Firmware Management on Smartphone Sensing Extender

Edwin Prasetio

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

2015

Committee:

Professor Joshua Smith, Chair

Professor Richard Anderson

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

Abstract

Firmware Management on Smartphone Sensing Extender

Edwin Prasetio

Chair of the Supervisory Committee:
Joshua R. Smith, PhD, Associate Professor
Department of Electrical Engineering
Department of Computer Science and Engineering

In this paper, I present my research on a firmware management system for a FoneAstra device [1] that was developed to extend the sensing capability of a smartphone. The work is focused on the development of an Android [2] applications ecosystem that replaces a conventional software tool for updating AVR microcontrollers' firmware [3, 4]. The ecosystem contains FROG (Foneastra pROGrammer), FROGHexProvider, and the FoneAstra bootloader. In addition, I implemented a mini operating system to allow FoneAstra users to run multiple sensor functions without having to flash in new firmware every time they switch from one function to another. This work will be used as a starting point for future work to configure the FROG ecosystem to support placing multiple-sensor firmware in specific memory locations on a single FoneAstra's flash memory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I extend my sincere gratitude to my advisors, Prof. Gaetano Boriello and Prof. Joshua Smith, for their mentorship, immense knowledge, and motivation throughout my master's study and related research. Their guidance has helped me over the course of the research and writing of this thesis.

In addition to my advisors, I would like to thank the other member of my thesis committee, Prof. Richard Anderson, for his insightful opinions and questions that helped me widen my research from various perspectives.

I would also like to thank my mentors, Dr. Rohit Chaudhri and Waylon Brunette, for their endless support during my research development in the University of Washington Computer Science Department's Change research group.

Last but not least, I would like to thank my parents and sisters for supporting me spiritually throughout the writing of this thesis and my life in general.

Chapter 1. INTRODUCTION

In the past couple of years, smartphones have been expanding their functionality to become sensing devices for obtaining some of the characteristics of real-world objects. Although some phones already have a large number of sensors, these sensors are often not enough to sense and collect data from the real world for some particular applications. Therefore, a mobile phone sensing extender device called FoneAstra [1] was developed to solve this problem.

The FoneAstra device is an AVR microcontroller [3]-based sensor hub board that allows a user to attach the desired additional sensors to the device and use the smartphone to harvest the sensor data wirelessly via a Bluetooth [5] connection. For the device to work with the sensor, a user must provide the FoneAstra board with the appropriate firmware for the attached sensors. This process includes programming a new firmware into FoneAstra's microcontroller flash memory and switching from one sensor firmware to another.

My research objective is to help both FoneAstra users and FoneAstra developers focus more on developing the code for processing their sensor data and worrying less about how to program or switch between multiple sensor firmwares. The project also simplifies the hardware requirement for programming FoneAstra and makes the programming process faster compared to other existing software tools for programming AVR microcontroller families.

My research focuses on the development of the FoneAstra Programmer Android [2] application, usually called FROG (Foneastra pROGrammer). The FROG application allows a user to program a new firmware to a FoneAstra device using an Android phone wirelessly over a Bluetooth connection. This application eliminates the need to have extra programmer hardware for updating the firmware on the device. With FROG, all the user needs to update FoneAstra's firmware is just an Android smartphone that has a Bluetooth module installed on it.

Additionally, I developed another Android application: FROGHexProvider. I developed this application to demonstrate how to store sensor firmware hex files [6] on an Android application and then use FROG to grab the file and update FoneAstra. This system is useful for showing FoneAstra developers how to store their sensor firmware with their sensor driver Android application and utilize FROG to handle the firmware update process.

Last but not least, I also wrote a lightweight operating system for FoneAstra that can handle the switching process from one sensor function to another. This work will be useful for future development of FoneAstra; my research group is planning to build a software tool that can download multiple sensor firmwares into several different memory locations on a single FoneAstra flash memory. The operating system will have a main function that handles the process of jumping from one sensor function to another depending on the input command from a FoneAstra user.

In this paper, I will first talk briefly about the FoneAstra device to give a better understanding about the board. Then, in chapter 4, I will explain the implementation of FROG and its supporting software applications: FROGHexProvider, FoneAstra bootloader, and FROG operating system. Lastly, I will discuss the performance of FROG and future work, and then ended this paper with a conclusion chapter.

Chapter 2. RELATED WORK

Before discussing my research project, I want to mention related work that has become the foundation for my project. The AVRDude [4] is the renowned software tool that is widely used to program new firmware in the AVR microcontroller family. The program was initially developed by Brian S. Dean and has become a common tool for programming the AVR microcontroller series because of its popularity among electronic hobbyists.

A user can use the AVRDude program to flash a new sensor firmware into a FoneAstra device using both wired and wireless connections. For a wired connection, the user needs a USB-to-RS232 Serial [7] converter cable to connect a FoneAstra device to a computer. AVRDude will use this connection to perform the in-system programming [8]. For a wireless connection, a user must have a computer that supports a Bluetooth connection because FoneAstra uses Bluetooth as its main channel for wireless transmission.

Another similar product is the AVRProg software tool [9]. This tool is actually the basic foundation of the AVRDude software and is distributed along with Atmel's AVRStudio software.

Even though these two software tools are powerful enough to perform a firmware update process on FoneAstra, the system that I have developed in my research project has two advantages. First, my system eliminates the need to have a computer when performing the firmware update. In

addition, it does not require any external programmer hardware. It is able to just use an Android smartphone with a Bluetooth module to perform firmware updates on FoneAstra.

Second, the system is significantly faster than AVRDude when performing a firmware update over Bluetooth. The next couple of sections will elaborate more on how the system works and present measurements of its performance.

Chapter 3. FONEASTRA

I will talk about the FoneAstra device briefly in this paper so that the reader can understand the relationship between the functionality of the board and my research more clearly. The FoneAstra device was created by Rohit Chaudhri, PhD, a former student in the University of Washington's Computer Science Department. Similar to Arduino [10], the device has an AVR microcontroller as its brain. Its main function is to extend the sensing capability of smartphones. FoneAstra users can attach external sensors to the board and open a corresponding application on the smartphone, and the device will be able to transfer information from and to the phone via a Bluetooth connection.

I was using a modified version of the FoneAstra v3.0 (see Fig. 1) for my research project. The modification reflects the changes that were made for the next version of the FoneAstra board (v3.1), which allows the Bluetooth module to be turned on initially during start-up. In other words, the board that I used is equivalent to the FoneAstra v3.1 board.

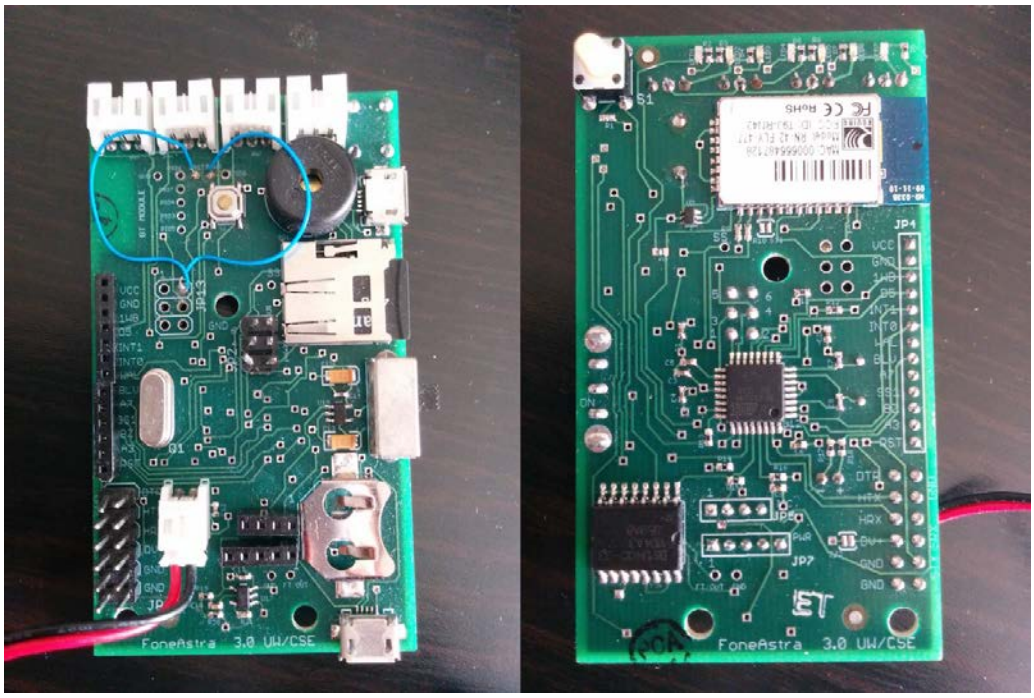

Figure 1: Custom FoneAstra Board Version 3.0.

The FoneAstra device has successfully become a low-cost, simple substitute for a monitoring system in developing countries because it uses only the FoneAstra board and a smartphone. One application of it is for remote monitoring of vaccine cold chains [11, 12]. The device has also been used for monitoring human breast milk temperature during a pasteurization process [13, 14]. Finally, a tracking system for water collection in Ethiopia used a FoneAstra device as its cost-effective solution [15].

The firmware management system that I developed helps extend the benefits of FoneAstra because users can now just use an Android smartphone to perform a FoneAstra firmware update. It simplifies the hardware requirement, making the deployment of the FoneAstra-based system cheaper and easier than a conventional monitoring system.

Chapter 4. FIRMWARE MANAGEMENT ON FONEASTRA

In this main chapter of the paper, I will explain the firmware management system for the FoneAstra device. The firmware management system for FoneAstra consists of four parts (see Fig.

2). The main part is the FROG program, an Android application that can program the FoneAstra board wirelessly via a Bluetooth connection. The second part is the FROGHexProvider Android application, whose function is to store sensor firmware for FoneAstra and call FROG to trigger the firmware update process. The third part is the FoneAstra bootloader, which handles receiving sensor firmware data from FROG and loads it into the appropriate memory location on the FoneAstra flash memory. Finally, the management system also includes the FROG operating system, a simple operating system that handles the firmware switching process.

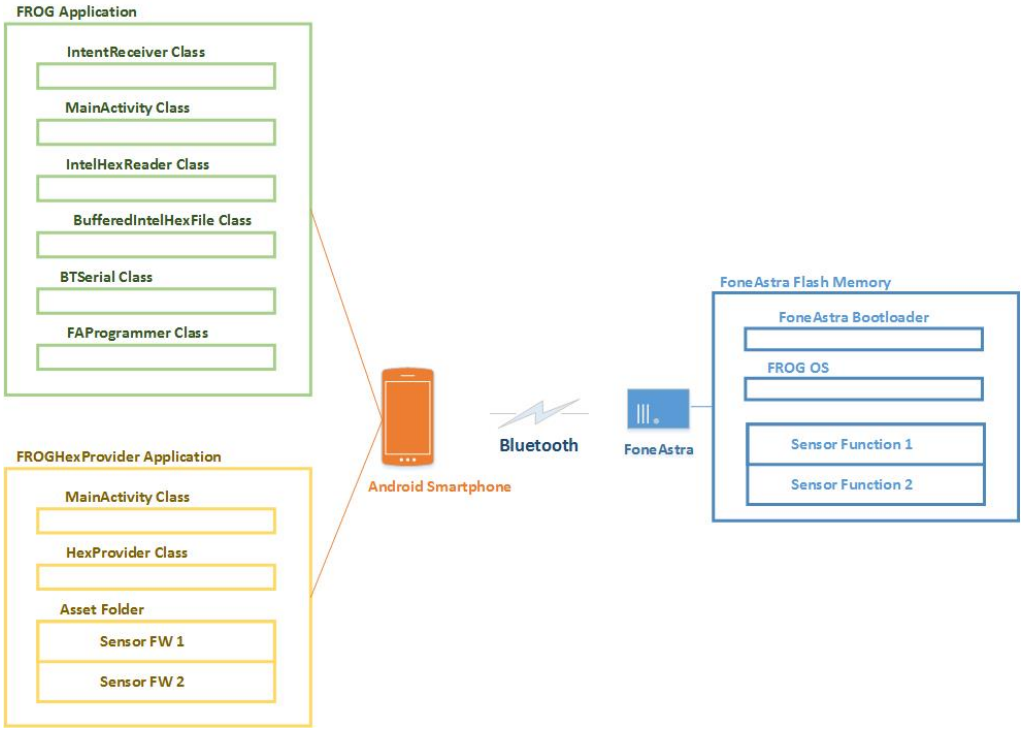

Figure 2: Firmware Management System on FoneAstra

4.1 FROG (FONEASTRA PROGRAMMER)

FROG stands for FoneAstra Programmer, an android application that can program a FoneAstra board wirelessly via a Bluetooth connection. The application grabs the target sensor firmware hex file from a separate Android application, called FROGHexProvider, and uses the information in it to flash a new firmware into the FoneAstra board. The application consists of several program classes [16], which will be elaborated on further in the next couple of sections.

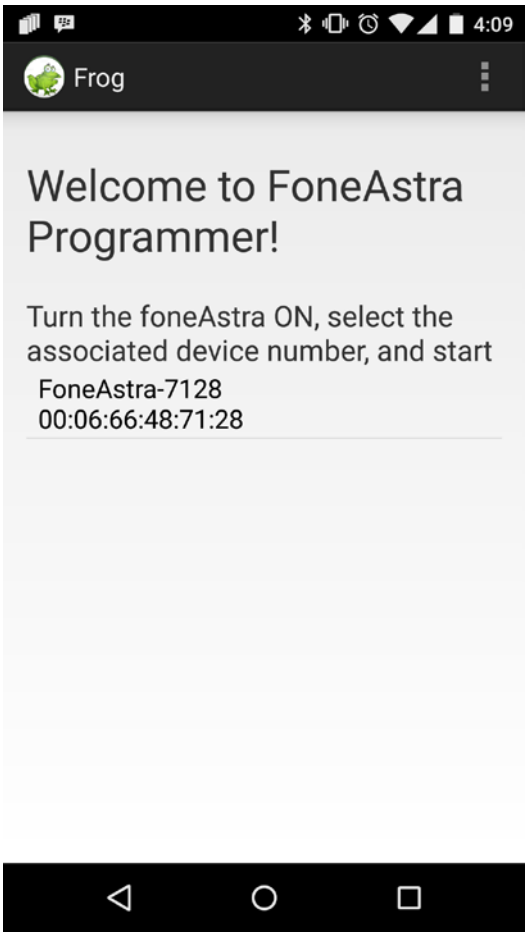

Figure 3: FoneAstra Programmer (FROG) User Interface

4.1.1 *MainActivity Class*

The MainActivity class controls the user interface (UI) of the FROG application. When the FROG application is started initially by the user, the init() function checks whether the Bluetooth module is enabled. If it is not, then the function will prompt the user to enable the Bluetooth module on his or her phone and pair it with the target FoneAstra device. Next, the function will use the BTFindDevices() function to query a list of all the FoneAstra devices paired with the phone.

Once the UI has shown all the paired devices, the user can select which FoneAstra he or she wants to program from the list. After the user selects the target device, the event will call the connectAndProgram() function, followed by BTSerial object's functions, to start connecting to the target device and initiate the firmware update process.

4.1.2 *IntentReceiver Class*

The class handles receiving an intent [17] from the FROGHexProvider application, which contains the uniform resource identifier (URI) of the hex file location. When an intent is received from the provider application, FROG application will call the Hex Provider class on the FROGHexProvider app and pass along the URI of the hex file. The class then uses the URI to create a socket for reading the content of the hex file from the asset folder of the FROGHexProvider app. Once the socket of the hex file has been created successfully, the class will then start the MainActivity of the FROG application.

4.1.3 *BTSerial Class*

This class manages the Bluetooth connection between the FROG application and the target FoneAstra device. It also triggers the firmware update process once a Bluetooth connection has been established. First, the MainActivity will call the connect() function from the BTSerial class. The connect function will start a separate thread from the MainActivity thread based on the BluetoothIOThread subclass. The reason for having a separate thread to handle the Bluetooth connection is because I want to prevent this process from blocking the MainActivity thread if an error happened during the connection establishment.

Once the BluetoothIOThread starts to run, it will try to connect to the target device for a maximum of seven attempts. If, after the maximum number of attempts, the phone is not able to connect to the target device, the thread will be terminated and an error message will be displayed to the user. If the phone successfully connects to the target device, the thread will call the startProg() function, which will use an FAProgrammer object to manage the firmware update process on the target FoneAstra device.

4.1.4 *BufferedIntelHexFile Class*

This class acts as temporary storage for the firmware hex file’s data. The reason for temporarily storing the hex file data in this object is to avoid the overhead caused by reopening and reclosing the Android file IO socket, thus improving the latency of the firmware update process.

The class uses the built-in Android ArrayList class to store the hex file data. The ArrayList class is chosen because of its flexibility in scaling its array size. To parse the hex file, the class uses an IntelHexReader object to interpret a line of the Intel hex file. This one-line parsing process is repeated until the object finished reading the entire hex file.

4.1.5 *IntelHexReader Class*

As I have mentioned above, this class handles the process of parsing one line from the input hex file. The input hex file uses a standard Intel hex file format [6]. The parsing process starts by reading the start character, ":". Then the process continues by reading the byte count, which is the number of bytes of data presented on the current line. Next, it reads the target address where the data will be stored, followed by the record type. If the record type is the same as the data record, the data information on that line will be read. Once the function finishes reading the data, it will then read the checksum [18] byte at the end of the process.

4.1.6 *FAProgrammer Class*

This is the core class for the FROG application; it manages the firmware update process for the FoneAstra device. An object of this class will be initialized inside the BluetoothIOThread class and will receive a BluetoothIOThread object for transmitting data via Bluetooth. The protocol that this class uses to talk to FoneAstra's bootloader is the STK500 protocol [19, 20, 9]. This class implements a stripped-down version of the protocol that supports only some core commands of the STK500 protocol. Such commands are shown in the table 1 below.

Table 1: Core Functions of STK500 Protocol

STK500 Command Name	FROG Command Name	Command Value	Description
Cmnd_STK_GET_SYNC	STK_GET_SYNCH	0x30	Regain synchronization with target device
Cmnd_STK_GET_PARAMETER	STK_GET_PARAMETER	0x41	Get parameter from target device
Cmnd_STK_SET_DEVICE	STK_SET_DEVICE	0x42	Set target device's programming parameter
Cmnd_SET_DEVICE_EXT	STK_SET_DEVICE_EXT	0x45	Set extended programming parameters for target device
Cmnd_STK_ENTER_PROGMODE	STK_ENTER_PROGMODE	0x50	Enter programming mode
Cmnd_STK_READ_SIGN	STK_READ_SIGN	0x75	Read signature bytes
Cmnd_STK_PROG_FUSE_EXT	STK_PROG_FUSE_EXT	0x65	Program fuse bits
Cmnd_LOAD_ADDRESS	STK_LOAD_ADDRESS	0x55	Load 16-bit address to target device
Cmnd_STK_PROG_PAGE	STK_PROG_PAGE	0x64	Download and program a block of data to target device
Cmnd_STK_READ_PAGE	STK_READ_PAGE	0x74	Read a block of data from device's FLASH memory
Cmnd_STK_LEAVE_PROGMODE	STK_LEAVE_PROGMODE	0x51	Leave programming mode

Initially, the programAVR() function will be called from the BluetoothIOThread class and will receive the URI of the target hex file. The function calls the assignHexFile() function at the beginning of the code, using an IntelHexReader object to read the hex file and put the firmware information into a temporary buffer object called BufferedIntelHexFile.

After putting the hex file information into the BufferedIntelHexFile object, the programAVR() function will start sending a sequence of programming commands to the target FoneAstra's bootloader. The sequence is shown in Figure 4. Moreover, the overall firmware programming sequence of the whole firmware management system is shown in Figure 5.

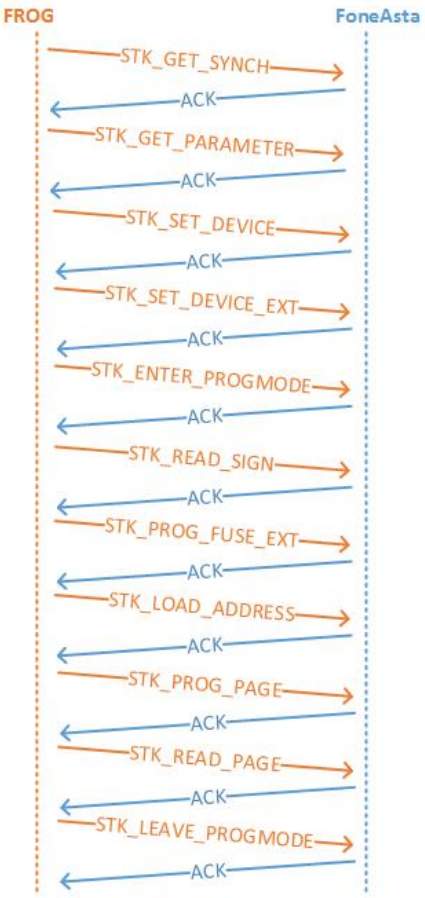

Figure 4: FROG Programming Commands' Flow

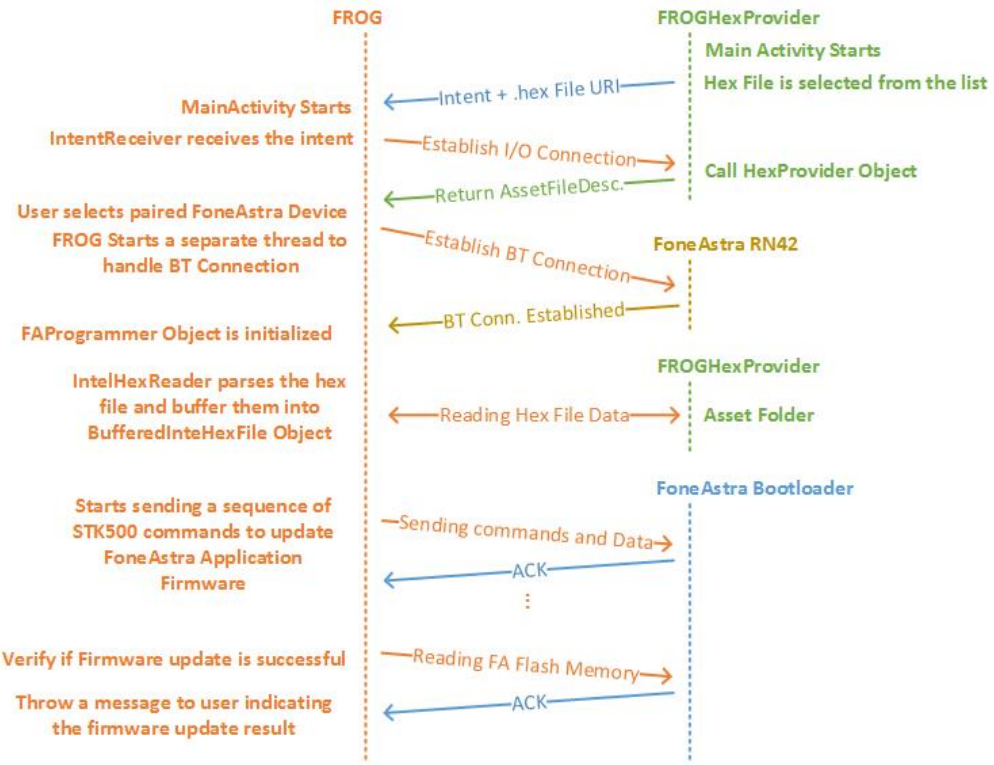

Figure 5: Firmware Update Process on FoneAstra

Each of the STK500 protocol functions has a retry mechanism where if, after sending a command packet, the function hasn't received the acknowledged bytes from the bootloader, the function will resend the packet to the bootloader. If after reaching the maximum number of retries, the function is still not getting any acknowledgment from the bootloader, the function will terminate the firmware update process and an error message will be displayed to the user.

Note that at the end of writing the flash memory process, a readPage command is invoked to read the content of the written flash memory. These data are used to verify whether the new content of the flash memory matches the content generated from the firmware hex file. If it does, then the firmware update process has completed successfully.

4.2 FROGHEXPROVIDER

The implementation of the FROGHexProvider application aims to demonstrate how FoneAstra developers can include their sensor firmware into their Android application packages

and use the FROG application to program the firmware into FoneAstra. This provider app stores the firmware hex files in its asset folder. In the source code, notice that an .mp3 name extension is used on top of the hex file's name. This is because Android has a known issue with reading the .hex file extension. The application consists of two classes: the MainActivity and HexProvider classes. In the next couple sections, I will elaborate on these classes in greater detail.

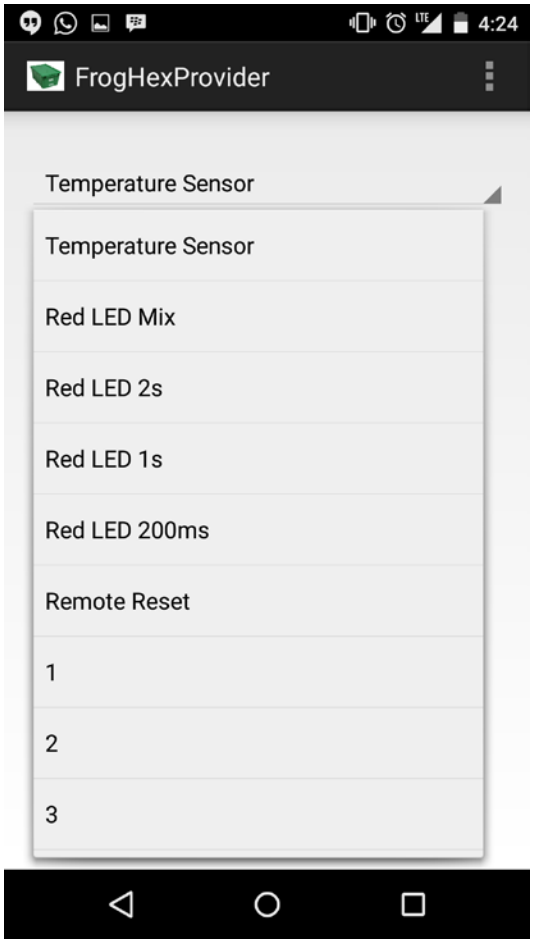

Figure 6: FROGHexProvider User Interface

4.2.1 *MainActivity Class*

The MainActivity class manages the user interface of the application. When a user first initializes the application, the onCreate() function will generate a list of all the available sensor firmware hex files that the FROGHexProvider application is carrying in its asset folder. When the user chooses a firmware hex file, the application will send an intent with the corresponding firmware hex file's URI to the FROG application. The FROG's IntentReceiver class will then

capture this intent and use the URI information to create a socket for reading the firmware hex file from the provider application's asset folder.

4.2.2 *HexProvider Class*

This class is responsible for receiving the sensor firmware hex file's URI and using it to create an AssetFileDescriptor object and send the object to FROG application. The FROG application will use this object to read the content of the firmware hex file from the asset folder. The URI is received from the FROG immediately after the application receives an intent from the provider app.

4.3 FONEASTRA BOOTLOADER

The bootloader for FoneAstra is a customized version of a serial bootloader for Atmel Atmega AVR controllers [21]. The bootloader is stored in the bootloader section of the flash memory inside the Atmega328p MCU on the FoneAstra board. It handles the communication between the FROG application and the FoneAstra device.

I carried out two modifications on the bootloader code. The first is to make the bootloader utilizes the two LEDs on the FoneAstra: the red and green led so that it can give the user extra information about what is currently happening during the firmware update process. If during the bootloader state the FoneAstra device receives a byte of data, the red LED will flash. When the device is transmitting a byte of data, the bootloader code will make the green LED flash. The second feature is a prolonged bootloader wait time of 10 seconds, giving the FROG application extra time to establish a Bluetooth connection with FoneAstra while the device is in the bootloader state.

4.4 FROG OPERATING SYSTEM

The FROG operating system is a firmware that helps FROG developers include multiple sensor functions in a single firmware and handles the switching process from one sensor function to another. The diagram of the system is shown in Figure 7.

The operating system consists of a main function that has an array of memory locations. These locations correspond to the locations of all the sensor functions stored in the flash memory.

The main function has an infinite loop, and with every iteration, it checks the control command data that are being received from a client Android application. Notice that the client application sends the control command data via a Bluetooth connection. Using this control command data and the array of memory addresses, the main function will then decide which sensor function to execute next. In addition, each of the sensor functions also has an infinite loop that enables it to put some lines of code for checking the control command data received from the client application in every iteration. The loop and the control command data checking code are necessary to allow the sensor function code to switch back to the main function or to another sensor function.

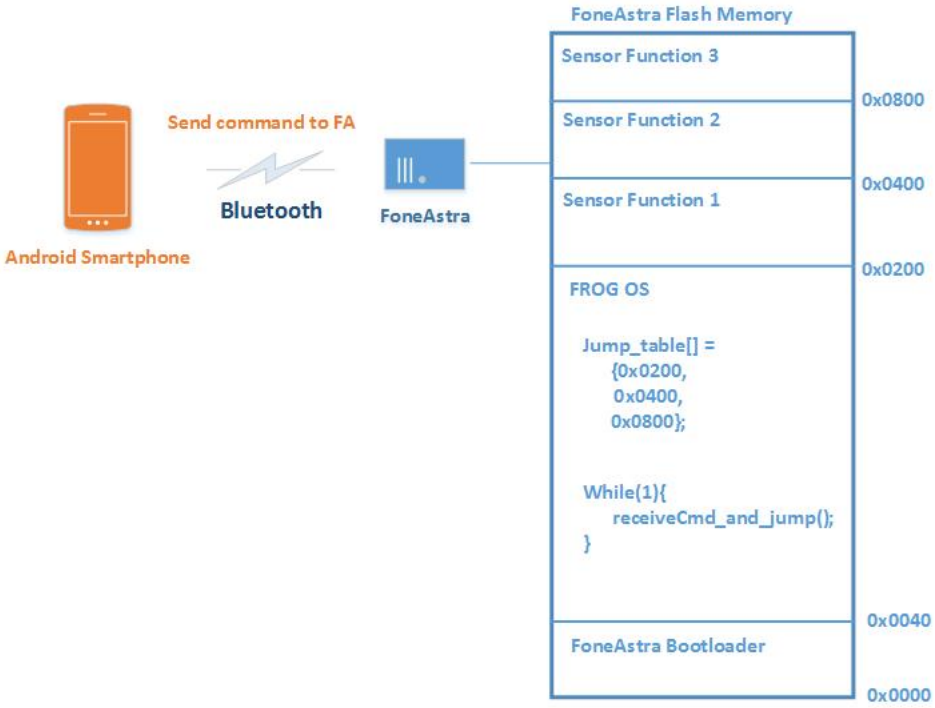

Figure 7: Illustration of the FROG Operating System

The FROG operating system will help our future work with FoneAstra in which we will attempt to enable the FROG application to program each of the sensor functions independently of each other into a specific location in FoneAstra's flash memory. In addition, this firmware programming process will not delete or overwrite the main function or other sensor functions that are already stored in the memory. This way, FoneAstra users can have more than one sensor functions in the device and do not need to do a firmware update every time they want to switch

from one sensor function to another. The FROG operating system will help facilitate the switching process on the fly.

Chapter 5. FROG PERFORMANCE MEASUREMENTS

The FROG performance measurement data indicate very good results when compared to the AVRDude software. Table 2 shows the comparison of timing results between the two software tools when they are used on both different hardware and connections. For the FROG timing measurement, I used three different types of hardware—Nexus 5, Nexus S, and Nexus 7 tablets—whereas for the AVRDude software, I used a MacBook pro 15″ retina display (2012).

According to the data from table 2, FROG firmware update timing is pretty much the same on all three mobile devices. A significant difference can be seen, however, when comparing the performance of FROG and the AVRDude software. When FROG is compared to AVRDude with the FoneAstra device connected physically to the computer, the AVRDude software is, as expected, significantly faster than FROG. This is because a wired connection will send the data transmission fewer software layers and handshaking processes. In the next measurement, FROG was compared with the same AVRDude software, but instead of a wired connection, AVRDude was using a wireless Bluetooth connection. The result is quite promising. The FROG is notably faster than AVRDude. The possible reason for this outstanding performance of the FROG is that the FROG implements fewer STK500 protocol functions, which means it is more lightweight than AVRDude, which implements the complete STK500 protocol.

Table 2: FROG and AVRDude Firmware Update Timing Measurements

Application + Hardware	Firmware Update Timing (s)				
	FoneAstra30TempSketch.cpp.hex (12kb)	**Blink1s (823b)**	**FoneAstra30.cpp.hex (39kb)**	**FoneAstraHMBStandalone (68kb)**	**FoneAstraVaxSketchV1.cpp.hex (58kb)**
FROG + Nexus 5 (Bluetooth)	13.73	2.11	41.36	59.18	57.76
FROG + Nexus S (Bluetooth)	13.64	2.17	40.97	59.43	57.3
FROG + Nexus 7 Tablet (Bluetooth)	13.22	2.23	41.44	59.55	57.75
AVRDude + MacBook Pro 15″ 2012 (USB-Serial Conn)	4.1	0.316	12.93	22.444	19.14
AVRDude + MacBook Pro 15″ 2012 (Bluetooth)	43.746	3.726	137.394	238.562	203.526

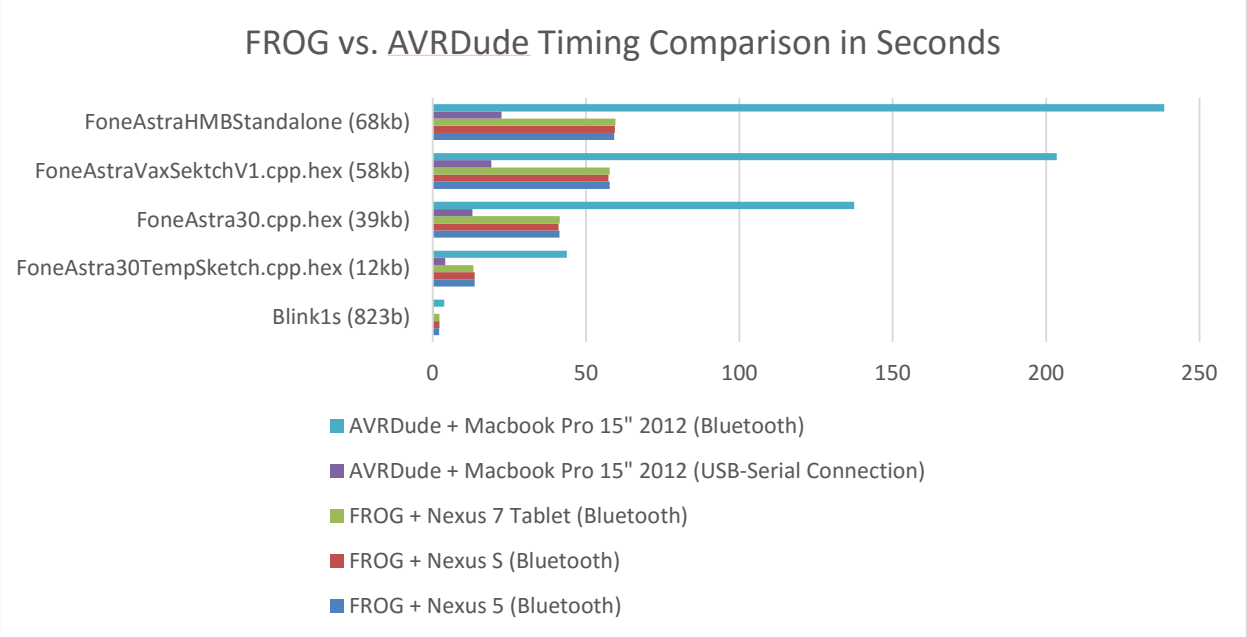

Figure 8: FROG and AVRDude Firmware Update Timing Diagram

Chapter 6. FUTURE WORK

In addition to all of the work described above, the firmware management project can still be improved further. One of our ideas is to expand FROG's functionality so that it can program a sensor firmware into a specific memory location without the need to delete the entire contents of the flash memory. This feature will complete the FROG operating system, allowing a user to flash in just one sensor function into a specific memory location without updating the whole flash memory and then use the operating system to handle switching between sensor functions.

The work will involve some modification of the compiler and linker software tools so that the resulting firmware hex file will have the right configurations for the program code that the file contains. Such configurations include the jump table information and the location of the main function.

Chapter 7. CONCLUSIONS

In this research project, I successfully developed the FROG Android application that is able to program new firmware into a FoneAstra board. This work is very useful for FoneAstra users because now they do not have to use a wired connection or a separate programmer device to program the board. Instead, they can just use the same smartphone that they use in tandem with FoneAstra, install the FROG application, and use it to flash a new sensor firmware.

Another of my achievements in this research project was that I was able to create a prototype system in which the firmware hex file can be packaged into an Android application and FROG can be used to perform the firmware update on FoneAstra. The Android application is called FROGHexProvider, and FoneAstra sensor developers can include their sensor firmware in their sensor's Android application and use the FROG application to perform the firmware download process.

Finally, I propitiously developed a lightweight operating system for FoneAstra that allows users to switch from one sensor function to another. This operating system will become the foundation for the next version of FoneAstra's firmware management system, for which we want

FROG to be able to program multiple sensor functions independently of each other without having to erase the entire contents of FoneAstra's flash memory.

All in all, the newly developed firmware management system for FoneAstra improves the benefits that the board already offers. It simplifies the hardware requirement for performing a firmware update on FoneAstra, which makes deploying a FoneAstra-based system cheaper compared to a conventional monitoring system. The system also eliminates the need for a wired connection to perform firmware updates because it uses a wireless Bluetooth connection. Equally important, the firmware update system is significantly faster than AVRDude in performing firmware updates on FoneAstra over Bluetooth, making the FoneAstra ecosystem more time efficient to deploy.

REFERENCES

[1] R. Chaudhri, G. Borriello and W. Thies, "FoneAstra: making mobile phones smarter," in *In Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions (NSDR '10)*, New York: ACM, 2010.

[2] "Android," Google, [Online]. Available: https://www.android.com/.

[3] "Atmel AVR Microcontroller," Atmel Corporation, [Online]. Available: http://www.atmel.com/products/microcontrollers/avr/.

[4] "www.nongnu.org - AVRDude Software," [Online]. Available: http://www.nongnu.org/avrdude/.

[5] "Wikipedia - Bluetooth," [Online]. Available: https://en.wikipedia.org/wiki/Bluetooth.

[6] Intel Corporation, "Hexadecimal Object File Format Specification," 6 January 1988. [Online]. Available: http://microsym.com/editor/assets/intelhex.pdf.

[7] "Wikipedia - RS232," [Online]. Available: https://en.wikipedia.org/wiki/RS-232.

[8] "Wikipedia - In-system programming," [Online]. Available: https://en.wikipedia.org/wiki/In-system_programming.

[9] Atmel Corporation, "AVRProg User Guide," January 1998. [Online]. Available: http://www.atmel.com/Images/doc1021.pdf.

[10] "Arduino," [Online]. Available: https://www.arduino.cc/.

[11] R. Chaudhri, G. Borriello and R. Anderson, "Monitoring Vaccine Cold Chains in Developing Countries," *IEEE Pervasive Computing (July 2012)*, vol. 11, no. 3, pp. 26-33, 2012.

[12] R. Chaudhri, E. O'Rourke, S. McGuire, R. Anderson and G. Borriello, "FoneAstra: enabling remote monitoring of vaccine cold-chains using commodity mobile phones," in *In Proceedings of the First ACM Symposium on Computing for Development (ACM DEV '10)*, New York: ACM, 2010.

[13] R. Chaudhri, D. Vlachos, G. Borriello, K. Israel-Ballard, A. Coutsoudis, P. Reimers and N. Perin, "Decentralized human milk banking with ODK sensors," in *In Proceedings of the 3rd ACM Symposium on Computing for Development (ACM DEV '13)*, New York: ACM, 2013.

[14] R. Chaudhri, D. Vlachos, J. Kaza, J. Palludan, N. Bilbao, T. Martin, G. Borriello, B. Kolko and K. Israel-Ballard, "A system for safe flash-heat pasteurization of human breast milk," in *In Proceedings of the 5th ACM workshop on Networked systems for developing regions (NSDR '11)*, New York: ACM, 2011.

[15] R. Chaudhri, R. Sodt, K. Lieberg, J. Chilton, G. Borriello, Y. Masuda and J. Cook, "Sensors and Smartphones: Tracking Water Collection in Rural Ethiopia," *IEEE Pervasive Computing (July 2012)*, vol. 11, no. 3, pp. 15-24, 2012.

[16] "Android Developer - Class," Google, [Online]. Available: http://developer.android.com/reference/java/lang/Class.html.

[17] "Android Developer - Intent," Google, [Online]. Available: http://developer.android.com/reference/android/content/Intent.html.

[18] "Wikipedia - Checksum," [Online]. Available: https://en.wikipedia.org/wiki/Checksum.

[19] Atmel Corporation, "AVR061: STK500 Communication Protocol," April 2003. [Online]. Available: http://www.atmel.com/images/doc2525.pdf.

[20] Atmel Corporation, "AVR068: STK500 Communication Protocol," June 2006. [Online]. Available: http://www.atmel.com/images/doc2591.pdf.

[21] Arduino, LLC, "Bootloader Development," [Online]. Available: https://www.arduino.cc/en/Hacking/Bootloader?from=Tutorial.Bootloader.

[22] "Wikipedia - USB," [Online]. Available: https://en.wikipedia.org/wiki/USB.

[23] R. Chaudhri, W. Brunette, M. Goel, R. Sodt, J. VanOrden, M. Falcone and G. Borriello, "Open data kit sensors: mobile data collection with wired and wireless sensors," in *In Proceedings of the 2nd ACM Symposium on Computing for Development (ACM DEV '12)*, New York: ACM, 2012.

[24] R. Chaudhri, W. Brunette, B. Hemingway and G. Borriello, "ODK sensors: an application-level sensor framework for Android devices," in *In Proceedings of the 3rd ACM Symposium on Computing for Development (ACM DEV '13)*, New York: ACM, 2013.

[25] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. V. Orden and G. Borriello, "Open data kit sensors: a sensor integration framework for android at the application-level," in *In Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12)*, New York: ACM, 2012.

[26] W. Brunette, M. Sundt, N. Dell, R. Chaudhri, N. Breit and G. Borriello, "Open data kit 2.0: expanding and refining information services for developing regions," in *In Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile '13)*, New York: ACM, 2013.