5-2017

Project Arduino

Kevin Ye
University of Tennessee Knoxville, kye2@vols.utk.edu

Gregory Rouleau
University of Tennessee, Knoxville, grouleau@vols.utk.edu

Alan Person
University of Tennessee, Knoxville, aperson1@vols.utk.edu

Jabril Muhammad
University of Tennessee, Knoxville, jmuhamm1@vols.utk.edu

Project Arduino: Fin
11/29/16

Team 6

Alan Person - Team Leader
Jabril Muhammad
Gregory Rouleau
Kevin Ye

Customer: Michael Thomason

Table of Contents

Executive Summary

The Arduino Integrated Development Environment (IDE) is an open source, multi-platform, software application designed for programming microcontrollers. The application itself attempts to make such programming a smooth experience by including several quality of life features, which makes things such as syntax checking and compile-and-load operations simple. However, the IDE does not support several features viewed as important by many of its users, which causes certain applications, such as assembly programming, to be exceedingly obscure.

The goal of this project is to expand the functionality of the Arduino IDE by adding a specific set of “missing” features that will benefit those who wish to experience ARM assembly programming. Furthermore, students taking Computer Science (COSC) 130, Computer Organization, at the University of Tennessee stand to benefit the most from these new features if properly utilized. ARM programming is not a particularly fun nor easy skill to learn and as COSC 130 students are required to learn this skill, Project Arduino needs to make assembly applications within the Arduino IDE a much simpler as well as consistent experience.

The major requirements for Project Arduino include multi-platform compatibility, assembly friendly features, and user-interface changes. These include adding the ability to compile, create, link, and filter assembly files. Users will be able to export a sketch - an Arduino project that is stored in a .ino file - to assembly and save the corresponding .s file for editing, execution, and/or general viewing. However, if a user wants to create a program in pure ARM assembly, then it will be possible to do so and link the resulting program in compilation. Filtering an exported assembly file for relevant, high-level user code will also be available and serves to benefit those learning about functional assembly code.

The improved IDE has been tested on multiple platforms to guarantee that it is fully functional on different operating systems. These platforms include Windows, Mac OS X, Ubuntu, Fedora, and Red Hat. Results from testing have shown that the new features function properly according to the requirements, though the actual implementation could use some refinement due to the problematic nature of how the Arduino IDE handles compiling projects.

Throughout this project, the team has encountered several issues. For the most part, these issues originated from the base implementation of the Arduino IDE; however, some issues stemmed from requirements that simply turned out to not be viable within the scope of this project. Despite the inevitable issues innate to group projects, the team managed to find solutions to all of the major requirements and complete the project.

While the project has been completed, there is plenty of room for improvement upon the new features of the IDE. Potential improvements include adding assembly syntax highlighting, a debug function, and a more sophisticated filter function to further reduce extraneous assembly.

Introduction

The Arduino open-source electronics platform is popular among teachers, students and DIY enthusiasts alike. The hardware side of Arduino is a series of microcontrollers that currently features ARM, AVR, and x86 architectures. These boards are cheap and the development software is open source. The Arduino programming language is derived from C++ and is thus familiar to many users.

All of the features offered by Arduino lead to it being an integral part of the curriculum in COSC 130, Computer Organization, here at the University of Tennessee. Students in Computer Organization do almost all of their labs in C++ and assembly using the Arduino Integrated Development Environment (IDE) to execute their programs on the Arduino Due microcontroller. The boards are used to teach students practical applications for ARM assembly and how to program hardware level functionality, such as analog and digital reading.

However, the Arduino IDE does not support working with assembly files. The way that students currently do their assembly labs involves wrapping their assembly code in the ASM wrapper provided by GCC. This adds an extra component to debugging and results in clunky code. It also does not provide a complete assembly experience to students, as they never work with pure assembly. There is a level of abstraction that C++ puts between students and the assembly they are working with; this creates the potential for extra confusion among students.

Project Arduino seeks to develop native support for assembly files. The goals of the project are to allow students to import and export assembly to and from a sketch, which is the code that is uploaded and run on an Arduino board. These features will be integrated in the graphical user interface (GUI) of the Arduino IDE for usability.

The new capabilities of the IDE will improve the way students interact with assembly in COSC 130. These students will be able to work with native assembly as an integrated part of their labs involving the Arduino Due board. Project Arduino endeavors to create a smoother, more involved assembly experience for students in Computer Organization without introducing any extra layers of abstraction.

Requirements

One of the first steps to creating and planning Project Arduino was establishing the requirements for the project, as determined by the customer. After many hours of planning and discussion, the team came up with the following list of requirements that the project would need in order to meet the customer's specifications.

1. Platform
    - 1.1. The features added must function on Fedora 23 and later.
    - 1.2. The features added must function on Red Hat Enterprise Linux 7 and later.
    - 1.3. The features added must function on Ubuntu 15 and later.
    - 1.4. The features added must function on Mac OS X Version 10.10 and later.
    - 1.5. The features added must function on Windows 7 and later.
2. Functionality
    - 2.1. The IDE must feature ASM file support.
        - 2.1.1. The IDE must be able to export a project in assembly.
            - 2.1.1.1. The IDE must be able to export assembly automatically when compiling.
        - 2.1.2. The IDE must be able to filter exported assembly to instructions.
        - 2.1.3. The IDE must be able to link assembly files.
        - 2.1.4. The IDE must include an assembly code template.
3. Views
    - 3.1. The interface must have an option to export a (.s) file
        - 3.1.1. The interface must have an option to automatically export a (.s) file.
        - 3.1.2. The user must be able to choose a folder to export to.
    - 3.2. The interface must have an option to import a (.s) file
4. Stretch goals (time permitting)
    - 4.1. The IDE must feature support for debugging.
        - 4.1.1. The IDE must be compatible with a hardware debug adapter.
        - 4.1.2. The IDE must have a debug function.
            - 4.1.2.1. The debug function must be includable in sketch code.
            - 4.1.2.2. The debug function must be able to extract register contents.
            - 4.1.2.3. The debug function must allow register contents to be altered.

Change Log

Stretch goals, requirement 4, from the original project requirements was removed. The financial cost to pursue debugging was beyond this project's budget and was subsequently agreed upon with the customer the pursuit of debugging is out of the scope of this project.

Design Process

Components of Design

The major requirements for the project were separated into four components: the export feature, import feature, GUI alterations, and testing. These components are comparatively sized and could enable working on the project in parallel. Like most projects, the majority of time spent on the project will be focusing on functionality and subsequent debugging. Early on, the team had a good idea of how to pair the minor requirements with the major ones so that duplicated work could be minimized. The components of design broke down into the exporting feature with filtering assembly code, the import feature with the assembly template, and the GUI changes with testing.

Questions That Need Answers

During initial project development, many open-ended questions were floating around about how to implement the export and import features. The requirements involved modifying existing compile commands used to compile sketches; however, the Arduino IDE uses a third-party compiler. Due to this, the team was unsure of the possibility of implementing the major features without also needing to modify the compiler that the IDE uses. Fortunately, the team found prototypes for the export and import features without needing to modify the third-party compiler, which also achieved the features' requirements.

Regarding the project's stretch goals, the team was unsure of the possibility of implementing features that would provide GDB-style debugging while using an Arduino board. Arduino boards lack components necessary to enable real-time manipulation of variables or breakpoint-and-step style debugging, notably a lack of permanent storage and the necessity of re-compilation-and-upload for any changes to take effect. This issue immediately limits the potential of debugging support since it means that individual registers cannot easily be modified during execution on an Arduino microcontroller without additional hardware. Even with an intelligent solution to this problem, research showed that every route to debug an Arduino board requires an ARM JTAG adapter or some combination of additional hardware and software to even enable debugging. The cost of this extra hardware was too expensive for the project's budget to pursue and the additional cost for students was deemed excessive compared to the added benefits. It was later agreed with the customer that pursuing debugging was not feasible for the project, as well as the EECS department at UT, and can be removed from the project's requirements.

Alternative Solutions

Several solutions were considered to fulfill the requirements of the project. The alternative solution for the export feature involved using complete disassembly dumps. An alternative for the filtering feature was simply to use a shell script to extract specific contents of an assembly file. As for the import feature, parsing lines of code within a sketch file and wrapping them in the ASM wrapper could achieve it.

The first major goal is compiling a user's Arduino sketch code to assembly, which can be completed in a variety of ways. The first potential solution was to compile the sketch using the existing Arduino compiler and then use an ARM disassembler to get the resulting dump of assembly. However, using a disassembler has significant disadvantages; namely, the output code is likely going to be difficult to read due to information lost in the process, such as comments and variables. The second solution is the addition of another compiler that will offer the desired abilities. This may offer more tailored output but it may be difficult to find a compiler that would offer the desired features on every target platform; however, use of an additional compiler may introduce licensing conflicts limiting redistribution.

The second major goal of the project is to ensure that assembly produced is filtered to only include assembly corresponding to user code. The Arduino Due uses an ARM microcontroller without an operating system, meaning that assembly files of any sketch are going to be very large due to having to include all of the operating system related assembly code as well. At first glance, this task seems fairly simple; however, the main difficulty arises from the variation within exported assembly files. For example, the names of a sketch's functions will not be kept once the program is exported to assembly. Beyond that, the order of function appearance in produced assembly files is not going to be the same each time. This means that if the user includes multiple functions in the original sketch, the functions in the corresponding assembly file will only be recognizable by examining the assembly code. For simple sketches, a simple shell script that extracts the assembly file contents contained in the required function names, setup and loop, would be sufficient; however, this strategy will not work for sketches that contain more than the required functions.

The third major goal of the project is to allow students to natively link in assembly files. Currently, in order to program in assembly, students wrap each instruction in the ASM wrapper provided by GCC. Using a wrapper for assembly instructions limits functionality to a subset of assembly instructions and hinders programming. Parsing assembly files into ASM wrappers is a partial solution to this problem; however, this would only provide a quality of life increase instead of extending functionality. This eliminates the need for the user to wrap the instructions themselves, subsequently increasing readability and productivity. This will provide better feedback to students because it allows them to create a direct connection between input and output instead of working through an extra layer of abstraction.

Clearly, these alternatives mostly involve partial solutions. These would be sufficient for the project but would be too simplistic and erroneous to implement all of the features in a flexible manner. However, if too many issues were encountered while attempting other implementation strategies, these alternatives would have provided a decent backup plan.

Selected Solution

The selected solution revolves heavily around the existing capabilities of the Arduino tool chain. Using the current Arduino tool chain is more efficient than making an entirely new tool and minimizes the potential for licensing conflicts when the project is redistributed to students taking Computer Organization at the University of Tennessee. The base tool chain offered simple, native solutions to the problems that this project involved; finding the right solutions were a matter of exploring the functionality and reverse engineering the tool chain to suit our needs. Exporting to assembly, filtering the output, as well as importing and linking assembly files can be done with existing tools.

The exporting feature compiles a user's sketch (.ino.cpp) into assembly (.s). The major components of the assembly exporting functionality were prototyping a compile statement, implementing an execute call in Java, matching compile flags to variables, and updating to use variables instead of hard-coded flags. To develop a prototype compile statement that would compile sketch code to assembly, the team examined the existing Arduino IDE compile statements that are used by the IDE to compile user code into an executable to run on Arduino boards. The next step to implement the exporting functionality was to add the compilation command from the previous step into the IDE. This added the minimum functionality to meet the requirement that the IDE be able to export a project to assembly. After the ability to compile was added into the IDE, the team matched some of the compile flags to variables used by the IDE. This greatly increases the robustness and flexibility of the additions to the IDE. Finding the variables to be used involved additional reverse engineering of the existing IDE to determine where flags could be found when the IDE compiles to an executable. The final step was to update the Java code to use these variables instead of hard-coded flags. This process of achieving the minimum exporting feature also implements the filtering feature. Exporting the user's sketch code results in almost pure user assembly. It reduces approximately ninety-percent of the non-user assembly that is outputted during exporting. This isn't perfectly "pure" in the sense that exported assembly still contains some extraneous output, especially the assembly header and footer, but it is reduced enough that student users can be told to ignore the extra assembly.

The idea behind importing is that users should be able to compile and link assembly files for execution regardless of source. For this functionality to meet the design requirements, the feature must be able to take an assembly file, with the .s extension, and do all of the standard compilation steps to produce an executable that can run on the Arduino Due microcontroller. This process includes the following components: compiling the assembly file itself along with any operating system related Arduino libraries, generating an assembly map, producing a .elf file, and producing the final binary executable. Each of these components breaks down to separate, but related, system commands. To develop the overall feature to meet the requirements, similar steps have been taken as discussed in the previous section. Each command was implemented as generic Java statements compared to the original hard-coded shell scripts. Implementing each of these components one-by-one into the IDE resulted in meeting the minimum requirement for the importing feature.

After an assembly file is compiled into a binary executable, the user must be able to upload the generated executable to an Arduino Due board. Before an executable can be uploaded to the board, the board itself must be erased. The IDE handles erasing the board, writing the executable, and verifying the data on the board. This simplifies the user experience. To implement the upload of an executable, the existing IDE was examined to determine the current implementation to be used.

Once the core import functionality was achieved, an assembly template would allow a user to program in assembly. Base assembly templates can be created by exporting a blank or existing sketch to assembly. Since the filtering is not perfect, there is some excess operating system related assembly in every template generated. However, it is reduced to the point that users can be directed to program in a certain part of the file. The resulting (user-edited) file can be compiled and executed by using the later three commands discussed in the previous section covering importing, which is generating a .map file, creating a .elf file and producing a final .bin executable.

When it came to expanding the GUI, adding the various options to the existing toolbar was the best method to present the new features to the user. It was determined to be sufficient to have the new features available in the menus since they will be only used on occasion for educational and debugging purposes. Anything relating to preferences conveniently went in the "preferences" menu. The changes to the user interface were the simplest feature to develop up to the project's requirements. As long as the new additions functioned as desired, the entire feature could be deemed complete and sufficient per the requirements. The Import and Export button functionality has been added under the sketch menu button. Additionally, shortcuts have also been implemented so that users have an extra method to import or export without having to explore the menus every time. A preferences checkbox has also been included so that users have, for example, the option to automatically export sketches to assembly for each verify action.

Verification Against Prior Phases

When progressing through the design process, the team constantly compared the current progress to previous stages of design to ensure that we were working towards satisfying a requirement. Beginning with potential project solutions, we carefully analyzed what needed to be accomplished in direct comparison with the project's requirements. Once we found possible implementation solutions to every requirement, the team progressed into the implementation phase. The prototype solutions that were created followed the guidelines set out by our selected solution. Once a prototype solution was believed to satisfy all of the given requirements of the associated feature, the prototype was integrated into the Arduino IDE. After all of the prototypes were implemented as features, tests were performed to make sure that each feature functions the same as the prototype solution. By ensuring that the implementations produce the same result as the associated prototypes, the team could confidently say that the feature satisfies the original requirement on which the feature was based.

Evaluation of Satisfied Requirements

To confirm that each and every project requirement was met, the team simply tested the newly implemented features and compared the results to their related requirement. The majority of this project's requirements are easily evaluated for completeness due the nature of the features themselves. At any phase of development, it was easy to check whether or not a feature satisfied a requirement since the feature would either work or fail. The only feature that did not have such simple success vectors was the assembly template, and this is because the functionality of the template itself is reliant on the user editing it. Due to this, we had to find another method to test the functionality for the assembly template that involved writing assembly by hand. The feature was considered a success if we could write assembly starting from the template, link it into an executable, and then successfully execute it on the Arduino Due board.

Project Results

The results of the tests were considered as pass or fail due to the interdependence of features and the clear distinction between whether the feature did what it was meant to or not. The exporting feature passed in that it successfully compiled and created an assembly file that contained only the user's sketch code in assembly. This means that extraneous operating system related code is mostly filtered out and not included in the assembly file that is produced by exporting. This test also revealed that the filtering of exported assembly is happening as intended; there is a ninety-percent reduction in code length from before any filtering was implemented. The importing feature allows the users to compile and link assembly files regardless of source. It is able to take an assembly file and do all of the standard compilation steps to produce an executable that can run on the Arduino microcontroller. Additionally, the executable generated from the importing feature can be uploaded to the Arduino Due board from the IDE, which can then be executed. The results of testing with the GUI have shown that the functionality added to the Arduino IDE work properly using the new GUI components. Preferences can be changed, saved, and reloaded with the accurate state upon the restart of the IDE; also the "Compile to assembly" and "Export to assembly" buttons correctly call the intended functions with no issues. Each of the new features can be used together or individually to cumulatively satisfy the project's requirements.

Lessons Learned

Team Issues

The team encountered several issues throughout the project's timeline, the first of which was the issue of needing to sort through hundreds of files quickly. A strong source-code editor made a huge impact on being able to sort and search through all of these files; the amount of time saved by using Sublime Text (for example) over VIM and shell commands is immeasurable. Another big issue the team encountered was that of how the Arduino IDE implements compiling sketches. The compiler that the IDE uses to compile sketches in a one-step operation, Arduino Builder, is external to the IDE source resulting in all of the requirements to involve "compiler hacks" essentially. The use of an external compiler means that any compilation flags are not easily obtained in the IDE for uses other than what the IDE was originally designed for. Due to this, the implementation of the required features was limited in terms of efficiency and robustness from the start. Other issues involved project requirements, specifically debugging, that turned out to be not viable. While debugging support for the Arduino IDE and the Due board can be very beneficial, the cost to enable that functionality is much too high for the semester budget in all cases.

Another issue encountered during integrating the import feature came in the form of a tiny button on the Arduino Due board named "ERASE". To be able to upload a compiled assembly file, which is a necessary stage in the import process, the user initially needed to press the erase button on the microcontroller before any upload to the board could begin. While this issue was solved, the process still involved reverse engineering the IDE source code even further as well as analyzing the call stack to find where and how the IDE made calls to perform an erase operation.

A small but technical issue the team had was with Git. Apparently, no one in our team understands the black magic behind what makes Git work. Accidentally deleting large files was occasionally a problem, however, these issues were fixed rather quickly. Managing parallel development on a remote source repository also presented other issues; a lack of synchronization and communication meant that sometimes one person's work would be overwritten by someone else's work. The solution to this issue would be more compartmentalization of work and directly imposing this onto sections of code being worked on. Perhaps the class itself could devote one or more lecture in the future specifically to teach students how to use version control software like Git and how to problem solve common issues such as the problems experienced by this team.

When focusing on team issues, it quickly became apparent that starting project work as early as possible is the best option for any large project. While the team did spread work on the project out across the semester, the later portion of the semester saw a significant decrease in available time for the project. The amount of project reports increased in both length and frequency, not to mention the increased workloads from other classes. In some cases, the increased workload actually required the project to be stopped temporarily so that certain assignments could be written. However, these kinds of issues cannot be predicted so the team could not have done any better with handling these time constraints.

One of the biggest issues our team had was that of work distribution. Due to the way that the project requirements were distributed, some team members had considerably more work than others. This was mostly due to incorrect assumptions when trying to predict the difficulty of implementing the various project requirements. A simple solution to this would have been more communication between team members as well as better time management.

An additional issue that the team encountered was the loss of team members. During the summer break the team lost Austin McEver; though, this was expected since he warned the team well in advance. During the first week of the fall semester, we lost another team member, Ajmeria Dushyant. This was completely unexpected to the team, as we were given no warning about this. Fortunately, the size of this project was still feasible for a group of four to complete.

Future Changes

After working with the Arduino IDE for a semester, the team would like to see certain things change with regard to the implementation of the IDE. For one, the Arduino IDE does not properly highlight syntax for assembly because it does not natively support assembly. Syntax highlighting greatly increases the readability of code and simplifies debugging. The requirements originally listed, as stretch goals, debugging abilities that would have been useful when directly working with assembly. The debug function would also provide benefits similar to common debug utilities, such as GDB. The function would allow real time access to the Arduino microcontroller's registers to view and/or change the contents. Another change that would achieve similar results would be the integration of a hardware debug adapter, which could allow the use of built-in debug and trace features of the Arduino Due's ARM Cortex-M3 processor. However, if debugging support is going to be implemented into the IDE, an entire future project should be dedicated to implementing such support considering that the size of the project is very large in terms of both research that will be involved and difficulty.

The IDE could also benefit from better assembly filtering. Being able to eliminate extraneous assembly that is generated from the user code even further would provide a better experience for students. Moving the filtering functionality to its own independent feature would greatly increase its usefulness as well. A nice improvement to the assembly template implementation would be to use a method that would split the assembly template into smaller pieces. One file "chunk" would simply consist of the "loop" function in assembly, which could then be edited by the user with their own assembly instructions. The file chunk edited by the user would then be stitched back together with the other assembly file chunks to form a valid assembly file. This would further reduce the time involved with writing assembly by reducing the amount of time that a user needs to spend searching for the places in the assembly template where assembly code can be inserted.

Team Contributions

Alan Person is a Computer Science major and is the team leader. His primary contributions directly relate to team management and project reports. He regularly met with Dr. Thomason, the customer, to update him with the current progress of the team. When not focusing on team managerial duties, he developed the functionality behind the importing/link feature and the creation of the assembly template. He also tested the new features of the IDE on Mac OS X since that was his primary platform. Alan also heavily contributed to writing and editing all of the papers regarding the project.

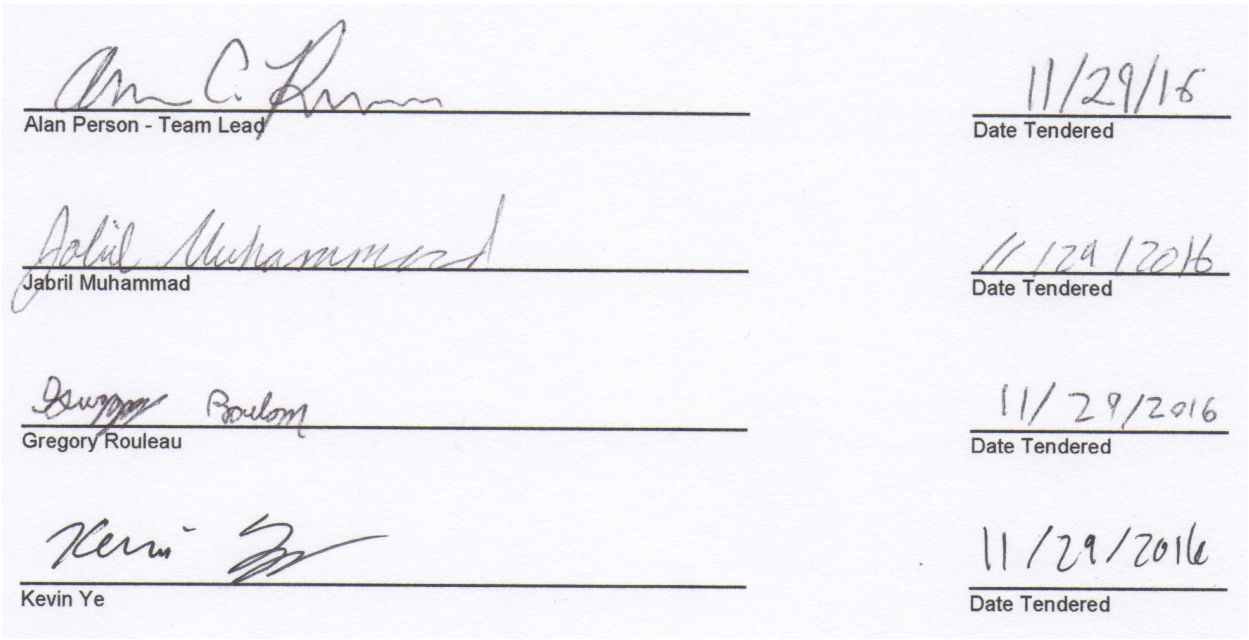
Jabril Muhammad is a Computer Science major and is the lead GUI developer in charge of making sure that the new features work with the existing GUI. He was also in charge of adding new GUI components to complement the new features, such as shortcuts and additional preference options. Part of his work involves testing the new features in the IDE to make sure other features were not broken in the process. Jabril tested the added IDE features on Ubuntu since that is his primary platform of choice. He also contributed to researching and writing involved in project reports.

Gregory Rouleau is a Computer Science major and is the lead programmer for Project Arduino. He conducted research into the compiler tool chain, which led to the development of the exporting processes. He then wrote the scripts to prototype the exporting functionality. He also integrated the import and upload feature into the IDE. Gregory also edited papers to ensure accuracy and proper grammar. He also assisted Alan with managing the team's time.

Kevin Ye is the lead writer, tester and a Computer Science major. He conducted research on debug adapters that led the team to conclude that that feature was not financially feasible to implement. The majority of his work was focused on assisting the team with the writing of project reports. After the main features of the project had been integrated into the IDE, Kevin was responsible for much of the testing as he had several computers running several versions of Linux to test the features on. With exception for Mac OS X, he oversaw all of the final testing among the various platforms, including Windows.

Team Agreement

I, as a contributor of Project Arduino, certify that the contents presented in this document are complete and accurate.



Alan Person - Team Lead — Date Tendered: 11/29/16

Jabril Muhammad — Date Tendered: 11/29/2016

Gregory Rouleau — Date Tendered: 11/29/2016

Kevin Ye — Date Tendered: 11/29/2016

Customer Agreement

As the customer, I acknowledge and agree that this document is consistent with the agreed-upon project requirements.

(Email confirmation plus extra stuff that may be ignored)

mthomaso Today at 5:44 PM

To: Person, Alan Christopher

Re: CS 402 Project - Final Paper

M

alan
no suggested changes, document approved. how about W at 11am in my office?

From: Person, Alan Christopher
Sent: Monday, November 28, 2016 1:37:12 PM
To: mthomaso
Subject: CS 402 Project - Final Paper

Sorry for the delay, I was having serious issues with formatting due to converting issues for a couple hours today. Here's the final paper, it is quite lengthy but most of it should seem familiar. Like the previous papers, please read through it and note any suggestions you have on it. Once you are finished, please send me an email to approve the document.

Also, you may know that poster presentations are this Tuesday and Thursday. Our project will be ready and open for demoing then, but I know you would like a demo during your office hours or otherwise during a time when you are free. So when would you like to meet for the project demo? Let me know.

Thanks,

Alan Person
Team 6