission was denied was altering the permissions. I guess this means I should have a UART-2 now? I suppose I have to restart.

First, I'll check to see if I already have it (doubtful). Nothing. Okay, I'll unplug and replug the Bone!

“ssh ubuntu@192.168.7.2” Gonna take a moment for the connection to form XD.

Damn. I rebooted the bone, but there's no ttyO1 or ttyO2... I look around but I'm bereft of explanation. Why don't I go look at uEnv.txt? Perhaps it's been altered back to its original form somehow? I run a cat on uEnv.txt under /boot/uboot, and I see optargs=capemgr.enable_partno=BB-UART2; Perhaps I've made a mistake somewhere else? Perhaps I should drag and drop the uEnv.txt over the USB connection to where the million-null file is?

4.d On the Subject of Optargs and Mmc

Perhaps I have found the correct file, but there is something else I must enable? I see #New Device Tree Boot: in the uEnv.txt file, and some materials after that. Perhaps my optargs should come afterward? I see this section running device_args, where device_args = run expansion_args; run mmcargs

Neither of these ever mention optargs... Hey, I've also finally found out what ttyO0 is! I see the line console=ttyO0, 115200n8.

Perhaps I should try =BB-UART2-00A0? No, that doesn't work my reference. But you know what's strange, I have dtbo files but no dts files! Why is that? But based on this resource:

http://elinux.org/Capemgr it is in fact the dtbo which is the 'final' device tree object I need, not the dts. I am also learning that the 00A0 is a type of version.

My BeagleBone Black tutorial says that uEnv is usually located under /media/ but this directory is empty for me. A “find -name “uEnv.txt” 2>/dev/null tells me this is the only uEnv.txt on my machine. So what happened?

Let's run a search on uEnv.txt and see if we can find a better sample. It seems the 'quiet' was kinda important. I'm looking at this resource, again concerning BeagleBone Black but I take what I can get: http://circuitco.com/support/index.php?title=BeagleBone_Black_FAQ

It asks “How do you permantly(sp) configure bone_capemgr.*/slots on boot time? Now I remember from doing that echo to bone_capemgr.*/slots that this is relevant to my interests. According to this, I see: “optargs=run_hardware_tests quiet capemgr.enable_partno=test1,bone_pwm_p8_13” It definitely looks like I'm on task. Let's see if I can make this work for me.

4.e Editing Remotely Trial C

First I want to see if I can edit the uEnv.txt file remotely using something I found called 'vim.': http://xmodulo.com/2012/11/how-to-edit-remote-file-over-ssh.html

I run vim using: “vim scp://user@remote_host//home/user/path/file” or “vim scp://ubuntu@
To my amusement, it began letting me edit the text file in-terminal. I've done this but now I want to save and I don't know how, as I've never done this before, hehe! Now looking at this resource: http://www.debian.org/doc/manuals/debian-tutorial/ch-editor.html I can see that 'vi' is the only editor that comes with almost every unix-like operating system. And 'vim' is an enhancement on 'vi.' Which arguably should mean anything that work with vi should work with vim, right?

Okay so I must have entered 'insert' mode by pressing 'I' while trying to type something (I noticed it was beeping at me but then suddenly let my insert). Now apparently I end insert mode by pressing ESC. Then I type in : to start writing a command. I type :wq, w for write (save) and q for quit. It prompts me for my ubuntu password on the Beagle, and then apparently writes the file. Let's see what happened.

I ran cat on /boot/uboot/uEnv.txt and I don't see any changes. But now that I know about vi, I wonder... can I use it from the ssh? Ha! I can!

Let's see if I can edit this file. Er, W10: Warning: Changing a readonly file. Ow, this is painful. This doesn't work anything at all like I thought it would. I've deleted the entire line just by using arrow keys, hahah! This hurts, I don't like it.

Let me try again. Okay first thing is, press the first few keys and wait patiently for things to appear. You really can't scroll away from where you started writing, or delete anything, and using Backspace just moves you back along your words like an arrow key. Anything you type works like Insert is being held for your new text, and like Insert is not being held for your old text. So if you have =|capemgr you can

type anything like “pizza” and get =pizza|capemgr, but when you push backspace, you end up with =pi|zzacapemgr, which can be confusing until you type something new in place like “napple” to get =pinapple|capemgr. Push escape and type :w! To write (the ! Overrides the readonly problem).

Drat! “uEnv.txt” E212: Can't open file for writing. How painful!

Perhaps we should try this as root? I sudo -s into root. Yes! I'm able to edit it. I had to remember not to use the arrow keys to move backwards, only Backspace. I typed i to start writing, then wrote “hardware” and messed up, I used backspace to get to the beginning and wrote the run_hardware_tests quiet and then hit ESC. After that I typed :w and :q and it worked! Whenever I accidentally wrote in arrow keys, I went back and used a :e! to throw out all changes (I couldn't seem to get u to work)

Now I have to turn off and turn back on everything and see if it worked.

4.f Is the Nonexistent MMC Responsible?

I turned the beagle off and back on, but I still see no tty2. Could it be that the BeagleBone black operates differently than the white because of the fact that it has a uEnv.txt in on board MMC?

I'm looking at this reference: http://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree/exporting-and-unexporting-an-overlay

This tells me to *create* a uEnv.txt, and mentions it only having a single line. It puts it under /mnt/boot/uEnv.txt, but I think this is still Angstrom we're talking about. Could there be some difficulty because I'm on Linux? I also notice they don't use optargs=..

...Bingo. I'm surfing through random locations when spot this line: mmcargs=setenv bootargs console=${console} ${**optargs**} vram=${vram} omapfb.mode=${defaultdisplay}:${dvimode} at: https://groups.google.com/forum/#!topic/beagleboard/zi6Ft6JI2OM

Notice optargs? Notice that it's being summoned in mmcargs, when I'm using a vanilla BeagleBone with no mmc? Let's go back and check that uEnv.txt. Yup, its the same. Run mmcargs. Now the question is, is this just some kind of convention, or is this honestly the difference between BeagleBone and BeagleBone black messing me up? One way to find out, only I'm not sure what the 'better way' for writing the capemgr stuff is. I guess I can just throw it in? So do I put it before or after the new device tree boot? Do I put it into deviceargs?

Let's just give it a shot while I'm researching. It takes a bit to do each reset, but I can certainly give them a shot. I have to get up early in the morning so I don't have much more time.

By the way it looks like you can save a file remotely without being root by using the command :w !sudo tee %, where % represents the current filename.

I don't see any ttyO1 or ttyO2, so let's see what else I can try.

I found a topic that may be relevant to my interests:
https://groups.google.com/forum/#!topic/beagleboard/-kHyFqZwEF4

It didn't work, and I noticed it was mounting a mmcblk (Black). What is it with these people, the person at the top was clearly asking about white BeagleBones!

4.g Disposing of The Infinite NULL File

I tried something new, and dragged and dropped the uEnv.txt file over the 'infinite null' text file.

Hey there is *a lot* of mmc based stuff in this uEnv.txt file. For example, the mmcroot=/dev/mmcblk0p2 ro. I don't have this folder under my /dev/ because I don't have mmc memory! I'm not sure where to put my capemgr information if not where it currently is, however. Well, let's unplug and replug and see what happens.

Ehe, I think I killed it. I probably should have saved a copy of the infinite null file! I cannot current ssh to ubuntu! Ah I fiddled with the VM recognizing and then unrecognizing the USBs and now it works. Well let's check... Alas. TtyO0 remains alone.

Here I can find an example of someone using a BeagleBone White who also is having trouble:
https://groups.google.com/forum/#!msg/beagleboard/-kHyFqZwEF4/tqmyo5Po7YUJ

5 BRILLIANCE STRIKES!

Ha! I'm looking under /boot/uboot/debug, and I did a cat cape.txt I see some yummy stuff.
It says slot #4: Override Board Name, 00A0, Override Manuf,BB-UART2;

Now I see “before slot-4 BB UART2;:00A0 (prio 0)”. There's a hint right there; I need to get the semicolon out! “Requesting part number/version based 'BB-UART2;-00A0.dtbo”

Muahahaha! It's the semicolon! But hey, I thought I edited the semicolon out of one of my files (the one I terminaled in, and the one I manually dragged over the infinite null file). Maybe I can figure out which one is doing the work if I check them both out? Oops, I accidentally deleted the supervital uEnv.txt while trying to delete the swap file! Well, now I'll never know.

Alright, I changed just the /boot/uboot/uEnv.txt file. If I see some debug errors, I'll drag the correct one into 'boot.'

5.a ...And Causes an Explosion

Quite suddenly, and slightly to my horror, only one USB device is being recognized from the bone. Furthermore, the LEDs aren't blinking in a heartbeat pattern. Oh no! I have broken it! Odd. Three of the four userLEDs have come on, and the data virtualized USB isn't showing up. That means I can't even reach the SD card over the USB link.

Is the ethernet connection still being made? I can't tell. I quickly take the one USB that IS showing up and I route it to the virtual machine. Nothing shows up drive wise, so I'm assuming this is the one I required for the ethernet connection. It's the FTDI one. BeagleBone/XDS 100v2. Crap. Ssh ubuntu@192.168.7.2 is hanging.

Wait a minute... Is there anything about the uEnv.txt that should cause this kind of alarm? One supposes it might be possible if, for example, the permissions on uEnv.txt got screwed up when I accidentally deleted it. For example, what if uboot can't even set itself up right now? Bullocks, once a file is gone in Linux it's gone. My accidental mistype where I forgot a ~ may have caused me hours of confusion now and denied me my prize! The biggest problem is that I now find myself unable to access the SD card! This is really weird, I don't even see the option to eject USB devices showing up in my Windows machine.

Again the LED lights come on just 3 activated, not toggled to any heartbeat. Something is terribly wrong! Crap. Whatever I did, I made it so that the little Bone isn't able to set up its own internal server.

5.b De-Exploding the Explosion

Okay no more panicking. Let's see if we can get the sdcard reader online to tell us what on earth happened. I have the card pressed into the reader, and I just got a signal from windows that the driver installed successfully. Now I'm waiting on the little guy to show up in MyComputer... It's... not showing up! I sincerely doubt I managed to fry an SD card with some programming but- bam it just appeared.

Fascinatingly, there is no uEnv.txt on my new I: drive that represents the BagleBone. Okay, I don't have the uEnv.txt file on my windows machine. I could toss the USB to Linux but it took so long to show up I'll just git pull, add, commit, push, and repull the little file to windows.

I dragged over the uEnv.txt file and I see the LEDs doing their heartbeat thing. Success! Disaster averted!

5.c And Guess What Suddenly Exists?

/dev/ttyO2

Booya. PinMuxing completed.

X Appendix B: Communications and Cross-Compiling

1 Libserial

The best way to describe the Linux experience is probably this: If there is one way to do something (and there is) then there are two ways to do something... and conversely, whenever you try to learn how to do something new, you will have to reconcile fifty-two different people telling you fifty-two different ways to do basically the exact same thing from fifty-two unrelated but easily confused attack routes, none of which exactly how to tell you exactly what you want to do.

On the first hand, however, if you run into a dead end trying to do something, the best route may back to back up, look around, and see if you can find a completely different path to your destination. The second path might be friendlier.

Before I went to bed last night, I cheated a bit to make sure I knew where I would be started on the next day; I opened up my bookmarks that dealt with reading to serial ports and I pinned up some tabs that had to do with a library called 'Libserial,' which I felt was most in line with things I already understood and there would be a good starting point for talking to my other board. I had other options, actually, such as ttystream and terminos, but I decided to go with Libserial as my first try. It seemed pretty darn straightforward to do things such as setting the baud rate.

Anyway, after downloading the files, extracting them, and trying frustratedly to compile them I realized I was in for a bit of trouble. So when I got to work today I started trying to ./configure, make, and make install the library. No dice. No matter what flags I set or what configuration commands I tried, nothing worked. After about an hour of frustration I 'backed out,' looked around, and found a 'package' on ubuntu called libserial-dev. After fooling around with apt-get (I'd never used the command aside from through a tutorial) I used apt-cache search libserial to find the package I wanted, and then apt-get install libserial-dev to get the package. When I popped open Eclipse and right clicked on SerialStream.h and clicked “Go to Declaration” I was able to zip straight away to the libserial header file SerialStream.h. I'm on target!

2 Organizing My Ducks

The objective then is to implement some SerialStream.h code and get it to compile without causing any explosions. If that goes well I'll hook the boards up to eachother and to one of the motors and start playing.

After getting my program to correctly hunt down and find SerialStream.h, the next thing I did was try and segregate the parts of my program into classes. I admit that I've been partially spoiled by all my Actionscript. Actionscript looks like C++ (it basically wants to *be* C++), but it has a number of conventions that I've grown accustomed to that simply don't work with C++. For example, in Actionscript, I typically have only one class per file and all of them are nestled in classes. So, obviously, if I want a bunch of utility functionality or enumerations, I create a class filled with nothing but constants and static functions. That's not how things go in C++. For now I've thrown some utility functions into a .hh (C++ header file) and it will have to work for now.

I am still getting a “architecture rejected target-supplied description” when I to run the debugger, but that doesn't matter because at least it's compiling.

Let's see if the program showed up on the BeagleBone and is functional.

Whoa!!! I just found something neat! When I typed ssh[space] and hit tab twice, a list of locations I could ssh to popped up. One of them was 192.168.7.2! Never lose your Beagle again!

I ran LEDHello and all my files failed to open XD I wonder why. Perhaps I need to run as root.

Yup. I switch to root and ./LEDHello blinked my LEDs. I'm sure when I eventually configure this program to run at boot, I'm going to encounter permissions issues I won't be able to see because I won't have a console open, so this is a mental reminder to log them somewhere.

3 Starting a New Project File

Alright! Time to duplicate this project, more or less, and name it something more appropriate. As easy as Copy and Paste with the right click context menu. Great! Let's git push this.

Oh before I do I try to build my new project named Chloe. I set up the automatic builder, which of course fails to launch. I get another 'white' (non-executable file) built to root@arm. This time I check out chmod –help and I manage to figure out that I need to write chmod +x Chloe not chmod -x Chloe. Chloe runs and is splendid in fiddling with the lights.

Uh oh, something interesting started to happen while I was coding and moving things around. Some time ago (I'm not sure when) I started getting an error with the file upload, not just with the target rejecting the architecture. I remember this error, although my concept of what the fix was was never exactly clear. It's 'Launching Chloe Debug' has encountered a problem. Error during file upload. Clicking Details>> yields a bunch of empty strings, like so:

Missing element for: ''

> Missing element for:''
>
> Missing element for:''

Odd, right?

I'm not seeming to be able to trace back to where the problem was. I rm .gdbinit from the folder. I checked out my run configurations and the location for the remote file was *weird.* Seeing this I changed it back to /home/ubuntu (It had ended up Chloe/Debug/Chloe somehow) and now I'm at least getting this:

Operation failed. File system input or output error.

Guess what? “Chloe” the program reappeared finally on the BeagleBone. Which means this is the error I get (remarkably similar I know, only differentiable by the details) that produces Chloe on the board. Now I have to chmod her. Yes, she runs!

It should be noted I'm not getting the 'target architecture rejected' error right now. Not sure why. I'm curious so I put a chmod +x Chloe to be executed before the application in run configurations, and then removed her file from the board. Her file still appears white. Hmmm cmod +x /home/ubuntu/Chloe refuses to give her the executable privileges she needs.

Can't figure out how to get the debugger to work. Focusing on git for a moment.

4 Slowly Coming Upon the Problem

Hey cool I got a new error! I was checking out this resource: https://groups.google.com/forum/#!msg/beagleboard/27JofM_ShS0/XhYBE7qHBJ4J which advised me to put the full path and name of the file I wanted to run on the remote end, not just the path. Now I'm having a problem: error while launching command gdb-multiarch –version. Not sure what that means, but why don't I go back and fiddle with some of the things I changed around earlier in the run configurations?

I changed the debugger back to gdb isntead of gdb-multiarch. Now I have:

Error in final launch sequence

Failed to execute MI command

source /home/tuliashuliva/MyDragon/Chloe/.gdbinit

error message is there is no such file or directory! Huh! That's odd cause I'm modifying it right now! Hehehe. Alright, let's see what happened. I changed that long path back to just .gdbinit Now I get the old familiar:

warning: Architecture reject6ed target-supplied description

Error in final launch sequence

Failed to execute MI command:

-target-select remote 192.168.7.2:2345

Error message from debugger back end:

Remote 'g' packet reply is too long

Whoa whoa whoa, what's 2345? Is a port not open?

Well that's interesting. I changed that .gdbinit to ./.gdbinit, assuming there was some magic .gdbinit in the sky acting as the default somewhere. And I get this: Error in sourced command file: undefined item: “arm” I take this to mean we've never before successfully targeted this little file? I throw in 'auto' instead of 'arm' (not sure why that broke it) and I get the old 2345 message again.

Remarkably, google cannot succesfully make 'set architecture arm' work as a key term XD. It turns up nothing. Although I would like to know that along the way I've been able to successfully chmod Chloe by running the debugger! Chloe is an ELF 32-bit LSB executable, ARM.

I don't think this is applicable, as it would have been updated before I downloaded my version: http://software.intel.com/en-us/forums/topic/456618

Let's try a apt-get install gdb-multiarch...

Aha! I was now able to use gdb-multiarch, something suggested by several sources including here:http://stackoverflow.com/questions/13678923/eclipse-failed-to-execute-mi-command Now I'm getting (and I might have been getting for a short whiel before this but didn't notice)

Error message from debugger back end:

192.168.7.2:12346: Connection timed out

I've tried a number of things, looking at bitbake and various problems with gdb, but I think my error may be more significantly in the timeout itself.

I can't get the debugger to work, but I have managed to come to understand a great deal about what's going on when it comes to my debugger, and to understand other things like how to set permissions. The greater understanding was worth the exploration; however I'll simply have to tackle the debugger another day.

https://bugzilla.redhat.com/show_bug.cgi?format=multiple&id=612578

5 Everything Seems To Compile Now, So...

I git pushed LEDControl that links in LibSerial, and I've started to program ServoControl.

Okay I have to initialize the stream communications. I know the Baud Rate but I need to know char size, stop bits, parity, and some other things. Let's see if I can find an old file in my bookmarks to help.

I couldn't find anything quickly so I found a quickstart guide for an unrelated beaglebone thing and applied its findings! http://beagleboardtoys.info/index.php?title=BeagleBone_RS232 Actually it looks like in this question:
https://groups.google.com/forum/#!msg/beagleboard/SgZtBcGHJvU/B42EpewnhMkJ the asker was directed to generic Linux Serial how-to! I hope this works. I looked at the settings here as well:
https://groups.google.com/forum/#!msg/beagleboard/SgZtBcGHJvU/B42EpewnhMkJ

6 Or Maybe Not

I am having a HELL of a time trying to get libserial to work. The problem seems to be a library linking one, as after everything looks clear I get errors that symbols cannot be resolved (for example, SerialStream() clears the first round of errors, only to be picked up by another.

So in trying to link in the library I need I'm confounded by all the Linux people who choose to build from command line. That might work for them but I personally like to stay in one program at a time and not leapfrog around like a crazy person. However getting advice on how to input materials into the eclipse editor can be irritating, especially with Linux 'elitists' complaining in every forum about how we newcomers have been corrupted by talk of adding libraries to projects.

Poo poo that. So you want to be old and do it the hard way, so what? I don't have time to learn everything there is to know about the universe to run something. Show me the fastest route from A to B; I'm a designer, not an engineer, and if you're on stackoverflow to have meaningful conversations about the intricacies of Linux-Awesomeness, you're in the wrong place. We're all newbs here, man, and we have projects to do. We have work, deadlines, schools; we can't operate on your hypothetical perfect-grasp-of-everything timetable.

Alright this is the error: undefined reference to 'LibSerial::SerialStream::SerialStream. A similar error for SetVMin and SetVTime, although strangely enough the IsOpen isn't throwing any errors. Odd. Maybe it's defined entirely in the header files? We'll find out. The strange thing is, the lib exists at libserial.so, but every time I try to link it with -l it tells me it doesn't exist!

So I'm trying everything I'm being told to try. I altered the compiler command to include a -lserial at the end, and I also included the /usr/lib/ folder for searching in. It seems Eclipse has just convinced itself this file doesn't exist, despite every evidence to the contrary. I can clearly see that libserial.so

exists under /user/lib/ It is a symbollic link most likely to libserial.so.0.0.0. I should note that there are possibilities that the libseria.so.0.0.0, which is an ELF 64-bit, is conflicting with something 32-bit and arm related. Its possible I had to compile it with arm libraries, though if that is the case I am definitely screwed at this period in time.

I tried #pragma comment(lib, "/usr/lib/libserial.so); but apparently eclipse is sent to ignore 'unknown pragma comments'? I know this is what I used back when I was including my own static libraries in 3D engine development. Looks like that might be windows only.

-L/user/bin/libserial.so doesn't seem to throw any errors, but it hasn't fixed any problems. I'm still getting undefined references. And I'm told this isn't the way to do it.

Cannot find -llibserial.so

Okayyy, how about I type in the name /libserial.so?

Cannot find -l/libserial.so

That's absurd, it's right under the -L I included. Weren't you paying attention? Because in the terminal window, file usr/lib/libserial.so definitely returns results.

Cannot find -l/usr/lib/libserial.so

Oh reallllllyyyy? Back up. This is getting us nowhere. Let's try another route. This reference explains how to do it from the command line: http://stackoverflow.com/questions/2272200/undefined-referance-to-libserial

which is to write: "g++ -Wall -lserial -o gen1 gen1.cpp string2num.o" However, I believe I am already doing this. In my Build Settings, under my arm-linux-gnueabi-g++ compiler, I see:

-O0 -g3 -Wall -c -fmessage-length=0 -lserial

I'll navigate to Discovery objects, and under Compiler invocation arguments I'll throw a -lserial. Nope, still covered in undefined references.

At last. I navigated under the linker tools and threw in the -lserial. Whatever I needed, it's been found. Let's see if it works on ARM, however.

I try to run a debug of Chloe and I see this:

Program file does not exist

/home/tuliashuliva/MyDragon/Chloe/Chloe/Debug/Chloe not found.

That's terrible. Let me go find it for you. Also I should remove the file remotely so I can tell when it's being replaced. Legasp it is true! There is no Chloe in here. How can I fix that?

Mm! I found the problem. It's in my console pane, it doesn't show up under problems for some reason.

/usr/lib/libserial.so: file not recognized: File format not recognized.

Let's check out this resource: http://stackoverflow.com/questions/3191706/shared-library-file-format-not-recognised where it says if I'm cross-compiling an executable, I also need to cross-compile all of the shared libraries it depends on, and link against those. Yowch, back to square one.

7 Back To the Drawing Board: Cross-Compiling

I need to take a moment to think about this revelation. I had run into bizarre technical difficulties trying to install LibSerial using my on board compiler, much less trying to cross compile. To be honest I've been skitting around truly understanding what 'cross compile' means because I've had prebuilt tools available to me.

A apt-cache search on libserial shows me libserial-dev, which was the think I used apt-get install in order to install.

There are a lot of keywords I could search for at this time and I'm not sure where to start. It might be time to put the BeagleBone down for the night and just focus on my presentation for the morrow and furthermore working on my thesis for a bit. After struggling with ./configure on my libserial tar, I'm not exactly eager to repeat the process.

Looking at a sample, such as at this resource: http://www.cs.cofc.edu/~robotics/wiki/index.php?n=Main.Help I can see a:

./configure --host=arm-linux --prefix=/opt/arcom/arm-linux

This will tell the make binary to cross compile your libraries, and to put them in the correct place instead of overwriting the system ones

I can't be sure what goes where with this little cross compiler, but I can at least tell that if I take this route, I'm going to have to first of all figure out where LibSerial is going to get ahold of bzero. But you know this is a learning process. So let's try this. It's just occurred to me I'm never going to be able to get any of my big sound libraries on Chloe if I can't figure out how to use the *cross compiler I already have*.

So I've download libserial-0.5.2, which hopefully is the version I already have or I'm in trouble. I'm studying this resource: http://www.ailis.de/~k/archives/19-arm-cross-compiling-howto.html#db and I see that ./configure –target=arm-linux is of significance; also ./configure arm-linux --target=arm-linux --with-shared –prefix=/usr But I don't understand the significance of either or what they truly mean. I know that I'm basically 'targeting' them towards arm, but does arm-linux mean *normal* linux to arm or linux-on-arm? Should I be replacing this with my arm-linux-gnueabi materials? What about that x86_64-linux-gnu bundle I have?

7.a I Find Guidance!

Ooh, this is a nice resource herE: http://stackoverflow.com/questions/11796127/how-to-cross-compile-c-library-with-dependencies . It reads: “Generally, to cross-compile an autotooled package, you pass a couple of extra arguments to ./configure: --host and --build. --host is the name of the system that the built programs will run on, and --build is the name of the system that does the compiling.” He gives a long example using i586 and i686. Let's puzzle out what my configure might look like- you know if it wasn't already missing simple things like string.

Say... what is my host architecture? “Each toolchain has its own subdirectory under /usr/” Holy this is a good resource. Look at this additional gem: “When I say "name of the system", I mean a tuple of the form ARCH-VENDOR-OS-LIBC”

arch... That's arm. Os is Linux. LibC is the gnueabi? He gives the example of: “i686-pc-linux-gnu.” Pretty close! This person is wonderful. Okay so my target (my build) is probably arm-linux-gnueabi. So who

am I? Am I x86_64-linux-gnu? I think I know a way to find out. Where did that old libserial library get to...

Let's do a file /usr/lib/libserial.so.0.0.0. I get this output: ELF 64-bit LSB shared object, x86-64, etc. Looks like my architecture is x86-64/x86_64, as this is where “ARM” would usually go. Something really really important to note is that 'host' doesn't mean here what I thought it would, and this might have turned me away from other resources in confusion. Here, he establishes clearly that the --host is the destination machine, not the machine 'hosting' the little BeagleBone. And –build is the machine that will build the libraries.

./configure --host=arm-linux-gnueabi --build=x86_64-linux-gnu --prefix=/usr/arm-linux-gnueabi

This looks very nearly correct. I honestly don't know if I should be using gnueabihf or not. He also recommends some --enable-static and some –disabled-shared. I won't try those just yet. Let's see what I get. The configuration worked, as nearly as I am able to tell.

However we're still having trouble with make, which cannot seem to find bzero. Digging up this old resource where I was able to get rid of bzero but not abs earlier: http://forum.arduino.cc/index.php/topic,39321.0.html I think I will manually go in and edit the.h files. Let's see what happens.

I have just liberally applied includes of string.h and stdlib.h in every .h file of the library. It looks like make successfully completed without errors. I ran a make install and it exited with an error, but looking at the last information I see “Cannot create regular file […] permission denied”

Let's try a sudo.

Okay. I think it worked. Its hard to know. Why don't we go check out our gnueabi and see if we see new files? I see a libserial.a, a libserial.la, and a libserial.so. Looks pretty darn good, actually.

Now that I know I most probably have all my libraries properly cross compiled (and it really wasn't so scary, man that post... That person was a wonderful person) I'm back in my eclipse trying to see where I left off and where to go. Right now I'm seeing undefined references to my stuff. I think I should go in and change some of my include paths.

7.b The Trouble With Libraries Is...

Just realized I had a -L/usEr/lib eheheh. I've changed it to /usr/arm-linux-gnueabi/lib, even though I'm pretty sure it already knows to look there. Now I'm going to try and put a libserial.so under -l. Perhaps the reason it could never find anything was because libserial.so simply wasn't compiled for ARM. Drat, cannotfind -l/libserial.so.

I would like to mention I just found a SerialStream.h underneathe user/arm-linux-gnueabi//include, which is listed as included in my project. If only I knew why they were undefined. Am I not being specific enough in my .h?

Is there a way I can see if libseria.so is making it 'into' Chloe? Not really, she isn't building at all.

Cannot find -l/usr/arm-linux-gnueabi/lib/libserial.so Hmph. That's unfortunate. Libserial.a? No...

This time I try talking to a different part of the interface. I go from Properties → C/C++ General → Paths and Symbols → Libraries (& Library Paths). I include a path to /usr/arm-linux-gnueabi/lib/libserial.so, which I was able to directly browse to.

I add a reference to .a for good measure. I just ended up with a make file error again, so let's see if I can resolve this. I change Builder Settings back to Internal Builder.

Cannot find -l/usr/arm-linux-gnueabi/lib/libserial.a

So, what's the scoop? Obviously they have to be there, because the interface I used let me navigate straight to them and select them. Let's go check out those files. Are they compiled for arm? They are:

ELF 32-bit LSB shared object, ARM, version 1 (SYSV), dynamically linked

Lovely. So what's the beef? Hmm, interesting, check this resource out:
http://whatwouldnickdo.com/wordpress/328/eclipse-cdt-and-linux-libraries/

He says he's adding libavcodec.so so all he needs to write is 'avcodec' sans lib or .so. Let's try this.

7.c Wow, That Actually Worked?

Something weird happened. When I put in 'serial' the errors concerning finding my libraries dissappeared! In fact, two of my undefined reference concerns also disappeared. All I have left is an 'undefined reference' concerning new LibSerial::SerialStream. Fascinating. Is this the same bug, or have I actually fixed something? Is there another library I need?

I swap out new SerialStream([..]) with all my options with a quick new SerialStream(); It works. Interesting. How about I go and manually call all the functions I need to set the baud rate and so forth?

Fascinatingly, nothing is broken. Build complete for project Chloe. Let's push it? Pushed. Now let's go over to our root@arm terminal.

./Chloe

./Chloe:error while loading shared libraries: libserial.so.0: cannot open shared objectfile: no such file or directory.

AHA! We are getting somewhere! Let's draw them lib files in. I'll try a simple apt-get install libserial-dev on the board itself. If that doesn't work, I'll backtrack.

Strange, something wicked happened with- Ah. Aha. The Eternet Cable. I shall summon it. Ethernet cable is in now, let's give this a shot. Nope, the Bone still doesn't want to ping www.google.com. I'll try unplugging it and see what happens.

Unplugged and replugged, and now I can ping google. Let's try that apt-get install now.

It has just occurred to me that Chloe can be configured with wireless and post tweets, such as when she can't reach her charger or is playing with Tychus. I am so happy. It looks like thei nstallation is working, and seeing as its only a hundred kB or so I'm not worried this is a risk.

8 Omigod

Omigod its running. Uh, uh, It's running!

Okay I see

“Beginning to commune with Serial
Enter Input: (but when I enter my input, I get a slew of InputToBeTransmitted EnterInput: which follow, suggesting I'm not handling the return key well.”

'quit' successfully terminates the program. Wahoo! I should wait for The Captain before I rewrite everything, to be safe.

Figured out what the issue with cin is: it's interpreting every space as a deliminator. So it's transmitting #0\r\n P50\r\n and T500\r\n

I'm looking at a reference here that might be able to help...
http://stackoverflow.com/questions/2765462/how-to-cin-space-in-c

noskipws? Really? Like: cin>>noskipws>>input; ?

Chloe is failing to build to the board again, heh. Let's try and find out what the problem is. Part of the trouble may have been I accidentally click 'run'. Good, the Chloe program showed up. Lol! That was not the way to solve the problem! I got an infinite loop of input this time. I think the problem might be that noskipws failed to skip newlines!

I'll try a getline(cin, input);

As we try to determine why it is our control board is unable to sustain a signal, I have finally got the servo to move in the reverse direction, but not by entering any commands that make any sense. Clearly it's a power problem, and when The Captain returns, I shall tackle it!

I call this: Victory.