Modular Integration Platform for Printed Electronics

Seiya Ono
Ana Claudia Arias, Ed.
Prabal Dutta, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-156
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-156.html

August 13, 2020

Modular Integration Platform for Printed Electronics

By Seiya Ono

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Ana Claudia Arias

Professor Ana Claudia Arias
Research Advisor

August 6th, 2020

(Date)

* * * * * * *

Prabal K Dutta

Professor Prabal Dutta
Second Reader

August 7th, 2020

(Date)

Modular Integration Platform for Printed Electronics

by

Seiya Ono

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ana Claudia Arias, Chair
Professor Prabal Dutta

Spring 2020

Modular Integration Platform for Printed Electronics

Abstract

Modular Integration Platform for Printed Electronics

by

Seiya Ono

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Ana Claudia Arias, Chair

The field of flexible and printed electronics has been on the rise, with an increasing demand for low cost, physically robust, conformal sensors and devices to be used in a variety of applications in the age of edge computing. It would be unreasonable, however, to expect all of these flexible organic electronics to perform at remotely similar orders of magnitude with respect to solid state electronics - this introduces a need for flexible hybrid electronics that integrate the speed and power efficiency of solid state electronics with flexible, printed electronics to occupy the middle ground between the two.

The development of flushed out hybrid electronics systems is often seen to be the end goal, utilizing new, ground breaking fabrication technology to make new sensors, in conjunction with high performance, solid state electronics for small form factor, well packaged final products. While this does make for very beautiful end products, the amount of prototyping and research effort put into making the flexible electronics interface, the data collection, the enclosure, and every perfectly tuned parameter is something that becomes very repetitive, akin to "reinventing the wheel".

I propose that a modular design approach could help alleviate much of the repetitive aspects pertaining to the prototyping efforts in flexible hybrid electronics. I will present my new integration tool, the Modular Integration Platform for Printed Electronics (MIPPE), and demonstrate the effectiveness of a modular approach to design through the use of the various platforms for development available in MIPPE and some of its example use cases. By distilling the most common variations of flexible electronic interfaces into interchangeable, reusable hardware blocks, the R&D time would be reduced, increasing research productivity in the field of flexible and printed electronics.

Contents

Contents i

List of Figures ii

Listings ii

1 Trade-offs of Organic Electronics 1

1.1 Organic Electronics 1
1.2 Printing Techniques and Implications on Manufacturability 1
1.3 Augmenting Shortcomings with Solid State Electronics 3

2 Past Integration Projects 5

2.1 Force Calibration 5
2.2 Wound Stimulation & Bioimpedance Monitoring 7
2.3 Integrated Wound Monitoring 9
2.4 Developer Notes 10

3 Modular Integration Platform for Printed Electronics 13

3.1 Initial Ideas and Concepts 13
3.2 Firmware Platform 14
3.3 Hardware Platform 15
3.4 Small Scale Manufacturing Choices 17
3.5 Final Conceptualization 18
3.6 Modularity in Design 21
3.7 Application Information 23

4 Future Work & Conclusion 29

4.1 More Powerful, Reliable Hardware 29
4.2 Scalable Firmware Environments 30
4.3 GUI & Software Optimization 30
4.4 Final Words 31

Bibliography 32

List of Figures

1.1 Printing Techniques[12] 3

2.1 Force Calibration System Schematic 6
2.2 Impedance Analyzer & Adjustable Current Source System Schematic[1,2,3] 8
2.3 Fully Integrated System with Printed Electrodes and Organic Oximeter[1,7] 11

3.1 Bootloader Burn-In Tool 16
3.2 Various Implementations of Atmel Microcontrollers on Custom Hardware 17
3.3 First System Overview 18
3.4 Leader-Follower Configuration for UART[5] 19
3.5 Cardboard Prototype of Module-Motherboard Interconnects 21
3.6 Final System Design 22
3.7 Design Modularity of MIPPE Modules 23
3.8 MIPPE User Interface Example 27
3.9 MIPPE Data Visualizer Example 28

Listings

3.1 Follower side Motherboard-Module Communication Standard 20
3.2 Generic MIPPE Module class 25

Acknowledgments

Before I begin, I must first thank my advisor, Professor Ana Claudia Arias, for her endless support and enthusiasm for my work. There were many periods where I felt lost, lacking motivation, or stunned at the challenges that were presented to me - Professor Arias was always reassuring in my worries while providing realistic, yet optimistic views that got me back up to continue pressing forwards. Thank you for all of the opportunities and resources you have provided me.

Huge thank you to Yasser for, on a whim, inviting me to attend one of the group meetings, effectively kicking off my research adventure with that kind gesture. To everyone I've had the pleasure of working with in the group, thank you for giving me the chance to participate in your outstanding work. Every time I listen to each of you present on your work, I am repeatedly amazed at what you all can accomplish. Thank you Mahsa, Yasser, Ting, Jasmine, Juan, Xiaodong, Maggie, and Matt for being so patient with me when training me on the equipment. To the fifth floor crew, I will miss our lunches together - the short break in the middle of the day with you all was something I always looked forward to. Thanks to Cher, Jasmine, and Natalie for being the fun, short lived, first year cohort that I never knew I wanted.

Outside the lab, I would like to thank my friends I have made during my five years at Berkeley; whether we crossed paths in Pioneers in Engineering, EE16A course staff, or just by chance, I appreciate all of you for the times we spent together cooking, eating, playing games, and adventuring, freeing my mind from the stresses and responsibilities that came with school.

I could not close this without thanking my parents who always encouraged my pursuit of education, being accepting when I couldn't come home for breaks, and giving me a comfortable, stress free space to work at home during the pandemic. Without their support, I would not be where I am today. I am eternally grateful.

Chapter 1

Trade-offs of Organic Electronics

1.1 Organic Electronics

Traditionally, electronics have been confined to a rigid platform, utilizing non-conformal, inflexible mediums. These limitations have proven to be quite cumbersome to deal with, as the new frontier of technology demands similar performance on flexible platforms to enable point of application sensing and computation. Unfortunately, silicon as well as its medium, the printed circuit board (PCB) does not perform well under mechanical stress, flexing, and stretching, if at all, making these two options, solid state electronics and PCBs, very limiting when aiming for conformal sensors and computation. Hence, organic electronics.

Organic electronics utilize non-rigid, solution processable polymers as the semiconducting layer. Together with conductive inks, flexible plastic substrates, and a polymer encapsulation layer, the devices can be fabricated to be fully printed and fully flexible. Additionally, the same fabrication technology can be used to create sensors, enabling the sensing mechanism, front end conditioning, and computation to occur on the same flexible medium. While in the present day this may be a stretch, fully flexible and printed integrated devices may not be a too distant future. In order to get there, we must first understand its technological niche.

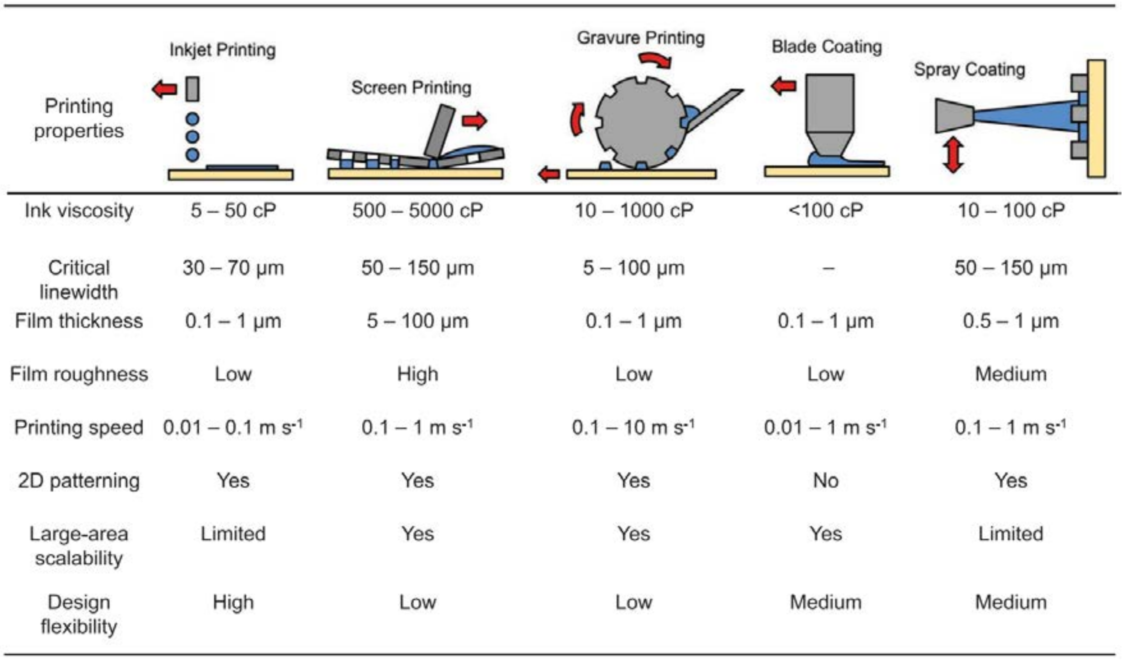
1.2 Printing Techniques and Implications on Manufacturability

Those familiar with fabrication technology pertaining to solid state electronics might understand more the intertwined nature of various fabrication techniques, materials, and device performances, not to even mention cost. Many hours go into optimization methods to try and get the highest performing devices by understanding the limitations of a particular process, be it deposition, lithography, or etching. The various process steps have even further implications on how different types of errors propagate and affect the end device performance; for solid state, these steps can be quite expensive and difficult to scale.

To name a few critical processes difficult to scale: projection lithography needs to utilize rastering, limiting exposure throughput; Deep Reactive-Ion Etching (DRIE) utilizes many cycles of gas flow, limiting etch throughput; Atomic Layer Deposition (ALD) also cycles precursors and target gas to deposit extremely conformal, tunable thin films, which once again, limits deposition throughput. Throughput, though a very surface level analysis, can be used to quickly determine cost effectiveness of a fabrication technique. The solid state electronics industry can sacrifice a small loss in throughput, as the performance gains from the aforementioned fabrication steps pays off the time loss by many fold - there also lacks any alternative techniques that can produce devices remotely close to the cutting edge technologies that use said fabrication steps; thus, with the lack of options and massive performance gains, the small sacrifice in throughput is an easy choice to make.

These fabrication techniques may yield higher performance per cost, but are mostly optimized solely for solid state electronics, and are often incompatible and are lacking in perspective if directly translated to work with organic electronics. So starting with the easier to digest one, compatibility. Many processes designed around solid state electronics have a large thermal budget, with temperatures often exceeding 300^oC - temperatures that would easily burn most thin films of polymers and lead to deformation. Many lithography processes would also be dismissed, due to the high amounts of energy shot at potentially semi-transparent organic layers that can ruin device performance. The composition of various polymers used in organic electronics do not lend themselves to be exposed to the harsh nature of solid state fabrication techniques. Moving to the second point of lacking perspective. Due to the performance shortcomings of organic electronics, it makes no sense to try and optimize them for extremely expensive fabrication technologies - any cost increase in the fabrication process for organic electronics takes away from one of their most promising features: cost. Additionally, even in the best case vision where the fabrication techniques are fully optimized to net absolute maximum performance, the already expensive fabrication methods taken from solid state foundries are often not large area compatible, and even less compatible in roll-to-roll manufacturing, nullifying the other two great features of organic electronics: large area scalability and roll-to-roll manufacturing. To preserve these two upsides, it is infeasible to try and optimize traditionally high performance fabrication techniques present in solid state fabrication. Instead, we should turn towards much cheaper fabrication techniques that can be scaled up to harness the strength of organic electronics.

The goal is to make organic devices using cost optimized methods with equipment that is easy to use and compatible with the previously mentioned large area or roll-to-roll manufacturing, while utilizing, at best, fully solution processable materials and with purely additive deposition. This way, material can be saved, waste can be minimized, and costs can be brought down. Here is a quick overview of the various printing techniques that are used in our lab, with figures and printing properties outlined in figure 1.1. In inkjet printing, a MEMS nozzle can precisely and accurately place drops of a solution onto a substrate to create patterns. Similar to regular household inkjet printers, they can be very slow, limited by the nozzle efficiency and the viscosity of the solution. Screen printing is a similar process to graphic T-shirt printing, where a squeegee is pushed along the screen that holds the pattern, and deposits the material only where the screen is open. Gravure printing is a rolling application of material, where the pattern is on the spinning wheel and gets applied to the substrate below. Blade coating is a method of applying a solution at a very controlled thickness across a large area using a doctor blade and an actuator to push the blade across a substrate very steadily. It can be imagined like a knife spreading butter on toast. Patterns can be applied by altering the surface energy of the substrate to create hydrophobic and hydrophilic regions. Finally, spray coating uses a nozzle that sprays the solution onto a mask to pattern the solution.



Printing properties	Inkjet Printing	Screen Printing	Gravure Printing	Blade Coating	Spray Coating
Ink viscosity	5 – 50 cP	500 – 5000 cP	10 – 1000 cP	<100 cP	10 – 100 cP
Critical linewidth	30 – 70 µm	50 – 150 µm	5 – 100 µm	–	50 – 150 µm
Film thickness	0.1 – 1 µm	5 – 100 µm	0.1 – 1 µm	0.1 – 1 µm	0.5 – 1 µm
Film roughness	Low	High	Low	Low	Medium
Printing speed	0.01 – 0.1 m s^{-1}	0.1 – 1 m s^{-1}	0.1 – 10 m s^{-1}	0.01 – 1 m s^{-1}	0.1 – 1 m s^{-1}
2D patterning	Yes	Yes	Yes	No	Yes
Large-area scalability	Limited	Yes	Yes	Yes	Limited
Design flexibility	High	Low	Low	Medium	Medium

Figure 1.1: Printing Techniques[12]

Much like the various steps in solid state fabrication, each printing technique has its own unique trade-offs and compatibilities with different materials, and need to be optimized for specific processes. In short, the strengths of organic electronics lie in their flexibility, scalability, and cost - and with almost everything in this world, these metrics come with trade offs.

1.3 Augmenting Shortcomings with Solid State Electronics

Solid state electronics may not be the focus of this paper, but it is always important to compare alternatives, as it gives a more realistic perspective when designing for application.

To begin, organic transistors do not compare to the performance metrics of solid state electronics. When comparing, for example, organic thin film transistors[13] and Intel's Trigate Transistor[4] figures of merit, the operating voltage can be almost ten times higher, the clocking frequency can be five orders apart, with on/off ratios being less than ideal. The Trigate transistors have an estimated ten year lifetime, where organic electronics are stuck generally to a few weeks at their best. As an aside, though much of the comparisons that have been made between solid state and organic electronics have been in the context of transistors, the fabrication and cost comparisons can be extended beyond transistors, including photovoltaics, photonics, and more. This vast difference in performance makes direct competition in the computational world impossible, but organic electronics can often find niches in the sensor realm, where immense speeds are not required and can be further augmented with solid state technology.

To overcome the major limitations of organic electronics, that is, high power consumption, less than ideal performance, and short lifetime, researchers have turned to a different approach: Flexible Hybrid Electronics.[10] Instead of optimizing one or the other, the approach is to combine both and utilize each of their strengths for versatile, semi-flexible sensing platforms with computation power of solid state electronics. This does, however, introduce a different type of complexity, in the form of integration. With organic and solid state electronics working side by side, there comes with it, compatibility issues, development time, and prototyping costs which may also burden the researchers. In many scenarios, there are often a very limited pool of interfacing hardware, creating a sense of repetition and monotony each time a new system is required for a particular sensing application. While certainly some optimizations can be made between each sensor's back end hardware interface, the focus of the research is largely not the back end hardware interface, but more a proof of concept for the front end sensor.

Chapter 2

Past Integration Projects

2.1 Force Calibration

A force calibration system was needed to test the durability, hysteresis characteristics, and longevity of one of our newly developed pressure sensors. It needed to precisely apply force to a surface and cycle the measurement an arbitrary number of times through an easy to use, minimal user interface to help calibrate and characterize the sensor. The new pressure sensor was fabricated on a thin flexible substrate and translated a change in pressure to a change in resistance when a normal stress was applied against the sensor. This meant that the force application had to happen similar to a clamping motion by pressing the sensor between two rigid bodies in a controlled, repeatable manner.

The approach was to use a stepper motor to control a linear actuator to move a rigid body to clamp down the sensor against a separate immoving structure. This way, the motion could be very finely controlled with repeatable outcomes. The stepper motor was controlled through a high power motor driver, driven by a microcontroller with firmware that could interface with a laptop through a serial connection. An important quantity to keep in mind for a stepper motor application is the precision of each "tick". The operating principle of a stepper motor involves open loop control, sending pulses to a motor driver that can move the stepper motor one tick, where a tick is defined by the structure and gear ratio of the stepper motor. This is translated into a linear motion through a gear box. The system was measured to have a resolution of $1.6\mu m$ per tick.

With the physical system complete, the next step was developing a user interface that could take in the user parameters and be left to run on its own. Processing was used to develop a minimal user interface for this project.

The UI has three main capabilities - (1) selecting the serial interface (2) programming a routine (3) starting and stopping the routine. Selecting the serial interface is straight forward: it lists all the available COM ports and gives the user the option to select the correct one that corresponds to the force calibration device. The programmable routine consists of three different inputs: the distance to cover, the speed to travel the distance,

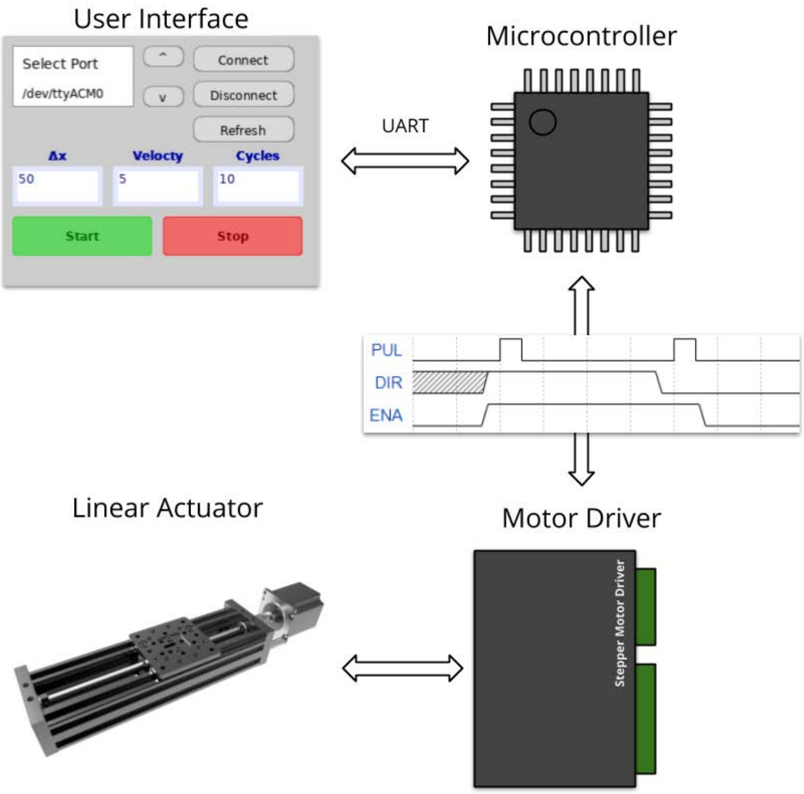


Figure 2.1: Force Calibration System Schematic

and the number of cycles it will operate. For example, the UI pictured in figure 2.1 has the base traveling 50 centimeters at a velocity of 5 centimeters per second, and will cycle 50 centimeters forward and 50 centimeters backwards a total of 10 cycles.

This system became a part of our characterization lab, enabling researchers to test the durability and repeatability of a variety of sensors and devices,[14] being able to program many thousands of cycles at repeatable intervals. It was used to characterize strain and pressure sensors, and also used for bend tests for a variety of flexible devices to test performance under non-planar conditions. While this project did not outline the integration difficulties that came along with flexible hybrid electronics, it gave me a taste of what it would be like to try and realize a product that fit the needs for flexible electronics, whether it be communication, broad expertise and know-how, or the general understanding for flexible electronic operating principles. Implementation details can be found here.

2.2 Wound Stimulation & Bioimpedance Monitoring

A collaboration project with UCSD led to the development of a prototype platform to test wound stimulation for accelerated healing and bioimpedance wound monitoring using our printed flexible electrodes. The goal was to have a monitoring system that could collect impedance data from an array of electrodes placed on a wound, while simultaneously stimulating the healing process by applying a small current through the tissue.

The deliverable was split into two distinct parts: the impedance analyzer for monitoring and the current source for stimulation. Each had to be able to interface with the same electrode array of fifteen individually addressable contacts. An impedance analyzer should be able to have a programmable frequency sweep that takes impedance measurements at each point to string together an array of data that describes a material's real and imaginary resistances. To accomplish this, I built a small proof of concept system that included an impedance analyzer front end chip, the AD5933.[1] This chip was capable of operating programmable frequency sweeps up to 100kHz, with adjustable excitation voltages while being able to measure complex resistances up to $100M\Omega$. The chip was outputting a DC biased sine wave and using that as the input for an Op-Amp inverting amplifier topology, with the characteristic impedance taking the place of the input resistor. This had implications on saturation and precision, as all Analog to Digital Converters (ADCs) must deal with. This then has limitations on the range of impedances that this could measure for any given reference resistor set by the user. If the user chosen reference resistor was too many orders larger than the target impedance, the ADC in the AD5933 would saturate, leading to inaccurate readings. To accommodate for this, I asked the researchers at UCSD to send me data on previous wound impedance measurements that were conducted using a lab bench impedance analyzer which would be much too bulky for a deployable application. I chose the reference resistance to be the same magnitude as the smaller range of impedances as to avoid any saturation problem.

The second half of the design was the tunable current source for wound stimulating to promote accelerated healing. Sourcing current is difficult due to the limitations that come with Ohm's law - any amount of current going through a large impedance requires a huge potential to drive said current. Looking at the previous data for wound impedances, the specification that the research had in mind was shown to be impossible to hit without the use of very expensive boost converters. With the power restriction, I was able to a hit a few hundred microamps of current for loads up to $100k\Omega$ using LM134[3] Texas Instrument adjustable current source, supplemented with two digital potentiometers, AD8400[2] in $1k\Omega$ and $10k\Omega$ variants to control the closed loop tunable current source.

Combining these two halves was yet another challenge, as both have non-infinite output resistances, effectively loading each other if not explicitly separated. One approach was to use an analog multiplexor that could select either of the two outputs to be presented with a mystery load. This would entail a two step process, the first of which to measure the impedance of the attached load, and then switch to the current source to stimulate the load if needed. A simple double-pole-double-throw switch was used to accomplish this. While this approach assumed only one load, given the context, this would need to be expanded to fit the larger fifteen array printed electrode schema. Additional hardware, namely more multiplexors, would be needed in order to address two specific electrodes to take impedance measurements from and drive. Given that the maximum number of electrodes was fifteen, I utilized two sixteen by one multiplexors to be able to address each end point electrode individually. The output of these two electrode addressing multiplexors could then be fed into the double pole double throw switch that could be addressed to select either the impedance analyzer or current source. Figure 2.2 illustrates an overview of the system with the electrode multiplexors omitted.

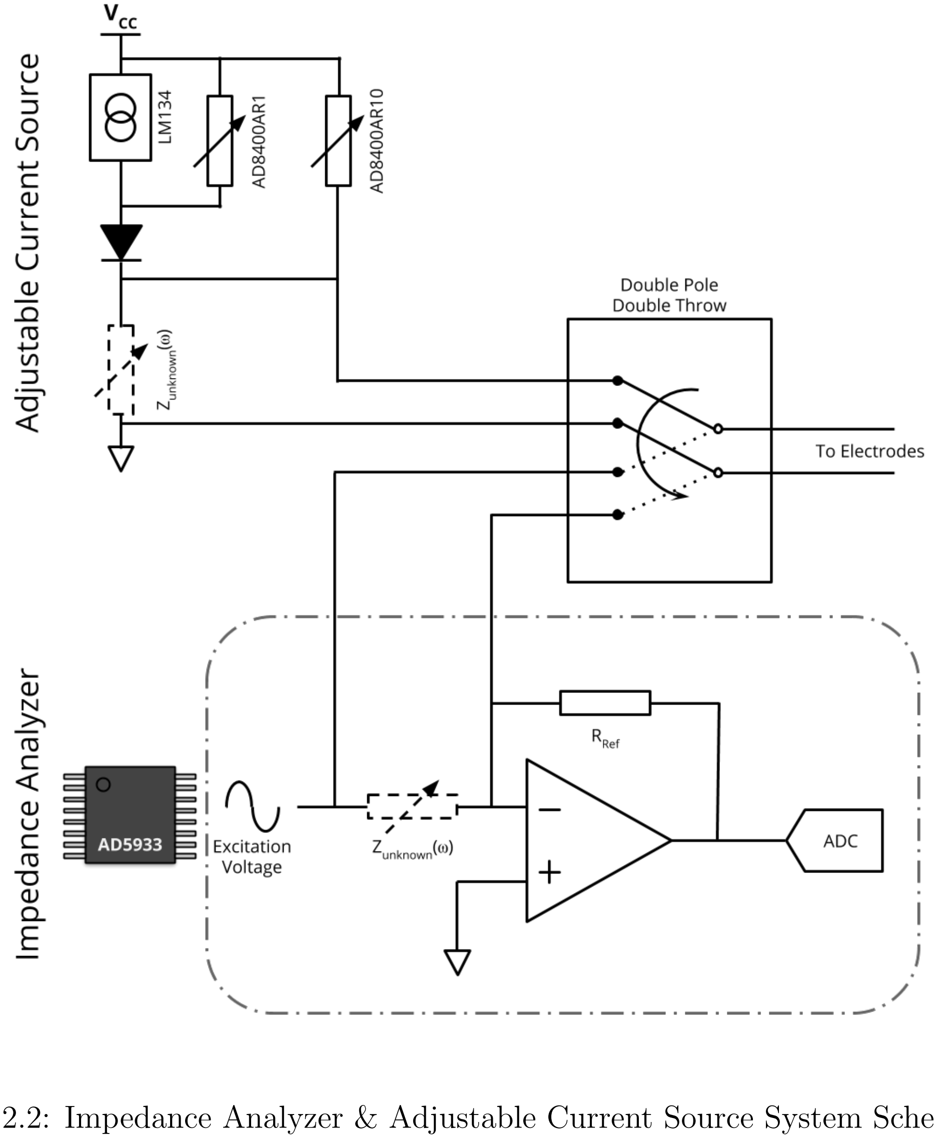


Figure 2.2: Impedance Analyzer & Adjustable Current Source System Schematic[1,2,3]

This project was the first to introduce an application specific system, which simultaneously imposed stricter limitations and gave clear goals for exactly what should be measurable. The flexible device in question, the printed electrodes, proved again how important it was to be able to integrate more powerful solid state electronics, namely the analog front end for frequency sweeping and impedance measurements, to create systems that could keep up with researchers' demands.

2.3 Integrated Wound Monitoring

The integrated wound monitoring project was in part with NextFlex, and required a deliverable that integrated various organic electronics into a flexible hybrid electronics system that could monitor both wounded tissue impedance as well as oxygenation levels using photoplethysmogram (PPG) measurements. It featured our printed flexible electrodes to monitor wounded tissue and our organic optoelectronics, 2 OLEDs and 1 OPD, for the pulse oximetry. The goal was to have both of the monitoring electronics integrated into one small system that could be deployed as a wearable. As I have touched on our printed electrodes previously as well as the impedance analysis in section 2.2, I will focus mainly on the oximeter design for PPG measurements.

I will first go over the working principle of oximeters and how our optoelectronics play a role in the design. Pulse oximetry is a non-invasive method to measure oxygenation in blood. This is done by measuring the difference in absorption of two different wavelengths of light through body tissue to back calculate the oxygen content in blood. In a traditional solid state pulse oximeter, there are three key devices at play: two light emitting diodes (LED) and one photodiode, as well as a transimpedance amplifier to read out the photodiode output. The photodiode is responsible for measuring the light that gets transmitted through the blood, while the LEDs are responsible for producing the two different wavelengths of light. Because oxygen content in the blood affects the absorption of the two wavelengths differently, it is important to use wavelengths that are most sensitive to the change in oxygen content; the two most commonly used wavelengths are red and near infrared (NIR).

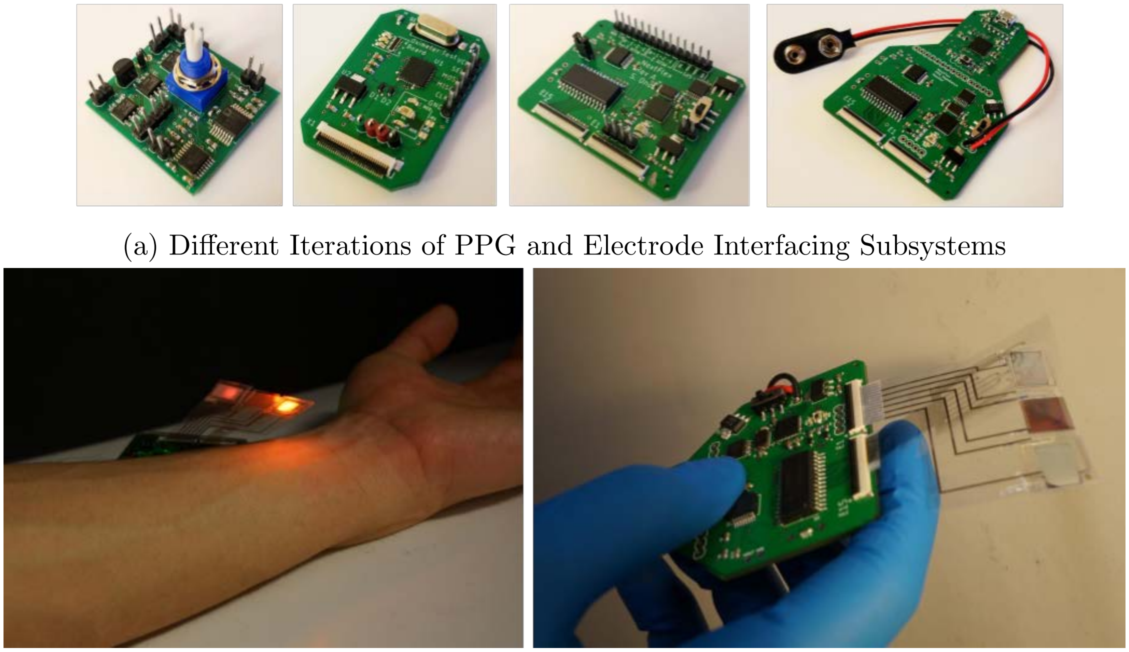
We had previously demonstrated the use of its organic LEDs (OLED) and organic photodiodes (OPD) in pulse oximeter applications.[6,8,9,11] Using these works as reference, I was able to reproduce my own pulse oximeter with the use of an analog front end AFE4490[7] from Texas Instruments. This chip was capable of driving LEDs at the needed thirty cycles per second while taking measurements from the photodiode to construct PPG waveforms. The difficulty of this subpart of the project was actually validating the hardware. The chip was packaged in a 40-pin VQFN, or Very thin Quad Flat No Lead, that hid away its contacts underneath the chip, making system fabrication and contact verification quite difficult. Many hours were spent closely inspecting contacts under a microscope to ensure good connection. After inspection, I tested communication and responsiveness of the system while connected to solid state optoelectronics, yielding promising results when stepping up to high power organic optoelectronics.

To transition from the prototyped systems into a full fledged deliverable, I had to first combine the two halves and gather data using a hardware backend that could send data over to a computer for processing. Unfortunately, combining systems is not as simple as lining them up nicely next to each other on a single PCB - there are a few key factors to think about before proceeding. The easy check is to make sure that there are enough communication channels available on the hardware backend to interface with both subsystems. This would entail simply counting the number of channels needed and cross checking both to make sure certain digital communication protocol buses do not overlap, and if they do, have a way to distinguish between the two. An example of this would be a shared Serial Peripheral Interface (SPI) bus which consists of four channels: host data, peripheral data, a clock, and a chip select. Each peripheral needs to use a separate chip select, else data from the host would go to peripherals that should not necessarily be listening, as they might execute commands that are unexpected, yielding unintended, non-deterministic outcomes. The data rate for both subsystems had a minimum sample requirement, which implied that important data could be lost if not sampled at a rate faster than the minimum. Unfortunately, the discrepancy between the two subsystems (impedance analyzer and pulse oximeter) was very large - the impedance analyzer had to operate very long frequency sweeps infrequently on the order of 1Hz, while the pulse oximeter had to take many measurements per second, on the order of 100Hz. This makes for scheduling challenges on a single threaded single core microcontroller when trying to avoid starvation while keeping up with the strict sampling requirement for the oximeter. Luckily, the goal was to make a system for long term, periodic monitoring, so the slowness would be acceptable, as the two operations could be separated in time, alleviating the necessity to interweave a scheduler to run both tasks simultaneously.

Figure 2.3 shows the system integrated with both the bioimpedance analyzer printed electrode array and pulse oximeter, fitted with a red OLED, NIR OLED, and an OPD. This was by far the most difficult integration to date - similar to the force calibration device, it was meant to go from end to end, from the device fitting and overlaying to the data collection on a host computer. Each layer in the system stack introduced more complexity, and the lack of device testing capabilities led to a starvation in tests until the day for full integration, leaving much to be desired in the design process. Though the bioimpedance printed electrode array analyzer was designed much earlier than this project's deadline, the lack of specification, testing equipment, and subjects led to uncertainty in the design, as stricter constraints would mean a tighter, robust system design that would most likely be more power hungry and expensive, affecting the choice of microcontroller. Another difficulty arose in the device integration - because organic electronics do not have a particularly long lifetime, exposure to the atmosphere would lead to performance degradation, both in the OLEDs and OPDs, making it very difficult to pinpoint the cause of the system's poor performance.

2.4 Developer Notes

For a moment, I would like to step back and give a personal perspective on what it feels like to be the integration developer. The research in this field of flexible electronics is typically



(a) Different Iterations of PPG and Electrode Interfacing Subsystems

(b) Final Formfactor with OLEDs, OPD, and Printed Electrodes

Figure 2.3: Fully Integrated System with Printed Electrodes and Organic Oximeter[1,7]

driven by the people who are in the fabrication lab making the devices and electronics first hand - the proposals and application focus can be guided by anyone, but the bulk of the cutting edge research is being done by creating an application for the printed flexible electronics. The supporting infrastructure is just as important, as there are no systems with only the sensors and flexible electronics, but to draw an analogy, no one focuses on the new plumbing and air conditioning systems in a new building - the immediate attention is brought to the architecture, the front facing interior design and the utilities. This isn't to say that the plumbing is not interesting, but is often a detail often ignored and underappreciated, as it is often the case that these systems have been constructed before without the flexible electronics, making the design aspect of the solid state platform feel a bit like reinventing the wheel.

Regardless of the interfacing flexible electronics, someone somewhere has made a system very similar to the one that is being designed - there exists little form of accomplishment when successful and crippling confusion when the system does not work the way a similar system was used in a different paper. To add to this, the system methodology is often not outlined in very much detail, glossed over in a few sentences and a system level diagram that can be quite unhelpful when development goes awry. To make matters worse, all developers have their own development environments that are curated to their own personal tastes, along with their preferences in types of hardware, PCB manufacture, and parts distributors, making the translation from one person's work to your own that much more cumbersome. Though each individual's experience may be different, I am confident that they have experienced at least a few of these points I have brought up.

After much contemplation, I decided to move forward with a new project of mine, a project aimed at alleviating part of the development cycle related to integration. The dream was to create a hardware platform that developers could use to prototype various systems that could easily interface with flexible electronics. While such a system would not be as streamlined or physically viable to be used as a final design, it would serve to give a starting point and lift the "proof of concept" step off of the developer and allow anyone to quickly test their flexible electronics integration. On top of that, I wanted to stick with a platform that could be further developed using environments that did not need personalized configuration, giving templates throughout each step in the design process for a true open source experience.

Chapter 3

Modular Integration Platform for Printed Electronics

3.1 Initial Ideas and Concepts

In the initial stages of conceptualization, I was weighing various aspects of the project that would eventually come around - this included end user usability, ease of use, learning curve, and effort that would need to be put in for someone other than myself, as well as my own personal fulfillment, implementation, and scalability with regards to the project. Two different approaches were conceptualized.

The first was a modular approach to design. Often, a huge time sink in design for systems is the chip selection and testing that needs to be done. For example, if a researcher requested a system with a quad channel differential ADC, I would need to peruse electronics vendors to find something that meets the project specification, as well as some personal preferences, including but not limited to package (ease of soldering and size), digital interface, and required passives. Because I would be the one producing the boards at a small scale, the chip packaging mattered a lot to me. Soldering QFNs over and over would not exactly be good for my sanity, much less reliable; larger chips would also not be great to use for systems that wanted to be made smaller. With the limited number of digital interfaces on a microcontroller, I had a strong preference for addressable communication, in this case, SPI or in some scenarios, Inter-integrated Circuit (I^{2}C). In the former case, the number of chips on the SPI line had to be limited to the number of available chip select pins as well as the data rates to avoid congestion. In the latter case, if I wanted to have multiple of the same chip on a single system, they needed to be addressable separately. The other important personal preference was the required number of passives. While capacitors, inductors, and resistors are quite small, it was often not preferable to have too many littered around a system, as it would limit the flexibility of the design. With a carefully curated catalog of common types of flexible electronic interfaces implemented in 2-D CAD software, the vision was to make a module library that had the chips, their required peripherals, and firmware all packaged in such a way that would allow designers to drag and drop premade layouts to put together larger systems.

While there were good ideas pertaining to module design, the truth of the matter was that it still did not fit in a tight enough mold, giving absolute freedom to the end designer when it came to layout and sourcing - I needed to place more constraints so there were fewer decisions that needed to be made. I held onto many of these design principles through my project, but decided to move away from this in-CAD module design approach.

The second idea I had was a bit further from the design of the system, but a catch all system that could aid in the characterization and paper writing for researchers in the flexible electronics field. It would be closer to a lab device characterization data acquisition tool that could also automatically generate plots over parametric sweeps and extract device parameters. One of the immediate challenges that came to mind was the limitation on device multiplicity; would the system be able to hand larger device substrates that held a large number of devices? If the approach taken accommodated this requirement, it meant that the end data acquisition tool would house a large number of electronics that would all need to communicate to a singular host in some manner. This could potentially be a lot of data being sent back and forth between a central host and its many peripherals, which would inevitably become congested if not designed with the utmost worst case design. While this problem can be solved by having an extensive project budget, that would go against the idea of having open source and accessible hardware. The idea of creating flexibility for the end user was an idea I held onto, but once again moved away from this idea as a whole.

3.2 Firmware Platform

There were a few subsystems that would definitely need to be flushed out before any final product would be prototyped, namely the microcontroller and programming environment. I had exposure to three different environments prior to this: Arduino, mbed, and Segger Studios. In my eyes, the difficulty ramp was in that order, Arduino being the token introduction-to-embedded-systems platform and Segger Studios being catered to more experienced users and industry. To briefly go over the three development environments, Arduino comes as an install package that can quickly interface with any of its development boards and upload sketches directly from the IDE, along with the ability to add user defined libraries and cores to interface with non-Arduino enabled boards and chips. The process to do this is usually as easy as finding a repository online and copy pasting one link into the IDE in order to add it to the compile path for a particular board, which is done automatically. mbed features an online compiler that can compile for any of its supported boards with any user library, similar to Arduino, with the added benefit that it can all be hosted online, only requiring the user to drag and drop a compiled output file into their local devices. Segger is aimed more towards a streamlined, custom development environment like many software programming IDEs, with many powerful compilation tools and access to a large library, but with the downside of everything needing to be configured locally, making for local environments to be tailored to each user's needs. While this does have a place in large scale, industry level development, I wanted something either Arduino or mbed could support, as they each require very little setup that can be either exported or documented in a very simple manner. The choice between the two came down to personal preference, as I had a decent amount of experience in both - but the choice would not just affect myself, but anyone who wanted to potentially use the platform to develop. Hence, I went with Arduino, being much more user friendly, though often lacking in performance power with their user tailored libraries.

3.3 Hardware Platform

With the firmware platform set, I created custom hardware that could interface with this platform to define a set of chips I would use. The Atmel ATMega microcontroller was chosen as a base. Looking through their datasheets and other open source projects that used them, the first chip I wanted to create a scalable platform for was the ATMega328P. This chip is used in a few of Arduino's development boards as well, making the integration between my hardware and Arduino as simple as flashing the special Arduino bootloader, which the IDE can do for me after setting up special hardware connections, leaving me to resolve how I was going to get the bootloader on the chips. Atmel AVR has a few solutions for this, using either In-circuit Serial Programmer (ISP/ICSP) or Joint Test Action Group (JTAG) pins to access the memory that holds the firmware. While these solutions are nice, it requires me to place large pinouts on every single board that will only end up being used once, consuming board area, vertical space, all at an increased cost. An argument could be made here that these specific pinouts and flashing solutions also come with the added benefit of doubling as debugging tools, but in the end, the formfactor was a higher priority, and without the use of bulky connectors, could trim down the size of each subsystem. I needed an impermanent solution that did not require populated components, and designed my own tool that utilized ATMEL ISP and Arduino's bootloader burning functionality.

As shown in Figure 3.1, I designed a self aligning ISP that can flash a bootloader on any Atmel chip using pogo pins and contact pads. The burn in tool is a custom designed Arduino Uno shield that is 3.3V safe, loaded with "Arduino as ISP" firmware, capable of bootloading chips connected to its ISP pogo pin headers. This way, the contacts on the bottom side of the board will not take up much board space with no need for a populated component. The burn-in tool's alignment holes and contact pads have been made into a device that can be used in EAGLE 2D CAD for ease of development. As long as there are few underside extrusions, contact can be made. As a side note, the ISP digital logic voltage being set to 3.3V does not disallow it to flash 5V devices, unless the chip's operating voltage must be at 5V; the fuses are set during the operation and will not trigger a fault until the device is powered at a later time. Once I had a firm grasp on how I would be able to flash a bootloader onto my custom designed microcontroller boards, the next step was to design around the burn-in tool.

Three different variants of the ATMega328 (328P-PU, 328P-AU, 328PB-AN) were chosen

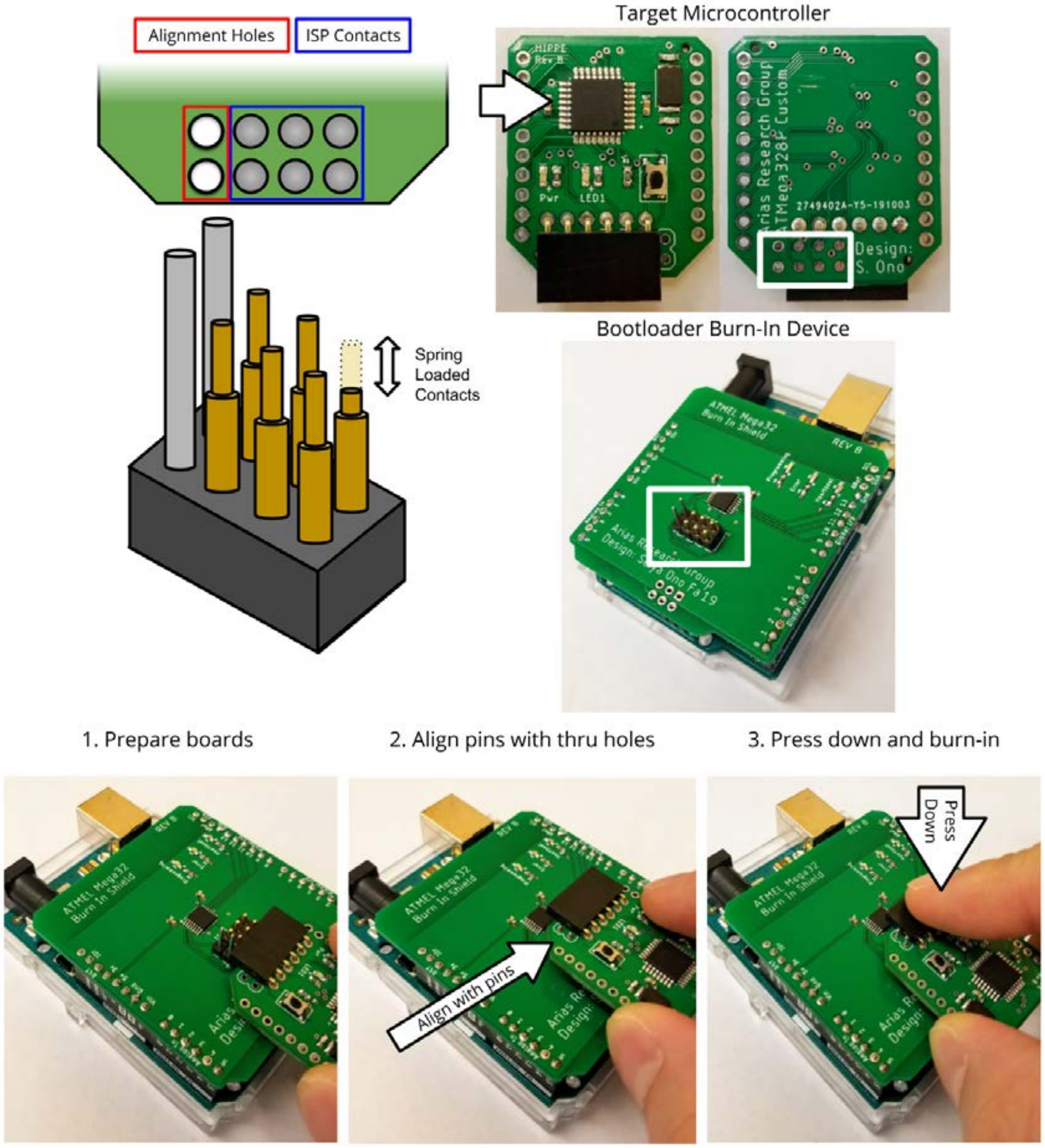


Figure 3.1: Bootloader Burn-In Tool

where each require slightly different bootloaders, and the ATMega32u4-MU (see figure 3.2 for each of their design). An interesting thing to observe from the image is the ATMega328's lack of USB capabilities, adding the need for a USB to Transistor-to-Transistor Logic (TTL) adapter. A very common choice for this is the Future Technology Devices International (FTDI) chips that can convert USB 2.0 into serial data that can be read over serial Universal Asynchronous Receiver-Transmitter (UART). Each of these boards were designed to have the least number of components possible while getting most if not all the functionality out of the microcontroller.

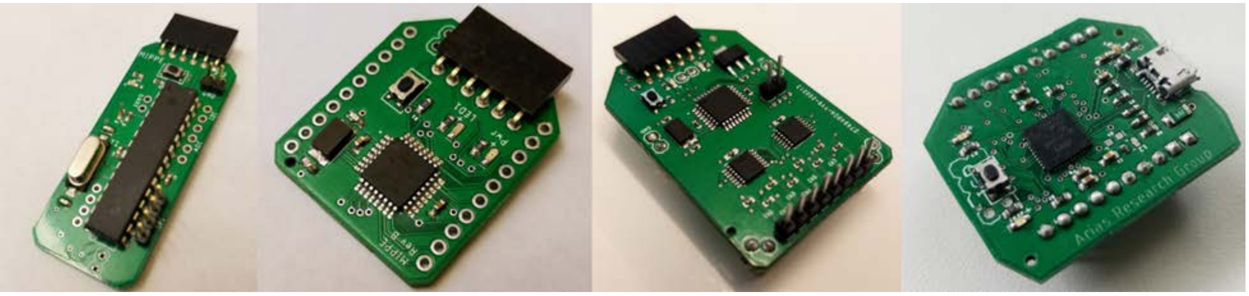


Figure 3.2: Various Implementations of Atmel Microcontrollers on Custom Hardware

3.4 Small Scale Manufacturing Choices

As discussed in chapter 2, the fabrication time that goes into designing solid state systems is neither cutting edge or exciting, with the only possible outcomes being a short lived sense of satisfaction or frustration that arises from faulty solder joints. In this section, I would like to outline my manufacturing selection, the process specific to it, as well as its pros and cons.

A common service that industry and small scale developers use is turn-key PCB assembly. Instead of ordering a PCB that comes unpopulated, the PCBs arrive with some if not all components already soldered on; this comes at an additional cost that can get quite expensive as well as an increase in turnaround time. My manufacturer and turnkey service is JLCPCB, a fabrication foundry based in Hong Kong that offers a variety of services - I will be focusing on their surface mount (SMT) assembly service.

JLCPCB sources their SMT catalog from LCSC, an electronics distributor based in Shenzhen, and houses a huge selection of parts, from standard passives to more application specific IC's. This leads to the second reason why I went with JLC, as they have the ability to pick and place the ATMegas that I was planning on using. This opens the possibility for me to use packages that are otherwise much more difficult to hand solder, such as ball gate arrays (BGA) and quad flat no-leads (QFN). The smaller packages freed up more board space for me to use in my designs with basically no additional cost. I decided on the 32 pin quad flat package (QFP) for the ATMega328, and the 44 pin QFN for the ATMega32u4. Another great feature of JLCzPCB was the ability to visualize the board and all the parts on it before placing the order. If the component placement list (CPL) and bill of materials (BOM) files are populated correctly, JLC will show all the parts and their orientations on a virtual render of your board.

All the components were soldered with no problems, enabling me to go from unboxing to flashing bootloader within ten minutes, only needing to populate a few components that JLC did not offer. This fast paced development without the need to solder helped make my boards much more reliable. All boards designed in the project were designed around JLC's manufacturing and turn-key service to enable quick development and expansion.

3.5 Final Conceptualization

In the final design, I explore the two approaches to find an agreeable middle ground in order to help facilitate a design and prototyping platform that could help ease R&D efforts for flexible and printed electronics researchers. The end product was a platform that could serve to quickly test flexible printed electronics while simultaneously providing enough complexity and robustness to create bulky but functional prototypes of envisioned systems that can feature multiple types of flexible electronics working in tandem.

Revisiting the ideas from earlier, I evaluated the two design approaches: a modular design platform and a catch-all solution device characterization device; both had their glaring flaws that would prove to be too much to overlook, giving rise to a need of a good middle ground that could inherit the good design principles from both, while easily understood and usable by other researchers. The intermediate idea I had was to use the modularity in design, but restrict it to the physical realm of hardware, not in CAD. This would entail being able to physically move certain component groups around and rearrange them to make larger systems. The envisioned system, pictured in figure 3.3, captures the overview of how I imagined the system.

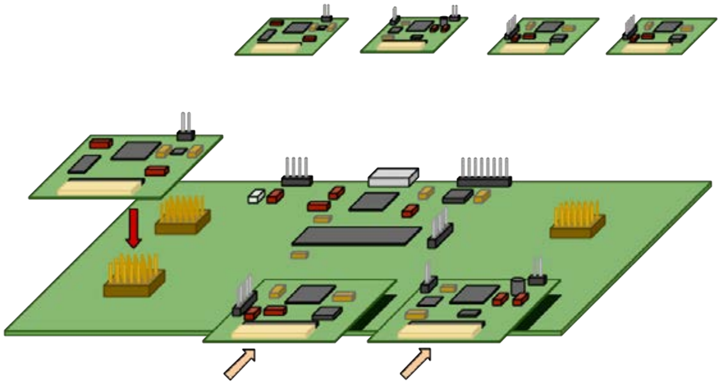


Figure 3.3: First System Overview

It would consist of largely two types of parts - the motherboard and the modules. The motherboard is the larger PCB that has connectors that can house the different modules. The motherboard would serve as some central hub where all the data would come through to interface with a computer, aggregating and scheduling the various modules to collect data. Because the motherboard would be interfacing with the computer, it was important for it to have a USB connection; this made the ATMega32u4 a smarter choice for the motherboard.

At the time, I was not set on a particular type of physical or digital interface between the motherboard and the modules, but I knew that I wanted to have the modules plug into the motherboard for both power and communication.

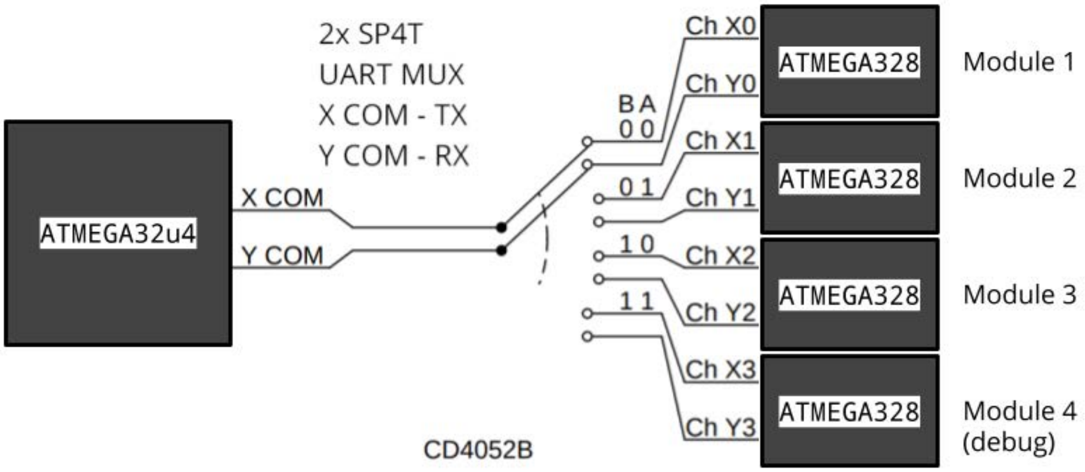


Figure 3.4: Leader-Follower Configuration for UART[5]

Module Communication

Understanding the limitations of the digital communication hardware enabled on the microcontrollers, I knew it would be difficult to set up a strict host and peripheral system like I^2C or SPI. For this reason, I settled on a peer-to-peer friendly communication protocol, UART. UART traditionally can only support single end point communication, meaning that at most, two chips can talk to each other. This is primarily due to the fact that UART only uses two communication lines, a transmit and receive, named exactly for their purpose. Immediately, this makes it sound like it would not work for a system that had a pseudo-host and multiple peripheral modules. To create a distributed communication network using UART, I added a host controlled 4:2 multiplexor so the leader could choose which module to talk to, shown in figure 3.4. This actually introduces yet another problem - the peripherals are never allowed to initiate meaningful communication with the host even though the protocol does not explicitly disallow this. In other host-peripheral communication protocols, the only way a peripheral can send information to the host is when the host asks for information; said a different way, the peripheral cannot actually speak up on its own. In this UART multiplexed system, there is nothing prohibiting the modules from speaking up and losing data. To build communication reliability, I had to implement my own standard for this system, modeling it off of how I^2C or SPI protocols are implemented. Every module's firmware is designed to handle UART interrupts in a manner that emulates the call-response nature of the aforementioned communication schemes.

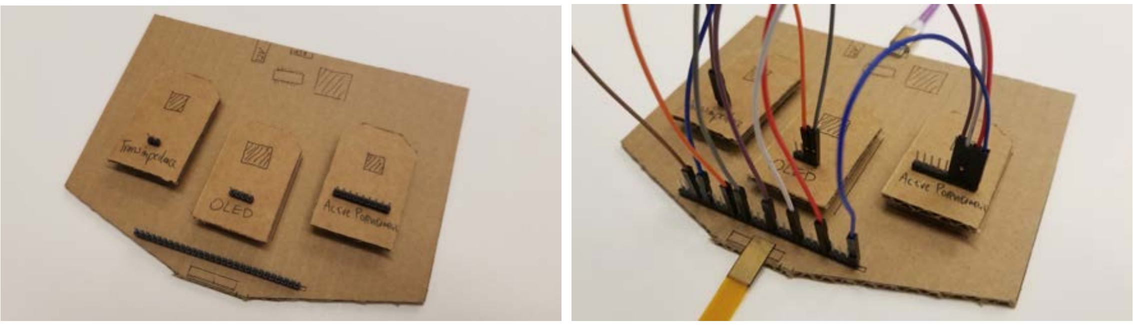
```
// Parsing UART Interrupt - only triggers on incoming data from leader
void serialEvent() {
  size_t len = Serial.readBytesUntil('\n', buff, 50);
  uint8_t func = find_func(buff, len);

  if (strncmp(buff, "req", func) == 0) {
    req();
  } else if (strncmp(buff, "info", func) == 0) {
    info();
  } else if (strncmp(buff, "start", func) == 0) {
    start();
  }
}
```

Listing 3.1: Follower side Motherboard-Module Communication Standard

Module Interfaces

Each module would be enabled to support one specific type of device interface: for example, one of the modules could support OPDs, having an adjustable reverse bias voltage transimpedance amplifier with high sampling rates. By containerizing each device type, this enables users to hand pick what types of device interfaces they want to use to test in parallel. In principle, this is what I wanted to accomplish, but there was one larger flaw in this system that I had to address. Because each module had one FFC on it, this meant that it would be very difficult to test substrates that housed multiple devices. Additionally, because the inter-PCB connections would already be set to specific pins on the FFC, it took away the design freedom from the device research for where their device contacts would go. Though I explained how certain freedoms were not favorable in this project, taking away freedoms from the device architecture is not a compromise that research should need to make when designing around a system whose goal is to provide a modular, flexible experience. Hence, a larger change had to be put into place before designing the modules.

Resolving this issue took some back-to-the-basics physical prototyping that would be best done using cutting edge prototyping equipment: cardboard. Before ordering expensive PCB prints that would take more than a week to arrive, I went for the easiest, quickest way to test out different approaches to solve this problem. The root of the problem was in the FFC; if the end user did not have absolute control over how they wanted to define every single interconnect, the development platform would basically be useless. Going back to the OPD module example, if the user wanted the cathode to be 4 pins wide from pins 4 - 7 and the anode to be 2 pins wide from pins 10 - 11, they should be able to choose exactly where each pin on the FFC should go. This level of flexibility is very difficult to implement in hardware, as it requires all 32 pins of the FFC to be able to fan out to any pin on all modules, which can have a varied number of pins. A quick calculation would show that for 32 pins connecting to three different modules that have four pins each would yield more than 10,000 permutations for how the end user would want the pins to be connected. This degree of freedom was not something I was prepared for and did not want to deal with, so I put all of that power in the hands of the user, shown by the cardboard prototypes in figure 3.5.



(a) Select Modules

(b) Connect Devices

Figure 3.5: Cardboard Prototype of Module-Motherboard Interconnects

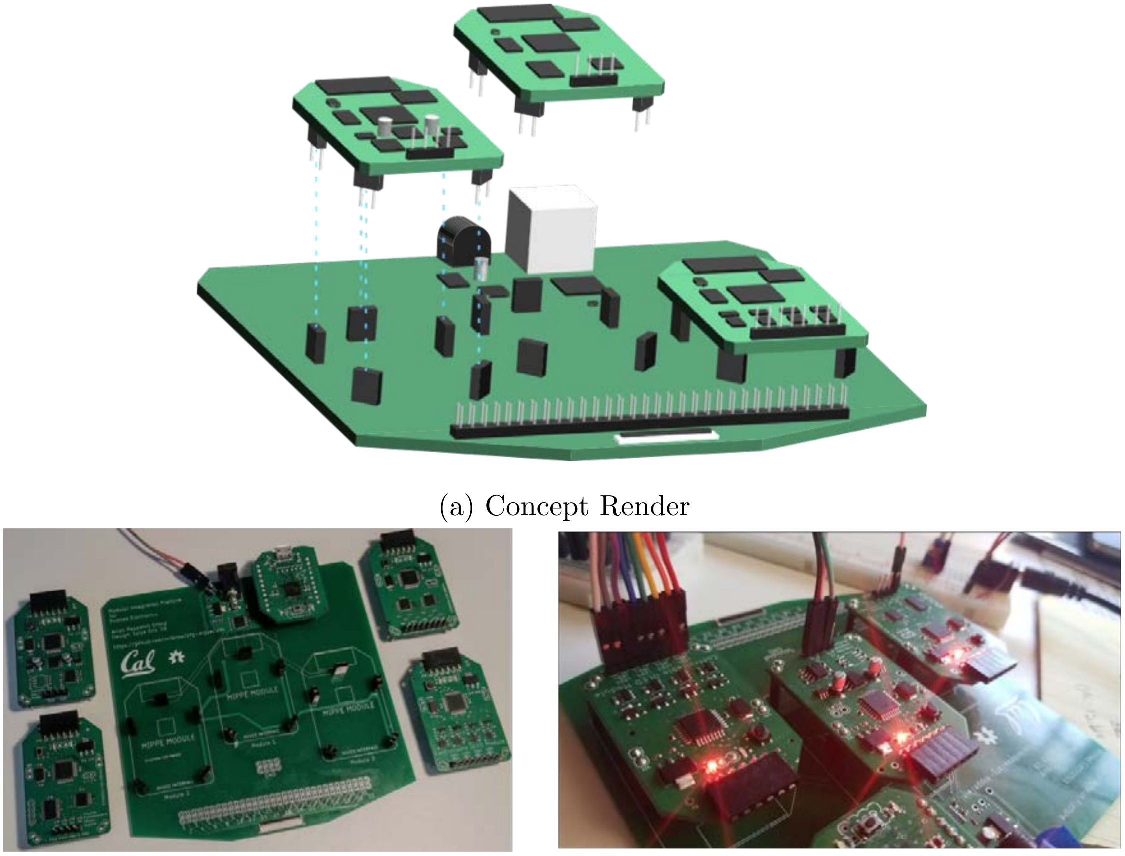
All pins of the FFC are broken out as exposed pins on the motherboard, allowing the user to connect them to whichever pin on whatever module they would like, eliminating the need for the system to handle the gross number of connection permutations. After briefly consulting my lab mates about this design, I moved forward with form factor, with the finalized system design coming out to figure 3.6. Here, the entire system has only one FFC interface with its pins broken out, along with the familiar three module setup that can be plugged in via five dual pin headers, delivering nine volts, ground, and UART that go to the back end of the motherboard to talk to the host chip.

3.6 Modularity in Design

The completed system is indeed modular - the end user has full control over what modules to use (albeit limited to using three of them) and still has full freedom over how the pins on the FFC will be used. However, the true power of modular design should not end at the application, but should also be embedded in the design. All three aspects of the design process, hardware, firmware, and software, have been packaged nicely in a way that allows new types of interfaces to go from prototypes to working modules on the system very smoothly.

Hardware Design

To keep all the modules roughly the same design, there is a template layout that has all the required components sans the analog front end that would interface with the devices. This includes the exact form factor, connectors, ATMega328, as well as any required peripherals that are needed for power and standard operation. To prevent the modules from getting too large, the area for the analog front end is slightly limited, but has enough space to include a few chips. On top of that, the board comes with a BOM and CPL file that can be directly



(a) Concept Render

(b) Implemented System

Figure 3.6: Final System Design

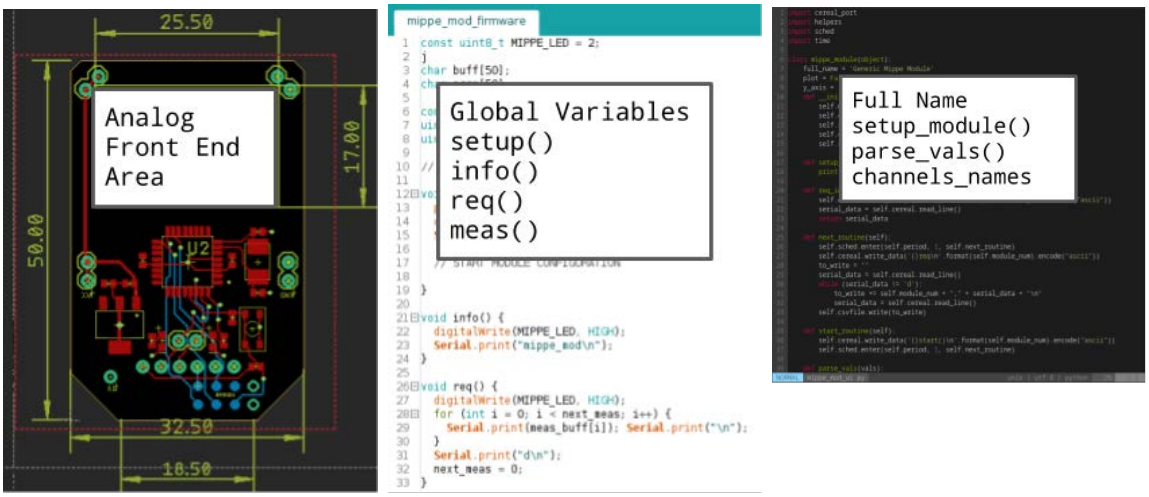
used with JLCPCB, making the design and order process much smoother. The designer can prototype the analog front end on any generic ATMega328 Arduino development board and translate it into a PCB with ease.

Firmware Design

Once the design is complete and the order is received, the module needs to be flashed with the proper bootloader and firmware that would enable it to communicate with the motherboard. There are, again, a few restrictions that come into play here to ensure a stable communication, such as data transfer rates and latency. An example firmware file can be found with guiding comments, making porting in previously used prototype code very easy. After implementing the required functions, a new module can be ready to communicate data back and forth between the host very quickly.

Software Design

Finally, when the entire module is prepped to be tested, it needs to have a way to talk to the user interface of the entire system, thus the need for a modular software implementation of the different types of modules. This is done all in Python through the use of inherited classes; there exists a super class that serves as the baseline for what all module classes need to implement to interface properly with the UI.



```
const uint8_t MIPPE_LED = 2;
j
char buff[50];
ch



co
ui
ui

//

vo




    // START MODULE CONFIGURATION

}

void info() {
  digitalWrite(MIPPE_LED, HIGH);
  Serial.print("mippe_mod\n");
}

void req() {
  digitalWrite(MIPPE_LED, HIGH);
  for (int i = 0; i < next_meas; i++) {
    Serial.print(meas_buff[i]); Serial.print("\n");
  }
  Serial.print("d\n");
  next_meas = 0;
}
```

```
import cereal_port
import helpers
import sched
import time

class mippe_module(object):
    full_name = 'Generic Mippe Module'
    plot = Fa
    y_axis =
    def __in
        self.
        self.
        self.
        self.
        self.

    def setup
        print

    def req_i
        self.                                                  "ascii"))
        serial_data = self.cereal.read_line()
        return serial_data

    def next_routine(self):
        self.sched.enter(self.period, 1, self.next_routine)
        self.cereal.write_data('{}req\n'.format(self.module_num).encode("ascii"))
        to_write = ''
        serial_data = self.cereal.read_line()
        while (serial_data != 'd'):
            to_write += self.module_num + ',' + serial_data + '\n'
            serial_data = self.cereal.read_line()
        self.csvfile.write(to_write)

    def start_routine(self):
        self.cereal.write_data('{}start{}\n'.format(self.module_num).encode("ascii"))
        self.sched.enter(self.period, 1, self.next_routine)

    def parse_vals(vals):
```

(a) Hardware (b) Firmware (c) Software

Figure 3.7: Design Modularity of MIPPE Modules

With proper scaffolding for all three design aspects, I hope that creating new modules can be done at a much faster pace than developing individual systems for each application. This way, designs can be shared across this platform eliminating the need for many developers to reinvent the wheel.

3.7 Application Information

The user interface is written completely in Python, utilizing `PySerial` to handle all USB communication, `numpy` and `scipy` packages for math and signal processing, `sched` for scheduling events, along with other common packages (`time`, `sys`, `glob`) for utility. The physical system is represented as `class MIPPE(self, serial_port, scheduler, csvfile)` that takes three different objects to hold as pointers for their respective tasks. During setup, it pings all three of the modules for `"info()"`, for which all modules are programmed to respond with their module type (`actpot`, `paspot`, `oled`, `tia`). The `MIPPE` class then makes a module object corresponding to the module that was discovered, which proceeds to walk through its own setup. After all the loaded modules have been set up, the user can start the data collection and let the system run.

Listing 3.2 demonstrates how a generic `MIPPE` module is implemented. There are a few class specific attributes that the designer will have to define, including its full name, whether or not a plot for it is necessary, and the name of the y-axis of said plot. When the module is initialized, it goes through a setup, noted in the last line of the `__init__` function. Here, any sort of user configuration can take place, from setting reference resistors, gain adjustments, and naming the different channels. Here, the module also needs to write all of this user configuration into the CSV file so that anyone viewing the data afterwards can still identify all the channels. The other functions that need to be defined are `req_info()`, `next_routine()`, `start_routine()`, and `parse_vals()`. `req_info()` is a function that asks the module for its module type. This is typically useful for hot plugging, if the user decides to swap out the modules before refreshing the module list. `next_routine()` and `start_routine()` go together, by sending the start flag to the modules to begin taking measurements and then adding the routine to the scheduler to periodically request from the module's data buffer. Looking closely, `next_routine()` will first add itself to the schedule again to run after a whole second, and then request a data dump from the module. It reads the data line by line, saving the data in the CSV file, preceded by its module number.

One of the important points is how the data communication is scheduled - each module is expected to have, implemented, a buffer that can hold at least one seconds worth of data. This is to free up the application's scheduler, as the simple scheduler that is implemented cannot handle lateness, else it ends up in a perpetual state of lateness, unable to recover. While it is true that this downside can be made a non-issue by either having it run on a slightly more powerful computer or through the use of multi-threading, I wanted to make the application very deterministic; this would make the debugging easier and thus, introduce predictable issues if a new module is not implemented to standard.

Implemented Modules

Active Potentiometric Module

The active potentiometric module can directly measure potentials, such as on chemical sensors that produce chemical potential between electrodes. It can measure up to eight channels from $-2.048V$ to $2.048V$ 10 times a second at 16 bit resolution.

Passive Potentiometric Module

The passive potentiometric module can measure resistive sensors using a voltage divider setup. The reference resistor is a tunable eight bit digital potentiometer, with the voltage division connected to the same ADC as the active potentiometric module. It can measure similarly 10 times a second at 16 bit resolution, but for four channels.

```python
class mippe_module(object):
    full_name = 'Generic Mippe Module'
    plot = False
    y_axis = 'Data'
    period = 1
    def __init__(self, num, cereal, scheduler, csvfile):
        self.module_num = str(num)
        self.cereal = cereal
        self.sched  = scheduler
        self.csvfile = csvfile
        self.setup_module()

    def setup_module(self):
        print("Setting up Generic Mippe Module")

    def req_info(self):
        self.cereal.write_data('{}info()\n'.format(self.module_num).encode("ascii"))
        serial_data = self.cereal.read_line()
        return serial_data

    def next_routine(self):
        self.sched.enter(mippe_module.period, 1, self.next_routine)
        self.cereal.write_data('{}req\n'.format(self.module_num).encode("ascii"))
        to_write = ""
        serial_data = self.cereal.read_line()
        while (serial_data != 'd'):
            to_write += self.module_num + "," + serial_data + "\n"
            serial_data = self.cereal.read_line()
        self.csvfile.write(to_write)

    def start_routine(self):
        self.cereal.write_data('{}start()\n'.format(self.module_num).encode("ascii"))
        self.sched.enter(mippe_module.period, 1, self.next_routine)

    def parse_vals(vals):
        return [x * 1.0 for x in vals]
```

Listing 3.2: Generic MIPPE Module class

High Power Management Module

The high power management module is used to supply a high voltage to a device, namely OLEDs. It goes through a calibration process that creates an eight bit mapping (256 values) to applied voltage by operating a sweep. This way, the applied voltage can be set very quickly after a calibration step, allowing for routines to be programmed to flash the OLEDs at very regular intervals.

Transimpedance Amplifier Module

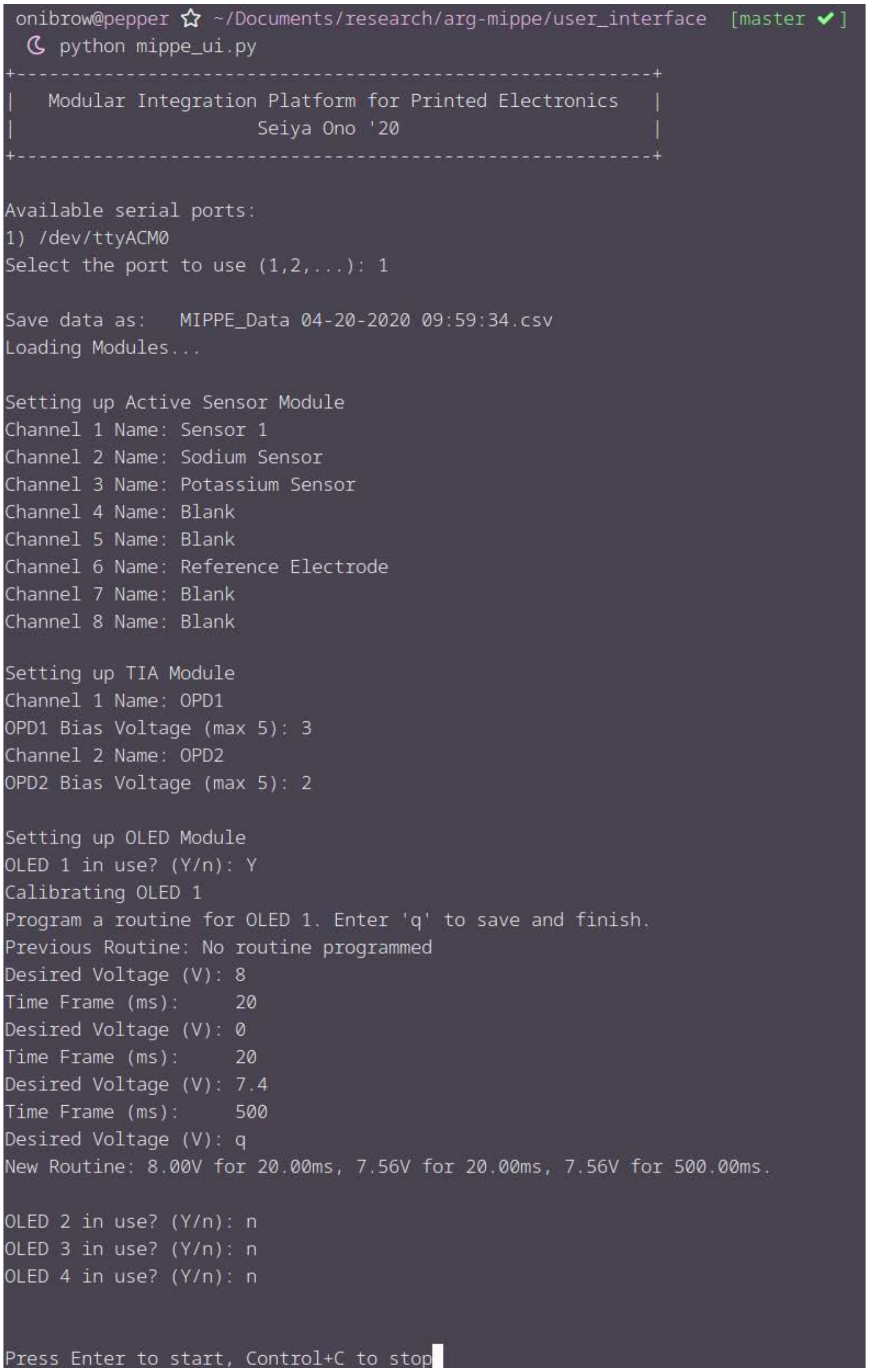
The transimpedance amplifier module, as the name implied, converts a current signal into a voltage through an amplifier, whose output gets fed into a 10 bit ADC. It can accurately measure from $50nA$ to $50\mu A$ ignoring noise. It can also supply a reverse bias voltage to an OPD using a 12 bit DAC to up its sensitivity.

UI Demonstration

Figure 3.8 is an example use case for the system. It has loaded the active potentiometric module, the transimpedance module, and the high power management module. As the UI shows, each module is being detected on the fly, and goes through each of the module specific setups. After the user hits enter, data collection can take place and is saved as a MIPPE formatted CSV file that can be used to automatically generate plots.

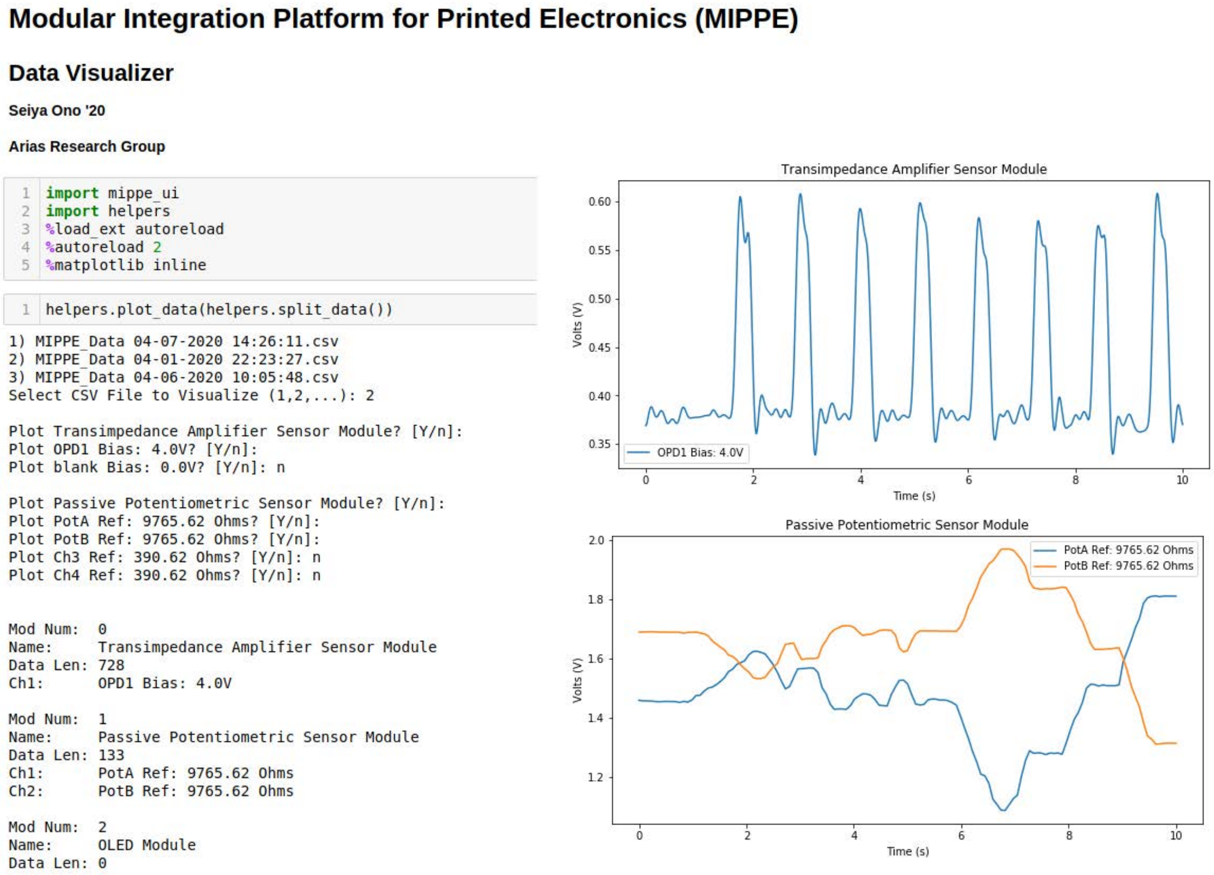
The autoplotting functionality is done through the use of `matplotlib`, creating color coded plots for multi-channel modules, auto labeling axis, and giving the user control over which channels they would like to plot. As shown in figure 3.9, the plots can also be visualized in an interactive format through Jupyter notebook, and then saved as images from there. It works by checking the CSV file loaded in to make sure it is in the proper MIPPE format that is autogenerated by the application, and then separates out all the data between the modules. In this example, there was a passive potentiometric module, a transimpedance amplifier module, and a high power management module powering LEDs attached to the system, with the LEDs pointing at a photodiode connected to the transimpedance amplifier. The routine can be observed on the transimpedance amplifier sensor module's plot, noted by the spikes in voltages, indicating that the LED was turned on. In the passive potentiometric sensor module plot, two channels out of the four were used to measure the resistance at the ends of a breadboardable potentiometer, showing the inverse relationship between the voltages seen at the ends.

This concludes the reach of my project, following its development as well as my own learning process through the year. I personally wanted to develop more modules for my system, but due to the circumstances that arose during this time, my hours in the lab were abruptly cut short and thus, shifted my focus to a more streamlined user interface that would be much more scalable. In the final chapter, I will discuss further improvements and possibilities for this project.



```
 onibrow@pepper ☆ ~/Documents/research/arg-mippe/user_interface  [master ✔]
  ☾ python mippe_ui.py
+-----------------------------------------------------------+
|   Modular Integration Platform for Printed Electronics    |
|                       Seiya Ono '20                       |
+-----------------------------------------------------------+

Available serial ports:
1) /dev/ttyACM0
Select the port to use (1,2,...): 1

Save data as:   MIPPE_Data 04-20-2020 09:59:34.csv
Loading Modules...

Setting up Active Sensor Module
Channel 1 Name: Sensor 1
Channel 2 Name: Sodium Sensor
Channel 3 Name: Potassium Sensor
Channel 4 Name: Blank
Channel 5 Name: Blank
Channel 6 Name: Reference Electrode
Channel 7 Name: Blank
Channel 8 Name: Blank

Setting up TIA Module
Channel 1 Name: OPD1
OPD1 Bias Voltage (max 5): 3
Channel 2 Name: OPD2
OPD2 Bias Voltage (max 5): 2

Setting up OLED Module
OLED 1 in use? (Y/n): Y
Calibrating OLED 1
Program a routine for OLED 1. Enter 'q' to save and finish.
Previous Routine: No routine programmed
Desired Voltage (V): 8
Time Frame (ms):     20
Desired Voltage (V): 0
Time Frame (ms):     20
Desired Voltage (V): 7.4
Time Frame (ms):     500
Desired Voltage (V): q
New Routine: 8.00V for 20.00ms, 7.56V for 20.00ms, 7.56V for 500.00ms.

OLED 2 in use? (Y/n): n
OLED 3 in use? (Y/n): n
OLED 4 in use? (Y/n): n


Press Enter to start, Control+C to stop
```

Figure 3.8: MIPPE User Interface Example



Modular Integration Platform for Printed Electronics (MIPPE)

Data Visualizer

Seiya Ono '20

Arias Research Group

```
import mippe_ui
import helpers
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

```
helpers.plot_data(helpers.split_data())
```

```
1) MIPPE_Data 04-07-2020 14:26:11.csv
2) MIPPE_Data 04-01-2020 22:23:27.csv
3) MIPPE_Data 04-06-2020 10:05:48.csv
Select CSV File to Visualize (1,2,...): 2

Plot Transimpedance Amplifier Sensor Module? [Y/n]:
Plot OPD1 Bias: 4.0V? [Y/n]:
Plot blank Bias: 0.0V? [Y/n]: n

Plot Passive Potentiometric Sensor Module? [Y/n]:
Plot PotA Ref: 9765.62 Ohms? [Y/n]:
Plot PotB Ref: 9765.62 Ohms? [Y/n]:
Plot Ch3 Ref: 390.62 Ohms? [Y/n]: n
Plot Ch4 Ref: 390.62 Ohms? [Y/n]: n


Mod Num:  0
Name:     Transimpedance Amplifier Sensor Module
Data Len: 728
Ch1:      OPD1 Bias: 4.0V

Mod Num:  1
Name:     Passive Potentiometric Sensor Module
Data Len: 133
Ch1:      PotA Ref: 9765.62 Ohms
Ch2:      PotB Ref: 9765.62 Ohms

Mod Num:  2
Name:     OLED Module
Data Len: 0
```

Figure 3.9: MIPPE Data Visualizer Example

Chapter 4

Future Work & Conclusion

4.1 More Powerful, Reliable Hardware

Microcontroller Choice

There were not many options when it came down to choice of microcontroller, considering the availability from the LCSC catalog as well as the availability of a development platform that could interface with the Arduino IDE. That said, there are plenty of newer development boards that have been released that support 32-bit ARM Cortex chips that have much more program memory, dynamic memory, and speed when compared to the 8-bit microcontroller that I had selected. This naturally comes with a slight increase in cost, but the trade-offs that can be made are much too big to ignore. If the modules could be translated to use larger, more powerful microcontrollers, this would give more flexibility in, for example, the number of channels they could interface with, as well as the sampling speed that they could achieve. An example of such a microcontroller could be the ATSAMD21, which can be found on a variety of different development boards. It can also be bought and placed by JLCPCB, making it a solid choice for any future development that can happen..

Power Management

At its current state, the MIPPE system as a whole uses the LM134[3] to regulate its $9V$ input down to $5V$ on the motherboard and all of the modules. While this linear DC-DC converter is very precise and can provide high amounts of current, it gets very hot due to its lossy step down conversion. To overcome this, there are a few methods that could more carefully manage power - select a buck-boost configuration with switch capacitor ICs to regulate power at higher efficiencies or USB-PD protocol to utilize the USB-C's high power capabilities to have do the power regulation as two examples.

More Modules

Some other modules I had in mind to implement include small scale impedance modules that use signal generators to excite reactive elements to calculate impedance, a transimpedance module that had a variable reference resistor and better filtering circuitry, an antennae module that could interface with printed antennas for transceiving capabilities, and a three terminal device parametric sweep to characterize and automatically plot different performance curves for transistors.

4.2 Scalable Firmware Environments

As it stands, the firmware on the modules works to support their respective analog front ends and sample data. However, these can be made faster and more reliable by reducing or removing the necessity for dynamically allocated heap memory. Embedded systems do not function too well utilizing a bi-directional growing stack, as memory abstraction does not exist like in operating systems. Additionally, most libraries that are being used are simple wrappers for chip to chip communication, save a few attribute variables that are kept for reference, meaning the dynamic element can be removed to optimize performance.

The development environment is also still tied to the Arduino IDE - while this is nice for a person getting into design for the first time, development environments can be packaged nicely to work in self contained environments, assuming proper compiler tools exist on the computer. Because of the limited selection of chips, the entire `Make` process can actually be made much smaller than the existing compile target directories that come included with Arduino. Creating a fast deploying programming environment is again something that can be done to lower the entry barrier for people to write firmware.

4.3 GUI & Software Optimization

The software environment, on the other hand, is much easier to handle, as all it requires is a few `pip` packages that can be included as a requirements file. However, it is all currently implemented as a command line interface utility, which is not exactly most scalable and easy to understand interface type. The creation of a dynamic graphical interface would make the application much easier to use - the user would be able to select parameters from drop down menus and text boxes rather than going through multiple layers of a command line.

The whole module scheduling is also not implemented in the most robust way, with a scheduler that perpetually falls behind once the first deadline is missed and is not exactly functioning as a scheduler. The data requests can be spread out utilizing multithreading, with each module writing to its own CSV file to rid the need of a specialized CSV format for the MIPPE Data Visualizer to read.

4.4 Final Words

My research adventure started with me stumbling into a new lab that I knew close to nothing about, fascinated yet confused out of my mind with this organic electronics field that I had never even heard of before. During my five semesters in the group, I watched graduate students and postdocs present on their work, asking questions I felt were too basic, frantically looking up polymers that I had never heard about. Over time, I learned about the complexity, the trade-offs, and fabrication difficulty that came with printed electronics, and began to fill a niche in the group, utilizing my previous experience with systems to help develop integration platforms with their printed organic electronics.

After being asked to participate in a few projects, I began to wonder what would happen once I had to leave, with no one to fill this role in the group. This thought soon developed into the initial ideas for the project, and eventually came into fruition in the form of this integration platform project. In a sense, I wanted to leave something for the group, something that could be used to make their research lives just a bit easier. While at the current stage, the project may not be complete enough to provide the amount of support I envisioned, but with a bit more work, I would like to see this sit in the characterization lab, ready for the next new revolution in printed electronics to come through the lab doors.

Bibliography

[1] *1 MSPS, 12-Bit Impedance Converter, Network Analyzer.* AD5933. Analog Devices. Sept. 2005.

[2] *1-/2-/4-Channel Digital Potentiometers.* AD8400. Analog Devices. Nov. 2001.

[3] *3-Terminal Adjustable Current Sources.* LM134. Texas Instruments. Mar. 2000.

[4] Mark Bohr and Kaizad Mistry. "Intel's revolutionary 22 nm transistor technology". In: *Intel website* (2011). URL: https://www.intel.se/content/dam/www/public/us/en/documents/presentation/revolutionary-22nm-transistor-technology-presentation.pdf.

[5] *CMOS Single 8-Channel Analog Multiplexer/Demultiplexer With Logic-Level Conversion.* CD4052B. Texas Instruments. Aug. 1998.

[6] Donggeon Han et al. "Flexible Blade-Coated Multicolor Polymer Light-Emitting Diodes for Optoelectronic Sensors". In: *Advanced Materials* 29.22 (2017), p. 1606206. DOI: 10.1002/adma.201606206. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adma.201606206. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201606206.

[7] *Integrated Analog Front-End for Pulse Oximeter.* AFE4490. Texas Instruments. Dec. 2012.

[8] Y. Khan et al. "Organic Multi-Channel Optoelectronic Sensors for Wearable Health Monitoring". In: *IEEE Access* 7 (2019), pp. 128114–128124.

[9] Yasser Khan et al. "A flexible organic reflectance oximeter array". In: *Proceedings of the National Academy of Sciences* 115.47 (2018), E11015–E11024. ISSN: 0027-8424. DOI: 10.1073/pnas.1813053115. eprint: https://www.pnas.org/content/115/47/E11015.full.pdf. URL: https://www.pnas.org/content/115/47/E11015.

[10] Yasser Khan et al. "A New Frontier of Printed Electronics: Flexible Hybrid Electronics". en. In: *Advanced Materials* n/a.n/a (), p. 1905279. ISSN: 1521-4095. DOI: 10.1002/adma.201905279. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201905279 (visited on 04/13/2020).

[11] Claire M. Lochner et al. “All-organic optoelectronic sensor for pulse oximetry”. en. In: *Nature Communications* 5.1 (Dec. 2014), pp. 1–7. ISSN: 2041-1723. DOI: 10.1038/ncomms6745. URL: https://www.nature.com/articles/ncomms6745 (visited on 04/16/2020).

[12] Aminy Ostfeld. “Printed and Flexible Systems for Solar Energy Harvesting”. PhD thesis. EECS Department, University of California, Berkeley, May 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-19.html.

[13] Mahsa Sadeghi et al. “Printed Flexible Organic Transistors with Tunable Aspect Ratios”. en. In: *Advanced Electronic Materials* 6.2 (2020), p. 1901207. ISSN: 2199-160X. DOI: 10.1002/aelm.201901207. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aelm.201901207 (visited on 04/13/2020).

[14] Xiaodong Wu et al. “Large-Area Fabrication of High-Performance Flexible and Wearable Pressure Sensors”. In: *Advanced Electronic Materials* 6.2 (2020), p. 1901310. DOI: 10.1002/aelm.201901310. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aelm.201901310. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aelm.201901310.