Cole Brooks

Thomas Butler

55:036 Embedded Systems

Post-Project Report

Introduction

When given the choice between a default project and choosing our own, we immediately jumped to finding our own way through the final project. We immediately started brainstorming. We realized that the goal of this project was to use all the knowledge we had acquired from this class to do something that both interested us and served a real purpose.

It was agreed upon very quickly that it would be interesting and rewarding to do something with a new technology that has peaked both of our interests in the past. We decided that we were going to do something with the block chain.

Integrating modern technologies that we had learned about outside of the class with the embedded systems technologies that we have learned about inside the class, we were able to create a simple cryptocurrency price tracking system. This system displays price data about preloaded cryptos and uses a Bluetooth system and a Python script to update data in real time. The system allows for the user to cycle between supported cryptos, and triggers an alarm when the price falls.

Description of Design Functionality

In this section we will discuss how the system behaves. We will begin with a high-level overview of how the system functions as an entire unit. After that, we will talk about the complexities unique to the hardware and the software we used in developing this system.

### Overview:

The high-level behavior of this system is quite straightforward to understand. Prices should be displayed on an LCD based on push button configuration from a user. These prices should update in real time based on information fed to the circuit from the Bluetooth module, and any time the price decreases over an interval, an alarm should sound to inform the user about price volatility.

### Hardware:

The LCD we used for this project was the same module as used in Lab 4. We had intended to use either a graphical display module or an LED matrix module for the display in order to show more information than simply the price, but due to the current COVID-19 related supply chain issues facing much of the world, it would have been impossible to get the components in time to figure out how to implement them for the project. However, we found that the LCD provided a nice proof of concept for our circuits display.

Prices are fed to the LCD via an HC06 Bluetooth Communications module. This component was acquired via the University of Iowa Engineering shop. This device receives information wirelessly from a PC which is running a Python script to get cryptocurrency data. The Bluetooth module was connected to the USART transmit and receive pins, and after some work it functioned very similarly to how the USB cable used in Lab 5 worked.

We used the passive buzzer found in the Arduino kit for the alarm feature we wanted to implement. This component was quite straightforward and was simple to add to the design.

### Software:

In order to receive real time information about the prices of the coins that we were interested in we needed two different pieces of software. The first program we will discuss is written in C and controls the embedded systems for this project. The second program which we will discuss briefly was a script written in Python which formats data from an API and sends the data over the Bluetooth connection between the PC and the Arduino.

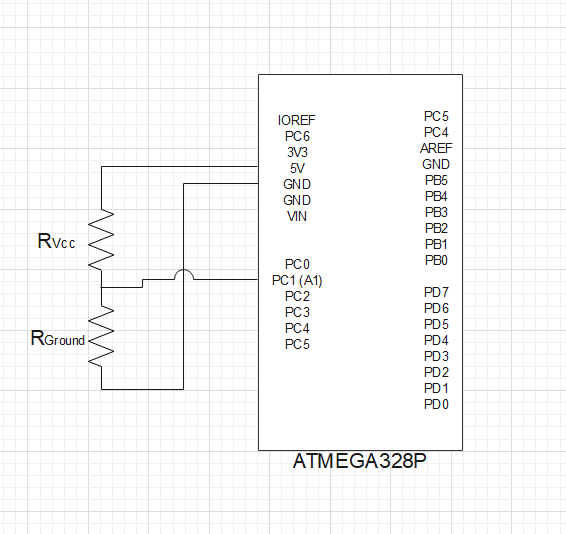
The C program is straight-forward and demonstrates quite a few concepts which we have learned over the semester. This C program initializes the hardware, controls the flow of serial (Bluetooth) communication, and implements an external interrupt for user interfacing.

The first part of the software simply initializes registers in the atmega328p for the LCD and for USART communication. This portion of the C program is largely made up of translated LCD code from lab 4, and reused USART code from Lab 5. It was also important to specify the inputs and outputs to and from the Arduino. It also enables external interrupts for user interfacing.

Next in the C program comes the bulk of the logic for controlling the flow of serial communications and output to the LCD. The first step in this process is reading in any updates that have arrived via the Bluetooth module. Any data read in needed to replace the previous data stored in an array of prices. Then we needed to compare the previous data to the new data to determine if the alarm should sound. If the alarm was triggered, this portion of the code would sound the alarm. After that, the system would continue and rewrite the name and the current price of the currently selected crypto the LCD. This loop continues infinitely while the system is continuously receiving data. Should the system ever stop receiving data, the loop will pause temporarily.

The final portion of the C software worth noting is the approach to user configuration. We needed a way for the user to select the coin they wanted to look at, and we thought a push button interrupt was a good choice of implementation. We opted for a rising edge triggered interrupt that simply moved an index along an array. In the future, the user may press this button many times to scroll through all the different cryptocurrencies, but because only Bitcoin and Ethereum are currently supported, the push button simply toggles between those two.

Schematic



Test Procedure

The following table was used to verify completeness of the circuit designed for this project.

|  |  |
| --- | --- |
| **Requirement** | **Status:** |
| The system begins by displaying a welcome message | Verified |
| The system will show no data until a Bluetooth communication connection has been established | Verified |
| Running a Python script shall establish a Bluetooth communications connection to the system | Verified |
| The system shall receive updates to price information via the Bluetooth communications connection. | Verified |
| The system shall display the updated information to the LCD. | Verified |
| The system shall sound an alarm 10 times in 3 seconds should the price of the currently selected cryptocurrency decrease. | Verified |
| The user shall control the currently selected crypto via a push button. | Verified |
| The system shall continue the cycle described in the previous requirements until either A) The system is reset  B) The communications connection is severed | Verified |

Discussion

As mentioned previously, we very quickly decided that we wanted to go a different route than the default option for this project. We thought that it would be more interesting, more fun, and more useful to develop something else. We talked to each other about a few different options, but this project was a clear winner almost immediately. In this section, we will discuss the evolution of this project from start to finish. We will write about the logistics of planning and implementing such a project for our team. Then, we will cover the timeline we took in development of this project, and expectations versus realities we faced throughout that timeline. Additionally, we will cover limitations of our current implementation. Finally, we will talk about how we plan on enhancing our design in the future.

The planning and organization for this project occurred entirely online. Like in all the labs, our primary mode of communication was an online messaging platform, Discord. We faced severe time constraints for this project due both to the scale of what we wanted to implement, and the commitments we had to other classes. Fortunately, we were able to agree upon a design specification for this project early on including key features that would be necessary for a bare minimum ‘proof of concept’ system which could be built upon in iterations. Doing this allowed us to focus on what was important, and to prune unnecessary features when they turned in to time sinks.

The project started out in the same way that many do. We spoke to each other about feature after feature that we could implement. We talked about making graphs of previous crypto prices, allowing users to select any number of cryptos to track, as well as customizable alarms based on user configuration. Originally, we even wanted to handle communications with the CoinMarketCap API onboard the Arduino via a Wi-Fi module. Fortunately, we spoke to the professor early on and cut down significantly on the features we would commit to and moved our focus from a Wi-Fi module to a Bluetooth module to decrease the complexity of interfacing with the API.

After that, we spent a few days investigating components which we would use for this project. Originally, we wanted to use something other than the LCD for the display module for this system. We thought that it would have been exciting and interesting to implement a graphical display module so that we could show coin symbols and graphs along with the price and historical data for each crypto. Unfortunately, as mentioned before, we faced serious supply chain issues with acquiring the hardware that we would have needed, so instead we opted to use the 16x2 LCD from the Arduino kit.

After getting all the required components, we did our best to come up with proof-of-concept demonstrations for all components we were not familiar with. We started off with the Bluetooth module, because we thought that would be the most complex. The HC06 is a very commonly used module in Arduino projects, so we expected resources to be widely available and tutorials to be plentiful. This is the case, but unfortunately almost all of the resources we found were related to C++ or Objective C code. This meant that these online resources were largely unsuited to our purposes. Instead we turned to the data sheet. After a while, we figured out that the HC06 module was basically initialized for our purposes right out of the box. This meant that all we really needed to do was refactor our code from Lab 5 to take data from the Bluetooth module and print it to the LCD.

While we were working on the Bluetooth module, we were also working on getting information via the Python script and the CoinMarketCap Api into the serial monitor. We have used Python many times in the past, so this was a problem that we were very comfortable with. After a few days of research and trial and error, we were able to come up with a working script that could send information to the embedded system.

After that, we unfortunately had quite busy weekends with other classes. We didn’t do much work on the project the weekend before it was do, but we laid a solid groundwork so from Monday the 3rd to Wednesday the 5th we were able to finish the key requirements for this project. Largely the challenges here were solving program flow issues. We needed to figure out in what order we needed to poll for USART data and poll for user input. Eventually we figured out that polling for both of those data streams would not work. Instead, we opted to use polling for the updated price information, and interrupt driven frameworks for the user input.

We learned a lot about what embedded systems design is about due to the time crunch faced in development of this project. We had to cut features we were excited about in order to prioritize getting the key features to work well. We had to improvise when we figured out that we could not get the components we wanted, and we had to spend late hours getting what we determined to be the key features working flawlessly so as to make for a solid proof of concept for our system. After presenting our system and discussing our process with the professor, we realized that this was all just a part of embedded systems design in the real world. We realized that good engineers remain flexible problem solvers even in the face of difficult decisions. Almost every project we ever work on will have strict requirements, tight budgets, and short schedules. This means that we must learn how to implement what we need with what we have available to us, in the amount of time that we are given. We think that this project was a great way to learn that.

This leads us to the discussion about the limitations of our current design. In this report we will discuss three of the most glaring limitations of our current design, and briefly speak about what we could have done to solve these limitations.

The first limitation that we wanted to speak about was our polling loop. The way it is currently implemented, the python script needs to constantly be feeding the display information in order for the display to update on time. If we had more time, we would have implemented some kind of interrupt driven data reading system such that while there was no data in the USART data register, the Arduino would focus on writing to the LCD and taking user input. That way, the microcontroller would be free to keep working instead of being forced to sit and wait for data to come from the script.

The next limitation we face with our current design is one of storage of previous data. Due to limited storage on the microcontroller, it was not feasible to keep an array of all of the previous prices for each crypto. Instead, we simply keep track of the current price, and the price that the currently viewed coin was at one loop ago. If we had more time, and access to more components, we could have solved this limitation in a couple different ways.

Given more time, we most likely would have implemented a way for the microcontroller to request data from the API instead of idly waiting for the data to come to it. The Python script has access to orders of magnitude more data than the microcontroller. Some of that data is about the 24 hour shifts in price for a coin. Using that data we could more meaningfully trigger the alarm.

Another option would have been to connect more memory to the Arduino. If we did that, we could likely figure out a way to implement a linked list like data structure so that we could keep adding to an array with historical data about each supported digital asset. Then we could process all this data on the Arduino side and trigger the alarm more meaningfully. This approach would better test our embedded systems design skills since we would not be relying so heavily on the Python script or the API.

A final limitation of our design is one that we have spoken about a few times throughout this report. We simply could not get anything other than the name and price of the token to display for the system. Like we have mentioned before, we originally intended this system to show graphs, images, and historical data about each token, but our capabilities were severely limited by the hardware we were not able to acquire. We attempted to solve this issue by defining our own characters for the LCD, but we were unsuccessful. If we had adequate storage, we may have been able to display enough data for this method to be successful, but again, that really wasn’t a possibility by the time we realized the challenge we were facing.

In the future, we also plan on implementing a cleaner user interface for this project. We are not sure whether we want to use a key pad entry system, an RPG based entry system, or simply a serial communication interfacing system, but we think that implementing something more robust than a simple push button toggling interface would make improvement on this system much easier to implement. Additionally, creating a modern and robust interface for this system would allow for much more configurability to be implemented in the future.

Fortunately, just because we are finishing up with the class does not mean that we need to be satisfied with this project where it is at. In the future, we have plans to implement each of the suggestions above as well as a few others not covered in this report. Working on a side project has the advantage of having far less strict time requirements. We believe that if we keep working on this project in iterations, eventually it will be something interesting and unique enough to be used by the average crypto-savvy consumer.

Appendix A: Source Code

main.c

/\*

\* CryptoTracker.c

\*

\* Created: 4/27/2021 5:32:56 PM

\* Author : Cole Brooks, Thomas Butler

\*/

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <string.h>

#include <stdio.h>

#include <avr/interrupt.h>

#include <stdlib.h>

// USART stuff

#define USART\_BAUDRATE 9600

#define UBRR\_VALUE (((*F\_CPU* / (USART\_BAUDRATE \* 16UL))) -1)

// Alarm definitions

#define alarm\_port PORTB // alarm connection

#define alarm\_bit PORTB0

#define alarm\_ddr DDRB

// LCD interface definitions

#define lcd\_D7\_port PORTC // lcd D7 connection

#define lcd\_D7\_bit PORTC3

#define lcd\_D7\_ddr DDRC

#define lcd\_D6\_port PORTC // lcd D6 connection

#define lcd\_D6\_bit PORTC2

#define lcd\_D6\_ddr DDRC

#define lcd\_D5\_port PORTC // lcd D5 connection

#define lcd\_D5\_bit PORTC1

#define lcd\_D5\_ddr DDRC

#define lcd\_D4\_port PORTC // lcd D4 connection

#define lcd\_D4\_bit PORTC0

#define lcd\_D4\_ddr DDRC

#define lcd\_E\_port PORTB // lcd Enable pin

#define lcd\_E\_bit PORTB3

#define lcd\_E\_ddr DDRB

#define lcd\_RS\_port PORTB // lcd Register Select pin

#define lcd\_RS\_bit PORTB5

#define lcd\_RS\_ddr DDRB

// LCD specific definitions

#define lcd\_LineOne 0x00 // start of line 1

#define lcd\_LineTwo 0x40 // start of line 2

// LCD instructions

#define lcd\_Clear 0b00000001 // clears the screen

#define lcd\_Home 0b00000010 // return cursor to first position on first line

#define lcd\_EntryMode 0b00000110 // move cursor from left to right on read/write

#define lcd\_DisplayOff 0b00001000 // turns display off

#define lcd\_DisplayOn 0b00001100 // display on, cursor off, blink off

#define lcd\_reset 0b00110000 // reset

#define lcd\_setTo4Bit 0b00101000 // sets to 4 bit data entry mode

#define lcd\_SetCursor 0b10000000 // sets cursor position

// Prototypes

const char\* get\_string(char input\_str[]);

char\* store\_prices(const char str[], char price\_array[10][10]);

*uint16\_t* usart\_rx(void);

void usart\_init(void);

void lcd\_write(*uint8\_t*);

void lcd\_write\_instruction(*uint8\_t*);

void lcd\_write\_char(*uint8\_t*);

void lcd\_write\_str(*uint8\_t* \*);

void lcd\_init(void);

void move\_to\_line\_2(void);

void alarm();

int currentCrypto = 0; // 0 denotes bitcoin, 1 denotes eth. Global for interrupt access

int resetPrice = 1; // changed to 1 when currentCrypto changes to get a starting price for alarm.

/////////////////////////////////////////////////

// Function: move\_to\_line\_2

// Purpose: moves cursor to line 2

/////////////////////////////////////////////////

int main(void)

{

// storage variables

char cryptos[2][10] = { "Bitcoin", // supported crypto names. Index = current crypto int

"Ethereum"};

char prices[10][10];

double previousPrice = 0; // previous price of currently viewed crypto. Used for alarm

// configure the data lines for output to LCD

lcd\_D7\_ddr |= (1<<lcd\_D7\_bit);

lcd\_D6\_ddr |= (1<<lcd\_D6\_bit);

lcd\_D5\_ddr |= (1<<lcd\_D5\_bit);

lcd\_D4\_ddr |= (1<<lcd\_D4\_bit);

// configure data lines for output to alarm

alarm\_ddr |= (1<<alarm\_bit);

// configure the data lines for controlling the LCD

lcd\_E\_ddr |= (1<<lcd\_E\_bit); // Enable

lcd\_RS\_ddr |= (1<<lcd\_RS\_bit); // Register Select

// init lcd and usart

lcd\_init();

usart\_init();

// Type welcome message

lcd\_write\_str("Welcome to");

move\_to\_line\_2();

lcd\_write\_str("CryptoTicker");

// display welcome message for 5 seconds

// and then clear the screen

*\_delay\_ms*(5000);

lcd\_write\_instruction(lcd\_Clear);

*\_delay\_ms*(80);

// set INT0 to trigger on rising edge

EICRA |= (1 << ISC00) | (1 << ISC01);

EIMSK |= (1 << INT0);

*sei*();

// main program loop

while(1){

// wait for prices to update

char input\_str[50];

get\_string(input\_str);

store\_prices(input\_str,prices);

// reset the previous price before checking alarm if resetPrice flag is set

// and then set flag to zero

if(resetPrice == 1){

previousPrice = *atof*(prices[currentCrypto]);

resetPrice = 0;

}

// check for a need to alarm

if(previousPrice > *atof*(prices[currentCrypto])){

alarm();

}

// either way, set previous price to the new price

previousPrice = *atof*(prices[currentCrypto]);

lcd\_write\_instruction(lcd\_Clear);

*\_delay\_ms*(80);

lcd\_write\_str(cryptos[currentCrypto]);

move\_to\_line\_2();

*\_delay\_ms*(80);

lcd\_write\_str(prices[currentCrypto]);

}

return 0;

} ///////////////////// END OF MAIN //////////////////////////

/////////////////////////////////////////////////

// function: ISR()

// purpose: toggles the current crypto via interrupt

/////////////////////////////////////////////////

ISR(INT0\_vect)

{

switch(currentCrypto){

case 0:

currentCrypto = 1;

break;

case 1:

currentCrypto = 0;

break;

}

resetPrice = 1;

}

/////////////////////////////////////////////////

// function: alarm

// purpose: sounds the alarm that the price dropped

// > 10 percent

/////////////////////////////////////////////////

void alarm()

{

int i = 0;

for(i; i < 10; i+=1){

alarm\_port |= (1<<alarm\_bit);// turn alarm on

*\_delay\_ms*(150);

alarm\_port &= ~(1<<alarm\_bit);// turn alarm off

*\_delay\_ms*(150);

}

}

/////////////////////////////////////////////////

// function: usart\_init

// purpose: initialize usart.

// - set baud rate

// - enable rx and tx

// - set frame format 8 data, 2 stop bit

/////////////////////////////////////////////////

void usart\_init(void)

{

// Set baud rate

UBRR0H = (*uint8\_t*)(UBRR\_VALUE>>8);

UBRR0L = (*uint8\_t*)UBRR\_VALUE;

// Set frame format to 8 data bits, no parity, 1 stop bit

UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);

//enable transmission and reception

UCSR0B |= (1<<RXEN0)|(1<<TXEN0);

}

/////////////////////////////////////////////////

// function: usart\_rx

// purpose: receives data from bluetooth module

/////////////////////////////////////////////////

*uint16\_t* usart\_rx(void)

{

// Wait for byte to be received

while(!(UCSR0A&(1<<RXC0))){};

// Return received data

return UDR0;

}

/////////////////////////////////////////////////

// function: get\_string

// purpose: gets string from bluetooth module

/////////////////////////////////////////////////

const char\* get\_string(char input\_str[]){

char buffer[10];

*uint16\_t* input = usart\_rx();

int i = 0;

while (input != '\n'){

*itoa*(input, buffer, 10);

input\_str[i] = *atoi*(buffer);

i = i + 1;

input = usart\_rx();

}

return input\_str;

}

//////////////////////////

// function: store\_prices

// purpose: gets price string and converts string to array of prices

char\* store\_prices(const char str[], char price\_array[10][10]){

const char s[2] = ",";

char \*token;

int i = 0;

char buffer[10];

/\* get the first token \*/

token = *strtok*(str, s);

/\* walk through other tokens \*/

while( token != *NULL* ) {

*sprintf*(price\_array[i], " %s", token );

i = i + 1;

token = *strtok*(*NULL*, s);

}

return price\_array;

}

/////////////////////////////////////////////////

// function: lcd\_init

// purpose: initializes the LCD in 4-bit data mode

/////////////////////////////////////////////////

void lcd\_init(void)

{

// delay for a bit so hardware can do it's thing

*\_delay\_ms*(100);

// note we start in 8 bit mode, so we gotta change that

// Set up the RS and E lines for the 'lcd\_write' subroutine.

lcd\_RS\_port &= ~(1<<lcd\_RS\_bit);

lcd\_E\_port &= ~(1<<lcd\_E\_bit);

// Setup the LCD

lcd\_write(lcd\_reset); // first part of reset sequence

*\_delay\_ms*(10);

lcd\_write(lcd\_reset); // second part of reset sequence

*\_delay\_us*(200);

lcd\_write(lcd\_reset); // third part of reset sequence

*\_delay\_us*(200);

lcd\_write(lcd\_setTo4Bit); // set 4-bit mode

*\_delay\_us*(80);

// Function Set instruction

lcd\_write\_instruction(lcd\_setTo4Bit);

*\_delay\_us*(80);

// Display On

lcd\_write\_instruction(lcd\_DisplayOff);

*\_delay\_us*(80);

// Clear Display

lcd\_write\_instruction(lcd\_Clear);

*\_delay\_ms*(4);

// Entry Mode Set instruction

lcd\_write\_instruction(lcd\_EntryMode);

*\_delay\_us*(80);

// Display On/Off Control instruction

lcd\_write\_instruction(lcd\_DisplayOn);

*\_delay\_us*(80);

}

/////////////////////////////////////////////////

// Function: move\_to\_line\_2

// Purpose: moves cursor to line 2

/////////////////////////////////////////////////

void move\_to\_line\_2(void){

lcd\_write\_instruction(lcd\_SetCursor | lcd\_LineTwo);

*\_delay\_us*(80); // 40 uS delay (min)

}

/////////////////////////////////////////////////

// function: lcd\_write\_str

// purpose: sends a string to the LCD to be

// displayed

/////////////////////////////////////////////////

void lcd\_write\_str(*uint8\_t* theString[])

{

volatile int i = 0; // character counter\*/

while (theString[i] != 0)

{

lcd\_write\_char(theString[i]);

i++;

*\_delay\_us*(80); // 40 uS delay (min)

}

}

/////////////////////////////////////////////////

// function: lcd\_write\_char

// purpose: send a byte nibble by nibble as

// a character

/////////////////////////////////////////////////

void lcd\_write\_char(*uint8\_t* theData)

{

lcd\_RS\_port |= (1<<lcd\_RS\_bit); // select the Data Register (RS high)

lcd\_E\_port &= ~(1<<lcd\_E\_bit); // make sure E is initially low

lcd\_write(theData); // write the upper 4-bits of the data

lcd\_write(theData << 4); // write the lower 4-bits of the data

}

/////////////////////////////////////////////////

// function: lcd\_write\_instruction

// purpose: send a byte nibble by nibble to

// the LCD as an instruction

/////////////////////////////////////////////////

void lcd\_write\_instruction(*uint8\_t* theInstruction)

{

lcd\_RS\_port &= ~(1<<lcd\_RS\_bit); // select the Instruction Register (RS low)

lcd\_E\_port &= ~(1<<lcd\_E\_bit); // make sure E is initially low

lcd\_write(theInstruction); // write the upper 4-bits of the data

lcd\_write(theInstruction << 4); // write the lower 4-bits of the data

}

/////////////////////////////////////////////////

// function: lcd\_write

// purpose: send a byte nibble by nibble to the LCD

/////////////////////////////////////////////////

void lcd\_write(*uint8\_t* theByte)

{

lcd\_D7\_port &= ~(1<<lcd\_D7\_bit); // assume that data is '0'

if (theByte & 1<<7) lcd\_D7\_port |= (1<<lcd\_D7\_bit); // make data = '1' if necessary

lcd\_D6\_port &= ~(1<<lcd\_D6\_bit); // repeat for each data bit

if (theByte & 1<<6) lcd\_D6\_port |= (1<<lcd\_D6\_bit);

lcd\_D5\_port &= ~(1<<lcd\_D5\_bit);

if (theByte & 1<<5) lcd\_D5\_port |= (1<<lcd\_D5\_bit);

lcd\_D4\_port &= ~(1<<lcd\_D4\_bit);

if (theByte & 1<<4) lcd\_D4\_port |= (1<<lcd\_D4\_bit);

lcd\_E\_port |= (1<<lcd\_E\_bit); // Enable pin high

*\_delay\_us*(1); // implement 'Data set-up time' (80 nS) and 'Enable pulse width' (230 nS)

lcd\_E\_port &= ~(1<<lcd\_E\_bit); // Enable pin low

*\_delay\_us*(1); // implement 'Data hold time' (10 nS) and 'Enable cycle time' (500 nS)

}

Appendix B: References

ATmega328p, 3/10/21 < <https://uiowa.instructure.com/files/14761502/download?download_frd=1> >

ARDUINO\_V2.pdf, 3/10/21 < <https://uiowa.instructure.com/files/14762893/download?download_frd=1> >