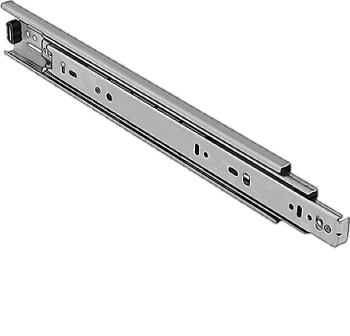
**Electrical Design of Beam Degrader:**

**OVERALL DESIGN:**

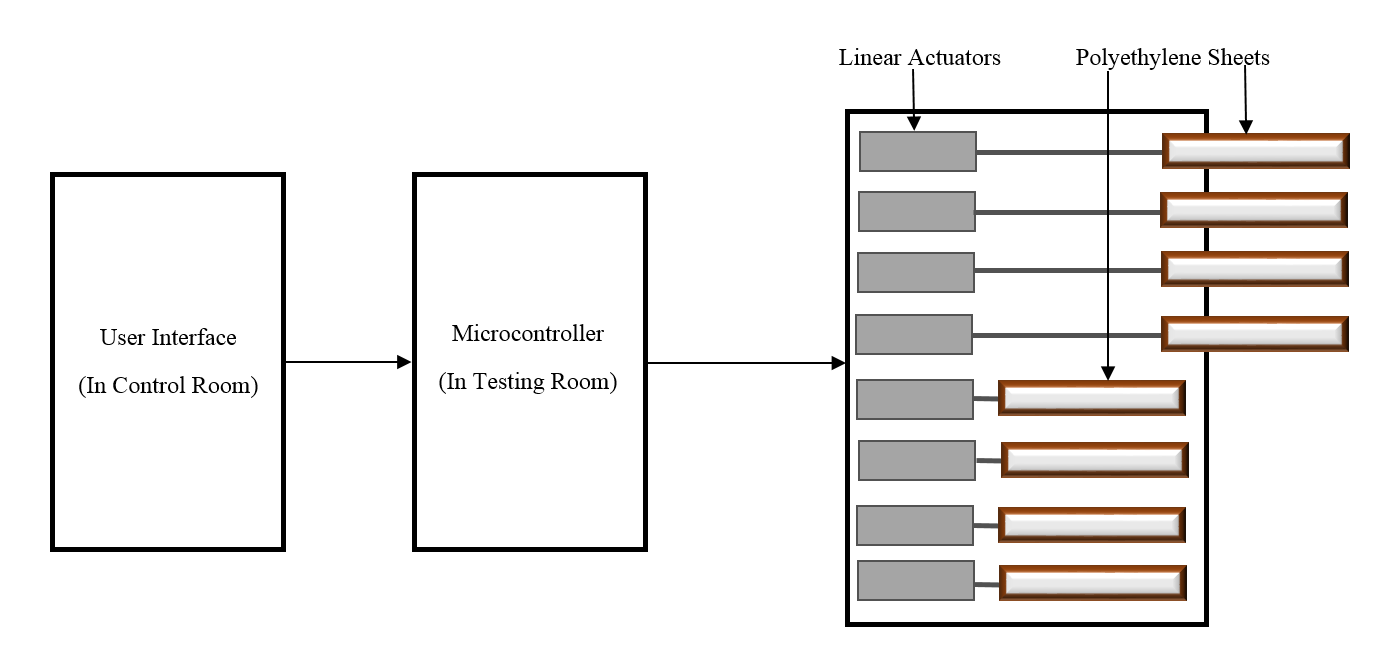
The overall design of this Beam Degrader project is to extend and retract EIGHT actuators upon command.

There are 8 individual actuators each with a different thickness of Polyethylene (in a binary number system) attached to it, starting with a 0.10mm thickness, 0.20mm, 0.40mm, 0.80mm, 1.60mm, 3.20mm, 6.40mm, and 12.80mm. Having these 8 thicknesses of polyethylene allows us to move the 8 actuators in different orders allowing us to create a thickness ranging anywhere from 0.00mm all the way to 25.50mm in increments of 0.10mm.

In order to move these Polyethylene sheets we use a **Linear Slide** and a **Linear Actuator**, shown below:



Right now, our Mechanical Engineer interns, have ordered two slides and two actuators and created a prototype of the design. I constructed the electrical design and we are able to move them upon command.

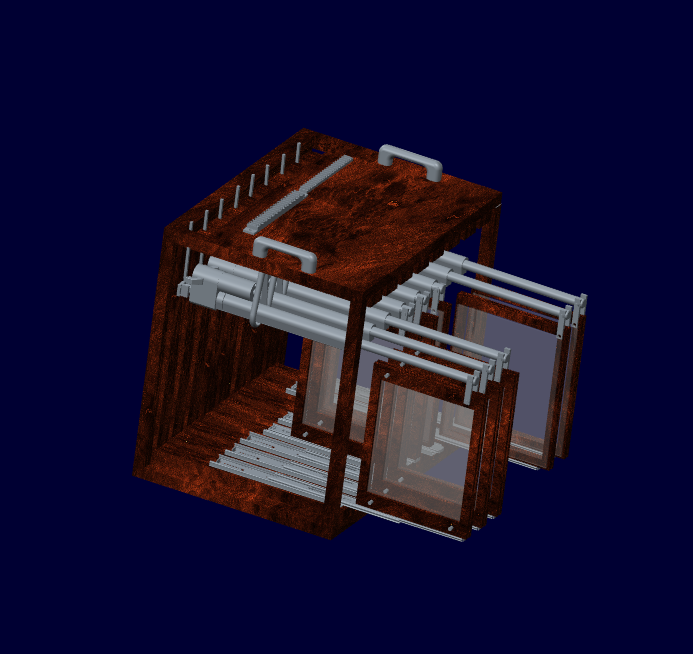


So basically for the Overall design, we want the User to input in a desired thickness into the Graphical user Interface (GUI) (say 0.30mm) and then the thickness number will be sent to the microcontroller (Arduino DUE) and the Arduino will output to the PCB and then to the Beam Degrader mechanism which sends out the appropriate polyethylene sheets (in this case 0.10mm + 0.20mm) and retracts any other polyethylene sheets that might need to be pulled back to reach the thickness of 0.30mm.

**MECHANICAL DESIGN:**

So ERIC and ISAAC probably talked about the Mechanical Design in their documentation, so you should probably look and read over their stuff to get a better understanding of the overall mechanical design. What I can say is that it will basically look like this:

(Right now we aren’t sure yet if we want to split it up in half into 4 Actuators and another 4 Actuators). Instead of the 8 Actuators all together in one box.



\*\*Look to ERIC and ISAAC’s documentation for a better explanation of the mechanical design!\*\*

**ACTUATORS:**

So a little about the Linear Actuators we are using, Eric found them online and probably wrote where to buy them, but I can tell you about the electrical design of the Actuators.

Each actuator is capable of moving 25 pounds and extending 10 inches. We won’t be providing a load of 25 pounds, these actuators can handle a lot more than we need but cost the same as smaller ones so we ended up getting these.

So the minimum current each actuator draws is 0.8Amps, but we estimated that each actuator with the Polyethylene load attached will draw a max of around 1Amp of current (which is actually lower than what is actually drawn). You should bump that up to 1.5Amps of current for future designs

Each actuator can run at voltage ranging from 6V to 12V, but will run much slower if we apply 6V compared to 12V, so we went with the 12v because it would be more efficient since time is a concern! So basically each actuator should be supplied with a Power Supply of 12v and at least 1Amp.

**POWER SUPPLY:**

We are currently using the INSTEK Power supply where we manually set the desired volts and current. We have this set at **12v** and **2Amps**.

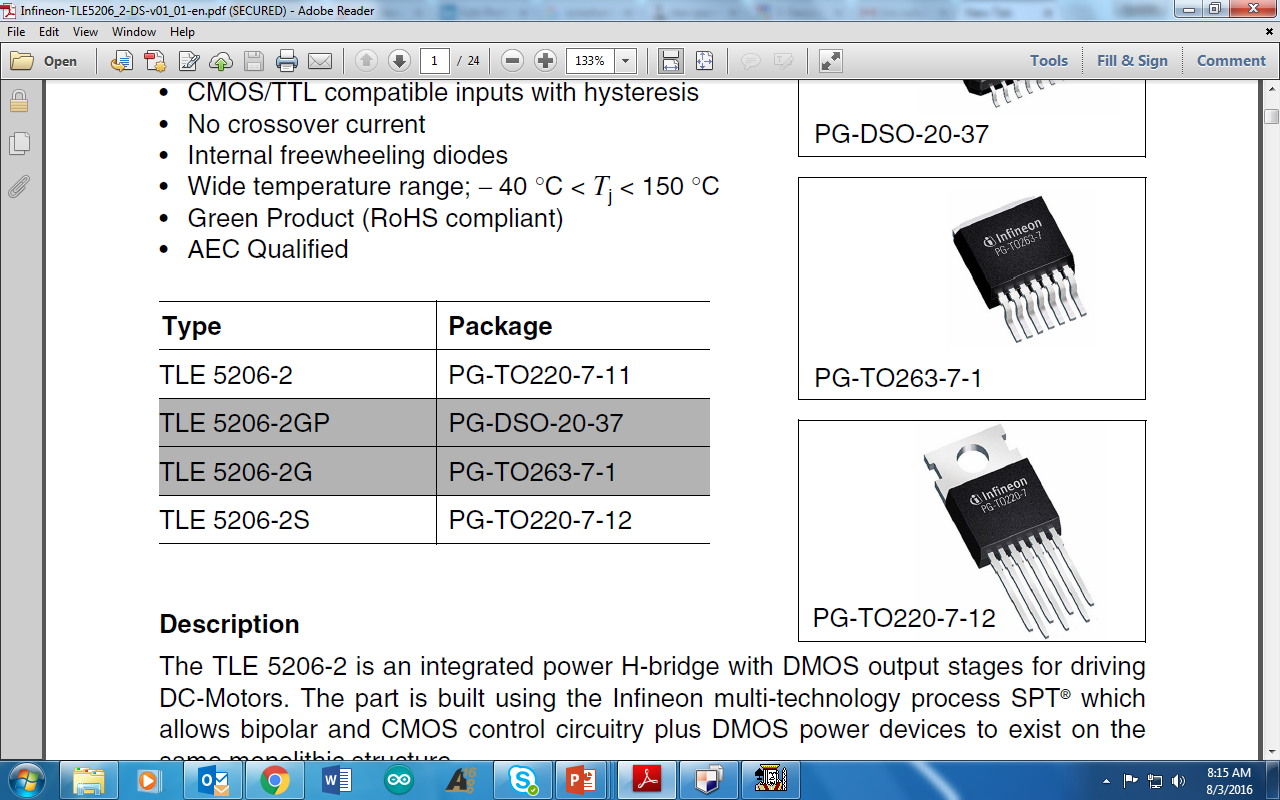
The overall Power Supply being supplied to the Actuator as of now is set up to be 12volts and 2Amps of Current. This of course limits the allowed amount of actuators being able to be powered at a time.

What I mean by that is, say we want a thickness of 25.5mm where all the actuators need to be sent out. We are unable to send all 8 actuators out at once because that would be drawing 1Amp per actuator, equaling to a total of 8Amps drawn at a 12V Power Supply only supplying 2 Amps of current. Therefore, we MUST limit the overall design to drive only 2 actuators at a time, allowing only 2 Amps to be drawn total. This is the MAIN limit and concern for the design.

We are limited to only 2Amps at 12v because the PCB that we designed can only handle 2Amps of current. The newer updated version of the Alitum Design PCB layout is updated so that it can handle 4Amps of current allowing 4 actuators to be driven at a time. But the PCB’s that we currently printed can only withstand 2Amps of current! Don’t try to drive more than 2 actuators at once please!

**H-bridge**:

Alright, now you must be wondering how in the world do you send out and pull in the actuators upon command!? Well, the trick is using the IC known as the “H-bridge”. This little component is shown below:

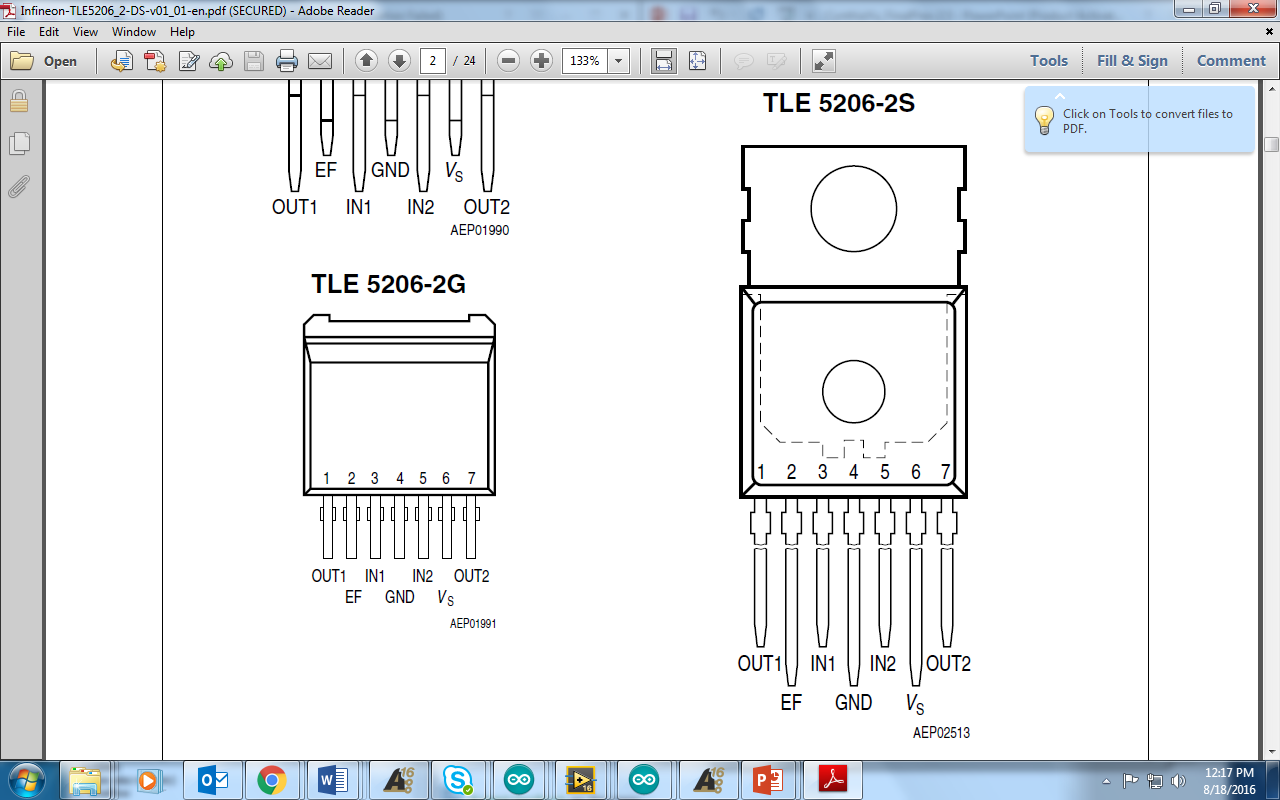


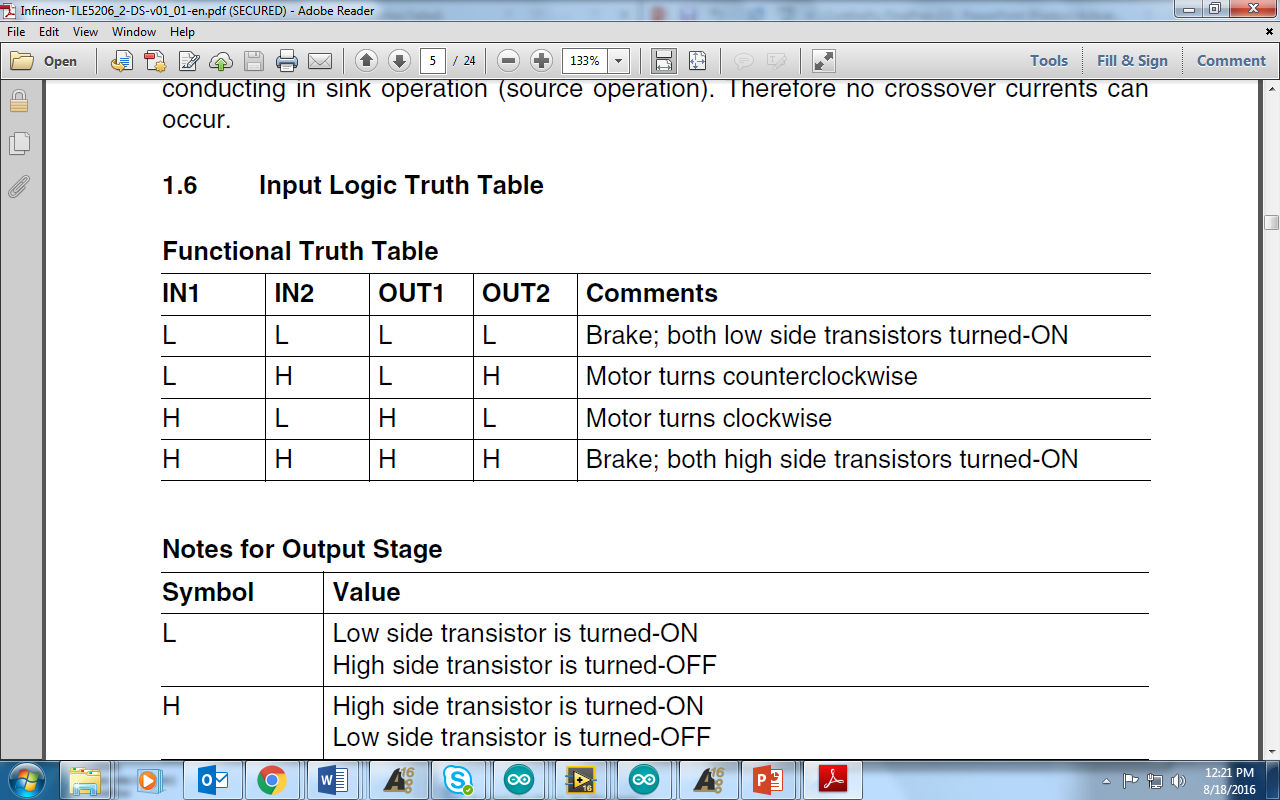
This little H-bridge can be ordered at **Digikey**: <http://www.digikey.com/product-search/en/integrated-circuits-ics/pmic-motor-drivers-controllers/2556626?k=mosfet&k=&pkeyword=mosfet&pv69=80&FV=fff40027%2Cfff802d2%2C2040011&mnonly=0&newproducts=0&ColumnSort=0&page=1&quantity=0&ptm=0&fid=0&pageSize=25>

\*\*The component is [TLE5206-2S-ND](http://www.digikey.com/product-detail/en/infineon-technologies/TLE5206-2S/TLE5206-2S-ND/1283081)\*\*

We already have 10 more H-bridges on the way, so you shouldn’t have to order any but just in case you need more information on it. Look at the link above. I also attached the data sheet in the folder, so look to that for a better understanding of the component.

So back to controlling the actuators. Basically using this component we are able to change the polarities of the outputs pins which extends and retracts the Motor within the Actuator.

So this little H-bridge component has 7 pins, number from 1-7 (left to right). What I said before about changing the polarities happens on pins 1 and 7(OUT1 and OUT2). Basically to have the output be 12v powered to the motor (extend the actuator), we must apply a digital logic into pins 3 and 5(IN1 and IN2). In order for the output to be -12v (retract the actuator) we apply another digital logic to pins 3 and 5. If we want the actuator to stay put and not move, we output 0v by applying another digital logic to pins 3 and 5. This logic applied is coming from the Arduino DUE and works accordingly to the chart shown below.



**L = LOW = 0**

**H = HIGH = 1**

\*When the input Logic is IN1 = LOW, IN2 = LOW (0 0), the output coming out from pins 1 and 7(OUT1 and OUT2) is LOW, LOW (0 0) therefore **holding the actuator in place**.

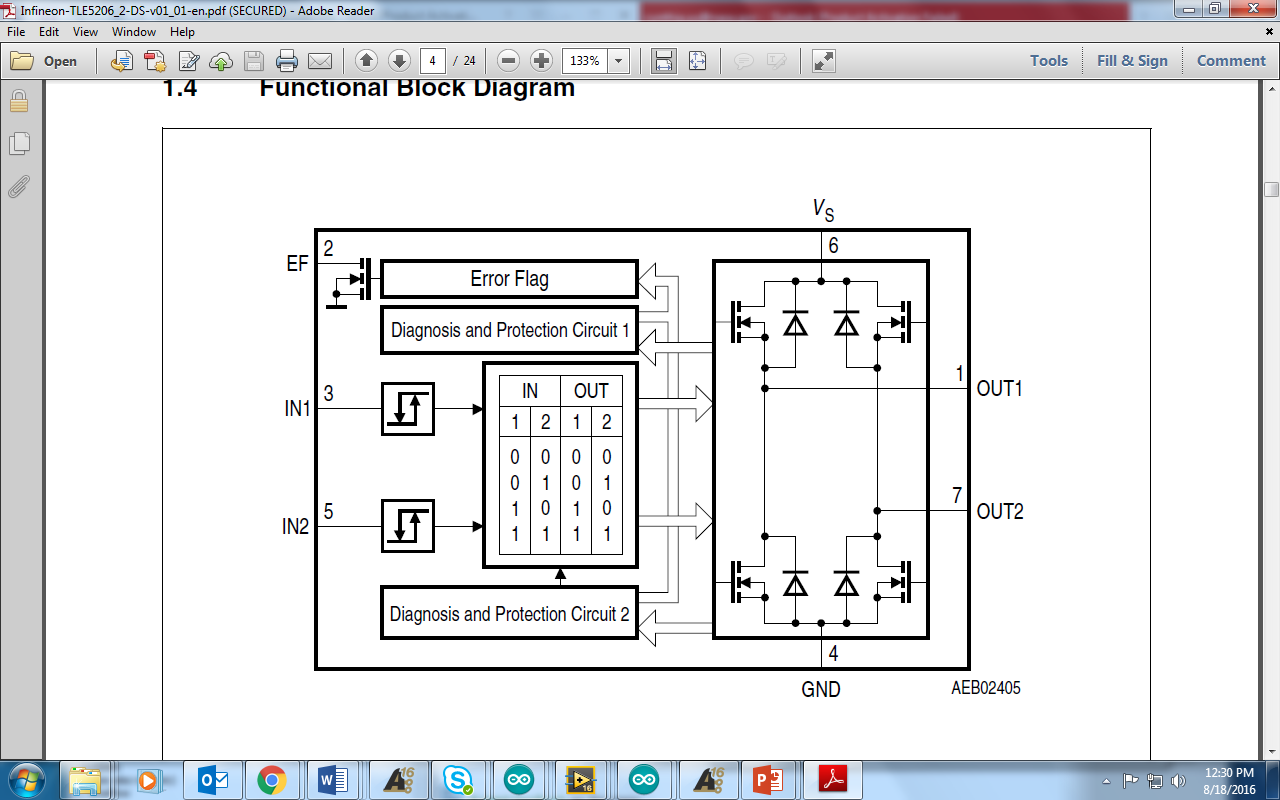
\*When the input logic is IN2 = LOW, IN2 = HIGH (0 1), the output coming from pins 1 and 7 is LOW, HIGH (0 1) turning the motor **counterclockwise** and **extending the actuator**.

\*When the input logic is IN2 = HIGH, IN2 = LOW (1 0), the output coming from pins 1 and 7 is HIGH, LOW (1 0) turning the motor **clockwise** and **retracting the actuator**.

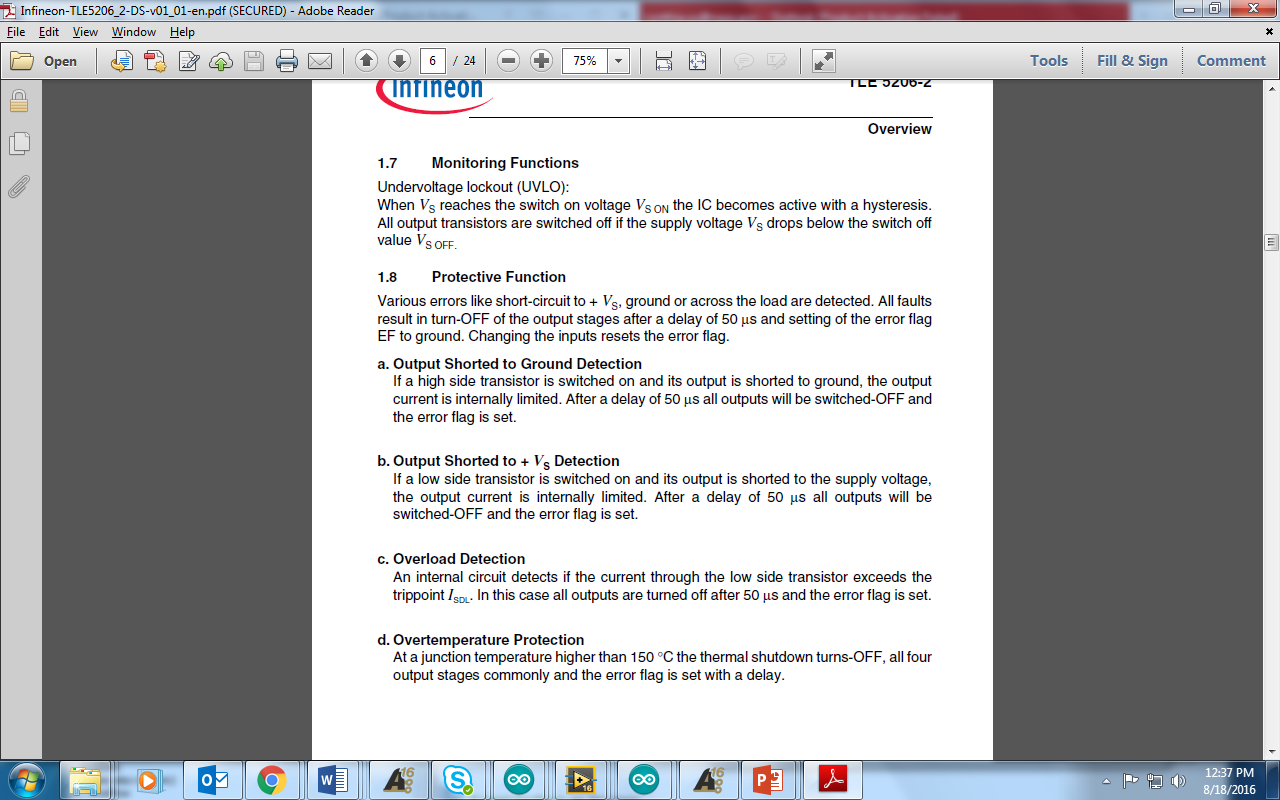
\*When the input logic is IN2 = HIGH, IN2 = HIGH (1 1), the output coming from pins 1 and 7 is HIGH, HIGH (1 1) therefore **holding the actuator in place**. (Does the same thing as logic 00.

So hopefully you understand how that works, but if you want a better understanding look to this block diagram. Basically the Arduino sends a digital logic to the H-bridge, once the H-bridge receives that logic, it will OUTPUT the polarity a certain way and that will determine whether the actuator is extended or retracted because the polarities are changing from one way to the other!

\*Oh and by the way, all of these little pictures I’m inserting in are from the Data sheet of the H-bridge. I attached that in the file and it is titled: “**Infineon-TLE5206**”, So please look to that for more info.



I told you what pins 1,3,5,7 do. Now I’ll explain the rest. **Pin 2** is dedicated to an output from the H-bridge which when we connect to the Arduino we are able to read if there is an Error detected from the Actuator. I connected a Red LED on the PCB to light up whenever there is an Error detected. So how it works is that it is always set at a logic HIGH when there is NO ERROR. I repeat it’s a logic HIGH when NO ERROR! It will be pulled down to a LOW when there is an ERROR. When there is an error detected the H-bridge automatically turns off the outputs from the pins so that they will not work correctly anymore. So whenever an error is detected it is not going to work correctly. So if there is an error detected it can be any of these reasons shown below:



I had the error flag go off on me in the beginning and it took a great deal of time to figure out what it exactly was, but it was because I wasn’t adding in the bypass capacitors, so right now if you connect everything to how I have it configured right now then there shouldn’t be any errors detected unless you accidently short a wire or something is connected wrong.

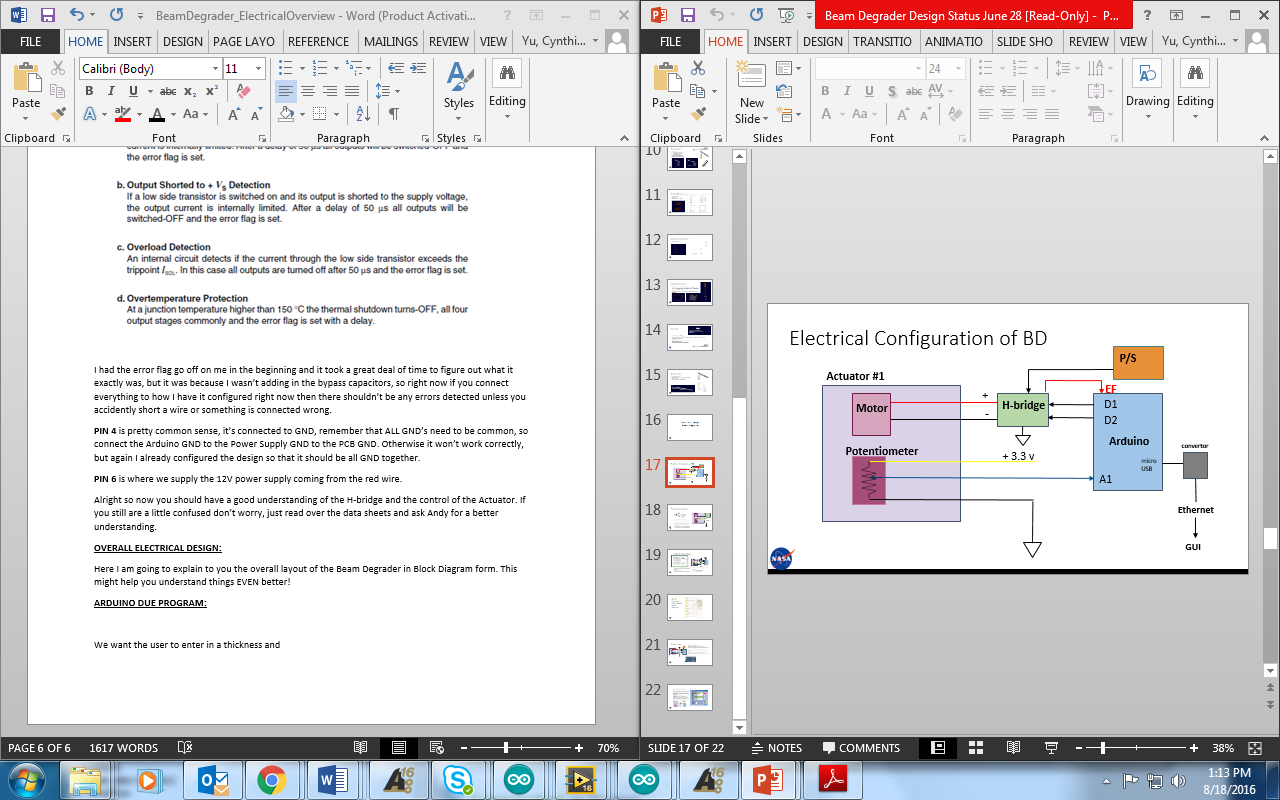
**PIN 4** is pretty common sense, it’s connected to GND, remember that ALL GND’s need to be common, so connect the Arduino GND to the Power Supply GND to the PCB GND. Otherwise it won’t work correctly, but again I already configured the design so that it should be all GND together.

**PIN 6** is where we supply the 12V power supply coming from the red wire.

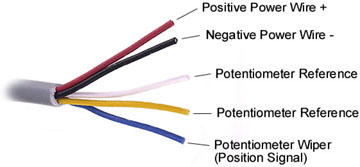
Alright so now you should have a good understanding of the H-bridge and the control of the Actuator. If you still are a little confused don’t worry, just read over the data sheets and ask Andy for a better understanding.

**OVERALL ELECTRICAL DESIGN:**

Here I am going to explain to you the overall layout of the Beam Degrader in Block Diagram form. This might help you understand things EVEN better! (well I hope so)



So the Big purple box on the left is the Actuator. One actuator has 5 Wires coming out of it that look like this:



The Black and Red Wire are connected to the internal Motor of the Actuator and is what drives the actuator in and out. The white, yellow, and blue wire are for reading the position of the actuator. The Blue wire aka “Wiper” is the indicator that reads a different resistance between the 3.3volts being supplied to the Potentiometer. The 3.3v is supplied from the Arduino 3.3v output and is connected to the yellow wire and the white is connected to ground. This allows us to see where the position of the actuator is, because the “wiper” moves along the potentiometer displaying a different analog value from ~30 all the way ~930. Where 30 is fully retracted and 930 is fully extended.

So the 2 motor actuator wires (BLACK and RED) are connected to the H-bridge as OUT1 and OUT2 because the H-bridge is outputting the polarities to the motor based on the input logic supplied from the Arduino.

The Arduino DUE was the microcontroller we chose to use because we found like 10 of them downstairs and didn’t have to pay to buy the UNO so lucky us. If you’re not familiar with the DUE, it’s pretty easy to use, it’s almost exactly like the UNO, but has like 5 times more I/O pins and a lot more Analog pins as well which is pretty awesome for us. So the Arduino is going to output a logic either (00, 01, 10) to the H-bridge from 2 1/O pins and that gets fed into the H-bridge and then based off of that logic it will adjust the polarity of the motor to either extend or retract. Hope you understood that! If so let’s move along. If not then re-read it.

That red little arrow coming out of the H-bridge is the Error Flag (EF), this is an Input to the Arduino because we need to be able to read if there is an error coming from the Actuator. If there is an error detected, the Arduino will read it and output this to the LED and GUI in Labview (I’ll talk about Labview later). The rest I’ve already explained above, but this was just in case you didn’t fully understand it all.

**ARDUINO DUE PROGRAM:**

I’m assuming you have a pretty good understanding of using the Arduino, so I won’t go into explicit detail, but I will try my best to explain everything clearly.

So just as I said before we are trying to have the actuators extend and retract upon the users request of an inputted desired thickness.

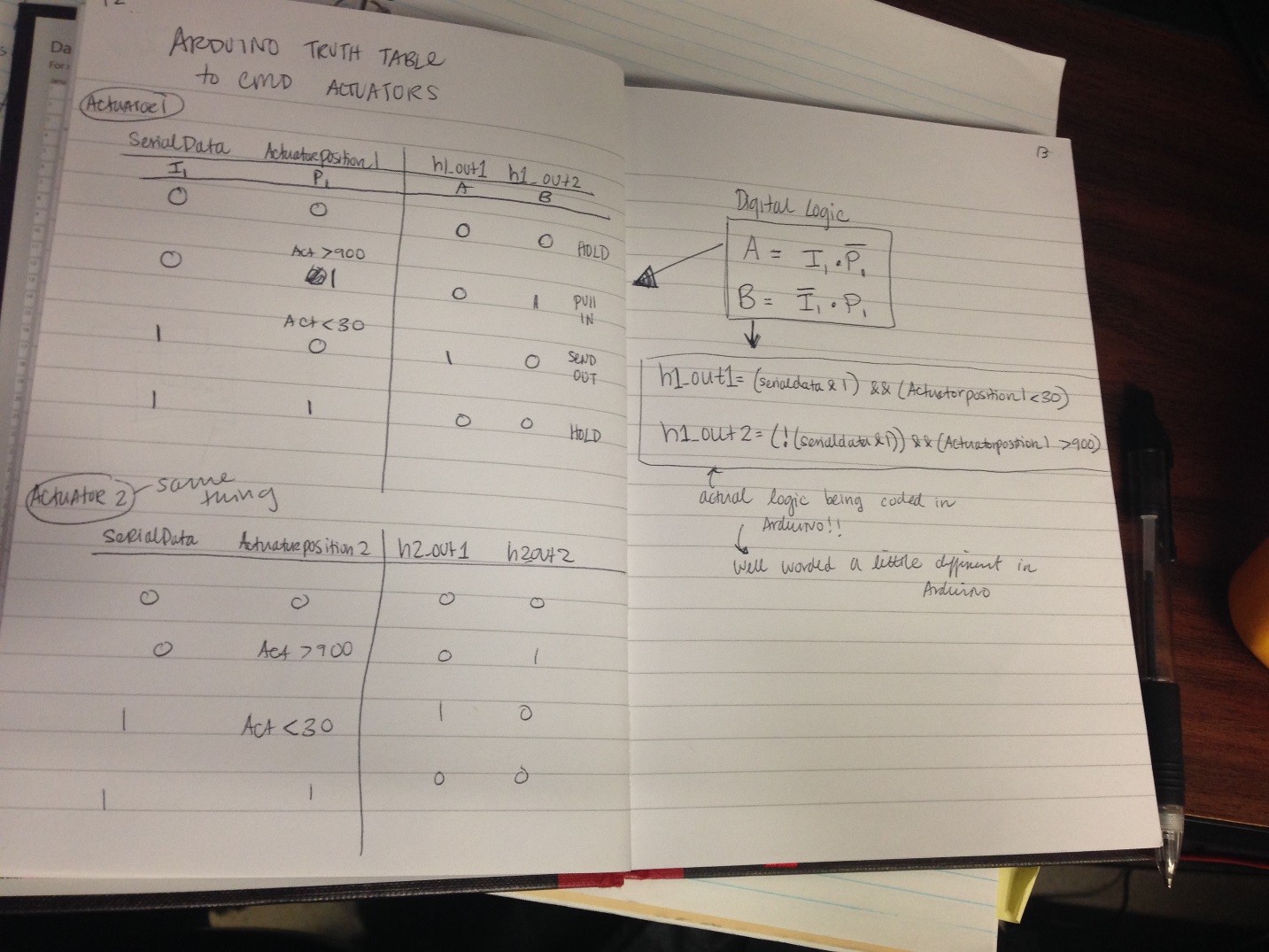
Therefore, Andy and I figured out that the best way to do that is by using a 8 bit binary number format. So basically each bit is matched according to an actuator.

What I mean by that is that we will have a 8 bit binary value(0000 0000) coming from the GUI ranging from 0.1 to 25.5 like mentioned at the very beginning. Since it’s unnecessary to use decimals, I’m going to multiply all the incoming values by 10 so that the numbers will vary 1 to 255 which is perfect for an 8 bit binary value.

Therefore, when an incoming thickness is read from the SerialPort on the Arduino, it will convert that number to an integer and be read as a binary value. Based off of this value coming from the SerialPort we can determine which actuators to be sent out by using something called digital logic.

If you don’t know what that is, then google it and get a good understanding of what “ANDing” to values does in binary.

So our goal is to send 00, 01, or 10 to the H-bridge from the Arduino, so we are going to use digital logic to do that. The best way of doing that is by writing out a truth table. Below I’ve shown a drawing of the truth table and what is happening in the Arduino logic.



So the table created above is showing that each actuator has 4 different possibilities based off of the data we are sending it and the current position of the actuator.

For instance, the first row with 0 0, is saying that if we want the actuator to pulled in(hence the first 0), and the actuator is already retracted(hence the second 0) then we output logic (0 0) because that’s the logic to hold the actuator. Remember?

Well for the second line, if the data received is telling us to pull in the actuator (first= 0), and the actuator is extended(where position is greater than 900), then we want to pull it in, therefore outputting logic 01(the logic to retract the actuator).

For the third line, if the received serialData is telling us to send out the actuator(first = 1), and the actuator is already pulled in(second = 0), then we want to extend out the actuator, therefore outputting logic 10(to send out the actuator)

For the last line, it is saying that when the serialData received is wanting to extend the actuator, and the position of the actuator is already extended(the 1), then we send out logic (0 0) to hold the position since it is already where we want it.

This table can be applied to each Actuator because the logic will be the same. From this truth table we are able to determine a digital logic equation shown on the right side of the page.

Where when H1\_out1 = 1, we know that serialdata = 1, and Actuatorposition1 = 0

Where when H1\_out2 =1, we know that serialdata = 0, and Actuatorposition1 = 1

From this we can write out the following equations:

**H1\_out1 = (serialdata &1) && (ActuatorPosition1 > 900)**

**H1\_out2 = (!(serialdata&1) &&(ActuatorPosition < 30)**

This equation above accounts for all the 4 possibilities shown in the truth table. Which is pretty convenient and cool if you ask me!

So after changing that into Arduino code terms we get the following code below.

**ARDUINO CODE: (part of Actuator 1 command)**

if ((serialData & 1) && (ActuatorPosition1 < 30))

{

digitalWrite(h1\_out1, HIGH);

// for testing purposes: Serial.println("1a");

}

else

{

digitalWrite(h1\_out1, LOW);

//for testing purposes: Serial.println("0a");

}

if (!(serialData & 1) && (ActuatorPosition1 > 900))

{

digitalWrite(h1\_out2, HIGH);

// Serial.println("1b");

}

else

{

digitalWrite(h1\_out2, LOW);

// Serial.println("0b");

}

Also if you are wondering why we “ANDed” the serialData with a 1, and then for actuator2 we “ANDed” it with a 2, and then for actuator3 we “ANDed” it with a 4, and so on…

It is because of the binary count we are using to control the actuators. Basically whenever we AND the incoming serialData with the different numbers we are testing to see if there is a command to send out that specific actuator or not. So by ANDing it with a 1, we are testing to see if that thickness needs an output of actuator 1. If it doesn’t, it will check to see if the position is extended or retracted. If it is retracted, t hen it won’t do anything if it doesn’t need to be extended. If it is extended and we don’t want actuator 1 to be out, then we will send a command to pull it back in. Sorry if that is hard to understand let me give you an example.

\*\*\*Example: Say the user wants a thickness of 0.1mm. That means we want actuator1 with a 0.1mm polyethylene sheet to be extended out and everything else to be pulled back in.

So what the Arduino will do is first read the position of the actuator coming in from the Potentiometer wiper. If it reads an analog value of less than 30, then we know the Actuator is retracted. Therefore, the Arduino will send out a command to extend the actuator (LOGIC = 01). But before it sends that command, it will check all the rest of the analog positions of the other 7 actuators to see if they are extended or retracted. Say that actuator 5 is extended, then the Arduino will read that the position is greater than 900 and know to send out the command to retract that actuator (LOGIC= 10). Once it does that the only actuator extended will be actuator 1 which is holding the 0.1mm polyethylene thickness.

\*You might be thinking right now, “Is there another way to write this code?” and YES there are several ways to write this code, but of the many ways I tried, this was the most efficient method(the method shown ABOVE). It is almost twice as short as other codes and you can use logic to write it so Why not! But just because I tried so many methods. I attached a sample code in Arduino below showing an alternative method to output the logic to the Arduino.

Here’s a little snippet of it below. It works exactly the same as the other one but WAY longer and unnecessary if you ask me.

I attached the other versions on the disk as well, they are named “TEST\_oneacc\_v1” and “TEST\_oneacc\_v2”.

if (serialData & 1)

{

if (ActuatorPosition1 < 30)

{

digitalWrite(h1\_out1, HIGH);

digitalWrite(h1\_out2, LOW);

Serial.print("10");

Serial.print("Actuator retracted EXTEND");

}

if (ActuatorPosition1 > 910)

{

digitalWrite(h1\_out1, LOW);

digitalWrite(h1\_out2, LOW);

Serial.print("00");

Serial.print("Actuator extended STAY");

}

}

else

{

if (ActuatorPosition1 < 30)

{

digitalWrite(h1\_out1, LOW);

digitalWrite(h1\_out2, LOW);

Serial.print("00");

Serial.print("Actuator retracted STAY");

}

if (ActuatorPosition1 > 910)

{

digitalWrite(h1\_out1, LOW);

digitalWrite(h1\_out2, HIGH);

Serial.print("01");

Serial.print("Actuator extended RETRACT");

}

}

\*\*Basically for each Actuator you need one H-bridge and each H-bridge needs 3 dedicated digital I/O pins from the Arduino, one for the ErrorFlag, and two for the INPUT to the H-bridge. You will also need 1 Analog input pin from the Arduino for each actuator for reading the Actuator Position coming from the Potentiometer.

-But I’ve already assigned the pins that you use in the Arduino code. You can take a look in the Arduino code to better understand what I’m talking about.