[1]

IKU Electronic Control Unit

Takuya Seaver, EE, Jack Walter, CSE, Yongjie Yang, EE, and Xueteng Qian, CSE

***Abstract*—As global temperatures are on the rise the need for new improvements in fuel injection is needed for today's vehicles. In order to achieve the goal of halting the effects of global warming, the IKU electronic control unit tests the limits of how many miles a vehicle can achieve on one gallon of gasoline. This device consists of two main components. The first component being the hardware that will be connected to the engine where it will convert information of one type to another, so that the other component, the software can process the data. The software will then send a command signal to the fuel injector. In the end, the fuel injector will spray fuel into the engine for the correct amount of time so that the engine can efficiently use its fuel source and get the most miles out of it.**

I. Introduction

A. Significance

Global warming is a topic that is in the news, on the internet, and affects the entire world. The most abundant greenhouse gas in our atmosphere is CO2[2]. The largest source of these rising CO2 emissions is in the transportation industry[3]. Until the electric car is made more attainable for people then lowering the amount of CO2 emissions created by the transportation industry is an intermediary solution.

B. Context and Existing Products

The UMass Super Mileage Vehicle team or SMV team for short, is a team of mechanical engineering students that test the limits of the modern-day combustion engine. The number of miles that can be brought out by one of these engines is mainly determined by the efficiency of the vehicle's method of mixing fuel and air to produce the optimal fuel-to-air ratio. Several options are available on the market today such as an off-the-shelf electronic control unit. Another option is to use a carburetor which mixes air and fuel to produce the desired ratio. These methods have been tried before, but what they lack is customization. Our solution differs in the fact that it allows for the SMV team to modify the device so that it can be tuned to their exact needs. As for the SMV team, their vehicle can have a fully tunable Electronic Control Unit (ECU) to get the most miles per gallon.

C. Societal Impacts

The SMV team would be our main use case. They need an ECU that can be modified, so that they can perform better at the yearly competition that they compete in[4]. This product could positively affect them by performing better than their current off-the-shelf ECU. One downside to our product is that it does not eliminate CO2 emissions. Our product will still be the control system for a fuel injection system that burns fossil fuels. There will still be some sort of CO2 emissions, even though not as much. This is seen as only an intermediary step to achieving less carbon emissions in our atmosphere.

D. Requirements Analysis and Specifications

We were given some details as to the requirements of the design. The system needed to be tunable and customizable. For this, there needs to be documentation provided. Documenting modifications and how to modify existing tables and data so that the vehicle can be tuned. This data is to be modified using information that is observed when the O2 sensor is mounted. The system needs to be able to take in information from the O2 sensor when needed for tuning. Safety is also an important factor. If the system fails and is not injecting fuel correctly then the system needs to be able to be turned off manually. Also, since the processes within the engine happen within milliseconds then our system needs to be responsive enough. The requirements and specifications are listed below in Table 1.

Requirement	Specification	Value
Tuning	O2 Sensor Mountable	N/A
Responsive	Duration	<=10ms
Safety	Switch	ON/OFF
Frequency	RPM Limit	Up to 6000RPM
Modifiable	Modify Tables	Instructional Documentation

Table 1: Requirements and Specifications

II. Design

A. Overview

As an intermediary solution, an Electronic Control Unit (ECU) is a good way to improve the modern combustion engine until electric vehicles become cheaper and more available. An ECU is an integral part of the throttle body fuel injection engine, because it can easily provide the accurate amount of fuel and air into the engine that increases the power. Besides, a carburetor is also a device that can mix air and fuel to inject the engine (*Appendix A*). Not many specifications of our design imposed significant tradeoffs. One

[1] T. Seaver from Plymouth, MA (e-mail: tseaver@umass.edu).
J. Walter from Westwood, MA (e-mail: jackwalter@umass.edu).
Y. Yang from Linzi, China (e-mail: yongjieyang@umass.edu).
X. Qian from Beijing, China (e-mail: xqian@umass.edu).

[2] See [1] in the References Section

[3] See [2] in the References Section

[4] See [3] in the References Section

that did was the responsiveness of our device. Having a 10ms worst case time limit of injecting fuel was difficult to work with. Some hardware just was not fast enough and when we were able to get the time below that threshold then we couldn't find an effective way of demonstrating it. The only effective way that we found was to use an oscilloscope. Even with this, we had to slow down our device to show that it could work for different pulse widths by adjusting the accelerator which was our synthesized Arduino signal.

Thus, we design an ECU. See the ECU block diagram in Figure 1. Firstly, we synthesized input signals because we have to simulate the signals from sensors. For the car, an ECU can read signals from the crankshaft sensor, O2 sensor, manifold air pressure sensor and intake air temperature sensor. After reading the input signals, then we have to design our ECU develop the software by using a microcontroller, in this case an ATtiny817[5]. Besides this, through an integrated circuit level shifter and driver, the control signal from the MCU can precisely control the fuel injector by accurate pulse width.

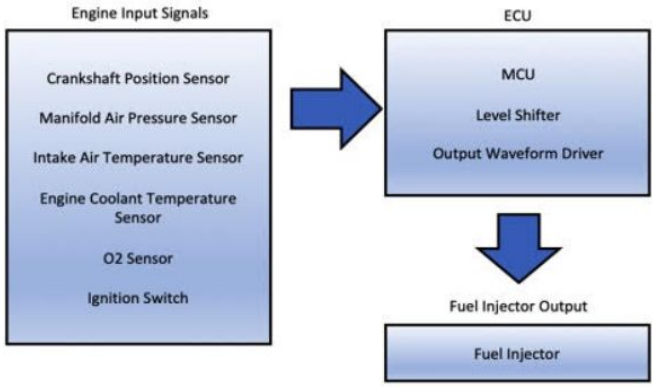

Figure 1: Block Diagram of ECU

For specifications in block 1, the ECU system should get a real-time processing of input data signals for fuel injection. And the system also should be able to get the feedback capabilities with the O2 sensor for optimization. In block 2, the microcontroller (MCU) can read signals supporting up to 6000 RPMs and able to have a correct pulse width to inject the correct amount of fuel within ~10ms for the specification. In addition, eventually, the hardware part of the level shifter and output waveform driver can easily and precisely be ready to drive a fuel injector by the control signal from the MCU.

B. Input Signals

Looking at Figure 1, the first big block is about simulated signals from four kinds of sensors and one switch. The crankshaft sensor measures the rotation speed (RPMs); the manifold air pressure (MAP) sensor measures the air density and air mass flow rate in the engine; the intake air temperature sensor monitors the temperature of the air entering the engine; the O2 sensor monitors how much unburned oxygen exits from engine. In addition, the ignition switch is a mechanical switch to control the open and close function of the engine to inject fuel. So, when we synthesized the signals, we used the Arduino Uno Rev3[6] to generate the input signals. To be specific, we also used potentiometers which are connected to the Arduino to modify the speed. From this part, we learned how to simulate the signals and how to write the Arduino code from what we learned in the course. Then we used an oscilloscope to watch the input signals when we changed the potentiometers for testing. *Details in Appendix B.*

C. MCU

Lookup Table

	LOAD							
RPM	0	20	25	30	40	60	80	100
1500	3.2	3.2	3.2	3.2	0	0	0	0
2000	3.4	4.8	5.2	6	7.6	7.5	7.5	7.5
2500	3.2	3.5	1.8	2.1	6	6.5	7.2	7.5
3000	3.2	5.2	4	3.5	4.8	5.5	6.5	7.2
3500	3.2	5.5	6.4	4.5	5.2	5.5	6.8	7.2
4000	3.2	4.4	4.7	5.1	6	6.2	6.9	7.2
4500	3.2	4.5	5	5	6.4	6.6	6.9	7.2
5000	3.2	4.7	5.1	5.3	7	7.1	7.2	7.3
5500	3.2	3.2	3.2	5.5	6.6	7	7.2	7.4
6000	3.2	3.2	3.2	5.4	6.2	6.8	7.4	7.8

Figure 2: Lookup Table of MCU

Looking at Figure 1, there is a microcontroller (MCU) in the ECU. One of the most important parts in ECU is to develop the software in our ATtiny817 microcontroller. First of all, looking at Figure 2, we have to consider the relationships between the data of the sensors to find the output pulse width in the lookup table. Then we write the C code by using the Atmel Studio 7 IDE[7]. Then the software can read the synthesized signals from the input and give a correct pulse width of the output. So, we accepted the coding idea from our embedded systems class. It really helps us to experience and understand the programming for the microcontroller. For testing this part, we also used an oscilloscope to watch the output signals when we changed the potentiometers. *Details see Appendix B.*

[5] See [4] in the References Section

[6] See [6] in the References Section

[7] See [5] in the References Section

D. *Level Shifter and Driver*

Figure 3: Level Shifter Waveform

From Figure 1, there are two parts about the level shifter and output waveform driver. The function of level shifter and driver is reading the control signal from MCU and giving the correct inverse pulse width to control the fuel injector, which is supported by a 12V power supply. It means when the voltage of the control signal is LOW(0V), the injector will inject fuel, and when the control signal voltage is HIGH (12V), then the injector will be closed and not inject fuel. Looking at Figure 3, we can see what the output of the level shifter looks like. We learned the method to design integrated circuits from our electronics course and this project provides an interesting experience for us. Especially for driving an inductive load, which is the internal circuit of the fuel injector. Then for testing this part, we used an oscilloscope to compare the input and output pulse width to see whether or not it has been inverted and has the same pulse width. Also, connecting the whole thing to a fuel injector to see if the integration of the whole system is working correctly.

III. The Product

A. *Overview*

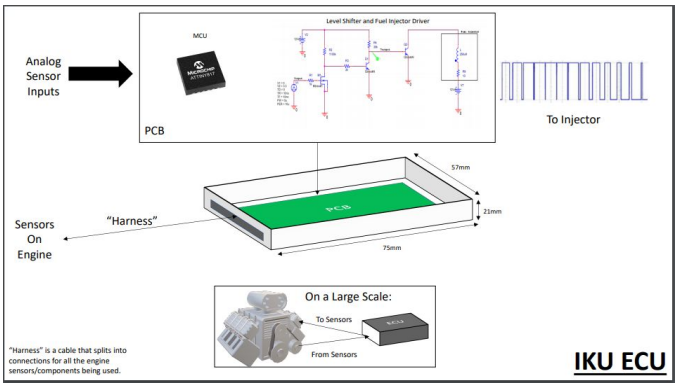

Figure 4: Product Sketch

Looking at Figure 4, the microcontroller, level shifter, and fuel injection driver are all on the PCB. This is similar to the block diagram provided in Figure 1. Through the harness, which is a bunch of input and output cables, we can send our output to the fuel injector and take in inputs from all of our input signals listed in Figure 1. This would be done by routing the correct input/output wires to certain pins which would be connected to the PCB. The output to the fuel injector is a pulse width modulated signal and the inputs to the PCB are analog signals. The output would change based on the "speed" of the vehicle which would be controlled by our Arduino based synthesized crankshaft position sensor. This change can be seen in Figure 5 and Figure 6 below.

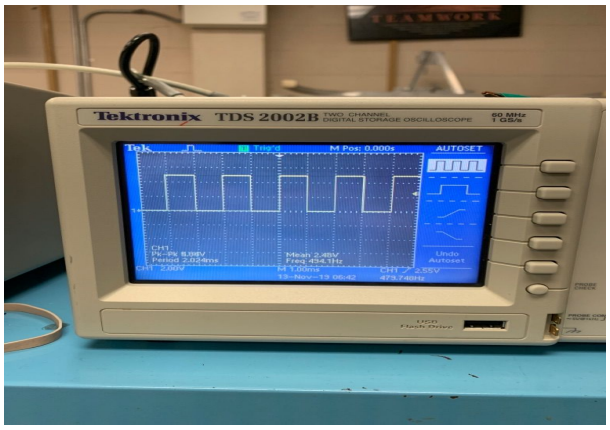

Figure 5: Output Before Changing Speed

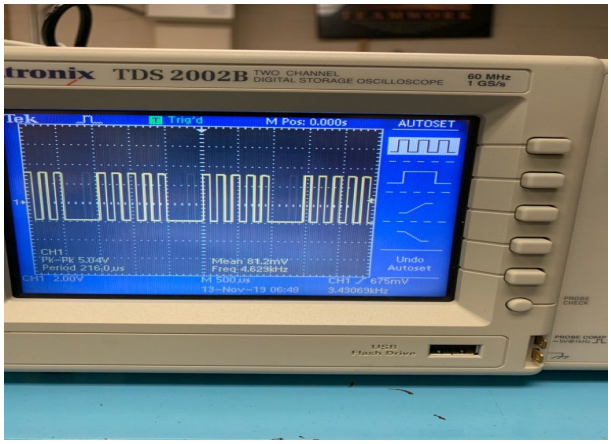

Figure 6: Output After Changing Speed

B. *Electronic Hardware Component*

The design of the hardware was difficult. Learning about how to use Altium was a challenge, but we were able to translate our working breadboard design to a PCB design. When we received the parts then we planned out how we were going to populate the board. We ended up designing two versions of the board. One with the MCU on it and one without the MCU. Our thought process was that the MCU was going to be the biggest challenge since we didn't know if using the Atmel-ICE debugger would work or not. The first design with the MCU on chip is shown in Figure 7 below.

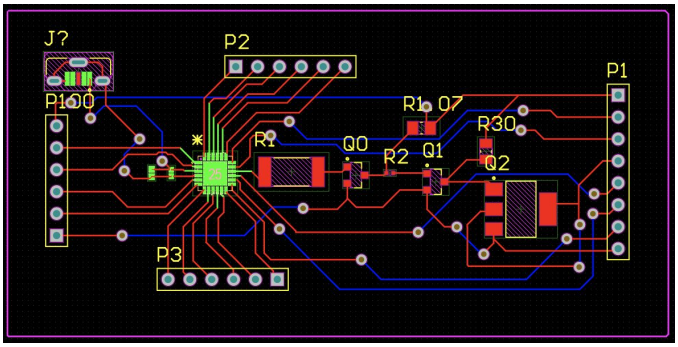

Figure 7: MCU On-Chip Design

This board had many problems. It suffered the same problems as our backup version. The backup version, which is shown below in Figure 8, had numerous output issues.

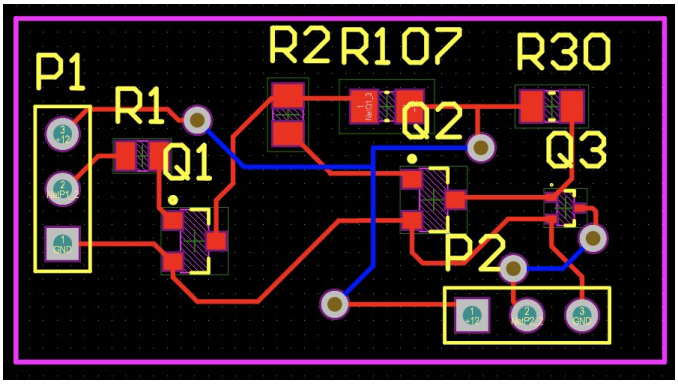

Figure 8: MCU Off-Chip Design

We were getting very inconsistent results from the NMOS and BJT inputs and outputs. We hand made a through hole PCB step by step to debug the issue, and we did not see any issues. When we went through populating each board we saw different issues each time. Some were due to burned out traces from soldering the board too much to hopefully fix them while others were showing dampened signals at the output of certain transistors. We also tried to use some breadboard components hoping that it would fix our problem since the breadboard worked completely fine. This also did not work. We eventually seeked the help from other Professors, but to no avail. Our next step would have been to work with our advisor and his PhD student to redesign the PCB from scratch.

C. Functionality

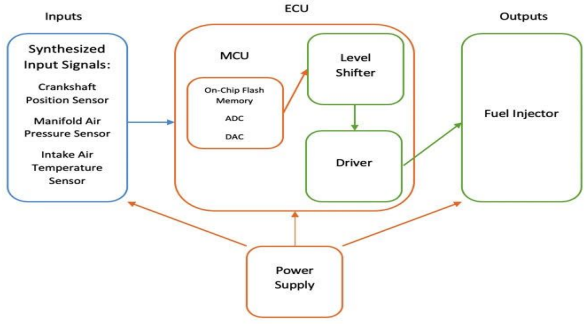

Figure 9: System Block Diagram

For our ECU, we have all the parts working on the breadboard but not on the PCB we design. The system in figure 9 is not very complex. Basically, we fetch sensor inputs and use Attiny 817 to process them and then output pulse width to the fuel injector.

For the inputs, we can successfully read them using the ADC built in the Attiny817 chip. To verify it, we first changed the value of the sensors and then used an oscilloscope to measure the voltage for calculating what value should appear on the ADC register, and we are correct if the value on the register is the same as our calculation.

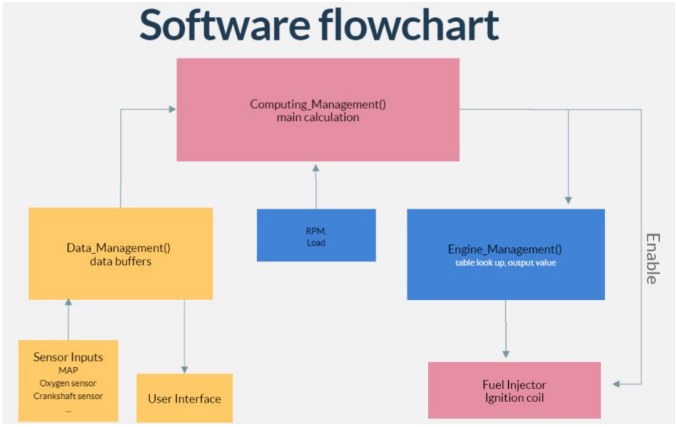

Figure 10: Software System Flowchart

After successfully reading the sensor inputs. We need to process them with software described in figure 10, and to process them we need to store them in buffers. We have data buffers, but we did not provide the user interface because it is not the essential part of our project, and we do not have time left to build it due to COVID-19.

The RPM and Load is calculated using the sensor inputs which are stored in the data buffers. The load is easy to compute because it is calculated from a formula using inputs, and we verified it by setting inputs to certain values and see if the Load is correct. However, RPM is hard to calculate because it is not directly calculated from the inputs but the duration between each two inputs. We verified this by setting the hall effect sensor to certain RPM and reading the RPM value in the program. Last in the engine management part we use the RPM and Load to get a pulse width which is the output from the lookup table. We use the built in PWM module to produce output pulse to level shifter and a driver circuit to shift voltage from 3.3V to 12V and drive the fuel injector.

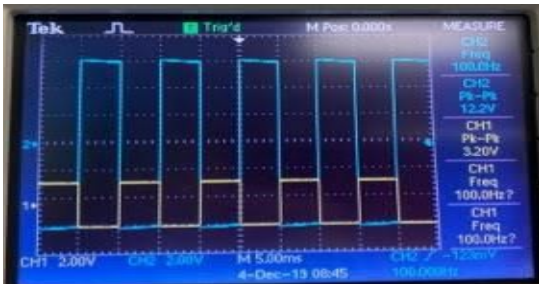

Figure 11: Level-Shifter Pulse Wave

Figure 11 is the picture for the level shifter. The yellow pulse is output from the MCU and the input for the level shifter. The blue one is the output from the level shifter. We verified that the driver circuit works by sending a pulse to the circuit and letting the injector inject compressed air.

We do not have quite complicated functionality and realize all functions individually and combined together in our system described in the diagram with correct functioning.

D. Performance

As Table 1 shows, there are five aspects that we want to achieve. But in fact, our product has not gone through the final testing and acceptance stage due to COVID-19, in other words, our product has not been tested with a real car engine in practice. Although this was disappointing, overall the experiments in the laboratory were very successful. First of all, we met the two important requirements in the laboratory testing phase, one is responsive requirement, the other is frequency requirement. For the frequency requirement, we easily got the engine response time below 6000 RPM. Besides, since processes within the engine occur in milliseconds, we need to get the response time with a range of less than 10ms for the responsive requirement. After many tests and experiments, it works perfectly. Secondly, in mechanical manufacturing, we tried to make a mechanical switch to control the electronic control unit rather than an electronic switch, but we didn't have enough time to finish it. since we didn't test it on the engine, unfortunately, we haven't had time to process the data we read from the oxygen sensor. Hopefully, we will have the opportunity to test more and optimize more for our project.

IV. Conclusion

The current state of the project is good. There are some key things to focus on soon. Getting the system to work for basic functionality is of the utmost importance. Once that is settled then we can focus on engine component integration. That will provide a lot of work for us to do. Going from perfectly working synthesized signals to somewhat unstable and possibly unreliable engine components could become problematic if enough time isn't allocated to troubleshooting any issues that may arise. To achieve our goal of using actual engine components we will need a large amount of time dedicated to it. Asking questions to the SMV team will also be important since none of us have worked with these parts before. Time is of the essence and to be where we want to be as a group then near-perfect time management needs to be the focus in the future.

Acknowledgment

As a group, we would like to thank our advisor, Prof. Xia, for working around our schedule and meeting with us when needed. Also, we would like to thank our evaluators, Prof. Hollot and Prof. Gong, for providing good constructive criticism. All points were made clear in their feedback and all were taken into consideration.

References

[1] "Inventory of U.S. Greenhouse Gas Emissions and Sinks," *EPA*, 11-Apr-2019. [Online]. Available: https://www.epa.gov/ghgemissions/inventory-us-greenhouse-gas-emissions-and-sinks. [Accessed: 17-Dec-2019].

[2] "Sources of Greenhouse Gas Emissions," *EPA*, 13-Sep-2019. [Online]. Available: https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions. [Accessed: 17-Dec-2019].

[3] "SAE Supermileage - Student Events - Attend - SAE International," *SAE International* ®. [Online]. Available: https://www.sae.org/attend/student-events/sae-supermileage. [Accessed: 26-Jan-2020].

[4] *ATtiny817 Xplained Pro*. [Online]. Available: https://www.microchip.com/DevelopmentTools/ProductDetails/attiny817-xpro. [Accessed: 27-Jan-2020].

[5] "Atmel Studio 7," *Atmel Studio 7 | Microchip Technology*. [Online]. Available: https://www.microchip.com/mplab/avr-support/atmel-studio-7. [Accessed: 27-Jan-2020].

[6] "Arduino Uno Rev3," *Arduino Uno Rev3 | Arduino Official Store*. [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3. [Accessed: 27-Jan-2020].

[7] "MC33812 and S12P Small Engine Control," *NXP*. [Online]. Available: https://www.nxp.com/products/power-management/smart-switches-and-drivers/low-side-switches/mc33812-and-s12p-small-engine-control:KIT33812ECUEVME. [Accessed: 27-Jan-2020].

Appendix

A. Design Alternatives

In our design, we designed our system around using the ATTiny817 chip. On paper it is fast enough for our computational speed requirement while also being recommended by a faculty member. Using this as our foundation we did see some drawbacks to the chip. We would only be able to output around a 5V peak voltage. This is not enough to open and close the fuel injector which needs around 12V. This is where the level shifting circuit comes into play. The level shifter will be able to shift the peak voltage up to the fuel injectors requirement. Then the output of the level shifter will feed directly into the driving circuit. The driver will be able to drive an inductive load which is the fuel injector. Without this then the fuel injector will not work correctly. An alternative would have been to use a chip that was designed for vehicle applications made by NXP[8], but the board is several hundred dollars. This would have been a good option since all of the components provided would have been for vehicle applications, but it would have taken a large portion of our budget. Ultimately, we decided to go with a microcontroller that we knew would be fast enough for our application and also was recommended by faculty. We also thought it would be a good experience to build our own level shifter and to learn more about how to drive an inductive load.

[8] See [7] in the References Section

B. Technical Standards

We followed several technical standards during the development of our system. We implemented a rigorous software testing system. We created Anomaly Reports following any bug or strange error encountered during testing and recorded the report in our shared drive. This follows IEEE 829 technical standard for Software Test Documentation.

This system is also POSIX compliant and has full interoperability with other POSIX systems. This follows IEEE 1003 and while the system itself is limited in what it can do, it should have no problem being further developed to interact with other machines with POSIX compliance.

This system also follows IEEE's guidelines for Floating Point Arithmetic, IEEE 754. Many older systems used to use numerous formats for dealing with floating point numbers that made them a lot less portable, but it is simplified a lot by all systems following a common standard.

C. Testing Methods

Until now, we made progress on the basic functionality of the ECU system. Then we must test each part whether it can successfully work by itself. We mainly used an oscilloscope to watch the signals to check them.

For the engine input signals part, when we changed the resistance of the potentiometers, there was a changed wave shown on the oscilloscope.

And for the MCU part, we also used an oscilloscope to watch and analyze the pulse width of the different resistance of potentiometers. The wave should be a rectangular wave and the pulse width is around about 10ms. In addition, we tried to use some data to see whether they are matched with the lookup table.

Last but not least, it was testing the level shifter and driver. Without other parts in the ECU, we used the function generator to build a simulated rectangular signal at 6000RPM (100Hz) and the high voltage being 3.3V to synthesize the output signal from our MCU. Then we checked the output signal of the level shifter and driver which looks like a rectangular wave with a high voltage of 12V, and it has the correct pulse width. This is the same as the input.

In conclusion, to test the whole system, we were connecting to a fuel injector. And the fuel injector will be injecting compressed air. Besides this, we also must watch the output pulse width of the fuel injector. We put a finger on the end of the fuel injector so that we could feel the intermittent air flow.

D. Team Organization

Our team is organized well. We meet every week with our advisor. We also meet throughout the week as a group to talk about what we've worked on or what we have planned. Attendance at our weekly meetings is great. It is very rare that we do not have everyone in attendance. Our expertise is split up from our majors. The two electrical engineering majors, Takuya and Yongjie, work as the hardware team. The two computer engineering majors, Jack and Xueteng, work as the software team. We tend to stick to our expertise, but since we all have a baseline knowledge of each other's work then we are never afraid to help each other out. We communicate through email, online messaging, and in person. Sometimes when communicating through email or online messaging isn't working then we will schedule a video call to get our points across without the obscurity of text messages. Overall, we help each other and get along well. There has not been a single time in which any of us have felt that our opinion or advice isn't wanted. Whenever one of us has some sort of idea or suggestion then we speak up and talk through it.

E. Beyond the Classroom

Takuya – Besides technical skills that I have needed to develop I think that the most important skills that I have applied from my professional life would be from a management perspective. I have used many of the techniques that I have seen in my professional life to make sure that we as a team have good communication and clear goals in mind. As for design, many resources have given me ideas as to how to approach the hardware problem. From coworkers at my internship to online resources, many have given me direction as to how to approach an engineering problem. I am always relating the issues that come up within our project to similar issues from my internship experience. Many of the issues with creating a product from the research and development phase is like what we are trying to accomplish within our project. It has all been very informative, especially from relating our issues to that of the engineering industry.

Jack – One of the main skills I have had to develop with this project is working with embedded systems. Most of the work I did was developing the software with Xueteng for our ATtiny817 microcontroller. We developed this software in C using Atmel Studio 7 IDE. I have had to become a lot more comfortable with reading data sheets for the microcontroller as well as understanding microcontroller concepts. Atmel Start's website has been very helpful as it has many examples of projects that help you get started with the ATtiny817. I have been drawing a lot of my experience in Computer Science Lab I & II as I worked with embedded systems in those classes. I can see a connection with this project and my life as a professional. I really enjoy programming and want to pursue that after I graduate; however, I don't know if I see myself working on embedded systems specifically.

Yongjie – I have obtained a lot of knowledge from our ECU project. As a hardware circuit designer, I mainly designed a circuit to amplify and modify a signal by using the topology circuit. For this project, I successfully designed a level shifter and driver, whose function is to let the ECU system drive the fuel injector. To design this part, I did a lot of research and changed different kinds of transistors repeatedly to optimize the circuit and minimize the noise signal as much as possible

to get a precise output control signal from the ECU. In fact, each integrated circuit design potentially enhances my mastery of hardware knowledge. Furthermore, I believe in the future of the professional career life, derived from this project, I can develop more technical skills in experiencing this kind of teamwork.

Xueteng – For this project, I mainly did the ECU programming part with Jack. We used the ATtiny817 and programmed in C. The ability to debug is very important, because we have encountered lots of problems each time adding a new function to the whole program. After MDR, we improved our debugging skills and we can kind of predict problems that might appear after we add new functions. The datasheet is one of the most important resources because it teaches us the functionality of each part and how to use them, and another good source is the sample code from the Atmel start website because we need lots of basic setup codes and it saves us lots of time. This project taught me that if I want to build something from scratch, I need to be patient and walk step by step from designing, building blocks, and assembling. Every step could have tons of questions, and the only solution is to face the problems at hand and solve them one at a time.