µLeech: A Side-Channel Evaluation Platform for Next Generation Trusted Embedded Systems

by

Michael Moukarzel

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical Computer Engineering

_____

August 2015

APPROVED:

_____	_____
Professor Berk Sunar	Professor Thomas Eisenbarth
ECE Department	ECE Department
Thesis Adviser	Thesis Committee

_____

Professor Yehia Massoud

ECE Department Head

Abstract

We propose a new embedded trusted platform module for next generation power scavenging devices. Such power scavenging devices are already in the current market. For instance, the Square point-of-sale reader uses the microphone/speaker interface of a smartphone for both communications and to charge up the power supply. While such devices are already widely deployed in the market and used as trusted devices in security critical applications they have not been properly evaluated yet. Our trusted module is a dedicated microprocessor that can preform cryptographic operations and store cryptographic keys internally. This power scavenging trusted module will provide a secure cryptographic platform for any smartphone.

The second iteration of our device will be a side-channel evaluation platform for power scavenging devices. This evaluation platform will focus on evaluating leakage characteristics, it will include all the features of our trusted module, i.e. complicated power handling including scavenging from the smartphone and communications through the microphone/speaker interface. Our design will also included the on-board ports to facilitate easy acquisition of high quality power signals for further side-channel analysis. Our evaluation platform will provide the ability for security researchers to analyze leakage in next generation mobile attached embedded devices and to develop and enroll countermeasures.

Preface

In this work I describe the research that was conducted during my graduate master studies at Worcester Polytechnic Institute. I designed and produced µLeech, a power scavenging Trusted Platform Module for smartphones. The next iteration of µLeech will be as a side-channel evaluation board to help secure other similar power scavenging devices.

The work I presented in this thesis would not have been possible without the help and support of a few people. First and foremost many thanks go out to my adviser *Professor Berk Sunar*. His advice and expertise resolved many hurdles that I encountered throughout my research. I am also grateful to *Professor Thomas Eisenbarth* for the valuable suggestions, comments and advise that he gave me as a member of my thesis committee. I would also like to thank *Professor Stephen J. Bitar*, his hardware expertise where essential in the design and understanding of the power aspect of µLeech.

I would also like to thank my colleague and friend *Yarkin Doroz*. His friendship and support helped me throughout the course of this thesis. Last but not least, I would like to thank my family for being patient with me and for their support during my graduate studies. Special thanks to my Mother and Father who continued to push and support me from start to finish.

To all of you thank you very much!

Michael Moukarzel

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction

1.1 Motivation

With the proliferation of cheap bulk SSD storage and better batteries in the last few years we are experiencing an explosion in the number of devices flooding the market, e.g. smartphone connected point-of-sale devices, home monitoring devices, e.g. NEST, fitness monitoring devices, e.g Fitbit, and smart-watches. The Square point-of-sale unit, in particular, has gained traction in the market. The versatility and convenience of the design has propelled the product into widespread adoption. What makes the Square design unique is that it uses the microphone/speaker interface to power the unit as well as to facilitate the communication between the device and the smartphone. Since speaker/microphone is the only universal interface present in all smartphones the Square device can be transparently moved from one smartphone to the other.

In the development of our design we started by examining the project Hijack. Hijack was a project developed at the University of Michigan by Professor Prabal Dutta, and his graduate students Sonal Verma and Andrew Robinson. They designed Hijack a cubic-inch peripheral sensor ecosystem for the mobile phone. Hijack was developed to optimally generate power from an iPhone to an MSP430 to collect sensor data and relay that information back to the iPhone. In this way we modeled our power generation circuit and communication protocol off of Hijack's. Although ultimately our communication protocol differs as we do not need to communicate continuous sensor data, but short burst of data, draining less power in the process.

Ultimately µLeech will act as a universal power scavenging Trusted Platform Module (TPM) for smartphones. The next iteration of µLeech will include a modified TPM, that will server as a side-channel evaluation platform. Such an evaluation platform can be used to help secure µLeech and other such power scavenging devices from side channel leakage.

1.2 Thesis Outline

Chapter 2 explains the need for a side-channel evaluation platform for power scavenging devices. That one design of µLeech will be to act as a side-channel evaluation platform.

Chapter 3 describes the design and implementation cycles that µLeech underwent. Giving an overview of the hardware and software tools used for its development.

Chapter 4 focuses on the power circuit of our µLeech. Focusing on our power siphoning circuit, sleep circuit, and capacitor bank implementations.

Chapter 5 describes the communication protocol in detail. Illustrating the Manchester encoding used, as well as the actual implementation. This section focuses on the two-way communication between a smart phone and our µLeech device.

Chapter 6 presents the results of our design implementation. Comparing on the different sleep modes, power consumptions, and AES implementations. Focusing on their influence on the overall design.

Chapter 7 shows the final stages of development for our µLeech. This includes the smartphone app developed to communicated with our devices. As well as the PCB prototype developed after our bread-boarded implementation.

Chapter 8 concludes this work with a short summary of the results and some recommendations for further work and optimizations

Chapter 2

2 Evaluation Platform

μLeech will have two final design iterations. The initial iteration will be to develop μLeech as a power scavenging Trusted Platform Module (TPM) for smartphones. The next iteration will be to transform μLeech into an embedded side-channel evaluation platform for next generation power scavenging devices.

2.1 Existing Side-Channel Evaluation Platforms and Tools

There are many existing evaluation boards, tools, and platforms available that can be used to assess vulnerabilities to side-channel attacks. Some are commercially available and others were developed for academic use. The following is a list of a few existing FPGA based side-channel evaluation platforms

Side-channel Attack Standard Evaluation Boards (SASEBO) are standard evaluation boards developed by National Institute of Advanced Industrial Science and Technology at Tohoku University by Toppan Technical Desin Center Co, Ltd. [7]. SASEBO Boards were developed to perform security tests for side-channel attacks. Their goal was to standardize the testing requirements of cryptographic modules. They are capable of evaluating side-channel attacks against cryptographic hardware with FPGAs and cryptographic software with micro processor function.

Flexible Open-source Board for Side-channel analysis (FOBOS) is an academic platform developed at George Mason University [5]. FOBOS is a platform for implementation attack resistance testing. FOBOS aims to provide an open-source platform. Their platform can be used to evaluate the effectiveness of side-channel analysis countermeasures on different FPGA platforms. FOBOS includes support for multiple FPGA devices and the necessary software to run differential power analysis attacks.

Side Channel Analysis Resistant Framework (SCARF) is an open-source academic tool developed by the Electronics and Telecommunications Research Institute [9]. SCARF can be use for testing countermeasures for side-channel and fault attacks. They include a number of custom evaluation boards to test the attack resistance of smart-cards, microprocessors, and FPGAs. SCARF only supports testing of the devices included in its custom evaluation boards.

The following are a few tools that can be used in conjunction with the previously listed side-channel evaluation platforms:

ChipWhisperer is an open-source tool-chain originally developed by Colin O'Flynn . His tool-chain is an open-source tool-chain for embedded hardware security research. It can be used for side-channel power analysis and glitching. This tool-chain is especially known for their innovative synchronous capture technology. Similar tools are commercially available but far more expensive and closed-source.

The DPA Workstation is a closed-source device developed by Cryptography Research Inc, a division of Rambus Inc. [10]. It can be used to perform side-channel analysis including differential power or electromagnetic analysis on embedded systems. The DPA Workstation includes its own environment and proprietary software that can perform side-channel analysis on all major standard ciphers.

InspectorSCA is a closed-source device developed by Riscure [11]. This platform can be used for side-channel analysis and fault analysis. It includes fault injection hardware and includes proprietary

software that can perform side-channel and fault attacks on standard ciphers.

2.2 µLeech Side-Channel Evaluation Platform

As smartphones and tablet continue to grow in popularity, there will be a growing development of power scavenging devices that can interface with your these devices using the auxiliary port. The Square point-of-sale reader is an example of a power scavenging device that has become extremely popular. The Square acts as a credit-card reader for you mobile devices, sending sensitive information to you mobile devices thru the auxiliary port. The square and other such devices are already widely deployed in the current market. They are used as trusted devices in security critical applications, such as credit-card information, and yet they have not been properly evaluated for side-channel leakage.

The initial iteration of µLeech is a power scavenging TPM that can be used with all mobile devices. It includes all the features expected of a device that uses the auxiliary port: Power is scavenged from the right audio channel, two-way communication through the left audio channel and the microphone.

The next iteration of our platform will be a side-channel evaluation platform. This platform will provided the ability for security researchers to analyze side channel leakage in next generation mobile attached embedded devices. Just like SASEBO, FOBOS, and SCARF the µLeech evaluation platform will include high quality on-board peripherals to our power and clock signals. Allowing for easy access of signals necessary for side-channel analysis. This platform will allow for the development and enrollment of countermeasures.

Chapter 3

3 Methodology

This chapter describes the design procedures used in the development of our device, µLeech. It also includes a description of the hardware and software tools used in the development of µLeech.

3.1 Design Cycle

The general design cycle for µLeech consisted of the following steps:

1. Research of other side-channel evaluation platforms
2. Research of smartphone power siphoning platforms
3. Focused research on Hijack
4. Modifying Hijack to work with an Android platform
5. µLeech hardware prototype, bread-boarded
6. µLeech firmware development
7. AES implementation on µLeech
8. Power measurements and optimizations of µLeech
9. µLeech PCB development and side-channel evaluation platform version

The steps outlined above were not all completed in this order, and some of the steps were

continuously being completed during the entire development process.

Steps 1 and 2 consisted of background research that was continuously performed during the entire development process. Step 3 was a focused research on Hijack, a result from our background research. We used Hijack as a template for our device, specifically for our power generation and communication protocol. Step 4 through 9 deal with the development of the software and hardware prototype of µLeech.

3.2 Tools

There were many different tools and environments used in the development of µLeech. The tools used can be broken into three main development parts: android app, processor firmware, and hardware design. The development of each of these parts used different programs, programming languages, and software environments. Even though each of these parts were developed independently, they eventually had to work together to create µLeech.

First we had to develop the smartphone application that would need to be installed and run to generate power and interface with our µLeech device. Then we had to develop the AVR processor firmware that would be running on our µLeech's UC3-L0 processor. Initially the processor firmware was being tested on an AVR development board, UC3-L0 explained. However as the development of firmware progress custom hardware had to be built. The hardware development progressed parallel to the firmware development. The hardware was initially bread-boarded and followed by a PCB iteration, once the hardware design was completed and tested.

It is important to note that the tools we used are not the only possible tools that could have been used. There are many programs and programing environments available, the following tools are the ones we decided to use in our development process.

3.2.1 Android App

Three software tools were used in the hardware design of µLeech: Eclipse Integrated Development Environment (IDE), Android Development Tool (ADT) Plugin, and Android Software Development Kit(SDK). Android apps are Java based and therefore can be programmed in Eclipse. There are many different programming environments that can be used for android app development. However the android SDK and ADT are required for android app development. These can be

conveniently downloaded and installed directly through Eclipse.

3.2.2 Processor Firmware

Only one software product was used for our processor firmware development: Atmel Studio 6.0. The AVR processor used in µLeech was Atmel's UC3-L0. As such Atmel provides the programming environments and compilers needed for developing firmware. Atmel Studio is an environment that lets you write, compile, and program your code. It also includes an Atmel Software Framework (ASF) wizard. The ASF wizard allows you to use and add any of Atmel's development libraries directly to your project. The processor firmware code we developed for µLeech is written in C and includes several of Atmel's libraries. The libraries used and added using the ASF wizard can be found in asf.h, Appendix D.

When programming the processor we used AVR ONE, a JTAG programmer. As our design is intended for low power use we decided to simply use JTAG and not add a USB in circuit programmer.

3.2.3 Hardware Design

Two software products were used in the hardware design of µLeech: ExpressPCB and CadSoft Eagle. Both of these software are used in designing PCB schematics. Our eventual goal were two separate PCBs, one of µLeech and the other its evaluation platform. Before we could manufacture or even create the final PCB iterations we had to first create and prototype the design. The design prototype was initially bread-boarded allowing us to build it in steps and to modify the circuit when we needed. Once the hardware design was confirmed to be working properly then we manufactured the final PCBs. However, some of the components could not be used on a bread-board because they were surface mount (SMT).

There are some SMT components that have bread-boarding sockets available for them. Some of the components we used had no sockets available. The biggest problem was our AVR processor, a 48-pin SMT component, had no available bread-boarding sockets available. As a result we manufactured our own. ExpressPCB was used to design the initial PCBs for bread-boarding and power measurement optimization tests. ExpressPCB was used because its an easy to use and quick design software.

Eventually once the hardware design was completed and tested thoroughly, the whole design was implemented using Eagle. CadSoft Eagle is a much more advanced PCB design environment software. Eagle was used to develop the two final PCB versions of µLeech. Eagle has a much more extensive library of component footprints for SMT layouts. There is also a user based library of custom footprints that is very extensive. Many of the components used in our design had costume made footprints or were downloaded from the user based library. The final PCB versions were designed using eagle. The final µLeech PCB versions can be found in Figure 24 and Figure 25.

Chapter 4

4 Power

When dealing with most cryptographic computations you are dealing with short highly intensive computation. Taking advantage of this we designed our power circuit with a capacitor bank, that could be discharged and recharged. This allows our processor to preform any cryptographic operation while draining the capacitor bank. We optimized our design to allow one round of AES to be computed before depleting our capacitor bank. Once the capacitor bank is depleted our processor will hibernate in sleep mode allowing the capacitor bank to recharge again for another round of computations when needed. To achieve this we developed a low power sleep circuit that could be used to notify our processor when to sleep and wake. Allowing our processor to operate off of the limited power generated by our smartphone.

4.1 Power Siphoning

In keeping with the concept of a universal jack our power has to be generated from the auxiliary jack. This would mean we would have to siphon power from an audio waveform. To determine the available power from our Nexus 4 headset port, we generated a 500mV peak to peak AC audio waveform. A load resistance was connected between the right audio channel and the common line of the auxiliary jack. While measuring the output voltage across the resistor and the load current, the load resistance was varied from 0Ω to 15KΩ. We measured a total of 24 different resistor values. Using this

data we generated a linear fit, represented by the red linear IV curve in Figure 1. Using this IV curve, we then generated the blue power transfer curve in Figure 1. The power transfer curves shows that the maximum power transfer for our Nexus 4 occurs at 85mVrms and 18.87mArms, with an ideal load of 4.51Ω.

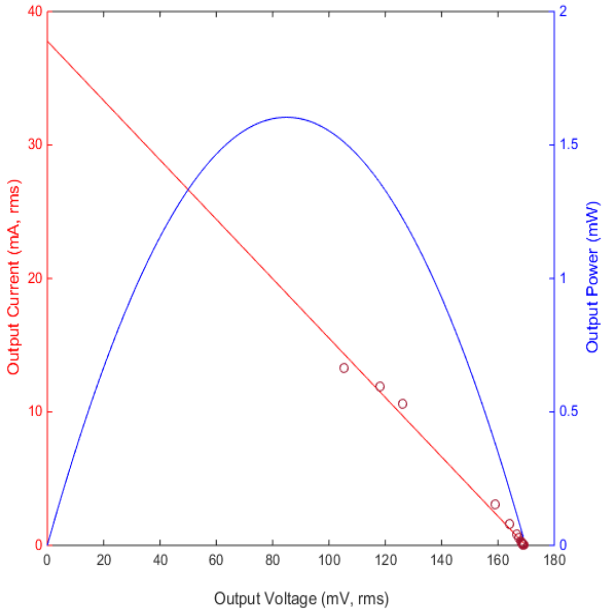

Figure 1: Nexus 4 headset jack available power

The data in Figure 1 shows that it is possible to draw 1.61mW from an ideal matched load of 4.51Ω. For this power to be useful our power siphoning circuit will have to be rectify the 500mV peak to peak waveform from AC to DC, boosted it, and filter it. The power siphoning circuit we implemented was modeled off of the Hijack's power siphoning circuit, with modifications to their voltage regulator and an additional capacitor bank. The circuit we implemented is depicted in Figure 2.

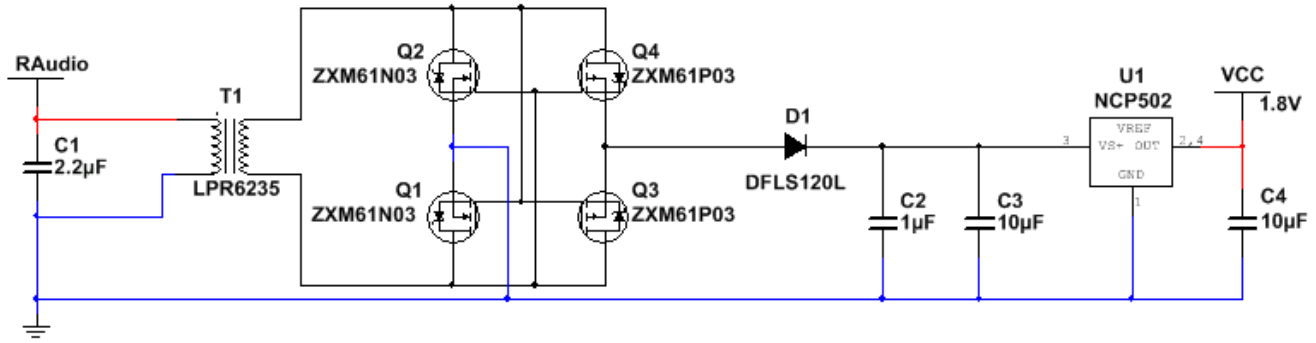

Figure 2: Power Circuit

The input of the circuit Figure 2 is the Right audio channel of the smartphone. The app we implement will generate a continuous 500mV peak to peak audio waveform only onto the right audio line. Using this audio line and this circuit we are able to generate a 1.8V source for our processor. Key points of the transformation of the 400mV peak to peak audio waveform through the circuit in Figure 2 were captured and illustrated in Figure 3.

The yellow signal in Figure 3 is the raw waveform coming out off the smartphone. The first step in trying to use this audio frequency is to step up the low supply voltage level. Using a 1:20 micro transformer we step up the incoming supply voltage level as shown by the red signal in Figure 3. Normally the next step would be to use a Schottky diode to preform low-loss blocking. However using a combination of N and P Mosfets depicted in Figure 2, designed by Hijack, we can preform an FET-based rectification from AC current to DC. This would allow us to be able to use the negative part of our input waveform to generate power as well. The spikes on the low edge of the red signal in Figure 3 are generated by the FET bridge that would have otherwise been clipped out, optimizing our power generating capability. Using the Schottky diode to provide low-loss blocking, will prevent the capacitor bank from being discharged through the FET bridge. The output at this point is illustrated by the Blue signal in Figure 3. The final step is to use a 1.8 voltage regulator to power our processor, the Green signal in Figure 3.

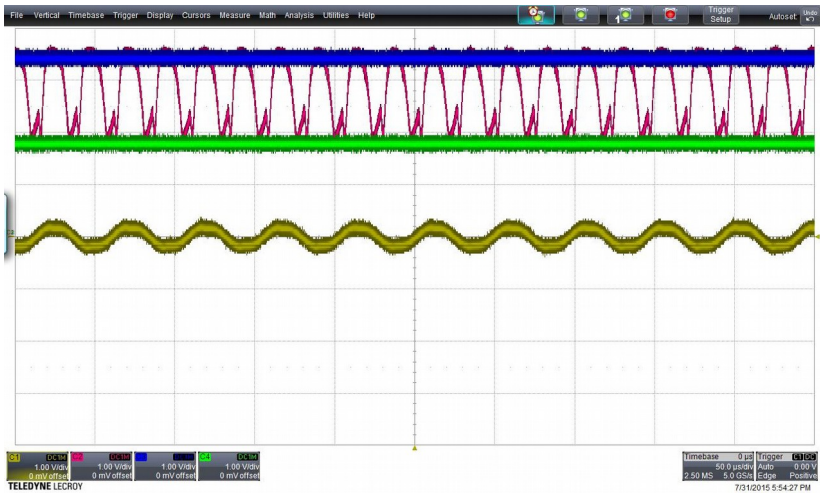

Figure 3: Power Circuit Analysis

Color	Position	Description
Yellow	before T1	original raw audio waveform
Red	before D1	after the FET bridge
Blue	after D1	after the Schottky diode
Green	after U1	after the voltage regulator

Table 1: Power Circuit Analysis Key

Our goal was to allowed our processor to be able to perform more power intensive cryptographic computations. Without using a battery and while staying within the limitations of our smartphone power generating capability. Therefore using this power siphoning circuit and a series of capacitors, that could be discharging and recharged simply by going into sleep mode, we are capable of allowing our device to remain charged for at least one round of AES.

4.2 Capacitor Bank

Not all cellphones can generate the same amount of power through the axillary port. Ever smartphone model will have its own power draw model and ideal matched load. To aid with power management we implemented a capacitor bank that will aid us in granting enough power for one round of 128-bit AES. As our processor implementations continue to evolve we will continue to optimize our power consumption. However, different cryptographic operations may consume more power than our current 128-bit AES implementation.

The capacitor bank will provided enough power for one round of 128-bit AES. That means it needs to contain enough of a charge between the time our sleep circuit switches between its wake and sleep state. Knowing that our sleep circuit activates the processors wake state at 1.95V and sleep state at 1.65V we can calculate the necessary capacitor bank we would need for one round of 128-bit AES, if we know the current consumption and amount of time it takes. As shown later in Table 5 and Table 6 AES consumes approximately 555µA and takes 1.4ms. Therefore we would need a 2.8µF capacitor bank to provided sufficient power for 1 round of AES.

Currently our capacitor bank was added after our 1.8 voltage regulator. This was done initially because our measured voltage change where measured after our voltage regulator. For optimal efficiency the capacitor bank should be recalculated and inserted before the voltage regulator.

Although the capacitor bank was initially design for one round of 128-bit AES, the design still works with whatever implementation our processor is computing. As long as the power drops the sleep circuit will activate sleep mode allowing for the capacitor bank to recharge. Such functionally will be useful with any smartphones as the bank will aid in allowing for more continuous power.

4.3 Sleep Mode

To allow the capacitor bank to recharge as efficiently as possible, we had to use as little power as possible during sleep mode. In addition we would need a way of telling the processor to go to sleep and not to wake up until the capacitor bank was adequately recharged. Otherwise our capacitor bank would be depleted to quickly, therefore any internal processor functionality would not do. For this we built a sleep circuit that would tell the processor through and external interrupt when to go to sleep, and

when to wake up. This circuit would also have to consume minimal power as to not draw too much power while in sleep mode, nor to deplete the processors power while active. For this we developed the sleep circuit illustrated in Figure 4.

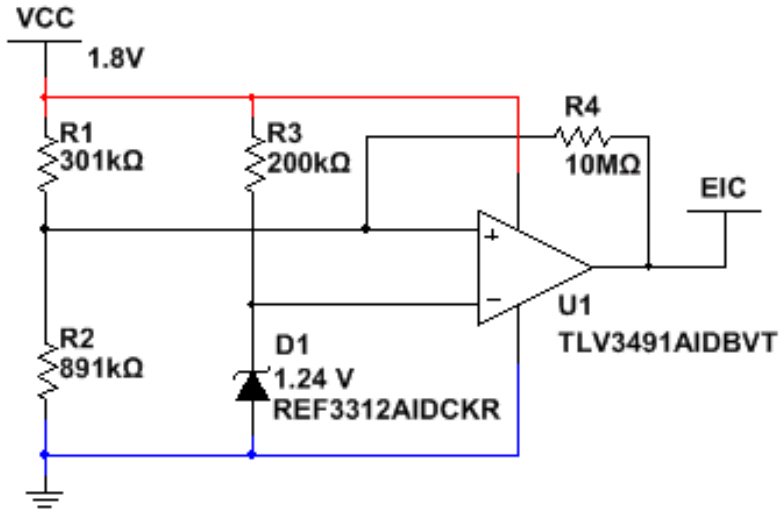

Figure 4: Sleep Circuit

Our sleep circuit uses a total of four 1% tolerance resistors, one low power voltage ref, and one low power comparator. This circuit has minimal power consumption and produces one output signal from the comparator. This signal is driven either high or low, and can be used as a digital input. When connected to the external interrupt controller of our processor this signal can be used to notifying our processor when to go to sleep, low for optimal power performance, and when to wake up.

Atmel's AVR UC3-L0 has multiple interrupts and external Interrupts. We were looking for a way to achieve the lowest power consumption in sleep mode, to allow our power capacitor bank to recharge as optimal as possible. The UC3-L0 has a total of seven sleep modes. Only the two highest power consumption sleep modes can wake up from a synchronous source, ruling out using an internal interrupt routine. However all sleep modes, except for shutdown mode can be achieved with an asynchronous source. Connecting the external signal generated from our sleep circuit to two external interrupt pin, we can use two external interrupt routines to go to sleep (low) and wake up (high). Both of these external sleep/wake interrupts have been given the highest priority and would supersede any other interrupt. This is to guarantee that when our capacitor bank is almost drained the processor will go to sleep before loosing power and allowing the capacitor bank to charge up again.

Our processor normally operates at 1.8V, but using its power thresholds our comparator will notify the processor to go to sleep at approximately 1.65V and to wake up at approximately 1.95V. Measuring the time it takes for one round of AES we modified our capacitor bank for ample time to complete before recharging.

4.4 Processor Power Mode

One of the many reasons we chose Atmel's low power AVR UC3-L0 is that it has three different power modes: 3.3V Single Supply Mode, 1.8V Single Supply Mode, and 3.3V supply Mode with 1.8V regulated I/O lines. For this design we used the 1.8V single supply mode with an external clock crystal to achieve minimal power consumption. The dual supply mode is the most optimal low power mode and the only way of achieving sleep shutdown mode, the lowest power consumption sleep mode. We chose to use the single supply mode for our design instead as it would ultimately be less of a drain on our siphoning circuit.

Chapter 5

5 Data Communication

The communication protocol we designed for our devices was modeled off of Hijack's communication protocol. The great thing about this design is that it does not require any analog to digital converters, which results in less power drain but slower communication speeds. The smartphone transmits in analog and the processor transmits in digital, but each communication is handled in the time domain. We are able to do this because the communication is handled in Manchester Encoding.

Manchester Encoding is normally a form of digital encoding where bits are represented by transitions from one logical state to the other, instead of being represented by a high or low signal. This means that instead of having to detect an analog high and low period, we now have to measure the periods between switches. This allows a smartphone that can only output and read audio oscillating waveforms to transmit and receive a waveform with Manchester encoded data.

For the processor, transmitting in Manchester encoding to the smartphone is straight forward. The processor can generate a square wave at any frequency, doubling the period for a bit flipping, that can simply be interpreted by a smartphone as an oscillating waveform. On the other hand, interpreting incoming data from a smartphone outputting an oscillating audio waveform instead of a digital signal to the processor is not as straightforward. The processor instead will have to determine the period lengths between the oscillations. Allowing the processor to determine if the incoming Manchester encoded data is flipping its bit, double the period, or not.

We have modified our communication protocol to work serially and in conjunction with the

overall process of the processor. Unlike Hijack, we are not recording sensor data and do not need a continuous transmission. We only transmit and receive data in bursts when necessary. This allows for more computation cycles and over all less power drain on our system. A flow chart of our current processor's AES implementation can be found in Figure 5.

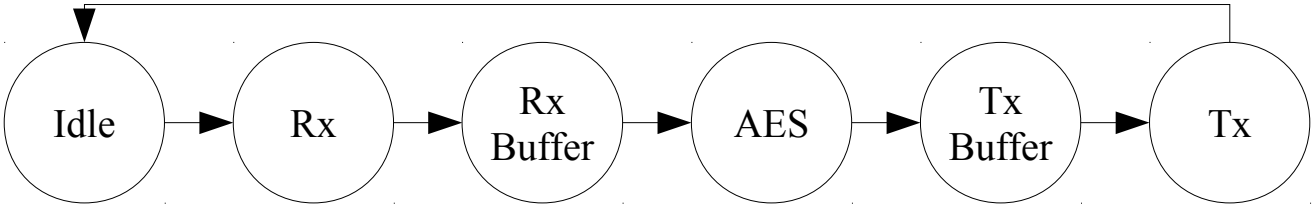

Figure 5: Processor Flow Chart

When in Idle our processor is continuously waiting to receive a transmission from the smartphone. Once data is received it triggers Hijack's communication state-machine that grabs the incoming data. The state-machine then converts the raw incoming Manchester data into binary data. This incoming data includes the instructions for our processor. Using the obtained data our processor then preforms it AES encryption/decryption. It then triggers the transmit state of the state-machine. The state-machine will then converting the raw data into Manchester encoded data and transmit this to the smartphone. The transmission burst is repeated four times by our processor, after which it returns to Idle waiting to receive further instruction from the smartphone.

Currently we are transmitting at approximately an average of 0.9kbps. Our current transmission speeds are unfortunately severely slower than normal digital communication. This is mainly due to the fact that our data is transmitted in Manchester Encoding that doubles the number of cycles. In addition, there are delays that are added through the use of Hijack's communication state-machine. In our next iteration we will be redesigning our own communication state-machine that will be optimized for our design implementation.

5.1 Manchester Encoding

Manchester Encoding is a form of digital encoding in which data bits are represented by transitions from one logical state to the other. The encoding of digital data in Manchester format

defines the binary states of a 1-bit and 0-bit to be transitions rather than static values. Manchester encoding was developed at the University of Manchester, where the coding was used to store data on the magnetic drum of a Manchester Mark 1 computer. There are two opposing conventions for Manchester encoding, both are used by numerous authors.

The first convention of Manchester Encoding was first published by G. E. Thomas in 1949. It stated that for a 0-bit the signal levels would be low-high and for a 1-bit the signal levels would be high-low. The second convention used by IEEE 802.4 and IEEE 802.3. It states that for a 0-bit the signal levels would be high-low and for a 1-bit the signal levels would be low-high. These two conventions are opposites of each other.

For our application we followed Hijack's implementation that follows the IEEE convention. Illustration examples of Manchester Encoded data can be found in Figures 6 through 9. These examples are illustrations of an implemented transmission using Hijack's communication protocol.

- Figure 6 is the header of the transmission.
- Figure 7 is the transmission byte length.
- Figure 8 and Figure 9 are two different data values that were transmitted.

We choose to only use two data values so that the illustration would be simpler to follow. The transmissions are handled one byte at a time. Each byte has a start bit of 0 and a parity bit at the end. The actual transmission streams of the processor and smartphone can be found in Figure 13 and Figure 17 respectively. The transmission captured match these illustration.

The reason we used Manchester Encoding is because Manchester Encoding ensures frequent line voltage transitions, which are directly proportional to their clock rate. Therefore, an oscillating audio waveform can be manipulated into representing Manchester Encoded data.

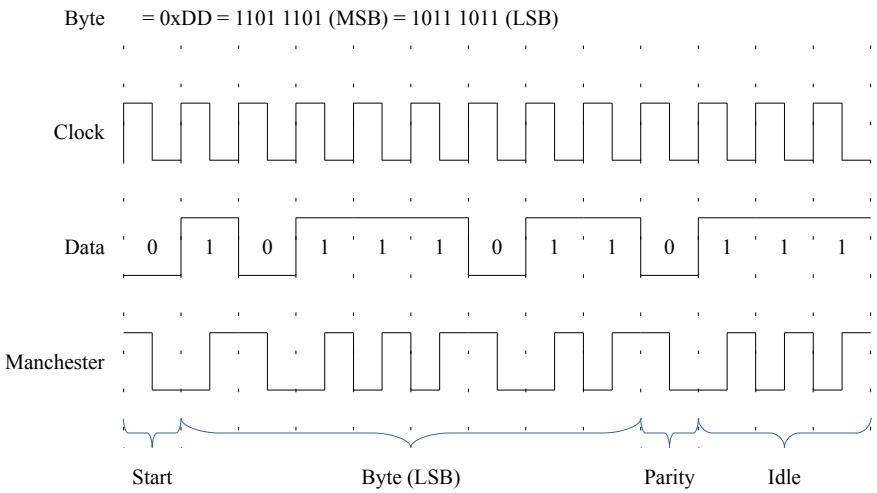

Figure 6: Manchester Encoding of header 0xDD

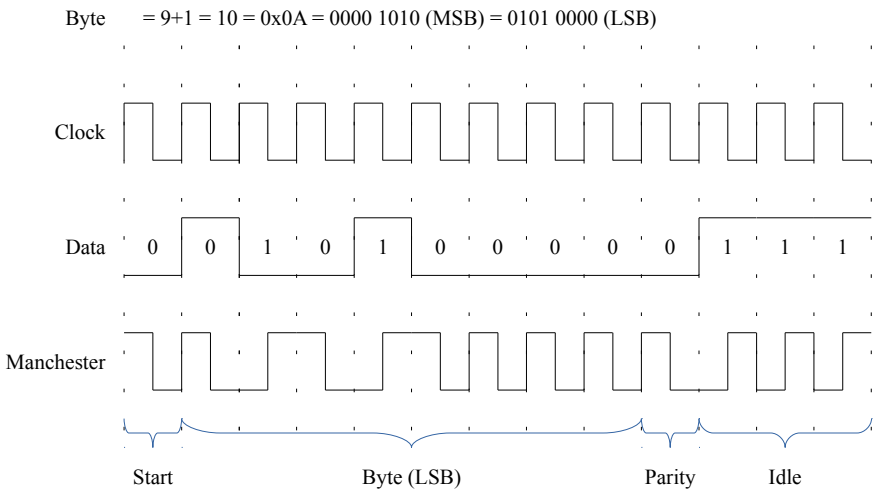

Figure 7: Manchester Encoding of length 0x0A

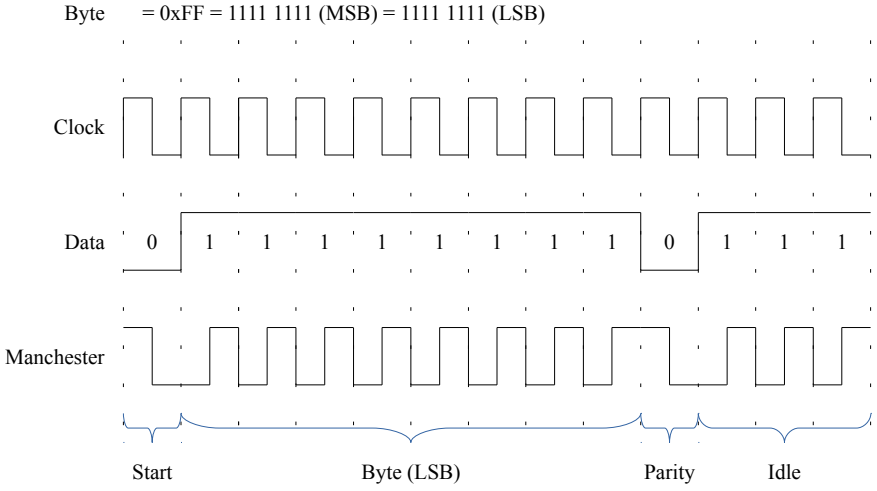

Figure 8: Manchester Encoding of data 0xFF

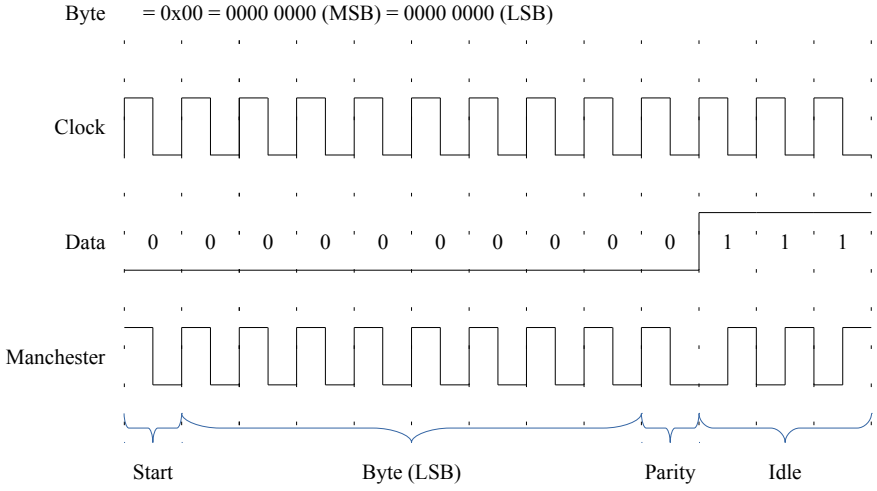

Figure 9: Manchester Encoding of data 0x00

5.2 Transmit

Data sent from the processor to the smartphone will be sent along the microphone channel of the auxiliary port. The communication protocols used for creating this audio waveform were modeled off of Hijack's communication protocol. The major difference being that when in this state we only use the transmit portion of Hijack state-machine, as shown in Figure 5. The protocol we implemented to activate the transmit portion of the state-machine is illustrated in Figure 10.

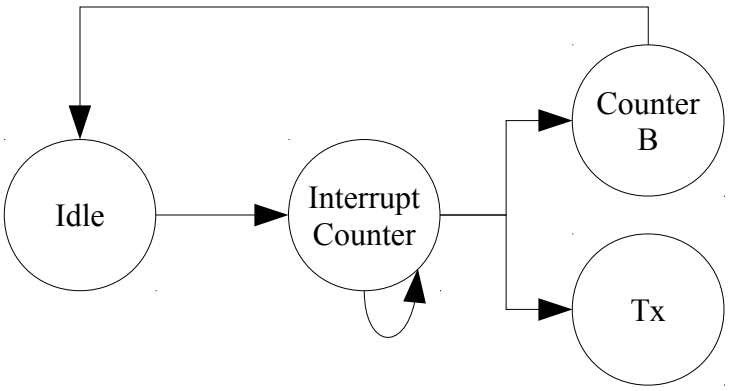

Figure 10: Processor Transmit Flow Chart

For our transmit protocol we used one of our processors built in synchronous counter and one asynchronous counter. The counters are only activated when the processor is in the transmit state (Tx), in Figure 5. When the counter is activated it will trigger an interrupt ever time it reaches a predetermined value. This value is the frequency of our generated signal, every time the interrupt is triggered is one clock cycle of our output stream. This event triggered interrupt will do two things: trigger the transmit state of the state-machine and increment a second asynchronous counter B. The second counter B is used to keep track of how many transmission have taken place, after four full transmissions the counter will exit the transmission stat and turn off the counter.

On the hardware level the only important thing with the transmission circuit is to have a 1k load resistor. The 1k Ohm resistor is necessary to activate the microphone channel on most smartphones, without it the smartphone would not know there was data coming in on the microphone channel. The processor will then output an oscillating square waveform approximately 800mv pk to pk. The transmission circuit we implemented is shown in Figure 11.

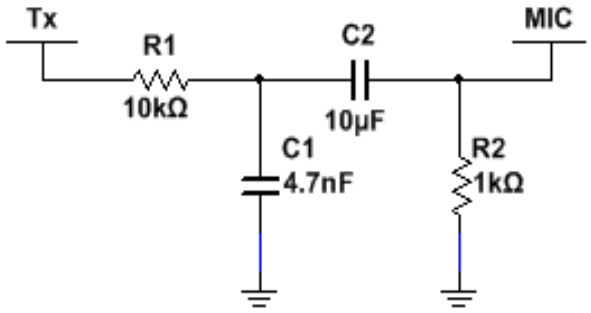

Figure 11: Transmit Circuit

An example of a transmission stream from our µLeech to our smartphone was captured and illustrated in Figure 13. Using the Manchester encoded byte examples in Figures 6 through 9 we can see the data captured in Figure 13. The communication protocol of the processor transmits in the format depicted in Table 2.

Byte	1	2	3	4	5	6	7	8	9	10	11	12
Content	Header	Length+1	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8	Check Sum
Value	0xDD	10	0xFF	0xFF	0x00	0xFF	0x00	0xFF	0x00	0xFF	0x00	-
Image	Figure 6	Figure 7	Figure 8	Figure 8	Figure 9	Figure 8	Figure 9	Figure 8	Figure 9	Figure 8	Figure 9	-

Table 2: Processor Transmission Burst

Each byte has a start bit of 0 and a parity bit at the end. This means each byte transmission is actually 10-bits which are then transmitted in LSB in Manchester encoding. However in between each byte transmission the state-machine goes into an idle state, where it continuous to transmit digital high for a fixed four delay count. Figure 12 depicts a zoomed in view of the Manchester encoded captured data burst shown in Figure 13.

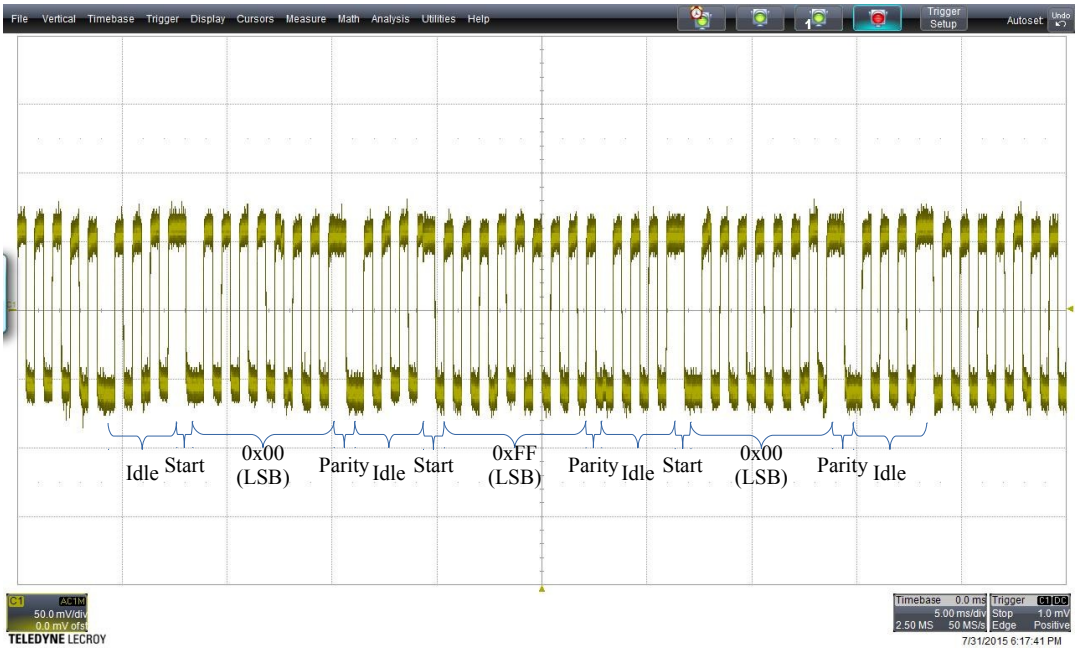

Figure 12: Processor Transmission Zoomed

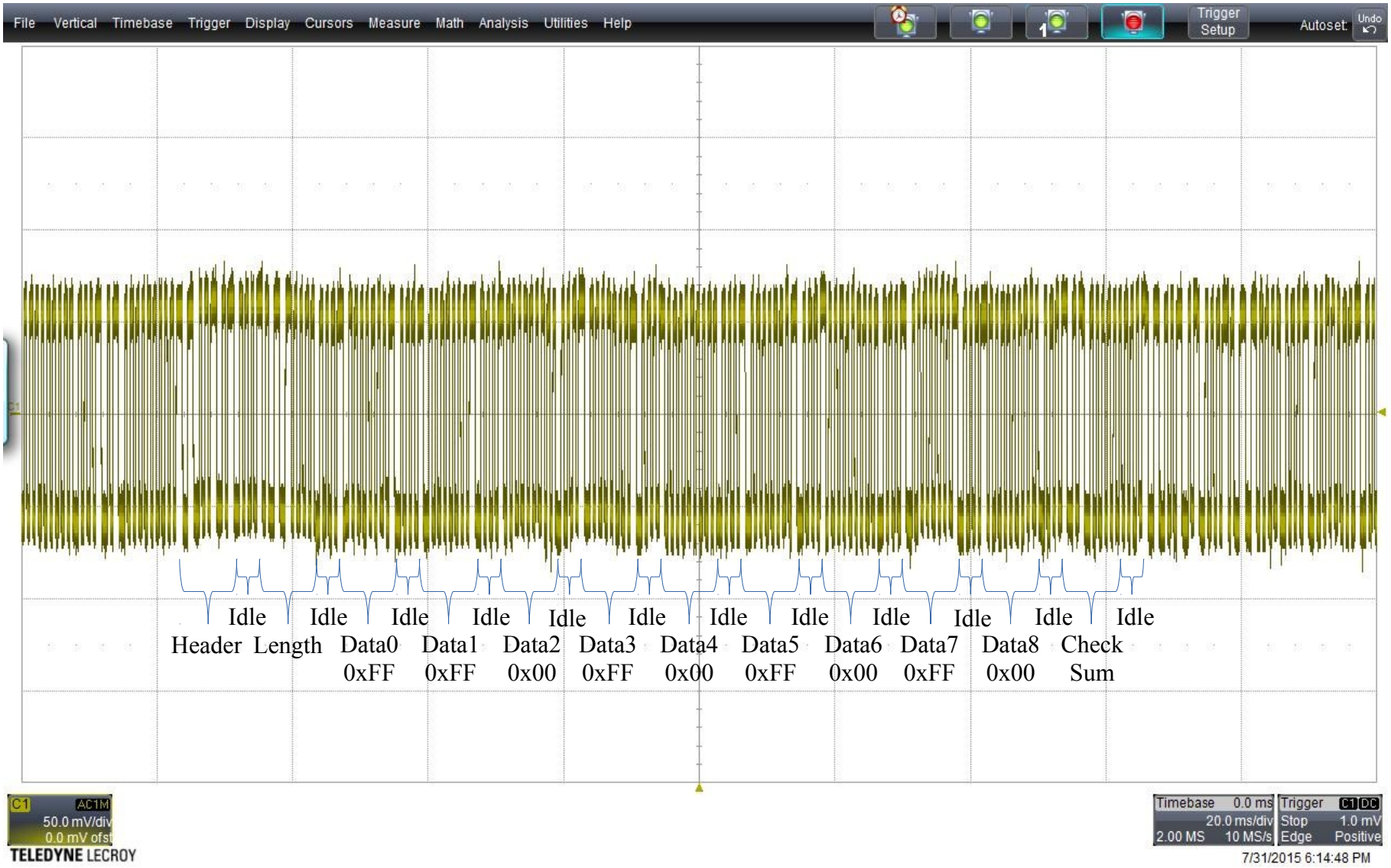

Figure 13: Processor Transmission

5.3 Receive

Data sent from the smartphone to the processor will be sent along the left audio channel of the auxiliary port. The communication protocols used for interpreting this audio waveform were modeled off of Hijack's communication protocol. The major difference being that when in this state we only use the receive portion of Hijack state-machine, as shown in Figure 5. The protocol we implemented to activate the receive portion of the state-machine is illustrated in Figure 14.

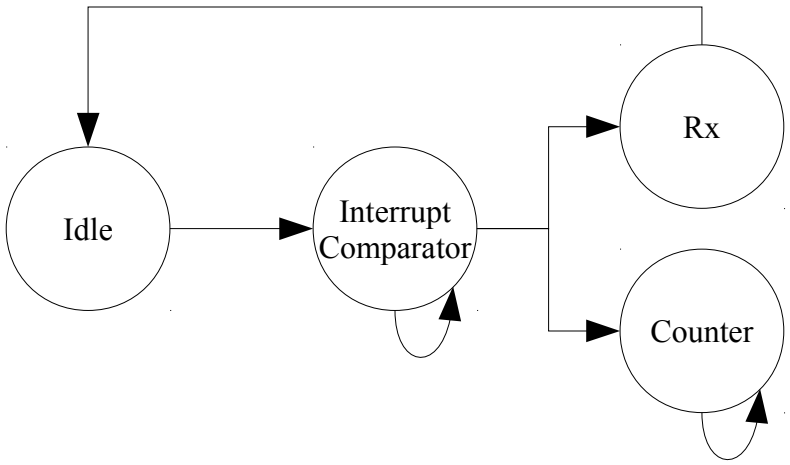

Figure 14: Processor Receive Flow Chart

For our receive protocol we used one of our processors built in synchronous comparator driven interrupt and counter. The comparator and counter are only activate when the processor is in the receive state (Rx), in Figure 5. When the comparator is activated it will trigger an interrupt ever time there is a bit flip on the incoming Manchester encoded data stream. This event triggered interrupt will do two things: trigger the receive state of the state-machine passing it the counter value and reset the counter. This will allow the state-machine to know the length of time between bit-flips. Since the incoming data is represented in Manchester encoding the state-machine will be able to determine if the previous bit was flipped or not. Allowing us to decode the income Manchester data into binary data.

What this means for our hardware, is that we have to feed the comparator in our processor two inputs. The oscillating audio waveform from the smartphone's left audio channel and a constant signal that is the mean voltage level of the oscillating audio wave, as shown in Figure 15.

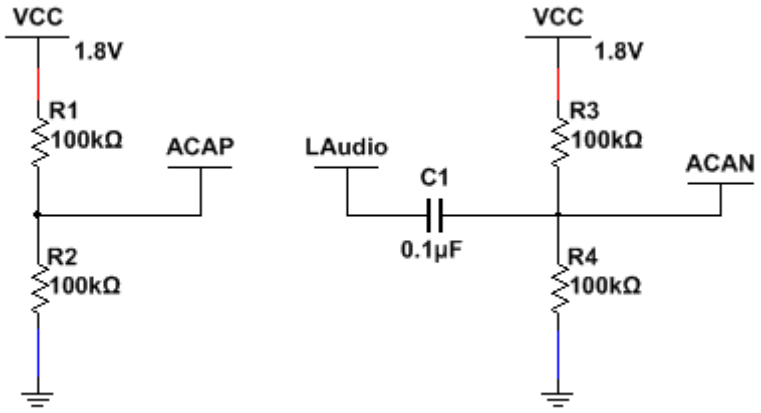

Figure 15: Receive Circuit

The smartphone will output an oscillating waveform approximately 900mV pk to pk with a lot of noise. The first step in our receive circuit is to drain out the noise with a capacitor, then using a pull up resistor and a pull down resistor we can generate an 800mV peak to peak oscillating at a mean of 800mV. However, when you take into account the additional pull down resistor built into the processor, on the input pin, the signal is oscillating at a mean of approximately 900mV. This means, for our comparator to trigger we need to feed it another constant 900mV signal. Since our processor is operating at 1.8V using a simple voltage divider on our voltage source with equal resistors we can divide the voltage level by half and generate a constant input signal of 900mV, with minimal power loss. This will allow our comparator to trigger exactly when our oscillating audio signal is switching from low to high, allowing us to retrieve the period length determining if the Manchester encoded audio waveform is bit flipping or not.

An example of a transmission stream from our smartphone to our µLeech was captured and illustrated in Figure 17. Using the Manchester encoded byte examples in Figures 6 through 9 we can see the data captured in Figure 17. The communication protocol of the smartphone transmits in the format depicted in Table 3.

Byte	1	2	3	4	5	6	7	8	9	10	11	12
Content	Header	Length+1	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8	Check Sum
Value	0xDD	10	0xFF	0xFF	0x00	0xFF	0x00	0xFF	0x00	0xFF	0x00	-
Image	Figure 6	Figure 7	Figure 8	Figure 8	Figure 9	Figure 8	Figure 9	Figure 8	Figure 9	Figure 8	Figure 9	-

Table 3: Smartphone Transmission Burst

Each byte has a start bit of 0 and a parity bit at the end. This means each byte transmission is actually 10-bits which are then transmitted in LSB in Manchester encoding. However in between each byte transmission the state-machine goes into an idle state, where it continuous to transmit digital high for a fixed twenty delay count. Figure 12 depicts a zoomed in view of the Manchester encoded captured data burst shown in Figure 13.

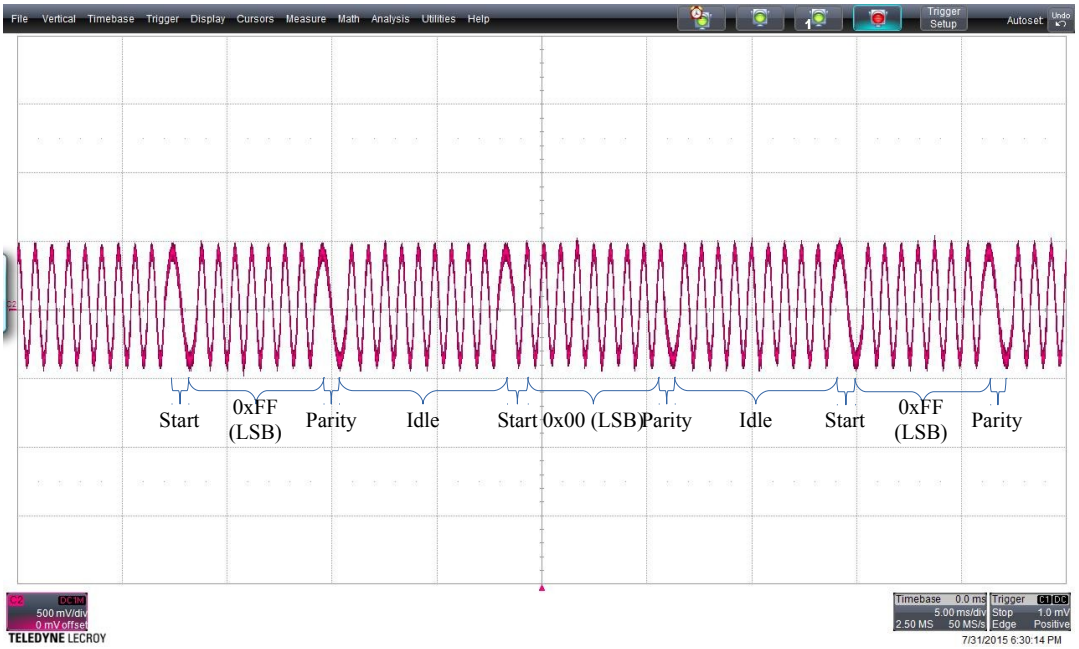

Figure 16: Smartphone Transmission Zoomed

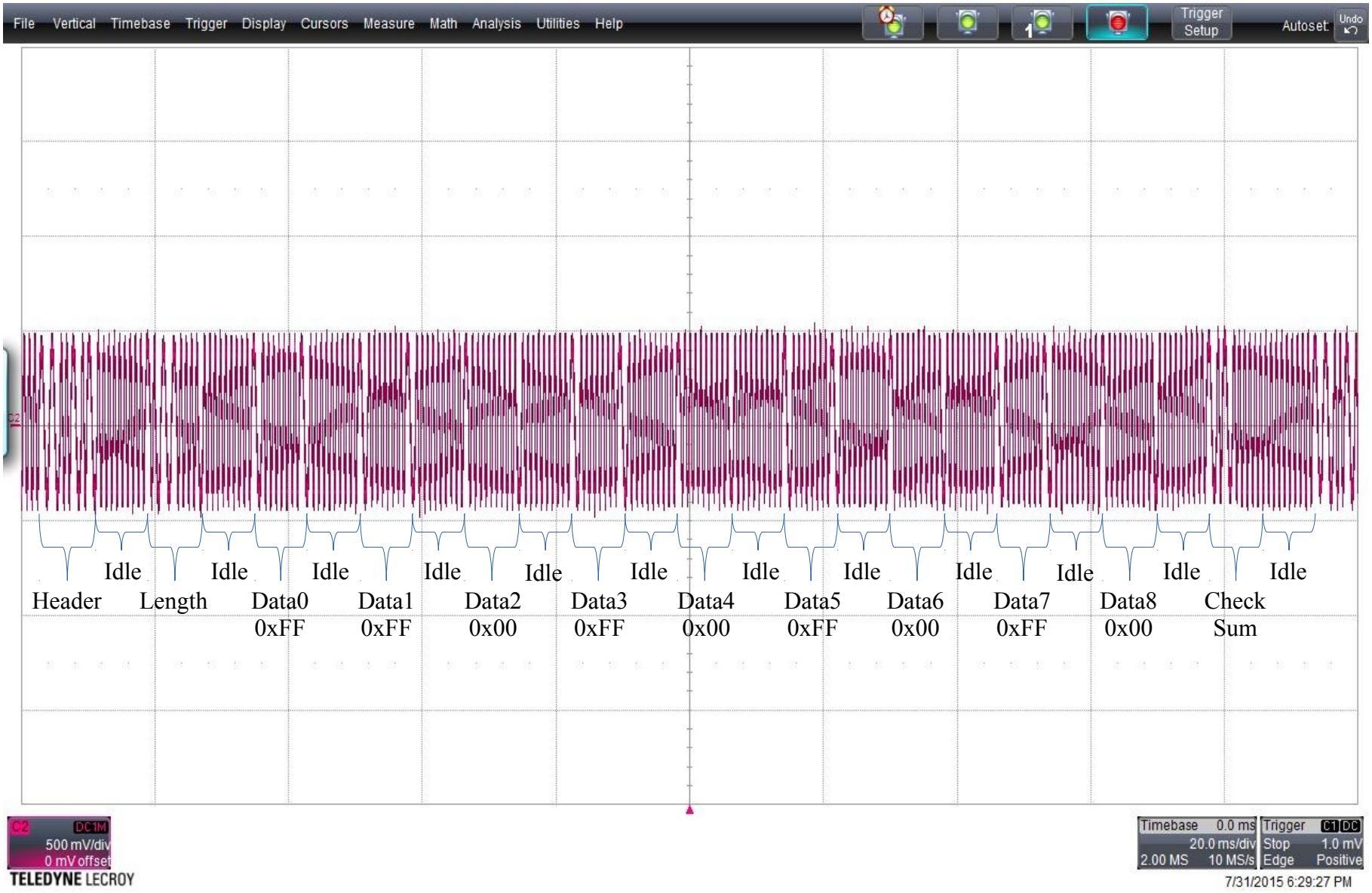

Figure 17: Smartphone Transmission

Chapter 6

6 Performance

The performance of our device is evaluated with a focus on power consumption. Our device is designed to operate solely off of the power generated from an audio waveform. As a result, we have made numerous modifications to optimize performance within our power generating capabilities. To optimize power we had to modify our capacitor bank to meet our designs power consumption, with a focus on allowing for one round of AES encryption before depletion. For this reason we measured power consumption of our different modes: idle, sleep, communication, and AES. Then we measured the duration of each of these modes, so that we could accurately estimate our capacitor bank.

6.1 Sleep Mode

As mentioned earlier in Section 5.2, there are many different sleep modes available to the UC3-L0. The ultimate sleep mode for minimal power consumption on the UC3-L0 would have been the sleep shutdown mode. In this mode wake up can only be triggered with an external reset or an external wake up pin. Although we are generating an external wake up signal with our sleep circuit, the shutdown sleep mode can only be used in the 3.3V supply mode with 1.8V regulated I/O lines. However, our setup uses the 1.8V single supply mode and therefore we cannot use the shutdown sleep mode.

With the shutdown sleep mode we would have consumed the least amount of power in sleep mode. However, a single supply mode is less of a constant power drain on the siphoning circuit and

therefore the whole system, than a dual supply mode. Table 4, shows the power consumption of the lowest three power consumption sleep modes available for single supply mode.

Sleep Mode	Power Consumption
Stop	92.6 µA
DeepStop	68.3 µA
Static	56.7 µA

Table 4: Power Consumption in Sleep Modes

The main differences between these three sleep modes are which clock sources are left enabled. For this reason our design uses an external clock crystal that allows us to use DeepStop and Static sleep modes. Static mode requires us to reinitialize some of our clock signal, and therefore the modules using them, on every wake up. For this reason currently our design uses DeepStop, which is less of a power drain on the overall system for every wake up cycle.

6.2 Active Mode

Hijack's smart phone platform is about continuous transmission of sensor data. Our communication data is based off of theirs but we are more interested in transmitting and receiving short bursts of data and doing high levels of short computations. For this reason we have modified our communication protocol, added a capacitor bank, and a sleep circuit. To that end we have tired to optimize our power consumption. Our processor only checks for incoming data when it is idling and waiting for instructions. Computations are only executed when the processor receives new incoming data or new instruction sets. The processor transmissions are executed only when there is a change in the outgoing data, computations have been completed. The transmissions are in short bursts repeated four times for redundancy error checking. The measured power consumption of these processor states is listed in Table 5.

Processor State	Power Consumption
Idle	539 µA
Receiving Data Burst	588 µA
Transmitting Data Burst	575 µA
128-bit AES Encryption	555 µA
128-bit AES Decryption	555 µA

Table 5: Power Consumption in Active Modes

As illustrated in Table 5, our designs highest power consumption currently is in our communication protocol. For this reason we implemented burst communication. Each data burst has its own header that includes the number of bytes of data being transmitted. Our app and processor can therefore read as many data bytes included in the communication burst. Therefore our communication burst will vary in length and time, optimizing our power consumption even further. Currently we measured an average data transmission of approximately 900bps or 0.9kbps.

6.3 128-bit AES

Atmel's AVR UC3-L0 was chosen for its low power consumption and its computation capacity. However, this chip has no additional built in cryptographic modules. 128-bit AES was software implemented with no additional optimizations and tested for power consumption. The power consumption of 128-bit AES encryption and decryption is shown in Table 5. The performance of the same 128-bit AES implementation is shown in Table 6.

128-bit AES	Clock Frequency	Time
Encryption	1 MHz	1.378 ms
Decryption	1 MHz	1.400 ms

Table 6: 128-bit AES Performance

The measured performances of our processor illustrated in Table 6 are operating at a clock speed of 1MHz. This clock is generated using an external 1MHz clock crystal. Modifying this crystal we can achieve higher performance implementations of AES, but in the process increase our power consumption.

One of our design goals was to allow for at least one round of AES to be completed before depleting power. Using the measure power consumption, Table 5, and the measure time one round of AES took, Table 6, we were able to modify our capacitor bank. Our capacitor bank will allow for at least one round of AES to be implemented before our sleep circuit will notifying the processor to go to sleep.

Chapter 7

7 Development

The development of µLeech is the culmination of all the information included in this thesis. For our hardware there are two final products to be developed: µLeech and µLeech Evaluation Platform. In terms of software, there is the processor firmware and the android application to be developed. Both hardware iterations will use the same processor firmware and communicate with the same smartphone app.

7.1 Smartphone App

The smartphone used during the development of µLeech was an android Nexus 4. The app we developed only runs on android platforms. Currently, there is no iPhone app for µLeech. Development of an iPhone app is a possibility, since the hardware will work with any smartphone. We decided to do our initial app iteration on android platforms since development is free and coding is simpler. Now that we have settle on a design and implementation we can develop a µLeech iPhone app.

It is important that the app we developed uses the same communication protocols as our µLeech processor and therefore Hijack, as explained in Chapter 5. Hijack's app was originally designed for iPhones, therefore written in Objective C, and included a library for the communication protocol. Hijack-Infinity created by Anro Robinson is an android implementation of Hijack's app, written in java. The Hijack libraries were converted to java to be used in Hijack-Infinity. Using the converted libraries as our communication protocol framework, we developed our own source code and GUI that would

interface with our processor. Hijack's libraries allowed us to power, transmit, and receive data. These libraries include the same state-machine and frame engine that preformed Manchester encoding on the processor.

The main app we used in the development of µLeech is show in Figure 18. This app has full functionality: it powers µLeech with a continuous oscillating waveform on the right audio channel, transmits data on the left audio channel, and receives data on the microphone channel.

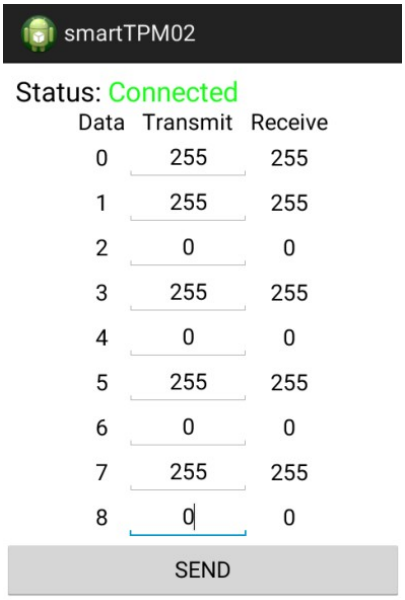

Figure 18: Android App

The app transmission is a continuous transmission of nine bytes that repeat. The nine bytes are each different integer inputs that the user can input in the transmit column. The app's continuous transmission is only updated when the send button is pushed. Eventually, like the processor the app can be modified to do a burst transmissions when send is pushed. Inputs from the processor that come on the microphone channel are displayed in the receive column. This column is only updated when a burst transmission is received and verified, when this happens the status indicator will switch to green and display connected.

As our processor implementations continues to evolve so will the app, reflecting the functionality needed.

7.2 Hardware

The hardware was a constantly changing design. As we our understanding and implementation evolved so did our hardware implementation. The hardware underwent three major development stage: protoboarding, power optimization, and final PCB iterations. Initial PCBs were developed using ExpressPCB with the final product developed using CadSoft Eagle. The object was two final products: µLeech and µLeech Evaluation Platform.

7.2.1 Protoboarding

Before we could manufacture or even create the final PCB iterations we had to first create and prototype the design. The prototype was initially bread-boarded. This allowed us to build the prototype in steps and to modify the circuits as we needed to. Since nothing is soldered modifications can be continuously made to the design. At this point we designed and implemented the processor, power siphoning, sleep, transmit, and receive circuits discussed in chapters 4 and 5.

Not all of the components we used could be bread-boarded. For example the biggest problem we had was our AVR processor, a 48-pin surface mount (SMT) component, that could not be bread-boarded directly. In such cases a SMT bread-boarding socket is use, however not all SMT components have bread-boarding sockets available for them. Most of the embedded chips we used were low power SMT components that had no sockets available to them. As a result we manufactured our own SMT breadboard sockets. Figure 19 is the bread-boarding socket we developed for our processor, using ExpressPCB. With this PCB and others like it we were able to bread board all the parts of our design. Allowing us to test our design before fabricating the final design on a PCB.

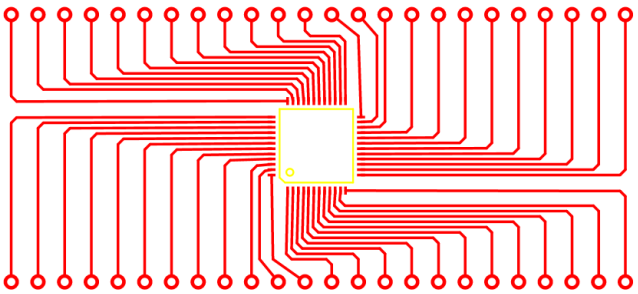

Figure 19: UC3-l0 bread-board socket

7.2.2 Power Optimization

During the course of the protoboarding development before we design the final PCB iteration we had one major PCB implementation iteration. This PCB iteration used all the same components that would be included in the final design. The main goal of this iteration was to verify our power measurements and consumptions. In addition this iteration verified that our design and components were working as intended. The power measurements of this PCB iteration are the measurements used in Chapter 6: tables 4, 5, and 6.

This PCB iteration was implemented using ExpressPCB, it includes most of our final design except for the transmit and receive circuits. This PCB iteration produced two modules. One of the modules included the power circuit and capacitor bank circuit: Figure 20 and Figure 21. The other was the processor circuit and sleep circuit: Figure 22 and Figure 23. These modules were split up so that they could be tested independently without affecting each other. After which, connecting the jumper headers in the bottom right corner of each modulo allowed us to test all four circuits concurrently. Verifying that our current implementation worked and measuring power consumption.

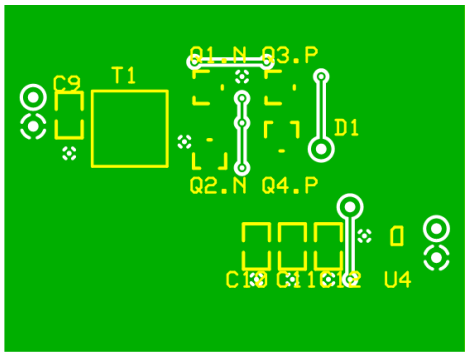

Figure 20: Power generation PCB Bottom

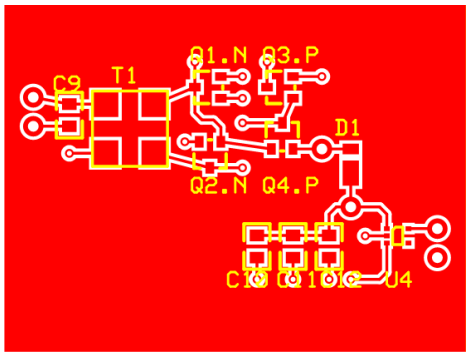

Figure 21: Power generation PCB Top

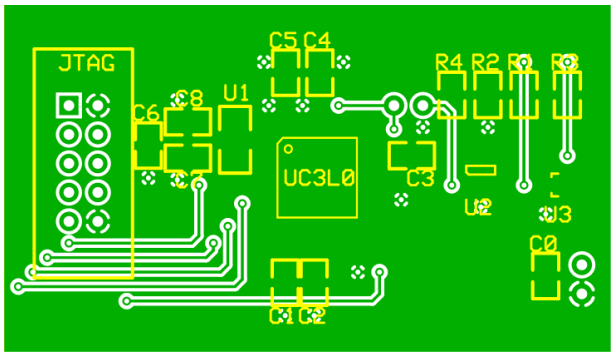

Figure 22: Power Evaluation PCB Bottom

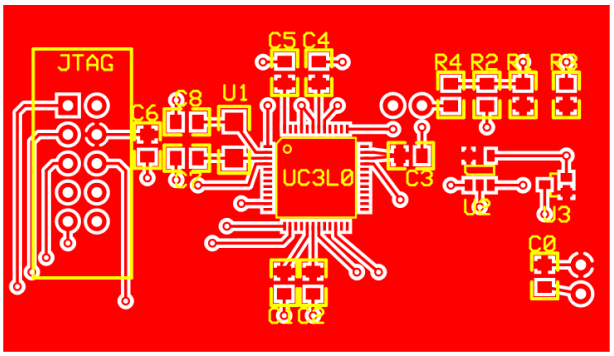

Figure 23: Power Evaluation PCB Top

7.2.3 Final PCB

Once the hardware design was completed and tested thoroughly, the whole design was implemented using CadSoft Eagle. Eagle was used to develop the two final PCB versions of µLeech. Eagle has a much more extensive library of component footprints for SMT layouts. However, many of the components used in our design had costume made footprints or were downloaded from the user based library. The final µLeech PCB versions can be found in Figure 24 and Figure 25. The next iteration will include the µLeech Evaluation Platform, which will include high quality on-board peripherals to our power and clock signals.

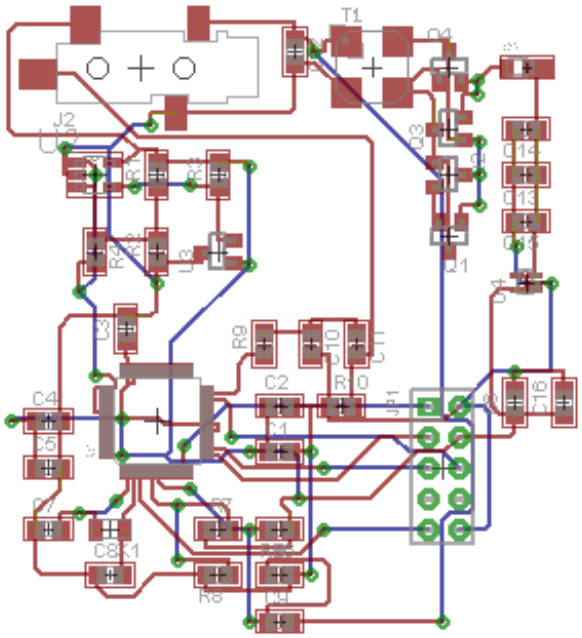

Figure 24: µLeech Board

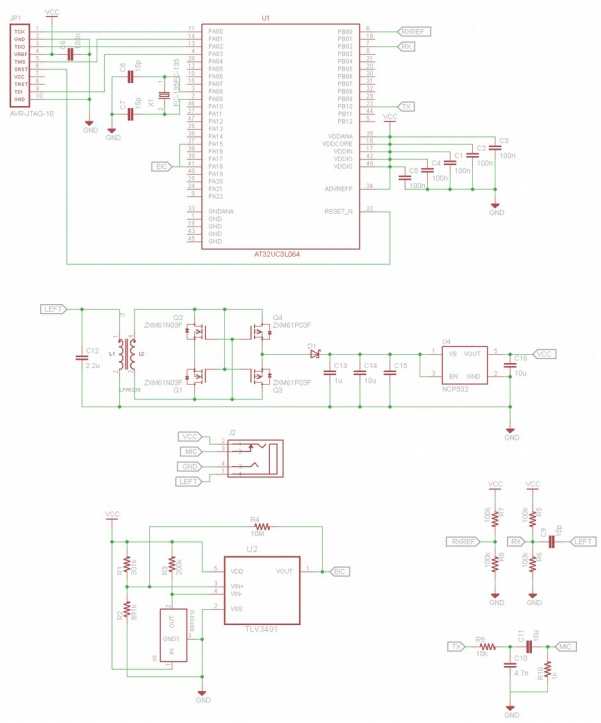

Figure 25: µLeech PCB Circuit

Chapter 8

8 Conclusion

µLeech is a fully functioning independent platform that can be used with any auxiliary port. This chapter concludes the thesis and the initial development of µLeech. It presents a summary of the current results and some recommendations for future work and development. The development and prototyping of µLeech has reveled more optimal implementations and current bottle necks of our design. Our current prototype is the platform on which we will continue to improve µLeech.

8.1 Summary of Results

µLeech is a low power platform on which to run cryptographic operations. As such, its performance is mainly determined in power consumption. The less power it consumes the less power it ultimately needs to drain from your smartphone. The power consumption of µLeech was evaluated by looking at different sleep and active implementation sates.

The current power consumption sleep implementations of µLeech are available in Table 4. This table shows that currently the lowest power consuming sleep state is Static Sleep at 56.7 µA. Currently we are using Deep Stop Sleep which consumes 68.3 µA. Using Static Sleep would require us to reset our interrupts, counter, and comparator clocks. As a result there would be a clock initialization phase at the start of ever wake cycle, ultimately consuming more power. In addition, depending on when the sleep state was activated an event reset may be required at wake, due to the clock reset, resulting in wasted cycles/power. For these reasons we are currently using Deep Stop Sleep, the second lowest

power consuming sleep state available. With the prototype completed now, further testing of the different wake protocols may show which sleep state is ultimately more efficient.

The current power consumption active implementations of µLeech are available in Table 5. This table shows that currently the highest power consuming phase of our µLeech is its communication phase. Transmitting and receiving data is consuming 575 µA and 588 µA respectively. While 128-bit AES is consuming 555 µA. This result shows that there are two things that need to be researched further, to optimize our design. First, can we achieve lower power consumption from our communication protocol implementation. Second, how far can we push our AES implementation while keeping an eye on our power consumption and capacitor bank. Possible allowing us to increasing our CPU clock speed, currently 1 MHz.

8.2 Recommendations for future work

Every hardware design iteration shows possible improvements for the next design iteration cycle. With the completion of our µLeech prototype, the development process has exposed several firmware and hardware recommendations for the next iteration of µLeech.

Currently the biggest bottle neck in our design is the implementation of Hijack's communication protocols. Implementing these protocols were necessary initial to develop a working platform. With this platform and a working understanding of their protocol, a more efficient protocol can be developed for our application. Hijack and µLeech transmit data in two very different ways. Hijack is a continuous transmission of sensor data that is constantly being updated. µLeech is a burst transmission of predetermined data, plain text, cipher text or keys. Therefore, a more efficient protocol for our device can be implemented. This would replace the Hijack's libraries we use on the smartphone app and Hijack's state-machine we use on the processor. In doing so, our communication protocol could:

- **Only need 1 Idle cycle.** At most we only need 1 Idle cycle between byte transmissions. Using Hijack's protocol there are four idle cycles for our processor and twenty idle cycles for our smartphone between every byte transmission. Ultimately improving transmission speeds.
- **Possibly attain higher baud rates.** Our data is predetermined at the start of the transmission, therefore the buffer can be loaded entirely to optimize speed. We could achieve higher baud rates, but the limitation will be the smartphone frequency modulation.

- **Not need a state-machine.** The communication protocols can be moved to the interrupt level entirely. We do not need a state machine to determine when to go to idle, transmit, receive, and when to build the transmit or receive buffers. Our communication occurs in a burst serial sequence. Therefore receiving and transmitting are a serial process and can never happen at the same time. In addition, the Manchester encoded buffers can be preloaded since the data is fixed when transmitting.
- **Not need a counter for our receive protocol.** The counter used to determine the space between comparisons when receiving data can be removed. Our AVR comparator is a synchronous comparator and can be used to decode the Manchester input directly, without using a counter. This would improve incoming data speeds and reduce power consumption.
- **Allow our smartphone app to transmit in bursts.** We have already modified our processor to only transmit and receive when necessary. Currently our smartphone app uses Hijacks converted java libraries for its communication protocol. Similarly, we could have our smartphone continuously transmit on the power channel but only transmit and receive data when necessary and in bursts. Optimizing smartphone power usage.

Such a communication protocol would allow for a much more efficient implementation protocol for μLeech. However like Hijack, we would still be using Manchester encoding. There maybe other methods to achieving higher baud rates for our communication protocol that do not involve Manchester encoding. Currently Manchester encoding doubles the clock rate since ever bit is represent by two bits. The reason for using Manchester encoding is because our smartphone is transmitting a continuously oscillating waveform. Therefore there is no high and low, Manchester encoding allows use to transmit by affecting the frequency without affecting the amplitude of the waveform. Other options may include modifying the amplitude of the waveform without affecting its frequency, such as volume control. If we could create a communication protocol that does not use Manchester encoding we could at the very least double our baud rate.

Appendix A

A. Processor Code: main.c

```
/*
 * Header files
 */
#include "asf.h"
#include "conf_board.h"
#include "framingEngine.h"
#include "codingStateMachine.h"
#include <aes.h>

/*
 * Method Declarations
 */
void initializeSystem(void);
void packetReceivedCallback(uint8_t * buf, uint8_t len);
void packetSentCallback(void);
void periodicTimerFn (void);
void captureTimerFn(uint16_t elapsedTime, uint8_t isHigh);



/*
 * Global variables to Store the module address
 */
volatile avr32_acifb_t *acifb = &AVR32_ACIFB; //volatile originally
volatile avr32_tc_t    *tx_tc = TX_TC;        //volatile originally
volatile avr32_tc_t    *rx_tc = RX_TC;        //volatile originally
```

```
/*
 * AES Global Variables
 */
unsigned char key[16];
unsigned char plain[16];
unsigned char cipher[16];
unsigned char result[16];
aes_encrypt_ctx enccx[1];
aes_decrypt_ctx deccx[1];

/*
 * Global Variables
 */
eic_options_t eic_options[2];
uint16_t tx_count = 0;
bool tx_update = false;
uint8_t outMessage[] = {0xFF, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00};

/*
 * EIC Interrupt Routine
 *  - Sleep Interrupt
 */
__attribute__((__interrupt__))
static void int_handler_SLEEP(void){
    eic_clear_interrupt_line(&AVR32_EIC, EXT_INT_SLEEP_LINE);
    //SLEEP(AVR32_PM_SMODE_STOP);
    SLEEP(AVR32_PM_SMODE_DEEPSTOP);
    //SLEEP(AVR32_PM_SMODE_STATIC);
}

/*
 * EIC Interrupt Routine
 *  - Wake Interrupt
 */
__attribute__((__interrupt__))
static void int_handler_WAKE(void){
    eic_clear_interrupt_line(&AVR32_EIC, EXT_INT_WAKE_LINE);
}

/*
 * ACIFB Interrupt Routine
 *  - Counter Comparator Interrupt
```

```
 */
__attribute__((__interrupt__))
static void tx_tc_irq(void){
    // Increment counter
    tx_count++;

    // Clear the interrupt flag.
    //  - This is a side effect of reading the TC SR.
    tc_read_sr(TX_TC, TX_TC_CHANNEL);

    //If transmission burst is completed
    // - stop transmitting
    // - startup comparator/receive interrupt
    if (tx_count>750){
        tx_update=false;
        tc_stop(tx_tc, TX_TC_CHANNEL);
        tc_start(rx_tc, RX_TC_CHANNEL);
        acifb_enable_interrupt(acifb, AVR32_ACIFB_ACINT4_MASK);
        tc_software_trigger(rx_tc,RX_TC_CHANNEL);

    //Else update the transmission line
    }else
        periodicTimerFn();
}

/*
 * ACIFB Interrupt Routine
 *  - Comparator Interrupt
 */
__attribute__((__interrupt__))
static void int_handler_ACIFB(void){
    uint16_t timerCounter = tc_read_tc(rx_tc,RX_TC_CHANNEL);

    if(timerCounter > 5){
        //Reset Counter
        tc_software_trigger(rx_tc,RX_TC_CHANNEL);

        //High Trigger
        if (acifb->sr & AVR32_ACIFB_SR_ACCS4_MASK){
            captureTimerFn(timerCounter, 1);
        //Low Trigger
        }else if (!(acifb->sr & AVR32_ACIFB_SR_ACCS4_MASK)){
```

```
            captureTimerFn(timerCounter, 0);
        }
    }

    // Clear the ACIFB interrupt flag
    acifb->icr = AVR32_ACIFB_ACINT4_MASK; //Clear interrupt flag ACIFB_4
    acifb_clear_comparison_interrupt_flag(acifb, ACIFB_ACA_CHANNEL);
}

/**
 * \brief Transmit TC Initialization
 *
 * Initializes and start the TC module with the following:
 * - Counter in Up mode with automatic reset on RC compare match.
 * - fPBA/8 is used as clock source for TC
 * - Enables RC compare match interrupt
 *
 * \param tc Base address of the TC module
 */
static void tx_tc_init(volatile avr32_tc_t *tc)
{
    // Options for waveform generation.
    static const tc_waveform_opt_t waveform_opt = {
        // Channel selection.
        .channel  = TX_TC_CHANNEL,
        // Software trigger effect on TIOB.
        .bswtrg   = TC_EVT_EFFECT_NOOP,
        // External event effect on TIOB.
        .beevt    = TC_EVT_EFFECT_NOOP,
        // RC compare effect on TIOB.
        .bcpc     = TC_EVT_EFFECT_NOOP,
        // RB compare effect on TIOB.
        .bcpb     = TC_EVT_EFFECT_NOOP,
        // Software trigger effect on TIOA.
        .aswtrg   = TC_EVT_EFFECT_NOOP,
        // External event effect on TIOA.
        .aeevt    = TC_EVT_EFFECT_NOOP,
        // RC compare effect on TIOA.
        .acpc     = TC_EVT_EFFECT_NOOP,
        /*
         * RA compare effect on TIOA.
         * (other possibilities are none, set and clear).
```

```
     */
    .acpa     = TC_EVT_EFFECT_NOOP,
    /*
     * Waveform selection: Up mode with automatic trigger(reset)
     * on RC compare.
     */
    .wavsel   = TC_WAVEFORM_SEL_UP_MODE_RC_TRIGGER,
    // External event trigger enable.
    .enetrg   = false,
    // External event selection.
    .eevt     = 0,
    // External event edge selection.
    .eevtedg  = TC_SEL_NO_EDGE,
    // Counter disable when RC compare.
    .cpcdis   = false,
    // Counter clock stopped with RC compare.
    .cpcstop  = false,
    // Burst signal selection.
    .burst    = false,
    // Clock inversion.
    .clki     = false,
    // Internal source clock 3, connected to fPBA / 8.
    .tcclks   = TC_CLOCK_SOURCE_TC3
};

// Options for enabling TC interrupts
static const tc_interrupt_t tc_interrupt = {
    .etrgs = 0,
    .ldrbs = 0,
    .ldras = 0,
    .cpcs  = 1, // Enable interrupt on RC compare alone
    .cpbs  = 0,
    .cpas  = 0,
    .lovrs = 0,
    .covfs = 0
};
// Initialize the timer/counter.
tc_init_waveform(tx_tc, &waveform_opt);

//Set the compare triggers.
tc_write_rc(tx_tc, TX_TC_CHANNEL, BOARD_OSC0_HZ/8/2320);
// configure the timer interrupt
```

```
    tc_configure_interrupts(tx_tc, TX_TC_CHANNEL, &tc_interrupt);
}

/**
 * \brief Receive TC Initialization
 *
 * Initializes and start the TC module with the following:
 * - Counter in Up mode.
 * - fPBA/32 is used as clock source for TC
 *
 * \param tc Base address of the TC module
 */
static void rx_tc_init(volatile avr32_tc_t *tc)
{
    // Options for waveform generation.
    static const tc_waveform_opt_t waveform_opt = {
        // Channel selection.
        .channel  = RX_TC_CHANNEL,
        // Software trigger effect on TIOB.
        .bswtrg   = TC_EVT_EFFECT_NOOP,
        // External event effect on TIOB.
        .beevt    = TC_EVT_EFFECT_NOOP,
        // RC compare effect on TIOB.
        .bcpc     = TC_EVT_EFFECT_NOOP,
        // RB compare effect on TIOB.
        .bcpb     = TC_EVT_EFFECT_NOOP,
        // Software trigger effect on TIOA.
        .aswtrg   = TC_EVT_EFFECT_NOOP,
        // External event effect on TIOA.
        .aeevt    = TC_EVT_EFFECT_NOOP,
        // RC compare effect on TIOA.
        .acpc     = TC_EVT_EFFECT_NOOP,
        /*
         * RA compare effect on TIOA.
         * (other possibilities are none, set and clear).
         */
        .acpa     = TC_EVT_EFFECT_NOOP,
        /*
         * Waveform selection: Up mode
         */
        .wavsel   = TC_WAVEFORM_SEL_UP_MODE,
        // External event trigger enable.
```

```
        .enetrg   = false,
        // External event selection.
        .eevt     = 0,
        // External event edge selection.
        .eevtedg  = TC_SEL_NO_EDGE,
        // Counter disable when RC compare.
        .cpcdis   = false,
        // Counter clock stopped with RC compare.
        .cpcstop  = false,
        // Burst signal selection.
        .burst    = false,
        // Clock inversion.
        .clki     = false,
        // Internal source clock 4, connected to fPBA / 32.
        .tcclks   = TC_CLOCK_SOURCE_TC4
    };

    // Options for enabling TC interrupts
    static const tc_interrupt_t tc_interrupt = {
        .etrgs = 0,
        .ldrbs = 0,
        .ldras = 0,
        .cpcs  = 0,
        .cpbs  = 0,
        .cpas  = 0,
        .lovrs = 0,
        .covfs = 0
    };
    // Initialize the timer/counter.
    tc_init_waveform(rx_tc, &waveform_opt);
}

/*
 * Initializes the ACIFB Comparator module
 * - Start the GCLK for ACIFB
 * - Initialize the trigger mode & compare interrupts for ACIFB
 */
static void acifb_init(void){
    genclk_enable_config(AVR32_SCIF_GCLK_ACIFB,GENCLK_SRC_CLK_CPU,40);

    /* GPIO pin/acifb - function map. */
    static const gpio_map_t ACIFB_GPIO_MAP = {
```

```
    {ACIFB_ACAP_PIN, ACIFB_ACAP_FUNCTION},
    {ACIFB_ACAN_PIN, ACIFB_ACAN_FUNCTION},
};

/* ACIFB Configuration */
const acifb_t acifb_opt = {
    .sut = 6, /* Resolution mode */
    .actest = TESTMODE_OFF,
    .eventen = false
};

/* ACIFB Channel Configuration */
const acifb_channel_t acifb_channel_opt = {
    /* Filter length */
    .filter_len = 0,
    /* Hysteresis value */
    .hysteresis_value = 0,
    /* Output event when ACOUT is zero? */
    .event_negative = true,
    /* Output event when ACOUT is one? */
    .event_positive = true,
    /* Set the positive input */
    .positive_input = PI_ACP,
    /* Set the negative input */
    .negative_input = NI_ACN,
    /* Set the comparator mode */
    .mode = MODE_CONTINUOUS,
    //.mode = MODE_EVENT_TRIGGERED,
    /* Interrupt settings */
    //.interrupt_settings = IS_COMP_DONE,
    .interrupt_settings = IS_OUTPUT_TGL,
    /* Analog comparator channel number */
    .ac_n = ACIFB_ACA_CHANNEL
};

/* Enable Analog Comparator clock */
sysclk_enable_pba_module(SYSCLK_ACIFB);
/* Disable pullup on ACIFB channel input pins */
gpio_disable_pin_pull_up(ACIFB_ACAP_PIN);
gpio_disable_pin_pull_up(ACIFB_ACAN_PIN);
/* Enable the ACIFB pins */
gpio_enable_module(ACIFB_GPIO_MAP, sizeof(ACIFB_GPIO_MAP) / sizeof(ACIFB_GPIO_MAP[0]));
```

```
    /* Configure the ACIFB peripheral */
    acifb_setup_and_enable(acifb, &acifb_opt);
    /* Configure the ACIFB channel with interrupt & trigger */
    acifb_channels_setup(acifb, &acifb_channel_opt, 1);
}



/*
 * MAIN Function
 *  - Setup Low Power Mode
 *  - Initialize Counters
 *  - Initialize Comparator
 *  - Initialize Hijack Communication protocol
 *  - Start Receive mode and AES
 */
int main (void){
    uint8_t i;

    board_init();
    sysclk_init();

    //disabling unwanted modules on PBA
    //sysclk_disable_pba_module(AVR32_INTC_CLK_PBA);
    //sysclk_disable_pba_module(AVR32_GPIO_CLK_PBA);
    //sysclk_disable_pba_module(AVR32_EIC_CLK_PBA); // Disable EIC PBA clock
    sysclk_disable_pba_module(AVR32_PDCA_CLK_PBA);  // Disable pdca
    sysclk_disable_pba_module(AVR32_WDT_CLK_PBA);   // Disable WatchDog PBA clock
    sysclk_disable_pba_module(AVR32_FREQM_CLK_PBA); // Disable FREQM PBA clock
    sysclk_disable_pba_module(AVR32_USART0_CLK_PBA);// Disable USART0 PBA clock
    sysclk_disable_pba_module(AVR32_USART1_CLK_PBA);// Disable USART1 PBA clock
    sysclk_disable_pba_module(AVR32_USART2_CLK_PBA);// Disable USART2 PBA clock
    sysclk_disable_pba_module(AVR32_USART3_CLK_PBA);// Disable USART3 PBA clock
    sysclk_disable_pba_module(AVR32_SPI_CLK_PBA);   // Disable SPI PBA clock
    sysclk_disable_pba_module(AVR32_TWIM0_CLK_PBA); // Disable TWIM0 PBA clock
    sysclk_disable_pba_module(AVR32_TWIM1_CLK_PBA); // Disable TWIM1 PBA clock
    sysclk_disable_pba_module(AVR32_TWIS0_CLK_PBA); // Disable TWIS0 PBA clock
    sysclk_disable_pba_module(AVR32_TWIS1_CLK_PBA); // Disable TWIS1 PBA clock
    sysclk_disable_pba_module(AVR32_PWMA_CLK_PBA);  // Disable PWMA PBA clock
    //sysclk_disable_pba_module(AVR32_TC0_CLK_PBA); // Disable TC0 PBA clock
    sysclk_disable_pba_module(AVR32_TC1_CLK_PBA);   // Disable TC1 PBA clock
    sysclk_disable_pba_module(AVR32_ADCIFB_CLK_PBA);// Disable ADCIFB PBA clock
    //sysclk_disable_pba_module(AVR32_ACIFB_CLK_PBA);// Disable ACIFB PBA clock
```

```
sysclk_disable_pba_module(AVR32_CAT_CLK_PBA);   // Disable CAT PBA clock
sysclk_disable_pba_module(AVR32_GLOC_CLK_PBA);  // Disable CAT PBA clock
sysclk_disable_pba_module(AVR32_AW_CLK_PBA);    // Disable CAT PBA clock

//pbb modules
sysclk_disable_pbb_module(AVR32_HMATRIX_CLK_PBB);
sysclk_disable_pbb_module(AVR32_SAU_CLK_PBB);

//hsb
sysclk_disable_hsb_module(AVR32_PDCA_CLK_HSB);
sysclk_disable_hsb_module(AVR32_SAU_CLK_HSB);
sysclk_disable_hsb_module(AVR32_HMATRIX_CLK_HSB_PBB_BRIDGE);
sysclk_disable_hsb_module(AVR32_PES_CLK_HSB);

//cpu
sysclk_disable_cpu_module(AVR32_OCD_CLK_CPU);
sysclk_disable_cpu_module(AVR32_CORE_CLK_CPU_COUNT);

//GPIO enable pin pull up
for (i = 0; i<45; i++) {
    gpio_enable_gpio_pin(i);
    gpio_enable_pin_pull_up(i);
}

//Transmit pin
gpio_disable_pin_pull_up(AVR32_PIN_PB10);
gpio_configure_pin(AVR32_PIN_PB10, GPIO_INIT_LOW | GPIO_DIR_OUTPUT);

//testing
gpio_disable_pin_pull_up(my_GPIO_1);
gpio_configure_pin(my_GPIO_1, GPIO_INIT_LOW | GPIO_DIR_OUTPUT);
gpio_disable_pin_pull_up(my_GPIO_2);
gpio_configure_pin(my_GPIO_2, GPIO_INIT_LOW | GPIO_DIR_OUTPUT);


//Initialize the ACIFB
acifb_init();

//por disabled
avr32_scif_t *scifopt=&AVR32_SCIF;
SCIF_UNLOCK(AVR32_SCIF_VREGCR);    scifopt->VREGCR.por33en=0;
SCIF_UNLOCK(AVR32_SCIF_VREGCR);    scifopt->VREGCR.por33mask=1;
```

```
//External interrupt Controller - Sleep Mode
eic_options[0].eic_mode   = EIC_MODE_EDGE_TRIGGERED; // Enable edge-triggered interrupt.
eic_options[0].eic_edge   = EIC_EDGE_FALLING_EDGE;   // Interrupt trig. on falling edge.
eic_options[0].eic_filter = EIC_FILTER_ENABLED;      // Enable filter.
eic_options[0].eic_async  = EIC_SYNCH_MODE;          // Initialize in synchronous mode
eic_options[0].eic_line   = EXT_INT_SLEEP_LINE;      // Set the interrupt line number.

//External interrupt Controller - Wake Mode
eic_options[1].eic_mode   = EIC_MODE_EDGE_TRIGGERED; // Enable edge-triggered interrupt.
eic_options[1].eic_edge   = EIC_EDGE_RISING_EDGE;    // Interrupt trig. on rising edge.
eic_options[1].eic_filter = EIC_FILTER_ENABLED;      // Enable filter.
eic_options[1].eic_async  = EIC_SYNCH_MODE;          // Initialize in synchronous mode
eic_options[1].eic_line   = EXT_INT_WAKE_LINE;       // Set the interrupt line number.

Disable_global_interrupt(); // Disable all interrupts.
INTC_init_interrupts();     // Initialize interrupt vectors.

// Register the EIC interrupt handlers to the interrupt controller.
INTC_register_interrupt(&int_handler_SLEEP, EXT_IRQ_SLEEP_LINE, AVR32_INTC_INT0);
INTC_register_interrupt(&int_handler_WAKE,  EXT_IRQ_WAKE_LINE,  AVR32_INTC_INT0);

// Map the interrupt line to the GPIO pin with the right peripheral function.
gpio_enable_module_pin(EXT_INT_SLEEP_PIN_LINE, EXT_INT_SLEEP_FUNCTION_LINE);
gpio_disable_pin_pull_up(EXT_INT_SLEEP_PIN_LINE);
gpio_enable_module_pin(EXT_INT_WAKE_PIN_LINE, EXT_INT_WAKE_FUNCTION_LINE);
gpio_disable_pin_pull_up(EXT_INT_SLEEP_PIN_LINE);

// Init the EIC controller with the options
eic_init(&AVR32_EIC, eic_options, 2);

// Enable the chosen lines and their corresponding interrupt feature.
eic_enable_line(&AVR32_EIC, EXT_INT_SLEEP_LINE);
eic_enable_interrupt_line(&AVR32_EIC, eic_options[0].eic_line);
eic_enable_line(&AVR32_EIC, EXT_INT_WAKE_LINE);
eic_enable_interrupt_line(&AVR32_EIC, eic_options[1].eic_line);

// Enable all interrupts.
sysclk_enable_peripheral_clock(TX_TC);
INTC_register_interrupt(&tx_tc_irq, AVR32_TC1_IRQ0, AVR32_INTC_INT0);
INTC_register_interrupt(&int_handler_ACIFB, AVR32_ACIFB_IRQ, AVR32_INTC_INT0);
acifb->ier = AVR32_ACIFB_ACINT4_MASK; //enable comparison interrupt
```

```
    Enable_global_interrupt();

    //Disable Comparator Interrupt at start
    acifb_disable_interrupt(acifb, AVR32_ACIFB_ACINT4_MASK);

    // Initialize the timer module
    tx_tc_init(tx_tc);
    rx_tc_init(rx_tc);
    tc_start(tx_tc, TX_TC_CHANNEL);
    //tc_start(rx_tc, RX_TC_CHANNEL);

    //Start Hijack's Communication Protocol
    initializeSystem();


    //Infinite AES encrypt and decrypt loop
    while (true){
        for (i=0; i<16; i++){
            key[i]=i;
            plain[i]=10*i;
            result[i] = 3;
        }
        aes_init();
        aes_encrypt_key128(key, enccx);
        aes_decrypt_key128(key, deccx);
        while (true){
            aes_encrypt(plain, cipher, enccx);
            gpio_set_pin_low(my_GPIO_1);
            aes_decrypt(cipher, result, deccx);
            gpio_set_pin_high(my_GPIO_1);
        }
    }
}

/*
 * Initialize Hijacks's Communication Protocol
 */
void initializeSystem(void) {
    // Initialize Hijack's State Machine
    csm_init();
    csm_registerReceiveByte(fe_handleByteReceived);
    csm_registerTransmitByte(fe_handleByteSent);
```

```
    // Initialize Hijack's Frame Engine
    fe_init();
    fe_registerPacketReceivedCb(packetReceivedCallback);
    fe_registerPacketSentCb(packetSentCallback);
    fe_registerByteSender(csm_sendByte);

    // Start transmitting
    fe_writeTxBuffer(outMessage, 9);
    fe_startSending();
}

/*
 * Functions to interact with Hijack's State Machine
 */
//Update Received Data
void packetReceivedCallback(uint8_t * buf, uint8_t len) {
    bool update = false;
    if(outMessage[0] != buf[0]){outMessage[0] = buf[0]; update=true;}
    if(outMessage[1] != buf[1]){outMessage[1] = buf[1]; update=true;}
    if(outMessage[2] != buf[2]){outMessage[2] = buf[2]; update=true;}
    if(outMessage[3] != buf[3]){outMessage[3] = buf[3]; update=true;}
    if(outMessage[4] != buf[4]){outMessage[4] = buf[4]; update=true;}
    if(outMessage[5] != buf[5]){outMessage[5] = buf[5]; update=true;}
    if(outMessage[6] != buf[6]){outMessage[6] = buf[6]; update=true;}
    if(outMessage[7] != buf[7]){outMessage[7] = buf[7]; update=true;}
    if(outMessage[8] != buf[8]){outMessage[8] = buf[8]; update=true;}

    if(update){
        tx_count=0;
        acifb_disable_interrupt(acifb, AVR32_ACIFB_ACINT4_MASK);
        tc_stop(rx_tc, RX_TC_CHANNEL);
        tc_start(tx_tc, TX_TC_CHANNEL);
    }
}

//Update Transmission Line
void periodicTimerFn (void) {

    uint8_t ats;
    ats = csm_advanceTransmitState();
```

```
    if (ats){
        gpio_set_pin_high(AVR32_PIN_PB10);
    }else{
        gpio_set_pin_low(AVR32_PIN_PB10);
    }

    csm_finishAdvanceTransmitState();
}

//Update transmit buffer
void packetSentCallback(void) {
    fe_writeTxBuffer(outMessage, 9);
}

//Pass Comparator information to Receive State
void captureTimerFn(uint16_t elapsedTime, uint8_t isHigh) {
    struct csm_timer_struct timingData;
    timingData.elapsedTime = elapsedTime;
    timingData.signal = !isHigh;
    csm_receiveTiming(&timingData);
}
```

Appendix B

B. Processor Code: conf_board.h

```
/**
 * Board Configuration Header File
 *  - External Interrupt Pin Mappings
 *  - Timer Counter Module Pin Mappings
 *  - Comparator Module Pin Mappings
 */

#ifndef CONF_BOARD_H
#define CONF_BOARD_H

/**
 * External Interrupt Pin Mappings - Sleep
 */
#define EXT_INT_SLEEP_PIN_LINE         AVR32_EIC_EXTINT_3_0_PIN
#define EXT_INT_SLEEP_FUNCTION_LINE    AVR32_EIC_EXTINT_3_0_FUNCTION
#define EXT_INT_SLEEP_LINE             AVR32_EIC_INT3
#define EXT_IRQ_SLEEP_LINE             AVR32_EIC_IRQ_3

/**
 * External Interrupt Pin Mappings - Wake
 */
#define EXT_INT_WAKE_PIN_LINE          AVR32_EIC_EXTINT_5_0_PIN
#define EXT_INT_WAKE_FUNCTION_LINE     AVR32_EIC_EXTINT_5_0_FUNCTION
#define EXT_INT_WAKE_LINE              AVR32_EIC_INT5
#define EXT_IRQ_WAKE_LINE              AVR32_EIC_IRQ_5
```

```
/**
 * Timer Counter Module Pin Mappings - Receive Counter
 */
#define RX_TC              (&AVR32_TC0)    //TC0 module is enabled
#define RX_TC_CHANNEL      0               //TC Channel 0
#define RX_TC_IRQ          AVR32_TC0_IRQ0  //IRQ0 line of TC0 Channel0.
#define RX_TC_IRQ_PRIORITY AVR32_INTC_INT0 //Interrupt priority 0

/**
 * Timer Counter Module Pin Mappings - Transmit Counter
 */
#define TX_TC              (&AVR32_TC1)    //TC1 module is enabled
#define TX_TC_CHANNEL      0               //TC Channel 0 is used.
#define TX_TC_IRQ          AVR32_TC1_IRQ0  //IRQ0 line of TC1 Channel0
#define TX_TC_IRQ_PRIORITY AVR32_INTC_INT0 //Interrupt priority 0

/**
 * Comparator Module Pin Mappings
 */
#define ACIFB_ACA_CHANNEL   4
#define ACIFB_ACAP_PIN      AVR32_ACIFB_ACAP_2_PIN      //AVR32_PIN_PB00 -> PIN 32
#define ACIFB_ACAP_FUNCTION AVR32_ACIFB_ACAP_2_FUNCTION
#define ACIFB_ACAN_PIN      AVR32_ACIFB_ACAN_2_PIN      //AVR32_PIN_PB02 -> PIN 34
#define ACIFB_ACAN_FUNCTION AVR32_ACIFB_ACAN_2_FUNCTION

/**
 * Test GPIO Pin Mappings
 */
#define my_GPIO_1           AVR32_PIN_PB03              //AVR32_PIN_PB03 -> PIN 35
#define my_GPIO_2           AVR32_PIN_PA22              //AVR32_PIN_PA22 -> PIN 22

#endif // CONF_BOARD_H
```

Appendix C

C. Processor Code: conf_clock.h

```
/**
 * Clock Configuration Header File
 */

#ifndef CONF_CLOCK_H_INCLUDED
#define CONF_CLOCK_H_INCLUDED

//#define CONFIG_SYSCLK_INIT_CPUMASK  (1 << SYSCLK_SYSTIMER)
//#define CONFIG_SYSCLK_INIT_PBAMASK  (1 << SYSCLK_USART0)
//#define CONFIG_SYSCLK_INIT_PBBMASK  (1 << SYSCLK_HMATRIX)
//#define CONFIG_SYSCLK_INIT_HSBMASK  (1 << SYSCLK_MDMA_HSB)

//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RCSYS
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_OSC0
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_DFLL
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RC120M
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLL0

//#define BOARD_OSC0_HZ                       32000  // 32 KHz
//#define BOARD_OSC0_HZ                       100000 //100 KHz
#define BOARD_OSC0_HZ                       1000000      //  1 MHz
#define BOARD_OSC0_TYPE                     XOSC_TYPE_XTAL
#define BOARD_OSC0_IS_XTAL            true
#define BOARD_OSC0_STARTUP_US         17000
```

```
/* Fbus = Fsys / (2 ^ BUS_div) */
#define CONFIG_SYSCLK_CPU_DIV         0
#define CONFIG_SYSCLK_PBA_DIV         0
#define CONFIG_SYSCLK_PBB_DIV         0

//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_OSC0
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_RCSYS
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_OSC32K
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_RC120M
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_RC32K

/* Fdfll = (Fclk * DFLL_mul) / DFLL_div */
//#define CONFIG_DFLL0_MUL            (48000000UL / BOARD_OSC32_HZ)
//#define CONFIG_DFLL0_DIV            1

#endif /* CONF_CLOCK_H_INCLUDED */
```

Appendix D

D. Processor Code: asf.h

```
/**
 * \file
 *
 * \brief Autogenerated API include file for the Atmel Software Framework (ASF)
 *
 * Copyright (c) 2012 Atmel Corporation. All rights reserved.
 *
 * \asf_license_start
 *
 * \page License
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The name of Atmel may not be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 *    Atmel microcontroller product.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * \asf_license_stop
 *
 */

#ifndef ASF_H
#define ASF_H

/*
 * This file includes all API header files for the selected drivers from ASF.
 * Note: There might be duplicate includes required by more than one driver.
 *
 * The file is automatically generated and will be re-written when
 * running the ASF driver selector tool. Any changes will be discarded.
 */

// From module: ACIFB - Analog Comparator Interface
#include <acifb.h>

// From module: Common build items for user board support templates
#include <user_board.h>

// From module: Compiler abstraction layer and code utilities
#include <compiler.h>
#include <status_codes.h>

// From module: EIC - External Interrupt Controller
#include <eic.h>

// From module: FLASH Controller Double-Word (FLASHCDW)
```

```
#include <flashcdw.h>

// From module: GPIO - General-Purpose Input/Output
#include <gpio.h>

// From module: Generic board support
#include <board.h>

// From module: INTC - Interrupt Controller
#include <intc.h>

// From module: Interrupt management - UC3 implementation
#include <interrupt.h>

// From module: PM Power Manager - UC3 L0 implementation
#include <power_clocks_lib.h>
#include <sleep.h>

// From module: Part identification macros
#include <parts.h>

// From module: SCIF System Control Interface - UC3L implementation
#include <scif_uc3l.h>

// From module: System Clock Control - UC3 L0 implementation
#include <sysclk.h>

// From module: TC - Timer/Counter
#include <tc.h>

#endif // ASF_H
```

Appendix E

E. Android Code: ApplicationInterface.java

```
/*
 *  This file is part of hijack-infinity.
 *
 *  hijack-infinity is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  hijack-infinity is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with hijack-infinity.  If not, see <http://www.gnu.org/licenses/>.
 */

package app.smartTPM;

import java.util.Timer;
import java.util.TimerTask;

import app.smartTPM.core.FramingEngine;
import app.smartTPM.core.OnByteSentListener;
import app.smartTPM.core.OnBytesAvailableListener;
import app.smartTPM.core.SerialDecoder;
```

```
import app.smartTPM.core.FramingEngine.IncomingPacketListener;
import app.smartTPM.core.FramingEngine.OutgoingByteListener;

public class ApplicationInterface {
    private SerialDecoder _serialDecoder;
    private FramingEngine _framer;

    private int[] _dataR = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0};
    private int[] _dataT = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0};

    private boolean _isConnected = false;
    private boolean _hasUpdated = false;

    private int _pendingTransmitBytes = 0;

    private Timer _watchdogConnectionTimer;
    private UpdateListener _listener = null;

    public ApplicationInterface() {
        _framer = new FramingEngine();
        _serialDecoder = new SerialDecoder();

        _serialDecoder.registerBytesAvailableListener(_bytesAvailableListener);
        _serialDecoder.registerByteSentListener(_byteSentListener);
        _framer.registerIncomingPacketListener(_incomingPacketListener);
        _framer.registerOutgoingByteListener(_outgoingByteListener);

        int[] toSend = encode();
        for (int i = 0; i < toSend.length; i++) {
            _framer.transmitByte(toSend[i]);
        }
        _framer.transmitEnd();
    }

    private OutgoingByteListener _outgoingByteListener = new OutgoingByteListener() {
        public void OutgoingByteTransmit(int[] outgoingRaw) {
            synchronized (ApplicationInterface.this) {
                _pendingTransmitBytes += outgoingRaw.length;
            }

            for (int i = 0; i < outgoingRaw.length; i++) {
                _serialDecoder.sendByte(outgoingRaw[i]);
```

```
            }
        }
    };

    private IncomingPacketListener _incomingPacketListener = new IncomingPacketListener() {
        public void IncomingPacketReceive(int[] packet) {
            for (int i = 0; i < packet.length; i++) {
                System.out.print(Integer.toHexString(packet[i]) + " ");
            }
            System.out.println();
            decodeAndUpdate(packet);
        }
    };

    private OnByteSentListener _byteSentListener = new OnByteSentListener() {
        public void onByteSent() {
            synchronized (ApplicationInterface.this) {
                _pendingTransmitBytes--;
                if (_pendingTransmitBytes == 0) {
                    int[] toSend = encode();
                    for (int i = 0; i < toSend.length; i++) {
                        _framer.transmitByte(toSend[i]);
                    }
                    _framer.transmitEnd();
                }
            }
        }
    };

    private OnBytesAvailableListener _bytesAvailableListener = new
OnBytesAvailableListener() {
        public void onBytesAvailable(int count) {
            while(count > 0) {
                int byteVal = _serialDecoder.readByte();
                _framer.receiveByte(byteVal);
                count--;
            }
        }
    };

    private void decodeAndUpdate(int[] packet) {
        if (packet.length != 9) {
```

```
            return;
        }

        synchronized (this) {
            _hasUpdated = true;

            _dataR[0]=(int)(packet[0]&0xFF);
            _dataR[1]=(int)(packet[1]&0xFF);
            _dataR[2]=(int)(packet[2]&0xFF);
            _dataR[3]=(int)(packet[3]&0xFF);
            _dataR[4]=(int)(packet[4]&0xFF);
            _dataR[5]=(int)(packet[5]&0xFF);
            _dataR[6]=(int)(packet[6]&0xFF);
            _dataR[7]=(int)(packet[7]&0xFF);
            _dataR[8]=(int)(packet[8]&0xFF);
        }

        OnUpdateListener();
    }

    private int[] encode() {
        //return new int[] {_dataT[0], _dataT[1], _dataT[2]};
        return _dataT;
    }

    public void start() {
        _serialDecoder.start();
        _watchdogConnectionTimer = new Timer();
        _watchdogConnectionTimer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                synchronized (ApplicationInterface.this) {
                    _isConnected = _hasUpdated;
                    _hasUpdated = false;
                }
                OnUpdateListener();
            }
        }, 1000, 1000);
    }

    public void stop() {
        _watchdogConnectionTimer.cancel();
```

```
        _watchdogConnectionTimer = null;
        _serialDecoder.stop();

        _isConnected = false;
        OnUpdateListener();
    }

    public synchronized void registerOnUpdateListener(UpdateListener listener) {
        _listener = listener;
    }

    public synchronized double getDataR(int i) {
        return _dataR[i];
    }

    public synchronized void setDataT(int i, int value) {
        _dataT[i]=value;
    }

    public synchronized boolean getIsConnected() {
        return _isConnected;
    }

    public interface UpdateListener {
        public abstract void Update();
    }

    public void OnUpdateListener() {
        if (_listener != null) {
            _listener.Update();
        }
    }
}
```

Appendix F

F. Android Code: MainActivity.java

```
package app.smartTPM;

import java.text.DecimalFormat;

import app.smartTPM.ApplicationInterface.UpdateListener;
import android.os.Bundle;
import android.app.Activity;
import android.graphics.Color;

import android.view.Menu;
import android.view.View;

import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    private ApplicationInterface _appInterface;

    private TextView _tvStatus;

    private TextView _tvDataR0, _tvDataR1, _tvDataR2, _tvDataR3, _tvDataR4, _tvDataR5,
_tvDataR6, _tvDataR7, _tvDataR8;
    private TextView _tvDataT0, _tvDataT1, _tvDataT2, _tvDataT3, _tvDataT4, _tvDataT5,
_tvDataT6, _tvDataT7, _tvDataT8;

    private Button _bSend;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    _appInterface = new ApplicationInterface();

    bindControls();

    _appInterface.registerOnUpdateListener(_listener);
    attachButtonListener();
}

private UpdateListener _listener = new UpdateListener() {
    public void Update() {
        runOnUiThread(new Runnable() {
            public void run() {
                updateBindings();
            }
        });
    }
};

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public void onPause() {
    _appInterface.stop();
    super.onPause();
}

@Override
public void onResume() {
    _appInterface.start();
    super.onResume();
}
```

```
    private void bindControls() {
        _tvStatus = (TextView) findViewById(R.id.textViewStatus);
        _bSend    = (Button)   findViewById(R.id.bSend);

        //
        _tvDataR0 = (TextView) findViewById(R.id.dataR_0);
        _tvDataR1 = (TextView) findViewById(R.id.dataR_1);
        _tvDataR2 = (TextView) findViewById(R.id.dataR_2);
        _tvDataR3 = (TextView) findViewById(R.id.dataR_3);
        _tvDataR4 = (TextView) findViewById(R.id.dataR_4);
        _tvDataR5 = (TextView) findViewById(R.id.dataR_5);
        _tvDataR6 = (TextView) findViewById(R.id.dataR_6);
        _tvDataR7 = (TextView) findViewById(R.id.dataR_7);
        _tvDataR8 = (TextView) findViewById(R.id.dataR_8);

        _tvDataT0 = (TextView) findViewById(R.id.dataT_0);
        _tvDataT1 = (TextView) findViewById(R.id.dataT_1);
        _tvDataT2 = (TextView) findViewById(R.id.dataT_2);
        _tvDataT3 = (TextView) findViewById(R.id.dataT_3);
        _tvDataT4 = (TextView) findViewById(R.id.dataT_4);
        _tvDataT5 = (TextView) findViewById(R.id.dataT_5);
        _tvDataT6 = (TextView) findViewById(R.id.dataT_6);
        _tvDataT7 = (TextView) findViewById(R.id.dataT_7);
        _tvDataT8 = (TextView) findViewById(R.id.dataT_8);
    }

    private void updateBindings() {
        DecimalFormat decimalFormat = new DecimalFormat("#.##");

        _tvStatus.setText(_appInterface.getIsConnected() ? R.string.status_connected :
R.string.status_disconnected);
        _tvStatus.setTextColor(_appInterface.getIsConnected() ?
Color.parseColor("#11FF11") : Color.parseColor("#FF1111"));

        _tvDataR0.setText(decimalFormat.format(_appInterface.getDataR(0)));
        _tvDataR1.setText(decimalFormat.format(_appInterface.getDataR(1)));
        _tvDataR2.setText(decimalFormat.format(_appInterface.getDataR(2)));
        _tvDataR3.setText(decimalFormat.format(_appInterface.getDataR(3)));
        _tvDataR4.setText(decimalFormat.format(_appInterface.getDataR(4)));
        _tvDataR5.setText(decimalFormat.format(_appInterface.getDataR(5)));
        _tvDataR6.setText(decimalFormat.format(_appInterface.getDataR(6)));
        _tvDataR7.setText(decimalFormat.format(_appInterface.getDataR(7)));
```

```
        _tvDataR8.setText(decimalFormat.format(_appInterface.getDataR(8)));
    }

    private void attachButtonListener() {
        _bSend.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                try {
                    _appInterface.setDataT(0, Integer.parseInt(""+_tvDataT0.getText()));
                    _appInterface.setDataT(1, Integer.parseInt(""+_tvDataT1.getText()));
                    _appInterface.setDataT(2, Integer.parseInt(""+_tvDataT2.getText()));
                    _appInterface.setDataT(3, Integer.parseInt(""+_tvDataT3.getText()));
                    _appInterface.setDataT(4, Integer.parseInt(""+_tvDataT4.getText()));
                    _appInterface.setDataT(5, Integer.parseInt(""+_tvDataT5.getText()));
                    _appInterface.setDataT(6, Integer.parseInt(""+_tvDataT6.getText()));
                    _appInterface.setDataT(7, Integer.parseInt(""+_tvDataT7.getText()));
                    _appInterface.setDataT(8, Integer.parseInt(""+_tvDataT8.getText()));
                } catch (Exception e) {
                }
            }
        });
    }
}
```

Appendix G

G. Android Code: activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="app.smartTPM.MainActivity" >

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

         <TextView
             style="@android:style/TextAppearance.Large"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:layout_marginLeft="10dp"
             android:layout_marginTop="10dp"
             android:text="@string/status"
             tools:context=".FramJack" />

        <TextView
            android:id="@+id/textViewStatus"
            style="@android:style/TextAppearance.Large"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:layout_marginLeft="5dp"
        android:layout_marginTop="10dp"
        android:text="@string/status_connected"
        android:textColor="#11ff11"
        tools:context=".FramJack" />

</LinearLayout>

<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/tableLegend_1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:text="@string/tableLegend_text_1"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/tableLegend_2"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:text="@string/tableLegend_text_2"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/tableLegend_3"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:text="@string/tableLegend_text_3"
            android:textAppearance="?android:attr/textAppearanceMedium" />
    </TableRow>
```

```
<TableRow
    android:id="@+id/tableRow2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center" >

    <TextView
        android:id="@+id/dataID_0"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:gravity="center"
        android:text="@string/dataID_text_0"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/dataT_0"
        android:layout_width="10dp"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:ems="10"
        android:gravity="center"
        android:inputType="number"
        android:text="@string/dataT_text" >

        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/dataR_0"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        android:gravity="center"
        android:text="@string/dataR_text"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</TableRow>

<TableRow
    android:id="@+id/tableRow3"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_1"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_1"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow4"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >
```

```
        <TextView
            android:id="@+id/dataID_2"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_2"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_2"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_2"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow5"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_3"
```

```
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_3"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_3"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_3"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow6"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_4"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
```

```
            android:gravity="center"
            android:text="@string/dataID_text_4"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_4"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_4"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow7"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_5"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_5"
            android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
        <EditText
            android:id="@+id/dataT_5"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_5"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow8"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_6"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_6"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_6"
```

```
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_6"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow9"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_7"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_7"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_7"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
```

```
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
            android:text="@string/dataT_text" >

            <requestFocus />
        </EditText>

        <TextView
            android:id="@+id/dataR_7"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataR_text"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>

    <TableRow
        android:id="@+id/tableRow10"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center" >

        <TextView
            android:id="@+id/dataID_8"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:text="@string/dataID_text_8"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/dataT_8"
            android:layout_width="100dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:ems="10"
            android:gravity="center"
            android:inputType="number"
```

```
                android:text="@string/dataT_text" >

                <requestFocus />
            </EditText>

            <TextView
                android:id="@+id/dataR_8"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_gravity="center"
                android:gravity="center"
                android:text="@string/dataR_text"
                android:textAppearance="?android:attr/textAppearanceMedium" />

        </TableRow>

        <Button
            android:id="@+id/bSend"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/buttonT_text" />

    </TableLayout>

</LinearLayout>
```

Appendix H

H. Android Code: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">smartTPM02</string>
    <string name="action_settings">Settings</string>
    <string name="status">Status:</string>
    <string name="status_connected">Connected</string>
    <string name="status_disconnected">Disconnected</string>

    <string name="tableLegend_text_1">Data</string>
    <string name="tableLegend_text_2">Transmit</string>
    <string name="tableLegend_text_3">Receive</string>
    <string name="dataT_text">0</string>
    <string name="dataR_text">0</string>
    <string name="buttonT_text">SEND</string>

    <string name="dataID_text_0">0</string>
    <string name="dataID_text_1">1</string>
    <string name="dataID_text_2">2</string>
    <string name="dataID_text_3">3</string>
    <string name="dataID_text_4">4</string>
    <string name="dataID_text_5">5</string>
    <string name="dataID_text_6">6</string>
    <string name="dataID_text_7">7</string>
    <string name="dataID_text_8">8</string>
</resources>
```

Appendix I

I. Andriod Code: AndriodManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="app.smartTPM"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="16" />
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Bibliography

[1] Y.S. Kuo, T. Schmid, and P. Dutta. Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface, August 2010. In *International Symposium on Low Power Electronics and Design*. ISLPED'10.

[2] Y.S. Kuo, S. Verma, T. Schmid, and P. Dutta. Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface, December 2010. In *First Annual Symposium on Computing for Development*. DEV'10.

[3] Robinson, A. Hijack-infinity. DOI=https://github.com/anroOfCode/hijack-infinity

[4] R. Velegalati and J.-P. Kaps. Towards a Flexible, Opensource BOard for Side-channel analysis (FOBOS), June 2013. In *Cryptographic architectures embedded in reconfigurable devices*, CRYPTARCHI 2013.

[5] R. Velegalati and J.-P. Kaps. Introducing FOBOS: Flexible Open-source BOard for Side-channel analysis, May 2012. In *Work in Progress (WiP), Third International Workshop on Constructive Side-Channel Analysis and Secure Design*, COSADE 2012

[6] T. Katashita, Y. Hori, H. Sakane, and A. Satoh, Side-channel attack standard evaluation board SASEBO-W for smartcard testing, 2012. In *Power*, vol. 3, page 400.

[7] Side-Channel Attack Standard Evaluation Board (SASEBO), 2014. DOI=http://www.risec.aist.go.jp/project/sasebo

[8] C. O'Flynn and Z. Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research, 2014. In *IACR Cryptology ePrint Archive*, page 204.

[9] J. Kim, K. Oh, D. Choi, and H. Kim. SCARF: profile-based side channel analysis resistant framework, July 2012. In *Security and Management International Conference*, SAM'12.

[10] Rambus. DPA Workstation Analysis Platform. DOI=http://www.rambus.com/security/dpa-countermeasures/dpa-workstation-platform/

[11] Riscure. InspectorSCA. DOI=https://www.riscure.com/security-tools/inspector-sca/

[12] Atmel. 32-bit Atmel AVR Microcontroller, 2012. In *datasheet for AT32UC3L064.* DOI=http://www.atmel.com/Images/doc32099.pdf

[13] Atmel. Atmel AVR32924: UC3-L0 XPLAINED Hardware User's Guide, 2013. In *datasheet for UC3-L0 XPLAINED.* DOI=http://www.atmel.com/Images/Atmel-32156-UC3-L0-XPLAINED-Hardware-Users-Guide_Application-Note_AVR32924.pdf

[14] Atmel. 32-bit AVR Microcontrollers, 2013. In *datasheet for UC3L.* DOI=http://www.atmel.com/Images/doc32129.pdf