**IZMIR UNIVERSITY OF ECONOMICS**

**FACULTY OF ENGINEERING**

**ELECTRICAL AND ELECTRONIC ENGINEERING**

**FENG 498 PROJECT REPORT**



Constant Current Digital Electronic Load with GUI

Author(s):

Furkan MAKASÇI

**1.Abstract**

In our project we tried to design a digitally controlled electronic load. We tried to eliminate the requirements and problems that we did not realize by going over our FENG 497 schematic. We created a user interface for our digitally controlled electronic load. Finally, we physically installed and tested our project. With these tests, we made the necessary small changes on very different points in the physical electronic load and the user interface for electronic load to work properly.

The motivation to bring this project to life was mainly about testing batteries and power supplies. But the real motivation for making an electronic load may be to test any circuit. An electronic load can be very useful for its controllability, whether when checking a circuit to see if it can deliver the required voltage or current, or when deciding on a load parameters in a circuit.

In our electronic load design, we used a shunt resistor, a differential and comparator opamp as a type of use, an analog-digital converter to monitor and control the current accordingly. We also used a UART connection to communicate with the user interface and show the results to the user. Basically, the methodology in this project is to drive the circuit (draw current) by utilizing a mosfet, convert this current to voltage with a shunt resistor, measure this current with a differential opamp, and control the mosfet and current driven by a comparator opamp with feedback.

But, at the end we couldn’t accomplish our goal. Our digitally controlled electronic load doesn’t work properly. Furthermore, this don’t mean we didn’t accomplish nothing. We created a feature-packed graphical user interface and design an electronic load which controlled digitally.

**2. Introduction**

Electronic load contains semiconductors and thanks to the properties of these semiconductors, we can use the electronic load as variable resistors. Semiconductors used in electronic load are generally MOSFET and IGBT. As we can understand from here, we use the electronic load to control the current and voltage in the circuit. That is, the electronic load acts as a variable resistance controlled by electrical signals. Therefore, the question arises, why do not we use resistors instead of electronic loads. However, there are two disadvantages to using resistors. First of all, to test different power supplies under different conditions, many different values ​​of resistors are required, and the resistors sometimes have to be of very high power and some resistance values ​​are non-standard. Secondly, the resistors dissipate a certain amount of power in order not to burn, so we cannot confirm that a power supply is working correctly. Therefore, the electronic load has come to the fore today.

**2.1. Problem Statement**

Electronic load is a variable resistor. We have another circuit element that can perform the same task, and that is the potentiometer. Just like the electronic load, we can provide current and voltage control in electronic circuits with a potentiometer. However, the potentiometer may not always be available at the desired power. The electronic load takes precedence over the potentiometer here because the electronic load can operate at any current put into it at maximum voltage. If we want to use a variable resistor such as a potentiometer and an electronic load, one of the first options that comes to mind for performing various tests is the use of resistors. However, we have to use different resistors according to each load amount and use many resistors in parallel and in series. Even in the ease of use here, the use of electronic load provides a great advantage.

**2.2. Motivation**

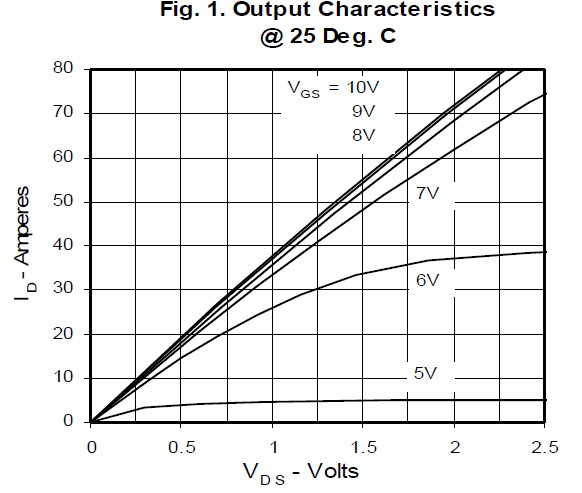
Electronic load is sold at very high prices in the market. The electronic load we will build will be designed in a cheap and simple way. At the same time, we can increase the power ratio of the system by simply changing our power layer. Also, finding a potentiometer that can distribute the watts we want in potentiometers is both expensive and less. Electronic load is used to test the working and performance life of batteries, accumulators and similar power supplies. In addition, it is used to measure whether smps, electronic switched supply sources, dc rectifiers with transformers and such dc supply sources can continuously support direct current and voltage values ​​as described. As a result, we can cheaply test power supply with electronic load.

**3. Litarature Review**

We have been examined a few studies that about electronic load. All studies we have been examined, have nearly the same methodology about electronic loads. One aspect of these and our project, electronic loads aren’t cheap and not fit for alternative use because of the incapability of upgrading or because of the power/price ratio. Three of the studies speak of it and act according to it. Moreover, our project will be cheaper than ready manufactured electronic loads. But studies aim different targets. Even so, all studies have common bullet points and they did provide great awareness and can provide knowledge to understand electronic load methodology further. For example, reference [4] studies small signals that can provide us to deal with noise issues while measuring very small ampere on load. Different type of loads or different futures in loads, serve different purposes. All uses power mosfets in their design because of two major desirable characteristics those the switching speed and power handling capability [1]. For example, in [4], author points out of small power supplies since the energy storage inductor can be made smaller as the frequency increases so mosfets are excellent choice for a electronic load that meant to test DC/DC convertor. But in [2], mentioned capability of mosfets in terms of sunk ampere and power. But in main circuit of [2] fast response time is solved with capacitors. Electronic loads can be used power electronics different ways like has been said before this time not the design or capability of electronic loads but the modes of electronic loads; constant resistant, constant current, constant voltage, constant power. One example of the use of a constant voltage load is to emulate a battery charger. In the second step of a normal battery charger, the voltage is held the same, and the current is decreased from the maximum allowed to very little as the battery charges. If the maximum voltage is exceeded during this step, there is a chance the battery could explode, which is why the power supply would need to be tested to be able to stay stable for the length of the charging time before a battery is fully charged [3]. We will mainly focus on constant current mode that can provide us knowledge if the battery is good or bad, load can get needed feed from battery. Also, current more can provide another future. If we sunk constant feed from a fully charged battery until it is empty while keeping track of time doing it. So, we can learn how big storage of battery in term of ampere hours (Ah/mAh). This way we will know battery has full capacity, it is brand new, used multiple times good battery or simply completed its life cycle dead battery.

**Mosfet Models and Simulations**

Main component of a electronic load is power mosfet. So, our first goal in our project that choosing a appropriate mosfet with accurate model that we can work with. For example, in reference [2] we can verbally see how the mosfet selection is made. We will also share the test results and inferences of the mosfet we will choose for our project. We selected a few mosfets four our project, IXTK80N25, IXTK100N25P, IRFP4668PbF, IRFP260NPbF.We need to investigate these models of mosfets provided by the manufacturers. Now, we need to check if model simulation graphs fit the datasheets.

**IXTK80N25**

****

Figure 1

Figure 2 [7]

We need to focus for our limits of project. For this graph, we just can check for 5V and 6V curves. Because we have no intention to suck current above these curves. 5V curve seems to be accurate for saturation start and sunk values. But, 6V curve enters saturation way above of graph from datasheet [7]. There is 4.5-5 ampere difference.

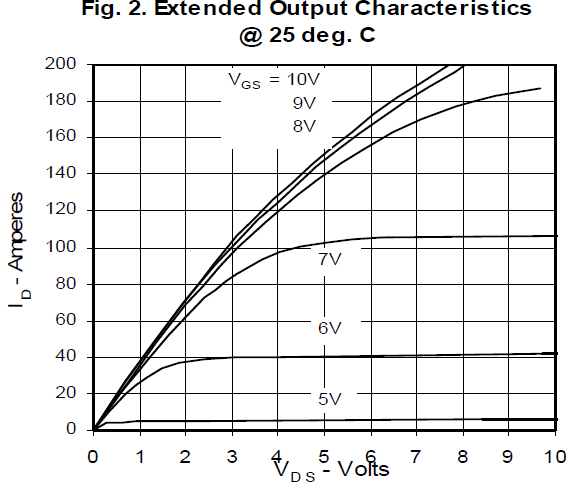


Figure 3

Figure 4 [7]

There is a huge difference for 6V curve but other differences is ignorable. For 5V, difference under a ampere as far as can be read from datasheet characteristic graph.



Figure 5

Figure 6 [7]

Common problem for 3 of the mosfets is temperature. They can’t simulate high temperatures. Just, 6V curve seems to be accurate but simply because the simulation graph has the same output from 25 degree. We can see if we change our Y boundary of graph again that has been changed to compare datasheet graph.



Figure 7

It is nearly the same with 25 degree. 7V curve need to be decrease at 3V but it is the same with previous graph. It is 86.5 ampere, it should be decreased to 50 ampere according to datasheet 125 degree.

**IXTK100N25P**

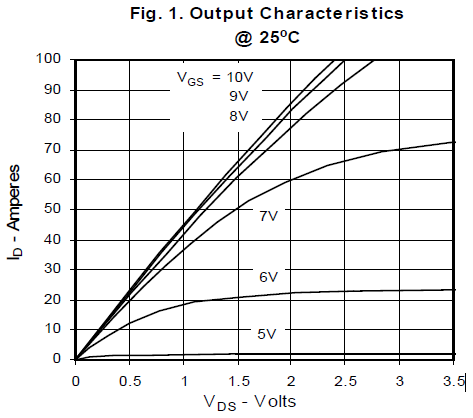


Figure 8

Figure 9 [8]

As you see, IXTK100N25P spice model is not accurate. It didn’t start to trasmit at VGS=5V curve.After trys it start to transmit at 5.3-5.5 V gate voltage. Here is the graph, gate voltage 5.3 to 6(increment 0.1V). Beside that 6V curve more accurate but 7V is higher than it needed to be and enters saturation region earlier.But at the end, except 5V curve it is a pretty accurate model.



Figure 10

It start to transmit with 450mA at 5.4V gate voltage. Before that ID contains at 25u for lower VGS.



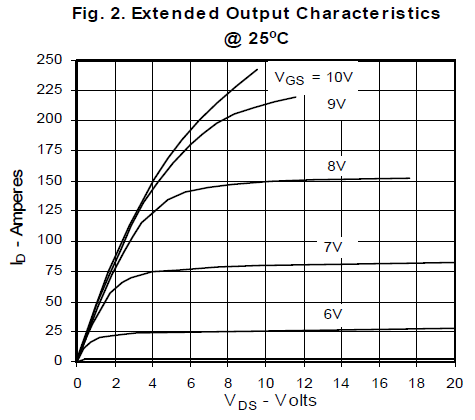
 (From datasheet) [8]

Figure 11 [8]



Figure 12

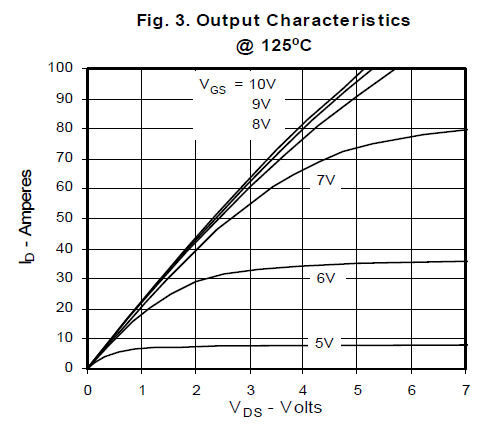
There is difference between graph curves around 4-5 voltage. Other than and late transmission start, model seems accurate. Now we need to check if it is accurate for 125 degree.



Figure 13

Figure 14 [8]

As we can see, IXTK100N25P can’t simulate high temperature case. Temperature doesn’t affect outcome of simulation that much like needed to be.

**IRFP4668PbF**

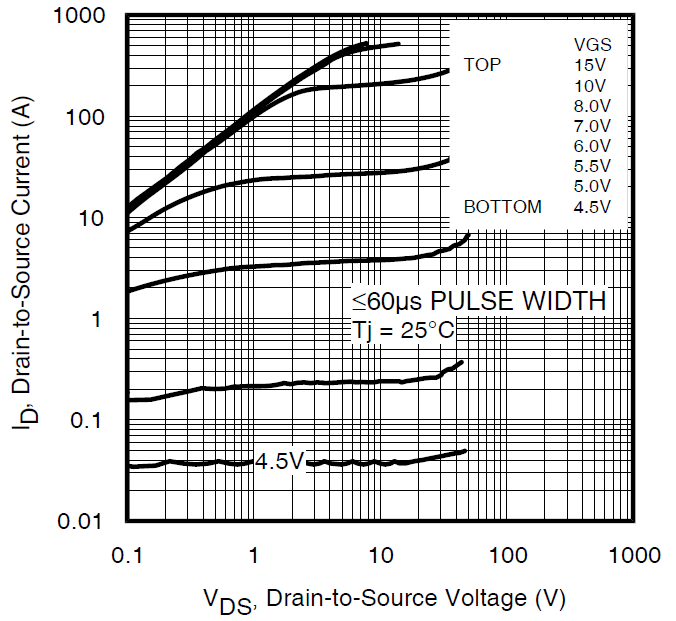
 

Figure 15

Figure 16 [9]

Firstly, 4.5 and 5V curves doesn’t apper on the graph near to 300V because, it not inside of the graph scale.

It is a not a accurate model ,it start to transmit late to. 4.5V curve need to be around 40mA but it is not. It starts from 10mA, it decreases down stable to point at VDS=1V ID=90u. Later it increases up to 20u. After VDS=200V, jumps to 500A and countine to raise unstable.So, it is not seen on graph properly. Also, 4.5 and 5V curves identical. This model is not accurate for gate voltage under 5V.

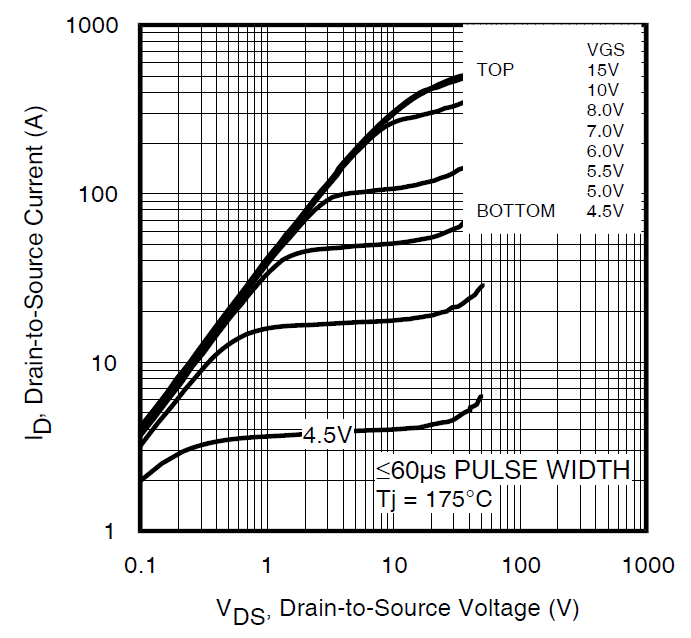
Also for 6 V curve, ID values should be around 20A 🡪 VDS =1V but is is 30V and 30A 🡪 VDS =10V but it is 40A. Now, for 5.5V curve it seems pretty accurate with very small differences. It is really insconsistent for different gate voltages.

Figure 18 [9]



Figure 17

It is not accurate at 175 degree celcius to. Temperature is not affective for this model to. Values are a lot lower than datasheet graphic [9] and again 4.5 and 5V curves more far from accurate. It is a inconsistent model and can’t reflect temperature affect on simulation.

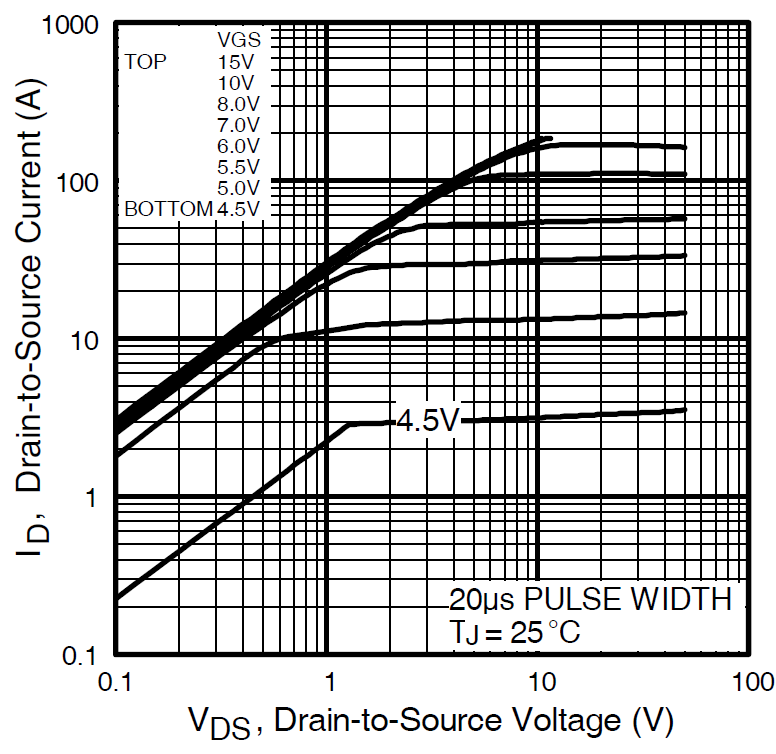
**IRFP260NPbF**

Figure 19 [10]



Figure 20

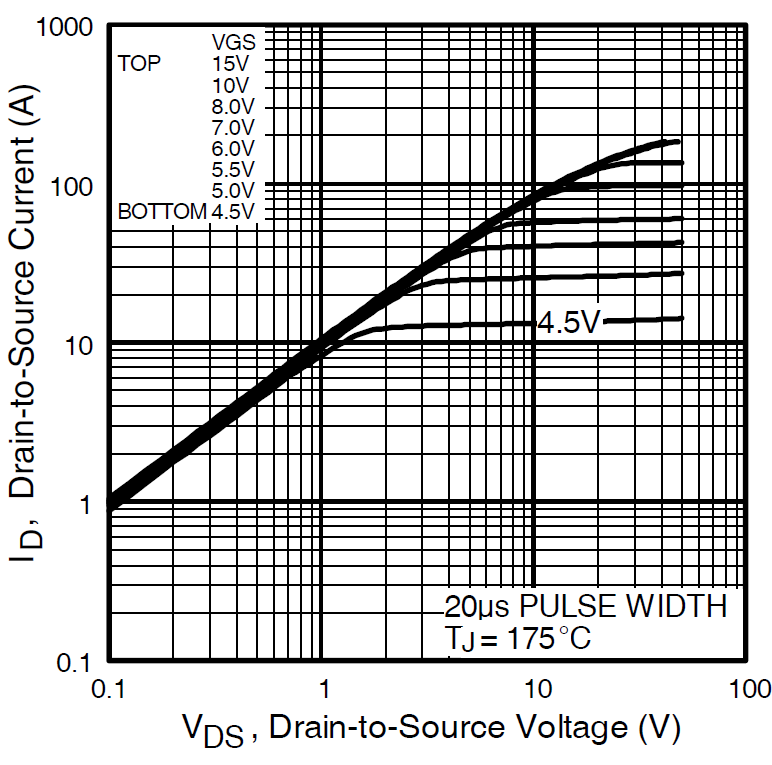


Figure 22 [10]

Figure 21

Like previous simulations, some points are have differences and low current simulated different from expected. But, around the all models IRFP260 [10] seems most accuret because it respond the temperature the most .

**Safe Operating Area and Power Rating for IRFP260**

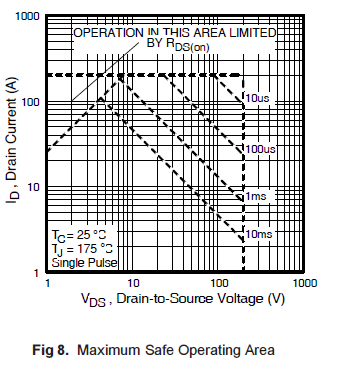
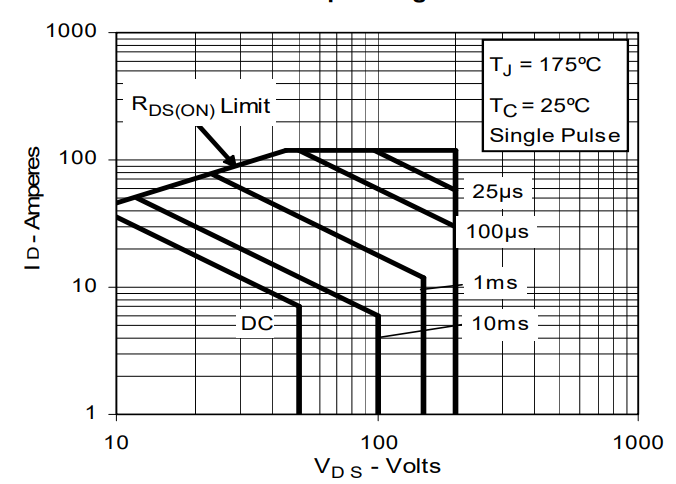


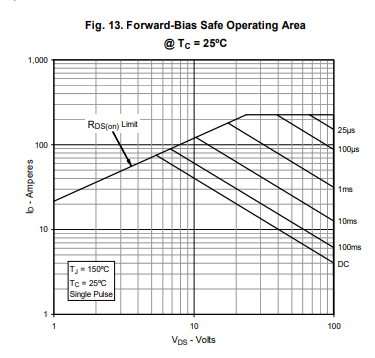
Figure 23 [10]

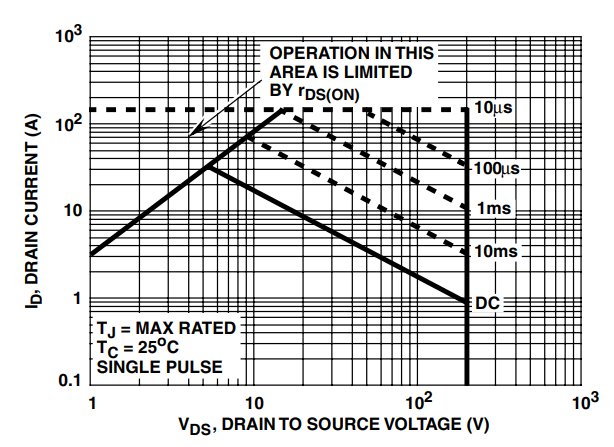
Every Power MOSFET has a Safe Operating Area (SOA) that it can work in and the design has to be cautious of that. The SOA is limited by the drain current and the drain - source voltage’ from reference [1]. Power mosfet usually used for their fast response to on/off operation. So, there is no DC line curve on SOA graph [10]. Our specs or limitations will be lower than 10ms line and it is hard to guess. Considering we are going to work in linear region, it restricts our operation area more. It seems maximum 100 A, 120 V for 10ms line. Also it is different for point of 10ms line, but it should be able to dissipate up to 500 W. But, we will have to work under these parameters if we chose this mosfet. Although we have examined the working of MOSFET models how accurate, these simulations have greatly helped us to understand the working principles of MOSFET.

**IXTQ50N20P (SAFE OPERATİNG AREA)**

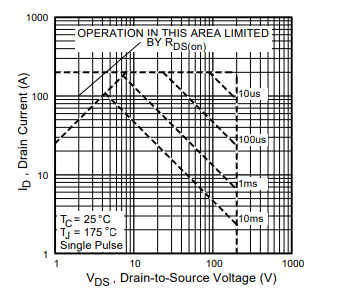


**IXTH75N10L2(SAFE OPERATİNG AREA)**



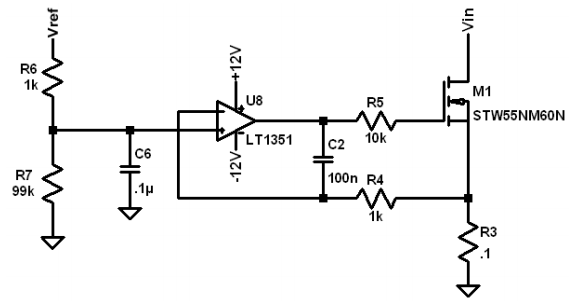
**IRFP250(SAFE OPERATİNG AREA)**

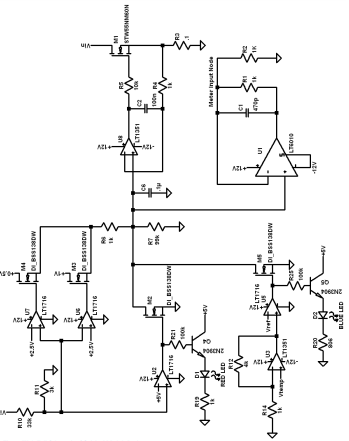
**IRFP260(SAFE OPERATİNG AREA)**



**Design**

The least common denominator of this project is actually a very simple schematic. It has a total of seven components. These components are the backbone of the whole system, form reference [1]. The simplest form of most electronic load circuits is the circuit diagram as seen in the reference [1] Figure. The full schematic of the original module design is Figure , reference [1].

****

****

As can be seen from the design in Figure in reference [11], there is a more advanced electronic load. However, reference [11] operates at a lower voltage than the voltage values ​​we work with.

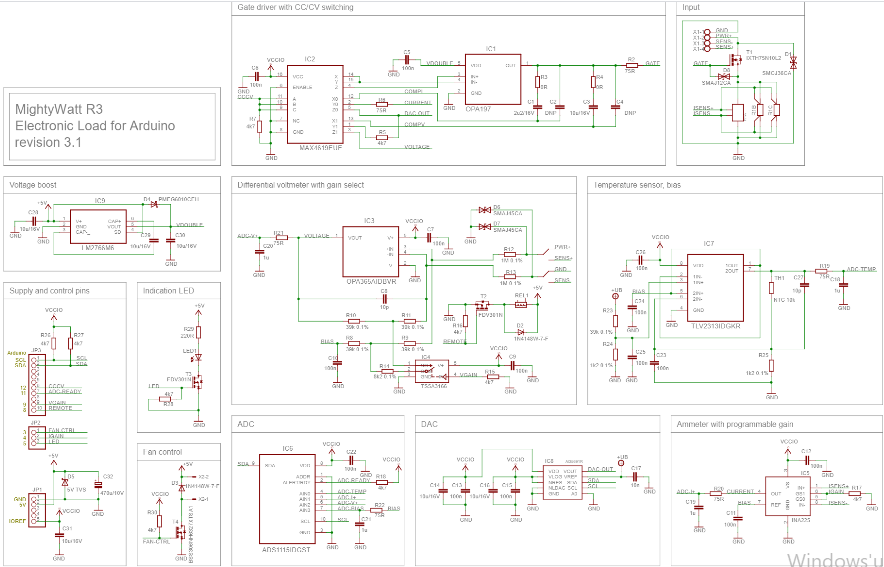
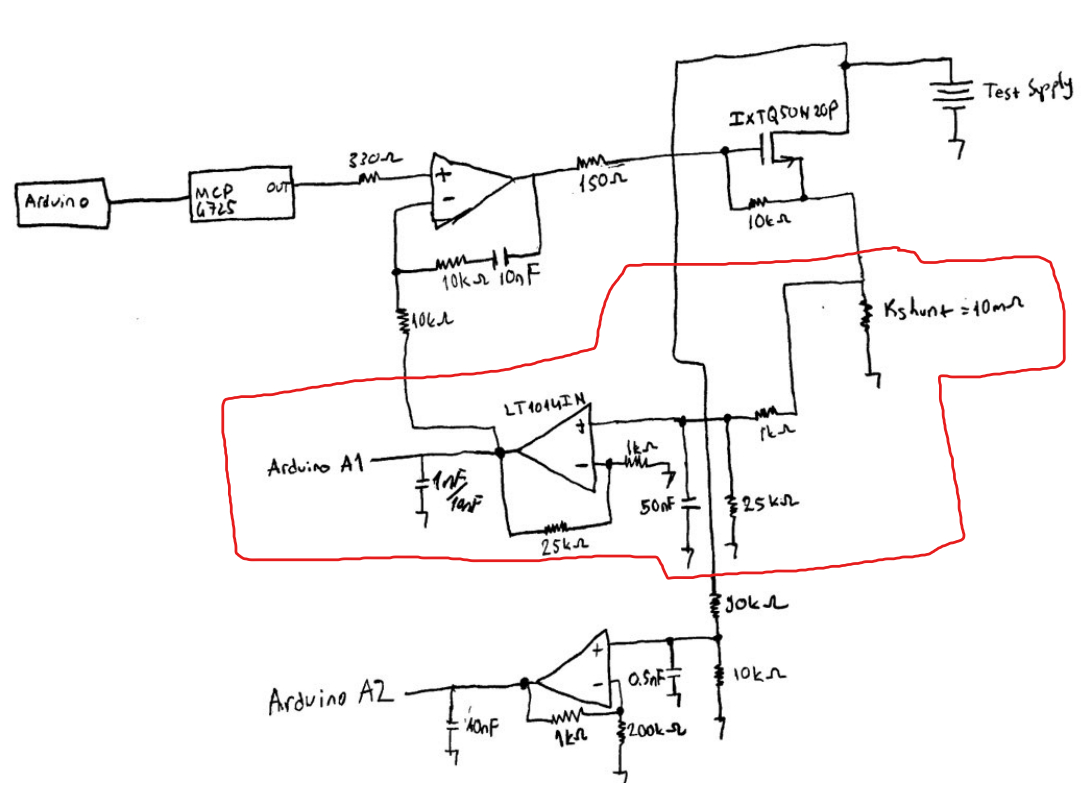
****

Figure shows the circuit we used in our project. Detailed information about the circuit is included in the methodology.



**Microcontroller**

Reference [2] uses the development board that contained Atmel ATXMEGA128A1, 16-bit microcontroller. Since the "digital" part of the design is quite complex and they wanted to use a microcontroller that can handle everything, they used this microcontroller. We used Arduino in our project and the digital design of our project is a comprehensive electronic load.

**Programming**

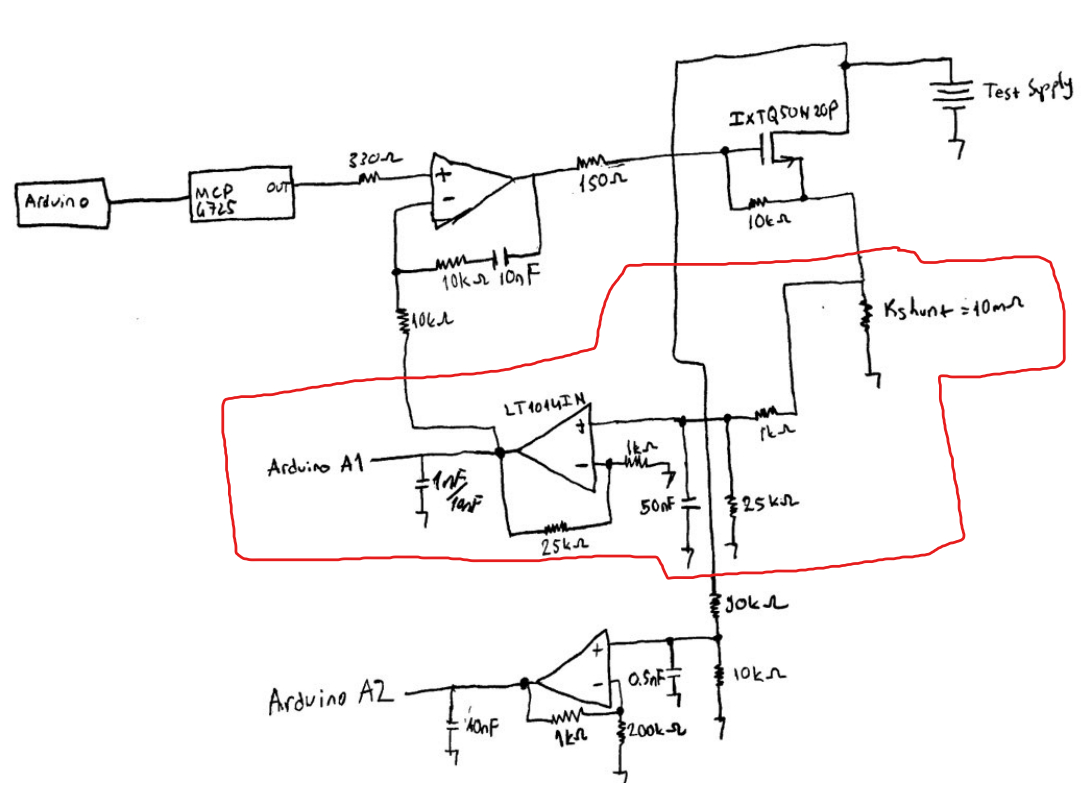
A C program was written and programmed to control the electronic load microcontroller in reference [2]. In reference [3] for the Microchip PIC microcontrollers they were programmed using the MPLABX environment using the C programming language. Also, due to the heat generated in the load cabinet during the test in reference [3], the required heat MATLAB was used to spread it as much as possible. We used C# for programming GUI. The codes we use are in the appendix section.

**4. Methodology**

**4.1. Reading the Current**

The most commonly used method to measure the current drawn is to read the voltage on a shunt resistor where the voltage drop will be less. Or, we the authors actually don’t know any other way. It can be thought that there are two reasons for this. First, the voltage drop does not affect the circuit, second, if the shunts show high resistance, this brings high power loss, hence the need for active cooling because the heat change in the shunt will change the resistance of the shunts and the value we read and calculate.

Before showing the schematic, the reason for using a differential opamp on the shunt is that the differential opamp is suitable for more sensitive measurements. However, unlike our implementation, the most important point to pay attention is that the input ends of the differential opamp should be as close to the shunt and direct as possible. If the negative and positive ends of the shunt need to be moved to other points, they must first go to the input terminals of the differential opamp. This will reduce the noise factor as much as possible and make this sensitive reading healthy.



The region marked with a red stripe is the part that converts our analog signal to digital information and does the actual reading. While choosing the gain values ​​of the shunt and differential opamp here, the current range of the project and the operating voltage of the microcontroller used should be considered. While mentioning this point, one thing that should not be forgotten is the supply voltage of the opamp. There are 4 opamps in the opamp package we use. A single integrated circuit is used for the 3 opamps seen here. As an example, our original plan is that we will already use the Arduino Uno development board and this board will already be connected to a usb connection. Idea was to use the regulator on the Uno development board to power the opamp. This design would work as follows, we would convert our readings to digital with the reference we gave to the development board over the 3.3V logic level. This was necessary because only 5V and 3.3V can be sourced from the Uno development board. However, when opamps approach the supply voltage, they may not give the output they should. This could cause our readings to be wrong as it would change the gain of the opamp. Therefore, we would get an output in the range of 0-3.3V from the one we used as the differential opamp of our opamp package, which we fed with 5V, and we would read it with the 10-bit ADC of the ATmega328P. However, we must feed the opamp with a source above the possible treshold voltage range of your mosfet. Because, although it is 2.5-5V for our mosfet, we tried not to keep our opamp source that we used externally below 9V when we saw that the mosfet did not work during the tests and then was driven with voltages of 5.5V and above.

Although we couldn't get up to 20 amps in our tests, as we said before, we strengthened the voltage on the shunt resistor with a differential opamp and converted the voltage we powered into digital information with the microcontroller's ADC. We set our current limit as 20 amps.

The maximum voltage that the value read from the shunt can take is:

ampere(max)\*rshunt = 20\*10m = 200mV

This means that we want to power 0-200mV in the range of 0-5V. So the gain of our differential opamp has to be 5/200m = 25 times. We chose our resistors accordingly.

Although the resistors are not the part where we have the most problems in reading, it has been beneficial to use resistors with a tolerance of 1% or even 0.1% as the resistors will directly affect the values ​​we will read on the screen. This affects not only the reading but also the current control for our design. More about that later…

This boosted value between 0-5V will take values ​​between 0-1023. However, here lies one of the biggest problems we experienced while doing the project. The source we use to feed the opamps from the outside and the source and test supply have common ground with the circuit and development source, making it worse that the read bit value currently experienced by analog digital converters fluctuates between 2 bits. Therefore, instead of basing a reading and printing it on the user interface, we averaged multiple readings and showed the actual or closer value to the user. Also, and there are RC filters to avoid noise before voltage read and ampere read through opamps. And, capacitor between opamps outputs and ADC inputs of microcontroller helps that fluctuation problem to stabilize more. You can see the relevant code here:

void loop() {

float value1 = analogRead(read1);

float value2 = analogRead(read2);

average1 += value1;

average2 += value2;

if (loopCnt >= 425) {

average1 = average1 / loopCnt;

int average11 = average1;

Serial.print(average11);

Serial.print("/");

average2 = average2 / loopCnt;

int average22 = average2;

Serial.println(average22);

average1 = 0;

average2 = 0;

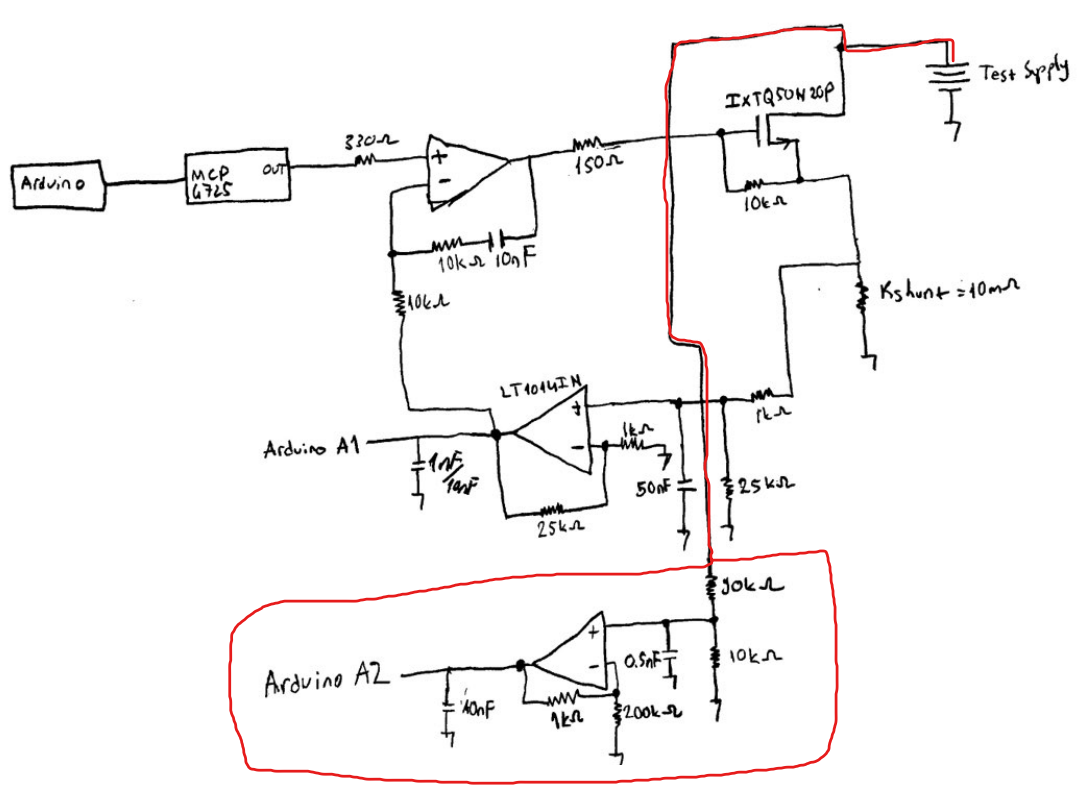
loopCnt = 0;

}

But we drop calculations for GUI(computer) to handle. Even it is not a complicated or heavy task for nowadays computer or even for the 16 MHz ATmege328P microcontroller, and we most certainly didn’t care about delay. You can see that at the capacitor values in our design. We did not put that duty on Arduino Uno. Because, we wanted to be timing of serial prints has regular time gaps so, GUI won’t freeze or crash. More on that later…

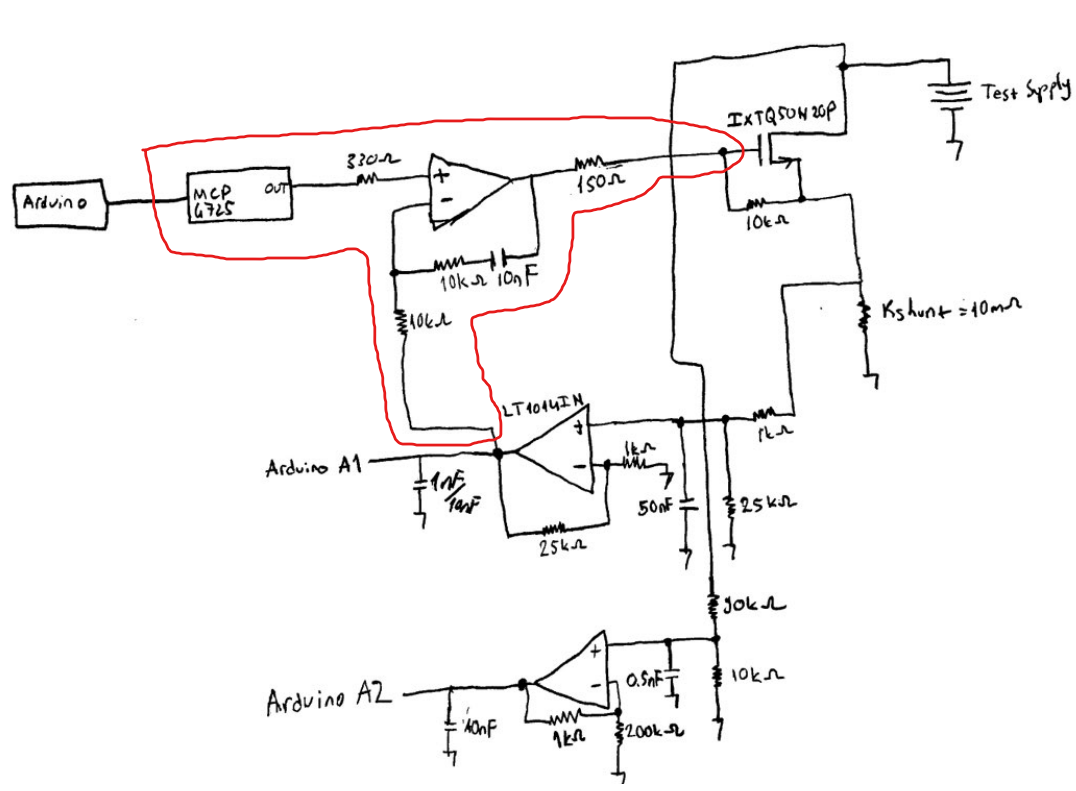
For the last time, we converted 0-200mV voltage drop on our 10m ohm 0-5V voltage signal with our differential opamp configuration. Then we read that signal(ampere and supply voltage) with Uno ADC, and we printed average read values on serial port of UART as text to send read ampere and voltage bit values GUI through serial communication.

Lastly, we did not stand on voltage read much because we didn’t bother about it because we don’t have a constant voltage mode and this mode should be driven with current. Also, a huge voltage drop will be on cables for our project any way. The true voltage may be measured or calculated through ampere, temperature values and mosfet internal resistance relation or putting voltage drop on cables into calculations. But, the configuration that you can see above. We used a voltage divider with 1/10 ratio, and non inverting opamp with 1.005 to be able to read 50V even it is the limit of DC work area for our mosfet. We wasn’t going to test 50V. So, for ampere and voltage, we took the biggest values.



**4.2. Constant Current Control**

First of all, the comparator opamp does all the job for us to control constant ampere. What it actually does? A normal comparator without a feedback is like a logic gate for equal or not or greater or not if the reference value is the 0 or DC offset. With a delayed feedback comparator, raises or drops the output voltage until other input voltage it equals to the reference. Here the related part:



For this all part a PID controller may fit better the situation, but we went with the comparator configuration. But either way, we need an digital analog converter(DAC) and usually a microcontroller doesn’t have. We can’t use the PWM function fort his job because, if our voltage reference a PWM function we will always send logic high voltage level of microcontroller when it is at on duty cycle. And our reference for comparator opamp will be 0 and 5V always according to duty cycle. It is okey when driving an LED because the average is the same with wanted signal value. But it’s not an analog signal so we can’t control our electronic load digitally without and DAC.

But, before we mentioning about DAC, we face a problem with the drive of mosfet.

On the graph above, blue is the percentage error of between setted ampere to draw and actual ampere sunk. This proportional difference normally increased as the amperage value increased. Because the average percentage error was more or less the same. This difference, which is not a problem in small currents, reached unacceptable limits after 1 amp. While we can't figure out why this happens, it could be due to the comparator opamp's integral gain. We solved this problem by simply adding this proportion to our calculations in the GUI and changing the data we sent to the development board according to these calculations. After the correction, our errors drop to 0-30mA range. But by changing the resistor and capacitor values in the feedback for the comparator opamp, this problem should be able to solve for specifik mosfet and opamp configurations.

We used MCP4725. It has I2C communication protocol, has couple Arduino libraries. So, it is pretty straight forward and easy to use. We send setted value from GUI to ATmega328P trough UART as text. And set MCP4725 according to it with Adafruit library as in 0-5V range reference voltage. Related code:

while (Serial.available() > 0) {

char chr = Serial.read();

if (chr == '\n') {

int dacInt = DACbitSTR.toInt();

DACbitSTR = "";

dac.setVoltage(dacInt, false);

loopCnt2 = 0;

}

else {

DACbitSTR += chr;

}

}

if (loopCnt2 >= 6001) {

dac.setVoltage(0, false);

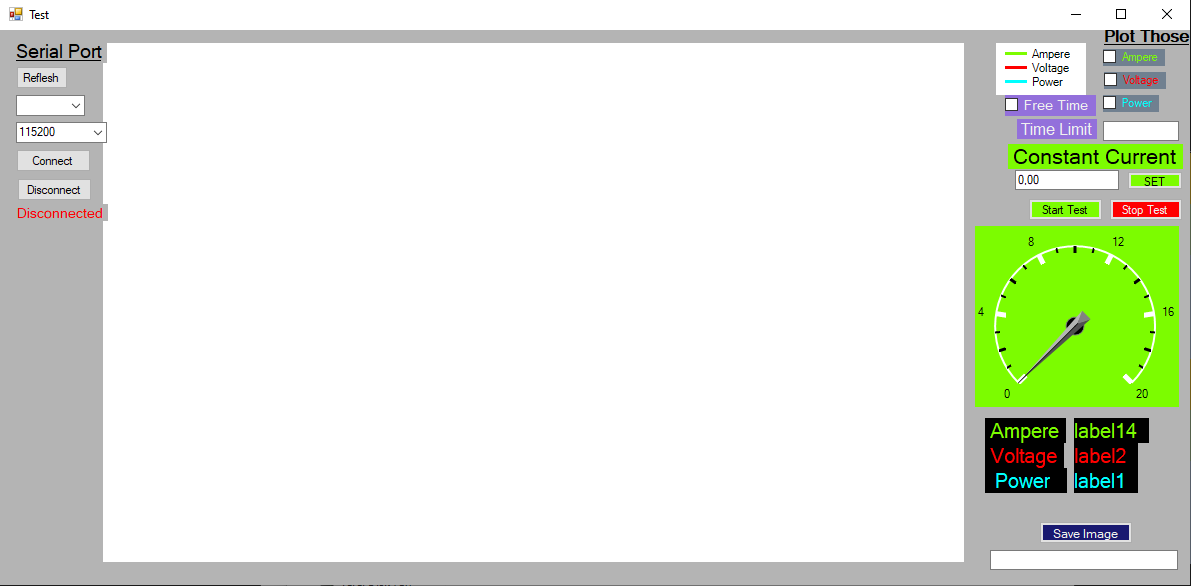
loopCnt2 = 0;

}

Also we considered, possible crush of GUI or PC. So, send this text information about wanted sunk ampere value from GUI. And, check if it coming regularly, if there is a change, we set the DAC again. If nothing comes we set the reference trough DAC 0V to prevent undesired problems.

Lastly, as you can see before on schematic of drive part of circuit. The output of differential opamp goes into as second input for our comparator. Before mentioned 1% or 0.1% tolerant resistance needs is going to matter at that point. Without considering the possible integral gain problem of comparator opamp if the output of differential opamp is not exactly correct for our calculations, then there will be always a difference between actual ampere sunk and the desired value which setted on GUI. So, output of differential opamp should be very precise in order to draw right amount of current through mosfet. After solving comparator problem, we changed 5% tolerant resistance with 1% ones. Differece drop from 0-30mA to 0-10mA. This is why low tolerant resistances should be use to duplicate our design. Or we should do the calculations according to it. We should get the range of differential opamp according to voltage drop on shunt and precise measured but unwanted resistance values.

**4.3. GUI**



First off all, let's talk about what our UI can do.

- Different serial communication speeds can be used according to your needs. However, we recommend the maximum speed supported by your microcontroller for the maximum sampling.

-The tests can be stopped manually or by setting a time limit before starting the test.

-The constant current value to be tested can be changed during the test.

-It can show instantaneous current, voltage and power information and convert them into graphics.

-These graphs can be saved on your PC as a image so that the tests can be reviewed later.

If the Windows Form user interface we have designed is to be imitated, the first thing to pay attention to is to avoid the freezing and crashing problems that we mentioned above in the methodology. In our user interface, the bit values corresponding to the signal read by the microcontroller are written to the serial monitor in text type. The text values written to the serial monitor are read by the timer element in our user interface at the time intervals determined when creating this element. Therefore, while the microcontroller is writing these values to the serial monitor, if the user interface tries to read these values, if these two processes conflict, it causes the test to stop.

int average11 = average1;

Serial.print(average11);

Serial.print("/");

average2 = average2 / loopCnt;

int average22 = average2;

Serial.println(average22);

private void timer1\_Tick(object sender, EventArgs e)

{

try

{

//read

string reads = serialPort1.ReadLine();

string[] values = reads.Split('/');

serialPort1.DiscardInBuffer();

Here you can see first the firmware then the software code. We print and read voltage and current values splitting them with a slash. So, if GUI try to read, before microcontroller print value/value to serial monitör, because there is nothing to split according to slash 🡪 ‘value1’ or there is nothing after the splash ‘value1/’ , the program doesn't know what to do.

We try to calculate and test according to delay of analog reads of microcontroller’s ADC, and set timer in the gui according to it. So, when we started the test, if there is no overlap on the start, most probably won’t be during test.

The second thing we did is we made the graph plot a transparent curve. This is because the limits of the graph are determined automatically according to the values. In this way, if the value drawn in case it coincides, or if the largest of the values met a full limit, there would be a hard-to-read graph that looked halfway at the limit of the graph.

//plot

if (plotA.Checked == true)

{

chart1.Series["Ampere"].Points.AddXY(time, ampere);

ampered = ampere;

}

if (plotV.Checked == true)

{

chart1.Series["Voltage"].Points.AddXY(time, voltage);

voltaged = voltage;

}

if (plotP.Checked == true)

{

chart1.Series["Power"].Points.AddXY(time, power);

powered = power;

}

double maximus = Math.Max(ampered, voltaged);

maximus = Math.Max(maximus, powered);

chart1.Series["Series4"].Points.AddXY(time, maximus + 0.1f);

As you can see at the bottom of this set of code, we have plotted 0.1 more of the current, voltage and power values, whichever value is the largest.

After that comes the part of controlling the current drawn. We took the current determined by the user in a text box, converted this value into a number, calculated over 4096 bits according to the DAC we used, and sent this information as text over the serial port with a timer element in GUI. In short, we repeat the method we used when reading voltage and current values. The only difference is that the microcontroller read these values digit by digit. In addition, we consider possible crash or problems and we check if the information about comes regularyly. That’s why we used a timer element in GUI again. Because if GUI crashes, microcontroller will be drawn setted current until the new current decided after re-opening the GUI. Here you can see the code:

private void timer2\_Tick(object sender, EventArgs e)

{

//constant current timer

try

{

float ampereFloat = Convert.ToSingle(constantCurrentString);

ampereFloat = ampereFloat / 20f \* 4096f / 103.61f \* 100f;

int intToSendDAC = Convert.ToUInt16(ampereFloat);

string strToSendDAC = Convert.ToString(intToSendDAC);

serialPort1.WriteLine(strToSendDAC);

}

catch (Exception ex1)

{

timer1.Stop();

timer2.Stop();

try

{

serialPort1.WriteLine("0");

}

catch (Exception ex2)

{

MessageBox.Show(ex2.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

MessageBox.Show(ex1.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

}

Also you can see with red at the beginning of the code, we added the error after the test we done. That will be explained over in result and discussion part later. Above you can see the firware code and red part is the verification part if current should be drawn still.

while (Serial.available() > 0) {

char chr = Serial.read();

if (chr == '\n') {

int dacInt = DACbitSTR.toInt();

DACbitSTR = "";

dac.setVoltage(dacInt, false);

loopCnt2 = 0;

}

else {

DACbitSTR += chr;

}

}

if (loopCnt2 >= 6001) {

dac.setVoltage(0, false);

loopCnt2 = 0;

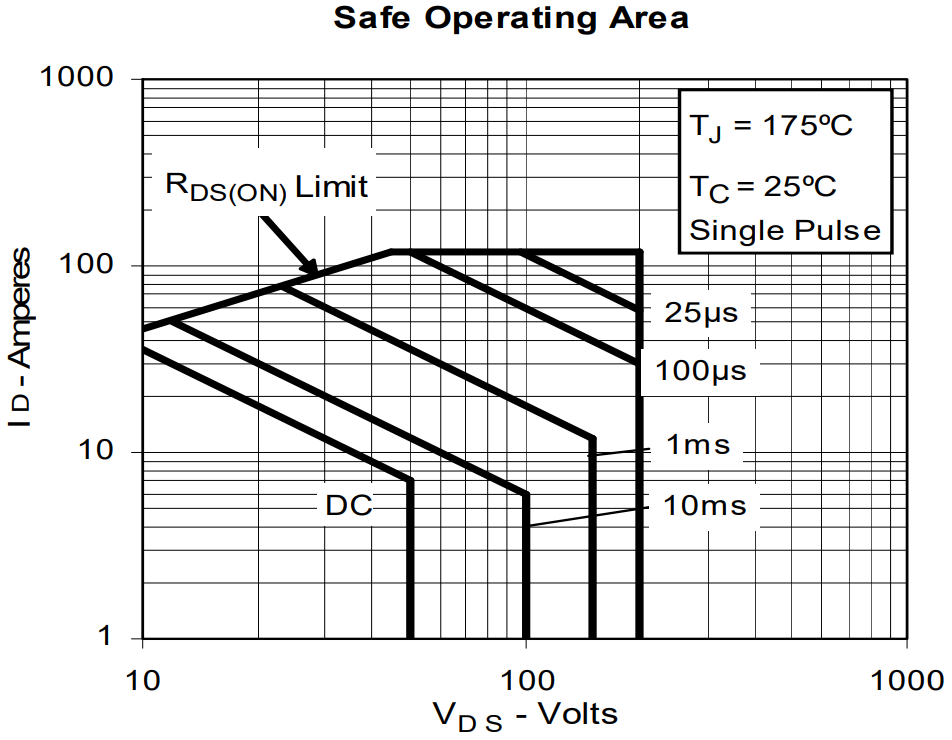
}

loopCnt++;

loopCnt2++;

**4.4. Mosfet and Opamp Selection**

At the litarature review, you can see the previous research of us for FENG497. As you can see, it becomes difficult to determine the voltage and current values that the mosfet can handle, for the pulse values you will work with, or if you are working in DC, if the DC lines are not given by the manufacturer in the safe operating are graphs. That's why we recommend you to use or choose linear mosfets. This is because mostly linear mosfets have a SOA with DC characteristic in their datasheets. This will be important for you to know the limits of your mosfet. Mosfets are largely used for very fast switch operation or working with pulses . However, in a test with an electronic load, this delay will not be that important as the mosfet will start to work once. Therefore, although they are known to be slow, linear MOSFETs can be used for electronic loads easily.



We can see here the DC characteristics of IXTQ50N20P. Although working close to the DC limits can be dangerous for the mosfet, we see here, it gives us an idea of the DC limits of the mosfet. In this way, you will not need to do a lot of energy calculations for DC characteristics based on other pulse characteristics when choosing your mosfet. Lastly, we didn't care the limits that much because we couldn't test at voltages and currents. Anyway, a desktop stock CPU fan that can be recycled can easily handle heat for our mosfet up to 10-15 amps.

Now, the opamp selection can be reviewed. The two most important factors to consider when choosing an opamp are: First, the offset voltages for the differential opamp you will use to read the values on the shunt should be low enough for your project. Secondly, the voltage supply value that your mosfet can handle must be above the gate treshold voltage of your mosfet.

**5.Results and Discussion**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setted** | **Source Measure** | **GUI** | **Diffirence(Multimeter - GUI)** | **Multimeter - Setted** |
| 0,05 | 0 | 0 | 0 |  |
| 0,1 | 0,09 | 0,01955 | 0,07045 | -0,01 |
| 0,15 | 0,14 | 0,0782 | 0,0618 | -0,01 |
| 0,2 | 0,2 | 0,1173 | 0,0827 | 0 |
| 0,25 | 0,25 | 0,17595 | 0,074047 | 0 |
| 0,3 | 0,3 | 0,21505 | 0,084947 | 0 |
| 0,35 | 0,36 | 0,2737 | 0,086296 | 0,01 |
| 0,4 | 0,41 | 0,33236 | 0,077645 | 0,01 |
| 0,45 | 0,47 | 0,37145 | 0,09855 | 0,02 |
| 0,5 | 0,52 | 0,4496 | 0,0704 | 0,02 |
| 0,55 | 0,58 | 0,5083 | 0,0717 | 0,03 |
| 0,6 | 0,64 | 0,5474 | 0,0926 | 0,04 |
| 0,65 | 0,69 | 0,60606 | 0,08394 | 0,04 |
| 0,7 | 0,73 | 0,6451 | 0,0849 | 0,03 |
| 0,75 | 0,78 | 0,6842 | 0,0958 | 0,03 |
| 0,8 | 0,83 | 0,7429 | 0,0871 | 0,03 |
| 0,85 | 0,87 | 0,782 | 0,088 | 0,02 |
| 0,9 | 0,92 | 0,8211 | 0,0989 | 0,02 |
| 0,95 | 0,97 | 0,8797 | 0,0903 | 0,02 |
| 1 | 1,03 | 0,9384 | 0,0916 | 0,03 |
| 1,05 | 1,08 | 0,9775 | 0,1025 | 0,03 |
| 1,1 | 1,14 | 1,0361 | 0,1039 | 0,04 |
| 1,15 | 1,19 | 1,0948 | 0,0952 | 0,04 |
| 1,2 | 1,25 | 1,1534 | 0,0966 | 0,05 |
| 1,25 | 1,3 | 1,21212 | 0,08788 | 0,05 |
| 1,3 | 1,35 | 1,2512 | 0,0988 | 0,05 |
| 1,35 | 1,4 | 1,29 | 0,11 | 0,05 |
| 1,4 | 1,45 | 1,3489 | 0,1011 | 0,05 |
| 1,45 | 1,49 | 1,388 | 0,102 | 0,04 |
| 1,5 | 1,54 | 1,4271 | 0,1129 | 0,04 |
| 1,55 | 1,58 | 1,4662 | 0,1138 | 0,03 |
| 1,6 | 1,64 | 1,5249 | 0,1151 | 0,04 |
| 1,65 | 1,7 | 1,5835 | 0,1165 | 0,05 |
| 1,7 | 1,75 | 1,64223 | 0,107772 | 0,05 |
| 1,75 | 1,81 | 1,7008 | 0,1092 | 0,06 |
| 1,8 | 1,87 | 1,75953 | 0,11047 | 0,07 |
| 1,85 | 1,93 | 1,81818 | 0,111819 | 0,08 |
| 1,9 | 1,98 | 1,8572 | 0,1228 | 0,08 |
| 1,95 | 2,02 | 1,916 | 0,104 | 0,07 |
| 2 | 2,07 | 1,955 | 0,115 | 0,07 |
| 2,05 | 2,12 | 1,9941 | 0,1259 | 0,07 |
| 2,1 | 2,16 | 2,0527 | 0,1073 | 0,06 |
| 2,15 | 2,21 | 2,0918 | 0,1182 | 0,06 |
| 2,2 | 2,26 | 2,1309 | 0,1291 | 0,06 |
| 2,25 | 2,32 | 2,1896 | 0,1304 | 0,07 |
| 2,3 | 2,38 | 2,2482 | 0,1318 | 0,08 |
| 2,35 | 2,43 | 2,3069 | 0,1231 | 0,08 |
| 2,4 | 2,49 | 2,3655 | 0,1245 | 0,09 |
| 2,45 | 2,55 | 2,4242 | 0,1258 | 0,1 |
| 2,5 | 2,61 | 2,4828 | 0,1272 | 0,11 |
| 2,55 | 2,66 | 2,5219 | 0,1381 | 0,11 |
| 2,6 | 2,7 | 2,561 | 0,139 | 0,1 |
| 2,65 | 2,75 | 2,6197 | 0,1303 | 0,1 |
| 2,7 | 2,8 | 2,6588 | 0,1412 | 0,1 |
| 2,75 | 2,84 | 2,6979 | 0,1421 | 0,09 |
| 2,8 | 2,89 | 2,7565 | 0,1335 | 0,09 |
| 2,85 | 2,94 | 2,7956 | 0,1444 | 0,09 |
| 2,9 | 3 | 2,8543 | 0,1457 | 0,1 |
| 2,95 | 3,06 | 2,913 | 0,147 | 0,11 |
| 3 | 3,11 | 2,9716 | 0,1384 | 0,11 |
| 3,05 | 3,17 | 3,0107 | 0,1593 | 0,12 |
| 3,1 | 3,23 | 3,069 | 0,161 | 0,13 |
| 3,15 | 3,28 | 3,128 | 0,152 | 0,13 |
| 3,2 | 3,33 | 3,1671 | 0,1629 | 0,13 |
| 3,25 | 3,37 | 3,2258 | 0,1442 | 0,12 |
| 3,3 | 3,42 | 3,2649 | 0,1551 | 0,12 |
| 3,35 | 3,47 | 3,304 | 0,166 | 0,12 |
| 3,4 | 3,51 | 3,3431 | 0,1669 | 0,11 |
| 3,45 | 3,56 | 3,4017 | 0,1583 | 0,11 |
| 3,5 | 3,62 | 3,4604 | 0,1596 | 0,12 |
| 3,55 | 3,67 | 3,4995 | 0,1705 | 0,12 |
| 3,6 | 3,73 | 3,5581 | 0,1719 | 0,13 |
| 3,65 | 3,79 | 3,6168 | 0,1732 | 0,14 |
| 3,7 | 3,85 | 3,6754 | 0,1746 | 0,15 |
| 3,75 | 3,91 | 3,7341 | 0,1759 | 0,16 |
| 3,8 | 3,95 | 3,7732 | 0,1768 | 0,15 |
| 3,85 | 4 | 3,8123 | 0,1877 | 0,15 |
| 3,9 | 4,05 | 3,8709 | 0,1791 | 0,15 |
| 3,95 | 4,09 | 3,91 | 0,18 | 0,14 |
| 4 | 4,14 | 3,9491 | 0,1909 | 0,14 |
| 4,05 | 4,18 | 3,9882 | 0,1918 | 0,13 |
| 4,1 | 4,23 | 4,0469 | 0,1831 | 0,13 |
| 4,15 | 4,29 | 4,1055 | 0,1845 | 0,14 |
| 4,2 | 4,35 | 4,1642 | 0,1858 | 0,15 |
| 4,25 | 4,4 | 4,2033 | 0,1967 | 0,15 |
| 4,3 | 4,46 | 4,2619 | 0,1981 | 0,16 |
| 4,35 | 4,52 | 4,3206 | 0,1994 | 0,17 |
| 4,4 | 4,57 | 4,3792 | 0,1908 | 0,17 |
| 4,45 | 4,62 | 4,4183 | 0,2017 | 0,17 |
| 4,5 | 4,67 | 4,4574 | 0,2126 | 0,17 |
| 4,55 | 4,71 | 4,5161 | 0,1939 | 0,16 |
| 4,6 | 4,76 | 4,5552 | 0,2048 | 0,16 |
| 4,65 | 4,8 | 4,5943 | 0,2057 | 0,15 |
| 4,7 | 4,85 | 4,6529 | 0,1971 | 0,15 |
| 4,75 | 4,91 | 4,7116 | 0,1984 | 0,16 |
| 4,8 | 4,98 | 4,7702 | 0,2098 | 0,18 |
| 4,85 | 5,04 | 4,8289 | 0,2111 | 0,19 |
| 4,9 | 5,09 | 4,8875 | 0,2025 | 0,19 |
| 4,95 | 5,15 | 4,9462 | 0,2038 | 0,2 |
| 5 | 5,2 | 4,9853 | 0,2147 | 0,2 |

Above you can see first result of our build. The differences between the read values and the actual values(Multi-GUI) have been largely eliminated up to 5 ampere. We added a parabolic function extracted from the samples we took to the calculations. We have added this function and the calculations which is suitable for up to 1.35-5A in the GUI. Also, for lower values before 1.35 ampere, we set proportional constants. These problem can be caused by a lot of problems. But, we consider the problem caused by the actual build. Because our build was not any good. Because signal transported with cables instead of solid conductive materials or our build has no PCB. Other problem which mentioned and solved with taking average reads a problem which can occur at finest PCB design which microcontroller has been feed directly from regulator with short distance tracks and ADC legs aren’t interfered with noise. That problem not our difference problem, makes the second digit after the comma to be exact value difficult. You also see this in [11] MightyWattR3 design test videos on YouTube. For these reasons, designer plot 2-3 times a read in a second. It was more difficult for us to overcome this problem because we had the graph drawn more with more samples (about 20 per second). But the other problem with our reading is that the difference between the reading and the true value problem can only be explained by the drop in wires. It has nothing to do with averaging to solve bits not jumping by two values problem. If this design is to be build by someone, for the problem bits jumping between 2 value, with a PCB design which should follow; ADC pins (or tracks between ADC pins and signals), powering microcontroller or the given reference microcontroller for ADC(analog reads) should not be interfered with noise or any other irregularties. And, taking average values and dismissing the not possible values which is error should help to stabilize this situation up to some point. But for our problem short distance and maybe with enough thickness PCB track between signal and ADC pins and not circulating ground(reference of circuit) so much can solve difference problem. But any way, here the graph of the error of reads above with yellow line and our fix code for that up to 5 amperes. In addition, how big is the range of samples, this application will fit in that range. Here code and the graph:

if (ampere > 1.36)

{

double eqInC = (ampere \* ampere \* 0.434733819) - (3.930096977 \* ampere)

+ 12.91697891;

ampere = ampere \* (eqInC + 100.0) / 100.0;

}

else if (ampere > 0.01 && ampere < 0.02)

{

ampere = ampere \* 5;

}

else if (ampere > 0.02 && ampere < 0.07)

{

ampere = ampere \* 3;

}

else if (ampere > 0.07 && ampere < 0.17)

{

ampere = ampere \* 1.7;

}

else if (ampere > 0.17 && ampere < 0.216)

{

ampere = ampere \* 1.43;

}

else if (ampere > 0.216 && ampere < 0.32)

{

ampere = ampere \* 1.35;

}

else if (ampere > 0.32 && ampere < 0.87)

{

ampere = ampere \* 1.19784005;

}

else if (ampere > 0.87 && ampere < 1.037)

{

ampere = ampere \* 1.0920181452;

}

else if (ampere > 1.037 && ampere < 1.36)

{

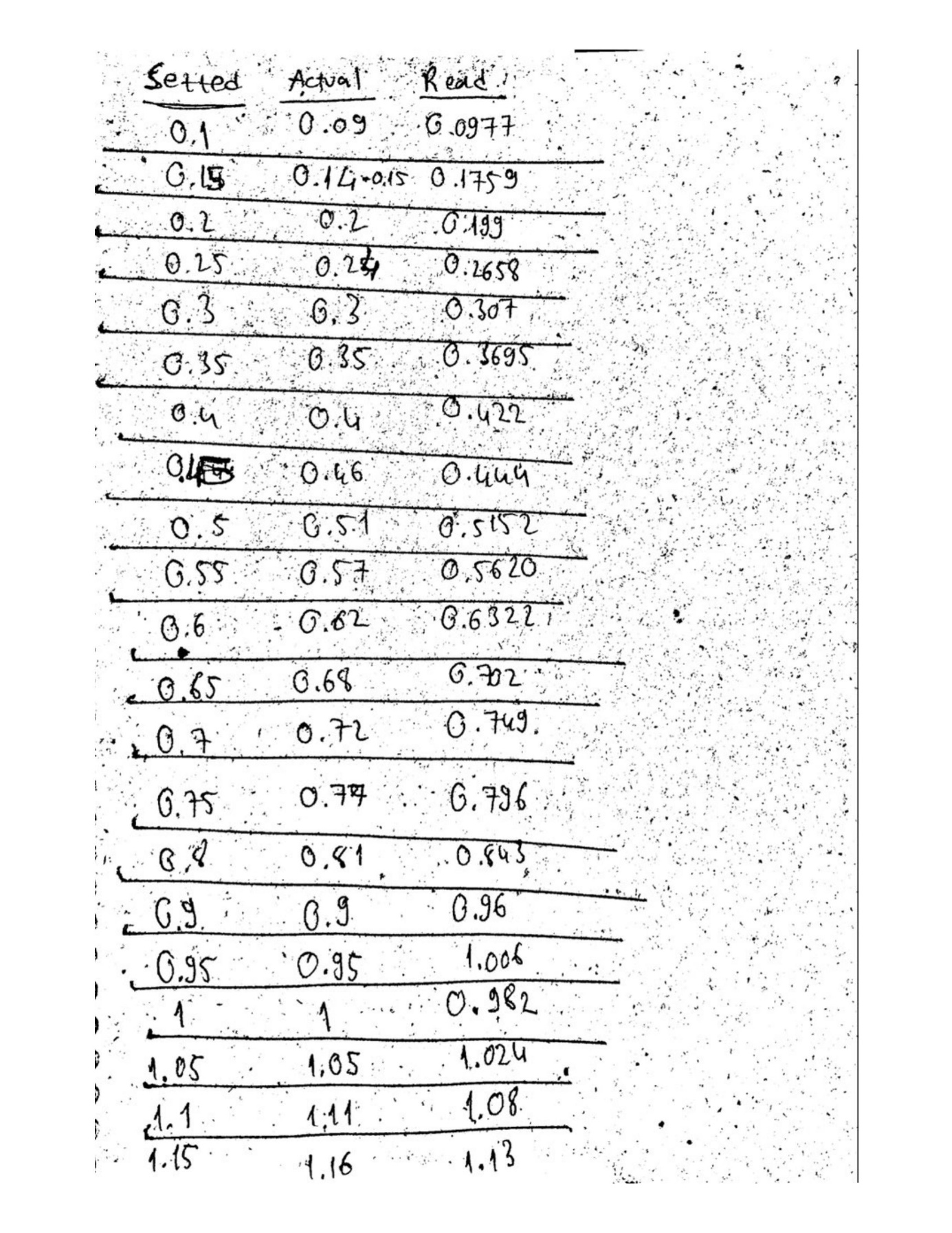
ampere = ampere \* 1.0743934586;

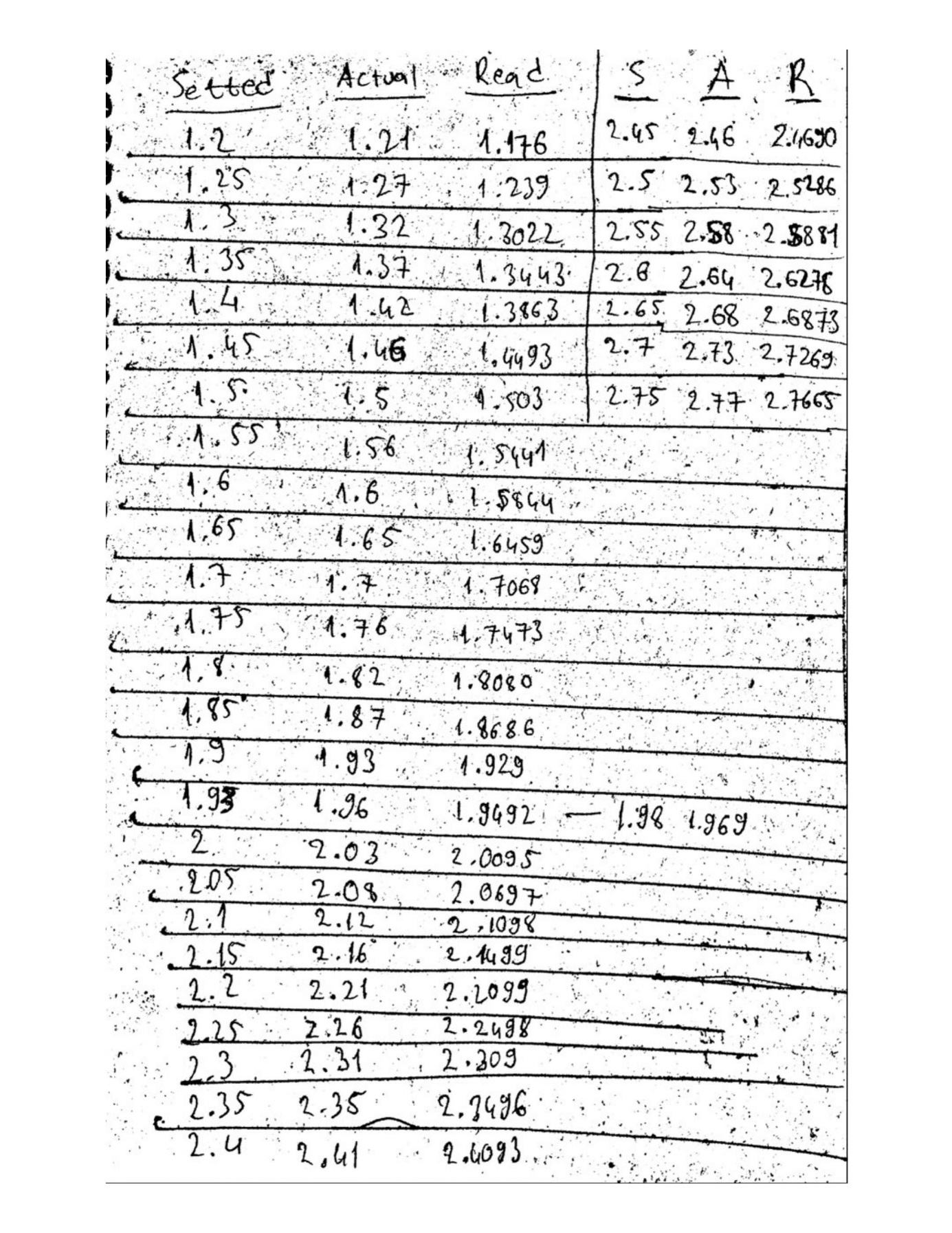
}

Second problem you can obserbe test table is the difference between setted(by the user) and actual ampere. This problem probably caused by non-tuned feedback of our comparator. You can consider to apply integral gain calculations to your application and select feedback resistance and capacitor parameters according to it. Furthermore, perfect solution for this can be using a PID control algorithm instead of a comparator opamp, and controlling mosfet with directly a DAC or a DAC with an buffer opamp according to PID control results. But in our example, our error was always proportinal in a range. And we took average of error and applied to calculations inside of GUI if you remember the previous mention of it:

ampereFloat = ampereFloat / 20f \* 4096f / 103.61f \* 100f;

At the end after adding these 2 solutions, we thought we solved the problem because result tested only with multimeters. Here is the results of tests.





Here we can see the result are really close to actual ampere values. And we thought we succeded to build our electronic load. But even we solved some problems our electronic load doesn’t work properly because there is shakey output at the comparator opamp and the ampere sunk. Here you can see an oscilloscope measure image of ampere sunk by our electronic load.



Here ampere to sunk by our build setted to 5 amp. This measurement is on our shunt which is 10m ohm. The measurement shakes 0-0.5mV, values small than that shakes less and values bigger than that shakes more. Result of this, ampere sunk is shakes 0-50mA at 5 amps. So, it is not constant current and we couldn’t accomplish our goal. This can be caused by pyhsical bad build of our design or integral gain of the comparator opamp.And again, this may can be solved, like we said before, with tuning for exactly used opamp and mosfet, or using a PID control system instead of comparator opamp to control mosfet.

**6. Conclusions**

Since we have clearly stated our suggestions for the problems in our design, the problems that may occur in the construction of the circuit and the problems that occurred for us, in the two main sections above. We can list possible solutions and useful features that can be added, considering that this project has not fully worked.

As a solution, the thing that gave us the most trouble was that the circuit was not built properly. For this, we recommend that you build your circuit on a PCB that can meet the needs of your circuit, taking into account our recommendations on PCB and circuit design that we mentioned in the sections above. In addition, as we said before, a PID system may easily overcome the biggest of problems, which is the uncontrollability and fluctuation of the current.

As features that can be added, constant voltage, constant resistance, constant power mode can be added as features as addition to constant current mode. Although the constant voltage mode is not that simple to implement, the constant resistance and constant power mode can be added with simple operations and calculations on the user interface or microcontroller, considering the voltage and current are properly measured. And, it can be pretty useful features. Finally, a temperature management system can be made using a fan-driven heat sink (just like in personal computers).

**7. References**

[1] Jason March, Modular Electronic Load A Senior Project Presented to The Faculty of the Electrical Engineering Department California Polytechnic State University, San Luis Obispo, August 2010

[2] Jason March, DIGITALLY CONTROLLED, MODULAR ELECTRONIC LOAD, A Master’s Thesis presented to The Faculty of California Polytechnic State University, San Luis Obispo, December 2011

[3] Justin Moses, Design and Testing of an Electronic Load Board for Power Supply Validation and Verification, Auburn University, May 4, 2014

[4] Lukas Rosen, Sahar Samimi, Small Electronic Load, Design and analysis of a small electronic load for testing on-board DC/DC converters, Chalmers University Of Technology, Goteborg, Sweden 2012

[5] https://www.allaboutcircuits.com ,accessed on February in 2021.

[6]<https://www.littelfuse.com/~/media/electronics/datasheets/discrete_mosfets/littelfuse_discrete_mosfets_n-channel_trench_gate_ixfn230n20t_datasheet.pdf.pdf> , accessed on February in 2021

[7] <https://ixapps.ixys.com/Datasheet/98953.pdf>, accessed on February in 2021

[8] <https://datasheetz.com/data/Discrete%20Semiconductor%20Products/MOSFETs%20-%20Single/IXTK100N25P-datasheetz.html>, accessed on February in 2021

[9]<https://www.infineon.com/dgdl/irfp4668pbf.pdf?fileId=5546d462533600a40153562c8528201d>, accessed on February in 2021

[10]<https://www.infineon.com/dgdl/irfp260npbf.pdf?fileId=5546d462533600a401535628a2ef1fe4>, accessed on February in 2021

[11] <https://github.com/kaktus85/MightyWattR3/blob/master/Schematic-and-BOM/MightyWatt%20R3%20Schematic.pdf>, accessed on April in 2021

**8. Appendix**

Here images of our built and used components and materials before the our codes for software and firmware.

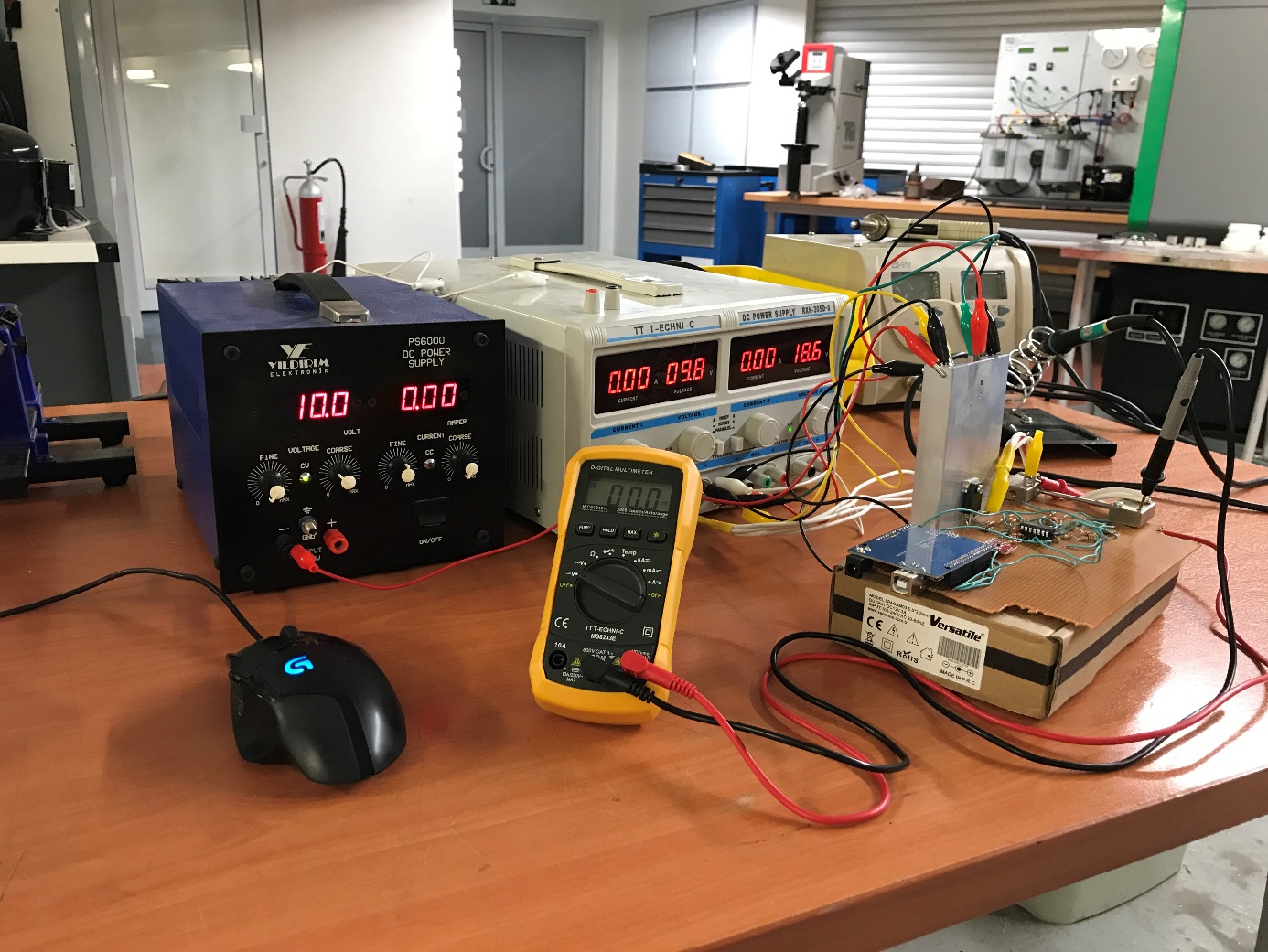
1 x LT1014IN

1 x Arduino Uno

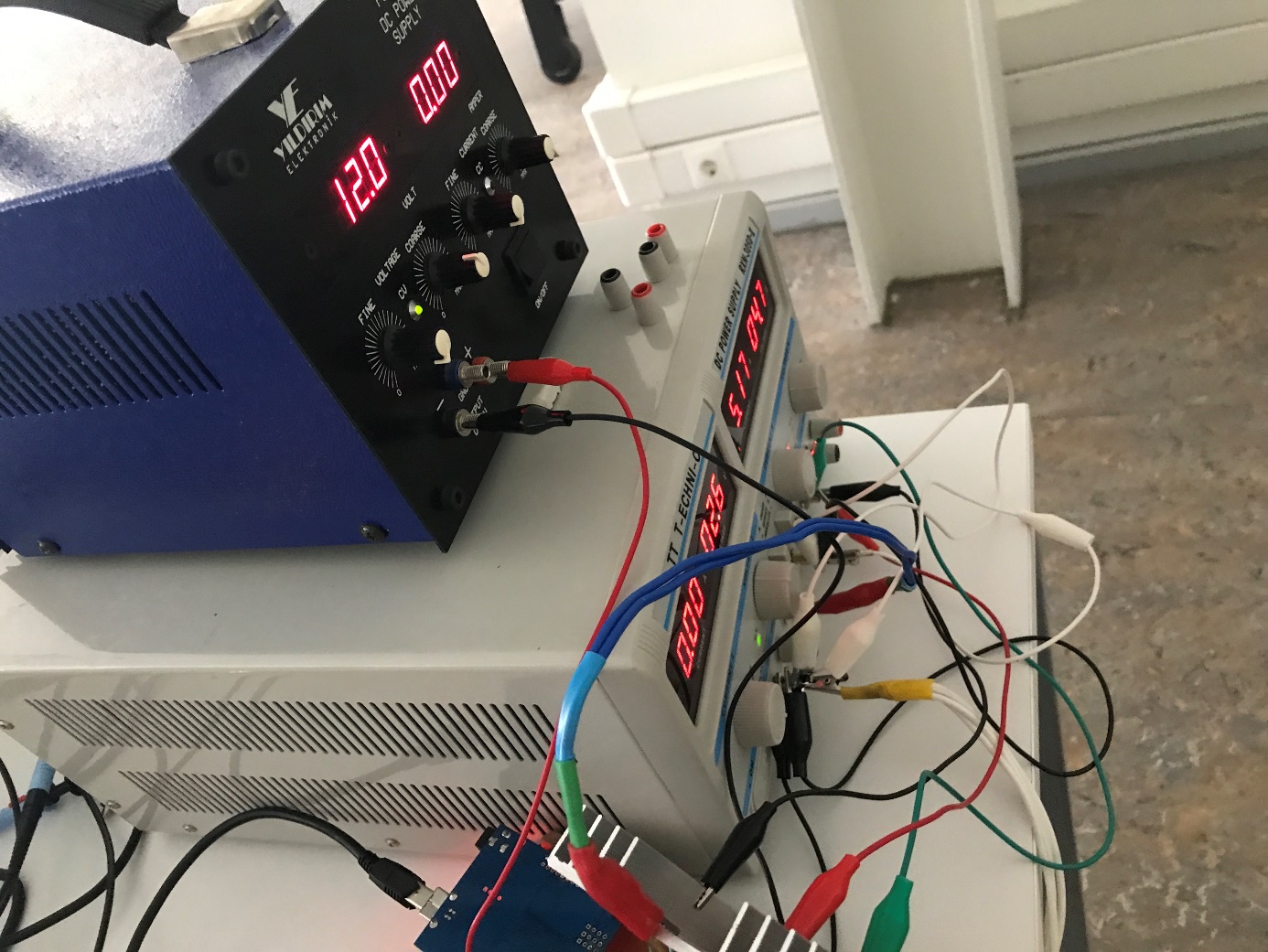
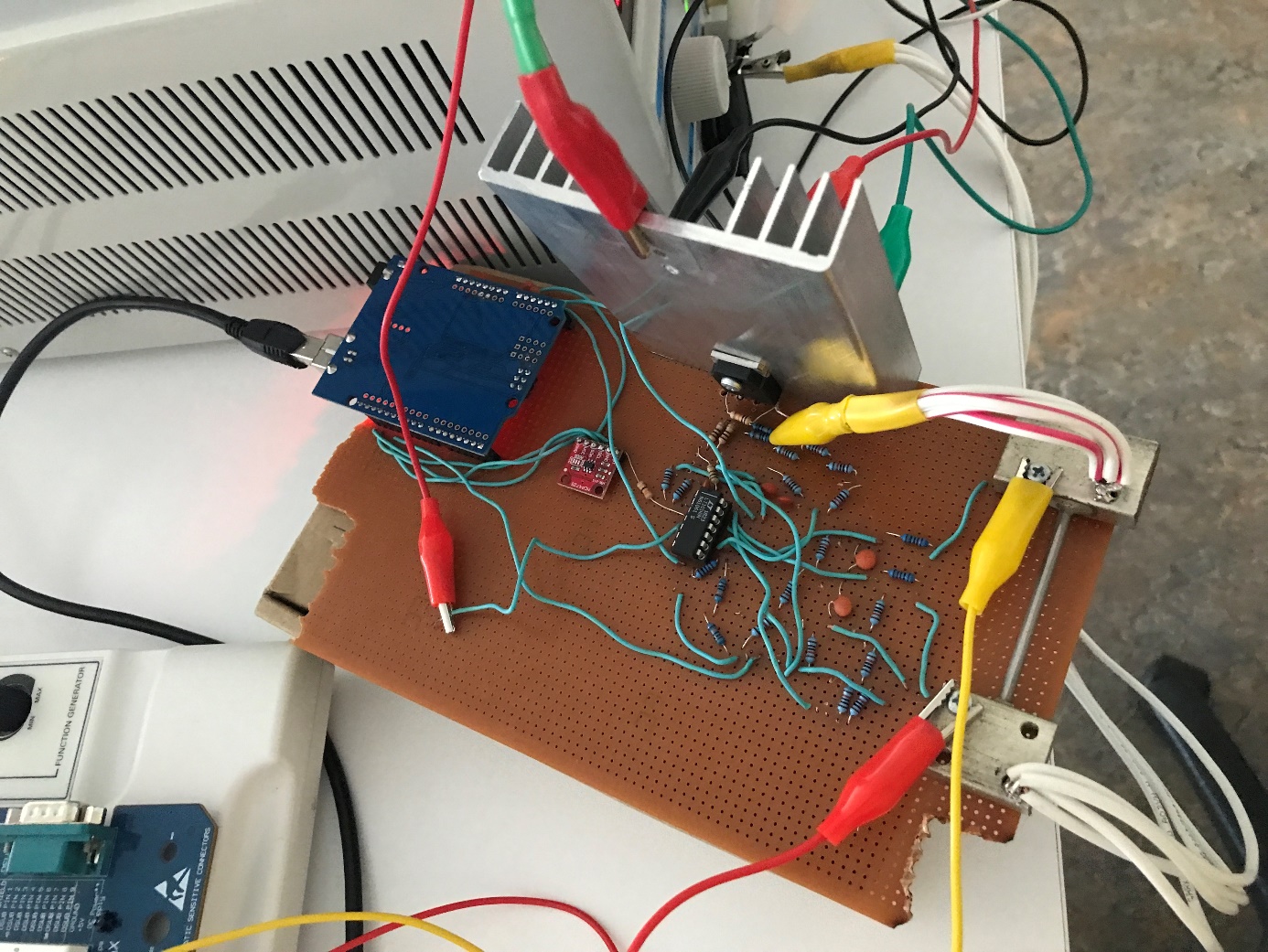
1 x MCP4725 Module

1 x IXTQ50N25P

Shunt resistor can’t be listed because it is custom made.







**8.1. Windows Form Application Software**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO.Ports;

using System.Windows.Forms.DataVisualization.Charting;

namespace FENG497

{

public partial class Form1 : Form

{

float testTimer;

string[] ports = SerialPort.GetPortNames();

double time;

string constantCurrentString;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

textBox1.Text = "0,00";

constantCurrentString = textBox1.Text;

time = 0f;

foreach (string port in ports)

{

comboBox4.Items.Add(port);

comboBox4.SelectedIndex = 0;

}

comboBox3.Items.Add("9600");

comboBox3.Items.Add("14400");

comboBox3.Items.Add("19200");

comboBox3.Items.Add("28800");

comboBox3.Items.Add("38400");

comboBox3.Items.Add("57600");

comboBox3.Items.Add("115200");

comboBox3.SelectedIndex = 6;

label3.ForeColor = Color.Red;

label3.Text = "Disconnected";

chart1.ChartAreas[0].AxisX.Title = "Time(s)";

//series color definition

chart1.Series["Ampere"].ChartType = SeriesChartType.FastLine;

chart1.Series["Ampere"].Color = Color.LawnGreen;

chart1.Series["Ampere"].BorderWidth = 3;

chart1.Series["Voltage"].ChartType = SeriesChartType.FastLine;

chart1.Series["Voltage"].Color = Color.Red;

chart1.Series["Voltage"].BorderWidth = 3;

chart1.Series["Power"].ChartType = SeriesChartType.FastLine;

chart1.Series["Power"].Color = Color.Cyan;

chart1.Series["Power"].BorderWidth = 3;

chart1.Series["Series4"].ChartType = SeriesChartType.FastLine;

chart1.Series["Series4"].Color = Color.Transparent;

}

private void timer1\_Tick(object sender, EventArgs e)

{

try

{

//read

string reads = serialPort1.ReadLine();

string[] values = reads.Split('/');

serialPort1.DiscardInBuffer();

//calculate

double ampere = float.Parse(values[0]);

double voltage = float.Parse(values[1]);

ampere = ampere \* 20f / 1023f;

if (ampere > 1.36)

{

double eqInC = (ampere \* ampere \* 0.434733819) - (3.930096977 \* ampere)

+ 12.91697891;

ampere = ampere \* (eqInC + 100.0) / 100.0;

}

else if (ampere > 0.01 && ampere < 0.02)

{

ampere = ampere \* 5;

}

else if (ampere > 0.02 && ampere < 0.07)

{

ampere = ampere \* 3;

}

else if (ampere > 0.07 && ampere < 0.17)

{

ampere = ampere \* 1.7;

}

else if (ampere > 0.17 && ampere < 0.216)

{

ampere = ampere \* 1.43;

}

else if (ampere > 0.216 && ampere < 0.32)

{

ampere = ampere \* 1.35;

}

else if (ampere > 0.32 && ampere < 0.87)

{

ampere = ampere \* 1.19784005;

}

else if (ampere > 0.87 && ampere < 1.037)

{

ampere = ampere \* 1.0920181452;

}

else if (ampere > 1.037 && ampere < 1.36)

{

ampere = ampere \* 1.0743934586;

}

aGauge1.Value = Convert.ToInt32(ampere);

voltage = voltage \* 50f / 1023f \* 1.031;

double power = voltage \* ampere;

//write to form app

label14.Text = ampere.ToString();

label2.Text = voltage.ToString();

label1.Text = power.ToString();

//time of plot

time += 0.030f;

double ampered = 0;

double voltaged = 0;

double powered = 0;

//plot

if (plotA.Checked == true)

{

chart1.Series["Ampere"].Points.AddXY(time, ampere);

ampered = ampere;

}

if (plotV.Checked == true)

{

chart1.Series["Voltage"].Points.AddXY(time, voltage);

voltaged = voltage;

}

if (plotP.Checked == true)

{

chart1.Series["Power"].Points.AddXY(time, power);

powered = power;

}

double maximus = Math.Max(ampered, voltaged);

maximus = Math.Max(maximus, powered);

chart1.Series["Series4"].Points.AddXY(time, maximus + 0.1f);

if (freeTime.Checked == false)

{//stopping test with timer

if (time >= testTimer)

{

timer2.Stop();

timer1.Stop();

time = 0f;

timeBox.Text = "";

serialPort1.WriteLine("0");

}

}

}

catch (Exception ex1)

{

timer2.Stop();

timer1.Stop();

try

{

serialPort1.WriteLine("0");

}

catch (Exception ex2)

{

MessageBox.Show(ex2.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

MessageBox.Show(ex1.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

}

private void timer2\_Tick(object sender, EventArgs e)

{

//constant current timer

try

{

float ampereFloat = Convert.ToSingle(constantCurrentString);

ampereFloat = ampereFloat / 20f \* 4096f / 103.61f \* 100f;

int intToSendDAC = Convert.ToUInt16(ampereFloat);

string strToSendDAC = Convert.ToString(intToSendDAC);

serialPort1.WriteLine(strToSendDAC);

}

catch (Exception ex1)

{

timer1.Stop();

timer2.Stop();

try

{

serialPort1.WriteLine("0");

}

catch (Exception ex2)

{

MessageBox.Show(ex2.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

MessageBox.Show(ex1.Message + "\n\nSerial port may be disconnected! Please, " +

"\ndisconnect and connect the port again." +

"\n\nElse, the problem is sampling speed;" +

"\ntimer interval of application can be too fast" +

"\naccording to serial port baud rate." +

"\nOr the setted ampere value is greater than 20 Ampere.");

}

}

private void button6\_Click\_1(object sender, EventArgs e)

{//constant current set button

constantCurrentString = textBox1.Text;

}

private void button8\_Click(object sender, EventArgs e)

{

//reflesh ports button

string[] ports = SerialPort.GetPortNames();

foreach (string port in ports)

{

comboBox4.Items.Clear();

comboBox4.Items.Add(port);

comboBox4.SelectedIndex = 0;

}

}

private void button10\_Click(object sender, EventArgs e)

{//serial port connect button

if (serialPort1.IsOpen == false)

{

if (comboBox4.Text == "")

{

return;

}

else if (comboBox3.Text == "")

{

return;

}

serialPort1.PortName = comboBox4.Text;

serialPort1.BaudRate = Convert.ToInt32(comboBox3.Text);

try

{

serialPort1.Open();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

if (serialPort1.IsOpen == true)

{

label3.ForeColor = Color.Green;

label3.Text = "Connected";

}

}

}

private void button9\_Click(object sender, EventArgs e)

{

//serial port disconnect button

timer1.Stop();

timer2.Stop();

serialPort1.Close();

if (serialPort1.IsOpen == false)

{

label3.ForeColor = Color.Red;

label3.Text = "Disconnected";

}

}

private void button7\_Click(object sender, EventArgs e)

{

//test start button

foreach (var series in chart1.Series)

{

series.Points.Clear();

}

if (plotA.Checked == false && plotV.Checked==false && plotP.Checked==false)

{

chart1.ChartAreas[0].AxisY.Title = "";

}

else if(plotA.Checked == false && plotV.Checked == false && plotP.Checked == true)

{

chart1.ChartAreas[0].AxisY.Title = "Power(W)";

}

else if (plotA.Checked == false && plotV.Checked == true && plotP.Checked == false)

{

chart1.ChartAreas[0].AxisY.Title = "Voltage(V)";

}

else if (plotA.Checked == false && plotV.Checked == true && plotP.Checked == true)

{

chart1.ChartAreas[0].AxisY.Title = "Voltage(V) / Power(W)";

}

else if (plotA.Checked == true && plotV.Checked == false && plotP.Checked == false)

{

chart1.ChartAreas[0].AxisY.Title = "Ampere(A)";

}

else if (plotA.Checked == true && plotV.Checked == false && plotP.Checked == true)

{

chart1.ChartAreas[0].AxisY.Title = "Ampere(A) / Power(W)";

}

else if (plotA.Checked == true && plotV.Checked == true && plotP.Checked == false)

{

chart1.ChartAreas[0].AxisY.Title = "Ampere(A) / Voltage(V)";

}

else

{

chart1.ChartAreas[0].AxisY.Title = "Ampere(A) / Voltage(V) / Power(W)";

}

if (freeTime.Checked == false && timeBox.Text != "")

{

testTimer = float.Parse(timeBox.Text);

}

else

{

freeTime.Checked = true;

}

timer1.Start();

timer2.Start();

}

private void button4\_Click(object sender, EventArgs e)

{

//stop test button

timer1.Stop();

timer2.Stop();

time = 0f;

timeBox.Text = "";

byte[] zero = { 0x00 };

try

{

serialPort1.Write(zero, 0, 1);

}

catch (Exception)

{

}

}

private void button6\_Click(object sender, EventArgs e)

{

//save image button

string ffileName = imageName.Text;

if (!ffileName.StartsWith(@"\"))

{

MessageBox.Show("Please write, file name as\nlike: "+

@"'\name.formatType'");

return;

}

try

{

string fileName = imageName.Text;

folderBrowserDialog1.ShowDialog();

chart1.SaveImage(folderBrowserDialog1.SelectedPath + fileName,

ChartImageFormat.Png);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message+"\nPlease write, file name as\nlike: "

+ @"'\name.formatType'");

}

}

}

}

**8.2. Arduino Firmware**

#include<Wire.h>

#include<Adafruit\_MCP4725.h>

Adafruit\_MCP4725 dac;

#define read1 A1

#define read2 A2

String DACbitSTR;

int loopCnt;

int loopCnt2;

float average1;

float average2;

void setup() {

Serial.begin(115200);

dac.begin(0x60);

pinMode(read1, INPUT);

pinMode(read2, INPUT);

dac.setVoltage(0, false);

loopCnt = 0;

loopCnt2 = 0;

average1 = 0;

average2 = 0;

}

void loop() {

float value1 = analogRead(read1);

float value2 = analogRead(read2);

average1 += value1;

average2 += value2;

if (loopCnt >= 425) {

average1 = average1 / loopCnt;

int average11 = average1;

Serial.print(average11);

Serial.print("/");

average2 = average2 / loopCnt;

int average22 = average2;

Serial.println(average22);

average1 = 0;

average2 = 0;

loopCnt = 0;

}

while (Serial.available() > 0) {

char chr = Serial.read();

if (chr == '\n') {

int dacInt = DACbitSTR.toInt();

DACbitSTR = "";

dac.setVoltage(dacInt, false);

loopCnt2 = 0;

}

else {

DACbitSTR += chr;

}

}

if (loopCnt2 >= 6001) {

dac.setVoltage(0, false);

loopCnt2 = 0;

}

loopCnt++;

loopCnt2++;

}