4-18-2023

Computer Organization and Assembly Language Embedded Firmware Overhaul

Elaine Yun Ru Chan
Western Michigan University

Table of Contents

Terms

Computer - An electronic device used to store and process data according to instructions provided by a program.

Program - A set of ordered operations for a computer to perform.

Software - A combination of programs and instructions to a computer that is designed to produce a specific output. Additional components like documentation are often considered to be part of the software.

Hardware - Any physical component of a computer.

Memory - Computer hardware used to store data that must be accessed quickly. This data may include program data, recently collected input, and other data transferred from more long-term storage devices.

Firmware - Software that controls and monitors low-level functions of the computer and often directly interacts with the computer's hardware. This will often refer to pre-programmed software on either the **MSP430** or the prototyped **GoodFET** board.

Operating System (OS) - Software that provides the basic functions of a computer.

Integrated Circuit (IC) - A set of electronic circuits on a piece or "chip" of semiconductor material. ICs are used in virtually all areas of computation, including microcontrollers.

I/O Port - I/O refers to input/output. An I/O port is simply any port on a computer that deals with the input and/or output of data. An I/O port is said to be programmable if the transfer of data through the port is initiated and controlled by a program.

Microcontroller - A small computer embedded in an IC which contains a small amount of memory, one or more processors, and programmable I/O Ports.

Texas Instruments (TI) - An American tech company that produces semiconductors and integrated circuits.

MSP430 - The MSP430G2553 integrated circuit was produced by TI. Contains a 16-bit processor and 16KB of flash memory.

GoodFET - An open-source **JTAG** adapter that is loosely based on the TI **MSP430** architecture. All hardware designs and software related to implementing a GoodFET microcontroller are freely available under their **BSD license**.

.

Ubuntu - A Linux distribution commonly used by programmers as an OS on computers and VMs**. CS2230** is taught assuming that students have access to a computer or VM running Ubuntu.

Virtual Machine (VM) - Software that allows the user to run programs and apps through the virtual machine rather than a physical computer. Each virtual machine runs its own OS and functions separately from its host computer.

CS2230 - Computer Organization and Assembly Language (CS2230) is a class offered by WMU and is part of the required CS curriculum. The class involves an introduction to the C programming language as well as an opportunity to learn the nature of low-level languages and computer processes, primarily through assignments. These assignments involve programming in C, programming in assembly language, and most importantly in our case, programming a microcontroller.

Toolchain - A type of software that provides a set of “tools” (small programs) that can be used for various purposes such as software development and debugging

JTAG - A collection of industry standards for printed circuit boards which include standards for programmable I/O ports as well as a dedicated debugging port.

BSD License - A license for computer-free software which imposes minimal restrictions on use and distribution.

Segmentation Fault - The program trying to access memory it's not supposed to.

Code Composer Studio (CCS) - An integrated development environment to develop applications for Texas Instruments embedded processors.

Mspdebug - A free debugger for use with MSP430 MCUs.

Limsp430.so - A shared library that communicates to the msp430.

Debugging - The process of identifying and removing errors from computer hardware or software.

Abstract

This software plan has changed mid-way through production, but the goal of the project has stayed the same: to fix and improve the software used by the CS2230 class. The client should be able to make changes to the firmware and software created, while users should only be able to install the toolchain, and have access to its debugger. However, because of a delay in chips needed for GoodFET, the plan changed to solving the segmentation fault through debugging. Along with fixing the error, firmware drivers must be merged to improve access in the future.

Introduction

Debugging is something that many new programmers can find intimidating even more so when having to debug on a microcontroller. The client understands this issue and has been using the software *mspdebug* to help improve the debugging experience. However, a memory error occurring on the latest version of Ubuntu has prevented the *mspdebug* folder and toolchain from being installed. This report will discuss what was done to achieve a working debugger and toolchain on Ubuntu. It's important to note that there was a change in plan halfway through the project because of a delay in hardware needed. Both the original plan milestones and the changed plan milestones will be discussed below.

There were three milestones for the original plan. The first was to modify the software to work on new hardware. This milestone talks about the fact that with using the GoodFET42 driver a different hardware is needed to run this software. This new hardware compared to the old holds less memory space, but it is cheaper so software needs to be modified to lower the amount of memory space used. Another milestone is to run the toolchain on Ubuntu version 22.04.1LTS the client has been having an error that is preventing the installation of its toolchain. High priority has been given for this problem to be resolved. The last milestone given is to prototype the GoodFet hardware since this is a new software/hardware to the client it must be tested so that no issues are found later on when the system is used in the field.

The alternative plan has two milestones instead of three. The first still is to modify software firmware that still needs to be merged. The other is to run the toolchain on the newest version of Ubuntu, but instead of using a new driver to fix our error we now must fix it by debugging. Both plans have the same idea behind them which is the reuse of software. This is called integration and configuration.

As stated above this project follows integration and configuration. This process is about the reuse of software components. The development plan chosen is the agile development plan which is explained in our project management plan section of this document**.**

As with all projects, standards and constraints affect the project. The software should be able to run on any computer as long as a Linux virtual machine is installed and has a USB slot to plug in the microcontroller.

The most costly area is acquiring the needed hardware which is in low supply but high demand. Other areas of the system are much less costly. On the maintenance side, it will be cheap and easy for developers to access. As requested by the client part of the team requirement is to organize and reduce the amount of files currently used for drivers and other systems.

Problem Statement

The Computer Science department at Western Michigan University offers a class teaching students the basics of computer assembly and organization. This class, CS2230, requires a significant amount of unique equipment and software. The primary piece of equipment required is the MSP430, a microcontroller that the students use to run their assembly code. The course also requires students to utilize the Ubuntu operating system to write and upload their code to these microcontroller boards. The course also requires the use of a piece of software called mspdebug to properly upload and debug code. Unfortunately, the newest version of the operating system, Ubuntu 22.04.01, contains several significant incompatibilities and issues that prevent proper functioning of mspdebug. Our project revolved around finding these issues and fixing them to allow mspdebug to properly run on Ubuntu 22.04.01. Furthermore, a large amount of critical firmware was scattered throughout multiple independent sources and repositories. Along with fixing the bugs regarding the newer version of the Ubuntu operating system, we plan to also merge these individual pieces of firmware into a single package.

Objective(s)

The objective of our project is to fix the error occurring and to merge all firmware into a single package to make future changes to the software easier to handle.

Problem Analysis and Research

Solving the segmentation fault

The main goal of this project is to allow the toolchain to run on Ubuntu 22.04.01 LTS. To accomplish this it was first decided to avoid the error entirely by replacing the driver with a new one called GoodFett42. GoodFett42 is open source, and allows the client to have more control over the program, because of removing the reliance on the TI library.

After some time it was discovered that the chips needed for GoodFett42 were unavailable. The only other alternative was to buy break-out boards and desolder to get the chips that way. This would take a long time to do so the plan was changed to keep the TI driver. The segmentation fault would still be resolved but requires the team to debug the TI driver and find a way to resolve the issue. There are some advantages of keeping the TI driver as the

testing section is greatly reduced because this is the driver that is currently working with the toolchain.

Libmsp430.so

Initially the client and the team had suspected that the error was coming from the driver or *mspdebug*. To confirm this a backtrace was needed to examine and find files where the issues are occurring. By default the files are compiled without debugging symbols turned on. This needs to be changed so that the backtrace can give the most useful data possible. Some makefiles were easier to change than others. To compile *mspdebug* symbols the -g simply needed to be added. While the other makefile needed to change an environment variable to override some if-else statements in the file.

After researching it was discovered that the error was not occurring from *mspdebug* or the driver but from the shared library libmsp430.so. This is a library created by Texas Instruments used by the Linux system to access the device. Instead of the error being caused by *mspdebug* it is instead caused by an outdated library that is trying to select a port that does not exist in the newest version of Ubuntu. While it is very hard for the team to make changes to such a library it was concluded that by using Code Composer Studio a new working version of the library could be found and used for the toolchain.

Requirements

NonFunctional

Environment

This software is going to be used in a university. Our client is also part of the university so for credit the university needs to be cited. Relating to credit only open source code can be used for the software to avoid copyright problems. This software will be used to help teach students of the university so all current CS2230 assignments must be supported.

Clean and clear

Software is always changing; this is something the client understands and wants to address. The software has many files in separate repos which has caused lots of difficulty for them. This needs to be fixed so future development is easier to perform. Project progress must be accurately tracked, and the repo that will hold the software needs to be clean, and clear to future developers. As part of this repo is the firmware which needs to be in a single package.

Project research

Due to the nature of this project, some of the specific steps we must take (such as the specific sections of code that must be removed) are currently unknown. The client would like the team to undertake this research ourselves, finding the optimal solution regarding the required changes to the firmware.

Functional

Required Software Version

The current firmware implementation has an issue running on Ubuntu 22.041LTS. The client currently believes this is caused by a bug in the “tilib” library. Transitioning to the GoodFet system should eliminate the need for tilib, and also fix the bug. If our client’s expectations of this are incorrect, however, the senior design team will be required to determine the cause of the bug preventing proper usage of the software on Ubuntu 22.04.1LTS and fix it.

MSP430G2553 Flashing

The MSP430G2553 is the current microcontroller that is being used to teach CS 2230. One critical requirement of this project is to ensure that this model of the MSP430 can be flashed using the GoodFet firmware. The client has mentioned the capability of using a software-based flashing technique that would prevent the need to have a hardware-based flashing chip, therefore reducing the cost of assembling the new board, however, this is a stretch goal, and while worth keeping in mind, is not critical for this project.

Merge Critical Drivers

The client’s current implementation of this toolchain and firmware currently uses several pieces of firmware, primarily drivers to control a potentiometer, the onboard EEPROM chips, and libemp. These must all be merged into a single software package that can then be compiled and loaded onto the MSP430 in the firmware package.

Binary size shall be reduced via Makefile modifications

The compiled binary will initially be reduced by simply modifying the Makefile. By changing the Makefile, we will be able to easily cut out large chunks of code that are not required for our final compiled binary.

Critical sections of code shall be rewritten

Some sections of critical code may not be able to be removed by excluding it from the Makefile but will still need to be decreased in size. To do this, the senior design team may be required to rewrite sections of code by hand, removing the parts that are not required, and keeping those that are.

Standards and Constraints

Applicable Standards

This project must run on a Linux-based environment with an installed MSP430 toolchain on the newest version of Ubuntu. Along with the programming languages Python, and C should also be installed on the machine. To connect to the microcontroller a USB cable and slot are required. This can be run on any machine as long as the above is true.

When future modifications occur the individual machine has the option to update or keep the current state of the software. Although not updated software may cause errors for the user. Any future changes must be made with the same hardware in place, and follow the same guidelines in place to prevent an increase in memory, which can affect the controller. All modifications must be approved by the client if the developer wishes to push to the remote repo for other users to download, and use.

Constraints

Cost

Economically the project is funded by the computer science club which is part of the university. The project will have a high hardware cost, but on the software side, it is free, because of the use of open-source software. The cost of maintenance is minimal as requested by the client all files need to be organized and placed into a GitHub repo this allows easier access to future developers. This will also have the effect of making maintenance easier and will allow other developers to understand the software and its components.

Scope

This project will be used by Western Michigan University so the sources used must be from only open sources, and progress must be reported. The operating environment this project will be in is a classroom environment so it must be able to handle what students throw at it, and not crash.

Time

The time constraint for this project spanned two semesters: the first semester in the fall of 2022 was used to establish requirements and plan for development. The second semester is used for the implementation and testing of the system.

Ethical Analysis

Identify the Moral Issue

This software is used in a classroom setting, all code created is uploaded to the website GitHub, so that student code can be graded. The moral issue that can occur is code leakage of users' code. Code leakage can lead to the stealing of code and plagiarism. There is also another issue which is reliance on third-party dependencies. The moral issue that can occur is the risk of malicious software being downloaded onto a user's computer unknowingly.

Identify Additional Facts Helpful to Making the Decision

1. Who has access to this information?
2. Are users aware of the different risks?
3. Is there security to prevent leakage and malicious code?

Identify the Alternatives Available

Some alternatives include moving away from third-party dependencies by creating libraries that only the client has excess to. Doing so greatly reduces the risk of unknown dangerous code from being downloaded and run on users' machines. For the issue of code leakage, an alternative is more challenging. The code must be accused by more than the creator so that it can be examined. GitHub which is where the code is uploaded has SSH keys that are needed to access the code inside a repo. Instead of using GitHub, the client could use a different grading approach for the code.

Identify the Personal Impacts on the Decision-Maker

Western Michigan University is a trusted school. Plagiarism is taken seriously at the school if leakage occurs it can hurt the reputation of the school. Also with the other issue if malicious code is downloaded and runs this can also hurt the school's reputation.

Ethics Assessment According to Contemporary Theories of Business Ethics

Shareholder Theory

Keeping Shareholder Theory in mind, our ethical analysis should focus on maximizing the financial returns for shareholders of the university. In our case, we should be focused on reducing the risk of code leakage and malicious code installation to prevent damage to the reputation of the university. Such damage to the reputation of WMU could result in considerable reductions to the financial returns seen by university shareholders. That being said, the impact that a fault in one system used in one course could have on university shareholders as a whole is assumed to be small. While faults in the project may have slightly negative impacts on university shareholders, the success of the project has little to no impact on the financial returns of shareholders. Because of this, the project may be viewed by shareholders as having some risk and no reward, making the reduction of said risks especially important to shareholders.

Stakeholder Theory

According to Stakeholder theory, our ethical analysis should consider the impact that the project may have on all stakeholders, not just the shareholders. In our case, the stakeholders include the students, faculty, and broader community of the university. We should aim to reduce the risk of code leakage and malicious code installation in order to mitigate the harm to stakeholders that such faults could cause. Similarly to Shareholder Theory, damage to the reputation of the university could negatively impact the interests of our stakeholders. Moreover, certain stakeholders may see more impact than others given our project’s close association with the Computer Science department at Western. While we feel confident claiming that the potential impacts on university shareholders and stakeholders are small, the same may not be true for stakeholders with explicit interest in the CS department. This includes CS faculty, CS students, and the community surrounding the CS department. Damage to the reputation of the CS department may considerably impact CS department stakeholders, even if the associated impact on the totality of university stakeholders is minor. Even so, CS stakeholders are also

more interested in the university's ability to offer CS2230 and use the msp430 microcontrollers than general university stakeholders.

a. Utilitarianism

Utilitarianism is a consequentialist ethical theory that holds that the morality of an action is determined by its consequences. The use of the msp430 and its current software package allows CS2230 students to better engage with the material of the course. Some potential negative consequences are that a user's code is less private or that malicious code is unknowingly downloaded onto a user's machine. From a utilitarian perspective, the use of the current msp430 package may not be morally justified if the potential for these negative consequences outweighs the consistent benefit that the msp430 provides to students. This does not seem to be the case since the likelihood of such negative consequences occurring is low, and the benefit provided to students is high.

b. Rights

The use of the msp430 and its software package raises questions about the students' right to code privacy and computer security. Determining whether using the msp430 constitutes an infringement of these rights is paramount under a rights-based ethical framework. While the potential for code leakage and malicious code installation are serious, the requests made of students in CS2230 (namely, to download the msp430 toolchain on a virtual machine, and upload your code to Github classroom) are not abnormal when compared to the requests of students in many other CS courses at the university. Because of this, it is unreasonable to assert that students have an established right to not upload their code to the internet, or to not download software from third parties. Such rights would call the morality of multiple courses into question.

c. Justice

In the case of business ethics, justice is concerned with fairness and impartiality. The use of the msp430 and its current software package may be seen as unjust if it unfairly impacts certain students over others. In CS2230, all students are provided access to the msp430, a virtual machine via VMware, and the toolchain. Because of this, all students have the ability to interact with the msp430 in the most secure and private way possible. Students who choose to not use a virtual machine may be at a higher risk than other students, but this is not an injustice because students are given every opportunity to use the course materials as is recommended. Students who use the msp430 as instructed will have a near identical experience to each other, leaving little room for injustices to occur.

Virtue Theory

Virtue Theory states that actions that display virtuous behavior and traits are ethical. Based on an analysis of this project using Virtue Theory, we can state that this project is ethical. Several parts of this project display virtuous traits. Examples of this are the increase in safety and security that comes with allowing students to use the new version of the Ubuntu operating

system. One of the most dangerous habits with software is to not update to newer safer versions, by supporting Ubuntu 22.04.01 we embody the virtuous trait of safety and stability. Furthermore, this project helps uphold the virtuous trait of frugality. This work sets the stage for allowing a further group to continue forward by producing a new board that is cheaper and easier to produce than the current combination of the blacktop board and the MSP430 Launchpad. This reduces waste in both a technical sense of saving physical resources and components, but also in a financial and a time sense of saving finances during board procurement.

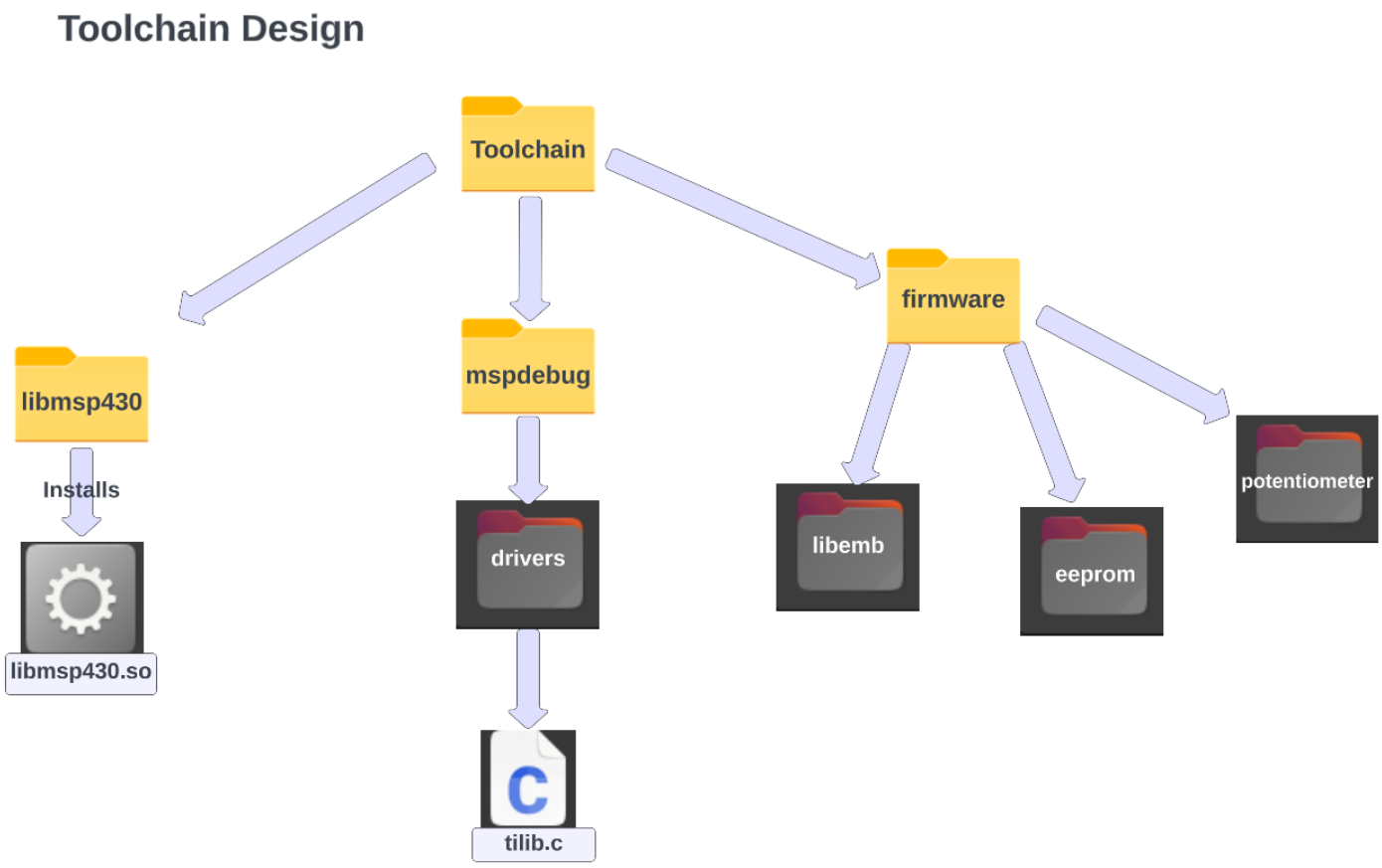
Conclusion

Based on the above ethical analysis, we believe that our project fits the required ethical requirements. This project satisfies all of the primary methods of ethical analysis. Whether that be providing value to the shareholders by preventing damage to the reputation of the university, or increasing the value to all stakeholders by ensuring students receive the best education that is available to them, our project improves the situation of all individuals involved. Furthermore, it satisfies many common virtues such as safety and frugality, by both increasing the security of the operating system students will be using, and reducing waste, both of finances, time, and resources. Based on all of these mentioned techniques, we can conclude that this project is ethical.

System Design

The software architecture consists of many folders and libraries. All are used to allow communication and flashing to the MSP430 microcontrollers. The main folder that holds most if not all is Toolchain which includes everything needed for the CS2230 class. Three folders were

focused on *libmsp430*, *mspdebug*, and *firmware*. The folders *libmsp430* and *mspdebug* already existed while the *firmware* was created by the team.



libmsp430 folder holds many files, but what's important is that this is where the shared library that has caused the segmentation fault is first called upon to be installed. Inside the makefile of this folder, we can find the commands used to install and place libmsp430.so into the locked *usr* folder.

mspdebug simply held the driver we are currently using and wish to use (GoodFET) this is also where some debugging occurred to trace the memory error. As discussed later on this is also where improvements can be made to reduce memory size in the future.

The modified firmware focused on several critical components. As seen in the above diagram, this firmware contains information for the potentiometer driver, the EEPROM driver, a seven-segment driver, and libemb. These components all support an individual part of the system and are required for the proper functionality of the device.

The first piece of custom firmware is the potentiometer driver. This code was written in the past by another student working on a similar project. It had to be found in several archived repositories so a significant amount of the work was finding the proper version for this code. In the end, this software is used to receive readings from the onboard potentiometer on the extended blacktop board.

Another important piece of firmware is the EEPROM drivers. Unlike the potentiometer driver, this was not created by a student but was instead externally provided. Nevertheless, this

firmware is extremely important as it supports reading and writing from the onboard EEPROM memory. Having this externally provided is a significant benefit to our team, as the complexity of this type of work adds a significant amount of development time to the total project.

Additionally, Libemb is a paramount component of the entire system. This piece of software handles large amounts of general embedded systems work. Working with an embedded system like the MSP430 comes with a large amount of unique code that is unlike other more traditional computing systems. Libemb does a large amount of the heavy lifting for this sort of thing and reduces our workload significantly. In the future, more modifications to Libemb can be taken to reduce the total binary size by removing unnecessary components, such as those intended to run on other microcontrollers.

Finally, we have the Seven-Segment driver. While this was initially on our roadmap, the team eventually pivoted away from it, as our client has expressed interest in moving away from the seven-segment displays to a system designed around a micro LCD display. This is primarily to save costs during building the boards as the seven segments take up a large portion of the total cost. In the future, a driver for these new LCD displays may have to be written.

Testing

It's important to note that both the driver and the libraries being used are the same that the client has been using. So these libraries and drivers are known to work with the programs used by the CS2230 class. Following the project's change in plans which moved from implementing to debugging, the plan for testing greatly changed. Instead of the original plan with the switching of drivers and having to check that each assignment given by the instructor works as intended this was no longer needed. The new plan involved the old driver that is known to work on all the assignments. With all the debugging done lots of testing was done to test if the code was working correctly. However, there is a difference between this type of testing and testing for the project as a whole. Recall that testing is a process that confirms a program or system meets all its specifications. Debugging is finding and fixing errors in a program or system.

The testing plan was a step-by-step process. As requested by the client, the requirement involving resolving the error that prevents flashing to the MSP430 was a top priority. Once this specification was met then the other requirements could be tested and implemented. The test case given by the client was the toolchain itself. For the first specification in which flashing to the MSP430 was required a program called *hworld.c* was used for testing. If the team could get the MSP430 to say hello world then the first specification is met.

The second specification which involves merging firmware into a single package the test case was the toolchain. If this updated pathing for the firmware is installed in the toolchain with no issue then our specification is met.

The client was updated about the project whenever progress was made. What this means is that whenever the progress was made the client was informed. For example, when a backtrace was obtained it was shown to the client and discussed about the next steps. Throughout this whole project communication between the client and the team was good. The client who is well versed in microcontrollers helped direct us in the best way to fix the segmentation fault, recommending we look at the compiling code of Code Composer Studio. With the new libmsp430.so library the test case of *hworld.c* worked as intended by the client.

Results

Realization of Requirements

The team was unable to fulfill all of the original requirements of the client, that is to completely replace the current driver of the msp430 toolchain with another using GoodFET. The initial requirements set were to prototype the GoodFET hardware, modify the software to be compatible with GoodFET, and ensure that the toolchain runs on Ubuntu 22.04.1LTS.

Finally, a new set of requirements have been determined through discussions with the client. This includes modifying the software by merging all firmware into a single package and running the toolchain on Ubuntu 22.04.1LTS.

Realization of Standards and Constraints

The standards and constraints of the project have taken various factors into consideration, not only on the technical aspects but also on the overall pipeline of the project, including financial and supply chain issues.

Specifically, through the delay of the GoodFET boards, specifically because of the current situation of a chip shortage, our progress had been significantly impacted. Alternative routes, on the other hand, seemed to impose a financial burden and the need for extra manpower as breakout boards had to be bought and manually desoldered to retrieve the chips.

However, our main project standard was to successfully run the toolchain on Ubuntu 22.04.1LTS, our goal was successfully achieved nonetheless.

Testing results

Through the development process, testing served as a vital way to verify that the changes done to the existing software didn't affect the toolchain. At the conclusion of development which included debugging and merging testing performed showed the application running as intended. Not much testing was needed for this project as much of the code is already known to work as needed for the client.

Future Work

With the delays that occurred that caused the change in plan there is future work that the client wishes to implement. Our client still wishes to replace the tilib drivers with GoodFett to remove the dependence on outside providers. There is also the requirement of the 7-Segment driver that still needs to be created. Besides the above that the client wishes to see next, this project has areas that can be improved upon.

Reducing *mspdebug*

mspdebug which is an open-source program has many folders and files that are not useful to the software or client. *mspdebug* has about eleven different drivers it can support two

being the tilib and GoodFET driver. Once the client has decided on the driver that they wish to use for a long time it may be useful to remove the irrelevant files that relate to other drivers. This has two pros, first, it cleans up this section of software making it easier for future developers to understand the code quicker. Second, it also reduces the memory size of the software making it quicker/easier to install.

Conclusion

The primary objective of our project was always to remedy the segmentation fault error occurring in the msp430. This was the primary concern of our client due to the microcontroller's importance in instructing CS2230, Computer Organization, and Assembly Language. Despite considerable changes in our initial plan, we were able to successfully resolve the error and provide the client with a newly merged software package to more easily install necessary drivers on the msp430 in the future.

Our initial plan involved using GoodFET to make large changes to the current hardware setup as well as completely replace the current TI toolchain. This plan aimed to produce a similar result to what we have achieved through debugging and merging but with the added benefit of moving away from TI-sourced hardware and software. Unfortunately, this plan had to be discarded after several delays and complications, the largest of which was caused by the global chip shortage. After realizing we did not have enough time or expertise to work with what was available to us through GoodFET, we started to explore a more direct solution to our primary objective.

Our revised plan involved fixing the segmentation fault through debugging and merging the existing software packages into one easy-to-download file. After having spent several months on the GoodFET idea, we wasted no time in identifying the location of the bug and obtaining the information we needed to fix it directly. This process involved communicating with external developers, enabling the required debugging symbols, obtaining backtraces, and using Code Composer to modify shared libraries that are otherwise difficult to change. After fixing the segmentation fault we moved on to merging the existing drivers used to operate the various components of the msp430 setup. Drivers to control the primary functions of the system, the onboard EEPROM chip, and the potentiometer were all merged into one file titled *libmsp430.h*. This file will serve as an organized baseline for future developments in the software of the msp430 setup.

While the primary objective of the project has been completed, there are still several areas where the msp430 setup can be improved. Future projects may involve replacing the current TI libraries using technology like GoodFET, reducing the size of the *mspdebug* program, writing a driver for the 7-segment display in *libmsp430.h*, and more.

References

1. Beer, Daniel. "Mspdebug: Debugging Tool for MSP430 Mcus." *GitHub*, https://github.com/dlbeer/mspdebug.

Appendices

Project Management Plan

Software processes are important for any software creation, this project follows integration and configuration. The process is about the reuse of software components originally GoodFett42 to build a new software system, but instead changed to debug the TI driver.

Specifically, we have chosen to utilize the agile software development methodology as it aligns best with our project use case. Starting with the need for better visibility into the project, we chose to use the agile development methodology as it provides improved project predictability, with the ability to identify and mitigate risks that may occur during our project timeline. This methodology greatly assisted in the work of the project. With all changes that occurred during the production. Being able to show the client pieces of progress over time helped reduce the fear the client may have held. Furthermore, we aim to prioritize the quality of our product, thus heavily focusing on the agile development methodology as it motivates us to continuously improve our product at each cycle with an emphasis on identifying the strengths and weaknesses of said product.

To give an overview, the team will meet on a weekly basis to identify the issues to prioritize in the current cycle. We will also determine the strengths and weaknesses of our previous iteration and how we can mitigate the downfalls accordingly. Furthermore, meeting minutes will be recorded per weekly meeting. Continuous communication will be adhered to by team members either via the group chat or the Microsoft Teams channel. Lastly, progress will be updated for our clientele on a regular basis.

The beginning phase of our project has been completed, that is the breakdown of the project into manageable chunks by identifying the major issues/requirements of the project. The next steps are to identify the priority of the issues and how it aligns with the needs of our client.

The development process was a step-at-a-time ordeal. With the delays in chips, the project plan had changed. Issues and milestones had to change to fit this new plan, and still follow what the client wanted. The development plan changed as well. Instead of simply accepting and finishing each issue like other projects a different type of strategy was created. There were still issues and milestones, but each step taken to resolve an issue had to be well documented. This project focused a lot on debugging. To avoid team members potentially doing the same thing to resolve the error notes were created that tracked what was done to fix the error. As the project advanced, the team loosely fell into roles, which aided in increasing efficiency. Some team members focused on resolving the error while the other half focused on merging the firmware into a single package for the client.

Progress Reports

Project Information: CS2230 Blacktop Design Project

Team Members: *Adam Pohl, Elaine Chan, Kelly Osborn, Samuoel Ryan*
Client: *Colin MacCreery/WMU Computer Club*
Advisor: *Allin Kahrl*

Report Date: *12/2/22*

Team Activity Report:

As this is our first report, what follows is a summary of what we have achieved with the project so far:

Functional and non-functional requirements were generated during a meeting with the client on 10/24/22.

This week, our requirements were expanded upon and converted into milestones. This was done after a meeting between Adam and the client on 11/28/22.

A final report document has been formatted and shared with all team members. Different sections have been assigned to be completed by different team members by the end of the semester.

A Github repository has been set up with proper privileges given to our client, advisor, team members, and current computer club leadership. Club leadership will be maintaining the repository after our 2-semester time frame ends.

Client Interaction Report:

Adam, Kelly, and Sam met with the client and advisor on 9/1/22. This was our first meeting where research materials, contact information, and schedules were exchanged, project possibilities were discussed, and risks were considered.

All team members met with the client on 10/24/22. Vague requirements for the project were discussed (necessary functionalities, organization under ccowmu github organization, software trimming/merging)

Adam met with the client on 11/28/22. During this meeting, the requirements generated on 10/24 were made more specific. This was done in an attempt to make them easier to convert into milestones for the group to start completing.

Milestone Review:

The project is still in the planning phase. Our requirements have only recently been converted into milestones. That being said, research and project organization have been major focuses in the last few weeks. Since we are planning to use pre-existing resources like GoodFET, time has been spent looking at the structure and content of the repositories we hope to take advantage of. Additionally, our own repositories have been set up and distributed to team members, and team members have been given distinct responsibilities pertaining to project progress reports, note taking, client interaction, final report organization/proofing and more. While little progress has been made on milestones, the group is on track to begin developing and prototyping in the second semester.

Issues (or stories):

As mentioned, no milestones have yet been targeted for completion, so no specific issues have been targeted either. However, our meeting with the client on 10/24 did provide us with scenarios that we will be able to convert into working stories. For example, a CS2230 student should be able to use ‘mspdebug’ on our circuit so long as they are using it with an Ubuntu virtual machine. This scenario involves several project requirements, therefore it will require the completion of several issues to make it feasible.

Problems and Risks:

Client interaction has been more difficult than expected. While working with a client associated with WMU provides us with frequent opportunities for communication, obtaining all the information required to get the project started has been difficult. This is exacerbated by the entire team (including the client) having large workloads at the end of the semester. We are gradually improving at getting required information from the client, and have recently contacted them to improve our existing requirements. We will continue to increase the quantity and quality of communication with the client.

As mentioned previously, the end-of-semester workload presents a significant risk to project progress. Significant steps have been taken to distribute the workload on project deliverables such as the final report in order to reduce this risk. We will continue to frequently communicate with each other in order to properly distribute the project workload.

Report Date: *12/9/22*

Team Activity Report:

The team is focused on generating the requirements document due 12/14. The team met virtually on 12/5 to discuss the work needed on the report as well as plan our activities over the winter break. Team members have also been reading through several github repositories from former projects related to CS2230 in order to find useful code and information.

Client Interaction Report:

A small meeting was had with the client this week. Other than going over the basics of the project, some minor changes were specified. Particularly, the client clarified that getting the toolchain working on Ubuntu 22.04.01 was a high priority item, as opposed to a low priority one as was initially assumed. Changing out the tillib driver with the goodfet driver should be enough to achieve this. Furthermore, the client specified that we will be required to confirm all of his assignments are working using the new firmware and drivers. This provides some unique challenges, as the client is hesitant to distribute the solutions to his assignments, meaning we will likely need to acquire a working copy of the finished assignment through some other means, perhaps through writing it ourselves or receiving an already compiled binary. Finally, the client continued to share more repositories related to the history of the project, which may contain relevant code and information.

Milestone Review:

Our current milestone is to read through the large amount of Github repositories and GoodFET source files in order to trim out unnecessary code while identifying the code that we can apply to the blacktop project. Work on this milestone will continue into the winter break and we aim to have a good understanding of the components we hope to use before the spring semester starts.

Issues (or stories):

The milestone mentioned above can be separated into several issues. One of which is to read through and comprehend the relevant GoodFET source files. Additionally, the Github repositories supplied by our client vary in size, complexity, relevance, and organization. Because of this, certain repositories may require an issue of their own, while others could potentially be grouped together or even discarded after initial review. Because of this, the issues within this milestone are not set in stone and will likely change as we continue to work on them.

Problems and Risks:

Client interaction has been more difficult than expected (still). While working with a client associated with WMU provides us with frequent opportunities for communication, obtaining all the information required to get the project started has been difficult. This is exacerbated by the entire team (including the client) having large workloads at the end of the semester. Additionally, when the client is asked for approval on project planning/progress, he tends to be very succinct and approves our plans without much note. It is good that he trusts us to achieve the goals of the project with much oversight, but sometimes additional oversight would be appreciated when we are a little unsure of our path. We are gradually improving at getting required information from the client, and have recently contacted them to improve our existing requirements. We will continue to increase the quantity and quality of communication with the client.
As mentioned previously, the end-of-semester workload presents a significant risk to project progress. Significant steps have been taken to distribute the workload on project deliverables such as the final report in order to reduce this risk. We will continue to frequently communicate with each other in order to properly distribute the project workload.

Report Date: 1/23/23

Team Activity Report:

The team has been working on several things since the start of the semester. Our updated SWOT analysis and mitigation document were completed this week along with the description of our project for the design conference. All team members met on 1/12/23 to get there hands on a MSP430 and start working with it together. This involved installing the toolchain used in C2230 and then attempting to use the new Goodfet software to interact with the MSP430 (unsuccessfully).

Client Interaction Report:

Kelly met with the client on 1/17/23 to discuss our project progress and options for obtaining the Goodfet boards. He informed us that he has a company in mind that is willing to do it and will provide them in a timely manner.

Milestone Review:

Initially, our first milestone of the semester was to experiment with the MSP430 boards and the Goodfet software. This was done at an in-person meeting on 1/12/23 where an MSP430 board was provided to all team members and our development environments were set up to interact with them properly. Once this was done, we attempted to use the proposed Goodfet software package to interact with the boards. After running into a wall and consulting our client, we have determined that using Goodfet software on the boards is not feasible without first getting the Goodfet hardware that attaches to the boards. Because of this, we have gone back to cleaning and organizing files as our current milestone while we wait for hardware.

Issues (or stories):

The first milestone we attempted to complete this semester involves several separate issues including getting our hands on an MSP430 board, setting up our environments to interact with them, using the existing toolchain to interact with them, installing the new Goodfet software, and then using the Goodfet toolchain to interact with the board and gain a practical understanding of how Goodfet operates on the boards. The final issue of this milestone was found to be impossible to complete with our current hardware, which prompted us to move to a different milestone.

As was mentioned in our last report, cleaning/organizing the files provided for our project is a large task that has been separated into several issues. One of which is to read through and comprehend the relevant GoodFET source files. Additionally, the Github repositories supplied by our client vary in size, complexity, relevance, and organization. Because of this, certain repositories require an issue of their own, while others could potentially be grouped together or even discarded after initial review. Because of this, the issues within this milestone are not set in stone and will likely change as we continue to work on them.

Problems and Risks:

The biggest problem we have run into this semester is having our project progress stalled because we are waiting on hardware. We have already contacted our client concerning this problem and our client and advisor are currently looking into solutions.

Client interaction is still a concern, but we are confident in our ability to communicate with the client regularly. Several team members have weekly interactions with him where short meetings can take place. He has also expressed interest in joining our future in-person meetings via Teams/Webex.

Report Date: 1/30/23

Team Activity Report:

The previous week's activity involved investigating several bugs and ordering the Goodfet hardware we require. Adam and Kelly met on 1/27/23 to work together on the project.

Client Interaction Report:

No formal client meetings took place last week outside of the regular communication that occurs within our group chat.

Milestone Review:

Current milestones involve obtaining Goodfet boards, investigating existing bugs in relation to the current MSP430 setup, and cleaning, merging, and /or organizing related files and repositories. Significant progress has been made on these milestones.

Issues (or stories):

Since the last progress report, our Goodfet boards have been ordered and shipped by a supplier. They should be available to us soon which will mark the completion of this milestone.

Cleaning/organizing/merging the files provided for our project is an ongoing milestone discussed in previous reports. Current issues within this milestone involve creating a make file that will compile several folders together (blacktop-master, libemb, and libmsp430) when make is called, and converting msp430.h and other "include <>" files into system headers.

Investigating current bugs relating to the msp430 is another milestone we have been working on while we wait for Goodfet hardware. An issue with the current toolchain in "script.sh" was fixed by changing "libboost1.71-all-dev" to "libboost1.74-all-dev". This allows the script file to correctly execute and install. Another issue which was investigated involved a segmentation fault when using the new SLAC460L+ API and calling MSP430_GetNumberOfUsblfs. After some debugging, it was found that the source of the error is in the tilib driver. Switching to the Goodfet software should fix this issue down the road if this is the case.

Problems and Risks:

The biggest problem we have run into this semester is having our project progress stalled because we are waiting on hardware. We have ordered the hardware and it has been shipped so all we can do now is wait.

Client interaction is still a concern, but we are confident in our ability to communicate with the client regularly. Several team members have weekly interactions with him where short meetings can take place. He has also expressed interest in joining our future in-person meetings via Teams/Webex, and has always been active in our chat channel within the CClub server.

Report Date: 2/6/23

Team Activity Report:

The team has continued to work on our current milestones as well as working on our respective parts of the team ethics report.

Client Interaction Report:

No formal client meetings took place last week outside of the regular communication that occurs within our group chat.

Milestone Review:

Current milestones involve obtaining Goodfet boards, investigating existing bugs in relation to the current MSP430 setup, and cleaning, merging, and /or organizing related files and repositories. The group has also been working on the group ethics report.

Issues (or stories):

Goodfet boards have been ordered and shipped by a supplier. They should be available to us soon.

Problems and Risks:

The biggest problem we have run into this semester is having our project progress stalled because we are waiting on hardware. We have ordered the hardware and it has been shipped so all we can do now is wait.

Client interaction is still a concern, but we are confident in our ability to communicate with the client regularly. Several team members have weekly interactions with him where short meetings can take place. He has also expressed interest in joining our future in-person meetings via Teams/Webex, and has always been active in our chat channel within the CClub server.

Report Date: 3/13/23

Client Interaction Report:

The client was last contacted by the team before the spring break. This was when the client suggested we put more effort into fixing bugs in the current MSP430 toolchain. It was also suggested that we contact Daniel Beer, the developer of mspdebug, in order to assist in our debugging efforts.

Team Activity Report:

The team has been continuing to work on fixing previously mentioned bugs while we wait for the Goodfet boards to be delivered. The boards continue to be delayed, so the current focus of the project has significantly shifted to accommodate the delays:

Original Plan

For an unknown reason the newest version of Ubuntu 22.04 has a segmentation fault with the tilib driver from MSPDebug. A software designed for debugging MSP430 MCUs while it is most known for debugging it also flashes our programs to the microcontroller. This error is preventing the client from keeping their assignments updated for new virtual machines introduced.

ERROR:

```
kellyozz@Kellyozz:~/Desktop/toolchain-main$ make flash
mspdebug tilib "prog hworld.elf"
MSPDebug version 0.25 - debugging tool for MSP430 MCUs
Copyright (C) 2009-2017 Daniel Beer <dlbeer@gmail.com>
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Chip info database from MSP430.dll v3.15.0.1 Copyright (C) 2013 TI, Inc.

Using new (SLAC460L+) API
MSP430_GetNumberOfUsbIfs
Segmentation fault (core dumped)
make: *** [Makefile:26: flash] Error 139
kellyozz@Kellyozz:~/Desktop/toolchain-main$
```

How do we know it's the tilib driver?

The command in our Makefile line 25 is mspdebug tilib "prog $(NAME).elf" changing tilib to a different driver removes the error from above. The client learning this wanted to switch drivers from tilib to instead GoodFett drivers. This driver is a open-source JTAG adapter loosely based upon the TI MSP430 FET UIF and EZ430 boards. The client wished for a team to experiment with this driver to see if it could run the assignments the client gives to students, and of course to remove the error described above.

Problem

Sadly the delay of the Good Fett has caused the stategry above to change.

With this problem the client has requested that we dive deeper into the segmentation fault to get a backtrace from gdb so that the client could contact Daniel Beer the developer of mspdebug to receive his input about the issue and potential solutions.

This was proving difficult because of some reason the debugging symbols were not appearing so the information given before was not useful. However after digging through many makefiles inside the toolchain folder it was discovered that in the makefile to compile mspdebug the -s flag needed to be removed allowing debugging symbols to appear. Allowing this backtrace to be outputted

```
0x00007ffff7b97533 in boost::system::detail::failed_impl(int, boost::system::error_category
const&) () from /lib/libmsp430.so
(gdb) bt
#0  0x00007ffff7b97533 in boost::system::detail::failed_impl(int, boost::system::error_category
const&) () from /lib/libmsp430.so
#1  0x00007ffff7bcd91e in
boost::asio::detail::descriptor_ops::get_last_error(boost::system::error_code&, bool) () from
/lib/libmsp430.so
#2  0x00007ffff7bce057 in
boost::asio::detail::reactive_serial_port_service::open(boost::asio::detail::reactive_descriptor_se
```

```
rvice::implementation_type&, std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, boost::system::error_code&) () from /lib/libmsp430.so
#3  0x00007ffff7bcb0ba in TI::DLL430::UsbCdcIoChannel::openPort() () from /lib/libmsp430.so
#4  0x00007ffff7bcb3bb in TI::DLL430::UsbCdcIoChannel::retrieveStatus() () from
/lib/libmsp430.so
#5  0x00007ffff7bcb4de in
TI::DLL430::UsbCdcIoChannel::UsbCdcIoChannel(TI::DLL430::PortInfo const&) () from
/lib/libmsp430.so
#6  0x00007ffff7bcb913 in TI::DLL430::UsbCdcIoChannel::createCdcPortList(unsigned short,
unsigned short, std::map<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >, TI::DLL430::PortInfo, std::less<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const, TI::DLL430::PortInfo> > >&) ()
   from /lib/libmsp430.so
#7  0x00007ffff7bcbd1f in
TI::DLL430::UsbCdcIoChannel::enumeratePorts(std::map<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> >, TI::DLL430::PortInfo,
std::less<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const, TI::DLL430::PortInfo> > >&, bool) () from /lib/libmsp430.so
#8  0x00007ffff7bbde47 in
TI::DLL430::IoChannelFactory::enumeratePorts(std::map<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> >, TI::DLL430::PortInfo,
std::less<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const, TI::DLL430::PortInfo> > >&, char const*, bool) () from
/lib/libmsp430.so
#9  0x00007ffff7bb6715 in TI::DLL430::FetHandleManagerImpl::createPortList(char const*,
bool, bool) () from /lib/libmsp430.so
#10 0x00007ffff7b9b688 in DLL430_OldApiV3::GetNumberOfUsbIfs(int*) () from
/lib/libmsp430.so
#11 0x00007ffff7b98a0d in MSP430_GetNumberOfUsbIfs () from /lib/libmsp430.so
#12 0x00005555555bed3a in new_GetNumberOfUsbIfs (number=0x7fffffffdcd8) at
drivers/tilib_api.c:355
#13 0x00005555555b3a9b in do_findUif (dev=0x5555556d9ea0) at drivers/tilib.c:555
#14 tilib_open (args=0x7fffffffde28) at drivers/tilib.c:617
--Type <RET> for more, q to quit, c to continue without paging--
#15 0x00005555555cc95d in setup_driver (args=args@entry=0x7fffffffde10) at ui/main.c:502
#16 0x00005555555ccf97 in main (argc=2, argv=0x7fffffffe0e8) at ui/main.c:566
```

This helped narrow the search a little bit more, pointing us to the libraries used in toolchain, specifically the file libmsp430.so. Unlike making debugging symbols output for mspdebug trying to get the libmsp430.so to output these symbols is trickier. The first obstacle

was finding exactly where the libmsp430.so file is created. Looking at the script.sh file we find that this file is created in the libmsp430 folder of the toolchain. Luckily past developers of this make file have stated at the top of the file the flags to be used for compiling so the flags -g and -O0 are added.

```
CXXFLAGS := -fPIC -std=c++0x -fvisibility=hidden -fvisibility-inlines-hidden \
        -I/usr/include/hidapi -g -O0
```

This however did not give us the debugging symbols. The reason this issue is occurring is because of the if else statements relating to debugging. Our environment was assuming FALSE for debugging leading to the compiler flags not running the flags needed. To resolve this issue the environment variable needed to be changed so with the command export DEBUG = true. Running gdb we were able to get a more detailed trace back of our error.
https://docs.google.com/document/d/1Qu2ph8h_bktb3TLUn86MwFP1AyNt22EjNTu0iSukvmY/edit?usp=sharing

```
(gdb) bt
#0  boost::system::detail::failed_impl (ev=0, cat=...) at
/usr/include/boost/system/error_code.hpp:410
#1  0x00007ffff7bcd91e in boost::system::error_code::assign (cat=..., val=0, this=0x7fffffffd238)
at /usr/include/boost/system/error_code.hpp:604
#2  boost::asio::detail::descriptor_ops::get_last_error (ec=..., is_error_condition=<optimized
out>)
        at /usr/include/boost/asio/detail/descriptor_ops.hpp:59
#3  0x00007ffff7bce057 in boost::asio::detail::descriptor_ops::open (ec=..., flags=2306,
path=<optimized out>)
        at /usr/include/boost/asio/detail/impl/descriptor_ops.ipp:37
#4  boost::asio::detail::reactive_serial_port_service::open (this=0x555555704f40, impl=...,
device="/dev/ttyACM0", ec=...)
        at /usr/include/boost/asio/detail/impl/reactive_serial_port_service.ipp:56
#5  0x00007ffff7bcb0ba in
boost::asio::basic_serial_port<boost::asio::execution::any_executor<boost::asio::execution::cont
ext_as_t<boost::asio::execution_context&>, boost::asio::execution::detail::blocking::never_t<0>,
boost::asio::execution::prefer_only<boost::asio::execution::detail::blocking::possibly_t<0> >,
boost::asio::execution::prefer_only<boost::asio::execution::detail::outstanding_work::tracked_t<
0> >,
boost::asio::execution::prefer_only<boost::asio::execution::detail::outstanding_work::untracked_
t<0> >, boost::asio::execution::prefer_only<boost::asio::execution::detail::relationship::fork_t<0>
>,
boost::asio::execution::prefer_only<boost::asio::execution::detail::relationship::continuation_t<0
> > > >::open (ec=..., device="/dev/ttyACM0",
        this=<optimized out>) at /usr/include/boost/asio/basic_serial_port.hpp:364
#6  TI::DLL430::UsbCdcIoChannel::openPort (this=this@entry=0x7fffffffd458) at
DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:382
#7  0x00007ffff7bcb3bb in TI::DLL430::UsbCdcIoChannel::retrieveStatus (this=0x7fffffffd458) at
DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:410
```

```
#8  TI::DLL430::UsbCdcIoChannel::retrieveStatus (this=0x7fffffffd458) at
DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:404
#9  0x00007ffff7bcb4de in TI::DLL430::UsbCdcIoChannel::UsbCdcIoChannel
(this=0x7fffffffd458, portInfo=...)
        at DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:99
#10 0x00007ffff7bcb913 in TI::DLL430::UsbCdcIoChannel::createCdcPortList
(vendorId=vendorId@entry=8263, productId=productId@entry=19,
        portList=std::map with 0 elements) at
DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:346
#11 0x00007ffff7bcbd1f in TI::DLL430::UsbCdcIoChannel::enumeratePorts (portList=std::map
with 0 elements, open=open@entry=false)
        at DLL430_v3/src/TI/DLL430/UsbCdcIoChannel.cpp:359
--Type <RET> for more, q to quit, c to continue without paging--
#12 0x00007ffff7bbde47 in TI::DLL430::IoChannelFactory::enumeratePorts (list=std::map with 0
elements, type=type@entry=0x7ffff7c1c829 "CDC",
        open=open@entry=false) at DLL430_v3/src/TI/DLL430/IoChannelFactory.cpp:53
#13 0x00007ffff7bb6715 in TI::DLL430::FetHandleManagerImpl::createPortList
(this=0x5555556f9710, type=0x7ffff7c1c829 "CDC", update=<optimized out>,
        open=<optimized out>) at DLL430_v3/src/TI/DLL430/FetHandleManagerImpl.cpp:68
#14 0x00007ffff7b9b688 in DLL430_OldApiV3::GetNumberOfUsbIfs (this=0x5555556f8f50,
Number=0x7fffffffdc94) at DLL430_v3/src/DLL430_OldApiV3.cpp:271
#15 0x00007ffff7b98a0d in MSP430_GetNumberOfUsbIfs (Number=0x7fffffffdcb4) at
DLL430_v3/src/DLL430_capi.cpp:81
#16 0x00005555555bed3a in new_GetNumberOfUsbIfs (number=0x7fffffffdcd8) at
drivers/tilib_api.c:355
#17 0x00005555555b3a9b in do_findUif (dev=0x5555556d9ea0) at drivers/tilib.c:555
#18 tilib_open (args=0x7fffffffde28) at drivers/tilib.c:617
#19 0x00005555555cc95d in setup_driver (args=args@entry=0x7fffffffde10) at ui/main.c:502
#20 0x00005555555ccf97 in main (argc=2, argv=0x7fffffffe0e8) at ui/main.c:566
```

Using the information discovered and reporting to the client it was suggested that we install the linux version of CCS which is Code Composer Studio an IDE for TI's microcontrollers to develop and debug embedded applications. CCS could hold a newer version of the libmsp430.so file that can be used to replace the older one. However a problem occurred where the newest Ubuntu could not download CCS because of missing dependencies. After researching a solution was found where the commands

Sudo apt install libc6-i386 libusb-0.1-4 libgconf-2-4 libncurses5 libpython2.7 libtinfo5

Needed to be run before the installation of CCS could begin. After testing CCS could flash to the board and after digging through CCS files and folders it was discovered that CCS still uses a libmsp430.so file to flash code. Since these files are in the admin part of the files we now just need to find the best way to move this file into the directory that the toolchain folder works from and see if this can correct the segmentation fault.

Milestone Review:

As mentioned briefly earlier, the milestone of obtaining Goodfet boards has been significantly delayed. In response, other milestones have been given additional attention to compensate.

Fixing bugs in the current MSP430 toolchain has become the primary milestone we are working on. This milestone involves several issues ranging from reading source files, general debugging efforts, communicating with external developers, and obtaining priority information from the client on which bugs are most important to fix.

Issues (or stories):

The specifics of most of our current issues were elaborated on in the Team Activity Report section.

Problems and Risks:

Delays in obtaining Goodfet boards have caused the project to slow down significantly. This has put us in the position where much of the work for our project must be done in the second half of the semester because of our slow start.

As the semester goes on, the team will become more and more burdened by their workload from other classes and their final projects/exams. This could impact the quality and/or amount of project progress.