Spring 2022

Additive Manufacturing Waste Management System - Plastic Extrusion Process

Gabriel Bennett
gjb45@uakron.edu

Lindsay Liebrecht
The University of Akron, ll125@uakron.edu

David Lyogky
The University of Akron, djl123@uakron.edu

Wilson Woods
The University of Akron, ww52@uakron.edu

Additive Manufacturing Waste System-

Plastic Extrusion Process

Gabriel J. Bennet
Lindsay Liebrecht
David Lyogky
Wilson Woods

Department of Computer & Electrical Engineering

Honors Research Project

Submitted to

The Williams Honors College
The University of Akron

Gabriel J. Bennett's Contribution

The additive manufacturing waste system – plastic extrusion process project had a variety of subsystems to design. My responsibilities as the team lead involved both designing some of the subsystems, as well as integrating all of the subsystems together, and coordinating team meetings as well as the overall projects direction. I designed the overall architecture of the system, and constructed the main platform for the project. I integrated the motor with the main screw conveyor and the outer metal shell. I then integrated this shell with the heating elements, temperature probes, and heat shields. I also created the main power board that connected into David's circuit designs. I modeled all of the 3D printed parts and integrated them into the design as well. This included the spooler and winder systems. The motors were then integrated with the electronic boards. Overall, my contribution was envisioning and building the physical system, as well as integrating the sensors and circuits into the main system.

Additive Manufacturing Waste System-

Plastic Extrusion Process

Project Design Report

Design Team 16

Gabriel Bennett

Lindsay Liebrecht

David Lyogky

Wilson Woods

Dr. Shivakumar Sastry

11/28/2021

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Table of Pseudo Code

Abstract

This report analyses and designs an additive manufacturing plastic recycling system to be utilized by 3D printer users to produce filament from the recycled waste. After identifying the key marketing and engineering requirements, a grinding and extrusion system were designed to meet the requirements. This is accomplished by grinding the PLA or PET plastic waste, heating the resulting pellets to the required melting points of the specific plastic, extruding the melted plastic through a specific nozzle size, cooling, and spooling the resulting filament. The resulting spooled filament will be viable for 3D printing applications that use a 1.75mm filament. The designed system is controlled by the user through a touch monitor which in turn contains an embedded control system that monitors speed, filament diameter and extrusion temperatures using sensors, actuators, and motors within the system.

- Adaptable for different types of plastic due to wide temperature range
- Operates on a standard 120V household outlet and 15A breaker
- Built in safety system
- Sized for a desktop office environment
- Self-contained system that doesn't require interference once running
- Adaptable for various motor speed control

(GB,LL,DL,WW)

1. Problem Statement

1.1. Need:

3D printing is a fast-growing market with its main source of waste being PLA and ABS plastics. In 2019, the global additive manufacturing market grew to over $10.4 billion, crossing the pivotal double-digit billion threshold for the first time in its nearly 40-year history. (SmarTech Analysis, 2020 Additive Manufacturing Market Outlook and Summary of Opportunities Report). The waste is generated from failed prints and rejected support structures which are common occurrences for personal use. Plastic recycling has become one of the leading discussions of environmental protection and waste management. The 3D market currently does not offer an effective and affordable solution for handling the waste generated. To help bring awareness to this need, companies such as Print Your Mind 3D has offered challenges to hobbyists and corporations to solve this problem. Due to the continuing growth of the additive manufacturing market, and the constant attention on plastics and how they harm the environment, a solution must be found to reduce or eliminate the waste stream of 3D printing. (GB)

1.2. Objective:

Design a working model to recycle plastic materials (PET and PLA) that produces a viable filament that can be used in 3D printing applications. The system will include a grinder, extruder, and spooler to create a viable filament. A main control board will allow for user input and monitoring of the entire system for the duration of the filament batch run. (LL)

1.3. Background:

“I want to say one word to you. Just one word. Plastics." (Mr. McGuire, The Graduate). Plastics are an inundated part of people’s everyday lives. From gifts, to garbage, to grocery shopping; plastic is used in many of the everyday products people consume. A primary use for plastics is in the packaging of other products. This packaging is almost always, unilaterally, discarded upon opening said product. Plastics are highly durable and easily molded but are either not biodegradable or their biodegrading process takes years if not decades to complete. These thoughts lead to the modern-day debate that surrounds the mass consumption of plastic products in the face of grave environmental concerns. The newest addition to this age-old dilemma is none other than the 3D printer. (GB)

Over the last several years, especially after the recent global pandemic, 3D printing has grown into a massive industry. This is due not only to the technology's massive range of applications, but its potential for reducing inventory due to being able to print on demand (Choong et al., 2020). One of the downsides, however, of this new up and coming industry is the large amount of plastic waste produced. As consumers, it is important to be conscious of the waste produced and available methods to recycle and

reuse the scrapped prints and wasted filaments. The 3D printer allows a user to utilize inexpensive plastic filaments such as PLA or ABS to print a design, provided the design abides by the laws of physics. 3D prints do not form properly every time, and many attempts are often needed before a quality final product is achieved. The inefficiency of creating a quality print highlights the environmental dilemma and asks an important question: "How does one create, design, and print as many prints as desired without negatively impacting the environment?". The short answer: recycling. What if the 3D plastic waste can be reused and recycled back into the same 3D printer? (GB)

The proposed project concept will accomplish the goal of recycling 3D waste. Due to the current increase in popularity of 3D printing, the recycling project is highly desired by many in the additive manufacturing community. According to M.A. Krieger et al.:

It is concluded that with the open-source 3-D printing network expanding rapidly the potential for widespread adoption of in-home recycling of post-consumer plastic represents a novel path to a future of distributed manufacturing appropriate for both the developed and developing world with lower environmental impacts than the current system. (p. 90)

There have been many studies conducted to show that not only is household recycling for 3D printers possible, but also practical. To push this thought further, imagine also being able to recycle a disposable water bottle and utilize the pellets in a 3D printer. The proposed project will recycle and reuse plastic from a 3D printer (PLA and ABS) as well as PET plastics from household waste, such as disposable water bottles, by grinding, extrusion, and spooling into filaments that the 3D printer can use for future prints. (GB)

Currently unused and rejected 3D printed projects are either thrown away via landfill or recycling centers or in some cases the failed prints, as well as other recycled plastics, are being upcycled by hobbyists using a homemade extrusion contraption. A quick internet search will produce a list of videos and articles that show how to make one of these contraptions using a DIY kit available for purchase or by buying the individual components and building from scratch. One example of a DIY project is the RepRapable Recyclebot (Woern et al., 2018). This project comes with all the instructions to make a homemade plastic recyclable extruder, but none of the tools or parts to make it. This makes it impractical for the average user as the average user does not have the skills or time to safely construct the build. Another concern with the DIY method is the lack of guarantee for filament quality and user safety. (DL)

One of the key challenges to the proposed project will be the limitations of working with the melting and cooling of plastics. Plastic can be melted and reformed only so many times before it begins to lose its crystalline structure. This crystalline structure and its properties must be closely monitored to allow for the plastic structure to operate as the filament design requires. 3D filaments require a specific brittleness, flexibility, and malleability to perform as desired during the printing process. If the same material is recycled multiple times without additives to correct the changes in structure, the

resulting filaments will degrade up to 30% after 3 cycles and 60% after 7 cycles. (Pillin et al., 2008; Brüster et al., 2016, as cited in Mikula et. al, 2021). Another study on the impacts of recycling by extrusion of plastics (Vidakis et al., 2021) found the following:

...mechanical properties (of PLA filaments) overall increase until the 4th recycling course regarding the tensile strength, while there is an overall increase until the 3rd recycling course in flexural strength. Moreover, the impact strength results follow a similar trend as the overall increase ceases at the 3rd recycling course. (p. 10)

Therefore, the amount of recycled vs fresh material used in the extrusion process will require monitoring to ensure the final product performs properly. Another factor in determining this ratio of fresh to used materials is the amount of recycled material in the fresh filaments from the manufacturers. "An increasing number of companies offer filaments from recycled PLA or ABS" (Mikula et al., 2021, p. 12323). As such, the amount of recycled material may be unknown from the start. A potential solution to this challenge can be the tracking of purchased lot numbers and creation of lot numbers for each recycled spool to keep track of the number of times the material has been recycled. Another potential solution is, "the addition of additives to the (pelletized) material may occur in the feed zone (of the extruder) in order to grant distinctive physicochemical properties to the end product, such as color, hardness, erosion strength, etc." but knowing which additives to add or obtaining these additives as a standard consumer may not be practical (US20160107337A1). Due to the sensitive nature of the filament structure, mixing of plastic types (PLA, ABS, PET, etc.) is not possible. Therefore, sorting plastic types is each waste type may be a barrier for some consumers. (LL)

Another limitation to the extrusion process involves the various filament colors available for 3D printers. The color of the recycled filament will need to be monitored to ensure that the extrusion process does not vary the color from the original raw material. If color is not monitored and more than one spool is used during the printing process, the final print could result in mismatched or different shades of the same color. Keeping this color consistency will be a challenge. (LL)

The last barrier faced with current designs is the physical space that the setup requires to produce a viable product. Many hobbyists and home consumers do not have a large space to hold both their 3D printer and a recycling unit which contains a grinder, extruder, cooling unit, and spooler. (LL)

The proposed project concept bears some similarities to existing extrusion systems in terms of its core mechanical processes and functionality. Like many current solutions, this system is focused specifically on the recycling of PLA, ABS and PET plastics. PLA and ABS are the current plastics of choice for consumer 3D printing while PET is a common household waste and, given the prevalence and suitable physicochemical properties of these three plastics, all are popular target materials for filament extrusion (Kreiger et al., 2014). In addition, the proposed extrusion mechanism consists of the combination of screw, cylinder, heating elements, and nozzle. This is a commonly

used configuration among systems currently in use, and its design is well-documented (Berchuk et al., 2016). The design proposed here seeks to incorporate mechanical and operational concepts that have proven successful at any scale while avoiding the pitfalls that have limited consumer access to a viable filament extruder. (WW)

Current technologies employ a variety of sensors to ensure output quality and proper behavior of the system. Correct temperature of the plastic at various points and consistent diameter of the resulting filament are crucial. Other measurements, such as the feed rate at the entrance of the extruder and environment variables may also be of value but may not be necessary or appropriate for this system. Current designs vary in terms of the types of sensors used as well as the locations in which these sensors are placed along the line of production. A study on real-time analysis of plastic filaments (del Burgo et al., 2019) made use of thermistors and an analog temperature sensor to monitor heating and cooling processes, as well as an optical encoder to measure feed rate. Another system employed similar technologies for temperature measurement (Woern et al., 2018). In addition, an optoelectronic coupler to detect undue filament stretching prior to cooling has been demonstrated (Teterin et al., 2016). Filament diameter measurement has been accomplished with an optical micrometer (*Filament sensor kit for 3D printers and filament extruders*, 2020) as well as a photodiode array sensor (del Burgo et al., 2019). All these technologies are implemented, in various combinations, in a collection of extrusion systems that bear certain similarities to one another, but are quite diverse in terms of scale, precision, and cost. (WW)

Among all the systems studied, sensors for filament diameter and temperature measurements provide a baseline for proper operation and output precision, and the design proposed here will accordingly employ both. Measuring the feed rate may be pursued as well since, while not necessary, sensors for this purpose have been implemented in relatively small-scale devices and can increase the reliability of the system (Teterin et al., 2016). Other sensors, especially those intended to monitor room temperature and humidity, will likely be omitted in favor of simplicity and with regard for budget constraints. This choice differs from many industrial and laboratory-grade systems, and instead bears more similarity to a DIY-type system that would be intended for usage more similar to that of the proposed device. (WW)

In broader terms, this system differs from presently available technologies in terms of the cost of the system, as well as the levels of safety, filament quality, and autonomy relative to that cost. Industrial and laboratory filament extrusion systems are technologically advanced and provide a high degree of precision but are very costly, and thus impractical for the average individual user (Woern et al., 2018). At the opposite end of the spectrum small-scale, DIY-type systems lack structural integrity, safety, electronic controls and autonomy, and filament output quality. Beyond prohibitively expensive systems, these low-quality extruders are, for the most part, the only solution currently available to average users. DIY extruders are typically of suboptimal construction and have limited electronic monitoring and user control. Accordingly, these devices face durability issues, and pose significant safety risks, especially in relation to the grinding and heating mechanisms central to their operation.

The system proposed here presents a middle ground between the scale and price points of the devices described above and will provide an affordable user experience that is safe, effective, and semi-autonomous. With the plethora of sensors outlined above and a robust embedded control system, this device will be safer, easier to monitor, and easier to operate than low-cost options on the market today. In contrast with the designs presently available, this system presents a solution to PLA and PET waste reclamation that is both practical and widely adoptable in the 3D printing community. (WW)

There are a few patents that are relevant to the proposed design. One such patent is the Plastic Extrusion, Apparatus and Control, designed by Arthur William Spencer. This patent focuses on obtaining a more uniform extraction of filament and steady flow rate after being heated and run through a die. This was accomplished by maintaining a consistent temperature and varying the pressure accordingly (US3148231A). Another patent process that explores a similar problem is introduced by Xiaofan Luo and Zhaokun Pei. This patent explores and offers a solution to the problem of 3D printer filament prematurely softening in the printing nozzles by suggesting the use of certain polymers with "…better resistance to heat-induced softening." (US20170066188A1). This is one of the major causes for the filament to jam in the nozzle during the extrusion process. This in turn required the extrusion process to stop, and the jam removed prior to operation restart. Both patents are relevant to the proposed project as both focus on the resulting filament being uniform and consistent. These concepts are key to creating a filament that has the same or better 3D print success rate as the purchased filaments from industry providers. (DL)

1.4. Marketing Requirements

The system shall:

1. Be safe to operate for any user whether at a commercial or home application
2. Be power efficient and can operate with standard power available in a home or office
3. Produce a filament that is usable with a 3D printer
4. Produce a high yield product from recycled materials in a reasonable amount of time
5. Use recycled PLA or PET material provided by the user
   (LL)

2. Engineering Analysis

This section outlines the analysis completed to support the marketing and engineering requirements of the system. The analysis is broken up into the following sections: Circuits & Electronics, Cooling Tank Analysis, System Speed Analysis, Embedded Systems, and Controls.

2.1. Circuits & Electronics

2.1.1. Power Analysis

To ensure the system design is power efficient, preliminary power calculations were completed. These calculations took into consideration the marketing requirement to use this product in a home or office with a standard 120V outlet. A successful analysis ensures the system can operate on a 15A breaker which is typical in these environments.

The design will have multiple items drawing current that need to be considered when determining total current draw. These items are as follows:

- Grinder Motor
- Hopper Latch Motor/Actuator
- Spooling Motor
- Two Pulling Motors
- Extrusion Screw Motor
- Control Board
- Extrusion Heating System

The operating assumptions used in the Power Analysis are as follows:

- 120V outlet is used
- Screw length to diameter ratio is 36:1
- Screw is at maximum length of 3 feet
- Screw will exhibit the minimum torque of 53lb-in Torque
- Output spooling speed will be 12 ft/min
- Grinder will operate in the same manner as a standard office paper shredder
- Puller motors will not exceed 50 rpm (based on below rpm calculations)
- Spooling motor will not exceed 50 rpm (based on below rpm calculations)
- Spooling & Pulling motors will be like the FIT0492-A motor (Digikey).
- The control board will be within the PIC32 family

Further explanation of power analysis assumptions and calculations are explained in Table 1.

Equation 1 defines a relationship between linear movement and rpm.

$$n = \frac{s}{\pi D}$$

Equation 1: RPM to linear movement s

Using the ratio and length assumptions, the following is calculated:

$$D = 36 * \frac{1}{36} = 1\ in$$

Next, the following is calculated using Equation 1:

$$n = \frac{\frac{12ft}{m}}{\pi * 1in * \frac{1ft}{12in}} = 45.84rpm$$

For Spooling at 12 ft/min, and using Equation 1 with D = 1in, n is calculated to be 45.84 rpm. Next, the Power Torque RPM relationship was defined.

$$Power(Hp) = Torque(lb * in) * \frac{RPM}{63025}$$

Equation 2: Power, Torque and RPM relationship

Using Equation 2, and the previous results, and the Torque assumption, the following is calculated:

$$Power(Hp) = 53lb * in * \frac{45.84}{63025} = \ 0.028885$$

Using the following hp to kW conversion rate defined in Equation 3:

$$1(Hp) = 0.7457(kW)$$

Equation 3: Hp to kW conversion

$$Power(kW) = 0.215395kW = 215.395W$$

This is the wattage for the Extruder screw's motor. The next equation to be defined is Equation 4:

$$\frac{Watts}{Volts} = Amps$$

Equation 4: Variation of Ohm's Law

Using this equation will show amperage based on each of the item's power usage. The results are laid out in the following table:

Table 1: Power Table

Power Item	Reasoning	Wattage	Voltage	Amps
Grinder Motor	The grinder motor will operate in the same manner to a paper shredder. The grinding necessary to pelletize plastic requires two	200 W	120VAC	1.666 A

	screws spinning in tandem just like a paper shredder.			
Spooling Motor & Pulling Motors	The Spooling and Pulling Motors will simply need to keep up with the same speed as the outflow from the nozzle which is 45.84 RPMs as calculated previously. Therefore, the power wattage will be similar to the FIT0492-A motor as it is a max speed 50 RPM motor.	5 W	12 VDC	0.42 A per motor
Hopper Latch Motor	The Latch Motor will either be servo or linear actuator.	5 W	12VDC	0.42 A
Extruder Heating	These calculations are done in extensive detail in the next section.	690.6 W	120VAC	6.3 A
Extruder Screw	The screw was already calculated above.	215.4 W	120VAC	1.795 A
Control Board	The control board needed for this design will be within the PIC32 family.	5 W	5 VDC	1 A

Totaling the amperage column of Table 1 and accounting for four 0.42 A motors, the following result was found:

$$1.666A + 4 * 0.42A + 6.3 + 1.795A + 1A = 12.441\ Amps$$

The resulting amperage when totaled is approximately 12.441 Amps. Since the grinder will not be operating during the extrusion process, the resulting total amperage would be 10.775. This means the system is safe to use on a 15 Amp standard household circuit. This includes utilizing a safety factor of 80% which is 12 amps.

(GB, DL)

2.1.2. Extruder Analysis

Below are chemical properties of both PLA and PETG that are required to complete the analyses for the extruder.

Table 2: Chemical Properties of Plastics

	PLA	PETG
Melting Point (^oC)	150-160	250-260
Specific Heat Capacity (J/kg ^oC)	1590	1300
Density	1.23	1.25

The following formulas are used to calculate the thermal energy required to melt both PLA and PETG.

$$P = c\rho Av\Delta T$$

Equation 5: Thermal Energy
Where P is Power, c is specific heat capacity, ρ is density, A is area, v is velocity and T is the change in temperature

Minimum Power Required to Melt 7mm pellet PLA at 160°C

$$\frac{1590\ \text{J}}{\text{kg°C}} * \frac{1kg}{1000g} * \frac{1.23g}{cm^3} * \frac{1cm^3}{1000mm^3} * \pi * \left(\frac{7mm}{2}\right)^2 * \frac{304.8mm}{1ft} * \frac{9.25ft}{min} * \frac{1min}{60s} * (160 - 25)^oC = 477.4W$$

Minimum Power Required 7mm pellet PETG at 260°C

$$\frac{1300\ \text{J}}{\text{kg°C}} * \frac{1kg}{1000g} * \frac{1.25g}{cm^3} * \frac{1cm^3}{1000mm^3} * \pi * \left(\frac{7mm}{2}\right)^2 * \frac{304.8mm}{1ft} * \frac{9.25ft}{min} * \frac{1min}{60s} * (260 - 25)^oC = 690.6W$$

As PETG requires more thermal energy to melt, the minimum power required to melt both plastics is 690.6W.

(LL)

2.2. Cooling Tank Size Analysis

Below uses thermal heat transfer equations to determine the minimum volume of water required to cool the molten extruded plastic to room temperature.

$$Q_{plastic} = Q_{water}$$

$$Q = mc\Delta T$$

$$m = \rho * V$$

$$V_w = \frac{(\rho V c\Delta T)_{plastic}}{(\rho c\Delta T)_w}$$

Equation 6: Cooling Tank Volume
Where Q is heat transfer, m is mass, c is specific heat capacity, T is temperature, ρ is density and V is volume

Minimum volume to cool PLA

$$\frac{V_w}{t} = \frac{\frac{1.23g}{cm^3} * \pi\left(\frac{1.75mm}{2}\right)^2 * \frac{9.25ft}{s} * \frac{304.8mm}{ft} * \frac{(1cm^3)}{1000mm^3} * \frac{1590J}{kg^oC} * \frac{1kg}{1000g} * (160^oC - 25^oC))}{\frac{1g}{cm^3} * \frac{4.184J}{g^oC} * (30^oC - 25^oC)}$$

$$\frac{V_w}{t} = \frac{85.6cm^3}{s} * \frac{1in^3}{16.3871cm^3} = 5.22\frac{in^3}{s}$$

Minimum volume to cool PETG

$$\frac{V_w}{t} = \frac{\frac{1.25g}{cm^3} * \pi\left(\frac{1.75mm}{2}\right)^2 * \frac{9.25ft}{s} * \frac{304.8mm}{ft} * \frac{(1cm^3)}{1000mm^3} * \frac{1300J}{kg^oC} * \frac{1kg}{1000g} * (260^oC - 25^oC))}{\frac{1g}{cm^3} * \frac{4.184J}{g^oC} * (30^oC - 25^oC)}$$

$$\frac{V_w}{t} = \frac{123.8cm^3}{s} * \frac{1in^3}{16.3871cm^3} = 7.55\frac{in^3}{s}$$

Assume the system will cool the heated filament to room temperature in 5 seconds yields a minimum water bath size of 1"x1"x8" tank.

(LL)

2.3. System Speed Analysis

Below is the analysis to determine the minimum extrusion, cooling and spooling speed required to produce 1kg of filament in 2 hours.

$$l = \frac{m}{\rho A}$$

Equation 7: Filament length
Where l is length, m is mass, ρ is density and A is cross sectional area

PLA length of 1kg

$$1kg\ PLA * \frac{cm^3}{1.23g} * \frac{1000g}{1kg} * \frac{1000mm^3}{1cm^3} * \frac{1}{\pi * \left(\frac{1.75mm}{2}\right)^2} * \frac{1ft}{304.8mm} = 1091\ ft\ PLA$$

PETG length of 1kg

$$1kg\ PETG * \frac{cm^3}{1.25g} * \frac{1000g}{1kg} * \frac{1000mm^3}{1cm^3} * \frac{1}{\pi * \left(\frac{1.75mm}{2}\right)^2} * \frac{1ft}{304.8mm} = 1109\ ft\ PETG$$

Next, the overall length of the filament is converted into a velocity based on the amount of time to create the desire length.

$$v = \frac{l}{t}$$

Equation 8: Velocity
Where v is velocity, l is length, and t is time

As PETG requires more length to achieve 1kg of material the minimum system run speed must be based on the PETG final length.

$$\frac{1109ft}{2hr} * \frac{1hr}{60min} = \frac{9.25ft}{min}$$

To achieve the engineering requirement to complete 1kg of material in 2 hours, a 9.25 ft/min minimum extrusion speed is required.

(LL)

2.4. Embedded Systems

Real time tracking within the extruder system is required due to the need to adjust temperature and extrusion motor speeds based on the extrusion output diameter. Specifically related to the hot filament diameter in relation to the heating elements, the best controller for this project will be a 32-bit microcontroller; in particular, a device in the PIC32MX family. 32 bits was deemed to be the necessary word length for a variety of reasons, including the need for 32-bit timers and high resolution for processing sensor signals and driving motors. A System on Chip (SoC) device such as Raspberry Pi or the TI Beaglebone, which are available in 32- and 64-bit varieties, were also considered, but the 32-bit PIC microcontroller was favored for its accompanying MBLAB Harmony development environment, and the device's compatibility with real-time operating systems. FreeRTOS will be used to implement real-time threads, priorities, and scheduling. This configuration of the software system is optimal for time-critical changes in heating element temperature and motor speed that will be necessary to the system's consistent production of usable filament. The RTOS is favorable over the more desktop-like, multi-tasking OS of the Raspberry Pi or similar devices, because the RTOS has maximum execution times for critical operations, low-latency interrupts, and a simple scheduler that adheres strictly to task priority rules. This combination of technologies makes it possible to develop a peripheral-intensive application that can run on a single processing unit without sacrificing execution time for the critical tasks of motor and temperature control.

(WW)

2.5. Controls

There will be a few key locations that will control the overall result of the product or the movement from one station to the next. Each of these controls will be monitored by the controller and decisions determined by the source code for each sensor and feedback control.

The first necessary control to the system will relate to all the safety functions, such as the status of the lid to the grinder hopper, the status of all E-stops, and the necessary safety

components needed for the cooling tank water and the surrounding electronics. This control needs to be a top priority interrupt such that in the event a safety issue occurs, the system shuts down immediately. All safety controls will either stop a system from starting or cut power to the entire system depending on the severity of risk, should the safety control not be in place.

The second necessary control to the system will be to ensure there is enough ground pellets prepared before stating the extrusion system. If the weight sensor in the catching hopped located after the grinder does not hold a weight greater to or equal to the final spooled filament weight plus the yield loss, the extruder motor cannot start. This ensures a constant extrusion of material which is necessary to create a consistent filament.

The third control relates to maintaining the diameter of the hot filament exiting the extruder. The filament diameter will be dependent on two factors: the temperature zone ranges and the screw speed. A diameter sensor must send real time feedback to the controller to make immediate adjustments to either the temperature zone ranges, the screw motor speed, or both. This will ensure the diameter maintains the tolerances specified in the engineering requirements. Should the diameter sensor not detect any filament, the extruder motor will be stopped so that trouble shooting can occur. Within this same control, each individual temperature zone must be monitored in real time and adjustments made to ensure the temperatures stay within the required temperature ranges to ensure a consistent melt of the pellets.

The fourth control system would relate to the temperature of the heating element. The heating elements will have to stay at their specified temperatures during the extrusion process. This will be completed by controlling the amount of power being delivered to the coils. If it gets to hot then power will be reduced, if it gets to cool then power will be increased. The controlling of power will depend on a few factors such as the temperature of the coils and how fast the temperature is changing. This will be implemented by reading the temperature and then sending the appropriate response to the circuit.

The final key control to the system will be the stop all processes functionality once the final spool weight is met. This process must be handled in a specific pattern to ensure the filament does not break or become too slack between the extruder and the spooler. All motors will need to decrease and stop at the same rate to maintain tension. The extruder temperature zones must be stopped in order to allow the system to cool, and the grinder must not be able to start again until the next production run is initiated.

(LL)

3. Engineering Requirements

Table 3: Marketing and Engineering Requirements and their justification

Marketing Requirements	Engineering Requirements	Justification
1	1. System will have an E-stop	E-stop will provide the user with the ability to cut power to all components

	2. All water from the system will be contained and kept away from all electrical components	of the system in an event of an emergency. Isolating the water required to cool the filament from the electrical components will limit the risk of an electrical fire because of water contact.
2	3. Be powered by a standard 120V outlet and on a 15A breaker	A 120V outlet on a 15A breaker is a standard setup in most home and office environments. By ensuring the power required to the system remains below the 15A threshold on a 120V outlet will allow the system to run in most user locations.
3, 5	4. Produce a spooled filament of 1.75mm +/- 0.05mm diameter 5. Filament spool will be between 200g and 1kg in weight 6. Max grinder pellet size of 7mm 7. Extruder will operate in a temperature zone between 150 ^{o}C and 300 ^{o}C 8. Extruder screw will be at or less than 3 feet in length 9. Extruder screw will maintain a 36:1 L/D ratio 10. Filament will spool at an ambient temperature	1.75mm filament on a 1kg spool is a standard size for 3D printing. The ability to produce less than 1kg will allow users to make less material if the 3D print does not require a full 1kg spool. By keeping the grinder pellet size at or below 7mm in size, will ensure an even melting of the pellets as it passes through the extruder. The melting points of PLA and PETG are 160^{o}C and 260^{o}C respectively. By operating in a range of 150 ^{o}C -300 ^{o}C, the plastics will be able to melt to be extruded into a filament. A key component of an extruder is the feeding screw. By keeping the screw length less than 3 feet ensures the system can be built on a standard desk at the user's location. An extruder screw design standard has the length to diameter ration of 36:1. If the filament is not cooled after exiting the extruder, the filament can lose shape and distort the size making it unusable in a 3D printer.
4	11. 1kg of filament will be produced in 2 hours or less time 12. System yield will be greater than 90%	An extrusion and spooling time of 1 full roll in 2 hours ensures the user can complete the recycling process in a reasonable amount of time without needing to risk errors occurring during a longer duration which is harder to manage.

		A yield of greater than 90% ensures the efficiency of the system and maximizes the use of the recycled material.

(GB, DL, LL, WW)

4. Engineering Standards Specifications

This section outlines the standards necessary to ensure this design is safe, professional, and not breaching any legal documentation or codes.

4.1. Safety

This system will have mechanical moving parts that could cause injury to unwary operators. The main extrusion screw will need to be always covered during operation. If the screw is exposed during operation, any foreign appendage inserted into the screw will cause injury. The screw operates using several thermal couples which will heat the screw housing to dangerously high temperatures. When unguarded, the housing could cause burns to unwary operators. The nozzle will also operate at a dangerous temperature and should not be touched until system has reached room temperature. The water bath should not be disturbed during operation as the heat of the bath may become mildly irritating. It is also unwise to disturb water near an electrical machine whilst it is still in operation. The system utilizes a grinding unit whic will cause severe injury to any foreign appendage inserted into said device. Use extreme caution when pelletizing scrap plastic. This system is designed to operate on a 120VAC outlet utilizing a 15A breaker system. It is important not to overload the system as this will flip said breaker. Ensure use of a dedicated 15A circuit for maximum safety.

(GB)

4.2. Communication

I^2C and SPI are serial communication protocols that will used to facilitate interaction between the microcontroller and peripheral devices.

(WW)

4.3. Design Methods

4.4. Programming Languages

The languages utilized will be C++, C, and Python.

(WW)

4.5. Connector Standards

NEC codes will be followed. This code states that the standard house will operate on a 15A breaker system, so this fact influenced this product's design.

(GB, LL, DL, WW)

5. Accepted Technical Design

This section outlines the standards necessary to ensure this design is safe, professional, and not breaching any legal documentation.

5.1. Hardware Design

5.1.1. Level 0 Hardware Block Diagrams

Error! Reference source not found. shows the fundamental level 0 hardware block diagram for the Additive Manufacturing Waste Management System – Plastic Extrusion Process. The system will accept recycled PLA or PET, 120 VAC, coolant (water), and user inputs such as plastic type and the desired weight of the filament. The output of this system will be spooled filament, the final weight of the spooled filament, extruder zone temperatures and the run time. A description of the functional requirements of this system is given in Table 4**Error! Reference source not found.** below.

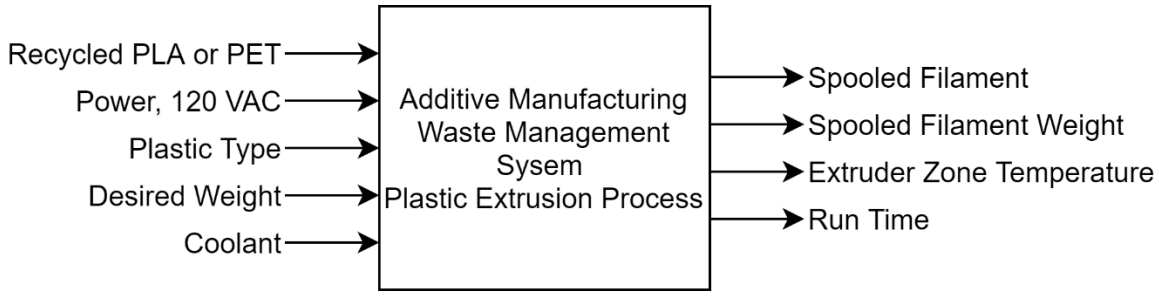

Figure 1: Level 0 Block Diagram

Table 4: Level 0 Hardware Functional Requirements

Module	**Additive Manufacturing Waste Management System – Plastic Extrusion Process**
Inputs	• Recycled PLA or PETG • Power, 120V AC • Plastic type • Desired weight • Coolant
Outputs	• Spooled 1.75mm filament • Spooled filament weight • Extruder zone temperatures • Run time
Functionality	Plastic extrusion process that converts 3D printing PLA waste and household PET waste into spooled filaments that can then be used in the 3D printing process.

(GB,DL)

5.1.2. Level 1 Hardware Block Diagrams

Figure 2 shows the level 1 hardware block diagram for the Additive Manufacturing Waste Management System – Plastic Extrusion Process. The system is broken up into subsystems which can be seen in Figure 2. Each subsystem takes in inputs and then sends the outputs to the next subsystem and information back to the controller. Material is placed into the grinding station where it is ground up into pellets. From there, the pellets move to the extruding station where they are melted and turned into filament. The filament is then cooled in the cooling station and wound up in the spooling station. This results in the final product of a spooled filament. The microcontroller sends signals to each system and powers the sensors in them. The power supply converts the 120V AC power into usable DC and AC voltages. Table 5 contains the functional requirements for all the subsystems.

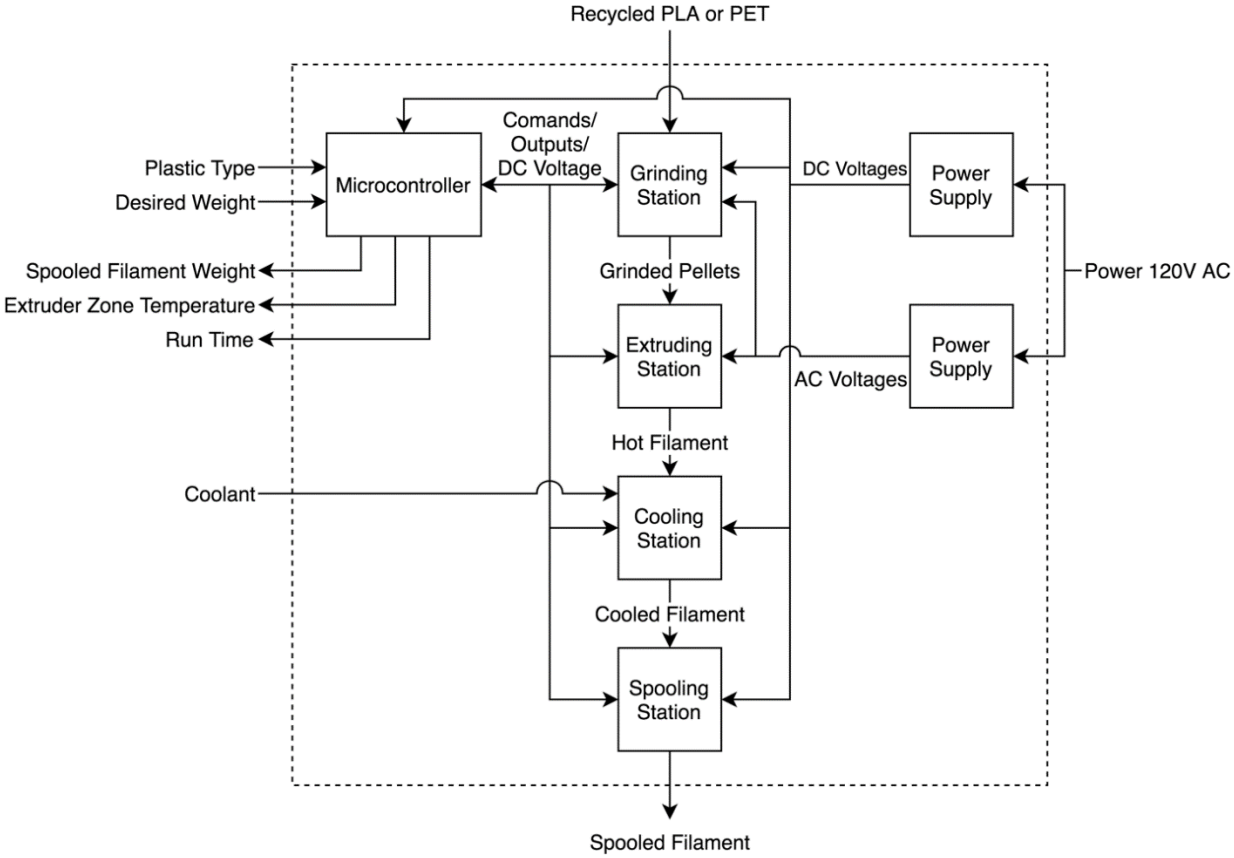

Figure 2: Level 1 Block Diagram

Table 5: Level 1 Hardware Functional Requirements

Module	**Power Supply**
Inputs	• Power, 120V AC
Outputs	• DC voltages • AC voltages
Functionality	Convert 120V AC wall outlet voltage to positive and negative DC output voltages and AC voltages to provide enough current to drive all motors and the microcontroller.
Module	**Microcontroller**

Inputs	• 5V DC • Plastic type • Desired filament weight • Initial pellet weight • Extruder zone temperatures • Filament diameter • Cooling station temperature • Solution level in tank • Flow rate • Spooled filament weight
Outputs	• Spooled filament weight • Extruder zone temperatures • Run time • Grinder start command • Extruder, cooling station and spooler start command
Functionality	The purpose of the microcontroller is to send the start commands for the motors and receive sensor inputs to monitor the system.
Module	**Grinding Station**
Inputs	• DC and AC voltages • Recycled PLA or PET • Start command
Outputs	• Grinded pellets • Initial pellet weight
Functionality	Grind the input material, PLA or PET into a desired pellet size of less than or equal to 7mm and weigh the final pellets to make sure there is sufficient material for the extruding process.
Module	**Extruding Station**
Inputs	• DC and AC voltages • Grinded Pellets • Heating elements start command • AC Motor start command
Outputs	• Hot filament • Filament diameter • Extruder zone temperatures
Functionality	Convert the grinded pellets into hot filament by moving the grinded pellets through an extruder that contains a heating system, screw and nozzle resulting in a 1.75mm hot filament.
Module	**Cooling Station**
Inputs	• DC voltage • Coolant solution (water) • Hot filament • Roller DC motor start command
Outputs	• Tensioned cooled filament • Cooling solution temperature

	• Cooling solution level • Flow rate
Functionality	Cool the hot filament from the extruder nozzle by running it through a cooling solution. Attach the cooled filament between rollers to guide the filament for spooling.
Module	**Spooling Station**
Inputs	• DC voltages • Tensioned cooled filament • Spooler DC motor start command
Outputs	• Spooled filament • Spooled filament weight
Functionality	Spool and weigh the cooled filament.

(GB,DL)

5.1.3. Level 2 Hardware Block Diagrams

Figure 3 though Figure 7 show the level 2 block diagrams for the subsystems in Figure 2. Each subsystem is further broken up to show the components that make up that system. Figure 3 shows the microcontroller block diagram. The user inputs are entered and then are stored in the microcontroller. The controller send signals and receives information about the other components in the system. Figure 4 shows the block diagram of the grinding station. The hopper moves the recycled material to the grinder. The grinder takes 120V AC and grinds the material into pellets that are less than or equal to 7mm in size. From there the pellets fall into another hopper with a weight sensor to ensure there is enough material to continue the process. From the grinding station, the pellets move onto the extruding station which can be seen in Figure 5. The pellets are transported through the extruder by a screw system and heated to the required temperature to ensure a melted state. The pellets melt and are forced through a nozzle at the end of the screw, producing hot filament. The heating element will take 120V AC along with a DC voltage from the microcontroller to measure the temperature. It will then produce enough heat to melt the pellets. An AC motor will take 120V AC to spin the screw at a controlled RPM. The filament is then cooled in the cooling station in **Error! Reference source not found.**. The holt filament goes through a cooling tank of water. The filament then goes through a set of rollers that help guide the filament to the spooling station, which can be seen in Figure 7. The rollers are powered by 12V DC. The tensioned filament is then connected to the spooler which contains a weight sensor to measure the final filament weight. The DC motor that turns this spooler is powered by 12V DC. **Error! Reference source not found.** shows the functional requirement table for the level 2 block diagrams.

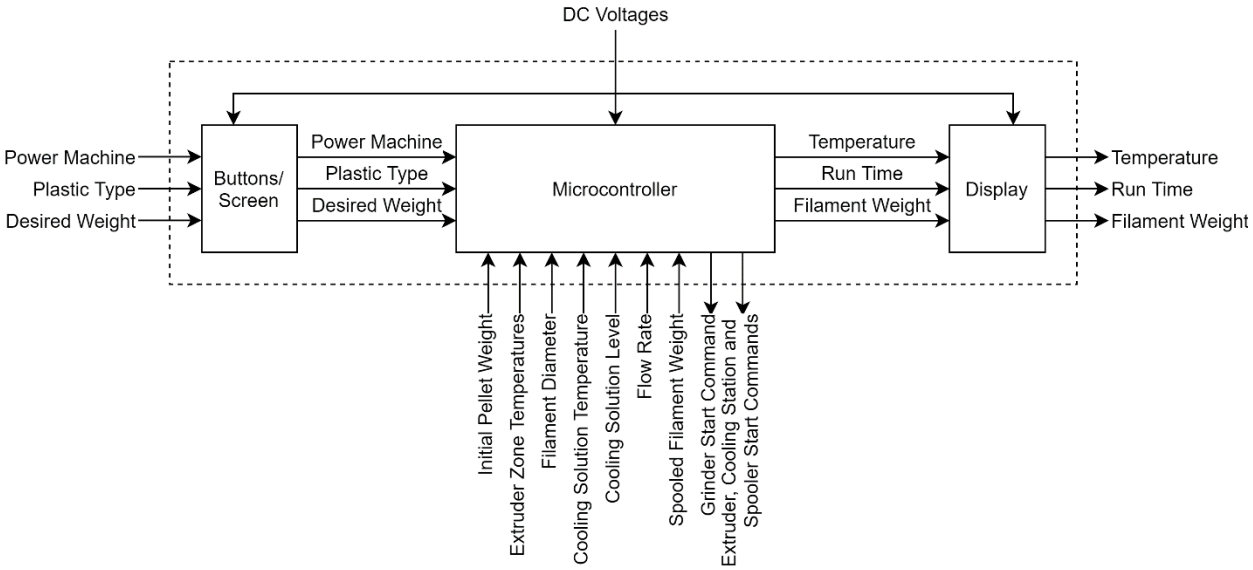

Figure 3: Level 2 Block Diagram - Microcontroller

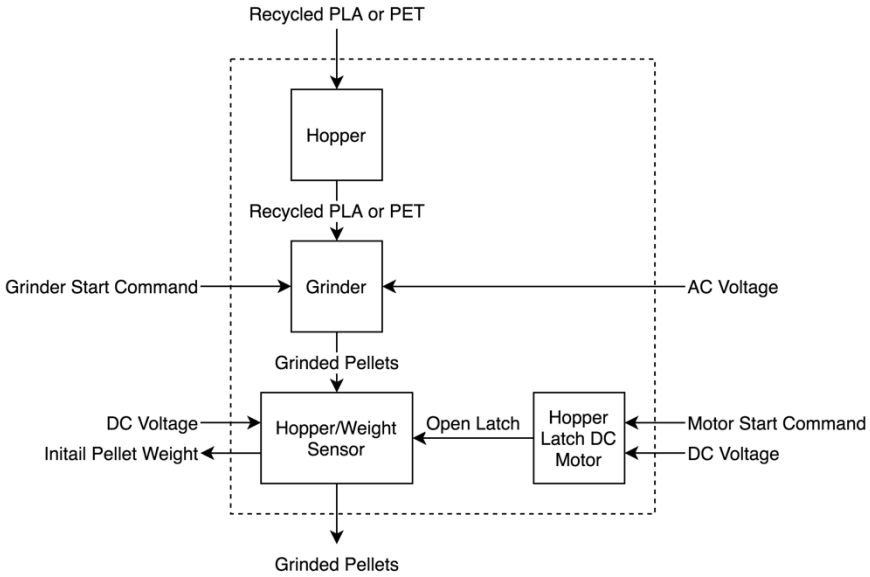

Figure 4: Level 2 Block Diagram - Grinding Station

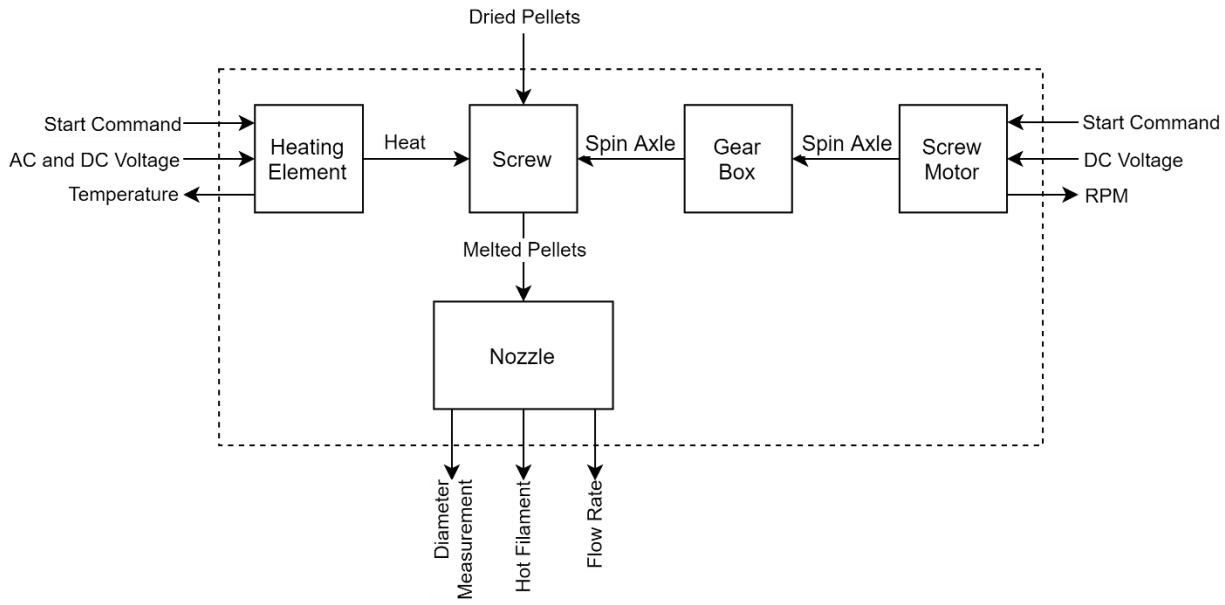

Figure 5: Level 2 Block Diagram - Extruding Station

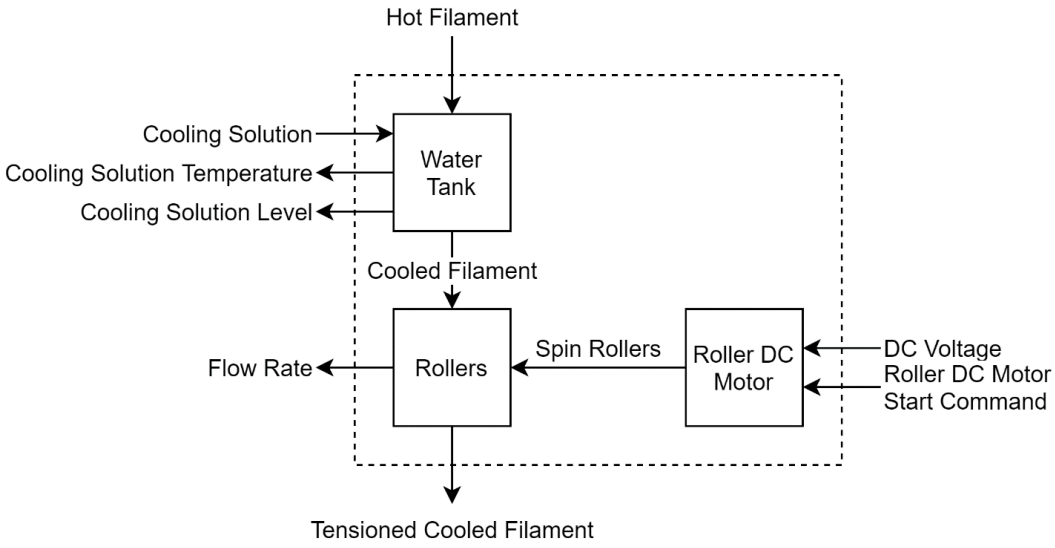

Figure 6: Level 2 Block Diagram - Cooling Station

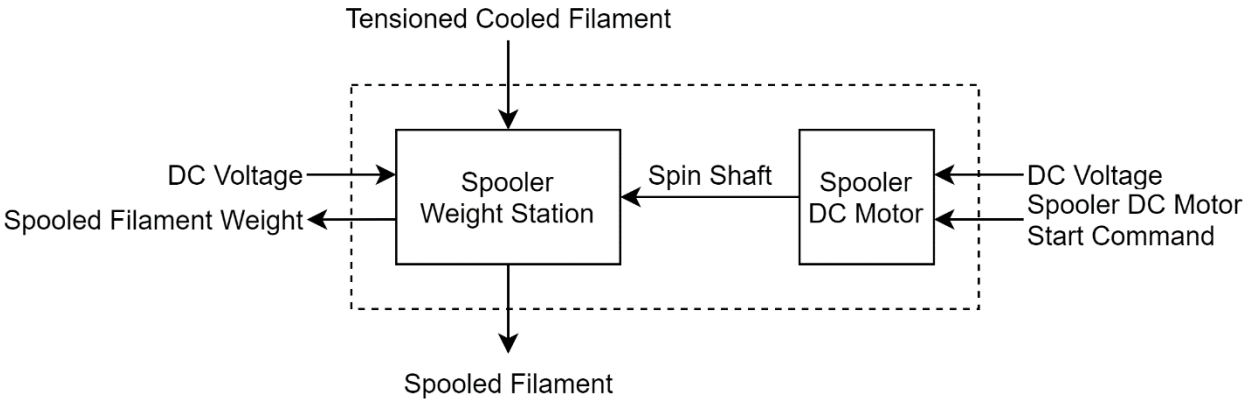

Figure 7: Level 2 Block Diagram - Spooling Station

Table 6: Level 2 Hardware Functional Requirements

Module	**Hopper**
Inputs	• Recycled PLA or PET
Outputs	• Recycled PLA or PET
Functionality	Hold the PLA or PET until the grinding process starts.
Module	**Grinder**
Inputs	• Recycled PLA or PET • 120V AC • Grinder start command
Outputs	• Grinded pellets
Functionality	Grind the PLA or PET into pellets that are less than or equal to 7mm in size.
Module	**Hopper/Weight Sensor**
Inputs	• Grinded Pellets • DC voltage for weight sensor
Outputs	• Grinded pellets • Initial weight of the pellets
Functionality	Total weight of the pellets to ensure enough has been processed to achieve desired end filament weight.
Module	**Hopper Latch DC Motor**
Inputs	• 12V DC • DC motor start command
Outputs	• Open Latch
Functionality	Open the latch to transfer the pellets from the hopper to the screw.
Module	**Screw**
Inputs	• Grinded pellets • Heat • Spinning shaft
Outputs	• Melted pellets
Functionality	Transport the grinded pellets through the heating element to melt the pellets.
Module	**Heating Element**
Inputs	• DC voltage for temperature sensor • 120V AC • Heating elements start command
Outputs	• Heat • Extruder zone temperature
Functionality	Provide heat to the screw encasement to melt the pellets.
Module	**Gear Box**
Inputs	• Spinning shaft
Outputs	• Spinning shaft
Functionality	Control the RPM from the DC motor to the desired RPM to turn the screw.
Module	**Screw Motor**
Inputs	• DC voltage • Motor start command

Outputs	• Spinning shaft
Functionality	Spin the shaft connecting to the gear box.
Module	**Nozzle**
Inputs	• Melted pellets
Outputs	• Hot filament • Filament diameter
Functionality	Force the melted filament through a 1.75mm nozzle to get the correct size hot filament.
Module	**Water Tank**
Inputs	• Hot filament • Cooling Solution
Outputs	• Cooled filament • Cooling solution temperature • Cooling solution level
Functionality	Run the hot filament through a cooling solution to cool the filament to a desired temperature to prevent the filament from distorting.
Module	**Rollers**
Inputs	• Cooled filament • Spinning roller shafts
Outputs	• Cooled filament • Flow rate
Functionality	Guide the cooled filament to the spooling station and providing tension to prevent the filament from getting tangled or breaking.
Module	**Roller DC Motor**
Inputs	• 12V DC • Roller DC motor start command
Outputs	• Spin roller shaft
Functionality	Spin the shaft connecting to the rollers
Module	**Spooler Weight Station**
Inputs	• Tensioned cooled filament • Spinning shaft • DC voltage for weight sensor
Outputs	• Spooled filament • Spooled filament weight
Functionality	Spool the filament.
Module	**Spooler DC Motor**
Inputs	• 12V DC • Spooler DC motor start command
Outputs	• Spin shaft
Functionality	Spin the spooler to spool the cooled filament

(GB, DL)

5.1.4. Level 3 Hardware Block Diagrams

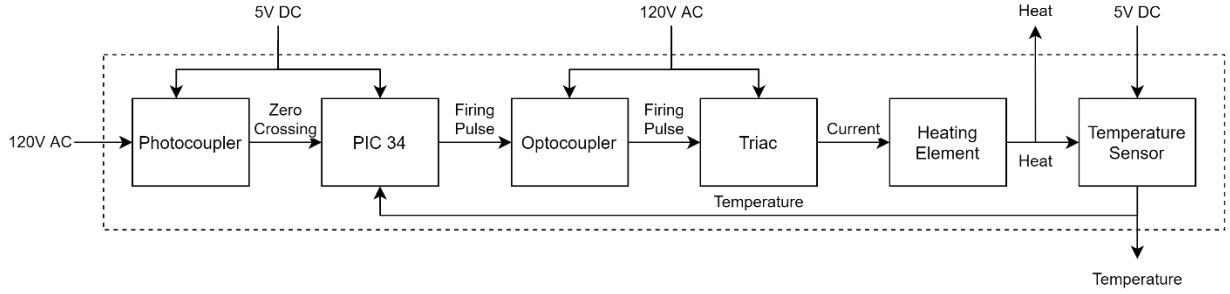

Figure 8: Level 3 Block Diagram - Heating Element

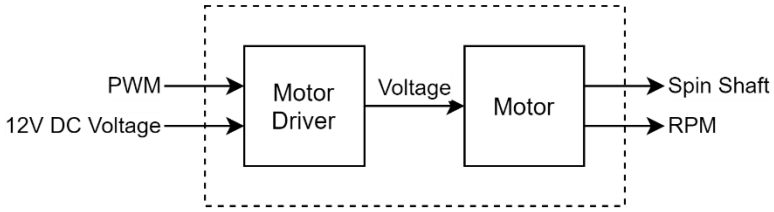

Figure 9: Level 3 Block Diagram - Motor Driver

Table 7: Level 3 Hardware Functional Requirements

Module	**Photocoupler**
Inputs	• 5V DC • 120V AC
Outputs	• Zero-crossings
Functionality	Locate when the 120V AC sine wave crosses the zero axis. Isolates the microcontroller from 120V AC voltage
Module	**PIC 34**
Inputs	• Zero-crossings • Temperature • 5V DC
Outputs	• Firing pulse
Functionality	Takes the current temperature of the heating element and determines how much power and when it should receive it to control the temperature
Module	**Optocoupler**
Inputs	• Firing pulse • 120V AC
Outputs	• Firing Pulse
Functionality	Takes the firing pulse from the PIC 34 and send it to the TRIAC. Isolates the microcontroller from 120V AC voltage
Module	**Triac**
Inputs	• Firing Pulse • 120V AC

Outputs	• Current
Functionality	Receives the firing pulse from the optocoupler which then activates the TRAIC. Completes the circuit and let the current flow through to the heating element
Module	**Heating Element**
Inputs	• Current
Outputs	• Heat
Functionality	Start heating up to the desired temperature based on the amount of power being sent
Module	**Temperature Sensor**
Inputs	• Heat • 5V DC
Outputs	• Temperature
Functionality	Measures the temperature and send it back to the microcontroller to maintain desired temperature
Module	**Motor Driver**
Inputs	• PWM • 12V DC
Outputs	• Voltage
Functionality	Control the speed and direction of the motor through PWM
Module	**Motor**
Inputs	• Volage
Outputs	• Spin shaft • RPM
Functionality	Receives a PWM signal and spins shaft accordingly

Figure 8 shows the level 3 block diagram of the heating element. It shows the process of controlling the amount of power that is delivered to the heating element. This process is further described in more detail and with circuit schematics in the circuit schematics section below. Figure 9 shows the level 3 block diagram of how the motors are to be driver. Each motor will be connected to a motor driver. The motor driver will receive a PWM signal which will then activate the circuit and make the motor spin. This process is also further described in circuit schematics section below. Even though the motor drivers are different, they behave in the same manner. Table 7 shows the functional requirement table for the level 3 block diagrams.

(GB, DL)

5.1.5. Circuit Schematics

To power the motors and the PIC34 board, a voltage regulator circuit was designed to output the desired DC voltages of 12V and 5V. This regulator was modeled in LTspice and can be seen in Figure 10 below.

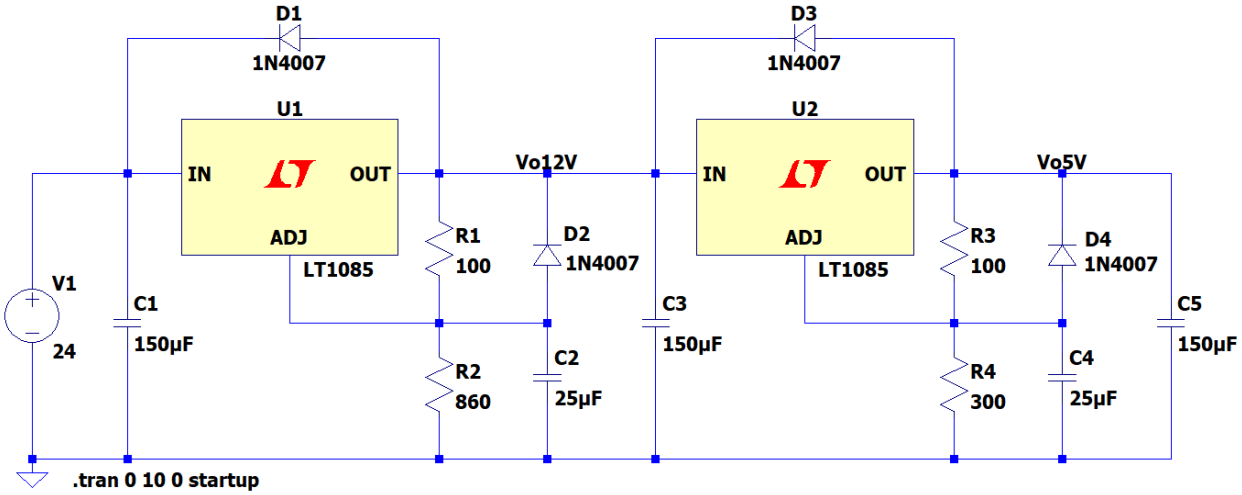

Figure 10: Voltage Regulator Circuit

This circuit uses a 24V power supply that provides up to 5 amps of current and will be purchased. The capacitors C1, C3 and C5 help improve the transient response while capacitors C2 and C4 help reject any ripple introduced. Diodes D1 and D3 protect the devices against any input short circuit while D2 and D4 protect against any output short circuit. The 1N4007 diodes were used in the simulation because of their voltage and amperage ratings, but any similar diode can also be used in its place.

The voltage regulator LT1085, U1 and U2, were used in the simulation. This is an adjustable voltage regulator that outputs the desired voltage based on the resistors connected to the output and the adjust nodes. The resistors R1 and R2 correlate to the 12V output while resistors R3 and R4 correlate to the 5V output. The values of these resistors were found using the equation in the data sheet for the LT1085 and can be seen in Equation 9. This equation was rearranged to get Equation 10, which simplified the calculations.

$$V_{OUT} = V_{REF} * \left(1 + \frac{R_2}{R_1}\right)$$

Equation 9: Finding R_1 and R_2

$$\left(\frac{V_{OUT}}{V_{REF}} - 1\right) R_1 = R_2$$

Equation 10: Finding R_2

V_{OUT} is the desired voltage while V_{REF} is the reference voltage across R1 that the regulator develops and is about 1.25V. To get a voltage of 12V, R1 was chosen to be 100Ω which resulted R2 to be 860Ω. To get a voltage of 5V, R3 was chosen to be 100Ω which resulted R4 to be 300Ω. The simulation results can be seen in Figure 11 below. It shows that the voltage at node Vo12V reaches 12V and the voltage at node Vo5V reaches 5V. These voltages reach their respective voltage level and remain there.

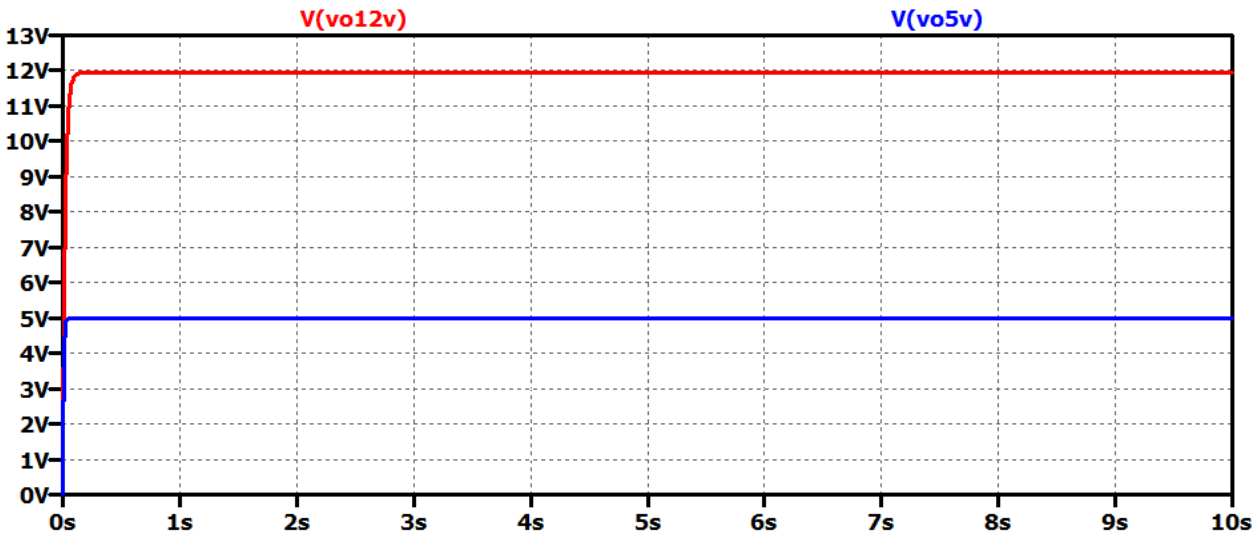

Figure 11: Voltage Regulator Circuit Simulation

A motor driver was designed to drive the main screw motor. This driver can be seen in Figure 12. Only one motor driver was designed while the rest were purchased for the smaller motors.

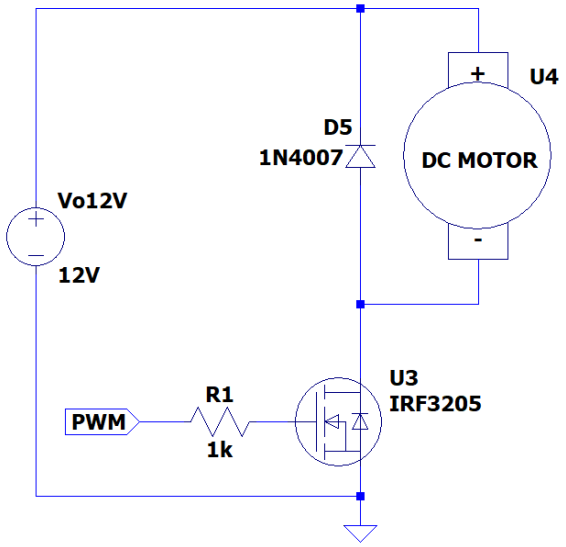

Figure 12: Motor Driver for Extruder Motor

The motor is powered by the 12V DC output supplied by the voltage regulator. The diode D1 is placed in parallel with the DC motor to help prevent any back EMF from entering the circuit and possibly damaging the components. A MOSFET is used to complete the circuit and allow the current to flow through. The IRF3205 MOSFET, U3, was chosen since it provides plenty of protection against voltage and current spikes. A PWM signal from the microcontroller is used to adjust the on and off times of the motor, in effect controlling the speed of the motor. To protect the microcontroller, a 1kΩ resistor, R1, is placed between the microcontroller and the MOSFET gate.

To power and control the temperature of the heating element, some sort of temperature control circuit is needed. A system was designed to detect the zero crossings of the input

voltage and then send a signal to the microcontroller. This system can be seen in Figure 13. The microcontroller would then determine the appropriate response and send a signal to the rest of the system. The rest of the system can be seen in Figure 15.

The system in Figure 13 was designed to detect the zero crossing of the input voltage. The input voltage is 120V RMS 60Hz, which is equal to the voltage coming from a standard wall receptacle. The voltage then passed through a full bridge rectifier, U5, before entering the photocoupler. This converts the sine wave into only positive voltages so that the microcontroller can read the values. The 4N25 photocoupler, U6, is used to detect the zero crossings of the input voltage and then limit the voltage to 5V since the microcontroller is limited to 5V. It has a LED inside pulses and shines a light onto the phototransistor when there is current flowing through it. This causes the phototransistor to draw current that is proportional to the LED brightness. When the voltage approaches zero, the LED gets dimmer which then signal the microcontroller that the zero-crossing point is detected. Figure 14 below shows the rectified voltage along with the output pulse detecting the zero-crossing point.

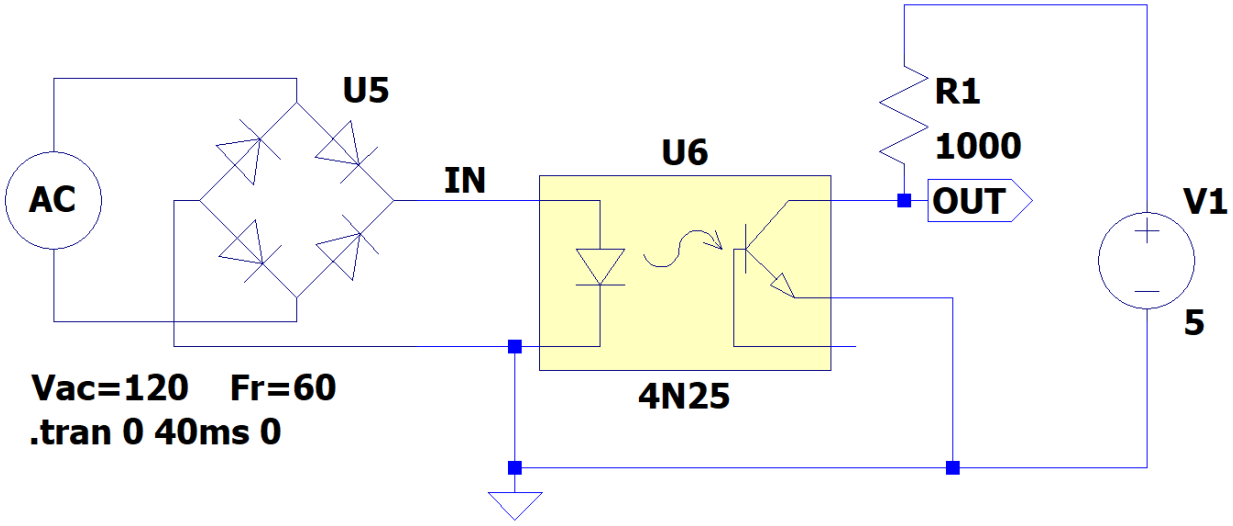

Figure 13: Zero-Crossing Detector Circuit

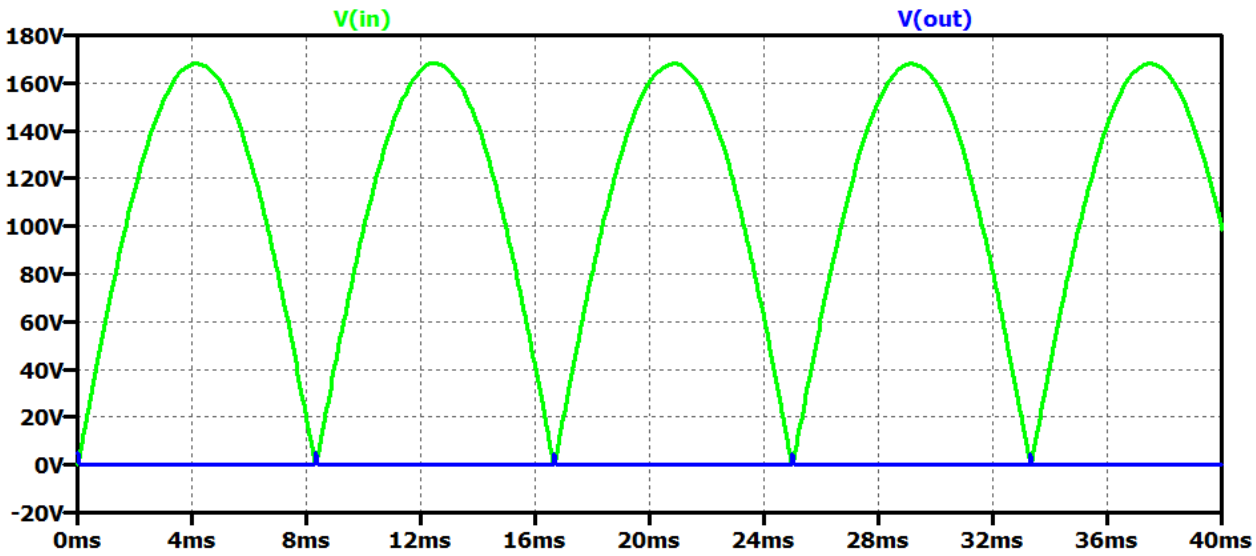

Figure 14: Zero-Crossing Detector Circuit Simulation

The system in Figure 15 was designed to control the voltage that is being applied to the heating element, which in turn controls the temperature. The signal from the microcontroller is passed through the MOC3020 optocoupler, U8. This operates in the same way mentioned previously. Inside the optocoupler there is a LED which sends a signal to the DIAC. This in turn sends a signal to the BTA1 TRIAC gate, U7, and activates it, allowing the current to flow through the heating element, U9. A PWM signal was generated to simulate the output of the microcontroller. The results of this simulation can be seen in Figure 16 below. The input signal is a 120V RMS 60Hz sine wave while the output signal is when the heater has a voltage across it. The delay in the output voltage is caused by the signal sent from the microcontroller. The capacitor C1 is used to control the amount of voltage ripple while the inductor L1 is used to control the current ripple.

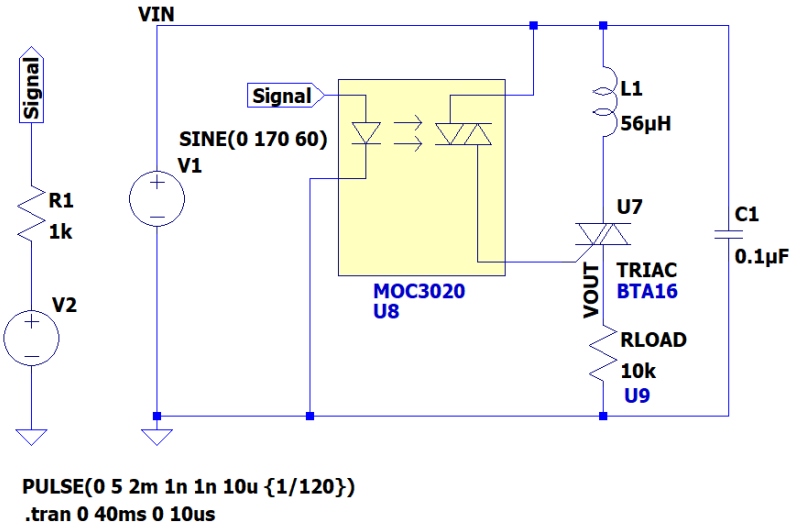

Figure 15: Voltage Control for Heating Element

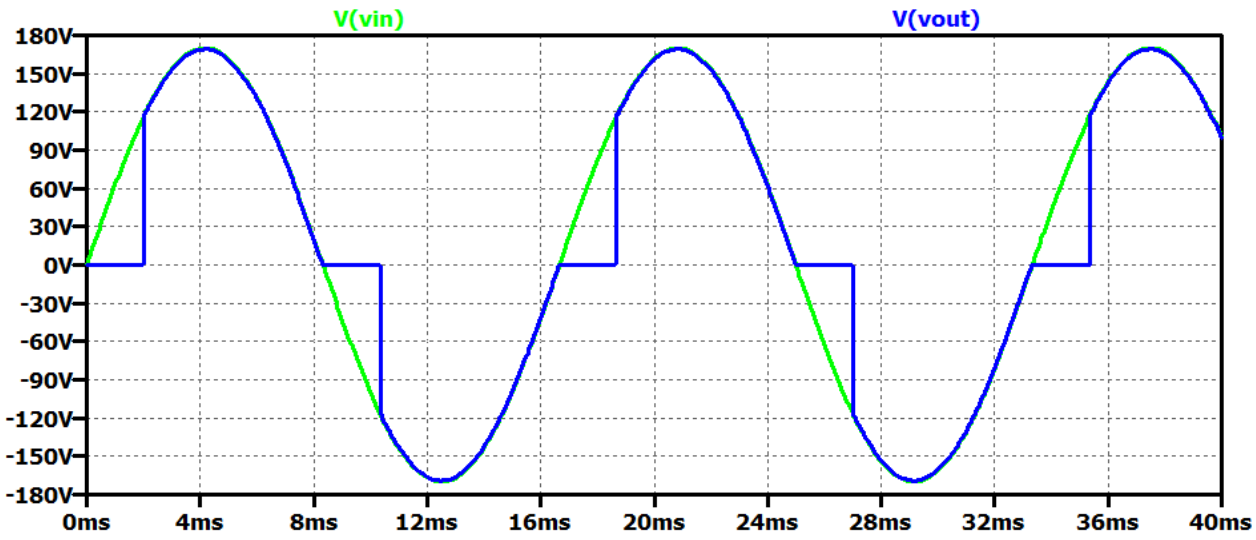

Figure 16: Voltage Control for Heating Element Simulation

(GB, DL)

5.1.6. I2C Motor Driver #108020103 Justification

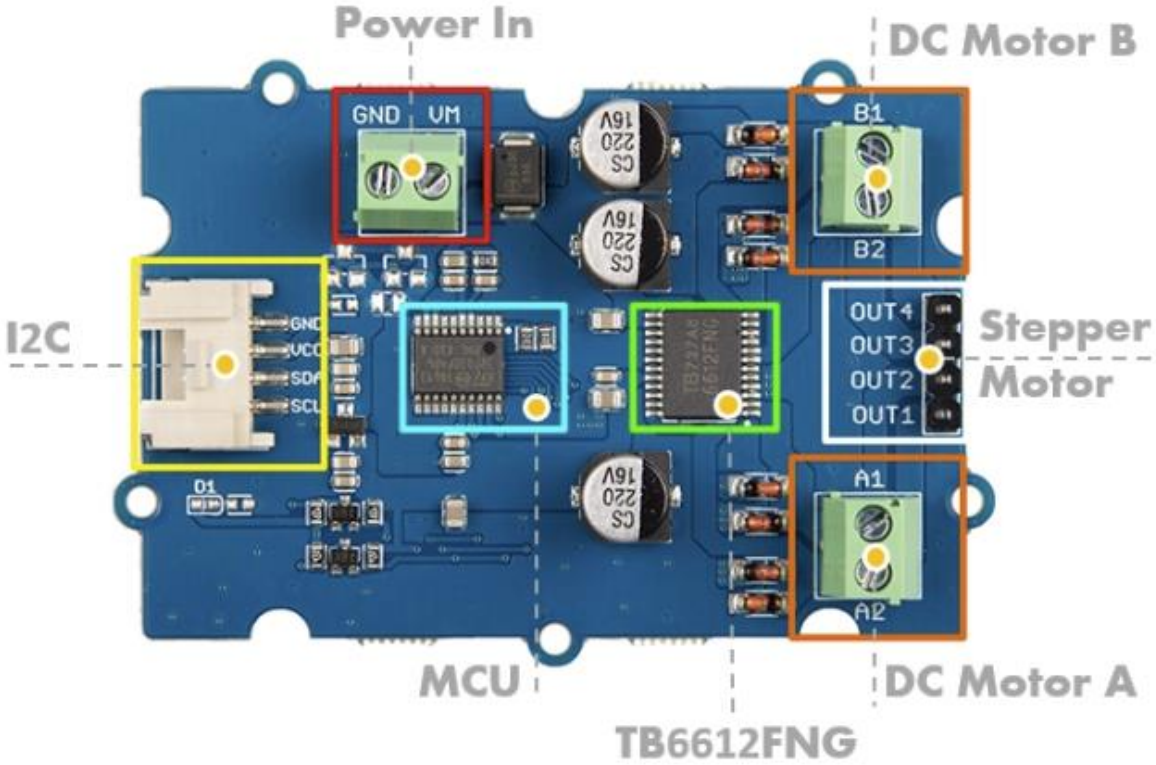

Figure 17: I2C Motor Driver #108020103 Top Down View

This is the motor driver utilized by both tension motors, as well as the spooling motor. This motor driver operates using the I2C communication protocol to communicate with said motors. This motor driver allows us to control the speed of the motors as well.

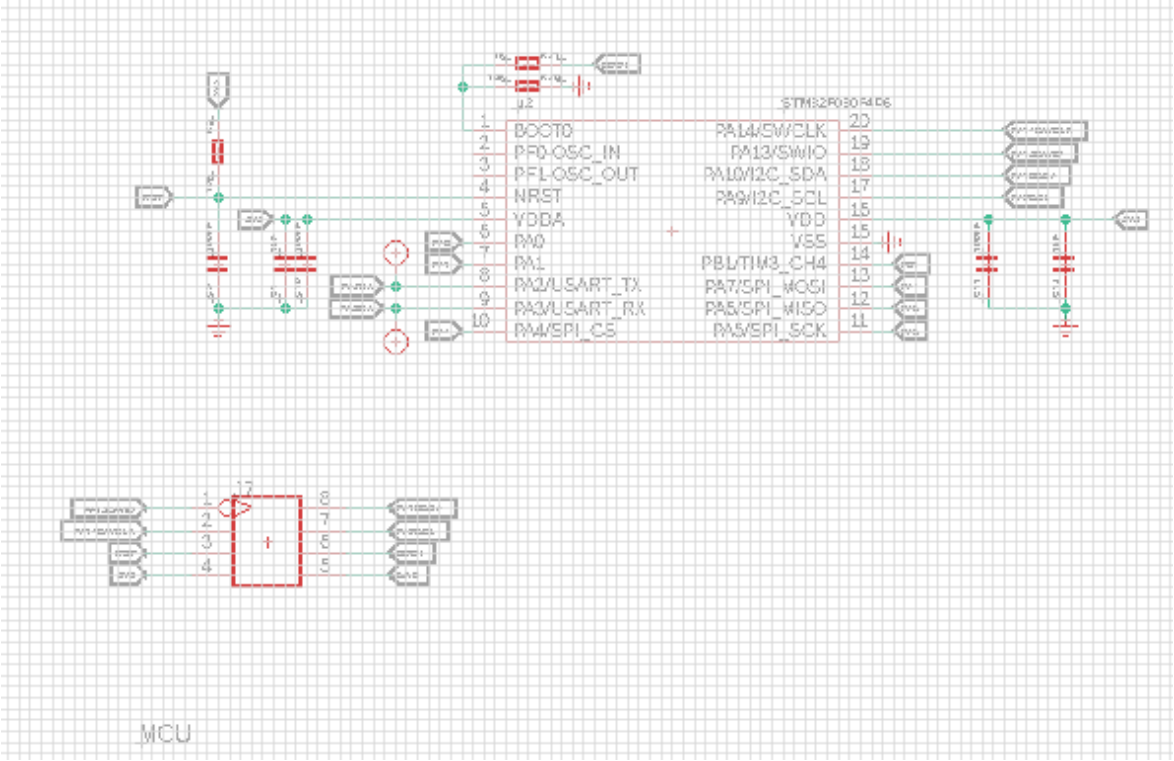

Figure 18: I2C Signal Converter

This diagram shown above is the microcontroller that converts the I2C into the logic signals for the driver chip. This microcontroller, #MCU-STM32F030, does this by advanced multivariable calculations within several transistors. It takes I2C commands, reads the data out of them, and uses this information to set logic lines high and low to the motor driver chip. This controls the state of the motor driver.

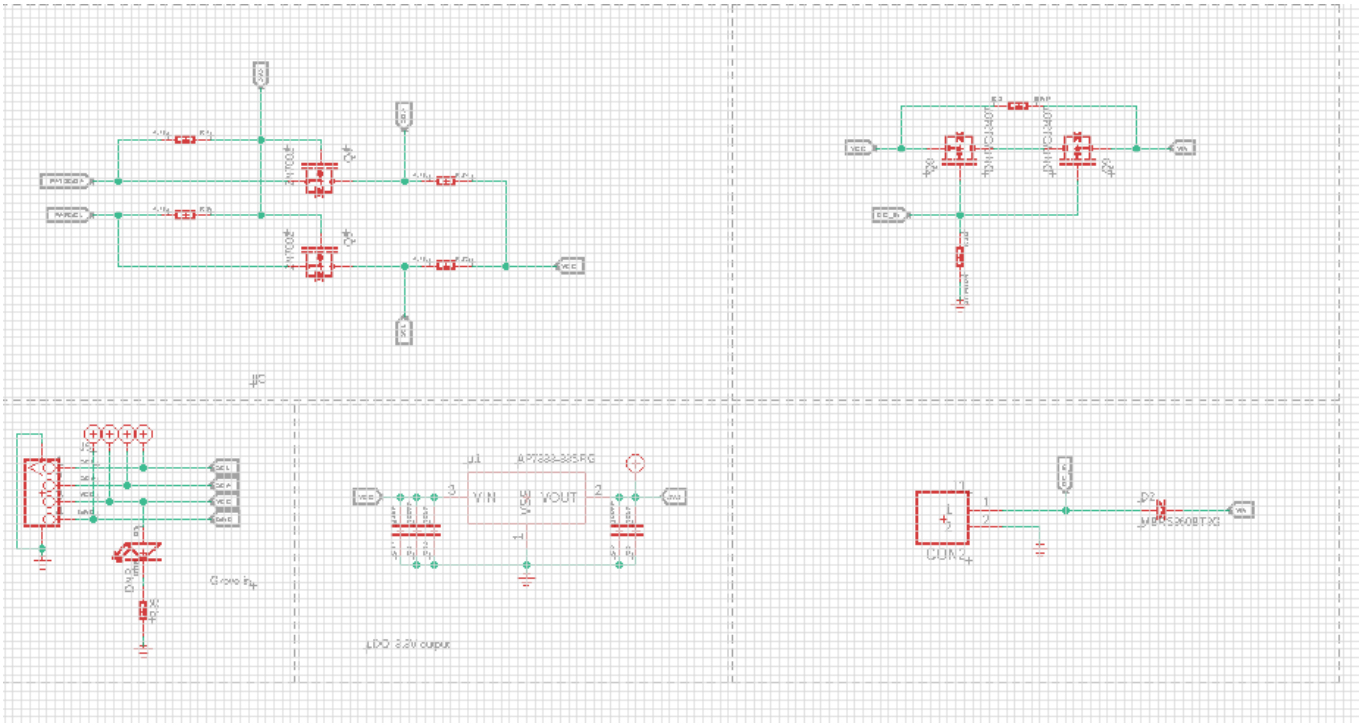

Figure 19: Power & Signal Interpretation (5 subcircuits from top left to bottom right referred to as c1, c2, etc.)

Circuit c1 is a logic level shifter. This converts from 3.3 volts to 5V Vcc for the I2C lines. This is to protect the MCU from overvoltage. Circuit c2 are two transistors that provide the ability to power another device through the I2C connector. Circuit c3 is the I2C connector which has SCL & SDA lines as well as output voltage and ground for optional powering of another device. Circuit c4 is a voltage regulator which regulates from VCC to 3.3 volts with decoupling capacitors to limit noise. Circuit c5 is the connector for input power with a diode to prevent reverse voltage.

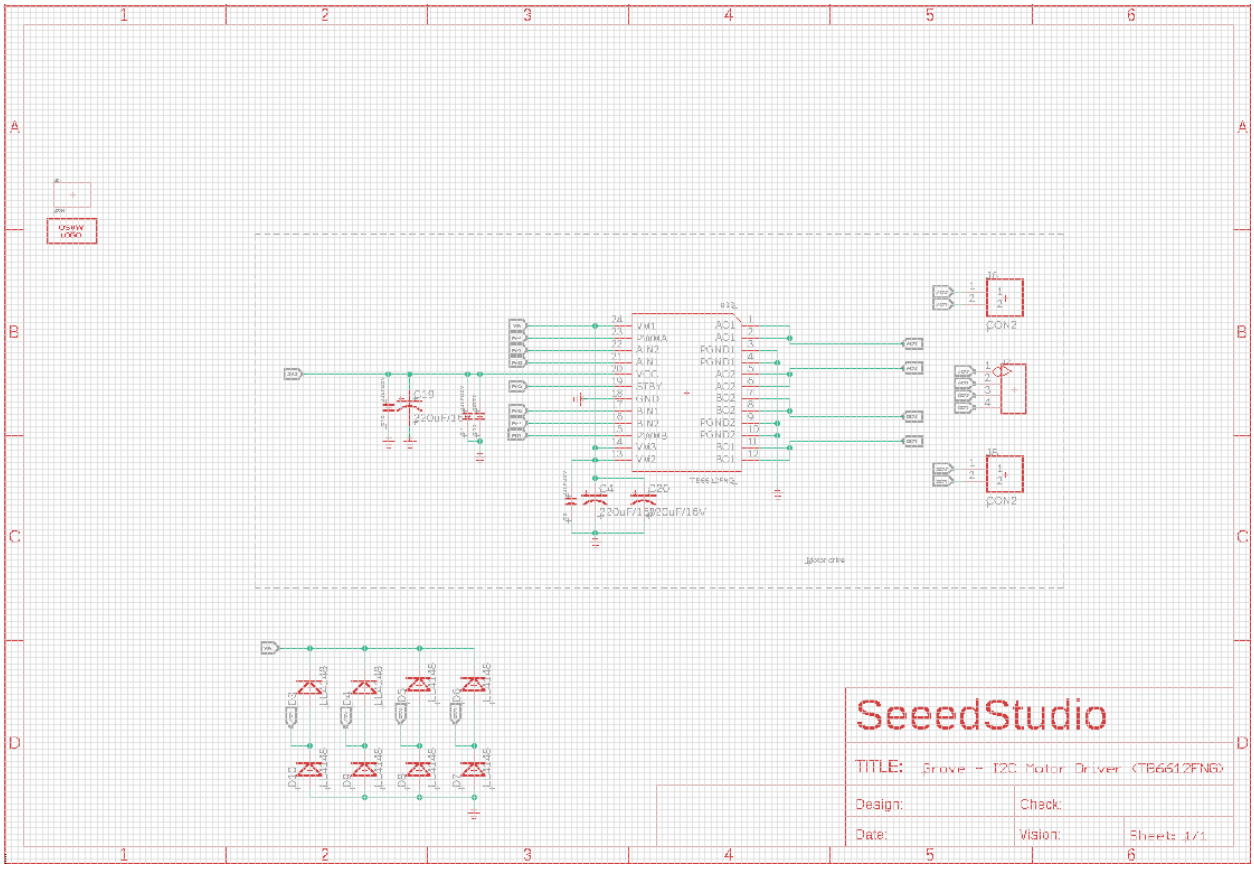

Figure 20: Motor Driver Chip Actual

This motor driver chip, shown above, is #TB6612FNG. This is the actual motor driver chip for this motor driving circuit. This chip contains twin H-bridges to allow for two motors to be driven simultaneously. Each H bridge is driven by the logic signals output by the MCU. Each H bridge operates by 4 transistors and 4 diodes connected in an H configuration. The logic signals will be driven high or low to turn on or off certain transistors which in turn allows the motors speed and direction to be controlled. The output is a positive or negative voltage whose magnitude determines the speed and direction of the motor rotation.

(GB, DL)

5.2. Software Design

This section outlines the software architecture, and defines the inputs, outputs, and responsibilities of each software component.

The software Level 0 block diagram is shown in Figure 21. Input and output for the full system is displayed here. Table 8 provides the accompanying functional requirements.

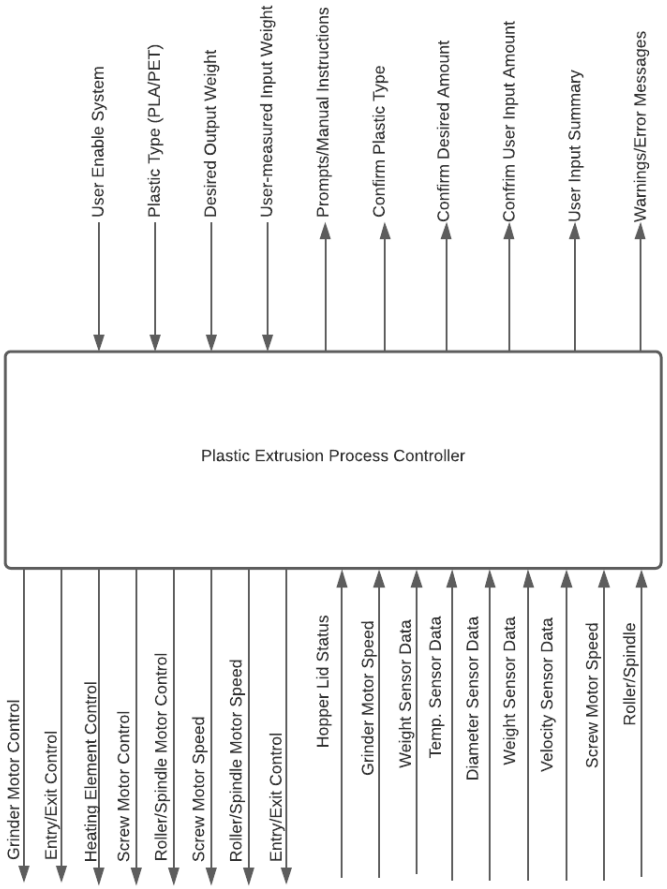

Figure 21: Software Level 0 block diagram

Table 8: Level 0 Software Function Requirements

Module	**Additive Manufacturing Waste Management System – Plastic Extrusion Process Controller**
Inputs	• User enable • Plastic type (PLA/PET) • Desired output amount (kg) • User-measured input amount (kg) • Hopper lid status • Grinder motor speed • Screw motor speed • Roller/spindle motor speed • Temperature sensor data • Diameter sensor data • Velocity sensor data • Weight sensor data
Outputs	• Grinder motor control • Exit/Entry Controls

	• Heating element control • Screw motor control • Roller/spindle motor control • Screw motor speed • Roller/spindle motor speed
Functionality	Control heating element and motors using sensor data and user input to successfully extrude plastic filament.

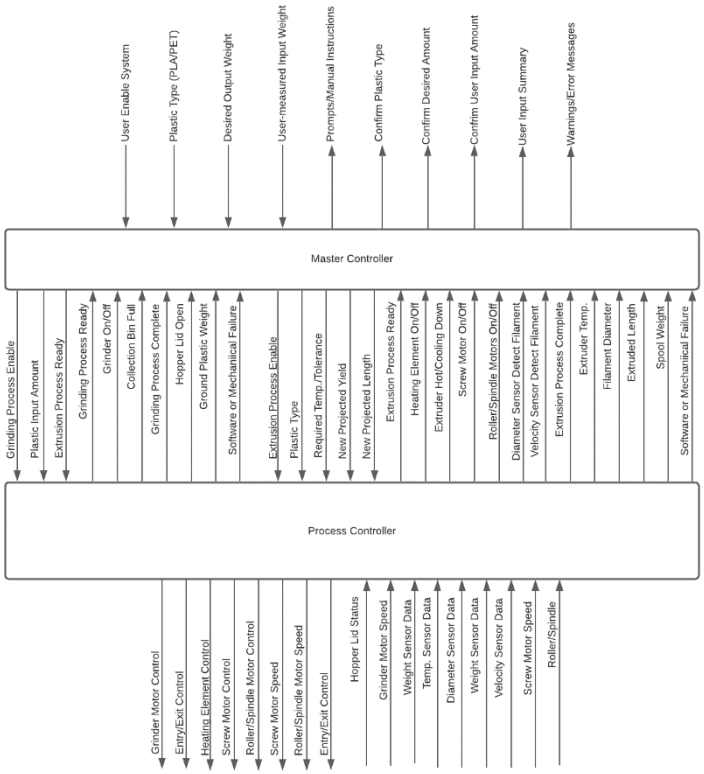

Figure 22: Software Level 1 block diagram

Table 9: Level 1 Software Function Requirements

Module	**Master Controller/UI**
Inputs	• Plastic type (PLA or PET) • Desired output amount (kg) • User-measured input amount (kg) • Grinder ready • Grinder status • Collection bin status • Hopper lid status • Ground plastic weight • Software or mechanical failure notification • Extruder ready • Extruder status

	• Heating element status • Screw status • Roller/spindle status • Diameter sensor detect filament (Y/N) • Velocity sensor detect filament (Y/N) • Filament diameter • Extruded length • Spool weight
Outputs	• Prompts for user input and manual extrusion steps • Confirm plastic type • Confirm plastic input amount • User input summary • Statuses, warnings, error messages • Grinder ready • Grinder enable • Plastic input amount • Extruder ready • Extruder enable • Plastic type • Required temperature and tolerance • New projected yield (kg) • New projected length (m)
Functionality	Accept all user input and make this information available to both the user and the system. Provide the user with instructions and error messages necessary to operation.
Module	**Process Controller**
Inputs	• User input summary • Grinder enable • Extruder enable • New projected yield (kg) • New projected length (m) • Grinder collection bin weight sensor data • Temperature sensor data • Diameter sensor data • Spool weight sensor data • Diameter sensor detect filament (Y/N) • Velocity sensor detect filament (Y/N) • Screw motor speed • Roller/spindle motor speed
Outputs	• Grinder motor control • Heating element control • Screw motor control • Roller/spindle motor control • Entry/exit control

	• Entry/exit status • Grinder motor speed • Screw motor speed • Roller/spindle motor speed • Temperature sensor data • Diameter sensor data • Velocity sensor data • Weight sensor data • Diameter sensor detect filament (Y/N) • Velocity sensor detect filament (Y/N) • Grinder stage entry/exit point control • Software or mechanical failure notification • Weight sensor data
Functionality	Facilitate grinding of input material to required pellet size. Read weight sensor and lid safety signals and control grinding motor accordingly.

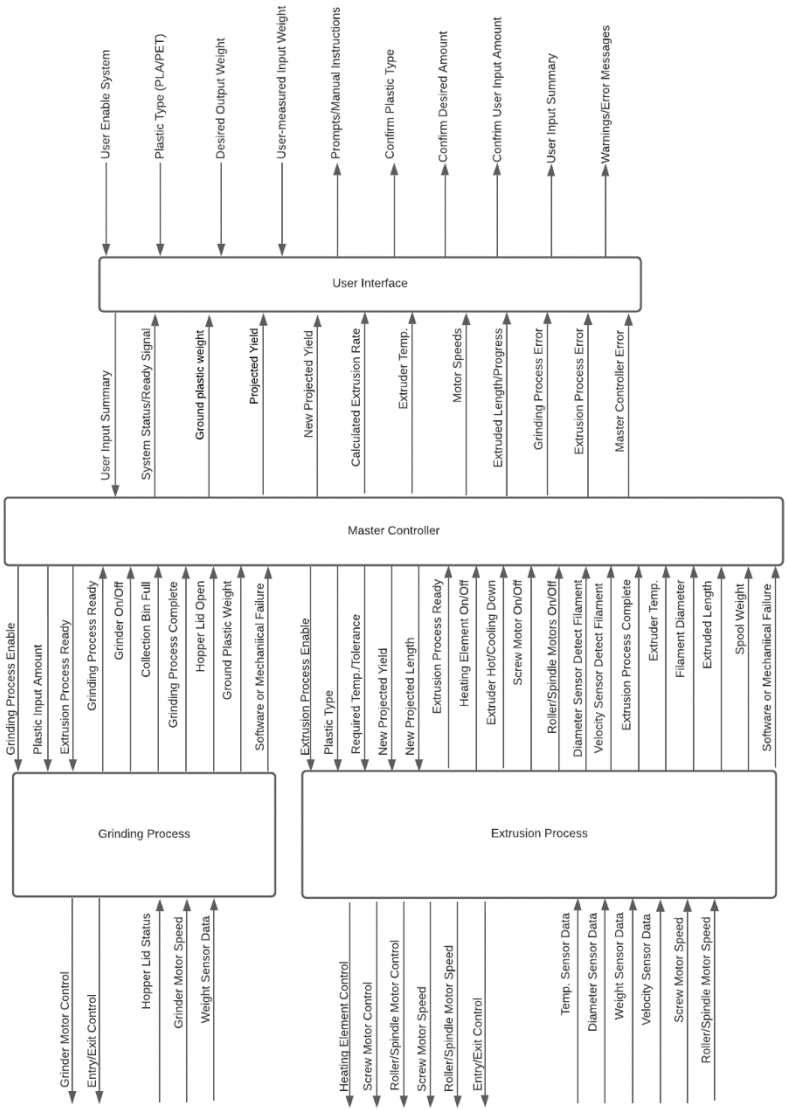

Figure 23: Software Level 2 block diagram.

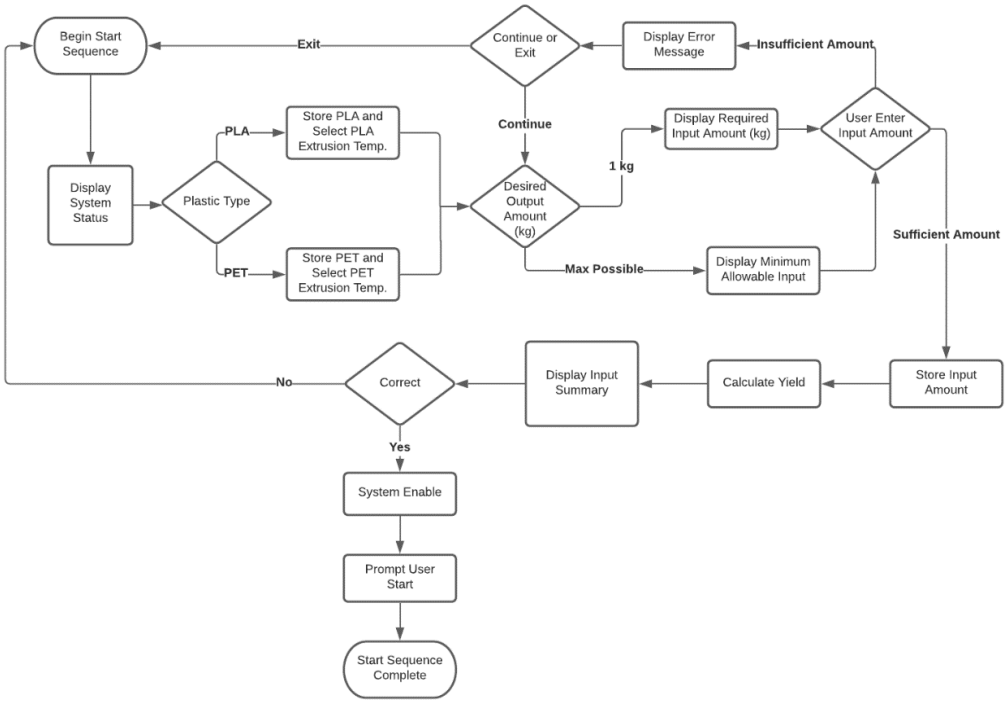

Figure 24: Software Level 2 Flowchart - User Interface Start Sequence

Pseudo Code 1: User Interface Start Sequence

```
// Level 2 User Interface Start Sequence Pseudo Code
// DT16

// Last Revised 10.6.21 LL


void displaySystemStatus()
{
    cout ('System Status: Start Sequence');
}

int userInputPlasticType()
{
    cout('What is input plastic type?')
    cin (str)
    if str == PLA
    {
        Extemp = PLAtemp
    }
    else if str == PET
    {
        Extemp = PLAtemp
    }
return  Extemp
```

```
}

int userInputDesiredWeight()
{
    cout ('The max allowable end weight is 1kg.\n')
    cout ('The min allowable end weight is 200g.\n')

    cout ('What is desired final filament weight?')
    cin (finalWeight)
    while finalWeight < minWeight || finalWeight > maxWeight
    {
        cout('Invalid Weight Entered. Please enter valid input')
        cin(finalWeight)
    }
    initialWeight = finalWeight/yield
    cout ('Inital input weight: ', initialWeight)
    return initialWeight
}

bool inputsAccurate
{
    cout ('Please review the below inputs.\n')
    cout ('Enter Y if all correct or N if incorrect')
    cin (correct)

    if correct == 'Y'
    {
        cout ('System ready to start - Press START when ready')
        return true
    }
    else
    {
        return false
    }
}

void systemEnable();

void systemStart()
{
    cout ('Start System? (Y/N)')
    cin (start)
    if start == y
        startSystem();
    else
        systemWait;
}
void startSystem();

void systemWait();
```

```
int main
{
    displaySystemStatus();
    inputsAcc = false;
    while inputsAcc == false
    {
        type = userInputPlasticType();
        initialWeight = userInputDesiredWeight();
        finalWeight = initialWeight * yield;
        inputsAcc = inputsAccurate();
    }
    systemEnable();
    systemStart();
    return 0;
}
```

The display is implemented using an inexpensive resistive touchscreen that is driven by an Arduino Uno (ATmega328P chip). The responsibilities of the Arduino are limited to generating graphics for the display and interpreting touchscreen input. The display itself is connected to the Arduino Uno board via SPI, and the Arduino connects to the master controller (PIC32MX470) via I^2C. The I^2C 2 module on the master controller is dedicated to display interaction, leaving the I^2C 1 module free to communicate with the I^2C motor controllers. The main loop of the Arduino display controller is shown below, and a complete listing of the display code is provided in the Appendix. The Arduino is not responsible for the flow of control of the overall software system and does not handle any signal processing. Additionally, the code for the Arduino is written to be as C/C++ compatible as possible, aside from the Arduino-specific graphics and touchscreen libraries, and implementation-specific functions for communication protocols that have equivalent implementations in C/C++.

```
void setup()
{
    Wire.begin(0x14);

    /*Event Handlers*/
    Wire.onReceive(I2C_receive_event);
    Wire.onRequest(I2C_request_event);
    Serial.begin(9600);
    byte complete = 0x07;
    UI.tft.begin();
    UI.tft.setRotation(1);
    UI.set_numeric_input_screen(UI.numeric_params, UI.desired_yield.ID);
    UI.get_numeric_user_input(UI.numeric_params, UI.desired_yield.ID);
    UI.required_input.VALUE = UI.desired_yield.VALUE * 1.11;

    UI.set_output_screen();
}

void loop()
{
    if(received_message == true)
    {
        if (is_status == true)
        {
            UI.direct_I2C_Status_Param(i2c_int_data.incoming_ID, i2c_int_data.incoming_status);
        }
        else
        {
            UI.direct_I2C_Numeric_Param(i2c_int_data.incoming_ID, i2c_int_data.incoming_value);
        }
        i2c_int_data.incoming_ID = 0;
        i2c_int_data.incoming_status = 0;
        i2c_int_data.incoming_value = 0;
        received_message = false;
        is_status = false;
    }

    UI.poll_inputs(UI.numeric_params, NUMERIC_PARAM_COUNT);
    UI.poll_inputs(UI.status_params, STATUS_PARAM_COUNT);
}
```

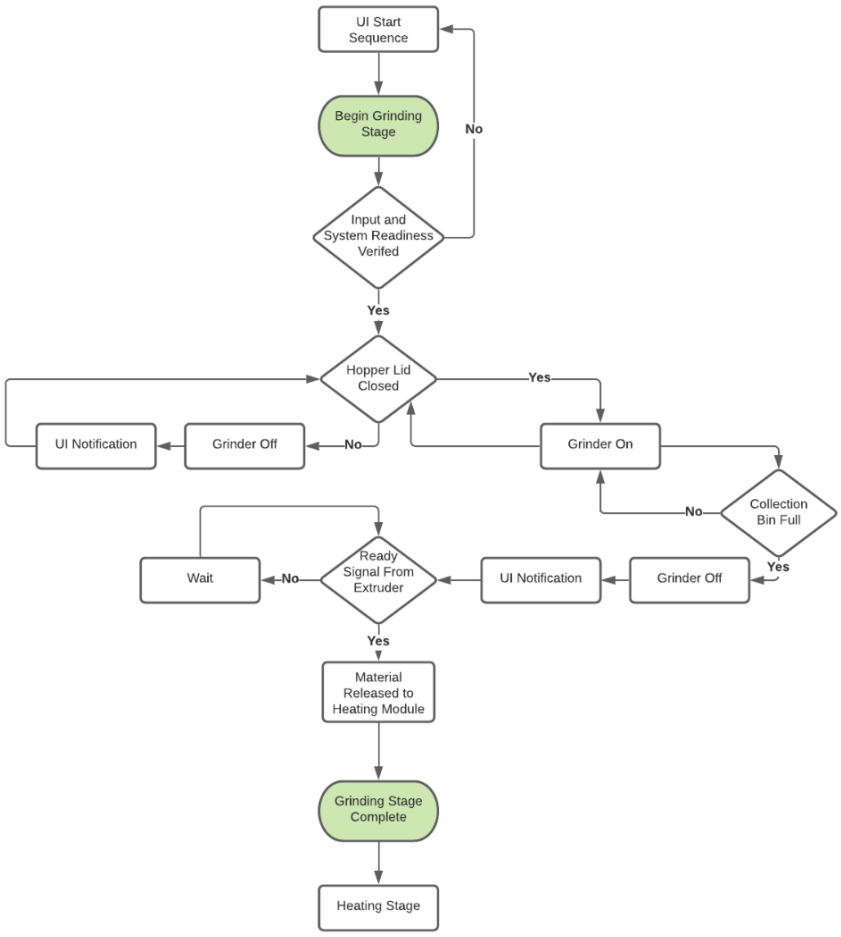

Figure 25: Software Level 2 flowchart - Grinding Stage.

Pseudo Code 2: Grinding Stage

```
// Level 2 Grinding Pseudo Code
// DT16

// Last Revised 10.6.21 LL

void grindingSystemReady()
{
    if status == grStart
    {
        grindingStart();
    }
    else
    {
        systemWait();
    }
}
```

```
void grindingStart();
void grindingStop();

void systemWait();
void systemMove();

bool getExStatus()
{
    if exStatus == true
    {
        status = true
    }
    else
    {
        status = false
    }
    return status
}
bool hopperReady()
{
    if lidClosed == true
    {
        ready = true

    }
    else
    {
        cout('Hopper Lid Not Closed. Please close to start')
        ready = false
    }
    return ready
}

bool catchHopper();
{
    if weight <= inputWeight
    {
        ready = true
    }
    else
    {
        ready = false
    }
    return ready
}

int main()
{
    status = grStop();
    while hopper = true;
    {
```

```
        hopper = hopperReady();
        status = grindingStart();

        while catcher = true;
        {
            catcher = catchHopper()
            status = grindingStart();
        }
        status = grindingStop();
    }
    if catcher == false
    {
        exStatus = getExStatus();
        if exStatus == true
        {
            systemMove();
        }
        else
        {
            systemWait();
        }
    }
    return 0;
}
```

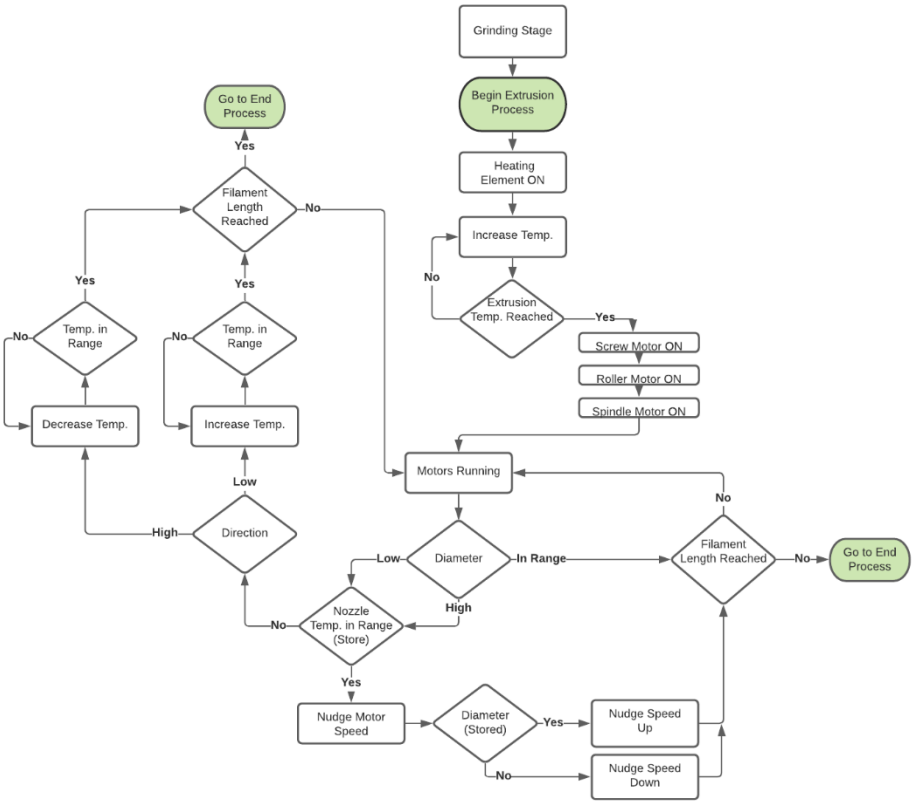

Figure 26: Software Level 2 flowchart – Extrusion Stage.

Pseudo Code 3: Extrusion Stage

```
// Level 2 Extruding Pseudo Code
// DT16

// Last Revised 10.6.21 LL

void heatStatus()
{
    while heat == false
    {
        heat = tempMet();
    }
}

void screwOn();
void rollerOn();
void spindleOn();

bool tempMet
{
    temp1 = tempSensor1();
    temp2 = tempSensor2();
    temp3 = tempSensor3();

    if temp1 && temp2 && temp3 within range
    {
        heat1 = off;
        heat2 = off;
        heat3 = off;
        status = true;
    }
    else
    {
        heat1 = on;
        heat2 = on;
        heat3 = on;
        status = false;
    }
    return status;
}

bool weightReached()
{
    weight = spoolWeightSensor;
    if weight < finalweight
    {
        status = false
    }
    else
    {
```

```
        status = true
    }
    return status
}

void diameterCheck()
{
    diameter = diameterSensor;
    if diameter < target
    {
        increaseScrewSpeed
        decreaseTempRanges
    }
    else if diameter > target
    {
        decreaseScrewSpeed
        increaseTempRanges
    }
}

int main()
{
   while systemMove == false
   {
        systemMove = weightReached();
        heatStatus();
        screwOn();
        rollerOn();
        spindleOn();
        diameterCheck();
   }
   return 0;
}
```

The software system is implemented on the PIC32MX470F512H using the FreeRTOS real-time operating system. In addition to the three threads (master, preparation, and extrusion) described previously, the extrusion thread is further divided into two more specified threads: extrusion input and extrusion control. Extrusion input is devoted solely to receiving and pre-processing sensor input, while extrusion control is dedicated to the implementation of feedback control of the heating element and motors using the pre-processed data provided by the extrusion input thread. The master, extrusion input, and extrusion control state machines are shown below. These state machines execute repeatedly in a pseudo-concurrent fashion for the duration of the device operation. Top level control flow for all threads is handled by the master thread, which is also responsible for passing data to and from the display. The DataManager class is a unique entity within the system, whose sole instantiation is defined globally. This object is responsible for tracking all sensor and actuator data. When sensor input is received and actuator states are changed, the resulting values are passed to the DataManager object (globalDataManager) which updates the values and makes these updates available to the entire system. The full listing for the master controller state machines and source files is provided in the Appendix.

Code Snippet: Master process state machine.

```
28  void MASTER_Tasks ( void )
29  {
30      switch ( masterData.state )
31      {
32          case MASTER_STATE_INIT:
33          {
34
35              I2C_2_Init();
36              bool appInitialized = true;
37
38              if (appInitialized)
39              {
40                  masterData.state = MASTER_STATE_SERVICE_TASKS;
41              }
42              break;
43          }
44
45          case MASTER_STATE_SERVICE_TASKS:
46          {
47              /*
48               * TO DO: Implement process control flow state machine
49               */
50
51              globalDataManager.pollNumericParams();
52              globalDataManager.sendAllFreshNumericParams();
53              CORETIMER_DelayUs(50);
54
55              break;
56          }
57
58          default:
59          {
60              break;
61          }
62      }
63  }
64
```

Code Snippet: Extrusion Input process state machine.

```
34  void EXTRUSION_INPUT_Tasks ( void )
35  {
36      switch ( extrusion_inputData.state )
37      {
38          case EXTRUSION_INPUT_STATE_INIT:
39          {
40              SPI_Init();
41              CORETIMER_DelayUs(50);
42              bool appInitialized = true;
43
44
45              if (appInitialized)
46              {
47                  extrusion_inputData.state = EXTRUSION_INPUT_STATE_SERVICE_TASKS;
48              }
49              break;
50          }
51
52          case EXTRUSION_INPUT_STATE_SERVICE_TASKS:
53          {
54              temp_1_float = tempSensor1.readTemp();
55              globalDataManager.setNumericParam(ZONE_1_TEMP_INDEX, temp_1_float);
56              temp_1_float = 0;
57              CORETIMER_DelayUs(10);
58
59              temp_2_float = tempSensor2.readTemp();
60              globalDataManager.setNumericParam(ZONE_2_TEMP_INDEX, temp_2_float);
61              temp_2_float = 0;
62              CORETIMER_DelayUs(10);
63
64              temp_3_float = tempSensor3.readTemp();
65              globalDataManager.setNumericParam(ZONE_3_TEMP_INDEX, temp_3_float);
66              temp_3_float = 0;
67              CORETIMER_DelayMs(500);
68
69              break;
70          }
71
72          default:
73          {
74              break;
75          }
76      }
77  }
```

Code Snippet: Extrusion Control process state machine.

```
void EXTRUSION_CONTROL_Tasks ( void )
{
    switch ( extrusion_controlData.state )
    {
        case EXTRUSION_CONTROL_STATE_INIT:
        {
            I2C_1_Init();
            CORETIMER_DelayMs(500);

            while(I2C_1_IS_BUSY);
            controller1.setMotorSpeed(MOTOR_1_ID, 100, -1);
            controller1.setMotorSpeed(MOTOR_2_ID, 100, 1);
            current_motor_speed = controller1.getMotorSpeed(0);
            current_motor_speed = (current_motor_speed / 255) * 100;
            globalDataManager.setNumericParam(ROLLER_SPEED_INDEX, current_motor_speed);

            while(I2C_1_IS_BUSY);
            controller2.setMotorSpeed(MOTOR_1_ID, 80, -1);
            current_motor_speed = controller2.getMotorSpeed(0);
            current_motor_speed = (current_motor_speed / 255) * 100;
            globalDataManager.setNumericParam(SPOOLER_SPEED_INDEX, current_motor_speed);

            bool appInitialized = true;

            if (appInitialized)
            {
                extrusion_controlData.state = EXTRUSION_CONTROL_STATE_SERVICE_TASKS;
            }
            break;
        }

        case EXTRUSION_CONTROL_STATE_SERVICE_TASKS:
        {
            current_temp_1 = globalDataManager.getNumericParam(ZONE_1_TEMP_INDEX);
            current_temp_2 = globalDataManager.getNumericParam(ZONE_2_TEMP_INDEX);
            current_temp_3 = globalDataManager.getNumericParam(ZONE_3_TEMP_INDEX);
            current_diameter = globalDataManager.getNumericParam(DIAMETER_INDEX);

            /*
             * TO DO: Implement feedback control
             */

        }

        default:
        {
            break;
        }
    }
}
```

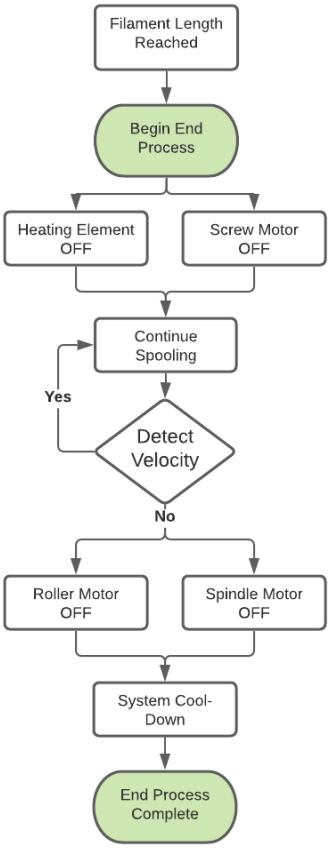

Figure 27: Software end sequence flowchart

Pseudo Code 4: End Sequence

```
// Level 2 End Sequence Pseudo Code
// DT16

// Last Revised 10.6.21 LL

int main()
{
   while finalWeight = true
   {
      heat = off
      screw = off
      roller = off
      spindle = off
   }
   systemStatus = End
}
```

Table 10: Level 2 Software Function Requirements

Module	**User Interface**

Inputs	**From User** • Plastic type (PLA or PET) • Desired output amount (kg) • Plastic input amount (kg) • User system enable **From Master Controller** • System status/ready signal • Ground plastic weight • Projected yield (from user input) • New projected yield (from ground plastic weight) • Calculated extrusion rate • Extruder temperature • Motor speeds • Extruded length/progress • Grinding Process error notification • Extrusion Process error notification • Master Controller error notification

Outputs	**To User**
	• **Originating from UI Logic**
	  ○ Prompts for input and manual extrusion steps
	  ○ Confirm plastic type (PLA/PET)
	  ○ Confirm desired output amount (kg)
	  ○ Confirm plastic input amount (kg)
	  ○ Warnings, error messages
	  ○ User input summary
	• **Originating from Master Controller**
	  ○ Projected yield (kg)
	  ○ Projected spool length (m)
	  ○ Extruder temperature
	  ○ Motor speeds
	  ○ Extruded length/progress
	  ○ Warnings, error messages
	To Master Controller
	• User input summary
	  ○ Plastic type (PLA/PET)
	  ○ Plastic input amount (kg)
	  ○ System enable

Functionality	Accept all user input, provide visual feedback for that input, and pass necessary information to the Master Controller. Receive process notifications and monitoring data provided by the Master Controller.
Module	**Grinding Process**

Inputs	**From Master Controller** • Grinding process enable • Plastic input amount • Extrusion Process ready signal **From sensors/actuators/hardware** • Hopper lid status (open/closed) • Grinder motor speed • Weight sensor data

Outputs	**To Master Controller**
	• Grinding Process status
	  o Ready
	  o Grinder motor on/off
	  o Collection bin full
	  o Complete
	• Hopper lid open
	• Ground plastic weight (kg)
	• Software or mechanical failure
	To actuators/hardware
	• Grinder motor control
	• Entry/Exit Control

Functionality	Read weight sensor and lid safety signals, and control grinder motor. Send and receive process information and control signals to/from the Master Controller.
Module	**Extrusion Process**
Inputs	**From Master Controller** • Extrusion Process enable • Plastic type (PLA/PET) • Required temperature and tolerance • New projected yield (kg) (from post-grinding weight) • New projected spool length (m) (from post-grinding weight) **From sensors/actuators/hardware** • Temperature sensor data • Diameter sensor data • Weight sensor data • Velocity sensor data (length) • Screw motor speed • Roller and spindle motor speed

Outputs	**To Master Controller**
	• Extrusion Process Status
	  o Ready
	  o Heating element on/off
	  o Extruder hot/cooling down
	  o Screw motor on/off
	  o Roller and spindle motors on/off

	○ Diameter sensor detect filament (Y/N) ○ Velocity sensor detect filament (Y/N) ○ Complete • Quality/Progress Monitoring ○ Extruder temperature ○ Filament diameter ○ Extruded length ○ Spool weight • Software or mechanical failure **To actuators/hardware** • Heating element control • Screw motor control • Roller and spindle motor control • Screw motor speed • Roller and spindle motor speed • Entry/Exit Control
Functionality	Read temperature, diameter, weight, and velocity sensor signals and control heating element as well as screw, roller, and spindle motors. Send and receive process information and control signals to/from the Master Controller.
Module	**Master Controller**
Inputs	**From User Interface** • User input summary ○ Plastic type (PLA/PET) ○ Plastic input amount (kg) ○ System enable **From Grinding Process** • Grinding Process status ○ Ready ○ Grinder motor on/off ○ Collection bin full ○ Complete • Hopper lid open • Ground plastic weight (kg) • Grinding process software or mechanical failure **From Extrusion Process** • Extrusion Process Status ○ Ready ○ Heating element on/off ○ Extruder hot/cooling down ○ Screw motor on/off ○ Roller and spindle motors on/off ○ Diameter sensor detect filament (Y/N) ○ Velocity sensor detect filament (Y/N) ○ Complete

	• Quality/Progress Monitoring
	  ○ Extruder temperature
	  ○ Filament diameter
	  ○ Extruded length
	  ○ Spool weight
	• Extrusion Process software or mechanical failure

Outputs	**To User Interface** • Ground plastic weight • Projected yield (from user input) • New projected yield (from ground plastic weight) • Calculated extrusion rate • System status/ready signal • Extruder temperature • Motor speeds • Extruded length/progress • Grinding Process error notification • Extrusion Process error notification • Master Controller error notification **To Grinding Process** • Grinding process enable • Plastic input amount • Extrusion Process ready signal **To Extrusion Process** • Extrusion Process enable • Plastic type (PLA/PET) • Required temperature and tolerance • New projected yield (kg) (from post-grinding weight) • New projected spool length (m) (from post-grinding weight)
Functionality	Coordinate the execution of UI Start Sequence, Grinding Process, Extrusion Process and End Sequence. Initiate and monitor processes and facilitate physical interaction and communication between UI and process.

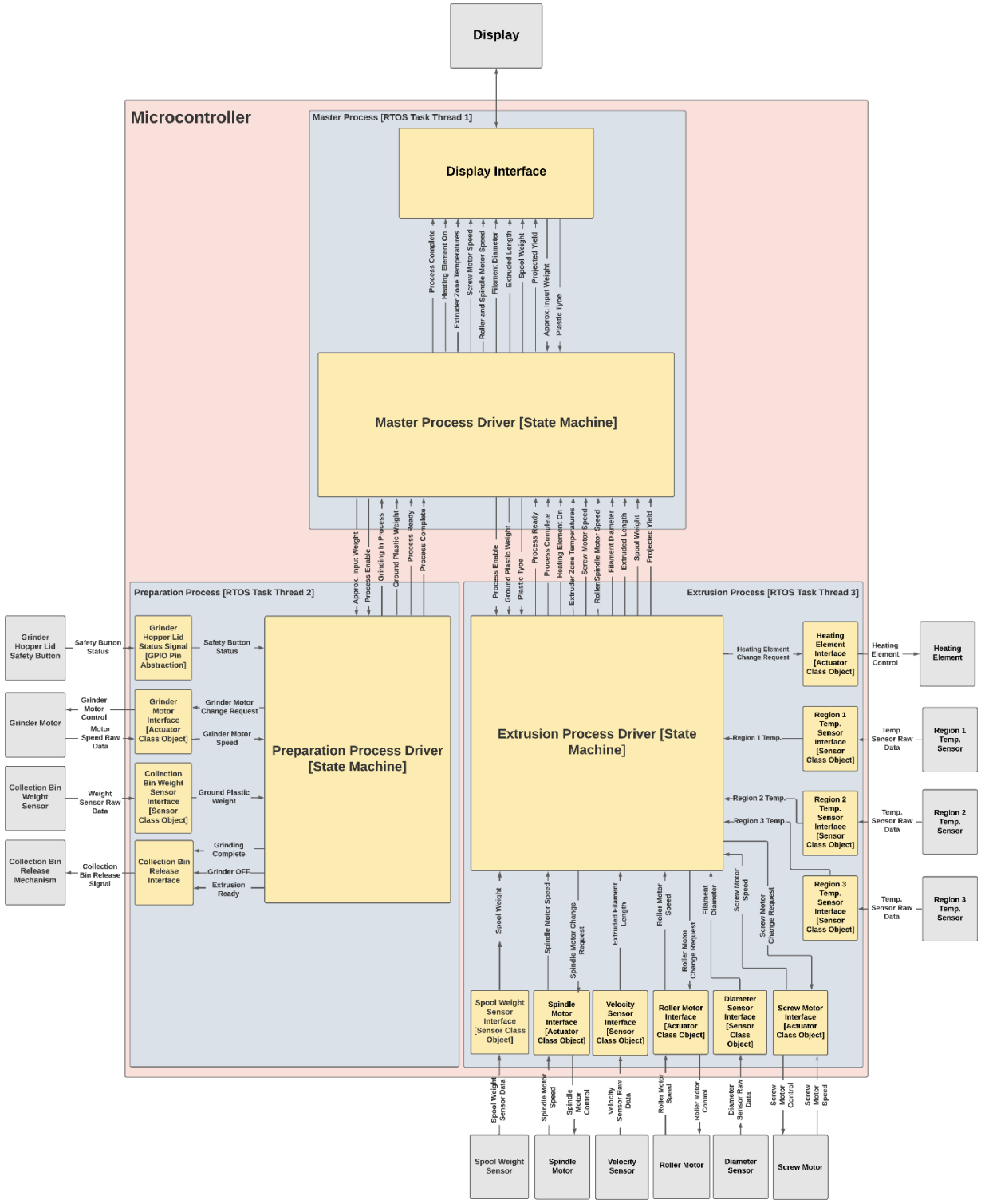

Figure 28: Level 3 software block diagram

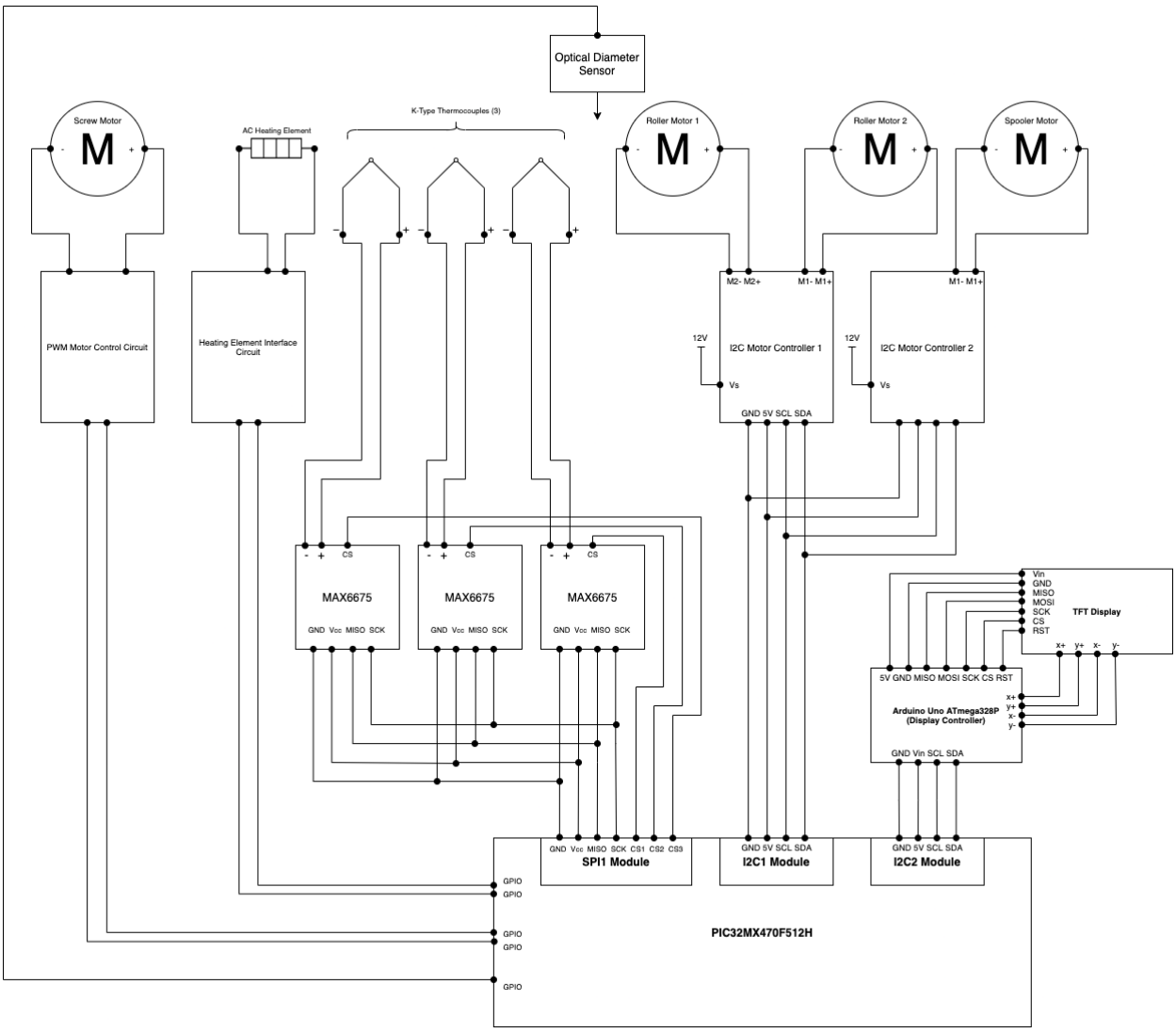

Figure 29: SPI, I2C and Pic32 Connection Schematic

(LL, WW)

6. Mechanical Sketch

This section outlines the basic mechanical sketches and models of the design. The numerical dimensions shown are for ratios only and are unitless.

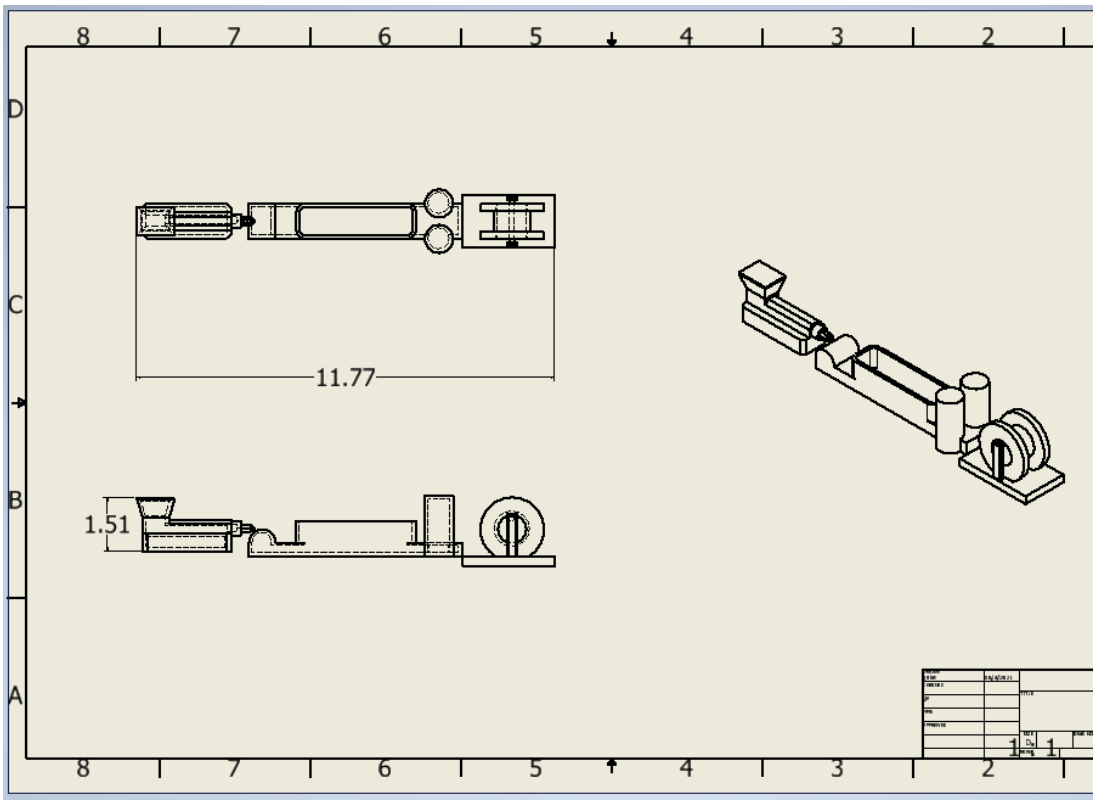

Figure 30: Mechanical Sketch - Extruder w/ Cooling Tank and Spooler

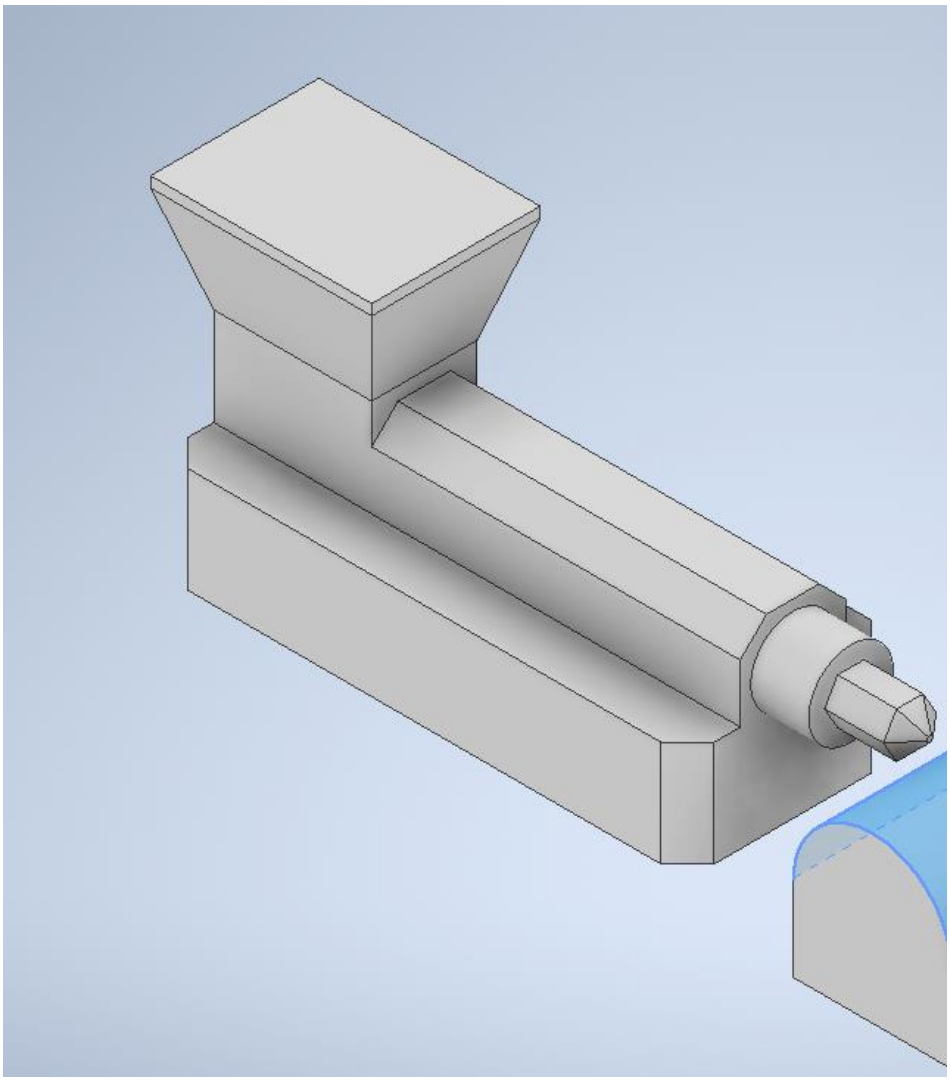

Figure 31: Mechanical Sketch - Extruder rendering. Isometric view.

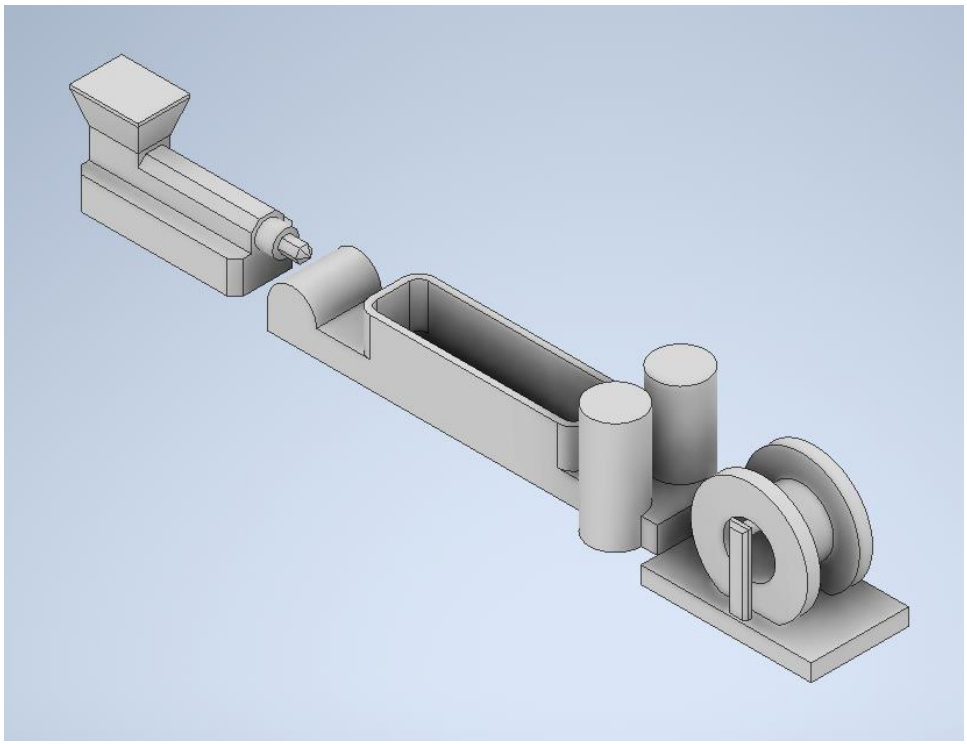

Figure 32: Mechanical Sketch - Overall System rendering. Isometric View

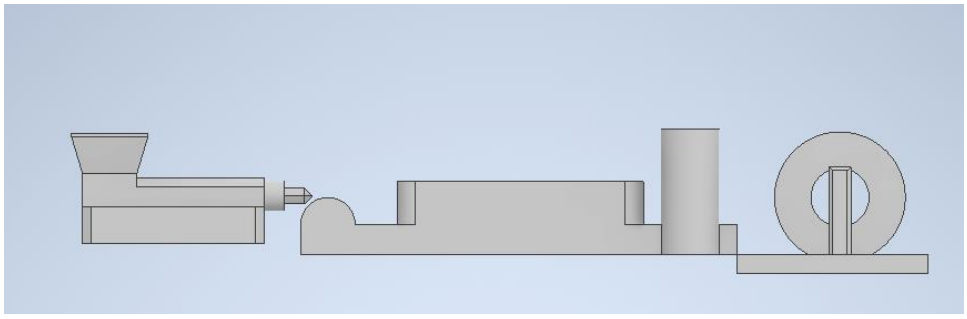

Figure 33: Mechanical Sketch - Overall System rendering. Side view

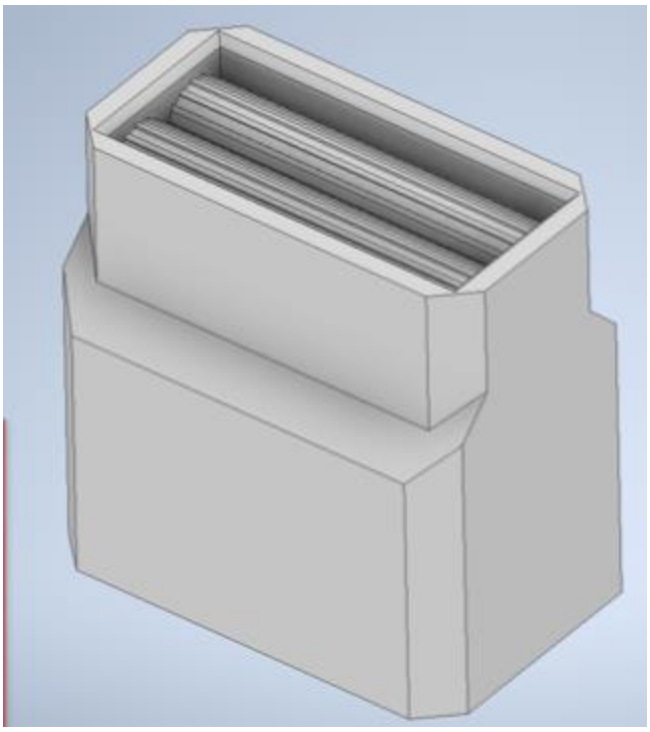

Figure 34: Mechanical Sketch - Grinder rendering. Isometric view.

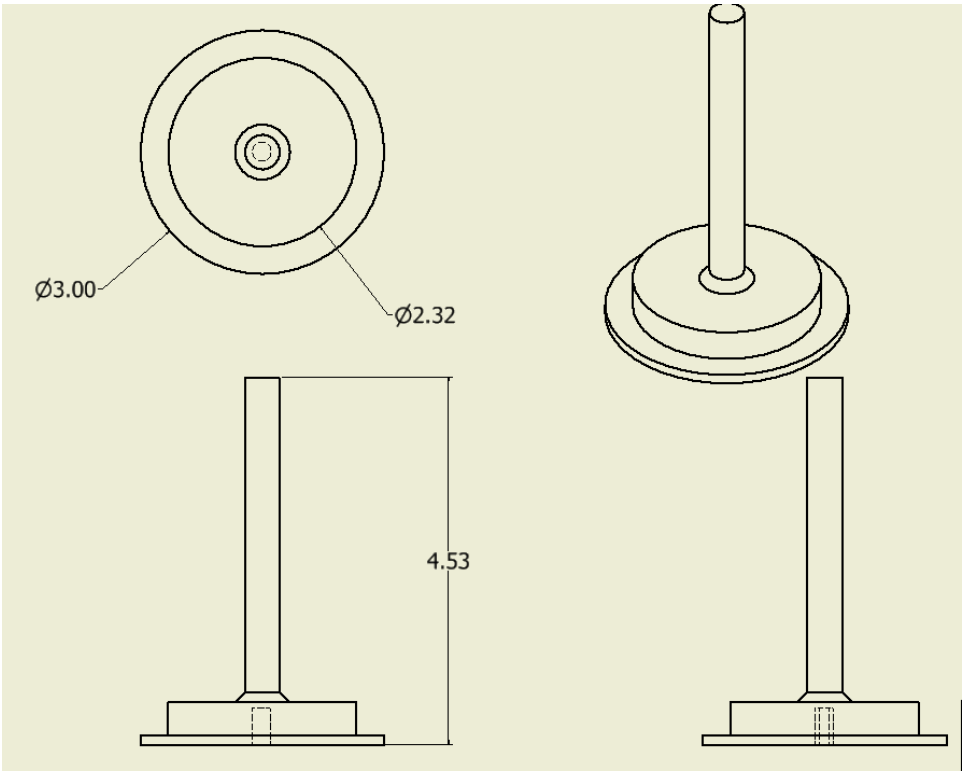

Figure 35: Mechanical Sketch – Spooler Connecting Rod

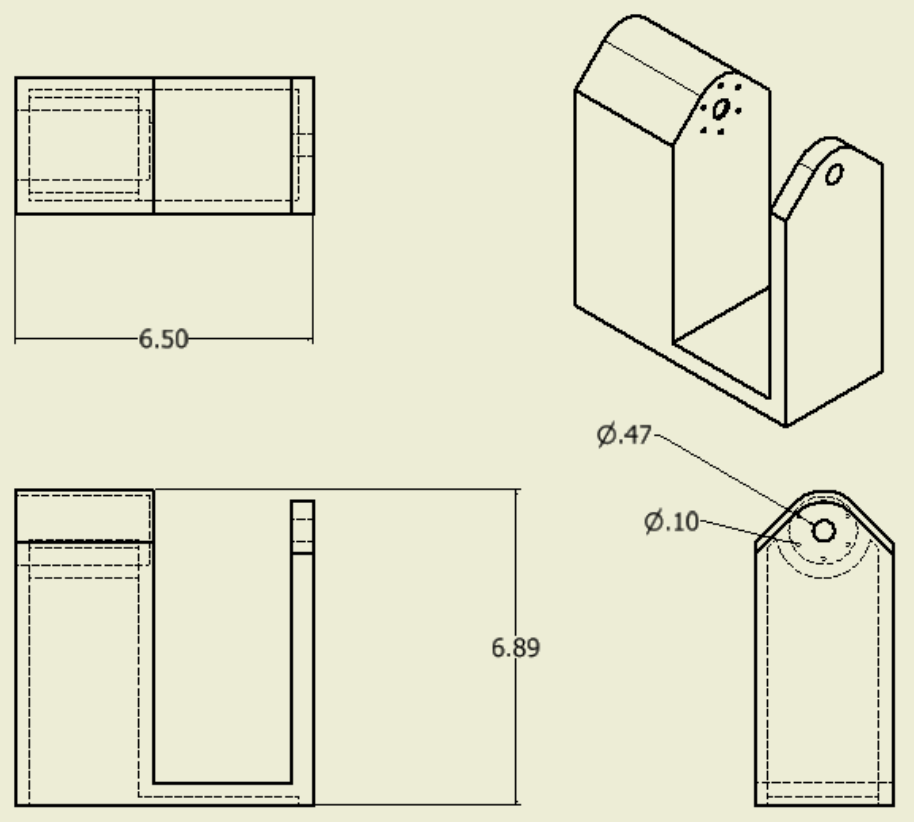

Figure 36: Mechanical Sketch – Spooling Motor Stand

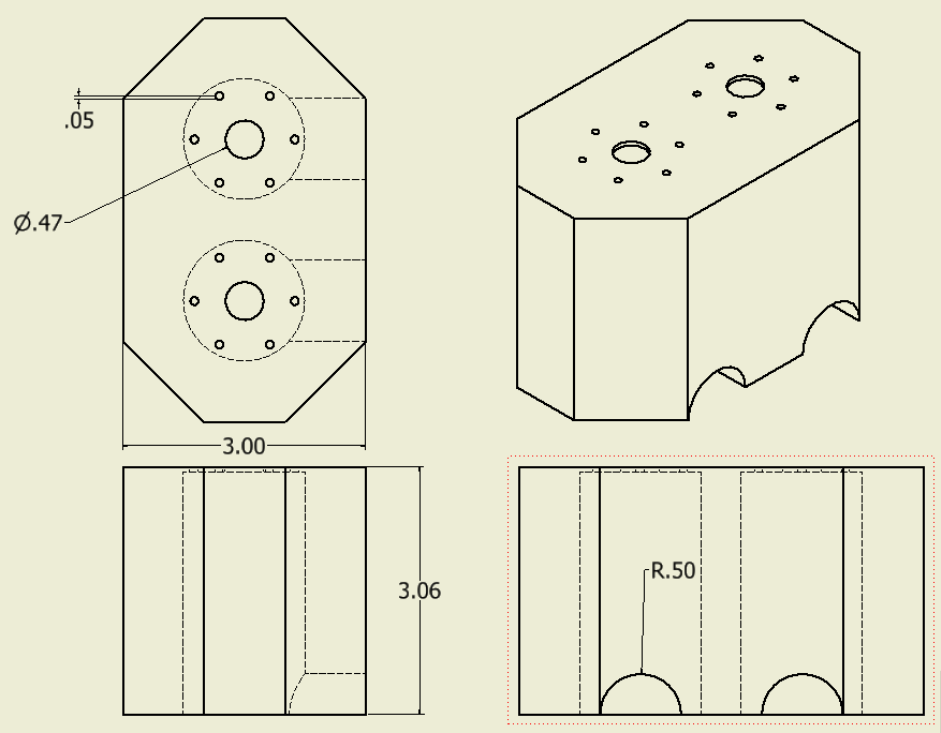

Figure 37: Mechanical Sketch – Motor Spooler Holder

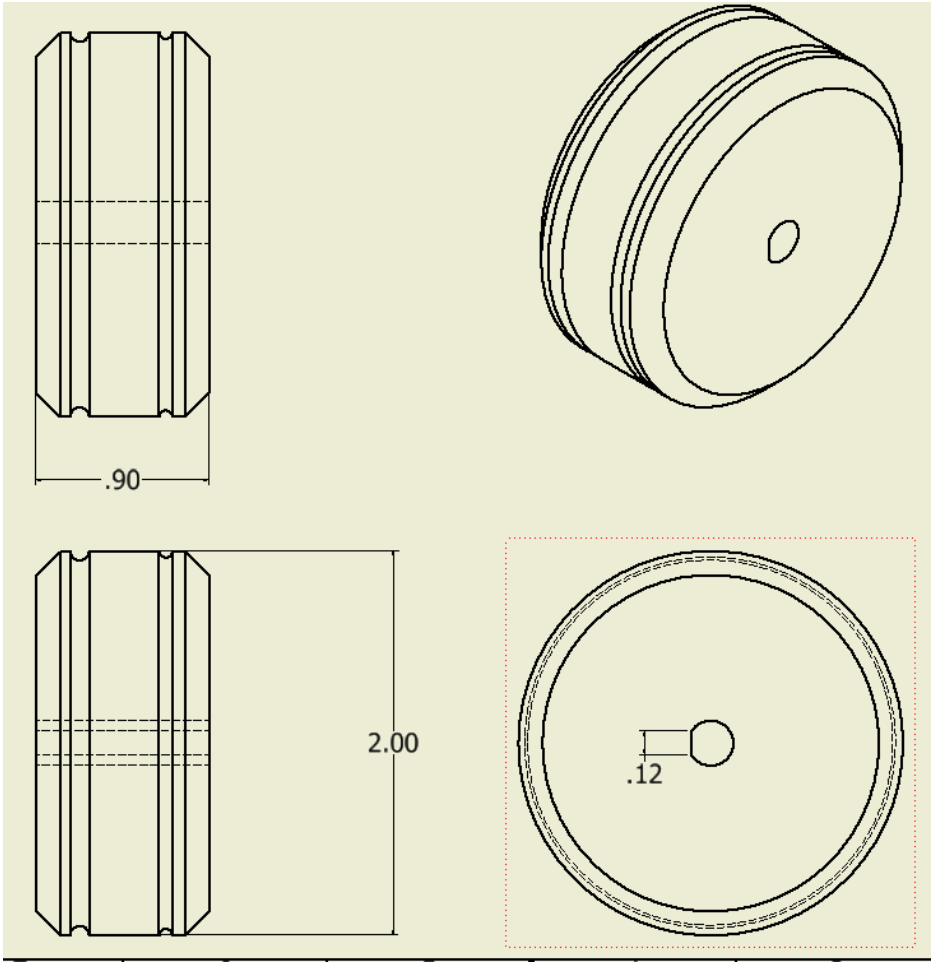

Figure 38: Mechanical Sketch – Tension Motor Discs

(GB)

7. Team Information

Name	Major	Role
Gabriel Bennett	EE/CPE/ME	Project Leader
Lindsay Liebrecht	CPE	Engineering Data Manager
David Lyogky	EE	Hardware Manager
Wilson Woods	CPE	Software Manager

8. Parts Lists

8.1. Accepted Technical Design

The parts list for the accepted technical design can be seen in **Error! Reference source not found.**. The selection of these parts can be seen in the above sections. The component

reference designator does not correspond to any PCB design as of yet. More components may need to be purchased if required in the unforeseeable future.

Table 11: Parts List – Accepted Technical Design

Qty.	Refdes	Part Num.	Description
1		DM320103	PIC32MX470 Curiosity Board - master controller
3		108020103	DC motor driver w/ I2C
1		2050	3.5" TFT 320x480 Display
1		A000066	Arduino Uno Board (drives TFT display)
1		-	LONGRUN Garden Auger Drill Bit for Post Hole Digger, Solid Spiral Drill Bits with 0.79" Inner Shaft for Powerhead -2" x 31.5"
3		-	12V DC Motor 550RPM
1		-	3D Printer Nozzle, V6, Brass, 0.4mm Opening Diameter, for 1.75mm Diameter Filament
1		-	3D Printer Nozzle, V6, Brass, 0.8mm Opening Diameter, for 1.75mm Diameter Filament
2		375	Magnetic Contact Switch (door sensor)
2		4541	Strain guage load cell - 4 wire, 5kg
3		CR40.112.01	Thermocouple probe
1		16621	MAX6675 thermocouple SPI interface
2	U1, U2	LT1085IT-12#PBF	IC REG LINEAR 12V 3A TO220-3
5	D1-D5	1N4007-TP	DIODE GEN PURP 1KV 1A DO41
1	U3	IRF3205PBF	MOSFET N-CH 55V 110A TO220AB
1	U5	KBP206G	BRIDGE RECT 1PHASE 600V 2A KBP
3	U8	MOC3020M	OPTOISOLATOR 4.17KV TRIAC 6DIP
1	U6	4N25	OPTOISO 5KV TRANS W/BASE 6DIP
3	U7	BTA16-800CW3G	TRIAC 800V 16A TO220AB
3	U9	-	High-Temperature Heater for Pipes and Tubes 60" Long with Wire Leads
1	U4	-	50:1 Metal Gearmotor 37Dx54L mm 12V (Helical Pinion)
1		LRS-150-24	AC/DC CONVERTER 24V 156W

(GB,DL,LL,WW)

8.2. Budget

Currently, the project has spent approximately $500 of the $600 budget. To remain below the budget maximum, motors were borrowed from the parts already owned by the university. All parts have not been purchased as of 11.28.21. Parts still required are sensors for velocity and filament diameter, the extruder pipe and insulation and parts for the grinder.

Table 12: Parts List – Budget

			Unit	Total
Qty.	**Part Num.**	**Description**	**Cost ($)**	**Cost ($)**
1	DM320103	PIC32MX470 Curiosity Board – master controller	35.69	35.69
3	108020103	DC motor driver w/ I2C	5.25	15.75
1	2050	3.5” TFT 320x480 Display	39.95	39.95
1	A000066	Arduino Uno Board (drives TFT display)	23.00	23.00
1	-	LONGRUN Garden Auger Drill Bit for Post Hole Digger, Solid Spiral Drill Bits with 0.79” Inner Shaft for Powerhead -2” x 31.5”	39.99	39.99
3	-	12V DC Motor 550RPM	11.88	35.64
1	-	3D Printer Nozzle, V6, Brass, 0.4mm Opening Diameter, for 1.75mm Diameter Filament	10.45	10.45
1	-	3D Printer Nozzle, V6, Brass, 0.8mm Opening Diameter, for 1.75mm Diameter Filament	10.45	10.45
2	375	Magnetic Contact Switch (door sensor)	3.95	7.90
2	4541	Strain guage load cell – 4 wire, 5kg	3.95	7.90
3	CR40.112.01	Thermocouple probe	14.98	44.94
1	16621	MAX6675 thermocouple SPI interface	23.99	23.99
2	LT1085IT-12#PBF	IC REG LINEAR 12V 3A TO220-3	12.55	25.10
5	1N4007-TP	DIODE GEN PURP 1KV 1A DO41	0.13	0.65
1	IRF3205PBF	MOSFET N-CH 55V 110A TO220AB	1.63	1.63
1	KBP206G	BRIDGE RECT 1PHASE 600V 2A KBP	0.61	0.61
3	MOC3020M	OPTOISOLATOR 4.17KV TRIAC 6DIP	0.72	2.16
1	4N25	OPTOISO 5KV TRANS W/BASE 6DIP	0.55	0.55
3	BTA16-800CW3G	TRIAC 800V 16A TO220AB	1.82	5.46
3	-	High-Temperature Heater for Pipes and Tubes 60” Long with Wire Leads	40.61	121.83
1	-	50:1 Metal Gearmotor 37Dx54L mm 12V (Helical Pinion)	24.95	24.95
1	LRS-150-24	AC/DC CONVERTER 24V 156W	22.79	22.79

(GB,LL,DL,WW)

9. Project Schedules

Below is the project Gantt chart for the first phase of this project (Fall semester). The only adjustments made to the original timeline were directed by the class coordinator. All other timelines were met or exceeded by the design team.

	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		SDP I 2021					
2		**Project Design**	**95.38 days**	**Wed 8/25/21**	**Sun 11/28/21**		
3		**Midterm Report**	40 days	Wed 8/25/21	Mon 10/4/21		
4		Cover page	1 day	Mon 10/4/21	Wed 10/6/21		LL
5		T of C, L of T, L of F	1 day	Mon 10/4/21	Wed 10/6/21		LL
6		**Problem Statement**	**40.38 days**	**Wed 8/25/21**	**Mon 10/4/21**		
7		Need	1.38 days	Wed 8/25/21	Thu 8/26/21		GB
8		Objective	1.38 days	Thu 8/26/21	Fri 8/27/21		LL
9		Background	0.38 days	Wed 8/25/21	Wed 8/25/21		DL,GB,LL,WW
10		Marketing Requirements	14.38 days	Mon 9/20/21	Mon 10/4/21		LL
11		Engineering Requirements Specification	14.38 days	Mon 9/20/21	Mon 10/4/21		DL,GB,LL,WW
12		**Engineering Analysis**	**40.38 days**	**Wed 8/25/21**	**Mon 10/4/21**		
13		Circuits (DC, AC, Power, ...) & Electronics (analog and digita	40.38 days	Wed 8/25/21	Mon 10/4/21		DL,GB
14		Cooling Tank Analysis	7.38 days	Mon 9/27/21	Mon 10/4/21		LL
15		System Speed Analysis	7.38 days	Mon 9/27/21	Mon 10/4/21		LL
16		Embedded Systems	37.38 days	Wed 8/25/21	Fri 10/1/21		WW
17		Controls	37.38 days	Wed 8/25/21	Fri 10/1/21		LL
18		**Accepted Technical Design**	**37.38 days**	**Wed 8/25/21**	**Fri 10/1/21**		
19		**Hardware Design: Phase 1**	**37.38 days**	**Wed 8/25/21**	**Fri 10/1/21**		
20		Hardware Block Diagrams Levels 0 thru N (w/ FR tables)	37.38 days	Wed 8/25/21	Fri 10/1/21		DL,GB
21		**Software Design: Phase 1**	**37.38 days**	**Wed 8/25/21**	**Fri 10/1/21**		
22		Software Behavior Models Levels 0 thru N (w/FR tables)	37.38 days	Wed 8/25/21	Fri 10/1/21		LL,WW
23		**Mechanical Sketch**	14 days	Wed 8/25/21	Wed 9/8/21		GB
24		**Team information**	1 day	Wed 8/25/21	Thu 8/26/21		LL
25		**Project Schedules**	**18.38 days**	**Mon 10/25/21**	**Fri 11/12/21**		
26		Final Design Gantt Chart	18.38 days	Mon 10/25/21	Fri 11/12/21		LL
27		**References**	37.38 days	Wed 8/25/21	Fri 10/1/21		WW
28		**Midterm Parts Request Form**	44.38 days	Wed 8/25/21	Fri 10/8/21		DL,GB,LL,WW
29		**Midterm Design Presentations Day 1**	0 days	Wed 9/22/21	Wed 9/22/21		DL,GB,LL,WW
30		**Midterm Design Presentations Day 2**	0 days	Wed 9/29/21	Wed 9/29/21		DL,GB,LL,WW
31		**Project Poster**	55.38 days	Mon 10/4/21	Sun 11/28/21	3	DL,GB,LL,WW
32		**Final Design Report**	55.38 days	Mon 10/4/21	Sun 11/28/21	3	DL,GB,LL,WW
33		**Abstract**	46.38 days	Mon 10/4/21	Fri 11/19/21	3	DL,GB,LL,WW
34		**Hardware Design: Phase 2**	**50.38 days**	**Mon 10/4/21**	**Tue 11/23/21**		
35		**Modules 1...n**	**50.38 days**	**Mon 10/4/21**	**Tue 11/23/21**		
36		Simulations	50.38 days	Mon 10/4/21	Tue 11/23/21	3	DL,GB
37		Schematics	50.38 days	Mon 10/4/21	Tue 11/23/21	3	DL,GB
38		**Software Design: Phase 2**	**50.38 days**	**Mon 10/4/21**	**Tue 11/23/21**		
39		**Modules 1...n**	**50.38 days**	**Mon 10/4/21**	**Tue 11/23/21**		
40		Code (working subsystems)	50.38 days	Mon 10/4/21	Tue 11/23/21	3	LL,WW
41		System integration Behavior Models	50.38 days	Mon 10/4/21	Tue 11/23/21	3	LL,WW
42		**Parts Lists**	**50.38 days**	**Mon 10/4/21**	**Tue 11/23/21**		
43		Parts list(s) for Schematics	50.38 days	Mon 10/4/21	Tue 11/23/21	3	DL,GB,WW,LL
44		Materials Budget list	50.38 days	Mon 10/4/21	Tue 11/23/21	3	DL,GB,WW,LL
45		**Proposed Implementation Gantt Chart**	50.38 days	Mon 10/4/21	Tue 11/23/21	3	LL
46		**Conclusions and Recommendations**	50.38 days	Mon 10/4/21	Tue 11/23/21	3	DL,GB,WW,LL
47		Parts Request Form for Subsystems	34.38 days	Wed 9/22/21	Tue 10/26/21	29	DL,GB,WW,LL
48		Subsystems Demonstrations Day 1	7.38 days	Wed 11/10/21	Wed 11/17/21		DL,GB,WW,LL
49		Parts Request Form for Spring Semester	8.38 days	Tue 11/23/21	Wed 12/1/21	32	DL,GB,WW,LL

(LL)

10. Conclusions & Recommendations

According to the analysis that has been performed, this project will achieve its overall objective of recycling PLA and PET by converting the waste into a viable filament for a 3D printer. This device can be safely operated in a standard household or in an indoor office environment. Based on the grinder and extruder design, the filament created will have a highly repeatable result. This design

is also very modular, allowing the user to add specialized stations such as a color pigmenting station, to the overall design.

The power analysis shows that the system can run on its own 120V outlet with 15A breaker, however this system should be the only high-powered device on that 15A breaker to prevent overloading the breaker or creating an electrical fire and only the grinding or extruding process should run at a time; they should not run in tandem.

For further consideration, this design is recommended to have a high-end control unit from within the PIC32 family such as the PIC32MX470. This design requires a real time operating system which the PIC32 family runs on. This system has high potential functionality and by utilizing an advanced control unit, these potentials will only be amplified. The PIC32 family will also allow for the user to expand on the functionality of the system, such as adding a plastic type and the necessary temperature ranges. Overall, utilizing a high-end control unit will allow for a more intricate design, and a more streamlined office work environment.

To date, the components listed in the parts list in **Error! Reference source not found.** have been purchased and a motor drive has been designed to handle the extrusion screw. Three motor drives were also purchased and programed to run in tandem for the filament spooler and the tension control dual motor pulling motors located in the colling tank. Also completed, is a base control loop that can control the speed of the motors in real time as well as the code required operate the k type thermocouples that will be stationed within the extrusion screw to maintain the appropriate temperatures required to melt the plastics. The feedback from the motor drives and temperature probes are displayed on the control screen.

To make this system realizable the project will require some more specific analysis and design base on the chosen parts required to build the design. The key next steps will be to identify specific components to fit within the general analysis parameters and to complete the parts list for purchasing. Missing components include the sensor for filament diameter, pipe and insulation for around the extrusion screw, and velocity sensor to measure speed of filament creation. Once specific components are chosen, a prototype design can be built to demonstrate proof of concept.

(GB, LL, DL, WW)

11. References

Acerbo, H., Girelli, T., Palazzo, G. (2016). *Method for producing a supply obtained from the recycling of plastic material of industrial and post-consumer residues, to be used by 3d printers* (Patent No. US20160107337A1). U.S. Patent and Trademark Office. https://patents.google.com/patent/US20160107337?oq=3d+filament+recycler

Choong, Y. Y., Tan, H. W., Patel, D. C., Choong, W. T., Chen, C., Low, H. Y., . . . Chua, C. K. (2020). The global rise of 3D printing during the COVID-19 pandemic. *Nature Reviews Materials, 5*(9), 637-639. doi:10.1038/s41578-020-00234-3

del Burgo, J., Damato, R., Méndez, J. A., Ramírez, A. S., Haro, F. B., & Heras, E. S. (2019). Real time analysis of the filament for FDM 3D printers. *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*. doi:10.1145/3362789.3362818

Filament sensor kit for 3D printers and filament extruders. (2020, September 1). Retrieved from https://objectswithintelligence.weebly.com/

Kreiger, M., Mulder, M., Glover, A., & Pearce, J. (2014). Life cycle analysis of distributed recycling of post-consumer high density polyethylene for 3-D printing filament. *Journal of Cleaner Production, 70*, 90-96. doi:10.1016/j.jclepro.2014.02.009

Luo, X., Pei, Z. (2017). *High crystalline poly (lactic acid) filaments for material-extrusion based additive manufacturing* (Patent No. US20170066188A1). U.S. Patent and Trademark Office. https://patents.google.com/patent/US20170066188?oq=PLA+filaments

Mikula, K., Skrzypczak, D., Izydorczyk, G., Warchoł, J., Moustakas, K., Chojnacka, K., & Witek-Krowiak, A. (2020). 3D printing filament as a second life of waste plastics—a review. *Environmental Science and Pollution Research, 28*(10), 12321-12333. doi:10.1007/s11356-020-10657-8

Teterin, E., Zhuravlev, D., & Berchuk, D. (2016). Mobile extrusion machine for the production of composite filaments for 3D printing. *2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*. doi:10.1109/icieam.2016.7910908

Vidakis, N., Petousis, M., Tzounis, L., Maniadi, A., Velidakis, E., Mountakis, N., & Kechagias, J. D. (2021). Sustainable Additive Manufacturing: Mechanical Response of Polyamide 12 over Multiple Recycling Processes. *Materials, 14*(2), 466. doi:10.3390/ma14020466

William, S.A. (1964). *Plastic extrusion, apparatus and control* (Patent No. US3148231A). US Patent and Trademark Office. https://patents.google.com/patent/US3148231A/en?q=plastic+extrusion&oq=plastic+extrusion

Woern, A. L., Mccaslin, J. R., Pringle, A. M., & Pearce, J. M. (2018). RepRapable Recyclebot: Open source 3-D printable extruder for converting plastic to 3-D printing filament. *HardwareX, 4*. doi:10.1016/j.ohx.2018.e00026

(WW)

12. Appendices

12.1. Data Sheets Hyperlinks

The datasheets for the components purchased can be seen in Table 13 below.

Table 13: Data Sheets

PIC32MX470 Curiosity Board - master controller
https://www.mouser.com/datasheet/2/268/70005283A-1075423.pdf
DC motor driver w/ I2C
https://www.mouser.com/datasheet/2/744/TB6612FNG_datasheet_en_20141001-2529381.pdf
3.5" TFT 320x480 Display
https://www.mouser.com/datasheet/2/737/HX8357_D_DS_April2012-2488187.pdf
Arduino Uno Board (drives TFT display)
https://www.mouser.com/ProductDetail/Arduino/A000066?qs=sGAEpiMZZMt1iCLsaqcCFuqYk3NvvZsVCTWC7OL1sws%3D
LONGRUN Garden Auger Drill Bit for Post Hole Digger, Solid Spiral Drill Bits with 0.79" Inner Shaft for Powerhead -2" x 31.5"
https://www.amazon.com/dp/B08KDKW53R/ref=syn_sd_onsite_desktop_283?pd_rd_plhdr=t&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUE5V0JINlNMVUFFRUomZW5jcnlwdGVkSWQ9QTA1MzU4ODE3SjlJVlEzRjFJQkcmZW5jcnlwdGVkQWRJZD1BMDc1NDI2ODFENzBUSUlNQ1Q0UVAmd2lkZ2V0TmFtZT1zZF9vbnNpdGVfZGVza3RvcCZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU&th=1
12V DC Motor 550RPM
https://www.amazon.com/dp/B0925WGSKT/ref=twister_B093LDNFJX?_encoding=UTF8&th=1
3D Printer Nozzle, V6, Brass, 0.4mm Opening Diameter, for 1.75mm Diameter Filament
https://www.mcmaster.com/3695N304/
3D Printer Nozzle, V6, Brass, 0.8mm Opening Diameter, for 1.75mm Diameter Filament
https://www.mcmaster.com/3695N307/
Magnetic Contact Switch (door sensor)
https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/375_Web.pdf
Strain guage load cell - 4 wire, 5kg
https://www.digikey.com/en/products/detail/adafruit-industries-llc/4541/12323573?s=N4IgjCBcoKwwDFUBjKAzAhgGwM4FMAaEAeygG0QAWAZnjkQF0iAHAFyhAGVWAnASwB2AcxABfUUQBM5LrwyCABAHEMAVyF4FAGWIYAJgoDCeLFgUBaBZQUB1Pjzw4QDUUA
Thermocouple probe
https://www.amazon.com/CrocSee-Temperature-Sensor-Thermocouple-58-572%C2%B0F/dp/B071DW3GVQ/ref=pd_bxgy_2/146-8502997-5622903?pd_rd_w=FDg4L&pf_rd_p=c64372fa-c41c-422e-990d-9e034f73989b&pf_rd_r=ZCHDDT2SHZ8VPXYHPZF9&pd_rd_r=49df6fee-42b5-443c-b584-3cc3e4acfa90&pd_rd_wg=dsymH&pd_rd_i=B071DW3GVQ&psc=1
MAX6675 thermocouple SPI interface
https://www.amazon.com/ACEIRMC-Thermocouple-Temperature-Compatible-Raspberry/dp/B092ZCSM7J/ref=sr_1_4?dchild=1&keywords=max6675+module&qid=1635191497&qsid=133-2789808-5693116&s=industrial&sr=1-

4&sres=B08LMXWYZ8%2CB00PVTH4MW%2CB092ZCSM7J%2CB01HT871SO%2CB0915MWCHR%2CB0932JKLLX%2CB096VLRH31%2CB07MDWNCFD%2CB0915D9Y9S%2CB092S76JHW%2CB07QBPGVZZ%2CB07PPRPZ6M%2CB07FZX7VPB%2CB083S9X73F%2CB07TZ7CCVL%2CB09B267DRP%2CB07FM4DGMX%2CB07WYY55RK%2CB094JP944P%2CB09DV3YJV2
IC REG LINEAR 12V 3A TO220-3
https://www.analog.com/media/en/technical-documentation/data-sheets/1083ffe.pdf
DIODE GEN PURP 1KV 1A DO41
https://www.mccsemi.com/pdf/Products/1N4001~1N4007(DO-41).pdf
MOSFET N-CH 55V 110A TO220AB
https://www.infineon.com/dgdl/irf3205pbf.pdf?fileId=5546d462533600a4015355def244190a
BRIDGE RECT 1PHASE 600V 2A KBP
https://www.diodes.com/assets/Datasheets/ds21205.pdf
OPTOISOLATOR 4.17KV TRIAC 6DIP
https://www.onsemi.com/pdf/datasheet/moc3023m-d.pdf
OPTOISO 5KV TRANS W/BASE 6DIP
https://www.vishay.com/docs/83725/4n25.pdf
TRIAC 800V 16A TO220AB
https://www.littelfuse.com/~/media/electronics/datasheets/switching_thyristors/littelfuse_thyristor_bta16_600cw3_d_datasheet.pdf.pdf
"High-Temperature Heater for Pipes and Tubes 60"" Long with Wire Leads"
https://www.mcmaster.com/3641K25/
50:1 Metal Gearmotor 37Dx54L mm 12V (Helical Pinion)
https://www.pololu.com/file/0J1736/pololu-37d-metal-gearmotors-rev-1-2.pdf
AC/DC CONVERTER 24V 156W
https://www.meanwellusa.com/upload/pdf/LRS-150/LRS-150-spec.pdf

12.2. Completed Code as of 11.28.21

Master Controller Code Listing

```
/*******************************************************************************
* Extruder_Controller
* master.cpp
* 11.18.2021
*******************************************************************************
#include <cstdint>
#include <vector>
#include <tuple>
#include <utility>

#include "config/default/peripheral/i2c/master/plib_i2c1_master.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "master.h"
#include "globals.h"
#include "DataManager.h"
#include "I2C.h"

MASTER_DATA masterData;

void MASTER_Initialize ( void )
{
	masterData.state = MASTER_STATE_INIT;
}

void MASTER_Tasks ( void )
{
	switch ( masterData.state )
	{
		case MASTER_STATE_INIT:
		{
			I2C_2_Init();
			bool appInitialized = true;

		if (appInitialized)
			masterData.state = MASTER_STATE_SERVICE_TASKS;
	break;
}


		case MASTER_STATE_SERVICE_TASKS:
```

```
                {
                        /*
                        * TO DO: Implement process control flow state machine
                        */
                        globalDataManager.pollNumericParams();
                        globalDataManager.sendAllFreshNumericParams();
                        CORETIMER_DelayUs(50);
                        break;
                }

                default:
                {
                        break;
                }
        }
}

/*********************************End master.cpp*********************************/

/*******************************************************************************
* Extruder_Controller
* extrusion_input.cpp
* 11.25.2021
*******************************************************************************/
#include <cstdint>

#include "extrusion_input.h"
#include "config/default/peripheral/gpio/plib_gpio.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "globals.h"
#include "SPI.h"
#include "TempSensor.h"
#include "DataManager.h"

float temp_1_float = 0;
float temp_2_float = 0;
float temp_3_float = 0;

TempSensor tempSensor1(1);
TempSensor tempSensor2(2);
TempSensor tempSensor3(3);

EXTRUSION_INPUT_DATA extrusion_inputData;

void EXTRUSION_INPUT_Initialize ( void )
```

```
{
	extrusion_inputData.state = EXTRUSION_INPUT_STATE_INIT;
}

void EXTRUSION_INPUT_Tasks ( void )
{
	switch ( extrusion_inputData.state )
	{
		case EXTRUSION_INPUT_STATE_INIT:
		{
			SPI_Init();
			CORETIMER_DelayUs(50);
			bool appInitialized = true;

			if (appInitialized)
				extrusion_inputData.state =
				EXTRUSION_INPUT_STATE_SERVICE_TASKS;

			break;
		}

		case EXTRUSION_INPUT_STATE_SERVICE_TASKS:
		{
			temp_1_float = tempSensor1.readTemp();
			globalDataManager.setNumericParam(ZONE_1_TEMP_INDEX,
		temp_1_float);
			temp_1_float = 0;
			CORETIMER_DelayUs(10);
			temp_2_float = tempSensor2.readTemp();
			globalDataManager.setNumericParam(ZONE_2_TEMP_INDEX,
		temp_2_float);
			temp_2_float = 0;
			CORETIMER_DelayUs(10);
			temp_3_float = tempSensor3.readTemp();
			globalDataManager.setNumericParam(ZONE_3_TEMP_INDEX,
		temp_3_float);
			temp_3_float = 0;
			CORETIMER_DelayMs(500);
			break;
		}

		default:
		{
			break;
		}
	}
```

```
}

/*******************************End extrusion_input.cpp*****************************/



/*******************************************************************************
* Extruder_Controller
* extrusion_control.cpp
* 11.25.2021
*******************************************************************************

#include <cstdint>

#include "extrusion_control.h"
#include "config/default/peripheral/i2c/master/plib_i2c1_master.h"
#include "config/default/peripheral/tmr/plib_tmr2.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "globals.h"
#include "DataManager.h"
#include "I2C.h"
#include "I2CMotor.h"

const uint16_t CONTROLLER_1_I2C_ADDRESS = 0x0C;
const uint16_t CONTROLLER_2_I2C_ADDRESS = 0x04;

I2CMotor controller1(CONTROLLER_1_I2C_ADDRESS);
I2CMotor controller2(CONTROLLER_2_I2C_ADDRESS);

const uint8_t MOTOR_1_ID = 0;
const uint8_t MOTOR_2_ID = 1;
float current_motor_speed = 0;

float current_temp_1 = 0;
float current_temp_2 = 0;
float current_temp_3 = 0;
float current_diameter = 0;

EXTRUSION_CONTROL_DATA extrusion_controlData;

void EXTRUSION_CONTROL_Initialize ( void )
{
        extrusion_controlData.state = EXTRUSION_CONTROL_STATE_INIT;
}
```

```
void EXTRUSION_CONTROL_Tasks ( void )
{
	switch ( extrusion_controlData.state )
	{
		case EXTRUSION_CONTROL_STATE_INIT:
		{
			I2C_1_Init();
			CORETIMER_DelayMs(500);
			while(I2C_1_IS_BUSY);
			controller1.setMotorSpeed(MOTOR_1_ID, 100, -1);
			controller1.setMotorSpeed(MOTOR_2_ID, 100, 1);
			current_motor_speed = controller1.getMotorSpeed(0);
			current_motor_speed = (current_motor_speed / 255) * 100;
			globalDataManager.setNumericParam(ROLLER_SPEED_INDEX,
		current_motor_speed);
			while(I2C_1_IS_BUSY);
			controller2.setMotorSpeed(MOTOR_1_ID, 80, -1);
			current_motor_speed = controller2.getMotorSpeed(0);
			current_motor_speed = (current_motor_speed / 255) * 100;
			globalDataManager.setNumericParam(SPOOLER_SPEED_INDEX,
		current_motor_speed);
			bool appInitialized = true;

			if (appInitialized)
			{
				extrusion_controlData.state =
			EXTRUSION_CONTROL_STATE_SERVICE_TASKS;
			}
			break;
		}

		case EXTRUSION_CONTROL_STATE_SERVICE_TASKS:
		{
			current_temp_1 =
		globalDataManager.getNumericParam(ZONE_1_TEMP_INDEX);
			current_temp_2 =
		globalDataManager.getNumericParam(ZONE_2_TEMP_INDEX);
			current_temp_3 =
		globalDataManager.getNumericParam(ZONE_3_TEMP_INDEX);
			current_diameter =
		globalDataManager.getNumericParam(DIAMETER_INDEX);
			/*
			 * TO DO: Implement feedback control
			 */
		}
```

```
                default:
                {
                        break;
                }
        }
}

/*******************************End extrusion_control.cpp************************/

/*****************************************************************************
* Extruder_Controller
* I2C.c
* 11.18.2021
*****************************************************************************/

#include "stdint.h"

#include "config/default/peripheral/i2c/master/plib_i2c1_master.h"
#include "config/default/peripheral/i2c/master/plib_i2c2_master.h"
#include "config/default/peripheral/tmr/plib_tmr2.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "I2C.h"

void I2C_1_Init()
{
   IEC1CLR = _IEC1_I2C1MIE_MASK;            // disable I2C master interrupt
   IEC1CLR = _IEC1_I2C1BIE_MASK;            // disable I2C collision interrupt
   I2C1CONbits.DISSLW = 1;                  // disable slew rate for 100kHz
   I2C1BRG = 235;                           // 1kHz = 235 | 4kHz = 55
   while ( I2C1STATbits.P );
   I2C1CONbits.A10M = 0;                    // 7-bit address mode
   I2C1CONbits.I2CEN = 1;                   // enable module
}

void I2C_2_Init()
{
   IEC1CLR = _IEC1_I2C2MIE_MASK;            // disable I2C master interrupt
   IEC1CLR = _IEC1_I2C2BIE_MASK;            // disable I2C collision interrupt
   I2C2CONbits.DISSLW = 1;                  // disable slew rate for 100kHz
   I2C2BRG = 235;                           // 1kHz = 235 | 4kHz = 55
   while ( I2C2STATbits.P );
   I2C2CONbits.A10M = 0;                    // 7-bit address mode
   I2C2CONbits.I2CEN = 1;                   // enable module
}
```

```
void I2C_1_Wait_For_Idle(void)
{
   while(I2C1CON & 0x1F);
   while(I2C1STATbits.TRSTAT);
}

void I2C_2_Wait_For_Idle(void)
{
   while(I2C2CON & 0x1F);
   while(I2C2STATbits.TRSTAT);
}

void I2C_1_Start( void )
{
   I2C_1_Wait_For_Idle();
   I2C1CONbits.SEN = 1;                    // initiate start condition
   while ( I2C1CONbits.SEN );              // wait for start condition
}

void I2C_2_Start( void )
{
   I2C_2_Wait_For_Idle();
   I2C2CONbits.SEN = 1;                    // initiate start condition
   while ( I2C2CONbits.SEN );              // wait for start condition
}

void I2C_1_Stop( void )
{
   I2C_1_Wait_For_Idle();
   I2C1CONbits.PEN = 1;
   while ( I2C1CONbits.PEN );
   CORETIMER_DelayUs(5);
}

void I2C_2_Stop( void )
{
   I2C_2_Wait_For_Idle();
   I2C2CONbits.PEN = 1;
   while ( I2C2CONbits.PEN );
   CORETIMER_DelayUs(5);
}

bool I2C_1_Send_Byte( char data )
{
   while ( I2C1STATbits.TBF );             // wait if buffer is full
   I2C_1_Wait_For_Idle();
```

```
   I2C1TRN = data;                          // pass data to transmit register
   CORETIMER_DelayUs(5);
   return (I2C1STATbits.ACKSTAT == 0);
}

bool I2C_2_Send_Byte( char data )
{
   while ( I2C2STATbits.TBF );              // wait if buffer is full
   I2C_2_Wait_For_Idle();
   I2C2TRN = data;                          // pass data to transmit register
   CORETIMER_DelayUs(5);
   return (I2C2STATbits.ACKSTAT == 0);
}

char I2C_1_Get_Byte( void )
{
   I2C1CONbits.RCEN = 1;                     // set RCEN, enable I2C receive mode
   while ( !I2C1STATbits.RBF );             // wait for byte to shift into register
   I2C1CONbits.ACKEN = 1;                    // master send acknowledge
   CORETIMER_DelayUs(5);
   return ( I2C1RCV );
}

char I2C_2_Get_Byte( void )
{
   I2C2CONbits.RCEN = 1;                    // set RCEN, enable I2C receive mode
   while ( !I2C2STATbits.RBF );             // wait for byte to shift into register
   I2C2CONbits.ACKEN = 1;                   // master send acknowledge
   CORETIMER_DelayUs(5);
   return ( I2C2RCV );
}

/**********************************End I2C.c*********************************/

/*****************************************************************************
* Extruder_Controller
* SPI.c
* 11.18.2021
*****************************************************************************/
#include "config/default/peripheral/spi/spi_master/plib_spi1_master.h"
#include "config/default/peripheral/gpio/plib_gpio.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "SPI.h"

void SPI_Init( void )
```

```
{
   uint32_t rdata = 0U;
   IEC1CLR = 0x8;
   IEC1CLR = 0x10;
   IEC1CLR = 0x20;
   SPI1CON = 0;
   rdata = SPI1BUF;
   rdata = rdata;
   IFS1CLR = 0x8;
   IFS1CLR = 0x10;
   IFS1CLR = 0x20;
   SPI1BRG = 23;                        // 1MHz clock
   SPI1STATCLR = _SPI1STAT_SPIROV_MASK;
   SPI1CON = 0x8560;                    // PIC32 master, 16-bit mode
}

void SPI_Transfer(char data)
{
   SPI1BUF = (0x00FF & data);           // pass data to buffer
   while(!SPI1BUF);                     // wait for data to be sent out
}

/**********************************End SPI.c*********************************/

/*****************************************************************************
* Extruder_Controller
* I2CMotor.h
* 11.19.2021
*****************************************************************************/
#ifndef I2CMOTOR_H
#define I2CMOTOR_H

#include <cstdint>
#include <vector>

class I2CMotor
{
   public:

      uint8_t controller_I2C_address  = 0;
      const char CLOCKWISE = 1;
      const char COUNTERCLOCKWISE = -1;

      enum MOTOR_ID
      {
         MOTOR_NULL,
```

```
    MOTOR_1,
    MOTOR_2
};

typedef struct
{
    MOTOR_ID motor_id;
    unsigned short current_speed;
    short current_direction;
} Motor_Object;

// motor_1 and motor_2 init speed = 0, direction = clockwise
Motor_Object motor_1 = { MOTOR_1, 0, 1 };
Motor_Object motor_2 = { MOTOR_2, 0, 1 };

std::vector<Motor_Object> motor_objects = { motor_1, motor_2 };

uint8_t DIRECTION_SET = 0xaa;
uint8_t MOTOR_DIR_BOTH_CW = 0x0a;
uint8_t MOTOR_DIR_BOTH_CCW = 0x05;
uint8_t MOTOR_DIR_M1CW_M2CCW = 0x06;
uint8_t MOTOR_DIR_M1CCW_M2CW = 0x09;
uint8_t MOTOR_SPEED_SET = 0x82;
uint8_t PWMFrequencySet = 0x84;
uint8_t MotorSetA = 0xa1;
uint8_t MotorSetB = 0xa5;
uint8_t NOTHING = 0x01;
uint8_t F_31372Hz = 0x01;
uint8_t F_3921Hz = 0x02;
uint8_t F_490Hz = 0x03;
uint8_t F_122Hz = 0x04;
uint8_t F_30Hz = 0x05;

// speed 0 to 255
unsigned char SPEED_MOTOR_1 = 0;
unsigned char SPEED_MOTOR_2 = 0;
// clockwise = 1 | counterclockwise = -1
int DIRECTION_MOTOR_1 = 1;
int DIRECTION_MOTOR_2 = 1;

I2CMotor(uint16_t i2c_address);
I2CMotor(const I2CMotor& orig);
virtual ~I2CMotor();
void setPWMFrequency();
float getMotorSpeed(unsigned char motor_id);
void setMotorDirection(uint8_t motor_directions);
```

```
        void setMotorSpeed(unsigned char motor_id, unsigned short new_speed, char
new_direction);
        void nudgeMotorSpeedUp(unsigned char motor_id, unsigned char amount);
        void nudgeMotorSpeedDown(unsigned char motor_id, unsigned char amount);
        int stopMotor(unsigned char motor_id);
};

#endif

/******************************End I2CMotor.h*********************************/

/*****************************************************************************
* Extruder_Controller
* I2CMotor.cpp
* 11.19.2021
*****************************************************************************/

#include <cstdint>
#include <vector>
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "I2CMotor.h"
#include "globals.h"
#include "I2C.h"

I2CMotor::I2CMotor(uint16_t i2c_address) {}

I2CMotor::I2CMotor(const I2CMotor& orig) {}

I2CMotor::~I2CMotor() {}

void I2CMotor::setPWMFrequency()
{
   I2C_1_IS_BUSY = true;
   I2C_1_Start();
   CORETIMER_DelayUs(5);
   I2C_1_Send_Byte(controller_I2C_address << 1);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(PWMFrequencySet);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(F_3921Hz);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(NOTHING);
   I2C_1_Stop();
   I2C_1_IS_BUSY = false;
   CORETIMER_DelayMs(4);
```

```
}

float I2CMotor::getMotorSpeed(unsigned char motor_id)
{
   return (float)motor_objects[motor_id].current_speed;
}

void I2CMotor::setMotorDirection(uint8_t motor_directions)
{
   I2C_1_IS_BUSY = true;
   I2C_1_Start();
   CORETIMER_DelayUs(5);
   I2C_1_Send_Byte(controller_I2C_address << 1);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(DIRECTION_SET);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(motor_directions);
   CORETIMER_DelayUs(10);
   I2C_1_Send_Byte(NOTHING);
   I2C_1_Stop();
   I2C_1_IS_BUSY = false;
   CORETIMER_DelayUs(200);
}

// motor_id = 0 for motor_1 and 1 for motor_2
void I2CMotor::setMotorSpeed(unsigned char motor_id, unsigned short new_speed, char
new_direction)
{
   motor_objects[motor_id].current_direction = new_direction;
   if (new_speed >= 255)
      motor_objects[motor_id].current_speed = 255;
   else
      motor_objects[motor_id].current_speed = new_speed;
   // Set the direction
   if (motor_objects[0].current_direction == 1 && motor_objects[1].current_direction == 1)
      setMotorDirection(MOTOR_DIR_BOTH_CW);
   if (motor_objects[0].current_direction == 1 && motor_objects[1].current_direction == -1)
      setMotorDirection(MOTOR_DIR_M1CW_M2CCW);
   if (motor_objects[0].current_direction == -1 && motor_objects[1].current_direction == 1)
      setMotorDirection(MOTOR_DIR_M1CCW_M2CW);
   if (motor_objects[0].current_direction == -1 && motor_objects[1].current_direction == -1)
      setMotorDirection(MOTOR_DIR_BOTH_CCW);
   // send command
   I2C_1_IS_BUSY = true;
   I2C_1_Start();
   CORETIMER_DelayUs(5);
```

```
    I2C_1_Send_Byte(controller_I2C_address << 1);
    CORETIMER_DelayUs(10);
    I2C_1_Send_Byte(MOTOR_SPEED_SET);
    CORETIMER_DelayUs(10);
    I2C_1_Send_Byte(motor_objects[0].current_speed);
    CORETIMER_DelayUs(10);
    I2C_1_Send_Byte(motor_objects[1].current_speed);
    I2C_1_Stop();
    I2C_1_IS_BUSY = false;
    CORETIMER_DelayUs(200);
}

void I2CMotor::nudgeMotorSpeedUp(unsigned char motor_id, unsigned char amount)
{
    short new_speed = motor_objects[motor_id].current_speed + amount;
    if (new_speed >= 255)
        new_speed = 255;
    setMotorSpeed(motor_id, new_speed, motor_objects[motor_id].current_direction);
}
void I2CMotor::nudgeMotorSpeedDown(unsigned char motor_id, unsigned char amount)
{
    short new_speed = motor_objects[motor_id].current_speed - amount;
    if (new_speed <= 0)
        stopMotor(motor_id);
    else
        setMotorSpeed(motor_id, new_speed, motor_objects[motor_id].current_direction);
}

int I2CMotor::stopMotor(unsigned char motor_id)
{
    setMotorSpeed(motor_id, 0, 1);
    return 0;
}

/*****************************End I2CMotor.c*********************************/

/*****************************************************************************
* Extruder_Controller
* TempSensor.h
* 11.25.2021
*****************************************************************************/
#ifndef TEMPSENSOR_H
#define TEMPSENSOR_H

#include <cstdint>
```

```
class TempSensor
{
   public:

      TempSensor(uint8_t sensor_id);
      TempSensor(const TempSensor& orig);
      virtual ~TempSensor();
      float readTemp();

   private:

      bool SPI_INITIALIZED = false;
      bool READ_CELSIUS = false;
      uint8_t SENSOR_ID = 0;
};

#endif

/******************************End TempSensor.h*********************************/

/*****************************************************************************
* Extruder_Controller
* TempSensor.cpp
* 11.25.2021
*****************************************************************************/
#include <cstdint>

#include "config/default/peripheral/gpio/plib_gpio.h"
#include "config/default/peripheral/spi/spi_master/plib_spi1_master.h"
#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "TempSensor.h"
#include "globals.h"
#include "SPI.h"

TempSensor::TempSensor(uint8_t sensor_id)
{
   SENSOR_ID = sensor_id;
   switch(SENSOR_ID)
   {
      case 1:
         SS_TEMP_1_OutputEnable();
         SS_TEMP_1_Set();
         break;
      case 2:
         SS_TEMP_2_OutputEnable();
```

```
            SS_TEMP_2_Set();
            break;
        case 3:
            SS_TEMP_3_OutputEnable();
            SS_TEMP_3_Set();
            break;
        default:
            break;
    }
}

TempSensor::TempSensor(const TempSensor& orig) {}

TempSensor::~TempSensor() {}

float TempSensor::readTemp()
{
    switch(SENSOR_ID)
    {
        case 1:
            SPI1CONbits.DISSDO = 1;
            SS_TEMP_1_Clear();                  // set ss low
            SPI_Transfer(0x00);                 // send dummy byte
            CORETIMER_DelayUs(16);              // wait 16 cycles for 2 bytes
            SPI1CONbits.DISSDO = 0;
            SS_TEMP_1_Set();                    // set ss high
            break;
        case 2:
            SPI1CONbits.DISSDO = 1;
            SS_TEMP_2_Clear();                  // set ss low
            SPI_Transfer(0x00);                 // send dummy byte
            CORETIMER_DelayUs(16);              // wait 16 cycles for 2 bytes
            SPI1CONbits.DISSDO = 0;
            SS_TEMP_2_Set();                    // set ss high
            break;
        case 3:
            SPI1CONbits.DISSDO = 1;
            SS_TEMP_3_Clear();                  // set ss low
            SPI_Transfer(0x00);                 // send dummy byte
            CORETIMER_DelayUs(16);              // wait 16 cycles for 2 bytes
            SPI1CONbits.DISSDO = 0;
            SS_TEMP_3_Set();                    // set ss high
            break;
        default:
            break;
    }
```

```
    int raw_temp = SPI1BUF;                         // read incoming data from buffer
    raw_temp >>= 3;
    float celsius_temp = (float)raw_temp;
    if (READ_CELSIUS == true)
        return celsius_temp;
    else
        return ((celsius_temp * (9 / 5)) + 32);
}

/*******************************End TempSensor.cpp*****************************/

/*****************************************************************************
* Extruder_Controller
* DataManager.h
* 11.18.2021
*****************************************************************************/

#ifndef DATAMANAGER_H
#define DATAMANAGER_H

#include <cstdint>
#include <vector>
#include <utility>
#include <tuple>

class DataManager
{
    public:

        const uint16_t DISPLAY_I2C_ADDRESS = 0x14;

        DataManager();
        DataManager(const DataManager& orig);
        ~DataManager();
        void setNumericParam(uint8_t index, float param);
        void setStatusParam(uint8_t index, uint8_t param);
        float   getNumericParam(uint8_t index);
        uint8_t getStatusParam(uint8_t index);
        void clearNumericParamFlag(uint8_t index);
        void clearStatusParamFlag(uint8_t index);
        void pollNumericParams();
        void pollStatusParams();
        void sendNumericParamI2C(uint8_t data_id, float value);
        void sendStatusParamI2C(uint8_t data_id, uint8_t status);
        void sendAllFreshNumericParams();
```

```
    void sendAllFreshStatusParams();

    std::vector<uint8_t>& getFreshNumericIDs();
    std::vector<float>&   getFreshNumericValues();
    std::vector<uint8_t>& getFreshStatusIDs();
    std::vector<uint8_t>& getFreshStatusValues();

  private:

    typedef union                // convert between float and char[4]
    {
       uint8_t buffer[4];
       float numeric_param_input;
    } FloatToBytes;

    FloatToBytes converter;

    typedef enum                 // status values
    {
       NONE,
       READY,
       ON,
       OFF,
       OPEN,
       CLOSED,
       IN_PROGRESS,
       COMPLETE
    } STATUS;

    typedef struct                    // numeric parameter object
    {
       const uint8_t data_index;      // index in numeric_params[] vector
       const uint8_t data_id;         // unique identifier, read by display
       float value;                   // numeric parameter value
       bool is_current;               // status flag
    } Numeric_Param;

    typedef struct                    // status parameter object
    {
       const uint8_t data_index;      //index in status_params[] vector
       const uint8_t data_id;         // unique identifier, read by display
       STATUS status;                 // status parameter value
       bool is_current;               // status flag
    } Status_Param;

    const uint8_t NUMERIC_PARAM_COUNT = 12;      // length of numeric_params[]
```

```
    const uint8_t STATUS_PARAM_COUNT = 4;          // length of status_params[]

    // Numeric_Param objects for all numeric values to be tracked
    Numeric_Param desired_yield     = { 0,  0x01, 0.0, false };
    Numeric_Param required_input    = { 1,  0x02, 0.0, false };
    Numeric_Param ground_weight     = { 2,  0x03, 0.0, false };
    Numeric_Param zone_1_temp       = { 3,  0x04, 0.0, false };
    Numeric_Param zone_2_temp       = { 4,  0x05, 0.0, false };
    Numeric_Param zone_3_temp       = { 5,  0x06, 0.0, false };
    Numeric_Param screw_speed       = { 6,  0x07, 0.0, false };
    Numeric_Param roller_speed      = { 7,  0x08, 0.0, false };
    Numeric_Param spooler_speed     = { 8,  0x09, 0.0, false };
    Numeric_Param filament_diameter = { 9,  0x0A, 0.0, false };
    Numeric_Param extruded_length   = { 10, 0x0B, 0.0, false };
    Numeric_Param projected_yield   = { 11, 0x0C, 0.0, false };

    // Status_Param objects for all status values to be tracked
    Status_Param hopper_lid_status  = { 0, 0x10, NONE, false };
    Status_Param grinder_status     = { 1, 0x20, NONE, false };
    Status_Param preparation_status = { 2, 0x30, NONE, false };
    Status_Param extrusion_status   = { 3, 0x40, NONE, false };

    // vector of Numeric_Param structs
    std::vector<Numeric_Param> numeric_params = { desired_yield,
       required_input, ground_weight, zone_1_temp, zone_2_temp,
       zone_3_temp, screw_speed, roller_speed, spooler_speed,
       filament_diameter, extruded_length, projected_yield };

    // vector of Status_Param structs
    std::vector<Status_Param> status_params = { hopper_lid_status,
       grinder_status, preparation_status, extrusion_status };

    std::vector<uint8_t> fresh_numeric_IDs;
    std::vector<float> fresh_numeric_values;
    std::vector<uint8_t> fresh_status_IDs;
    std::vector<uint8_t> fresh_status_values;
};

#endif

/*******************************End DataManager.h*******************************/

/*****************************************************************************
* Extruder_Controller
* DataManager.cpp
* 11.18.2021
```

```
*******************************************************************************/
#include <cstdint>
#include <vector>
#include <tuple>
#include <utility>

#include "config/default/peripheral/coretimer/plib_coretimer.h"

#include "DataManager.h"
#include "globals.h"
#include "I2C.h"

DataManager::DataManager() {}

DataManager::DataManager(const DataManager& orig) {}

DataManager::~DataManager() {}
/**
* DataManager::setNumericParam()
*
* @param index Parameter index in numeric_params[] vector
*
* @param value Parameter value to be added to numeric_params[] vector
*
* Update global DataManager numeric parameter vector
* Values added to this vector are automatically sent via I2C to the display
*/
void DataManager::setNumericParam(uint8_t index, float value)
{
   numeric_params[index].value = value;
   numeric_params[index].is_current = false;
}

/**
* DataManager::setStatusParam()
*
* @param index Parameter index in status_params[] vector
*
* @param value Parameter value to be added to status_params[] vector
*
* Update global DataManager status (non-numeric) parameter vector
* Values added to this vector are automatically sent via I2C to the display
*/
void DataManager::setStatusParam(uint8_t index, uint8_t status)
{
   status_params[index].status = (DataManager::STATUS) status;
```

```
    status_params[index].is_current = false;
}

/**
* DataManager::getNumericParam()
*
* @param index Parameter index in numeric_params[] vector
*
* @return Numeric parameter value stored in numeric_params[] at index
*
* Retrieve the numeric value stored in numeric_params[] at index
*/
float DataManager::getNumericParam(uint8_t index)
{
    return numeric_params[index].value;
}
/**
* DataManager::getStatusParam()
*
* @param index Parameter index in status_params[] vector
*
* @return Status parameter value stored in status_params[] at index
*
* Retrieve the status (non-numeric) value stored in status_params[] at index
*/
uint8_t DataManager::getStatusParam(uint8_t index)
{
    return status_params[index].status;
}

/**
* DataManager::clearNumericParamFlag()
*
* @param index Parameter index in numeric_params[] vector
*
* Clear flag associated with the numeric value stored in numeric_params[]
* at index
*
* When flag is cleared (is_current == true) DataManager will not attempt
* to publish parameter value to display
*/
void DataManager::clearNumericParamFlag(uint8_t index)
{
    numeric_params[index].is_current = true;
}
```

```
/**
* DataManager::clearStatusParamFlag()
*
* @param index Parameter index in status_params[] vector
*
* Clear flag associated with the status value stored in status_params[]
* at index
*
* When flag is cleared (is_current == true) DataManager will not attempt
* to publish parameter value to display
*/
void DataManager::clearStatusParamFlag(uint8_t index)
{
   status_params[index].is_current = true;
}

/**
* DataManager::pollNumericParams()
*
* Populate fresh_numeric_IDs[] and fresh_numeric_values[] vectors with
* new numeric parameter ID/value pairs to be sent to display via I2C
*
* Clear numeric parameter is_current flag for every parameter added
*/
void DataManager::pollNumericParams()
{
   for (uint8_t index = 0; index < NUMERIC_PARAM_COUNT; ++index)
   {
      if (numeric_params[index].is_current == false)
      {
         fresh_numeric_IDs.push_back(numeric_params[index].data_id);
         fresh_numeric_values.push_back(numeric_params[index].value);
         numeric_params[index].is_current = true;
      }
   }
}

/**
* DataManager::pollStatusParams()
*
* Populate fresh_status_IDs[] and fresh_status_values[] vectors with
* new status parameter ID/value pairs to be sent to display via I2C
*
* Clear numeric parameter is_current flag for every parameter added
*/
void DataManager::pollStatusParams()
```

```
{
   for (uint8_t index = 0; index < STATUS_PARAM_COUNT; ++index)
   {
      if (status_params[index].is_current == false)
      {
         fresh_status_IDs.push_back(status_params[index].data_id);
         fresh_status_values.push_back(status_params[index].status);
         status_params[index].is_current = true;
      }
   }
}

/**
* DataManager::sendNumericParamI2C()
*
* @param data_id Identifier shared by DataManager and Extruder_Display that
* indicates the source/destination of the parameter
*
* @param value Numeric parameter value to be sent to the display via I2C
*
* Convert float value to 4 integer bytes
*
* Send display I2C address followed by data_id and 4 bytes of numeric data
*/
void DataManager::sendNumericParamI2C(uint8_t data_id, float value)
{
   converter.numeric_param_input = value;

   I2C_2_IS_BUSY = true;
   I2C_2_Start();
   CORETIMER_DelayUs(5);
   I2C_2_Send_Byte(DISPLAY_I2C_ADDRESS << 1);
   CORETIMER_DelayUs(10);
   I2C_2_Send_Byte( data_id );
   CORETIMER_DelayUs(10);
   I2C_2_Send_Byte( converter.buffer[0] );
   CORETIMER_DelayUs(10);
   I2C_2_Send_Byte( converter.buffer[1] );
   CORETIMER_DelayUs(10);
   I2C_2_Send_Byte( converter.buffer[2] );
   CORETIMER_DelayUs(10);
   I2C_2_Send_Byte( converter.buffer[3] );
   CORETIMER_DelayUs(10);
   I2C_2_Stop();
   I2C_2_IS_BUSY = false;
   converter.numeric_param_input = 0;
```

```
    CORETIMER_DelayUs(100);
}

/**
* DataManager::sendStatusParamI2C()
*
* @param data_id Identifier shared by DataManager and Extruder_Display that
* indicates the source/destination of the parameter
*
* @param value Status parameter value to be sent to the display via I2C
*
* Send display I2C address followed by data_id and status which is an integer
* byte that is translated by the display into a status parameter string value
*/
void DataManager::sendStatusParamI2C(uint8_t data_id, uint8_t status)
{
    I2C_2_IS_BUSY = true;
    I2C_2_Start();
    CORETIMER_DelayUs(5);
    I2C_2_Send_Byte(DISPLAY_I2C_ADDRESS << 1);
    CORETIMER_DelayUs(10);
    I2C_2_Send_Byte( data_id );
    CORETIMER_DelayUs(10);
    I2C_2_Send_Byte( status );
    CORETIMER_DelayUs(10);
    I2C_2_Stop();
    I2C_2_IS_BUSY = false;
    CORETIMER_DelayUs(100);
}

/**
* DataManager::sendAllFreshNumericParams()
*
* Iterate through the fresh_numeric_IDs[] and fresh_numeric_values[] vectors
* and send every ID + value pair to the display via I2C
*/
void DataManager::sendAllFreshNumericParams()
{
    for (uint8_t index = 0; index < fresh_numeric_IDs.size(); index++)
    {
        sendNumericParamI2C(fresh_numeric_IDs[index], fresh_numeric_values[index]);
    }
    fresh_numeric_IDs.clear();
    fresh_numeric_values.clear();
}
```

```
/**
* DataManager::sendAllFreshStatusParams()
*
* Iterate through the fresh_status_IDs[] and fresh_status_values[] vectors
* and send every ID + value pair to the display via I2C
*/
void DataManager::sendAllFreshStatusParams()
{
   for (uint8_t index = 0; index < fresh_status_IDs.size(); index++)
   {
      sendStatusParamI2C(fresh_status_IDs[index], fresh_status_values[index]);
   }
   fresh_status_IDs.clear();
   fresh_status_values.clear();
}

/**
* DataManager::getFreshNumericIDs()
*
* @return fresh_numeric_IDs vector containing the identifiers of all new
* numeric parameter values
*/
std::vector<uint8_t>& DataManager::getFreshNumericIDs()
{
   return fresh_numeric_IDs;
}

/**
* DataManager::getFreshNumericValues()
*
* @return fresh_numeric_values vector containing every new numeric
* parameter value
*/
std::vector<float>& DataManager::getFreshNumericValues()
{
   return fresh_numeric_values;
}

/**
* DataManager::getFreshStatusIDs()
*
* @return fresh_status_IDs vector containing the identifiers of all new
* status parameters
*/
std::vector<uint8_t>& DataManager::getFreshStatusIDs()
{
```

```
   return fresh_status_IDs;
}

/**
* DataManager::getFreshStatusValues()
*
* @return fresh_status_values vector containing every new status
* parameter value
*/
std::vector<uint8_t>& DataManager::getFreshStatusValues()
{
   return fresh_status_values;
}

/*********************************End DataManager.cpp*****************************/

  /*************************************************************************
* PLA/PET Filament Extruder
* 10.22.2021
* extruder_display.ino
* User Interface Application
*    Arduino Uno
*    Adafruit HX8357
*************************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <stdint.h>
#include "Arduino.h"
#include "Adafruit_GFX.h"
#include "Adafruit_HX8357.h"
#include "TouchScreen.h"
#include "Display.h"

Display UI;

union floatToBytes
{
   uint8_t buffer[4];
   float numeric_param_input;
};

union floatToBytes converter;
uint8_t incoming_data_ID = 0;
uint8_t status_param_input = 0;
```

```
struct I2C_INT_DATA
{
   uint8_t incoming_ID = 0;
   uint8_t incoming_status = 0;
   float incoming_value = 0;
};

I2C_INT_DATA i2c_int_data;

bool received_message = false;
bool is_status = false;

void setup()
{
   Wire.begin(0x14);

   /*Event Handlers*/
   Wire.onReceive(I2C_receive_event);
   Wire.onRequest(I2C_request_event);
   Serial.begin(9600);
   byte complete = 0x07;
   UI.tft.begin();
   UI.tft.setRotation(1);
   UI.set_numeric_input_screen(UI.numeric_params, UI.desired_yield.ID);
   UI.get_numeric_user_input(UI.numeric_params, UI.desired_yield.ID);
   UI.required_input.VALUE = UI.desired_yield.VALUE * 1.11;

   UI.set_output_screen();
}

void loop()
{
   if(received_message == true)
   {
      if (is_status == true)
      {
         UI.direct_I2C_Status_Param(i2c_int_data.incoming_ID, i2c_int_data.incoming_status);
      }
      else
      {
         UI.direct_I2C_Numeric_Param(i2c_int_data.incoming_ID,
i2c_int_data.incoming_value);
      }
      i2c_int_data.incoming_ID = 0;
      i2c_int_data.incoming_status = 0;
      i2c_int_data.incoming_value = 0;
```

```
      received_message = false;
      is_status = false;
   }

   UI.poll_inputs(UI.numeric_params, NUMERIC_PARAM_COUNT);
   UI.poll_inputs(UI.status_params, STATUS_PARAM_COUNT);
}

void I2C_receive_event(int howMany)
{

   uint8_t index = 0;
   while (Wire.available())
   {
      if (index == 0)
      {
         i2c_int_data.incoming_ID = Wire.read();
         index++;
         continue;
      }
      else
      {
         if (i2c_int_data.incoming_ID > 0x0F)
         {
            i2c_int_data.incoming_status = Wire.read();
            is_status = true;
         }
         else
         {
            converter.buffer[index - 1] = Wire.read();
         }

      }
      index++;
   }
   received_message = true;
   i2c_int_data.incoming_value = converter.numeric_param_input;
   converter.numeric_param_input = 0;
}

/*********************************End extruder_display.ino*************************/

/*********************************************************************
* PLA/PET Filament Extruder
* 10.22.2021
* display_functions.h
```

```
* User Interface Application
*    Arduino Uno
*    Adafruit HX8357
**************************************************************************/
#ifndef DISPLAY_H
#define DISPLAY_H

#include <stdbool.h>
#include "Arduino.h"
#include "Adafruit_HX8357.h"
#include "Adafruit_GFX.h"
#include "TouchScreen.h"

#define TFT_CS 10
#define TFT_DC 9
#define TFT_RST 8
#define YP A2
#define XM A3
#define YM 7
#define XP 8

const unsigned char NUMERIC_PARAM_COUNT = 12;
const unsigned char STATUS_PARAM_COUNT = 4;

class Display
{
   public:
      /* hardware interface objects */
      Adafruit_HX8357 tft = Adafruit_HX8357( TFT_CS, TFT_DC, TFT_RST );
      TouchScreen ts = TouchScreen( XP, YP, XM, YM, 285 );

      struct Numeric_Param
      {
         const char* LABEL;
         const unsigned char ID;
         float VALUE;
         bool IS_CURRENT;
         const char  COLUMN;
         const unsigned char ROW;
      };
      struct Status_Param
      {
         const char* LABEL;
         const unsigned char ID;
         char* VALUE;
         bool IS_CURRENT;
```

```
    const char  COLUMN;
    const unsigned char ROW;
  };
  struct X_Y
  {
    float x = 0.0;
    float y = 0.0;
  };                                        /*____Status___________I2C Code__*/
  const char* STATUS_NONE  = "--";          /*  STATUS_NONE      0x00    */
  const char* STATUS_READY = "Ready";       /*  STATUS_READY     0x01    */
  const char* STATUS_ON  = "ON";            /*  STATUS_ON        0x02    */
  const char* STATUS_OFF = "OFF";           /*  STATUS_OFF       0x03    */
  const char* STATUS_OPEN = "Open";         /*  STATUS_OPEN      0x04    */
  const char* STATUS_CLOSED  = "Closed";    /*  STATUS_CLOSED    0x05    */
  const char* STATUS_IN_PROGRESS  = "In Progress";                  /* 0x06 */
  const char* STATUS_COMPLETE     = "Complete";                     /* 0x07 */
/*****************************************************************************/
X_Y pr;
Numeric_Param desired_yield     = { "Desired Yield (kg): ",     0,  0.0,     1, 'L', 60  };
Numeric_Param required_input    = { "Required Input (kg): ",    1,  0.0,   1,  'L',80 };
Numeric_Param ground_weight     = { "Ground Weight (kg): ",     2,  0.0, 1, 'L', 100 };
Numeric_Param zone_1_temp       = { "Zone 1: ",                 3,  0.0,         1, 'R', 100 };
Numeric_Param zone_2_temp       = { "Zone 2: ",                 4,  0.0,         1, 'R', 120 };
Numeric_Param zone_3_temp       = { "Zone 3: ",                 5,  0.0,         1, 'R', 140 };
Numeric_Param screw_speed       = { "Screw Speed: ",            6,  0.0,         1, 'R', 160 };
Numeric_Param roller_speed      = { "Roller Speed: ",           7,  0.0,         1, 'R', 180 };
Numeric_Param spooler_speed     = { "Spooler Speed: ",          8,  0.0,         1, 'R', 200 };
Numeric_Param filament_diameter = { "Diameter (mm): ",          9,  0.0,         1, 'R', 220 };
Numeric_Param extruded_length   = { "Extruded Length (m): ",    10, 0.0,         1, 'R', 240 };
Numeric_Param projected_yield   = { "Projected Yield (kg): ",   11, 0.0,         1, 'R', 60  };
Status_Param hopper_lid_status  = { "Hopper Lid Status: ", 0,  STATUS_NONE, 1, 'L', 120 };
Status_Param grinder_status     = { "Grinder (On/Off): ",  1,  STATUS_NONE, 1, 'L', 140 };
Status_Param preparation_status = { "Status: ",   2,  STATUS_NONE, 1, 'L', 40  };
Status_Param extrusion_status   = { "Status: ",               3,  STATUS_NONE, 1, 'R', 40  };
  const uint8_t DESIRED_YIELD_ID       = 0x01;
  const uint8_t REQUIRED_INPUT_ID      = 0x02;
  const uint8_t GROUND_WEIGHT_ID       = 0x03;
  const uint8_t ZONE_1_TEMP_ID         = 0x04;
  const uint8_t ZONE_2_TEMP_ID         = 0x05;
  const uint8_t ZONE_3_TEMP_ID         = 0x06;
  const uint8_t SCREW_SPEED_ID         = 0x07;
  const uint8_t ROLLER_SPEED_ID        = 0x08;
  const uint8_t SPOOLER_SPEED_ID       = 0x09;
  const uint8_t DIAMETER_ID            = 0x0A;
  const uint8_t EXTRUDED_LENGTH_ID     = 0x0B;
  const uint8_t PROJECTED_YIELD_ID     = 0x0C;
```

```
    const uint8_t HOPPER_LID_STATUS_ID    = 0x10;
    const uint8_t GRINDER_ON_OFF_ID       = 0x20;
    const uint8_t PREPARATION_STATUS_ID   = 0x30;
    const uint8_t EXTRUSION_STATUS_ID     = 0x40;

/*****************************************************************************/
    Numeric_Param* numeric_params[NUMERIC_PARAM_COUNT] = { &desired_yield,
&required_input, &ground_weight, &zone_1_temp, &zone_2_temp, &zone_3_temp,
&screw_speed, &roller_speed, &spooler_speed, &filament_diameter, &extruded_length,
&projected_yield };
    Status_Param* status_params[STATUS_PARAM_COUNT]    = { &hopper_lid_status,
&grinder_status, &preparation_status, &extrusion_status };

/*****************************************************************************/
    template <class T> void set_label_and_value(T Params_Array, unsigned char label_type);
    template <class T> void update_output(T Params_Array, unsigned char ID);
    template <class T> void poll_inputs(T Params_Array, unsigned char SIZE);
    template <class T> void set_numeric_input_screen(T Params_Array, const unsigned char
ID);
    template <class T> void get_numeric_user_input(T Params_Array, const unsigned char ID);
    void direct_I2C_Numeric_Param(uint8_t data_ID, float value);
    void direct_I2C_Status_Param(uint8_t data_ID, uint8_t status);
    void set_text(unsigned char S, unsigned short C);
    void set_new_numeric_value(float new_value, unsigned char ID);
    void set_new_status_value(char* new_value, unsigned char ID);
    void set_default_background();
    void set_output_screen();
};

/* set_label_and_value() */
template <class T> void Display::set_label_and_value (T Params_Array, unsigned char
label_type)
{
  unsigned short text_color = 0;
  unsigned char item_count = 0;
  unsigned short label_cursor = 0;
  unsigned short value_cursor = 130;

  set_text(1, HX8357_CYAN);
  tft.setCursor(0, 20);
  tft.print("Preparation Stage");
  tft.setCursor(235, 20);
  tft.print("Extrusion Stage");
  tft.setCursor(235, 80);
  tft.print("Temperatures");
  if (label_type == 0) { item_count = NUMERIC_PARAM_COUNT; }
```

```
   else { item_count = STATUS_PARAM_COUNT; }
   set_text(1,HX8357_WHITE);
   for (int ID = 0; ID < item_count; ID++)
   {
      if (Params_Array[ID]->COLUMN == 'R') { label_cursor = 235; value_cursor = 385; }
      tft.setCursor(label_cursor, Params_Array[ID]->ROW);
      tft.print(Params_Array[ID]->LABEL);
      tft.setCursor(value_cursor, Params_Array[ID]->ROW);
      tft.print(Params_Array[ID]->VALUE);
   }
}
/* END set_label_and_value() */

/* update_output() */
template <class T> void Display::update_output( T Params_Array, unsigned char ID )
{
   unsigned short value_cursor = 130;
   if (Params_Array[ID]->COLUMN == 'R') { value_cursor = 385; }
   set_text(1, HX8357_WHITE);
   tft.fillRect(value_cursor, Params_Array[ID]->ROW, 95, 15, HX8357_BLACK);
   tft.setCursor(value_cursor, Params_Array[ID]->ROW);
   tft.print(Params_Array[ID]->VALUE);
   Params_Array[ID]->IS_CURRENT = true;
}
/* END update_output() */

/* poll_inputs() */
template <class T> void Display::poll_inputs(T Params_Array, unsigned char SIZE)
{
   for (int ID = 0; ID < SIZE; ID++)
      if (Params_Array[ID]->IS_CURRENT == false) { update_output(Params_Array, ID); }
}
/* END poll_inputs() */

/* set_numeric_input_screen() */
template <class T> void Display::set_numeric_input_screen(T Params_Array, const unsigned
char ID)
{
   unsigned short x_cursor = 42;
   unsigned char row_1_item, row_2_item;
   unsigned short bottom_label_positions[] = { 5, 115, 235, 310 };
   char *bottom_labels[]                   = { "Go Back", "Clear", ".", "Enter" };
   set_default_background();
   tft.setCursor(0, 20);
   set_text(2, HX8357_WHITE);
   tft.print(Params_Array[ID]->LABEL);
```

```
    // horizontal lines
    for (unsigned short n = 0; n < 3; n++)
        tft.drawLine(0, 50 + (n * 90), 480, 50 + (n * 90), HX8357_WHITE);
    for (unsigned char n = 0; n < 4; n++)
        tft.drawLine(96 * (n + 1), 50, 96 * (n + 1), 320, HX8357_WHITE);
    for (unsigned char i = 0; i < 5; i++)
    {
        tft.setCursor(x_cursor, 85);
        tft.print(i + 1);
        tft.setCursor(x_cursor, 175);
        if (i != 4) { tft.print(i + 6); }
        else { tft.print(0); }
        x_cursor += 96;
    }
    for (unsigned char i = 0; i < 4; i++)
    {
        tft.setCursor(bottom_label_positions[i], 265);
        tft.print(bottom_labels[i]);
    }
}
/* END set_numeric_input_screen() */

/* get_numeric_user_input() */
template <class T> void Display::get_numeric_user_input(T Params_Array, const unsigned char
ID)
{
    unsigned char row = 0;
    unsigned char col = 0;
    unsigned char index = 0;
    unsigned char decimal_index = 0;
    unsigned short sum = 0;
    float input_value = 0.0;
    char button = 'n';
    const char *buttons[] = { "12345", "67890", "bc.en" };
    unsigned short text_box_cursor_x = 230;
    set_text(2, HX8357_WHITE);

    while (1)
    {
        TSPoint p = ts.getPoint();
        delay(300);
        if (p.z > 3)
        {
            pr.x = (float)p.y * (480.0 / 1023.0);
            pr.y = 320.0 - ((float)p.x * (320.0 / 1023.0));
            if      (pr.x < (96 - 2)) { col = 0; }
```

```
        else if (pr.x > (96 + 2) && pr.x < ((96 * 2) - 2)) { col = 1; }
        else if (pr.x > ((96 * 2) + 2) && pr.x < ((96 * 3) - 2)) { col = 2; }
        else if (pr.x > ((96 * 3) + 2) && pr.x < ((96 * 4) - 2)) { col = 3; }
        else if (pr.x > ((96 * 4) + 2) && pr.x < 480) { col = 4; }
        else    { col = 1000; }
        if      (pr.y > (50 + 2)  && pr.y < (50 + 90 - 2)) { row = 0; }
        else if (pr.y > (50 + 90 + 2) && pr.y < (50 + (90 * 2) - 2)) { row = 1; }
        else if (pr.y > (50 + (90 * 2) + 2) && pr.y < 320) { row = 2; }
        else    { row = 1000; }
        pr.x = 1000;
        pr.y = 1000;
        button = buttons[row][col];
        if (isDigit(button) || button == '.')
        {
           text_box_cursor_x += 12;
           tft.setCursor(text_box_cursor_x, 20);
           tft.print(button);
           if (isDigit(button))
              sum = (sum * 10) + (button - '0');
           else
              decimal_index = index;
           index++;
        }
        else if (button == 'b')
           return;
        else if (button == 'c')
        {
           tft.fillRect(230, 20, 300, 20, HX8357_BLACK);
           text_box_cursor_x = 230;
           button = 'n';
           sum = 0;
           index = 0;
        }
        else if (button == 'e')
           break;
      }
   }
   input_value = sum / pow(10, (index - decimal_index - 1));
   set_new_numeric_value(input_value, ID);
}
/* END get_numeric_user_input() */

#endif

/*****************************End display_functions.h*****************************/
```

```
/**************************************************************************
* PLA/PET Filament Extruder
* 10.22.2021
* display_functions.cpp
* User Interface Application
*    Arduino Uno
*    Adafruit HX8357
**************************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <math.h>
#include "Arduino.h"
#include "Adafruit_HX8357.h"
#include "Adafruit_GFX.h"
#include "TouchScreen.h"
#include "Display.h"

void Display::direct_I2C_Numeric_Param(uint8_t data_ID, float value)
{
   bool is_status_input = false;
   if (data_ID > 0x0F)
      is_status_input = true;
   unsigned char ID = 0;
   switch(data_ID)
   {
      case 0x01:
         ID = 0;
         break;
      case 0x02:
         ID = 1;
         break;
      case 0x03:
         ID = 2;
         break;
      case 0x04:
         ID = 3;
         break;
      case 0x05:
         ID = 4;
         break;
      case 0x06:
         ID = 5;
         break;
      case 0x07:
         ID = 6;
```

```
            break;
        case 0x08:
            ID = 7;
            break;
        case 0x09:
            ID = 8;
            break;
        case 0x0A:
            ID = 9;
            break;
        case 0x0B:
            ID = 10;
            break;
        case 0x0C:
            ID = 11;
            break;
        default:
            break;
    }
    set_new_numeric_value(value, ID);
}


void Display::direct_I2C_Status_Param(uint8_t data_ID, uint8_t status_ID)
{
    unsigned char ID = 0;
    char* status_value;
    switch(data_ID)
    {
        case 0x10:
            ID = 0;
            break;
        case 0x20:
            ID = 1;
            break;
        case 0x30:
            ID = 2;
            break;
        case 0x40:
            ID = 3;
            break;
        default:
            break;
    }

    switch(status_ID)
```

```
    {
      case 0x00:
        status_value = STATUS_NONE;
        break;
      case 0x01:
        status_value = STATUS_READY;
        break;
      case 0x02:
        status_value = STATUS_ON;
        break;
      case 0x03:
        status_value = STATUS_OFF;
        break;
      case 0x04:
        status_value = STATUS_OPEN;
        break;
      case 0x05:
        status_value = STATUS_CLOSED;
        break;
      case 0x06:
        status_value = STATUS_IN_PROGRESS;
        break;
      case 0x07:
        status_value = STATUS_COMPLETE;
        break;
      default:
        break;
    }
    set_new_status_value(status_value, ID);
}

/* set_text() */
void Display::set_text(unsigned char S, unsigned short C)
{
    tft.setTextSize(S);
    tft.setTextColor(C);
}
/* END set_text() */

/* set_default_background() */
void Display::set_default_background()
{
    tft.fillScreen(HX8357_BLACK);
    set_text(2, HX8357_MAGENTA);
    tft.setCursor(0, 0);
    tft.print("PLA/PET Filament Extruder");
```

```
}
/* END set_default_background() */

/* set_output_screen() */
void Display::set_output_screen()
{
   set_default_background();
   tft.drawLine(230, 20, 230, 320, HX8357_WHITE);
   set_label_and_value(numeric_params, 0);
   set_label_and_value(status_params, 1);
}
/* END set_output_screen() */

/* set_new_numeric_value() */
void Display::set_new_numeric_value(float new_value, unsigned char ID)
{
   if (new_value != numeric_params[ID]->VALUE)
   {
      numeric_params[ID]->VALUE = new_value;
      numeric_params[ID]->IS_CURRENT = false;
   }
}
/* END set_new_numeric_value() */

/* set_new_status_value() */
void Display::set_new_status_value(char* new_value, unsigned char ID)
{
   if (new_value != status_params[ID]->VALUE)
   {
      status_params[ID]->VALUE = new_value;
      status_params[ID]->IS_CURRENT = false;
   }
}
/* END set_new_status_value() */

/*****************************End display_functions.cpp*************************/
```