UCF

COLLEGE OF ELECTRICAL ENGINEERING

AND COMPUTER SCIENCE

ELECTRONIC LEGO SORTER

Group 33

Nike Adeyemi

David Carey

Katrina Little

Nickolas Steinman

Senior Design 1 - Fall 2014

TABLE OF CONTENTS

1.0 Executive Summary

LEGO has become a household name – you can ask just about anyone if they have ever played with LEGO's as a child and the answer will most likely be yes. Not only are LEGO's a toy, they also help children develop important skills such as problem solving, organization, and planning through construction (among many other critical development skills). LEGO has evolved tremendously through the years. They have even developed LEGO kits for adults such as the "Mindstorms NXT" which includes a programmable robotics kit. There are really only two drawbacks when it comes to LEGO's. The first drawback is the price. The second drawback is for the true LEGO collectors. If you collect LEGO's for many years, you wind up with many pieces and it is very difficult to sort through 100 pounds of bricks to replicate one of the original LEGO "sets" that you purchased a long time ago. This project is being designed to tackle both of these problems.

2.0 Project Description

This section outlines the motivation, goals & objectives, and team member responsibilities of the project.

2.1 Project Motivation

More than 400 billion LEGO bricks have been produced since 1949. LEGO bricks are available in 53 different colors and 19 million LEGO elements are produced every year. There are more than 8 quadrillion (8,181,068,395,500,000) possible combinations of minifigures that can be made using all of the unique minifigure parts over the last 30 years. The motivation for an electronic LEGO sorter is extremely significant. There are many blogs online of LEGO fanatics with approaches to sort LEGO's using home storage solutions to tackle the organization problems associated with LEGO's. Unfortunately, most LEGO collectors do not have the technical knowledge to build a sorter to solve this problem. The motivation behind this project is to tackle the two main difficulties with LEGO's. The first shortcoming being the price and the second being the headache of sorting LEGO parts by hand.

Figures 2.1-A and 2.1-B illustrate a before and after depiction to the LEGO sorting process. Even after spending hours hand sorting an entire LEGO collection you can easily see there is still a lot of room for improvement. Hand sorting 20 pounds of LEGO's takes about 8 hours and by the end of it all the small parts get thrown into a bin unsorted due to exhaustion.

Figure 2.1-A : Huge Collection of Unsorted LEGO

Figure 2.1-B: Huge collection of LEGO's that have been hand sorted

Currently, there is only 1 type of LEGO sorting element that you can purchase. These plastic sorters consist of four tiers, each tier having different sized slots. The figure below easily sums up the problem with this method to LEGO sorting.

Figure 2.1-C: Multi-Tiered LEGO sorting element

The goal of this project is to sort through large quantities of LEGO's autonomously (meaning you turn on the sorter and leave it alone). You may wonder how this tackles the first drawback to LEGO's which is the price. Some LEGO "Star Wars" sets sell for thousands of dollars. You can buy a 20 pound lot of LEGO's for around $200 (give or take). So, the idea is that if you buy large bulk lots of LEGO's you can use this "Electronic LEGO Sorter" to slowly piece together these more expensive sets slowly over time as a more affordable option to buying a new set in stores. This project aims to tackle both problems of price and inconvenience.

2.2 Goals and Objectives

The goal of this project is to sort a set library of LEGO® bricks as accurately as possible by color or by shape, which is decided by the user using a touch screen. This way if the user wants to further sort the bricks such as by color and shape they would just remove the bin they wish to sort and put it back in for further sorting. Because of the variety of LEGO®s that are out in the market sorting through them to get the exact part needed becomes very time consuming. So for hobbyist who cannot find a certain part or sellers who need an efficient way to find and catalogue their LEGO® parts this project could be the solution to relieve this problem. The user needs is a Library of the different LEGO® parts and colors.

The main objective of this project is accuracy therefore speed was not as important of a factor for this LEGO® sorter. With the library loaded in both the AtMega2560 and the BeagleBoneBlack, the user will use the touch screen to specify how to sort the bricks whether it is by color or by shape. Then the user decide which of the seven containers the

blocks will go in with one of the containers being used to collect any errors or bricks the image processor could not determine. The user can assign multiple buckets to hold the same types and there exists common colors like yellows, whites, blues, etc. The bricks to be sorted will be loaded into a container with a lift system built in. The lift will carry a hand full of LEGO® parts up and dump them on a two-stage conveyor belt. The conveyor belts are used to space out the parts before they enter the image processor. Setting the first stage belt to a slow speed and the second stage belt to a higher speed does this. The bricks then slowly drop to the faster belt spacing the bricks apart. Each brick enters the image processor ideally one by one. Once a LEGO® enters the processor the conveyor belt stops and the image processor system becomes active. Depending on what the user decided how to sort, the image processor either check color and compare it to some color constants or using a mirror, edge detection algorithm, and other algorithms to compare the given part to the parts in the library in memory to decide what kind of part it is. Once the processor is done and decides which bucket to put it and rotates the arm to the correct position. The sorter then pushes the part into a chute, which will guide it to the appropriate container. The process will continue until all parts have been dealt with. All these processes will be done using the AtMega32u4 for motor, lift, and LCD control and a Beagle Bone Black to act as an image processor.

2.3 Team Member Responsibilities

The group involved in the project consists of two Electrical Engineers and two Computer Engineers. This helps to determine a natural way to divide the work within the project. The Electrical Engineers were able to focus more on the hardware side of the project, dealing with the circuit boards and power, while the Computer Engineers were devoting the majority of their time to the software of the project, coding everything up and making sure all of the parts are communicating properly. Beyond that, each person has an even more specific role that they play within the project.

Nike Adeyemi (Computer Engineering)
Nike's focus in the project will was the User Interface. Her role was to build a user interface that will be quick to learn, easy to understand, and very effective in aiding the user in setting up the organization of the project in exactly the way that the user desires.

David Carey (Computer Engineering)
David's primary area of focus for the project was on the image processing portion of the sorter. His role was to make sure that the Legos are properly analyzed and then sorted correctly based upon their color shape and size.

Katrina Little (Electrical Engineering)

Katrina's primary area was that of the power supply, lift arm subsystem, and interfacing with the Atmega and BeagleBone. She designed the source of power for the entire project to ensure that the sorter receives enough power to be able to function properly, without receiving so much power that it becomes more harm to the system than good. Also she is the one in charge of making sure that communications are working properly between the brains of the project and the mechanical parts.

Nick Steinmann (Electrical Engineering)
Nick's primary roles included setting up the Conveyor Belt and Rotating Arm systems as well as setting up the PCB for the Atmega. The Atmega chip was utilized on a custom PCB board built by Nick for this project.

3.0 Related Research

This next section will be devoted to showing the research done in different areas that ultimately led to the decisions that were made regarding the Lego sorter.

3.1 Existing Projects and Products

The first place that research for the project was started was with similar projects and products. This includes other Lego sorters as well as other projects that used components similar to those being used in the Lego sorter.

3.1.1 Existing (Electronic) LEGO Sorters

When choosing this project it was already known that there were already a handful of sorters that organize blocks and other various LEGO parts. Many of those were created using the Mindstorm NXT LEGO smart brick as the main controller. The sorters that we focused on were those that could sort LEGO bricks and other parts and of those sorts they were organized into three categories: Sorters that sort only by Size and shape, Sorters that organize by color, and sorters that are organized by some combination of the previously stated types.

There were sorters we saw organized only specific parts by size or length for example is a sorter that only sorts lift arms, created by a user on YouTube known as Akiyuky, and uses a simple system to process them. First the sorter will orient the parts a certain way to enter the system for example the sorter in figure 3.1.1 – a used two lifts that alternate going up a down to change the lift arm orientation and to load the lift arm onto the

conveyor belt. The sort then makes sure the that the parts only enter one at a time and back to back and any part that ends up onto of the other or does not fit in the entryway is pushed into a chute and back into the loading bucket. The lift arms that made it into the sorter then travel across the conveyor belt until it slides into the first hole that is big enough to fall in the hole getting bigger the longer it travels and into the correct containers.

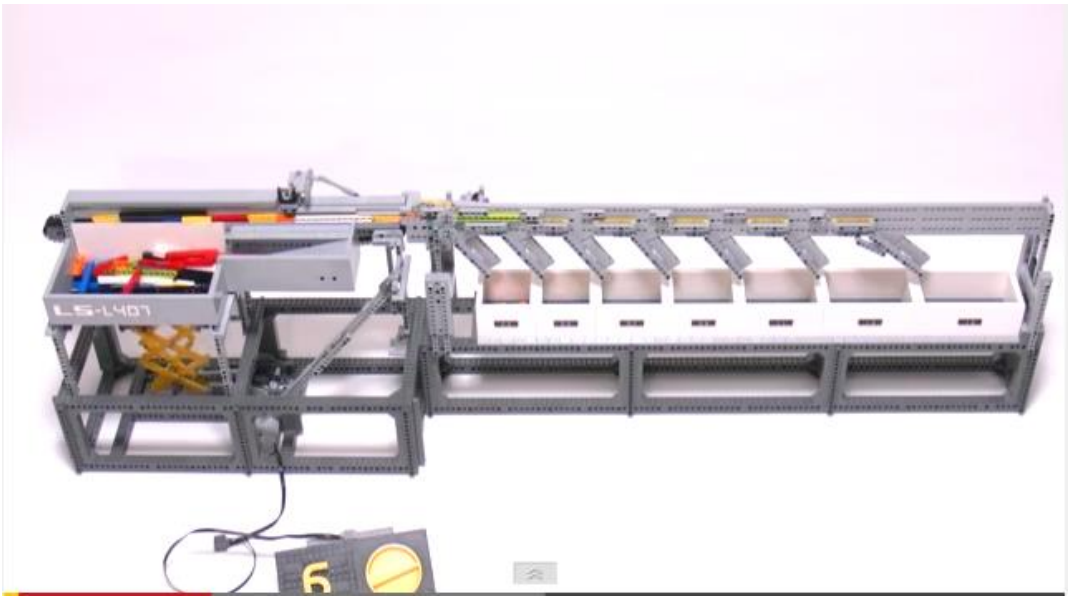


Figure 3.1.1-A LEGO Lift Arm Sorter

There is also another sorter that is able to separates a wider variety of bricks both using a similar set up. The same YouTube user Akiyuky created this sorter as well using image processing as a means to sort the parts. In his sorter, the parts entered the sorter with a lift arm by the hand full and carried though the two-stage conveyor belt. The two conveyor belts are used for spacing out the parts so they can enter the image processor one by one and sorted into the appropriate bucket. This being a good start but they only sort by shape and the sorter that we build must also sort by color.

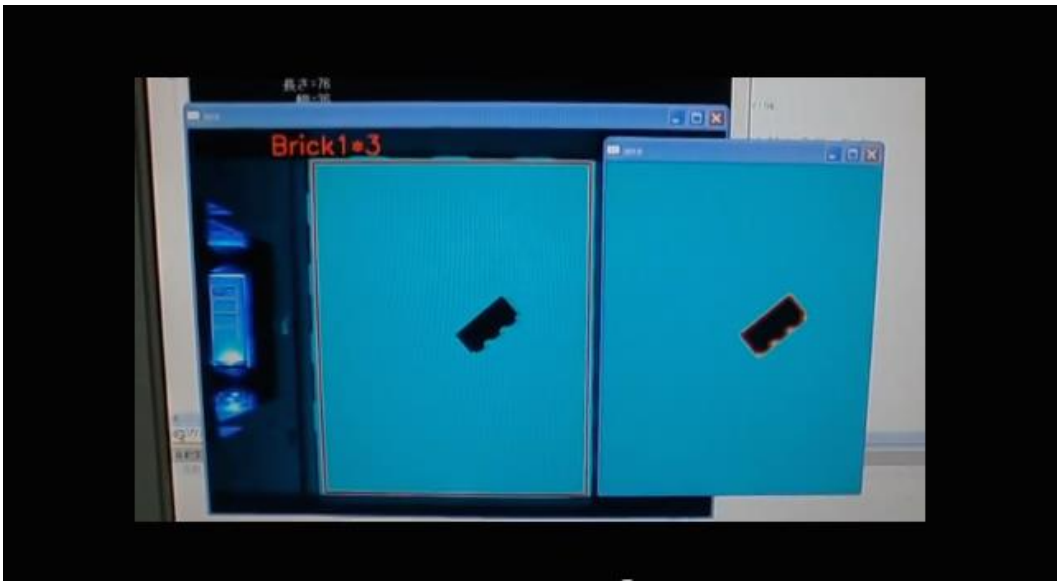


Figure 3.1.1-B Image recognition screen

There also LEGO sorters that organized by color. It goes through a similar process, as the part sorter only that a color sensor is needed to detect the brick color. However many of the sorts seen using this method was only using the same brick type, as color was the main focus of their machine.

Finally there were sorters that are able to sort both by shape and color and are what we want for our own sorter. Of the ones that were seen they were simple as that only used simple bricks like the 2 x 2 and 2 x 4 bricks and sort them. It makes sense though as sorting a variety of parts with a variety of colors will need a large amount of containers to use. While our sorter will sort through many parts there is only a limited amount of container to use.

So with everything considered Akiyuky's model is the closest to the model we had in mind. It was mainly because of its use of image processing. Because of this it can easily be adjusted in order to sort by shapes in one setting and sort by color in a different setting. This essentially turns our sorter into a hybrid sorter as in order to sort by color and shape the user will just have to go one pass to sort by color or shape and a second pass with the opposite settings.

3.1.2 Projects with Relevant Components

For our sorter the design will require use of image processing, a conveyor system, and a loading system. There are a wide amount of parts to choose from so to narrow down our choices we looked at other senior design project and what parts they used for their project that relates to our project by providing similar need to what we need.

T-100 Watchdog - The T-100 Watchdog was an earlier senior design project in spring 2014 in UCF. It was an all-terrain vehicle that uses computer vision and autonomy to track and pursue a target based on the heat signature. Using the web camera and thermal camera, the image processor it is able to track any movement that has occurred, acquire the target that was the source of that movement, and track that target all in real time.

The team was able to program these functionalities using OpenCV an open source computer vision library running in a Linux based operating system as OpenCV has the majority of the algorithms needed for their task. All the image processing was done through the BeagleBone Black as it acted as the main controller for all the subsystems that were running in this vehicle.

For the motors each motor that controlled each wheel individually were controlled by separate microcontrollers each of them connected to the main controller the BeagleBone black.

Solar Tracker – The solar tracker is another UCF Senior design project. The purpose of this project is to use a mobile application to control the solar tracker to convert the solar energy in order to provide power to other devices. Since they are dealing with solar tracking it is important that the solar panels are facing the sun to get maximum energy input. Therefore motors are needed to rotate the panels and are program to rotate to the right orientation depending on the suns location. Here they considered using motor controllers as opposed to microcontrollers believing that just only using microcontrollers might not have enough power to drive the motors. When used in conjunction with the microcontroller the motor controllers just receive information from the microcontroller on how to run with just a few inputs. They use DC motors to rotate the panels and the servomotor to determine how much the motor should rotate. The LCD screen they had was just a simple 2 x 16 screen that is linked the microcontroller used to monitor the status of the device.

3.2 Initial Project Considerations

This section will cover existing projects with similar components to be used in this project, as well as similar LEGO sorters, subsystem designs that were rejected, and finally subsystem specific research.

There is no "one size fits all" on how one sorts their LEGO collection. Some people sort by color, others by shape and size, and others try to sort their collection by sets. It all depends on what you plan TO DO with your LEGO's.
Some parents buy new sets for their children because they want their children to develop engineering skills by following a specific detailed instruction manual to build a project. Also, many adults stick solely to "Star Wars" collections (among others). These groups of people would require a sorter to sort their pieces specifically by sets.
"Lego artists" are more inclined to sort their bricks by color and shape. Another thing to consider is that many people have such large collections of LEGO's that sorting just the "bricks" and "plates" by color separates the collection enough to find the other smaller parts that are needed. This is because the "bricks" and "plates" make up the majority of most sets.

3.2.1 Lego Separation, Transportation, & Identification

There are countless ways to tackle the sorting issue. This section describes initial considerations for the transportation, separation, and identification components of the project design.

One of the ideas considered was dumping the parts into a large bucket that had a camera inside. The camera would photo the parts as they fell through the bucket. As the parts fell there would be fans to blow the parts into a specific section for sorting. This idea was too complicated mechanically and was therefore ruled out.

Another consideration was to pre-sort the parts using a sifting device such as the one shown in figure 2.1- c (above) before reaching a camera for image recognition. This was not ideal due to the huge variety of LEGO parts.

The use of a conveyor belt system using sensors and fans was also considered. The parts would pass a series of sensors and fans situated on the sides of the conveyor belt. If the sensor was tripped the data would be sent to the microcontroller. The microcontroller would turn on the fan and blow the part off of the belt into the appropriate sorted bin of similar parts. This would be done using color sensors. This would sort parts by color but not by size or shape. This idea was ultimately ruled out because LEGO's are made from

plastic which limits the type of sensor that could be used to detect the size of parts that pass by. The only sensors that can detect plastic are capacitive sensors. Capacitive sensors are not as readily available as Inductive sensors (which are used to detect metal). Capacitive sensors are more so used to detect different types of plastic materials (not shape or size) which is needed for the application of this project. The other issue with this method is that if many parts pass by the sensors (as well as fans) at the same time they will all be blown into the bins and ultimately the sorting process is destroyed.

3.2.2 Lego Sorting

After deciding specifically which Legos will be sorted, the next issue to arise was exactly how to separate them into different receptacles.

One of the first ideas that was discovered in research was a sorter that was able to separate parts using different sized gates to catch the various Legos as they were transported down the conveyor belt. While it would allow for a bit more speed in the separation, it only accounts for a specific Lego type. With multiple different types of Legos being introduced to the sorter all at the same time, simple gates would end up putting Legos of multiple different types into the same receptacles.

The next idea was that of dropping the Legos down into waiting receptacles. So essentially we would have a set of buckets waiting below the end of the conveyor belt which would dispense the most recently identified Lego into the proper bucket. This became the idea that was ultimately settled upon. With the method of using image processing to determine the type of Lego, the difficult part was trying to find a way to cycle through the different available buckets in order to separate the Legos.

When it came to the orientation of the buckets, two different options were presented: a circular arrangement, and a linear arrangement. The linear arrangement would essentially line up the buckets underneath the conveyor belts. It would then extend or retract the set of buckets so that the bucket used for the current Lego would sit underneath the conveyor. This option has the advantage of being an option that the space the sorter would occupy would be minimalized a bit……. Or at least that would be the case when the sorter is not active. That's where one of the major downsides occurs. The entire idea for the sorter is that the user does not have to be actively involved after initially starting it, therefore the user would not be totally aware of when the buckets would extend from underneath the conveyors. This could cause a potential safety hazard for the user. It also creates a requirement that extra space needs to be accounted for when setting up the sorter, otherwise the bucket system could end up running into a wall. Another issue to consider is the weight of the Legos. After a rather long session of sorting, the buckets may begin to contain rather large amounts of Legos that could cause

more stress on the motors involved in the system. While this wouldn't be an immediate issue, it could cause fairly extensive wear and tear to the system.

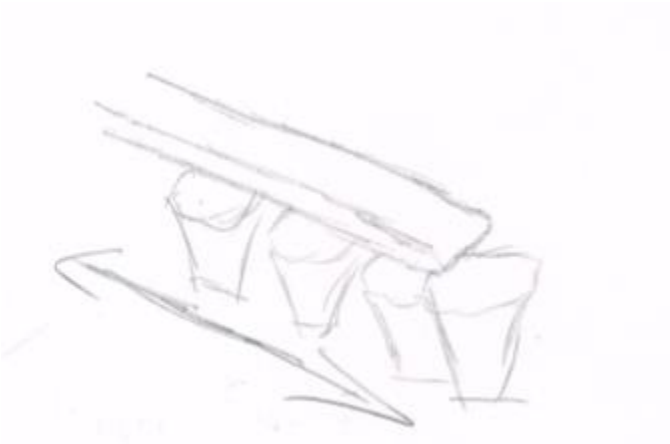


Figure 3.2.2-A Initial drawing of a linear bucket system

In the case of the circular configuration, the more major issue of the linear system is accounted for. In a circular configuration, the bucket setup, while still able to move, would never take up unexpected space. This eliminates the safety hazard of the unexpected extension of the linear system. However, the issue of extensive wear and tear of the motor would still occur if the buckets were to rotate. So the ideal situation for the sorter would be a circular configuration for the buckets in which the buckets would not have to be the subject of the rotation. That is when the idea for a rotating arm was proposed.

In the case of the rotating arm, the buckets would be set up in a static circular configuration still, but rather than the conveyor belt dropping the Legos off into the bucket below, it would drop them into a very simple slide that would rotate between the buckets as necessary. The arm would first rotate to the desired receptacle, and then the conveyor would drop off the next available Lego onto the slide. This would ensure that the amount of weight handled by the motors would only ever handle the weight of the arm itself and nothing more. This way the motors would be able to last for much longer.

Ultimately, this rotational arm system is what was settled upon. This was the safest, most efficient way to sort out the Legos while causing as little wear and tear to the motors as possible.

3.2.3 User Input

One of the most important considerations that needed to be made concerning the Lego sorter was that of the user input. Without a proper system for user input there is no way

to manage the Legos in a way that's truly useful. The Lego sorter needs to be able to accommodate specifications specific to what the user has in mind, and, more importantly, it needs to be simple to use and understand. The difficult part is deciding what the best way to implement a user input system would be.

One of the first ideas that was brought up was the use of a mobile app. Smartphones are everywhere and it would make a good deal of sense to simply make the user interface into a mobile app that the user could carry with them everywhere, allowing them to simply connect wirelessly to their sorter. While at first glance it seems to be a good idea, there are a couple of factors that set this idea back a bit. The first immediately recognizable downside to using a mobile app is the screen size for the interface. The goal of the user interface is to create something that a first time user would be able to approach for the first time and be able to use with relative ease. Restricting the screen size to that of mobile devices would be fairly restricting to the abilities of the user interface in that cramming an efficient and easy-to-use interface onto a small screen would be fairly difficult. Another downside would be that using a mobile app would require multiple versions. With both iOS and Android being the primary competitors in the mobile market, it would require at least two versions of the app. And even after those two have been covered, it still limits our user audience. The thought behind the Lego sorter is that ANYONE would be able to approach it and use it, so creating the user interface exclusively for mobile devices would NOT be ideal.

So if a mobile application isn't the answer, then the next logical step is a simple built-in interface. But now this raises the question as how to narrow this down even further. Would a keyboard and a screen be suitable, or would a touch of some sort make more sense? In the case of a buttons and a screen, there are so many factors that would have to be accounted for. It could be trickier to navigate from one screen to another or to assign different parts to different buckets. Overall it just doesn't seem to be practical. It would essentially have to be a full computer setup complete with a mouse to be a truly easy-to-use system. However, a touch screen is entirely reasonable. The Legos that are being sorted will already be programmed into the system, so little typing will be needed, and touch screens have become such a basic part of everyday life that it would be easily recognizable by virtually everyone, including children. This settles the goal of securing a system that would be usable by users of virtually every age. It creates a simple environment with very little learning curve for the user which makes the system very approachable.

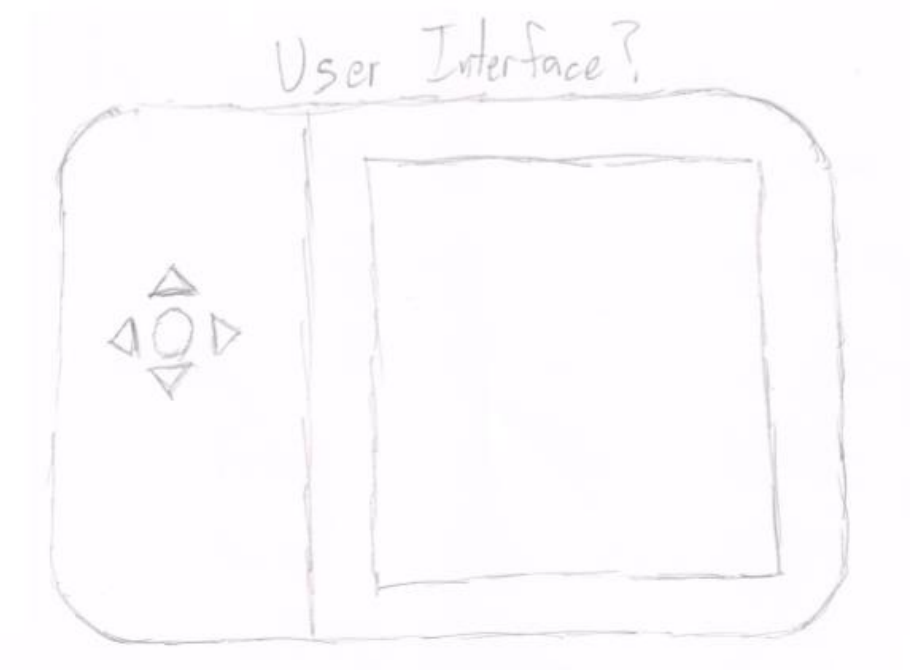


Figure 3.2.3-A Initial user interface considerations with buttons

While there are a number of different ways to approach the user interface, the initial goals of the project create limitations that ultimately lead to a built-in touch screen interface being the most ideal selection for the sorter. Users from ages 5 to 85 are able to simply approach the system and learn how to use it with incredibly little difficulty.

3.2.4 Control

This project requires a central control unit to respond to and control each subsystem. Figure 3.2.4-A shows a rough data-flow diagram of the subsystems that are connected to the main controller. The arrows indicate the direction of the data flow.

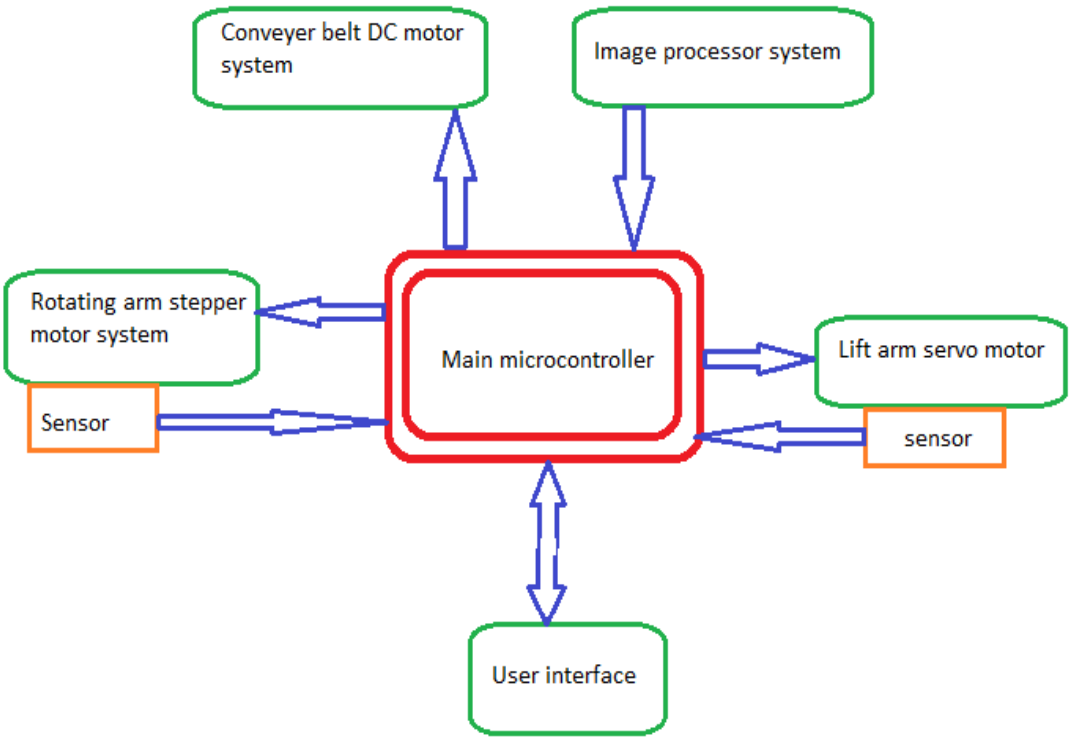


Figure 3.2.4-A: Block diagram of control

For the conveyer belt subsystem, the central control unit can turn off and on the DC conveyer belt motors as well as vary the DC motor speeds. Pulse width modulation (PWM) from the main controller is used to vary the speeds. Each of the two conveyer belt DC motors take one I/O port from the main controller.

The rotating arm subsystem utilizes a stepper motor and a feedback sensor to rotate towards the correct bucket for sorting. The main controller controls the rotation of the motor through a stepper motor driver circuit and receives feedback from the sensor to make sure the arm is in the correct position. The stepper motor uses 4 digital control lines and the sensor uses one ADC channel from the main microcontroller. The lift arm subsystem utilizes a DC motor to lift the LEGOs onto the belt and two boundary switches. The main controller controls this motor and receives the boundary switches as inputs. The upper switch tells the main controller to stop moving the lift arm up when it is hit. The same process happens for the lower switch when the lift system is lowering itself. Each switch uses one digital I/O line from the MCU. The main controller also transmits and receives data from the user control subsystem. This system is a touchscreen with an LCD and resistive touch overlay. The main controller can render simple graphics on the LCD as well as receive the touch coordinates signal from the touch overlay. The image processing subsystem communicates with the central controller via UART. The

image processor identifies the LEGO piece and sends the data to the main controller. This data lets the main controller know which type LEGO piece was identified with the image processor. The main controller then tells the rotating arm to move to the appropriate location. The MCU also has UART communication pins.

3.2.5 Image Processing

One of the ways discussed to sort the LEGO pieces is through the use of image processing. Image processing is a way to take a picture or video frame input from a camera device and analyze it with algorithms.

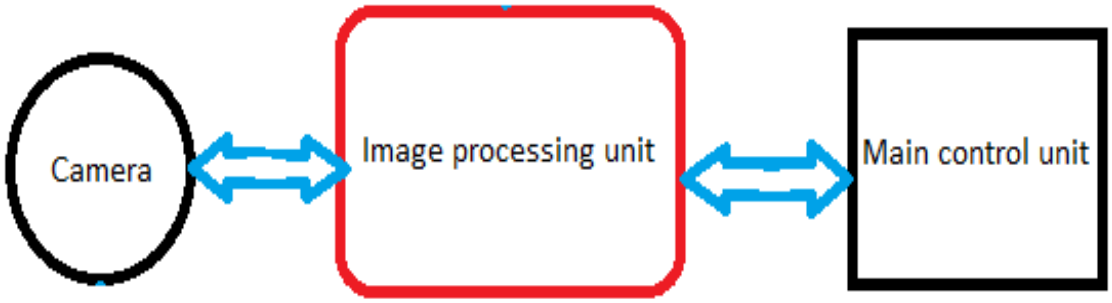


Figure 3.2.5-A

Figure 3.2.5-A above shows the general data flow diagram for the image processing subsystem. The image processor operates on the input images received from the camera. The processor then runs algorithms on the image and relays information about it to the main controller. For this project, LEGO pieces are separated based on their shape and color attributes. The color identification is fairly straightforward. The camera inputs its image and the processing device identifies the color based on the pixel information from the camera sensor. Shape identification is a more complicated task. The image processor needs to perform more complicated algorithms like edge detection to process the shape seen by the camera sensor. A major issue to consider is that the LEGO pieces will enter the camera's field of view at arbitrary angles and orientations on the conveyer belt. A library of Lego shapes is used to compare against the Lego in question. Once identified, the information is then sent to the main microcontroller to have the piece be transported into the correct bucket. One of the objectives of this project is for it to be a completely embedded solution. This means that it should not have to be hooked up to a laptop or outside computer to function. Image processing is a fairly computationally intensive procedure and consideration was taken to pick a capable platform. Technologies initially considered were; microcontroller, microcomputer, digital signal processor (DSP) development board, and field programmable gate array (FPGA) development board. Detailed research on the various technologies capable of image processing was done. The

image processor unit interfaces with a camera sensor to identify the LEGO pieces. Interfacing is done through a USB communication port. The image processor communicates with the main MCU via a two wire UART connection.

For this project, the image processing can be done on either an image taken every few seconds or on a constant stream of video input. The method of receiving a snapshot every few seconds vastly decreases the processing demands from the image processing unit but would require additional sensors to tell the camera when to take a picture. Each time a LEGO piece enters the camera's field of view, it must be completely within the view without any of it sticking out. Because there are many different shapes and orientations the LEGO pieces can arrive, the sensors may not be a great solution for consistent results. Alternatively, the image processing unit can take in a video stream, frame by frame. This is much more demanding of the processing unit, but has the advantage of not needing extra sensors to detect whether the LEGO pieces are completely within the camera's field of view. An edge-detect algorithm can run on the processor to determine that the piece's outline is entirely within the image. This method is much more robust than the snapshot method as it does not rely on outside sensors to determine what is in its field of view.

3.2.6 How to Gather the Image

For the image processing chamber that is used within the project, the most immediate thing that had to be addressed was what method would be used to gather the image that would ultimately be analyzed. It was very clear from the beginning that there would somehow need to be more than one angle of view on the Lego in question in order to gather all of the necessary details for analysis. A couple of different approaches were arrived at for this issue.

The first proposed solution was the most obvious, and that was the use of two separate cameras. One camera would be mounted above the Lego, while the other would be mounted on the side. This would give the two angles that were needed in order to gather enough information to properly analyze the Lego and correctly determine how it needs to be sorted. This creates a couple of minor issues in the process though. With the two separate cameras, it creates a need for more space for the cameras. It also requires a good deal more processing power in order to handle the two separate images, and it would also increase the difficulty of the analysis, in that the coding would also have to handle two separate images. While these issues are certainly not massive, finding a simpler method would be ideal.

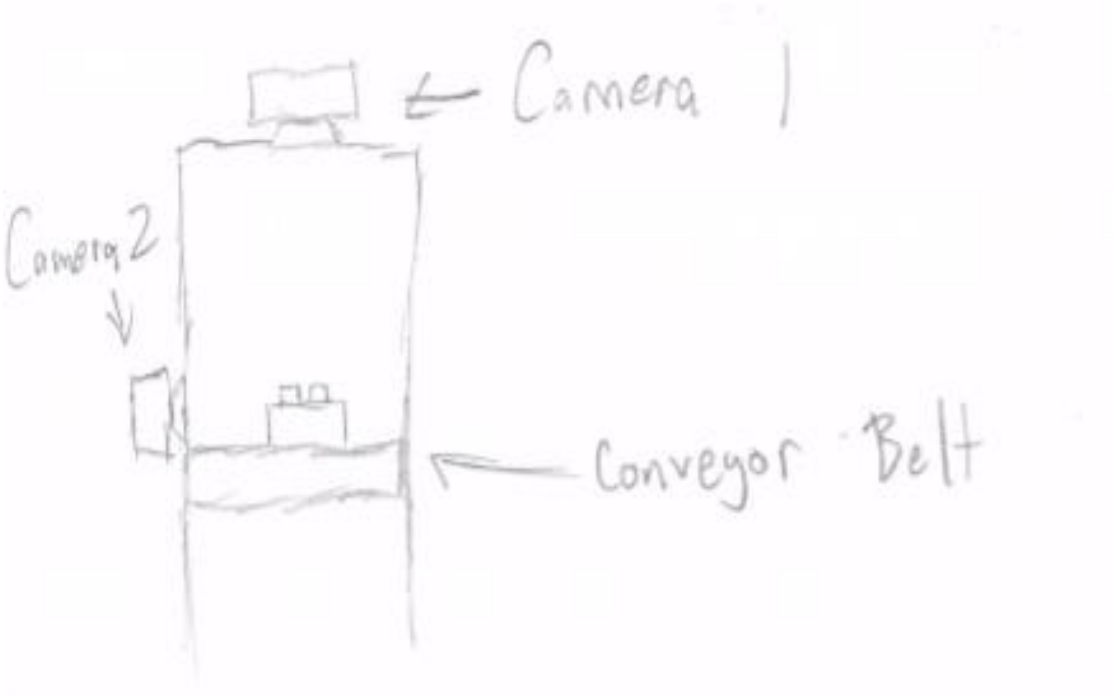


Figure 3.2.6-A Dual Camera option

After seeking advice from a reputable source, the idea of using a mirror was introduced. The thought was that rather than introducing an entirely new image source to get the second angle, the extra angle could be taken by the exact same camera, but by using a mirror at a 45 degree angle to the conveyor belt. This would essentially allow two separate images be processed at the exact same time. This would reduce the camera requirement down to one, and reduce the processing power needed for the image, however it would make setup slightly more difficult with the mirror. Perfectly placing the mirror will take a great deal of precision.

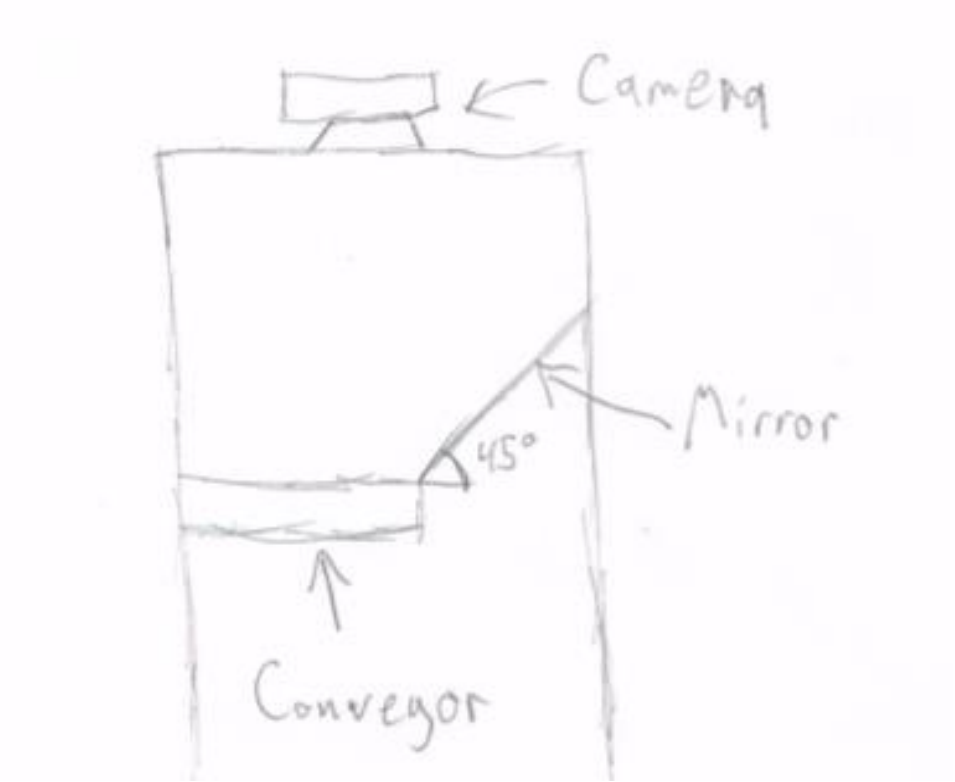


Figure 3.2.6-B: Single Camera w/mirror option

In the end, the ultimate decision was to go with the option that required only one camera and the mirror. While took more care to prepare the chamber with that setup, the belief is that the multiple advantages of using that system outweigh the single disadvantage.

3.3 Subsystem Specific Research

For organizational purposes, the project has been broken down into subsystems. Since this project requires a lot of mechanical parts, much of the subsystem specific research is devoted to the achievement of mechanical parts in an electrical/computer engineering project as well as research devoted to the electrical & computer engineering aspects of the project. The figure in 3.3-A below provides a rough sketch of the mechanical components of the project.

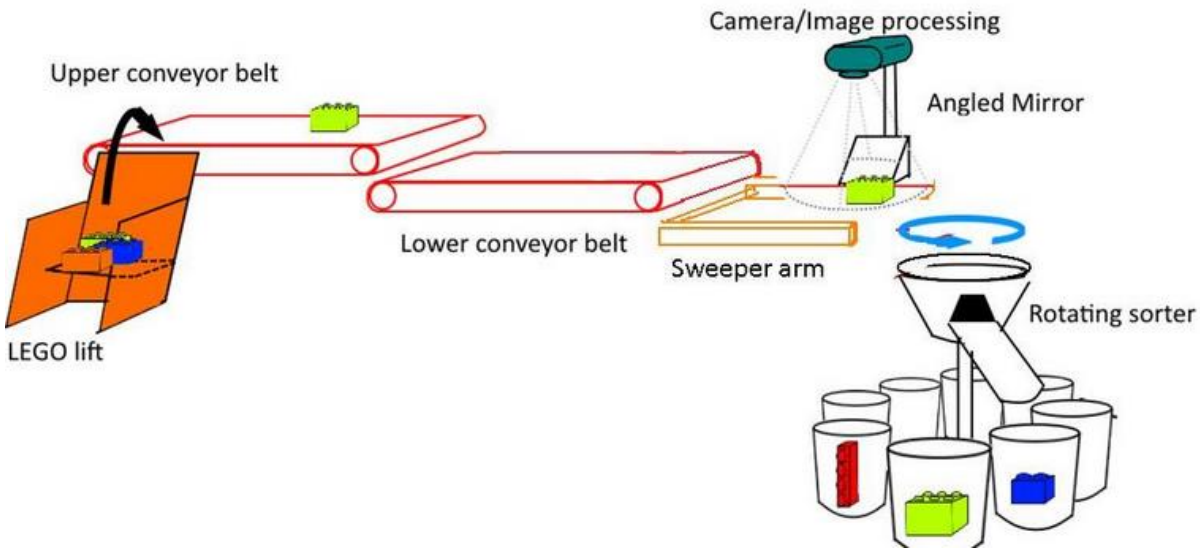


Figure 3.3-A: Mechanical subsystems

3.3.1 Conveyor Belts

Mechanical Component- The conveyor belt subsystem generated some issues of concern during research. It is understood that mechanical parts are allowed to be purchased. When browsing pre-fabricated conveyor belts online, it became evident that the conveyor belt system would have to be fabricated instead of purchased due to cost (the minimum price was around $1300 upwards of $20,000). Since the project requires two belts, the cost was too hefty.

It was decided that there would be 2 belts utilized to initially separate the LEGO parts. The first belt will move as slow as the DC motor will allow for continuous operation. This configuration is needed so that if a huge pile of parts is clumped on the first belt, the slow motion of the belt will allow 1 part to drop onto the second belt at a time (with some error). The second belt will move much faster since the parts will be separated. The exact speed is unknown at this point but will be assumed to be 3-4 times the speed of the first belt. The second belt will feed into the sweeper arm subsystem, which is situated underneath the lower conveyor belt, as well as the image processing chamber.

Summary - It was ultimately decided to build the structure of the conveyor belts out of LEGO parts as shown in Fig 3.3.1- A.

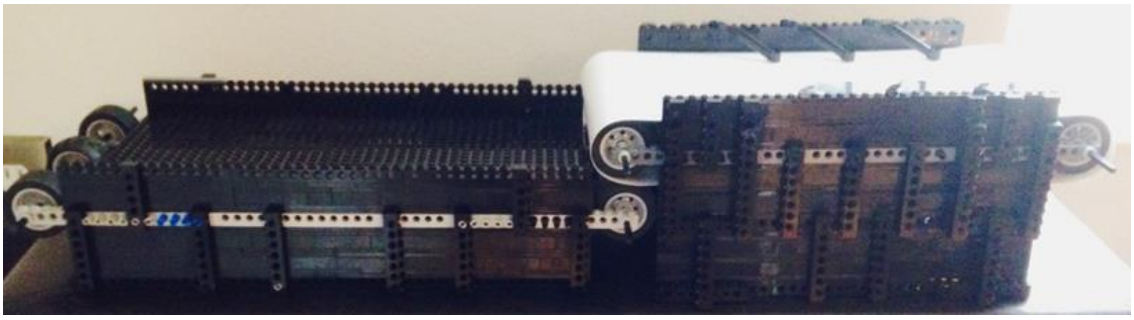


Figure 3.3.1-A Duel Conveyor System Constructed from Technic LEGO and Photo Paper

The beauty of building the structure out of LEGO's is that the size of the belt can be adjusted accordingly as needed. Furthermore, there are many different sizes, of rims, treads, and tire parts that can be selected to turn the belt during trial and error testing. Photo paper was selected for the material of the belt. The photo paper is $0.75/square foot and comes in large reams at office max which are 36" in width. This allows for minor design constraint when it comes to size. The paper also pairs nicely with LEGO since it is lightweight.

Electrical Component- The conveyer belts serve a couple functions in this project. The first function is moving the LEGO pieces from the lift arm, through the image processing system, and finally into the proper sorting container. The second function the belts serve is as a method of separating the pieces from one another. The first belt, which is elevated above the second belt, will be moving slower. The second quicker moving belt will separate pieces that are dropped within close succession and proximity to each other. This is so the image processing system only looks at one piece at a time without overlaps. The belts need to be stopped and speed controlled depending on the code in the main controller. Basic DC motors seem to be the most logical first consideration to accomplish this task. There are two general approaches to using DC motors with the conveyer belts. The first approach is to use two DC motors, one per belt. The second approach is use one motor, but have a gear system spin the second belt faster than the first. The problem with the second approach is that if one belt stops, then the other must stop as well because of the physical geared connection between them. Independent control of the belts is not possible. The second approach also leaves no room for speed-up efficiency in sorting time when both belts have to stop for the same amount of time. The first approach of using two motors, while more power consuming, leaves much more room for control versatility and speed-up efficiencies.

A way to control the speed of DC motors is through pulse width modulation (PWM). The microcontroller controlling the motors provides pulsed signals to vary the speed. Longer pulses make the motor spin faster while shorter pulses make the motor spin slower. Figure 3.3.1-B below shows the PWM signal generated by the main microcontroller which controls the motor speeds.

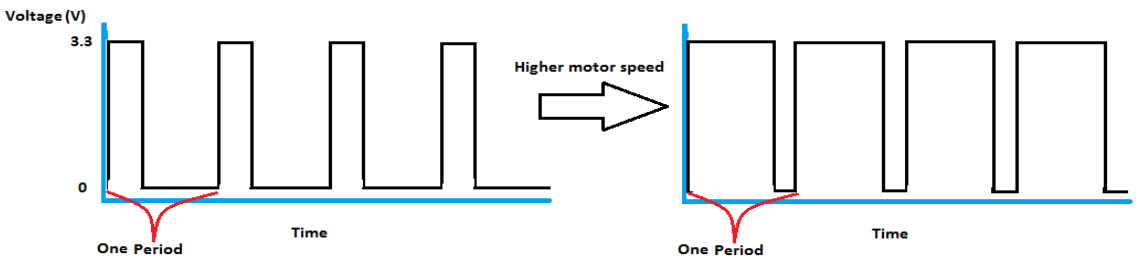


Figure 3.3.1-B PWM signals from the microcontroller vary the speed of the motors using pulse width modulation

As shown above in Figure 3.3.1-B, the longer the pulse is high during each period determines how fast the motors turn. The bottom conveyor belt will need to run at a higher speed than the top to separate LEGO pieces as they fall onto the bottom belt. The bottom belt's motor will be fed a higher duty cycle signal than the top, which will be implemented with code functions and on the microcontroller. Since the motors require higher voltage and current than microcontrollers can supply, extra circuitry is needed. A simple transistor circuit used as a switch should work. Figure 3.3.1-C below shows a simple transistor circuit that separates the PWM control signal from the high power needed to turn the motor.

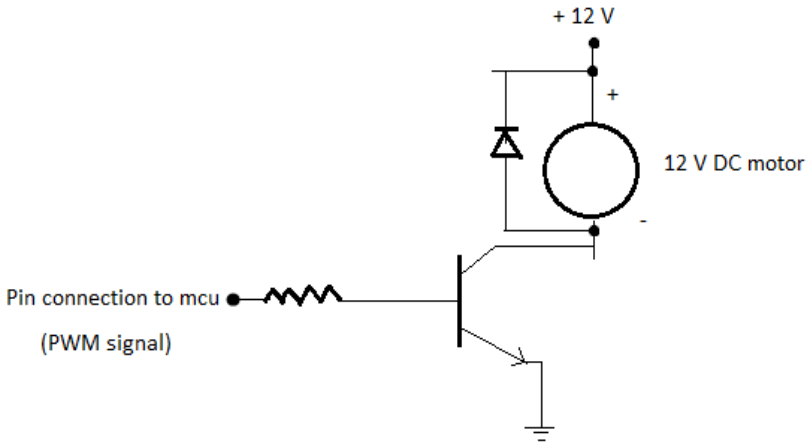


Figure 3.3.1-C: Transistor circuit with flyback diode for DC motor control

Figure 3.3.1-C above shows a transistor acting as a switch to control the 12 V DC motor. The resistor tunes the base current to make sure the transistor is in saturation mode. The diode connected across the DC motor terminals is there to remove the voltage kick-back from the motor coils when the motor is turned on and off. This circuit implementation can be repeated for each of the two DC conveyer belt motors.

3.3.2 Lift System

Initial Construction Research- The lift system is the first electro mechanical stage of the sorting process. The original idea was to construct the lift arm out of LEGO parts as shown in Fig 3.3.2-A. The moving platform utilizes Gear Racks and LEGO technic gears to move the platform up and down. The lift system was to be sandwiched in between a dump bucket for unsorted LEGO's and the first conveyor belt. To start, the platform would be level with the dump bucket. The dump bucket was to be placed at a slight angle so that the unsorted Lego parts would fall directly onto the platform. The moving platform would then traverse a few inches downward until it hits a mechanical micro switch. The unsorted LEGO parts would fall onto the moving platform, filling the space. The lower mechanical micro switch would send feedback to the microcontroller to reverse direction. Another mechanical micro switch was placed in such a way that the platform would move upward until the LEGO parts dump onto the first conveyor belt and the platform would hit the switch. The top switch would send feedback to the microcontroller to reverse the direction of the motor, repeating the process.

Servo Motor Selection – There are two types of Servo motors that were considered for the project:

- Positional Rotation
- Continuous Rotation

Positional rotation servos rotate a half circle or 180°. These servos rotate between a neutral point and either left or right depending on the control signal applied. This would only be able to move the platform a couple of inches up and down which is not helpful for the application of this project. Continuous rotation servos operate on the same principle of positional rotation servos except they can turn in either direction indefinitely. The control signal of this configuration controls the speed of rotation as well as the direction of rotation (CW or CCW) which make the movements much more precise than positional rotation servos.

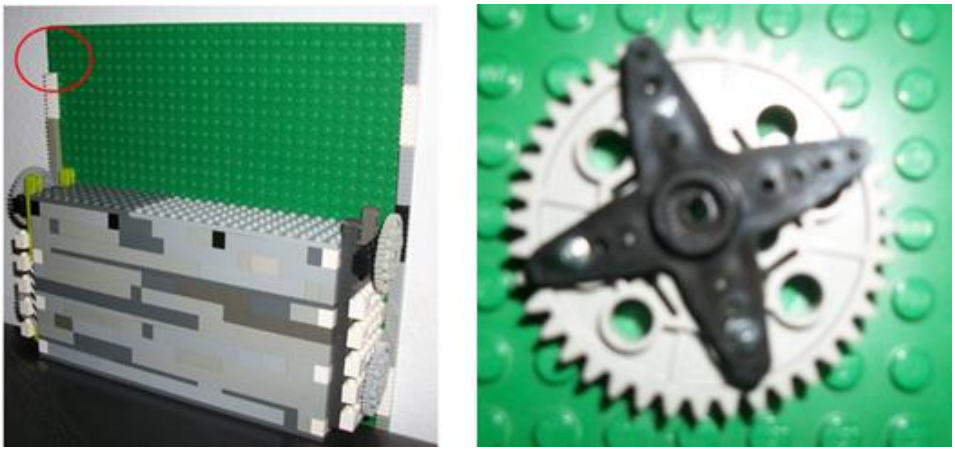


Figure 3.3.2- A: Initial Lift Arm Construction

The servo motor selected for the initial construction plan was a Parallax S148 Servo as shown in Table 3.3.2-A

Parallax Futaba S148 Servo Motor Specifications					
Weight	Dimensions	Stall Torque	Operating Speed	Operating Voltage	Torque
43g	40.5 x 20.0 x 38.0 mm	(4.8V): 2.4kg/cm	(4.8V no load) 0.28 sec/ 60°	4.8 - 7.2 Volts	3.4 kg-cm
-	-	(6.0V): 3.0 kg/cm	(6.0V no load) 0.22 sec/ 60°	-	-

Table 3.3.2- A Continuous Rotation Servo Specification

Boundary Switches- To control the upper and lower boundaries of the lift system, a pair of mechanical micro switches were selected. The switches have an AC Voltage Rating of 250V with a current rating of 5A as shown in Fig. 3.3.2- B.

Figure 3.3.2-B Lift System Mechanical Micro switch Boundary Control

Conclusion of Lift System Initial Construction Research- The Initial construction plan was to use two Parallax S148 Servo motors and mount large LEGO gears to the servo horns as shown in Fig- 3.3.2-A . The servos were threaded onto a 10" LEGO axle rod. The rod was to be fed through the entire technic platform connecting the two servos in a drive shaft fashion. The initial construction plan failed before testing began because the LEGO Gear rack parts began to fall off as more weight was added to the platform as shown in Fig 3.3.2-A. Although the initial construction plan for the lift system did not work in this application the hardware and gear/rack idea was recycled to build the "Sweeper Arm" subsystem described in section 3.3.7.

Final Construction Research – The final lift system built was linear actuated lift system. The use of mechanical micro switches was implemented as specified in the initial lift system research. This time, the moving platform was to be built out of wood. The moving platform had a ¼" coupler epoxied to a piece of wood. A DC motor was used to drive the platform up and down. The DC motor had a 12" threaded rod coupled to the motor shaft. The moving platform supported by the motor was built using ultra-light weight balsa wood. The outside supporting frame was built using sturdier 4 ply wood. 12" square shaped dowels were used to build the guide rails to move the structure up and down. The construction of the lift system can be seen in Fig. 3.3.2-C.

Figure 3.3.2-C : Final Construction of Lift System

Final Construction Motor Selection- Since processing LEGO's using image processing is not the fastest method of sorting LEGO parts, it was decided that a high RPM motor would be utilized to move the platform up and down. Below is a summary of the pros and cons of linear actuators and standard DC motors.

Linear Actuator Pros:

- All in one construction
- Easy to control

- Built in limit switches

Linear Actuator Cons:
- Extremely Expensive $80+
- Constricted to set Size
- RPM not quite high enough

DC Motor Pros
- High RPM
- Easy to control with H-Bridge Circuit

DC Motor Cons:
- Must use micro switches to control the boundaries

It was ultimately decided to use a high RPM DC motor. The RS-455PA DC motor was selected. The specifications are shown in Table 3.3.2-B.

RS-455PA DC Motor	No Load		Stall
Operating Voltage	Speed	Current [A]	Current [A]
12-42 [V]	5500 [rev/min]	0.055 [A]	0.1 A

Table 3.3.2-B Lift System DC Motor Specifications

The stall current was measured to be 100mA.

Conclusion of Lift System Final Construction Research- The lift system will utilize a 550 RPM DC motor to construct a linear actuator using a ¼" coupler and threaded rod in conjunction with two mechanical micro switches to send feedback to the microcontroller of the upper and lower boundaries of the system.

3.3.3 Rotating Arm System

Motor - The sorting bins will be placed in a circular pattern below and around the rotating arm system. The rotating arm will ultimately sort the LEGO pieces by rotating to the proper location and let the LEGO piece fall through it and into the proper bin. The range of motion of the sorting arm will need to be at least 360 degrees to accommodate the location of all the bins. There are two types of motors that were considered for this task: servo motor and stepper motor. Servo motors are great for high speed applications, at thousands of RPM. At high speeds, servo motors hold their torque rating up to 90%. The drawback of servos is their complicated circuitry and high price. Servos must have an encoder to provide feedback to the motor. Because servos have a small number of magnetic poles for the motor shaft to rest at, the encoder and motor will be using more current to keep adjusting itself to maintain the correct position. Stepper motors, on the

other hand, have a large number of magnetic poles (between 50 and 100) and require a much less complicated driving circuit. Because of the greater number of poles, the stepper motor draws less current to keep the motor at a position under load. For this reason, stepper motors are ideal for lower speed applications. The rotating arm only needs to rotate at a low speed, possibly in the range of 60 RPM or less. The load will be very small on the motor, considering it will just rotate a light-weight shaft and LEGO slide. With this information in mind, the stepper motor would be the better choice for the task.

There are two main types of stepper motors to consider: unipolar and bipolar. Unipolar stepper motors require less circuitry than bipolar but are less efficient and provide less torque. Bipolar motors are simpler in design, but the driving circuitry is more complex. Both motor types can be implemented so that they use two or four pins from the main microcontroller. The bipolar motor will be used for this project because of its higher efficiency. The more complex circuitry to drive the bipolar motor is not an issue as it can be packaged in a tiny IC chip. A simplified diagram of a bipolar stepper motor is shown below in Figure 3.3.3-A.

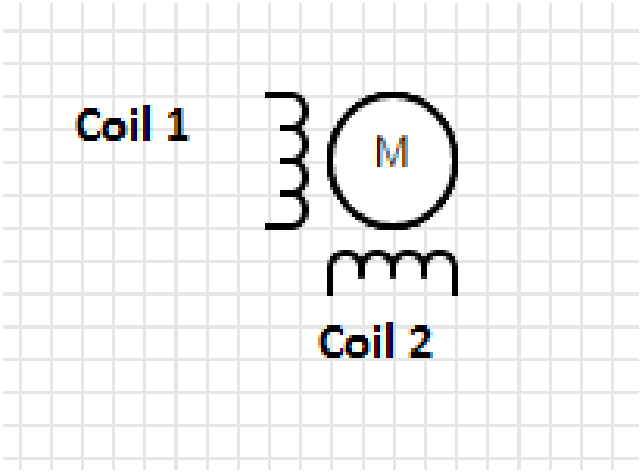


Figure 3.3.3-A: Simplified bipolar stepper motor diagram

In the Figure 3.3.3-A above, the two coils would be polarized in an alternating fashion to step the motor through its rotation. In an actual stepper motor, there are many poles to increase the step resolution and total possible number of angles the motor can hold at.

Motor Circuitry - The bipolar stepper motor is a possibility for the task of moving the rotation arm. For the motor to be controlled by the main microcontroller in the project, a stepper motor driver circuit must be integrated. A common way to control stepper motors is through the use of an H-bridge circuit. The H-bridge circuit allows for current to flow through a device in either direction. Stepper motors usually are connected to an H-bridge to allow its coils to be energized in either direction. An H-bridge concept diagram is shown below in Figure 3.3.3-B.

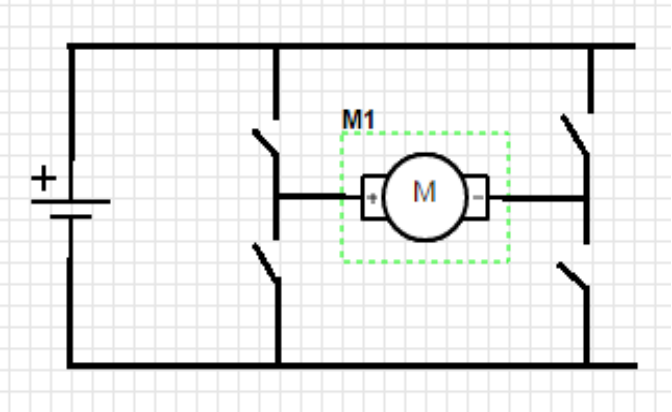


Figure 3.3.3-B: H-bridge conceptual circuit

The switches in Figure 3.3.3-B above are transistors in the actual H-bridge circuit. Biasing resistors and flyback diodes are common in H-bridge circuits as well. To save project complication and space on the PCB board, the bipolar stepper motor can be driven by an IC chip. Since stepper motors have two coils to control, the use of two H-bridge circuits is needed. The L2394D package has two integrated H-bridges and is ideal for this project. Consideration must be taken to find an appropriate bipolar stepper motor that will not overload the H-bridge chip in any way. The recommended operating conditions of the H-bridge IC are shown below in Table 3.3.3-A.

		Min	Max	Unit
Supply Voltage	Vcc1	4.5	7	V
	Vcc2	Vcc1	36	V
V_{IH} High Level Input Voltage	Vcc1 ≤ 7 V	2.3	Vcc1	V
	Vcc1 ≥ 7 V	2.3	7	V
V_{IL} Low Level Output Voltage		-0.3	1.5	V

Table 3.3.3-A: Recommended voltage levels of the L2394D duel H-bridge IC

The motor supply voltage on pin “V2” in Figure 3.3.5-1 can be in the range of 4.5V – 36V. This gives a fair amount of flexibility for the choice of motor. The pin “V1” shown in the same figure can be in the range of 4.5V to 7V. Connected to that pin will be a logic output pin from the main microcontroller in the 5 V range.

Sensor - The rotating arm system should use a sensor as feedback to ensure the correct positioning of the arm during each rotation. At the very least, a “home” position for the rotating arm stepper motor needs to be established each time the system is started up. This can be done many ways with various types of sensors. The initial consideration is

through use of a color sensor that can identify which sorting container it is facing at all times. Each container can be a different color with the sensor attached to the rotating arm. The advantage of this method is that it provides a continuous stream of feedback to the microcontroller to reduce poor arm positioning. A sketch of the possible physical layout of the sensor incorporated into the rotating arm system is shown below in Figure 3.3.3-C

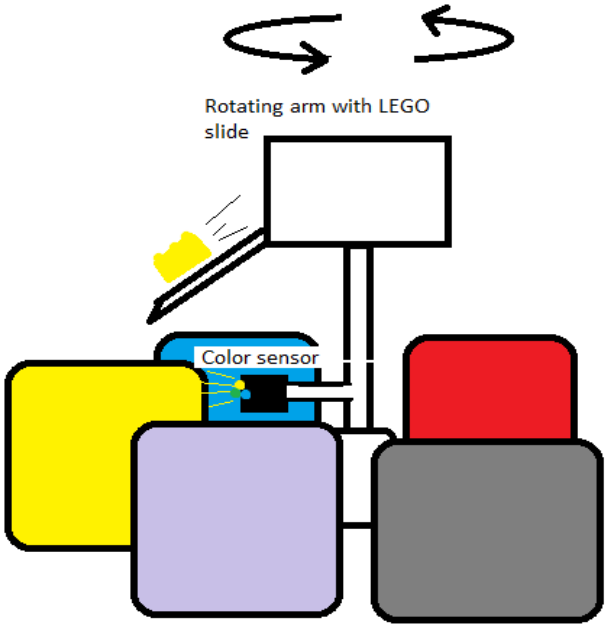


Figure 3.3.3-C: Sketch of rotating arm with color sensor and LEGO bins

As can be seen in Figure 3.3.3-C above, the color sensor is attached to the shaft of the rotating container that the LEGO falls through. The sensor is extended outwards to face the colored sorting containers as the shaft and LEGO slide rotate around. The sensor makes sure that the initial rotation by the stepper motor is adequate by relaying the color information of the containers to the microcontroller.

A possible drawback to this configuration is outside lighting. If the sensor is not completely contained in a system with controlled lighting, ranges of color values may be different from location to location. This issue can be mitigated with a completely contained sensor system, or perhaps through some clever programming.

Possible sensors considered are the Color Sensor Breakout HDJD S822, Geeetech TCS230 and TCS34725. Specifications and features of each sensor are shown below in Figure 3.3.3-B.

	Supply voltage	Unique Features	Interfacing	Price
HDJD S822	5 V	Analog output	GPIO	$24.95
TCS230	2.7 – 5.5 V	Light intensity current to frequency conversion	I2C	$14.00
TCS34725	3.3 – 5 V		I2C	$7.95

Table 3.3.3-B

The HDJD S822 has a more intuitive design with the RGB sensors each outputting an analog value based on the intensity of the respective colors detected. The TCS320 is cheaper and has a more powerful integrated feature that converts the current of each color channel into a frequency that can be read by the microcontroller. The TCS34725 requires the least amount of pins and is the most cost effective solution of the color sensors.

There is another issue with having a color sensor rotate on the arm, however. Wire tangling will occur if some wire management system is not devised to prevent it. Another sensor possibility is using an IR reflectance sensor, to the side of the arm, to reflect IR light off of a surface on the rotating arm system. A marking on the arm can then be used as a homing position for the system. A considered sensor is the QRE1113 IR reflectance sensor. This sensor has an analog output and would require an ADC channel from the MCU. Advantages of the QRE1113 include easy coding, worry free wires, and only one output pin.

3.3.4 Image Processing (technologies research)

There are a few technological barriers to consider when deciding on the image processing device. One potential barrier is the data size of the image or video frame, which is determined by a number of factors. For this project, LEGO pieces need to be identified by their color and shape. With this in mind, the color depth as well as the resolution of the image must be sufficiently high to identify the pieces successfully.

There are a total of 58 different LEGO colors[1] that exist that can potentially be ran through the LEGO sorter. The image processing device must operate on a color depth that can distinguish each color. A couple issues to note is that some LEGO pieces are very similar in color and there may be situations where shadows or reflections on the LEGO pieces are registered as the wrong color by the image processing device. The way the pieces enter the camera's field of view cannot easily be controlled. The color depth will have to be able to accommodate for these variations as well. Some typical color depths that are used in various camera sensor and display applications are 8-bit color, 16-bit High color, and 24-bit True color. The numbers of colors these depths can "resolve" are 256, 65k, and 16.7M respectively. Because of the issues mentioned above, 256

identifiable colors may not be enough to resolve the pieces reliably. The 16-bit High color may be a good depth to use with over 65,000 colors. Resolution of the image is also important for edge detection of the LEGO pieces. The resolution does not need to be high definition, but needs to be high enough for detecting the edge of the LEGO against a solid color background. VGA format (640x480) should be a sufficient resolution for this project. With the color depth of 16 bits and a resolution of 640x480, this gives an image size of 614,400 bytes, or about 614 Kb. The image processing device must have enough onboard memory to hold an image of this size. It may be possible to have a smaller image, but for reliability purposes, the device used in the project should have hardware that is able to exceed this value. Knowing the above, there are various technologies that can accomplish the image processing needed for this project. Among the platforms to consider are: microcontrollers, microcomputers, digital signal processing (DSP) development boards, and field programmable gate array FPGA development boards.

Microcontrollers, while generally not suitable for higher end image processing, are inexpensive and consume little power relative to the other technologies listed above. A table of some popular microcontroller development boards, their prices, and relevant specifications is shown below in Table 3.3.4-A.

Microcontroller name	CPU	SRAM (bytes)	Code flash memory (bytes)	Clock (Hz)	Development board Price	CPU Price
MSP430G2553	16-bit RISC CPU	0.5K	16K	16M	$9.99	$2.80
MSP430FG4618	16-bit RISC CPU	8K	116K	8M	$117.00	$17.54
Arduino Due	Atmel SAM3X8E ARM Cortex-M3	96K	512K	84M	$44.93	$12.28
Tiva C Series TM4C1294	ARM Cortex-M4	256K	1M	120M	$19.99	$18.64

Table 3.3.4-A Microcontroller development boards being considered

The MSP430G2553 is a great, very low power and cost effective device, but the on-board SRAM of only 0.5KB is not anywhere near the 614KB figure calculated for the size of the image. Not only that, but there is no USB interface for USB webcam. It seems there would be no simple way for this microcontroller to handle the image processing for the project. The MSP430FG4618 is a development and experimenter board used for learning and developing embedded projects. This board also comes up short in the memory department for this project at only 8K in on-board SRAM. There is also no USB interface for the camera. It will not be used. The Arduino Due is a higher end microcontroller that is popular among hobbyists. The Arduino was initially considered for this project because of the large online community support around the board. However, looking at the

technical specifications, the SRAM is again, not high enough to hold a relatively low resolution image to be processed. The Tiva C series TM4C1294 was also initially considered because of its higher overall specs relative to other popular microcontrollers. It's 256KB SRAM memory may be high enough to hold lower resolution images, but it might be pushing the limits for use in this project. The hardware specs should exceed the rough estimates of memory usage for the project, which this device does not do.

Because none of the microcontrollers initially considered for image processing in the project met the specification requirements, the technology will not be used. Another technology that was initially considered is microcomputers. Microcomputers, like microcontrollers, have a general purpose central processor. Unlike microcontrollers, microcomputers are capable of running an operating system and control a large variety of peripherals. The use of an operating system, while more power consuming, has the benefit of allowing for easier use of code libraries such as OpenCV. Microcomputers also have the added benefit of more onboard memory compared with microcontrollers. Table 3.3.4-B below shows some relevant specifications of two popular single-board microcomputers.

	CPU	RAM(bytes)	Clock(Hz)	Relevant peripherals	Storage	Price
Raspberry Pi B	ARM1176JZFS	512M	700M	(2) USB host, (17) GPIO pins	External SD (up to 2GB)	$39.95
Beaglebone Black Rev C	AM3359 Sitara ARM Cortex-A8	512M (DDR)	1G	(1) USB host, (65) GPIO	Onboard 4GB eMMC	$55.00

Table 3.3.4-B

As shown in the Table 3.3.4-B above, The Raspberry Pi Model B is plenty capable of the image processing requirements with 512MB of onboard RAM and a 700 MHz CPU clock speed. The Raspberry Pi also has a dedicated floating point unit (FPU). The Beaglebone Black Rev C is also very capable of meeting and exceeding the image processing demands for the project. The Beaglebone is $15 more than the Raspberry Pi, however, it beats the Raspberry Pi in computing power and memory. Both boards are capable of running OpenCV and have large support communities, albeit, Raspberry Pi's community is a bit larger. Both boards have a USB host port(s) that can be used for webcam integration. The Beaglebone is a fair amount faster, which will aid in the speed of the LEGO piece identification.

Another technology initially considered is DSP development boards. DSP boards are similar to microcontrollers in their functionality and lack of an OS, however they have specialized built-in hardware for dealing with real-time digital signal processing. One DSP board considered was the TMS320C6748 DSP Development Kit by Texas Instruments. It features a dedicated DSP CPU at 456MHz, 128 MB DDR3 SDRAM, and

(1) USB host. It costs $195.00. The DSP kit is faster in computing than both microcontrollers and microcomputers, but they are pricier and may take longer to develop due to the lack of community support.

FPGA boards were also initially considered. FPGAs differ from the previous technologies in that there is no already-built processor. Instead of a general CPU, the programmer designs the circuitry to directly handle the processing of a signal or image. This technology is much faster than the previously listed ones. The speed that FPGAs process at is not required for this project. The price range of FPGA kits is generally higher than the other tech with the Altera Cyclone II FPGA Starter Dev Kit at $199.00. When all is considered, FPGA technology is overkill in terms of processing speed for this project. Due to the limited amount time to design and build the project, FPGA is not a feasible solution.

Conclusion - Most microcontrollers are too slow and do not have enough onboard memory to process images. DSP development kits have more than enough speed and memory to handle the task, but are more expensive and have less resources and community help to prevent and resolve design roadblocks. FPGA kits are expensive and way overkill in terms of processing speeds for the comparatively low level image processing demands of the project. While microcomputers use more power to run an OS on the CPU, they contain enough processing power and memory to get the job done. The two microcomputers considered, the Raspberry Pi and Beaglebone Black Dev C, both have good online support and can run useful libraries like OpenCV. In the end, the Beaglebone Black Dev C seems like the best choice for the image processing of the project because of its quick processing speeds, high amount of memory, and it's relatively low price.

3.3.5 Main Microcontroller

The main microcontroller will be the central processing device that connects all the sub systems together. It will receive data from the image processing system, touchscreen controller, and the various sensors throughout the project. The microcontroller will also be in charge of controlling the lift arm motor, sorting arm motor, conveyer belt motors, as well as displaying the user interface on the LCD. The main controller needs enough I/O pins to communicate with and control each sub system. Figure 3.3.5-A below shows the estimated pin usage of the various devices that will be connected to the main microcontroller.

Device	Pin count	Connection type
Image processing (Beaglebone Black)	4	SPI
LCD display	4	SPI
Touch screen control	4	SPI
Sorting arm stepper motor	2 or 4	GPIO
Conveyer belt motors (2)	2 (1 per motor)	GPIO
Lift arm servo	1	GPIO
Sensors	3 - 6	GPIO
Other (LEDs, buzzers,	2 - 3	GPIO

Figure 3.3.5-A

From the table, the absolute high estimate is 22 pins in total could be used by devices connected to the controller. Six of those pins must be involved with SPI master-slave interfacing. While the GPIO pin count may be reduced by adding logic devices like shift registers, it may be best to err on the side of caution to select a controller with more I/O pins.

Other considerations during microcontroller selection include: logic voltage of the GPIO pins, supply voltage, current rating, serial/parallel interfacing, CPU speed, and price. Figure 3.3.5-B below shows various microcontroller platforms initially considered for the project with relevant specifications:

	GPIO Count	Logic Voltage	Input Voltage	Communication Interfaces	Code Memory	CPU Speed	Price
MSP430FG4618	80		1.8–3.6V	I2C, SPI, UART	116 KB	8 MHz	$17.54
MSP430G2553	16		1.8–3.6V	I2C, SPI, UART	16 KB	16 MHz	$2.59
TM4C1294N	90		5V	I2C, SPI, UART, CAN	1 MB	120 MHz	$17.73
ATmega32U4	26		7 – 12V	I2C, SPI, UART,	32 KB	16 MHz	$6.32

Figure 3.3.5-B Specifications on the considered MCU platforms.

From figure 3.3.5-B it can be seen that all but one microcontroller provide enough I/O pins for the project. The MSP430G2553 was initially considered because of the familiarity of its usage as well as its low power reputation. The controller, however, falls short of the 22 pin estimate for this project and will therefore not be used.

The rest of the controllers meet the project requirements so selection comes down to the most power for the price. The ATmega32U4 is the cheapest of the remaining choices and it contains enough pins as well as medium range clock rate. It also has a recommended 7 - 12 V input, which can be shared with the other devices connected to the 12 V power supply output. An extra voltage output does not need to be designed on the power supply to accommodate the controller.

3.3.6 LCD Screen

Size - The menu for the user interface will need to be moderate in size. A 3" display was considered but it was too small to accommodate all of the menu options. A 7" display was considered but was too large for the scope of the project. A 5" display was chosen for the project.

Resolution and Response Time - Resolution and Response time were a couple more considerations. Since the project requires a simple user interface it was ultimately decided to use an LCD with a lower resolution display. Response time is also not a huge issue since the user will only need to input a couple of selections.

Graphic vs. Character Display - It was decided that a "graphic" LCD display will be utilized to create a menu for the user to select the method of sorting. A "character" display is too simple for the user interface desired. A TFT LCD is more than enough to accomplish the job at an affordable price. The only drawback is the TFT LCD consumes more power to drive the backlight (in comparison to a top of the line OLED display). There are two options for a TFT LCD display: High or Low level. The high level version interfaces to an on-board microcontroller whereas the low level version interfaces to a display controller. The high level LCD displays were a lot pricier and not necessary for the project but they do have different interfacing options such as UART/RS232 connections, USB, and Ethernet/wifi interfaces as shown in Table 3.3.6-A

Comparison of High and Low Level Graphic TFT LCD Displays							
Crystalfontz	USB	RS232/UART	Ethernet / Wifi	SPI	IIC	RGB/DotClk	Parallel 8,16,24 Bit
High Level	✓	✓	✓	✓	✓		
Low Level				✓	✓	✓	✓

Table 3.3.6-A Comparison of High and Low Level Graphic TFT Displays from Crystalfonz

Interfacing the LCD display Controller to the Main Microcontroller- Interfacing the LCD display will propose some of the biggest challenges from the hardware point of view. It is imperative that the LCD is paired with an appropriate microcontroller that has a compatible interface. The Table 3.3.6-B offers a clear representation of some specifications that need to be considered.

Vendor: Buydisplay.com	ER-TFT035-1	ER-TFTM050-2	ER-TFTM070-5
Processor	8051, PIC, AVR, ARDUINO, ARM	8051, PIC, AVR, ARDUINO,and ARM	8051, PIC, AVR, ARDUINO, and ARM
Size	3.5"	5"	7"
Colors	65K/262K/16.7M	256/65K	256/65K
Resolution	320 x 420	480x272	800x480 Dots
Interface	6800 8080 8, 9,16,18 bit parallel,3-wire,4-wire serial spi, rgb	6800 8080 8,16, bit parallel,3-wire,4-wire I2C serial spi	6800 8080 8,16 bit- parallel,3-wire,4-wire,I2C serial spi
Typical Supply [V]	3	3.3 ,5	3.3V, 5V
Supply Current for LCM(Max)	"No"	230mA (Vdd=3.3V) 180mA (Vdd=5.0V)	480mA (VDD=3.3V) / 300mA (VDD=5.0V)
Sample Code 1	8080 (8 or 16) bit Parallel Interface	3-wire SPI Serial Interface Demo Code	3-Wire SPI Serial Interface+Resistive Touch Panel Demo Code
Sample code 2	4 Wire SPI Interface Code	4-wire SPI Serial Interface DemoCode	4-Wire SPI Serial Interface+Resistive Touch Panel Demo Code
Sample Code 3	N/A	8080 8-bit Parallel Interface Democode	I2C Serial Interface+Resistive Touch Panel Demo Code
Sample Code 4	N/A	8080 16-bit Parallel Interface Democode	8080 8-bit Parallel Interface+Resistive Touch Panel Demo Code
Sample Code 5	N/A	I2C Serial Interface DemoCode	8080 16-bit Parallel Interface+Resistive Touch Panel Demo Code
Touch Panel	4 Wire Resistive	4-wire resistive touch panel or 5 points capacitive	4-wire resistive touch panel or 5 points capacitive
Price	$17.84	$27.61	$34.05
Controller (IC Equivalent)	SSD2119	RA8875	RA8875
Memory	172,800 Bytes GDDRAM	768KB DDRAM	768KB DDRAM

Table 3.3.6-B: LCD Specification Comparison

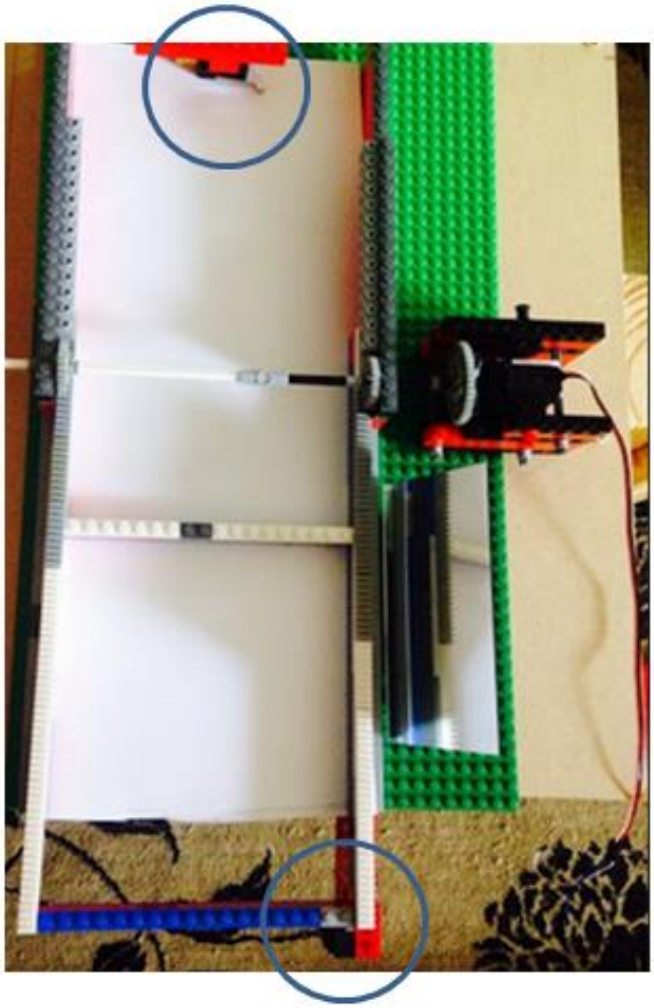
The TFT LCD displays were all found on buydisplay.com. The nice thing about the TFT displays found on buydisplay.com is that they all have sample code to help initialize interfacing.Buydisplay.com also offers “accessories” for interface, meaning that each TFT LCD from their website offers a selection of options for interfacing instead of selling their displays with a fixed interface for each device as crystalfontz.com does. This made it much easier to select an LCD more than 1 interface option can be added. Buydisplay.com also offers PDF datasheets for the touch panel controllers, TFT (Thin Film Transistor) LCD (Liquid Crystal Display) module, and for the IC equivalent controllers.

Conclusion- In summary, the RA8875 controller TFT, resistive touch, 5” display was chosen from Buydisplay.com for the project since the website provides ample documentation and set up examples. It also allows has many interface options for flexibility.

3.3.7 Sweeper Arm System

The sweeper arm is essentially a bar that sits below the second conveyor belt. At the output of the second conveyor belt is the “Image Processing Chamber.” After a LEGO part has been identified, it needs to be transferred to the “Rotating Arm” system. The idea for the sweeper arm was recycled from the initial construction plan of the “Lift System,”

which was mentioned briefly in section 3.3.2. The sweeper arm utilizes LEGO technic parts to form a gear and rack system. A Large LEGO gear was coupled to the Parallax S148 continuous rotation servo motor horn. The specifications of the continuous rotation Parallax S148 servo can be found in table 3.3.2- A. The gear was used to thread a LEGO axle onto it. The sweeper arm has a row of gear racks on either side, connected by a bar in between as shown in Fig. 3.3.7-A.



3.3.7-A Sweeper Arm Gear & Rack System

To start, the blue part of the sweeper arm shown in Fig 3.3.7-A is positioned completely underneath the second conveyor belt against the inner micro switch allowing a LEGO piece to fall in front of the mirror. After the LEGO has been identified, the main microcontroller turns on the servo motor to push the bar forward "sweeping" the LEGO part into the rotating arm system as shown in Fig. 3.3.7- B. The sweeper will extend forward until the micro switch positioned above the rotating arm is triggered. Once the switch is triggered, a message is sent to the main microcontroller to turn the servo in the opposite direction. The sweeper arm will then go backwards until it triggers another

micro switch positioned below the second conveyor belt. When this switch is triggered, the servo will stop rotating and wait for the next LEGO part to be processed.

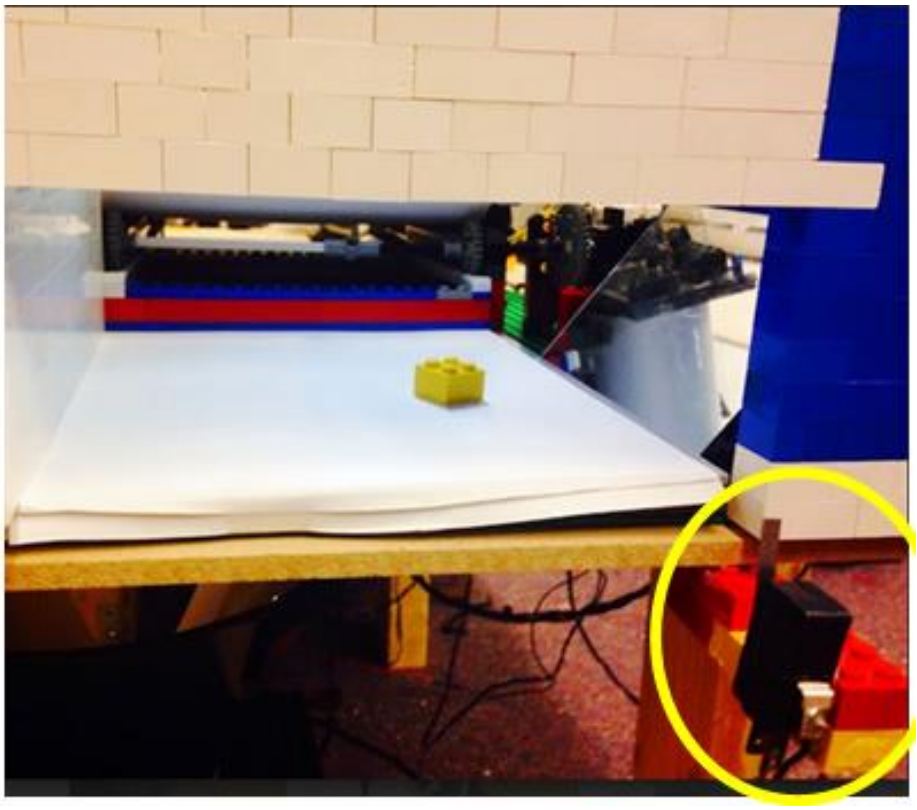


Figure 3.3.7-B: Sweeper Arm Positioned Under the Second Conveyor Belt.

The switches used to control the boundaries of the Sweeper Arm Subsystem are the same 250V AC 5A mechanical micro switches that were used to construct the "Lift System" described in section 3.3.2.

3.3.8 Power Supply

Overview - A power supply is nothing more than a device that is used to "step down" or "step up" a voltage. Since the objective of this project is for a user to turn on the sorter and leave it on overnight, it is imperative that the supply is stable enough to run continuously for long periods of time without dissipating a lot of heat. Considering that the sorter is to be used in home at all times, the input will come from a standard 120V AC 15A US wall outlet.

Subsystem Power Consumption - The first step in designing a power supply is to find the voltage and current of each device used in the system. The project has been broken down into subsystems for simplicity. The subsystems to be dissected: Conveyor System,

Lift System, Sweeper Arm System, Image Processing System, Rotating Arm System, and the User Interface. Every component used in each subsystem that consumes power will need to be taken into account by finding the voltages and currents of which these devices operate. It is also important to note that some devices may operate at the same voltage but different current. Once the summation of voltages and currents drawn by each device from each and every subsystem is determined, the power supply can be rated and designed accordingly. Table 3.3.8- A indicates the operating voltages and currents of the devices to be used in the project. The LCD screen RA8875 controller has a typical output voltage of 3.3 or 5V. Since the Beaglebone Black, Sweeper Arm servo, and Switches all operate at 5V, the RA8875 controller supply voltage was chosen to be 5V as well for simplicity. The Atmega32u4 specifies an operating voltage range of [7-12] V. The upper operating range of 12V was selected since the project has many other devices operating at 12V. The DC current for the I/O pins is rated for 40mA. The project requires 12 I/O pins which is equivalent to 0.48 A. An extra I/O pin was taken into consideration to be on the safe side and the Atmega32u4 was rated for 0.52A. The Beaglebone Black has an operating current range of [210-460] mA. The higher current range was accounted for in the power consumption table.

COMPONENT POWER CONSUMPTION			
Component	Rated Voltage [V]	Rated Current [A]	Power Consumed [W]
	Supply 1		
Lift Arm DC Motor	12	0.3	3.6
Conveyor Belt DC Motor #1	12	0.3	3.6
Conveyor Belt DC Motor #2	12	0.3	3.6
Atmega32u4	12	0.52	6.24
Total		1.42	17.04
	Supply 2		
Lift Arm Microswitch #1	5	0.17	0.85
Lift Arm Microswitch #2	5	0.17	0.85
Lift Arm Controller (L293D)	5	1.2	6
Rotating Arm Stepper Motor	5	0.32	1.6
Rotating Arm Controller (UNL3003)	5	0.5	2.5
Rotating Arm Photoelectric Sensor	5	0.02	0.1
Beaglebone Black	5	0.46	2.3
Total		2.84	14.2

Table 3.3.8- A: Subsystem Component Power Consumption

POWER RATING		
	Supply 1	Supply 2
Rated Voltage [V]	12	5
Rated Current [A]	1.42	2.84
Rated Current +20% [A]	1.7	3.41
Power Consumed [W]	20.4	17.05

Table 3.3.8-B: Power Rating of Overall System

Design Flaws and Non-Idealistic Characteristics - In real life, the voltages and current ratings given in data sheets are only very good estimates at best. There are statistical deviations in all electronic devices no matter how well they are designed. Electrical circuit components are very sensitive devices. For example, transformers store energy in the form of electromagnetic fields. These EMF fields can create noise in nearby circuit components if they are not isolated appropriately. The wall supply has an internal resistance which can alter design calculations as well. Diodes and transistors are heavily temperature dependent devices which can be damaged easily if there is too much current flowing through them. There are many other limiting factors involving circuit components that can alter an expected output voltage. To have a reliable power source, these non-idealities need to be considered and designed around accordingly. To accommodate possible design flaw & non-ideal characteristics of circuit components to be used, as well as internal resistance supplied by the AC wall voltage source, and to allow for continuous operation, 20% current will be added to the maximum current that has been rated for the power supply. The power supply required two DC voltages for the project. Vo1= 12V and Vo2 = 5V. The final power supply ratings can be found in Table 3.3.8-B.

Note- The "Sweeper Arm" subsystem was implemented in the project after the PCB board for the power supply was sent out. The sweeper arm servo motor operates at 5V with a rated current of 300mA. The micro switches have negligible current consumption since they can operate at a maximum voltage of 250 VAC. The rated current for the 5V supply is 2.84A. If the sweeper arm servo is included the rated current becomes 3.14A which is still lower than the rated current +20% of 3.41A.

AC-DC Converter- An open frame switching mode power supply was selected for the AC to DC conversion of the project. The supply is manufactured by Mean Well model: PS-45-12. The input voltage is 115VAC rated at 0.8A (typical). The supply has an output of 12V rated at 3.7A. This output was used as the input to the two DC voltages Vo1=12V rated at 1.7A and Vo2 = 5V rated at 3.41 A that were designed for the project.

Kill switch – A simple Single Pole Single Throw (SPST) kill switch was selected from home depot that operates at 115V AC 15A to cut off power to the AC input to the Mean Well AC-DC open frame power supply.

4.0 Project Design Details

Section 4 will be dealing with the overall design plan of the Lego sorter. These are the plans that were used to build the foundation of the project and will help to be the guide for the entire construction process.

4.1 Block Diagram

To begin, the user assigns what color or size LEGO parts they want sorted into each bucket using the RA8875 LCD Screen. After the selection has been made the RA8875 Controller signals the Atmega32u4 to begin the sorting process. The Atmega32u4 is the main microcontroller that controls all of the mechanical systems, including the Lift System, Dual Conveyor Belt System, Sweeper Arm System, and the Rotating Arm System.

The "Lift System" is the first mechanical subsystem to the sorting process. The user drops their unsorted LEGO parts onto the slanted platform of the lift system which starts out positioned below the first conveyor belt. This allows room to dump the unsorted LEGO parts. The lift system traverses upwards until it hits a limit switch and the unsorted LEGO parts are then dropped onto the first conveyor belt (Recall that the top of the Lift System has an angled piece.

The Top Conveyor Belt is positioned directly above the Bottom Conveyor Belt. The lift system moves back down until it hits the bottom switch and repeats the process. The first conveyor belt moves at the slowest speed possible that allows the motor to turn the belt. This ideally drops the LEGO parts onto the second belt one at a time. The second belt moves 3 to 4 times faster than the first belt in order to separate the LEGO parts out even further.

The second belt drops the LEGO parts onto a stationary platform inside the "Image Processing Chamber." The Image processing chamber consists of a Logitech Webcam placed above the platform where the LEGO parts drop and a mirror that is positioned 45° to the platform. The mirror allows the webcam to gather two views of the LEGO parts for data processing. Once the LEGO part has been identified it needs to be deposited into the appropriate sorting bucket that has been set up by the user.

The BBB sends a message to the Atmega32u4 to position the rotating arm to the appropriate bucket. For example, if the BBB detects an orange LEGO part, and the user

has identified bucket #5 to collect orange LEGO's, the rotating arm will rotate to bucket #5.

The final step is for the LEGO part to be swept into the rotating arm. The Sweeper arm is positioned completely below the second belt when a LEGO part is inside the image processing chamber. Once the LEGO part has been identified and the rotating arm is positioned to the appropriate bucket, the Atmega32u4 turns on the continuous rotation servo motor. This servo motor is coupled to a gear and rack system with a bar attached. The gear and rack system pushes the bar forward, pushing the LEGO into the rotating arm, which drops the LEGO into the appropriate sorting bucket. Fig 4.1-A shows the functional block diagram of the entire system.

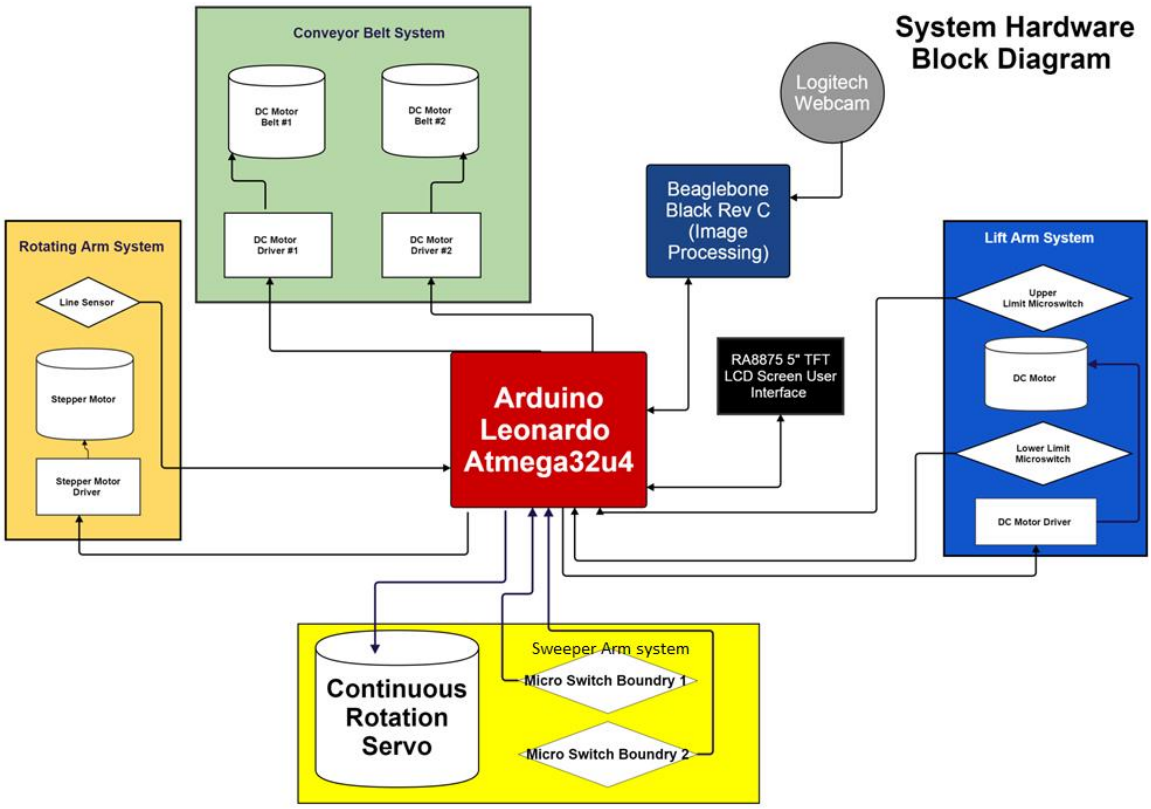


Figure 4.1-A: Overall System Block Diagram

4.2 Hardware Design

Section 4.2 will outline the hardware design details for each of the separate subsystems. Breaking down the project into various subsystems makes it simple to test everything

individually and then when each part is working on its own the subsystems will be brought together at the end of the construction process.

4.2.1 Conveyer Belt System Design

The conveyer system is comprised of two belts driven at different speeds. The bottom belt moves at a quicker speed than the top belt to allow for separation of clumped LEGO pieces. Each belt is animated by its own 12 V motor connected to a wheel shaft. Figure 4.2.1-A below shows a physical layout of the conveyor belt system viewed on profile.

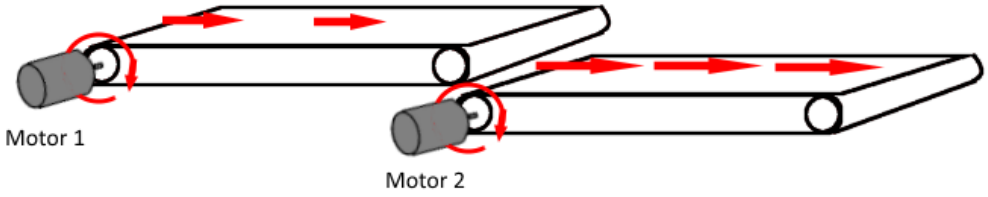


Figure 4.2.1-A: Conveyor belt system with belts, belt motors, and IR reflectance sensors.

The motors connect to driver circuitry, which in turn is connected to and controlled by the main microcontroller. A general block diagram of the conveyor belt system electronics is shown below in Figure 4.2.1-B.

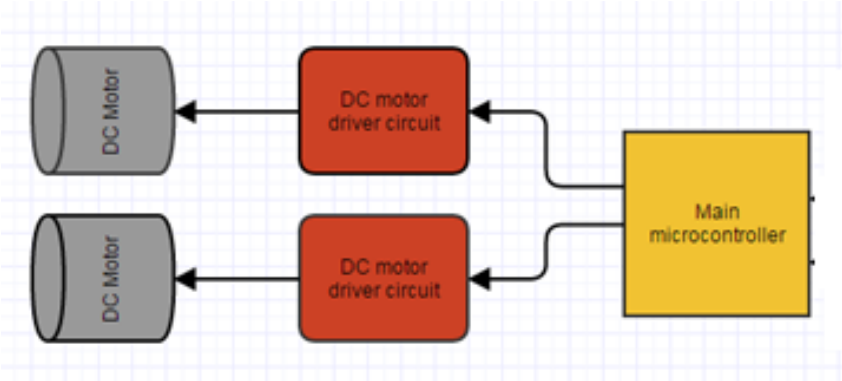


Figure 4.2.1-B Conveyer belt system circuitry block diagram

Motor Driver Circuitry Design - Motor circuitry design is identical for both of the conveyor belt motors. The conveyer belts need only move in one direction, so advanced circuitry like an H-bridge for bi-directional movement is not needed. A simple transistor switch circuit allows for a small signal from the main microcontroller to control the motors. A schematic of the DC motor circuitry is shown below in Figure 4.2.1-c.

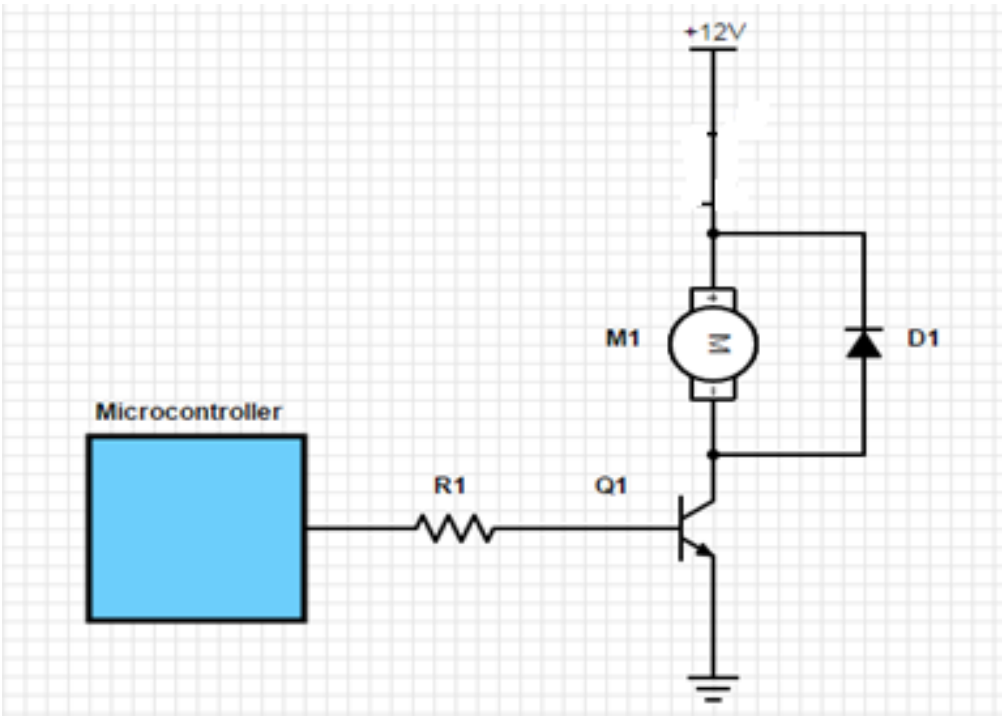


Figure 4.2.1-C DC motor driver circuit

Besides a transistor, Figure 4.2.1-C above shows resistor R1, diode D1, transistor Q1, and motor M1. The transistor acts as a switch when given a signal from the main microcontroller. The signal is a repeating pulse that has its width modulated to control the motor speeds. When the transistor base receives a 5V high pulse from the microcontroller signal, current flows through the motor to turn the conveyor belts. The transistor used is a TIP120, which is a Darlington pair circuit packaged into a three legged piece. Diode D1 acts as a flyback diode to suppress voltage spikes from the motor when it is turned on and off. Resistor R1 biases the signal from the microcontroller so that the transistor is in complete saturation when the signal is pulsed high. A 2.2k-ohm is used in the simulation as a current limiting resistor. The simulation of the DC driver circuit is shown below in Figure 4.2.1-D.

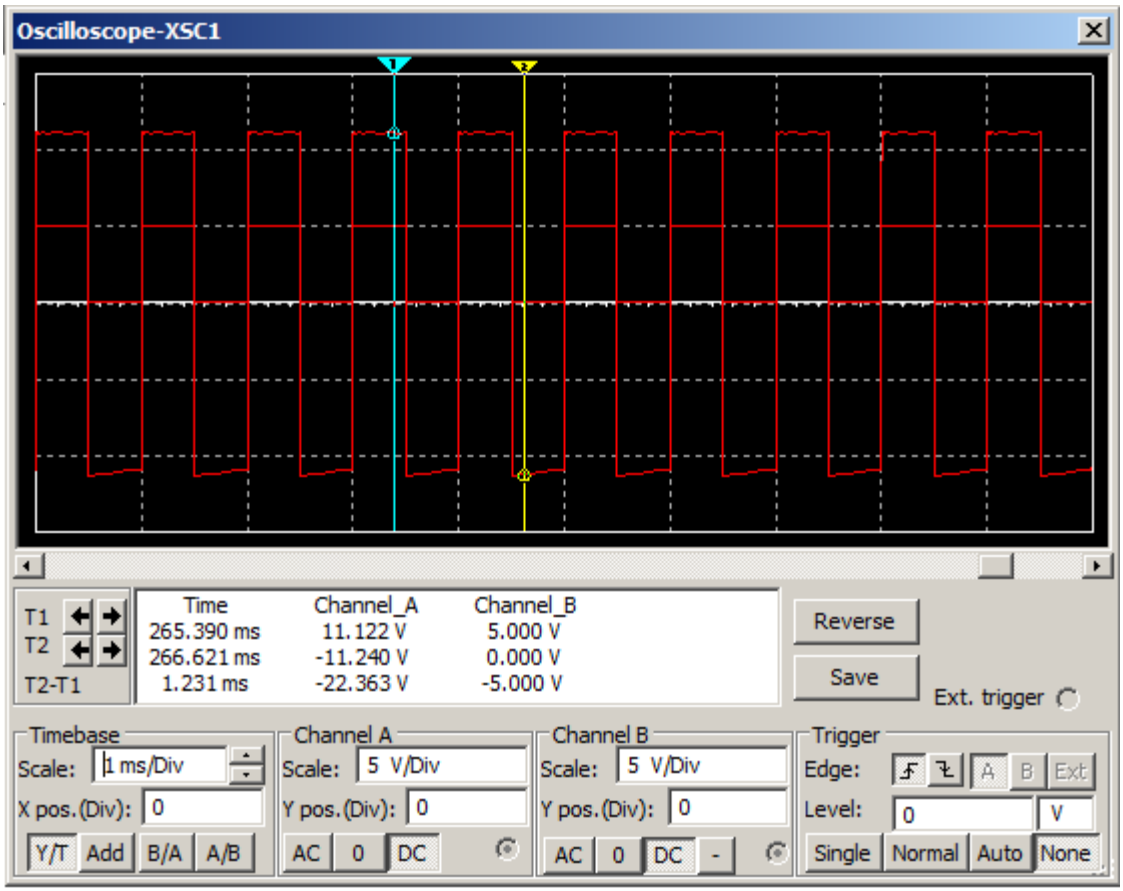


Figure 4.2.1-D Simulation of DC driver circuit

A pulse period of 0.5 milliseconds with 50% duty cycle was used in the simulation to mimic a PWM signal from the microcontroller. The simulation shows a voltage of 12V applied across the motor when a high pulse of 5V is sent to the transistor base. Channel B in Figure 4.2.1-D above refers to the input signal from the MCU. Channel A refers to the voltage across the motor. The inductor model parameters were changed to match the selected motor specifications, which are shown below in Figure 4.2.1-E below.

Parameter	Value
Rated voltage	12V
Rated current	0.33A
Winding resistance	32.6Ω
Winding inductance	48mH

Figure 4.2.1-E Electrical characteristics of the motor

Looking again at Figure 4.2.1-C, the 12V source comes from the power supply. A parts list of the schematic in Figure 4.2.1-C is shown below in Figure 4.2.1-F.

Schematic part	Description	Value
Q1	NPN Darlington pair	
R1	Resistor	2.2k-ohm
D1	Schottky diode	
M1	12V DC motor	

Figure 4.2.1-F: Conveyor belt component descriptions and values

The input signals that control the DC motor driver come from the main microcontroller. The microcontroller will have a pulse width modulation (PWM) peripheral on-board that can send a modulated signal to the motor driver. The higher the duty cycle of the modulated signal, the faster the motors will turn.

4.2.2 Lift System

Overview- The lift system consists of a 5500 RPM DC motor, an L293D dual – H bridge controller, and two mechanical micro switches to control the upper and lower boundaries of the system. The Atmega32u4 takes input from the upper and lower micro switches. When the switches are triggered the Atmega32u4 sends a message to the L293D H-bridge to reverse the direction of the motor. The system block diagram for the Lift System is shown in Fig. 4.2.2-A.

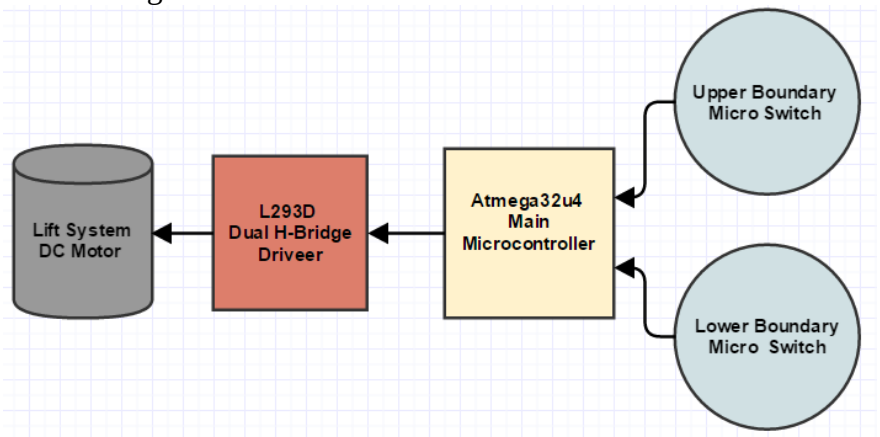


Figure 4.2.2-A: Lift System Hardware Block Diagram

H-Bridge Direction Control Theory- A simple interpretation of an H-bridge can be thought of as 4 switches connected to a motor. When S1 and S3 are closed, the motor is off since there is no connection to ground. When S2 and S4 are closed, the same is true since the motor has no supply voltage connected. When S1 and S4 are closed, the current flows from the 12 V input, through S1, the motor, and S4 to ground. This turns the motor forward. When S3 and S2 are closed, the current flows from the 12V input, through S3, the motor, and S2 to ground. This causes the motor to reverse direction. The diagram of H-bridge direction control can be seen in Figure 4.2.2-B.

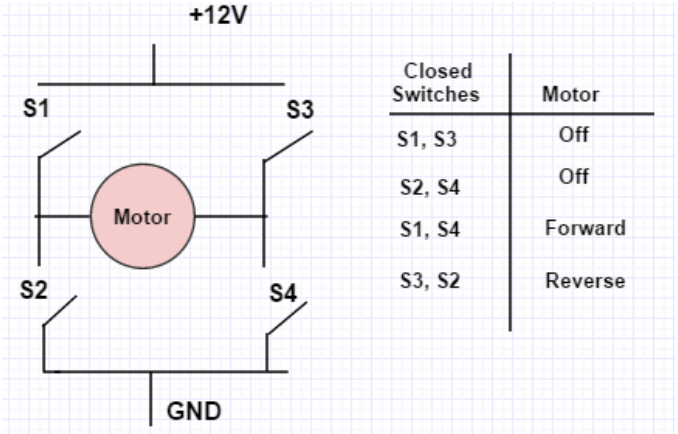


Figure 4.2.2-B H-Bridge Direction Control

L293D H-Bridge Control- The L293D operates at 5V. The enable line for the L293D also operates at 5V. The DC Motor is connected to output 1 and output 2 of the chip. The motor runs on 12V. Input 1 and Input 2 are the control lines fed to the Atmega32u4 main microcontroller as seen in Fig 4.2.2-B.

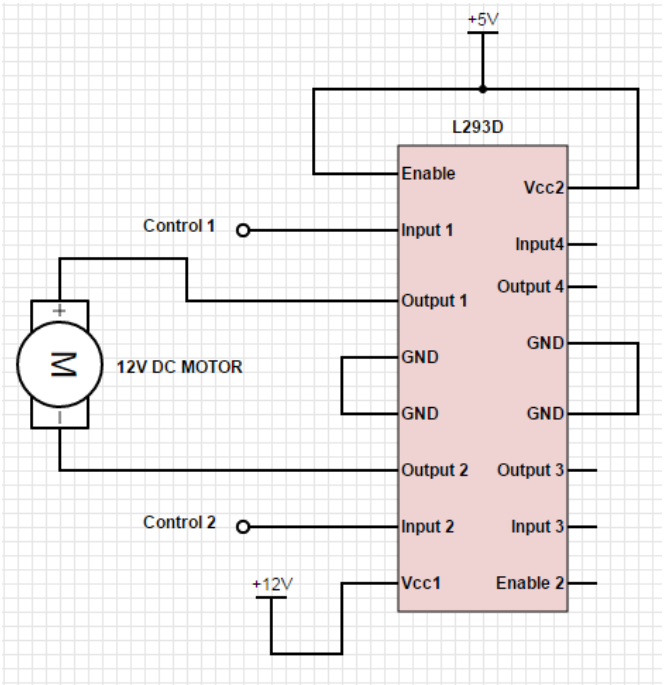


Figure 4.2.2-B Lift arm L293D Pin Diagram

4.2.3 Rotational Arm Design

The rotating arm's task is to position itself over the proper bin to allow the identified LEGO piece to fall in. Specifications for the rotating arm include the ability to rotate 360 degrees and have an accuracy of at least 10 degrees. Stepper motors offer 360 degree continuous rotation as well as incremental and accurate stepping. Because the rotating arm shaft is light-weight and can be coupled directly to the motor, a high voltage, high torque motor is not necessary. As such, the small 5V rated 28BYJ-48 stepper motor is used to move the rotating arm. While stepper motors can be very accurate, they lack sense of relative positioning. With extended use the rotating arm will become more and more inaccurate, straying further and further from the correct positions. A simple but effective solution has been developed to solve this issue. A sensor must be used to provide positioning feedback to the system. The method used in this project utilizes an infrared reflectance sensor and a white disk with an IR absorbing line as shown in Figure 4.2.3-A below.

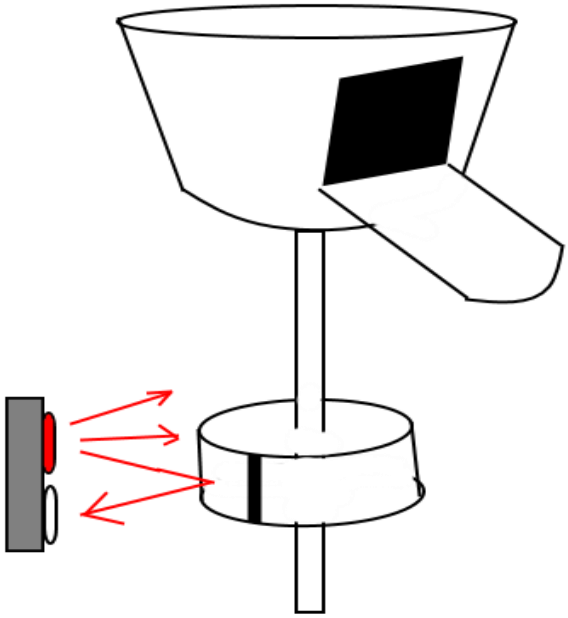


Figure 4.2.3-A: Rotating arm sensor feedback

The IR absorbing line on the white disk provides an identifying point or "home" position for the sensor to read. To home the rotational arm, the arm is rotated and a polling loop is ran on the sensor until the home line is read. When the sensor outputs a significantly smaller analog value from the black line, the rotational arm is at home position. From this point, the MCU can calculate and send the stepper motor the number of steps required to get from bin to bin. The stepper motor's gear ratio is approximately 64:1 on full step mode. Because the exact ratio is not an integer, the motor loses accuracy over time. This is a relevant consideration for this project as the sorter is to be left on for extended periods of time. To mitigate this issue, the motor is sent to home position periodically between sorting pieces.

Sorting time is also an issue for this project. The movement of the rotating arm is optimized to ensure the arm never moves more than half way around when moving from bin to bin. The code calculates the two distances between the current location of the arm and the target bin. Figure 4.2.3-B below describes visually these two distances.

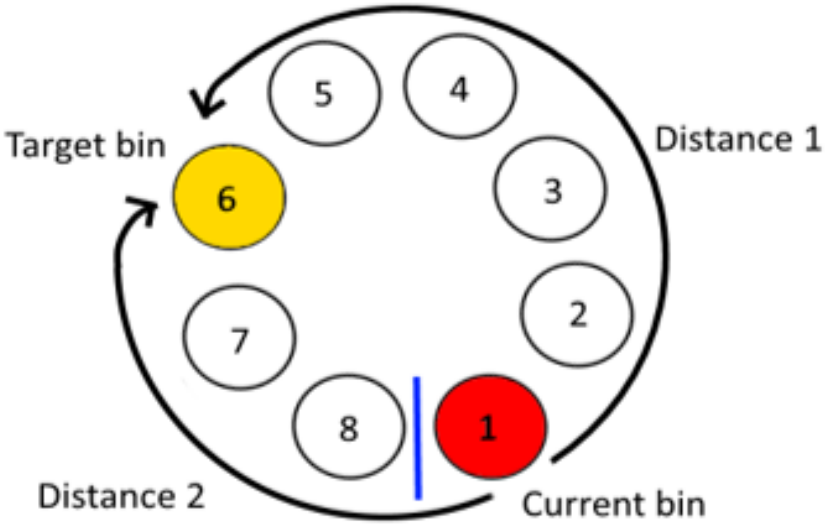


Fig. 4.2.3-B The rotating arm calculates the shortest distance between the current location and the destination.

With an 8 bin configuration, the maximum distance the rotating arm will move during any bin to bin movement is 4 bins. In Figure 6 above, the arm will take the path "Distance 2" because it is the shorter path. The distances are calculated with the following two equations:

Distance1 = bigger – smaller **(1)**

Distance2 = (number_of_bins - bigger) + smaller **(2)**

In the equations above, the term "bigger" refers to the higher bin number of either the current bin or target bin. Likewise, the term "smaller" refers to the smaller of the two bins. The algorithm calculates both distances, and through a series of nested if statements, it determines which direction it should rotate. The determined bin distance is then converted from bin numbers to number of steps the motor must take to reach the target using equation (3) shown below.

Number_of_steps = Distance*(512/number_of_bins) **(3)**

The 512 term comes from the number of steps it takes the stepper motor to make a full revolution using a half step coil energizing sequence. With these equations the motor takes the least time and least amount of movements to get from current bin to target bin.

Stepper motor driver circuitry – The detailed block circuitry diagram of the rotational arm system is shown below in Figure 4.2.3-C.

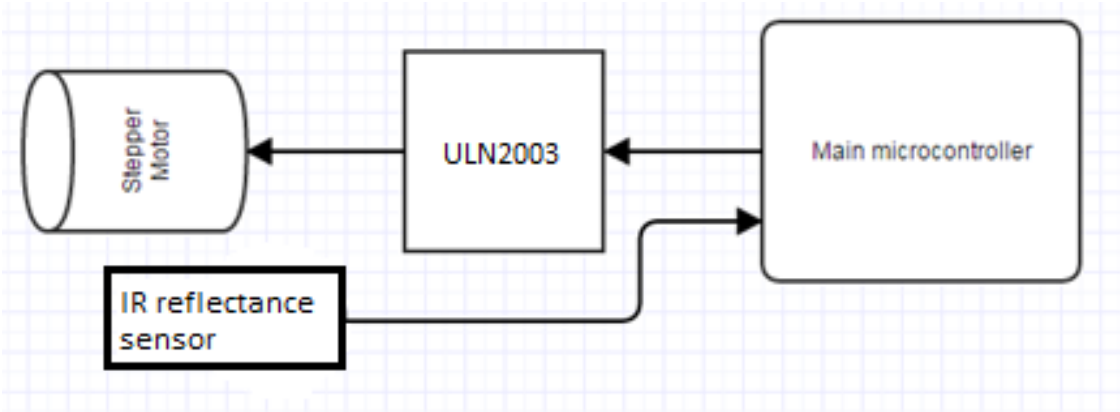


Figure 4.2.3-C Rotational arm system block diagram

The block diagram shows the data path of the various components. When the LEGO piece has been identified by the image processing system, the main controller feeds signals to the stepper motor driver which turns the arm to the correct position. The IR sensor provides feedback to the microcontroller to ensure correct positioning. Stepper motors are more complex than the standard DC motor as there multiple poles to energize to step the motor into each position. The motor used is a 12V rated unipolar stepper motor with an accuracy of up to 0.7° degrees per step in half-step mode. The MCU selected cannot handle the current that the motor draws. The ULN2003 Darlington array package has 7 Darlington transistor pairs for a maximum total of 7 inputs and 7 outputs. The stepper motor used has 4 control wires to turn the motor shaft and thus only uses 4 of the input and 4 of the output pins on the ULN2003. The ULN2003 Darlington transistor array IC pin-out and connection diagram is shown below in Figure 4.2.3-D.

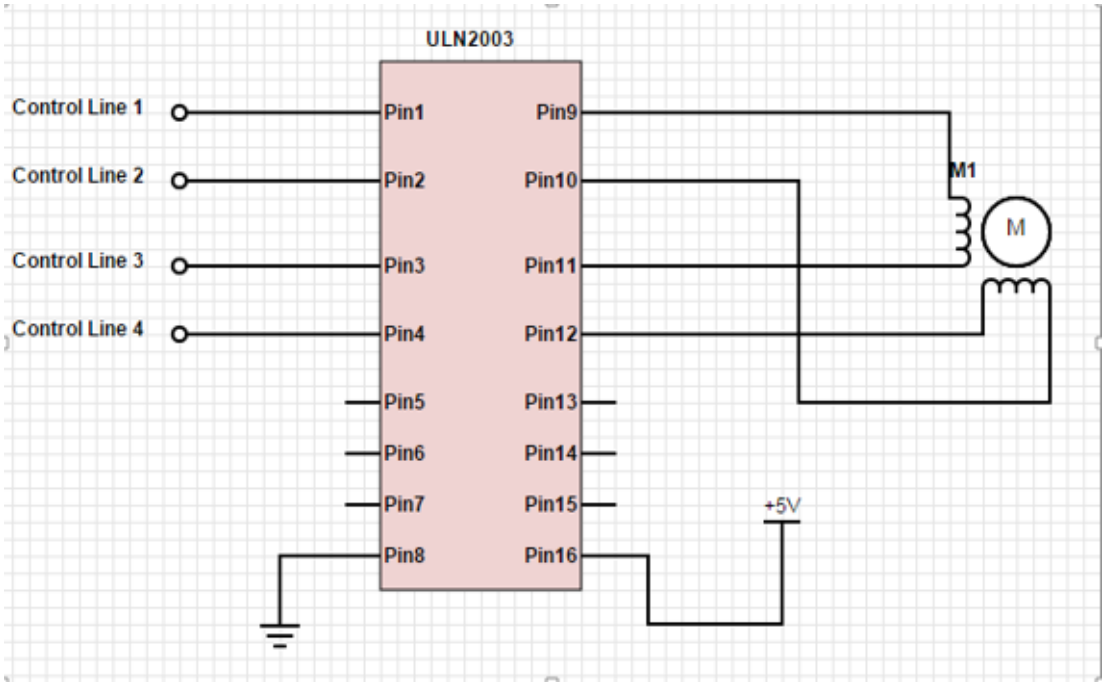


Figure 4.2.3-D ULN2003 IC pin-out and connection diagram

The ULN2003 IC acts as a buffer in between the low voltage logic signals from a microcontroller and high voltage high, high current power supply for the motor. Figure 4.2.3-E below describes each pin on the IC and its function in the system.

Pin	Label	Description
1	IN 1	Control signal 1
2	IN 2	Control signal 2
3	IN 3	Control signal 3
4	IN 4	Control signal 4
5	IN 5	Not used
6	IN 6	Not used
7	IN 7	Not used
8	GND	
9	OUT 1	Motor wire 1
10	OUT 2	Motor wire 2
11	OUT 3	Motor wire 3
12	OUT 4	Motor wire 4
13	OUT 5	Not used
14	OUT 6	Not used
15	OUT 7	Not used
16	COM	Motor power +5V

Figure 4.2.3-E ULN2003 pin description

IR Reflectance Sensor - The sorter arm utilizes an IR reflectance sensor to ensure the system has relative positioning. The sensor chosen is a QRE1113 which is shown below in Figure 4.2.3-F.

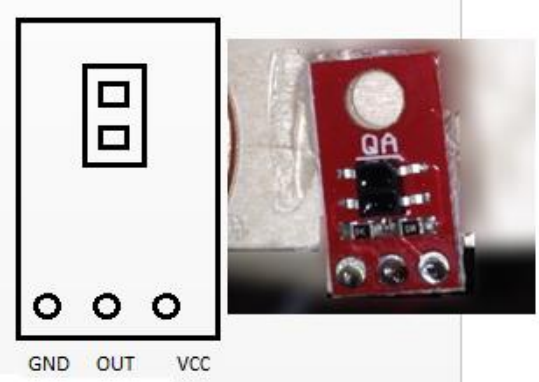


Figure 4.2.3-F Pin diagram of QRE1113 IR reflectance sensor

The sensor has three pins. GND is the ground pin, OUT is the analog output signal pin, and VCC powers the IR LED and receiver. The VCC pin is connected to the MCU's 5V reference supply. The analog output signal from the sensor connects to an analog to digital converter(ADC) channel on the MCU. The MCU's ADC converts the analog signal into a digital value between 0 and 255 (inclusive).

4.2.4 Image Processing Design

The Beaglebone Black Rev C is the device used to complete the image processing task of the LEGO pieces. An image of the Beaglebone Black is shown below in Figure 4.2.4-A.

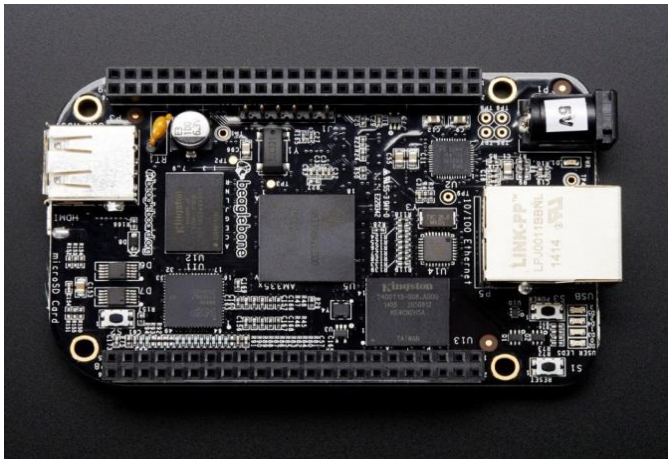


Figure 4.2.4-A Beaglebone Black Rev C

The electrical connections and design are minimal for the Beaglebone. It is powered from the 5V power supply line. Because the Beaglebone Black uses 3.3V logic and the ATmega32U4 uses 5V logic, a logic level shifter is needed for communication between the two devices. The logic level shifter used is shown below in Figure 4.2.4-B.

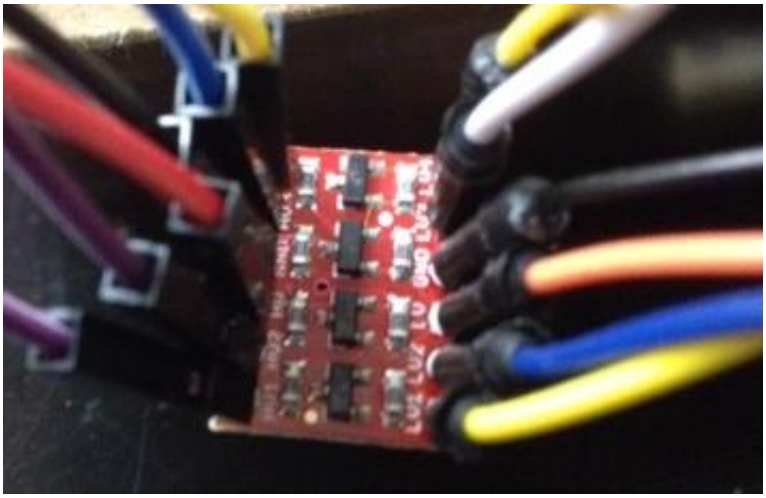


Figure 4.2.4-B Logic Level Shifter

The logic level shifter has 4 bi-directional communication channels, of which two will be used for UART interfacing between the Beablenone and ATmega32U4. Along with the communication channels, the logic Vcc from each device must be connected to the respective pins on the logic level shifter. A webcam is also connected to the Beaglebone through USB interfacing. The Beaglebone has two USB ports already built on the board as well as available pins for UART interfacing. A wire diagram of all the connections is shown below in Figure 4.2.4-C.

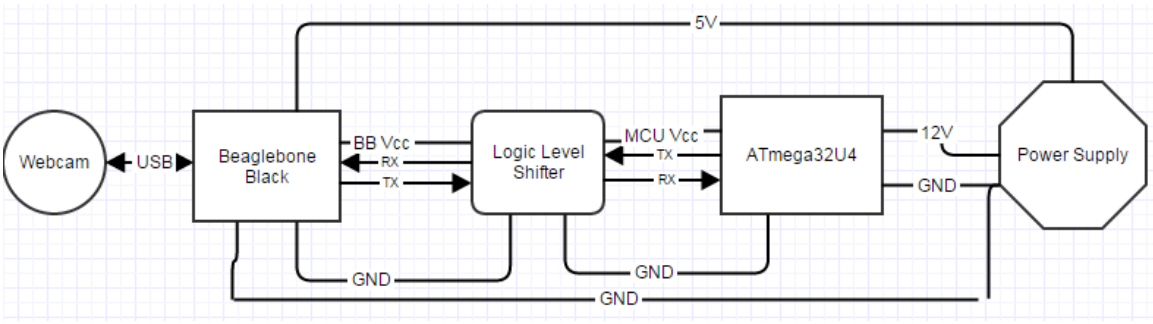


Figure 4.2.4-C Wiring diagram for the Beaglebone Black

The camera selected to feed images into the Beaglebone is the Logitech C110. The camera was selected because of the good online support for Logitech products. The C110 is a low resolution VGA camera as high definition is not needed for this project's purposes. The camera is shown below in Figure 4.2.4-C.

Figure 4.2.4-D Logitech C110 webcam

PCB Software - The PCB vendor chosen is 4PCB. The chosen software is Eagle PCB Design. The software has extensive libraries for relatively quick designing. The software allows the designer to first make a schematic layout of the circuit with traditional circuit symbols and wiring. The software can automatically create a rough PCB layout from the schematic, which then requires the designer to place components and reroute wiring where needed. The software is free to download and use for. Eagle PCB will be used for the PCB design layout of the project.

Rotating Arm Testing - The stepper motor requires a specific sequence of signals to operate as intended. Testing will begin once it is completely hooked up as designed. After hooking up the stepper motor and driver circuit to the ATmega32U4 microcontroller and power supply as shown in Figure 4.2.4-B, the code will be written. Testing will make sure the timing in the code allows for smooth operation of the stepper motor. Care will be taken to make sure delays are put in the code to accommodate for the slower electro-mechanical processes that occur in the motor. Being that the motor has 1.8° steps, it should not be a problem to get the rotating arm slide to line up with each LEGO bin. Testing will make sure it does any way. The color sensor on the rotating arm will also be tested. It will be connected via I2C with the microcontroller before testing. The sensor has a register and some integrated features that will be read into to properly test its functionality

4.2.5 Main microcontroller Design

The main microcontroller is the central command of this project. It is in charge of taking in sensor and image processing data, and controlling the conveyer belt, lift arm, and rotational arm accordingly. The microcontroller chip used is the ATmega32U4.

The ATmega32U4 has all the peripherals and technologies needed to control each of the subsystems. The project demands use of the MCU's PWM channels, ADC channels, general purpose digital I/O, SPI and UART serial interfacing. The pinouts of all the subsystems are designed around these features. Figure 4.2.5-A below shows the ATmega32U4 physical pin layout as well as each pin's available functionalities.

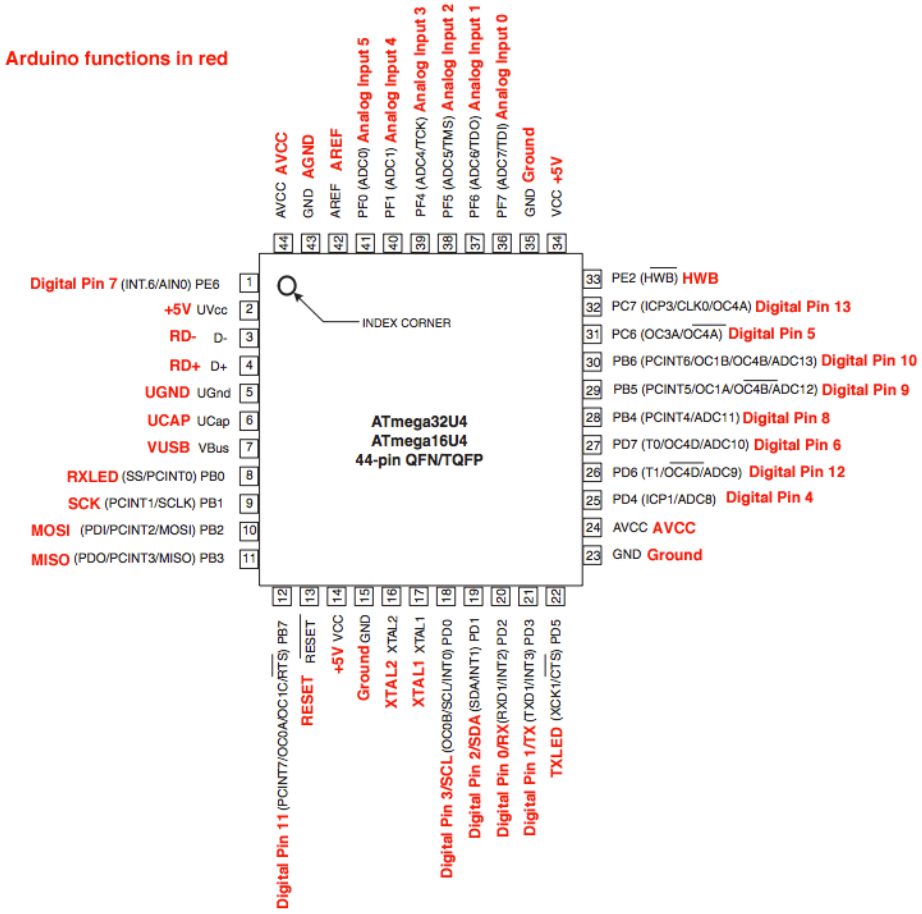


Figure 4.2.5-A ATmega32U4 pin layout and functionality (permission pending)

The LEGO sorter devices and their pin requirements of the ATmega32U4 MCU are shown below in Figure 4.2.5-B.

Device	Pin count	Pin function(s)
Conveyor Motor 1	1	PWM
Conveyor Motor 2	1	PWM
Rotating sorter motor driver	4	Digital I/O
IR reflectance sensor	1	ADC channel
Lift system motor driver	2	Digital I/O
Boundary switch 1	1	Digital I/O
Boundary switch 2	1	Digital I/O
Beaglebone Black	2	UART
LCD screen	4	SPI
LCD touch signals	2	Digital I/O, interrupt

Figure 4.2.5-B Devices and their pin requirements of the ATmega32U4 MCU

In total, 19 data pins are required to control all of the LEGO sorter devices. The interrupt pin is used to stop the program when it detects touch on the touchscreen. The PWM pins are used on the conveyor motors allow for speed control. The ADC pin on the IR sensor converts the analog signal into a digital signal that the MCU can interpret. The digital I/O pins can send or receive a voltage level of 5V or 0V. The UART and SPI are communication pins that will allow data transfer to/from the MCU to the Beaglebone and LCD screen respectively. The MCU and subsystem circuitry was designed in Eagle PCB software. The total embedded schematic is shown below in Figure 4.2.5-C.

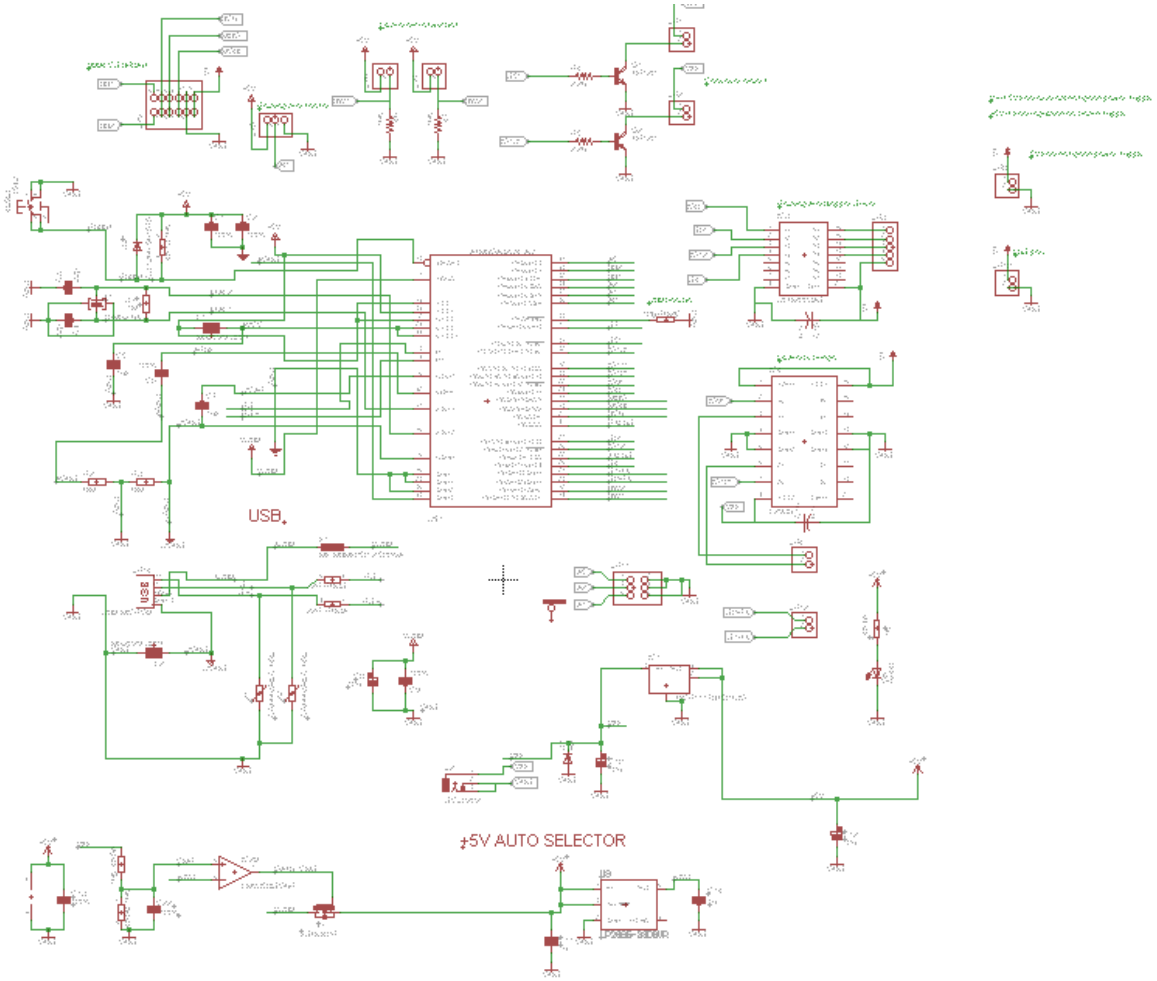


Figure 4.2.5-C ATmega32U4 and subsystem schematic

The PCB board designed is shown below in Figure 4.2.5-D.

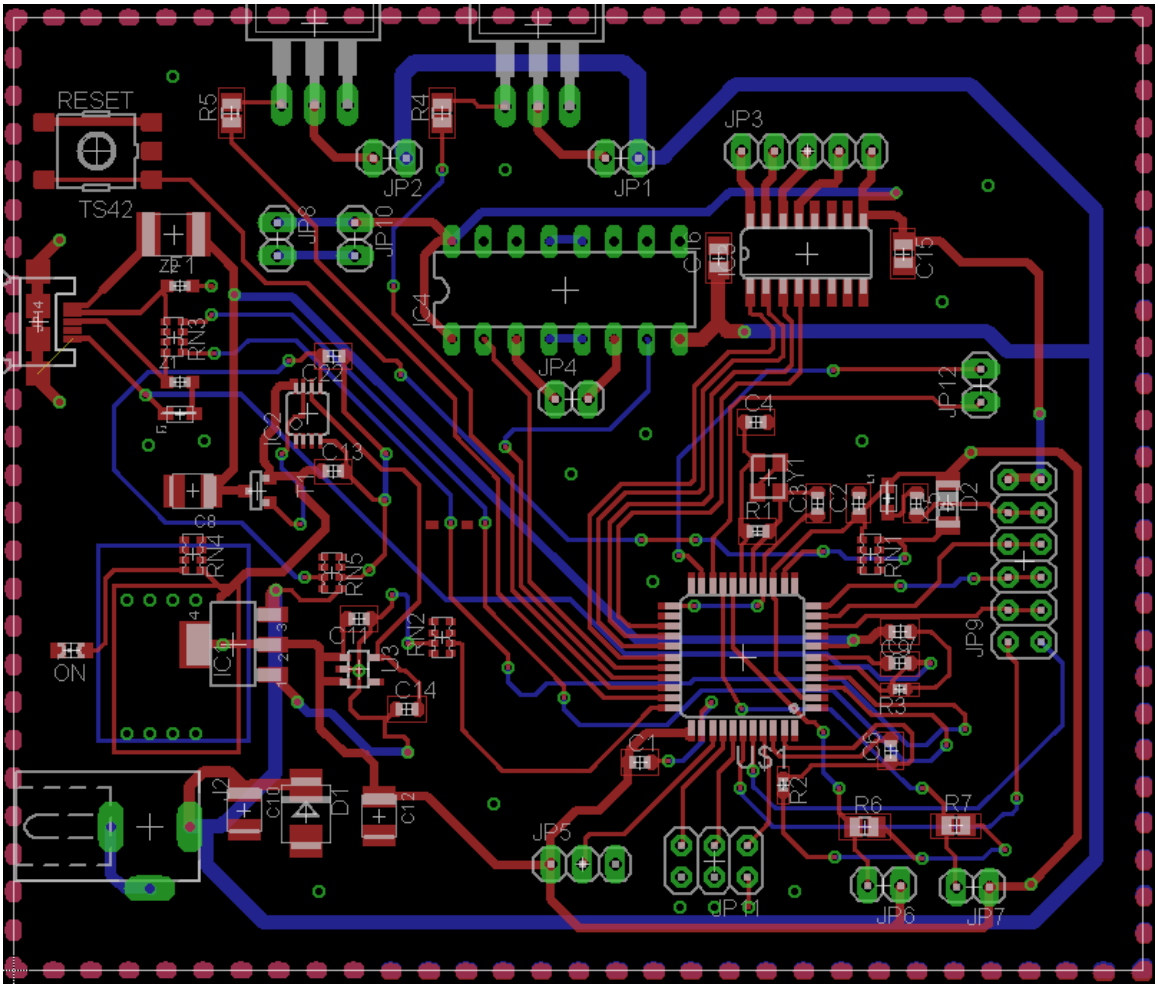


Figure 4.2.5-D ATmega32U4 and subsystem PCB design

The PCB is designed with the MCU, supporting power and USB components, and the subsystem hardware circuitry. Pin header pads were designed on the PCB for easy connectivity with the subsystem hardware such as the motors and sensors.

4.2.6 LCD

The pinouts in this section come from the RA8875 controller for the TFT LCD display specifications PDF. Some sections of specifications have been passed over. These sections specify a default setting for all pinouts and therefore will not be discussed and will be left to their default settings.

SPI MCU Interface to Atmega32u4- Since there are three devices that are to be interfaced to each other using SPI, a 4 wire connection is required. The 4th connection is for the clock line. A 3 wire connection is only compatible if two devices are to be interfaced using SPI. Figure 4.2.6-A shows the pins for SPI interface of the RA8875 controller to be configured.

Pin Name	I/O	Pin Description
SCL	I	*SPI Clock* 4- Wire Serial
SDI	I/O	*4 Wire SPI Data Input* 4 Wire SPI I/F: Data input for serial I/F
SDO	I/O	*4 Wire SPI Data Output* 4 wire SPI I/F: Data output for serial I/F
SCS#	I	*SPI Chip Select* Chip select pin for 4-wire serial I/F
IICA[1:0]	N/A	*IIC I/F: IIC Address Select* Other I/F: NC, please don't keep floating
SIFS[1:0]	I	*Serial Interface Selection* 00:NC 01: 3 Wire SPI 10: 4- Wire SPI 11:IIC If serial I/F is no use, please connect them to 00

Table 4.2.6-A LCD RA8875 Controller Atmega32u4 SPI Interface

The chart includes some IIC connections which will not be used so NC will be needed for these pins. For the SIFS pin the logic will be set to "10" or binary "2" for 4-WIRE SPI interface. The RA8875 operates as a slave pin to the Atmega32u4. The SPI can be configured in command/data write mode or status/data read mode by setting the MSB two bits of first byte of protocol as seen in Figure 4.2.6-B.

Cycle Type	RW#	RS	Description
Command Write	0	1	Register number write Cycle
Status Read	1	1	Status read cycle
Data Write	0	0	Corresponding Register data/memory data write cycle following the command write cycle
Data Read	1	0	Corresponding Register data/Memory data read cycle following the Command Write cycle

Table 4.2.6-A: Read/Write Ra8875 Cycle Commands

Since the RA8875 acts as a slave controller it has 3 input lines coming from the Atmega32u4 SCS (chip select), SCL (SPI clock), and SDI (data input). The RA8875 has 1 output line to the Atmega32u4 SDO (data output). This is shown in figure 4.2.6-C below. The serial interface selection line (SISF1) and the power supply are connected to Vdd and the serial interface connection line (SIFSO) must be connected to ground according to Figure 4.2.6-C.

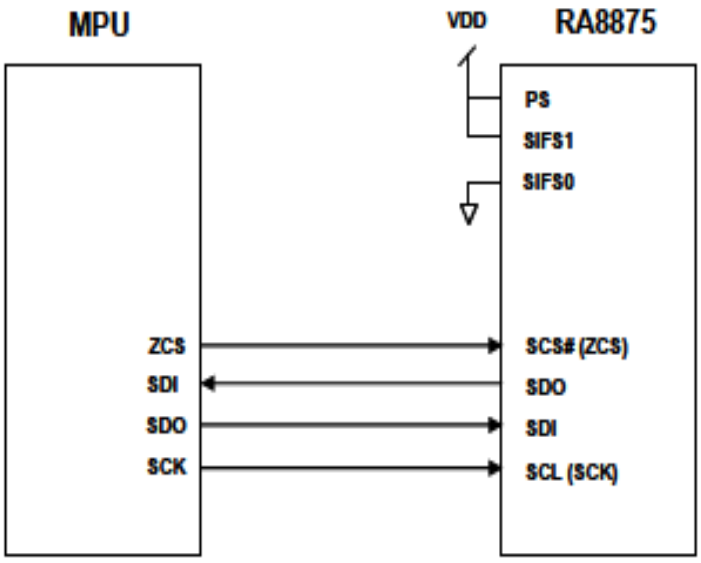


Figure 4.2.6-B Atmega32u4 Interface 4 Wire SPI

RA8875 Pin Configurations - Serial interfaces on the RA8875 are disabled by default. The following jump point connections need to be made to enable the device for 4-wire SPI interfacing:

Short Circuit	J10, J12, J14, J16
Open Circuit	J9, J12, J14, J15
Resistor Values	R1= 10k, R2= 10k, R3=10K

Table 4.2.6-C Jump Point Connections to enable 4-Wire SPI on the RA8875 Controller

The Jump Point 1 / Connection 1 (JP1/CON1) pins of the RA8875 controller need to be configured for 4-Wire SPI interface according to Figure 4.2.8-D.

Pin #	Symbol	Description
1,2	Vss	Ground
3,4	Vdd	Power Supply
5	/SCS	SPI Chip Select Chip select pin for 4 wire SPI
6	SDO	4 –wire SPI I/F: Data Output for serial I/F
7	SDI	4-wire SPI I/F: Data input for serial I/F
8	SCLK	SPI Clock 4-wire Serial I/F Clock

Table 4.2.6-D Pin Configuration – JP1/Con1 Four Wire SPI

Summary - The datasheet for the RA8875 controller provided by buydisplay.com was followed carefully to initialize the appropriate pin configurations for a 4-wire SPI interface with the Atmega32u4. There are 3 PDF files as well as some example code to get the LCD display up and running. This section includes an overview of interface protocol for the RA8875 controller but may not include all the initializations that need to be made.

4.2.7 Sweeper Arm System

Overview- The sweeper arm system utilizes a continuous rotation servo motor. The servo motor has three lines: +Vcc, Ground, and a control line. The Atmega32u4 main MCU controls the direction and speed of the servo. The boundaries of the servo motor are controlled by two mechanical micro switches. The mechanical micro switches are inputs to the Atmega32u4 as shown in Fig 4.2.8-A.

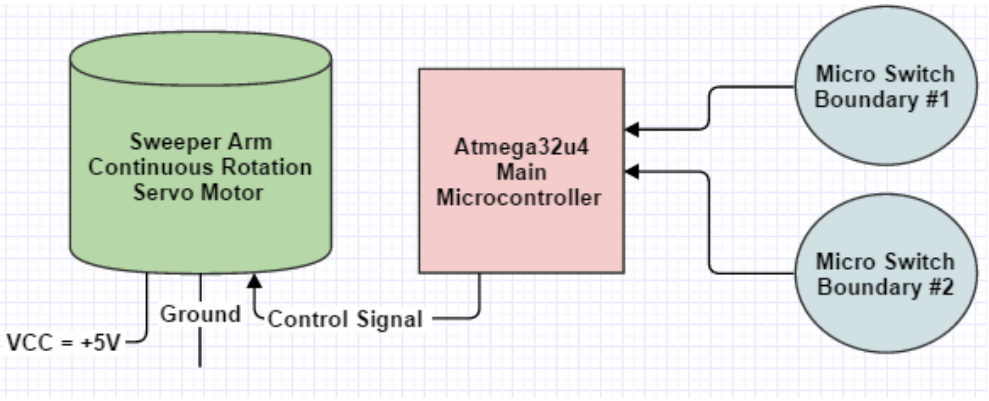


Figure 4.2.8 – A: Sweeper Arm Subsystem Hardware Block Diagram

Continuous Rotation Servo Motor Control- The continuous rotation servo operates much like the positional rotation servo except there is no limit on its' range of motion. It either rotates CW, CCW, or has no rotation. Instead of PWM correlating to fixed positions at pulse width integers of 1ms, 1.5ms, or 2ms ,the continuous rotation servo uses the same three pulse width integers but includes an interval which correlates to the speed of the CW/CCW direction traveled. Table 4.2.8-B shows the direction and speed control of the Parallax S148 continuous rotation servo.

Input (ms)	Rotation Speed (%)	Direction of Rotation
1.0	100	Clockwise
1.1	80	Clockwise
1.2	60	Clockwise
1.3	40	Clockwise
1.4	20	Clockwise
1.5	0	N/A
1.6	20	Counter-Clockwise
1.7	40	Counter-Clockwise
1.8	60	Counter-Clockwise
1.9	80	Counter-Clockwise
2.0	100	Counter-Clockwise

Table 4.2.8-B: Continuous Rotation Servo Motor Speed and Direction Control

Software- The Atmega32u4 will use polling to detect whether or not the switches have been triggered. Interrupts were originally used in testing. The problem with interrupts is that they stop all control of the main MCU in order to read the switches. The switch under the belt will be set low and the switch at the rotating arm output will be set high. When the switch under the belt is triggered it will go from low to high. When the switch at the rotating arm is triggered it will go from high to low. At this point the Atmega32u4 will reverse the direction of the servo until it hits the switch under the second conveyor belt, at which point the sweeper arm will stop underneath the second conveyor belt while another LEGO is being processed.

4.2.8 Power Supply Hardware Design

Overview- As stated in section 3.2.8, the power supply to be designed was to have two DC supply voltages.

- Vo1 = 12V rated at 1.7A
- Vo2 = 5V rated at 3.41 A

The AC to DC conversion from the 115 V AC 15 A standard wall supply was purchased from Jameco and manufactured by Mean Well. The input to the Mean Well supply is a standard two prong live, neutral connection.

Webench Design Tool – The two DC Load voltages for the project were designed using the Webench DC Power Architect tool provided by Texas Instruments (T.I.). The DC input source to the two supplies to be designed comes from the Mean Well Open Frame AC-DC SMPS. The design input is shown in Fig.4.2.8-A.

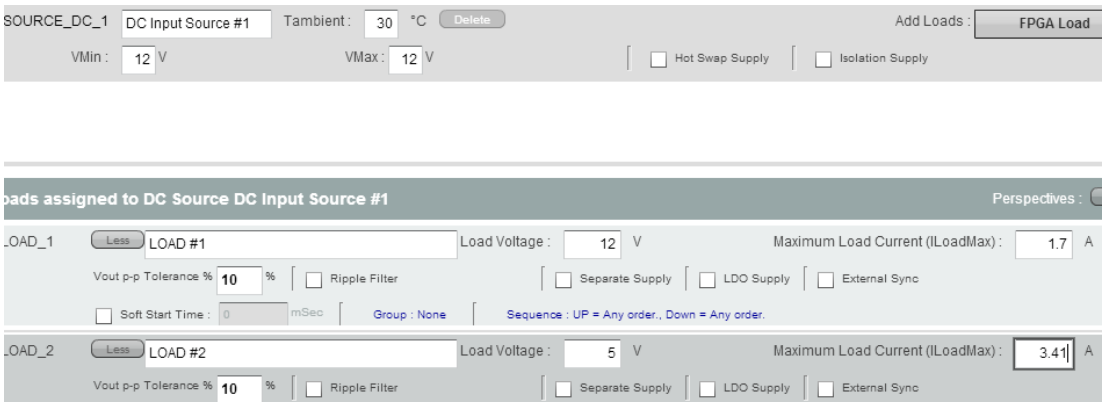


Figure 4.2.8-A Webench DC Power Architect Parameters.

The Webench DC power Architect tool generated the appropriate current needed for the two DC voltage supplies to be designed. The current needed to supply the two DC Supply voltages was rated at 3.33A. The Mean Well AC-DC supply selected has a slightly higher current rating at 3.7A as shown in Fig. 4.2.8-B. The input to the two voltage supplies was attached to a SPST standard 120VAC 15A switch to cut- off power to the system. The total system efficiency is 93.79%, with a footprint of 458 mm^2, and total system power dissipation of 2.478 W.

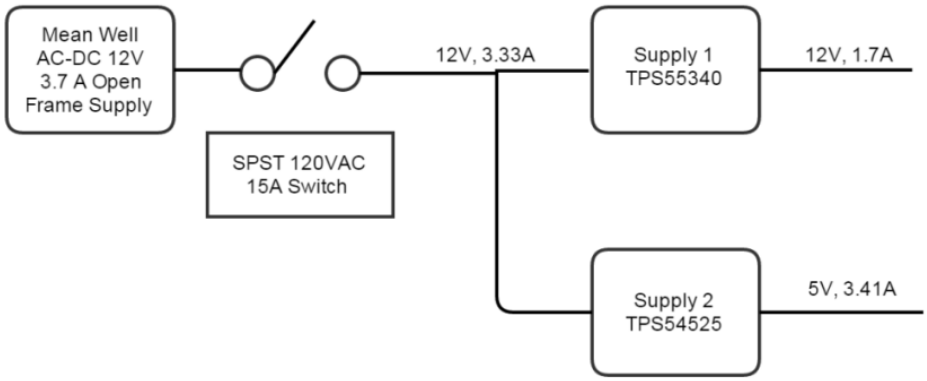


Figure 4.2.8-B: Overall Block Diagram of Power Supplied to Project

Supply 1 Eagle Schematic - The 12V, 1.7A supply that was designed is a TPS55340RTER WQFN 16 Pin package Boost converter shown in Fig 4.2.8-C.

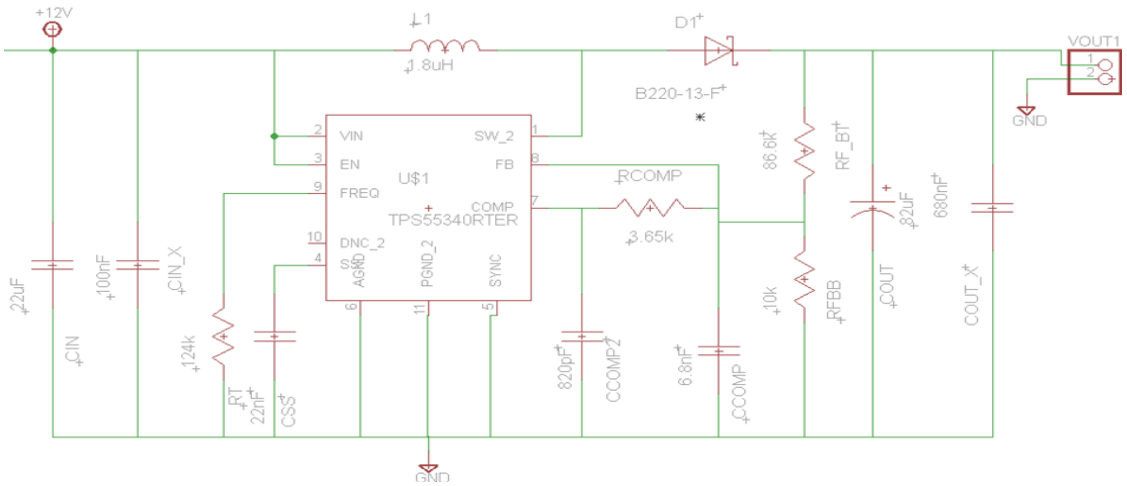


Figure 4.2.8-C: Supply 1 TPS55340RTER Eagle Schematic

Vo1 Operation- The TPS55340RTER regulates the output with pulse width modulation (PWM) control. The 12V 3.7A input supplied by Mean Well is applied across the inductor. The pwm controller is governed by an internal oscillator clock cycle. The switch is turned on by the PWM control block at the beginning of each clock cycle supplied by the oscillator. The switch turns off when the inductor current reaches a certain threshold level. The schottky diode becomes forward biased, allowing current to flow to the output capacitor. The duty cycle is determined by the PWM control. The frequency of oscillation is set by the external resistor R_t. An RC network is utilized for feedback loop stability and transient response optimization on the compare (COMP) pin. The output of the internal error amplifier is connected to the PWM control feedback loop that regulates the forward bias pin, (FB). The internal block diagram of the chip from the TPS55340 datasheet supplied by T.I. is shown in Fig. 4.2.8-D. **[1]**

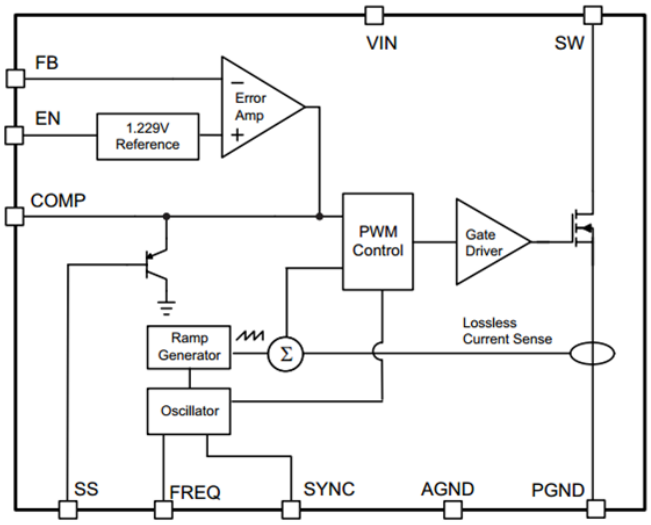


Figure 4.2.8 – D: TPS55340RTER Internal Block Diagram.

Vo1 Component Values – Although the Webench DC power architect tool provides the values needed to build the circuit, the datasheets for the TPS55340RTER and the TPS54525PWPR were thoroughly checked for consistency.

Vo1 Switching Frequency- The switching frequency is set by the resistor, Rt, connected to the (FREQ) pin of the TPS55340. The resistance, Rt, given by Webench was 124kΩ. According to the graph Fig.4.2.8-E, this corresponds to a switching frequency f_{sw} = 390 kHz

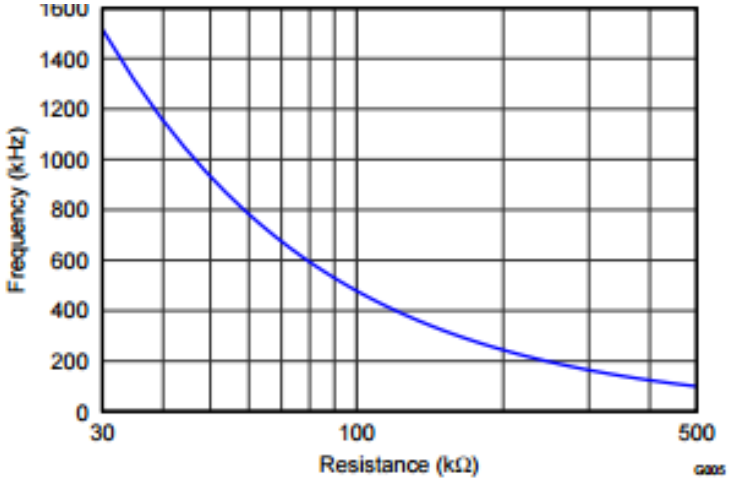


Figure 4.2.8-E TPS55340 Switching Frequency vs. Resistance

Vo1 Voltage Reference and Setting the Output Voltage- The internal error amplifier shown in Fig. 4.2.8-D provides a 1.229V voltage reference to the non-inverting input. The output voltage is set by the forward bias pin resistors, R_{fbt} and R_{fbb} according to equation (1) The desired nominal output voltage is 12V.

$$Vout = 1.229V\ x\ \left(\frac{Rfbt}{Rfbb} + 1\right) \qquad (1)$$

$$\frac{Rfbt}{Rfbb} = \frac{12}{1.229} - 1 = 8.764$$

*Letting Rfbb = 10k*Ω *, we can find Rbt = 86.6k*Ω

Vo1 Determining the Duty Cyle – When running in continuous conduction mode (CCM), the inductor maintains a minimum DC current. The duty cycle is related to the input and output voltages. According to the TPS55340 datasheet, the voltage drop v_D across the schottky diode is assumed to be 0.5V. The duty cycle is calculated from equation (2).

$$D = \frac{Vout+Vd-Vin}{Vout+Vd} \quad (2)$$

$$= \frac{12+0.5-12}{12+0.5} = 0.04 = 4\%$$

Vo1 Selecting the Inductor L1- According to the TPS55340 datasheet, in a Boost Converter, maximum inductor current ripple occurs at 50% duty cycle. For applications where the duty cycle is always smaller or greater than 50% equation (4) should be used. Recall the switching frequency was previously found to be around 390 kHz. K_{ind} is the coefficient that represents the amount of inductor ripple current relative to the maximum input current ($I_{inDC} = I_{LAVG}$). The maximum input current can be estimated from equation (3). The inductor ripple current, K_{ind} values in the range of [0.2 to 0.4] for CCM operation. This results in a smaller footprint and improved transient response at the expense of potentially lower efficiency. The efficiency η_{est} was conservatively chosen to be 85% in order to calculate the maximum input current $I_{inDC.}$ The design requires an output of 12V, 1.7A.

$$Iin(DC) = \frac{Vout\ x\ Iout}{\eta est\ xVin} \quad (3)$$

$$Iin(DC) = \frac{12V\ x\ 1.7A}{0.85\ x\ 12} = 2A$$

The inductor L1 can now be calculated from equation (4) with the results found in equations (2) and (3) and the frequency of the switch f_{sw}, which was previously found from figure 4.2.8-E.

$$L1 \geq \left(\frac{Vin}{IinDCx\ Kind}\right)\left(\frac{D}{fsw}\right) \quad (4)$$

$$L1 \geq \left(\frac{12V}{2A\ x\ 0.4}\right)\left(\frac{.04}{390E3}\right) = 1.5385\mu H$$

Vo1 Inductor Current Ratings –After the inductance is calculated, the required current ratings can be calculated. The inductance ripple current is calculated from eq. (5).

$$\Delta IL(Ripple) = \left(\frac{Vin}{L1}\right)\left(\frac{D}{fsw}\right) \quad (5)$$

$$\Delta IL(Ripple) = \left(\frac{12}{1.5385E-6}\right)\left(\frac{.04}{390E3}\right) = 800mA$$

The RMS and peak inductor current can be calculated from equations (6) and (7) below.

$$IL(RMS) = \sqrt{(IinDC)^2 + \left(\frac{\Delta IL}{12}\right)^2} \qquad (6)$$

$$IL(RMS) = \sqrt{(2A)^2 + \left(\frac{800E-3}{12}\right)^2} = 2.0011A$$

$$IL(peak) = IinDC + \frac{\Delta Il}{2} \qquad (7)$$

$$IL(peak) = 2 + \frac{2.0011}{2} = 3.0006\,A$$

A Bourns 1.8µH inductor with a current rating of 2.91A, and saturation current of 4.4A, and 42mOhm DCR was selected for the TPS55340 12V supply.

Vo1 Selecting the Input Capacitors Cin and Cinx – A capacitance of at least 4.7µF of ceramic material is recommended. High quality X5R or X7R ceramic capacitors are recommended to minimize capacitance variations due to temperature. Capacitors must have an RMS current rating greater than the maximum RMS inut current of the TPS55340 given by equation (8).

$$Icin(rms) = \frac{\Delta IL}{\sqrt{12}} \qquad (8)$$

$$Icin(rms) = \frac{800E-3}{\sqrt{12}} = 231mA$$

The capacitor chosen for Cin is an X5R series capacitor. The parameters of Cin are as follows:

- Cin = 22µF
- ESR = 2.0 mOhm
- VDC = 25.0 V
- IRMS = 3.67 A

The input ripple voltage is calculated from equation (9).

$$Vi(ripple) = \frac{\Delta IL}{4xfswxCin} + \Delta IL\ x\ Rcin \quad (9)$$

$$Vi(ripple) = \frac{800E-3}{4x390E3x22E-6} + 800E-3\ x\ 2E-3 = 24.9mV$$

Although a lower voltage rated capacitor could be used for Cin, a 25V rated capacitor was chosen to limit DC biasing effects. An additional capacitor, Cinx with parameters of X7R series, 100nF, ESR = 280 mOhm, VDC =25 V, and I(rms) = 0.0A was utilized close to Vin and Ground for extra decoupling.

Vo1 Setting the Soft Start Time (C_{ss})- The capacitor, Css sets the soft start time of the chip. Increasing the soft start time reduces overshoot during startup. Therefore, a longer soft start time is desired. Css was selected with the following parameters: C= 22nF, ESR = 14.635 mOhm, Vdc= 25.0V, and Irms = 0.0A. The soft start time , τ_{ss}, was calculated from equation (10) below.

$$\tau ss = ESR[Ohm]x\ Cap\ [F] \qquad (10)$$

$$\tau ss = 14.635E-3\ x\ 22E-9 = 322pS$$

The start up time of 322 [pS] was more than enough to reduce the in-rush current and output voltage overshoot.

Vo1 Computing Maximum Output Current- The maximum output current must be calculated to find appropriate output capacitor values. The maximum output current is a function of Vin, efficiency, Vout, and the minimum peak current limit, ILIM, which is given in the datasheet to be 5.25A. Iout(max) can be calculated from equation (11) below.

$$Ioutmax = \frac{Vinmin\ x\ \left(ILIM - \frac{\Delta IL}{2}\right) xnest}{Vout} \qquad (11)$$

$$Ioutmax = \frac{5\,x\,\left(5.25 - \frac{800E-3}{2}\right)x0.85}{12} \quad = \quad 1.74A$$

The calculated value of Iout(max) = 1.74 A is approximately equal to the desired output current of 1.7 A specified for the design.

Vo1 Selecting Output Capacitors – Capacitors in parallel at the output create a negligible equivalent series resistance (ESR). The output ripple voltage is related to the output capacitance and its' ESR. Use of high ESR capacitors contributes to additional ripple at the output. The datasheet specifies a maximum transient voltage ΔVtran = 960mV and ΔItran = 400mA with a control bandwidth frequency fbw = 6kHZ. From these values the datasheet requires Cout to have a minimum output capacitance of 11.1uF.Equation (12) specifies the output capacitor RMS current.

$$Icout(RMS) = Iout\,x\,\sqrt{\frac{D}{1-D}} \qquad (12)$$

$$Icout(RMS) = 1.74A\;\;x\,\sqrt{\frac{.04}{1-\;.04}} \quad = 0.355\,A$$

Cout was selected to be 82µF, with an ESR of 27mOhm, VDC = 16V, and Irms = 3.0A. Coutx was selected to be 680nF, with an ESR of 16mOhm, VDC=25V, and Irms = 0.0A. Choosing a higher DC voltage than the desired output voltage reduces DC biasing effects.

Vo1 Compensating the Control Loop – Loop compensation should be designed for minimum operating voltage of the TPS55340. The COMP pin is the output of the internal error amplifier. Rcomp and Ccomp are connected to the COMP pin to provide a pole and a zero. This determines the closed-loop frequency response needed for stability and transient response. Ccomp places a high-frequency pole in the control loop. The datasheet suggests starting with Rcomp =2kohm and Ccomp = 0.1uF. Increasing Rcomp or reducing Ccomp increases closed loop bandwidth and improves transient response. Rcomp was chosen to be 3.65kohm and Ccomp was chosen to be 6.8nF.

Vo1 Selecting the Schottky Diode D1- The average peak current ratings of the diode must exceed the average inductor output current , IL(rms) which was found to be 2.0011 A. Also, the reverse breakdown voltage of the diode must exceed the regulated output voltage which is 12V for this design. The diode chosen was a B22013-F schottky diode by Bourns Inc. The parameters are Vbr = 20V, which exceeds the regulated output of 12V with an average rectified current of 2A.

Vo1 Conclusion– The datasheet was studied carefully, and the values provided by Webench were in agreement with the values calculated from the equations provided in the TPS55340 datasheet.

<u>Supply 2 Eagle Schematic Supply 1 Eagle Schematic</u> - The 5V, 3.41 A supply that was designed is a TPS54525PWPR 5A Synchronous Step-Down DCAP2 Mode converter. The Eagle Schematic diagram is shown in Fig 4.2.8- F.

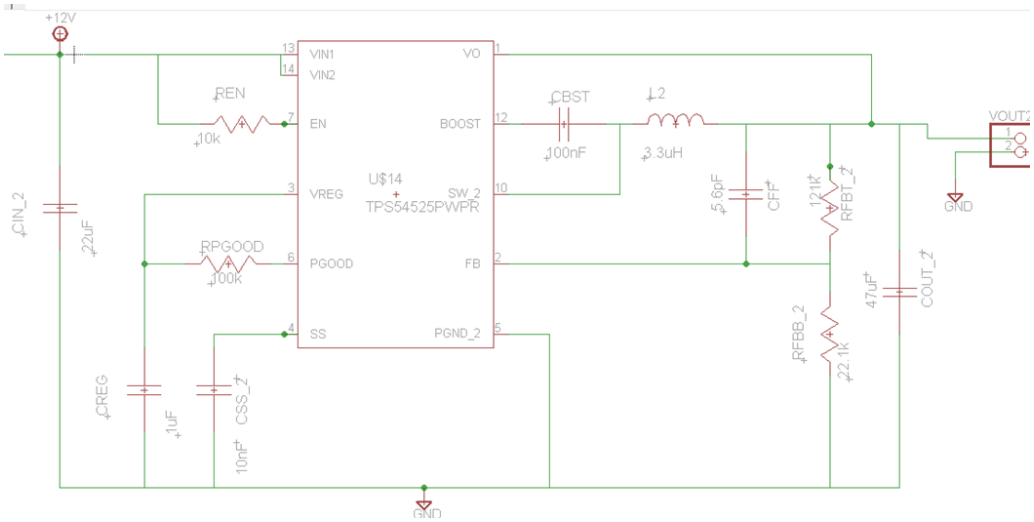


Figure 4.2.8-F: TPS54525 DC Supply 5V , 3.41 A Output

Vo2 Output Voltage Selection – The output voltage is set by a resistors divider from the output node to the feedback pin FB. Improved efficiency occurs when moderate resistors are chosen for Rfbb and Rfbt. The equation for Vo2 is given by (13) below.

$$Vout = 0.765\ x\ \left(1 + \frac{Rfbt}{Rfbb}\right) \quad (13)$$

Letting Rfbb = 22.1kohm, and designing Vo2 = 5V, the resistance Rfbt is found by rearranging eq (13)

$$Rfbt = \left(\frac{5V}{0.765} - 1\right) x\ 22.1 = 121\ kohm$$

Vo2 Recommended Component Values- According to the datasheet, a feed forward Capacitor, Cff is to be added in parallel to Rfbt for additional phase boost for output voltages that are to be designed at or above 1.8V. This will be needed since the desired output, Vo2 = 5V > 1.8V.

Vo2 Inductor Selection L2 – The inductor peak-topeak ripple current, peak current and RMS current are calculated using equations (14), (15), and (16). The switching

frequency, fsw is given to be 650kHz by the TPS54525 datasheet. **[2]** For an output of 5V, L2 is given to be 3.3µH by the datasheet.

$$ILpp = \left(\frac{Vout}{Vin}\right)\left(\frac{Vin - Vout}{L2x\ fsw}\right) \qquad (14)$$

$$ILpp = \left(\frac{5}{12}\right)\left(\frac{12V - 5V}{3.3E - 6x\ 650E3}\right) = 1.36A$$

$$Il(peak) = Iout + \frac{ILpp}{2} \qquad (15)$$

$$Il(peak) = 3.41A\ + \frac{1.36A}{2} \quad = 4.09\ A$$

$$ILout(RMS) = \sqrt{Io^2 + \frac{1}{12}(\ ILpp)^2} \qquad (16)$$

$$ILout(RMS) = \sqrt{(3.41)\text{^}(2) + \frac{1}{12}(\ 1.36)^2} \qquad = 3.43\ A$$

The inductor selected for this design is a 3.3 µH, 9A, 7.8 mOhm inductor by Bourns.

Vo2 Output Capacitor Selection- The output ripple voltage is determined by the ESR and capacitance value. The TPS54525 is intended for use with ceramic or other low ESR capacitors. The recommended values range from [22-68] µF. The output capacitor RMS current rating is determined by equation (17) below.

$$Ic(out) = \frac{Vout\ x\ (Vin - Vout)}{\sqrt{12}\ x\ Vin\ x\ L2\ x\ fsw} \qquad (17)$$

$$Ic(out) = \frac{5V\ x\ (12 - 5)}{\sqrt{12}\ x\ 12\ x\ 3.3E - 6\ x\ 650E3} = 0.393\ A.$$

Cout was selected to be 47µF rated at 1A RMS with a low ESR rating of 1.71mOhm.

Vo2 Input Capacitor Selection – The input capacitor is recommended to be over 10µF. The capacitor voltage rating needs to be greater than the maximum input voltage, Vin=12V. Cin was selected to be 22µF with a 16V rating supplied by TDK.

Vo2 Bootstrap Capacitor Selection – The datasheet recommends a 0.1µF ceramic capacitor connected between the BOOST and SW pin for proper operation.

Vo2 Feed Forward Capacitor Cff – The feed forward capacitor, Cff is recommended to be in the [5-22]pF range. Cff was selected to be 5.6PF.

Vo2 Creg – The voltage regulator pin should have a 1.0µF capacitor connected between Vreg and ground.

RpGood – The TPS54525PWPR has a power-good drain output. The power good function is enabled after soft start has finished. Power good becomes active after 1.7 times the soft start time. Rpgood should be in the range of [25 to 150] kohm. Rpgood was selected as 100kohm.

Ren – The datasheet recommends Ren be connected from the EN Pin to ground in the [220 – 880] kohm range, with 440kohm being the typical value. Ren was selected to be 100kohm, which is out of the recommended range suggested in the datasheet.

Vo1 & Vo2 PCB Design – The PCB design for the two supply voltages was designed using eagle. The chips were laid out with polygons around Vin, Vout, and power ground, as close as possible to the recommended layout designs shown in Fig 4.2.8-G and Fig 4.2.8-H.

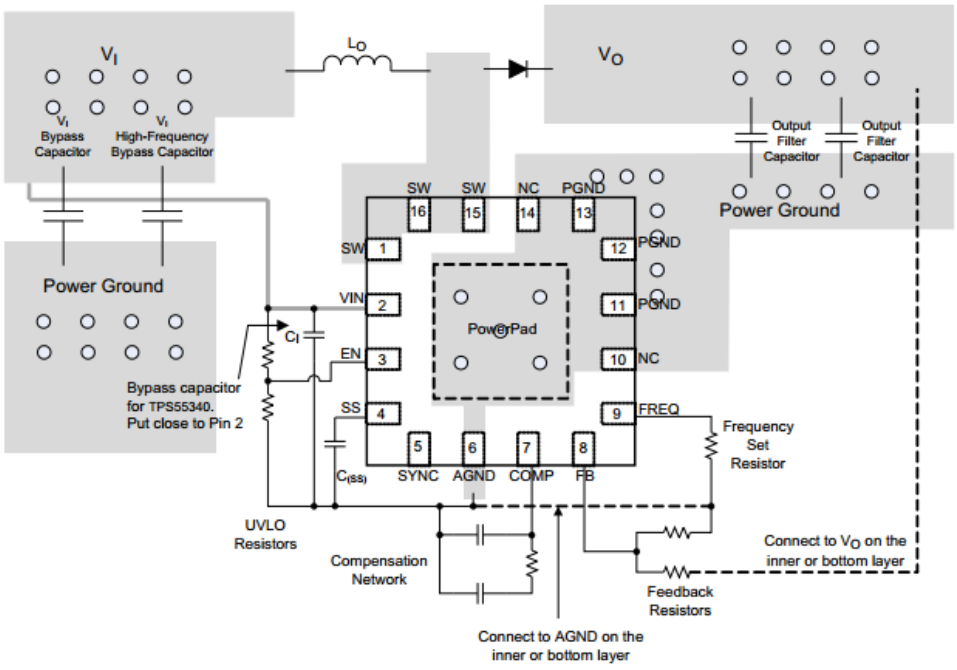


Figure 4.2.8- G: TPS55340RTER Suggested Layout Considerations for Vo1 = 12V

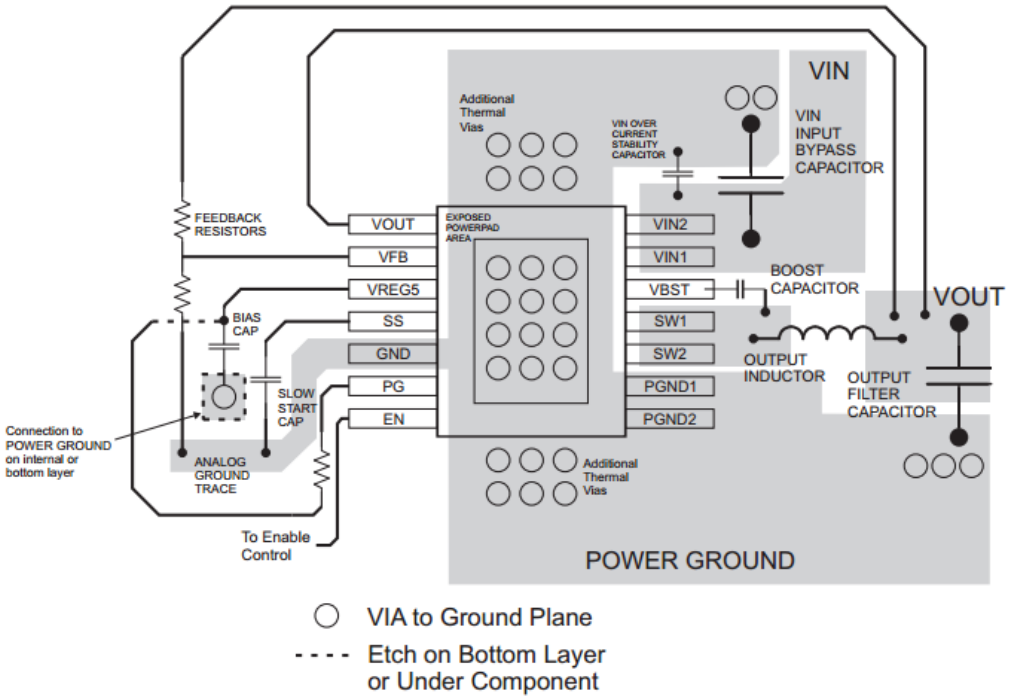


Figure 4.2.8-H TPS54525PWPR Suggested Layout Considerations for Vo2 = 5V

The combined PCB board for Vo1=12V and Vo2 = 5V using the TPS55340RTER and TPS54525PWPR is shown in Fig. 4.2.8-I.

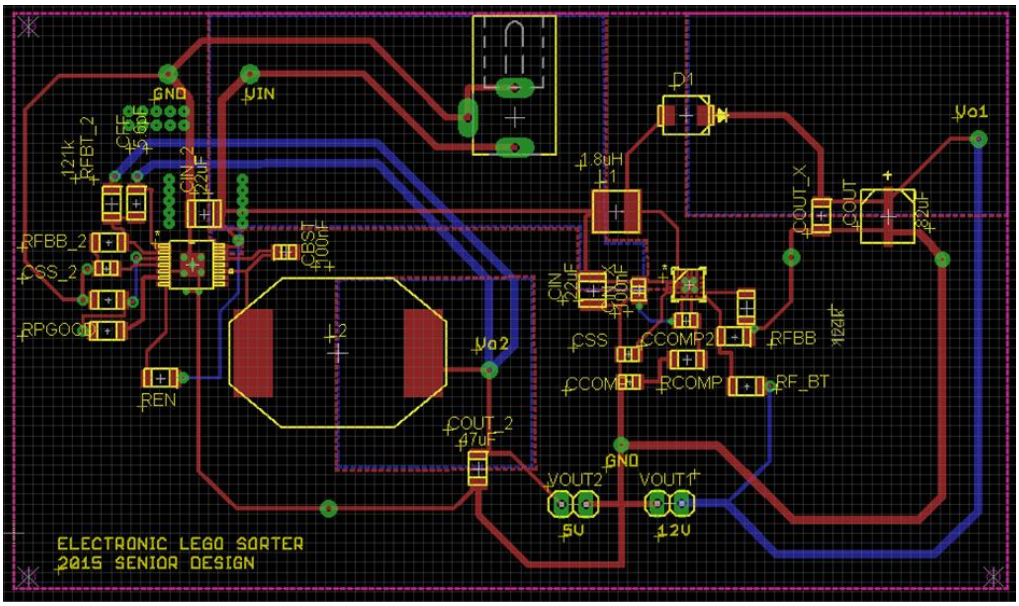


Figure 4.2.8-I: PCB Board for Power Supply

Fig 4.2.8-J Shows the polygon tied around Vin, Fig 4.2.8-K shows the polygon tied around Vo1, and Figure 4.2.8-L shows the polygon tied around Vo2.

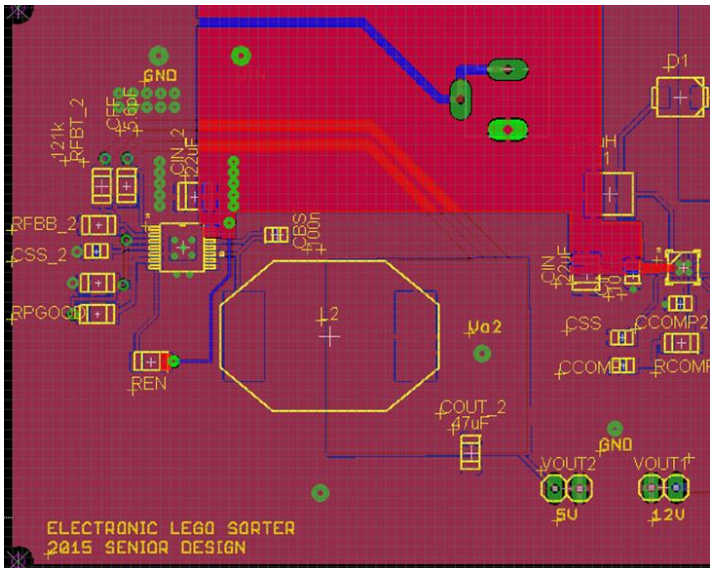


Figure 4.2.8-J: Vin polygon

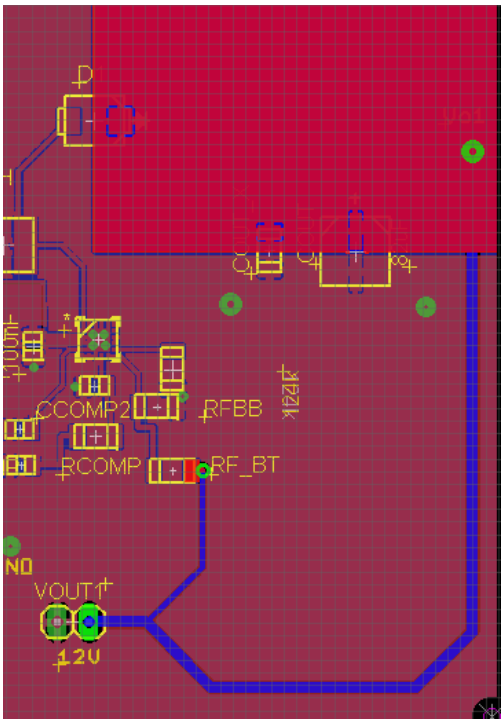


Figure 4.2.8-K Vo1 = 12V Polygon

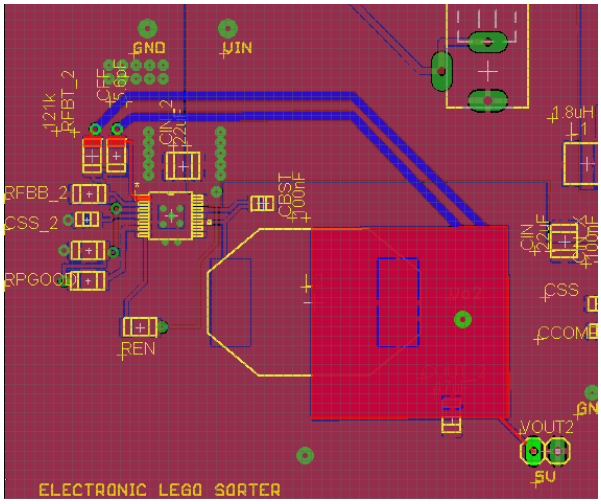


Figure 4.2.8-L Vo2 = 5V Polygon

Conclusion- The 12V supply ended up working fine as shown in figure 4.2.8 –M, but the 5V supply never worked.

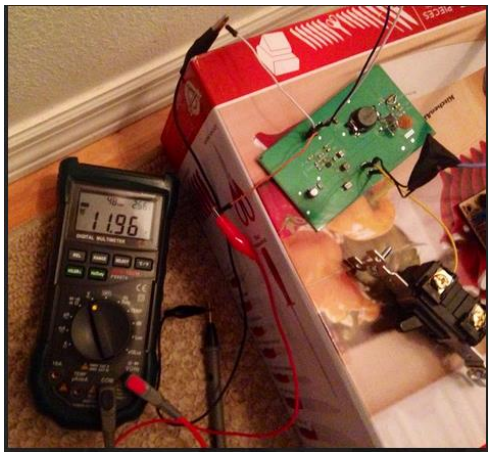


Figure 4.2.8- M Vo1=11.96V Measurement from PCB

There are a few things that could have contributed to this. The reference design provided by T.I. had the enable resistor, Ren tied to a third layer. Since price goes up significantly if you add more than 2 layers, it seemed unnecessary to add a third layer for a single route. Another issue is that Ren was out of the suggested range provided by the datasheet of [220 – 880] kohm with 440kohm, being typical. Ren was generated by Webench to be 100kohm. Unfortunately this was overlooked until after the PCB was ordered. Another thing that could have contributed to the Vo2 =5V supply not working is some of the thermal vias were removed. The original design had 16 more ground thermal vias above Ren. Before I sent out my PCB I used 4PCB.com DFM Gerber file checker. The file checker threw close drill hits for a lot of my thermal vias, and I unfortunately took them out so there would not be a cam hold.I previously had 12 vias on the TPS54525 as shown

in Fig 4.2.8-H. The vias were too small, and not compatible with 4PCB's board house. I took them out and replaced them with 5 larger vias to tie the exposed power pad to ground as shown in Fig. 4.2.8-L. In conclusion, I learned that the board house specification limits should be learned before laying out any part of a PCB design. I also learned that laying out a power supply PCB takes a lot of patience due to the polygons that must be tied to the chips for thermal protection.

Power Component Budget -

Power Supply BOM					
Index	**Quantity**	**Part #**	**Description**	**Reference**	**Extended Price**
1	3	399-1167-1-ND	CAP CER 0.1UF 16V 10% X7R 0805	CBST / VS5	0.33
2	3	478-1465-1-ND	CAP CER 5.6PF 50V NP0 1206	CFF / AVX / VS5	0.93
3	2	490-1882-1-ND	CAP CER 22UF 16V 10% X5R 1210	CIN / MURATA / VS5	2.58
4	2	490-1882-1-ND	CAP CER 47UF 10V 20% X6S 1206	COUT / TDK / VS5	2.16
5	3	478-1567-1-ND	CAP CER 1UF 25V 10% X7R 1206	CREG / AVX / VS5	0.72
6	3	399-1158-1-ND	CAP CER 10000PF 50V 10% X7R 0805	CSS / KEMET / VS5	0.3
7	2	SDR2207-3R3MLCT-ND	INDUCTOR POWER 3.3UH 9.0A SMD	L1 / BOURNS / VS5	2.24
8	3	SDR2207-3R3MLCT-ND	RES SMD 10K OHM 1% 1/4W 1206	REN / PANASONIC / VS5	0.3
9	3	P22.1KFCT-ND	RES SMD 22.1K OHM 1% 1/4W 1206	RFBB / PANASONIC / VS5	0.3
10	3	P121KFCT-ND	RES SMD 121K OHM 1% 1/4W 1206	RFBT / PANASONIC / VS5	0.3
11	3	311-1135-1-ND	CAP CER 6800PF 50V 10% X7R 0805	CCOMP /YAGEO AM. / VS12	0.3
12	3	311-1194-1-ND	CAP CER 820PF 50V 10% X7R 0805	CCOMP2 / YAGEO / VS12	0.3
13	2	490-3889-1-ND	CAP CER 22UF 25V 10% X5R 1210	CIN / MURATA / VS12	2.56
14	3	478-3755-1-ND	CAP CER 0.1UF 25V 10% X7R 0805	CINX / AVX / VS12	1.44
15	2	P16482CT-ND	CAP POLYMER 82UF 20% 16V SMD	COUT / PANASONIC / VS12	0.72
16	3	478-1565-1-ND	CAP CER 0.68UF 25V 10% X7R 1206	COUTX / AVX / VS12	1.08
17	2	445-2674-1-ND	CAP CER 0.022UF 25V 5% C0G 0805	CSS / TDK / VS12	0.72
18	2	B220-FDICT-ND	DIODE SCHOTTKY 20V 2A SMB	D1 / DIODES INC / VS12	1.08
19	2	SDR0403-1R8MLCT-ND	FIXED IND 1.8UH 2.91A 42 MOHM	L1 /BOURNS / VS12	0.94
20	3	P3.65KFCT-ND	RES SMD 3.65K OHM 1% 1/4W 1206	RCOMP/ PANASONIC / VS12	0.3
21	2	P10.0KFCT-ND	RES SMD 10K OHM 1% 1/4W 1206	RFBB / PANASONIC / VS12	0.2
22	3	P86.6KFCT-ND	RES SMD 86.6K OHM 1% 1/4W 1206	RFBT / PANASONIC / VS12	0.3
23	3	P124KFCT-ND	RES SMD 124K OHM 1% 1/4W 1206	RT / PANASONIC / VS12	0.3
24	2	296-37677-1-ND	IC REG BST FLYBK SEPIC 16WQFN	U1 / TPS55340RTER/ VS12	9.6
				Total	32.51

Table 4.2.8 – A: Power Supply Component Budget

4.3 Software Design of the System

While the hardware will provide the parts that are needed to perform the task at hand, the instructions for how to do that are all provided by the software. The software will be what tells the hardware what to do, when to do it, and how to do it. Most importantly though, software is what will be used to identify the Legos in order to separate them.

4.3.1 User Interface

The only means the user has to communicate with the LEGO® sorter is though the LCD touch screen connected to the main controller. Therefore simplicity is kept in mind when designing the interface so that the user will not be confused one how to navigate through the choices. The user interface of the touch screen is designed using an open source RA8875 library. There will be at most 4 different screens to navigate through are the following: the start-up screen, the sorting type and bucket assignment screen, confirmation screen, and pause screen.

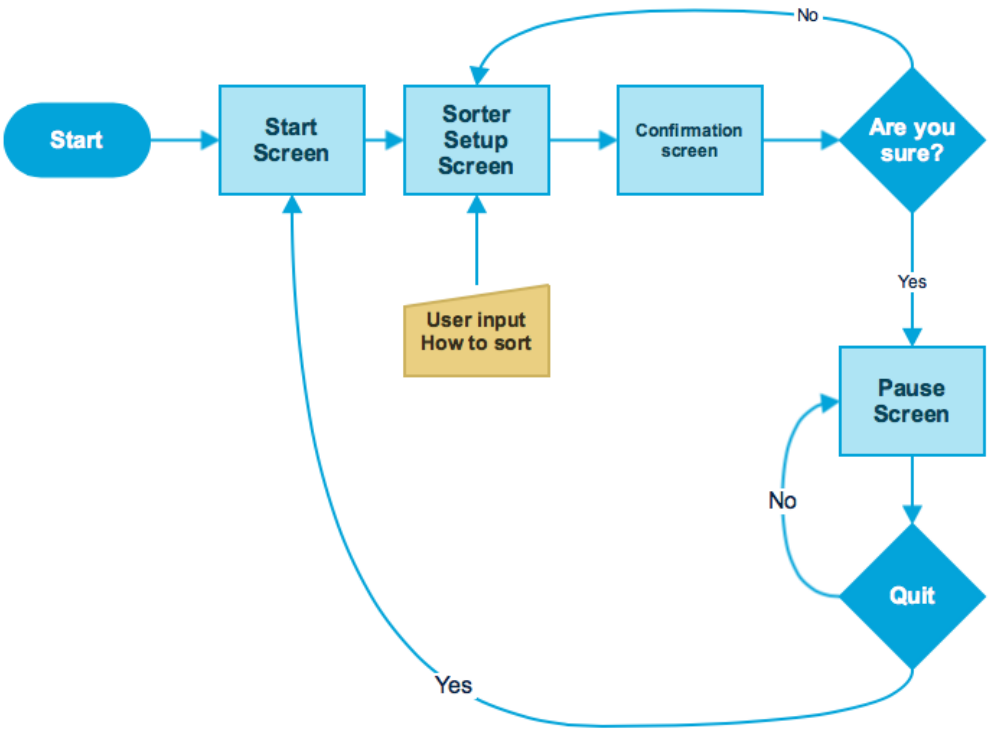


Figure 4.3.1 - A Basic GUI Screen flowchart

Start Screen- This will be the simplest screen to display it will first display the software name and version of the software. Pressing the start button will take the user to the next screen.

Figure 4.3.1 - B Example Start screen

Sorting Type and Bucket Assignment - Here the user will choose how to sort the LEGO® parts by either color or shape. If they want to do color and shape the user will just have to run the sorted bricks through the sorter again choosing the setting they did not use on the first run though. Depending on if they chose to sort by color or by part shape will change the bucket assigning choices. Because the interface should be simple a circle with a number in the middle is displayed which will correspond to the buckets at the end of the sorter and labels for the user to see what the bucket will hold during the sorting process. When the user presses the circle, it will highlight and a new window will open beside it.(as shown in figure…). Here the user can press the larger back arrow to exit the window or go to the previous window, double tap the selection to assign that value to the bucket or proceed further into the selection, or use the smaller arrows at the bottom of the window to scroll through the choices. The information in the windows will come from the library in the AtMega as a structure. When chosen, the label below the bucket will update displaying the value that is assigned to that bucket. The user must have at least one bucket be assigned to be the error bucket or "Misc." If that is not the case when the user hits the next button a pop-up appears notifying the user that they cannot proceed until one bucket is assigned to be the error bucket. The user is allowed to assign more than one bucket to hold the same items. For example the user is allowed to assign two buckets to hold red pieces since red is one of the most common colors it is expected to fill the bucket quickly. Once set the user must press the next button to move to the next screen.

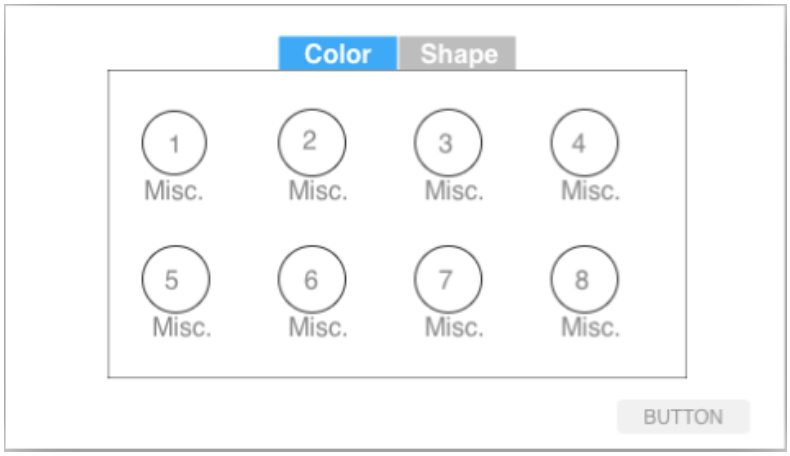


Figure 4.3.1 - C Example Setup screen

Confirmation -

Here the user will see a review page of how everything is going to be sorted. It will display a list of where they assigned the bricks. If they are okay with their selection then sorting will begin once they select the confirm button else the user will hit the back button and reconfigure their sorting set up. While this serves as a checkpoint in case the user made some sort of mistake in set up.

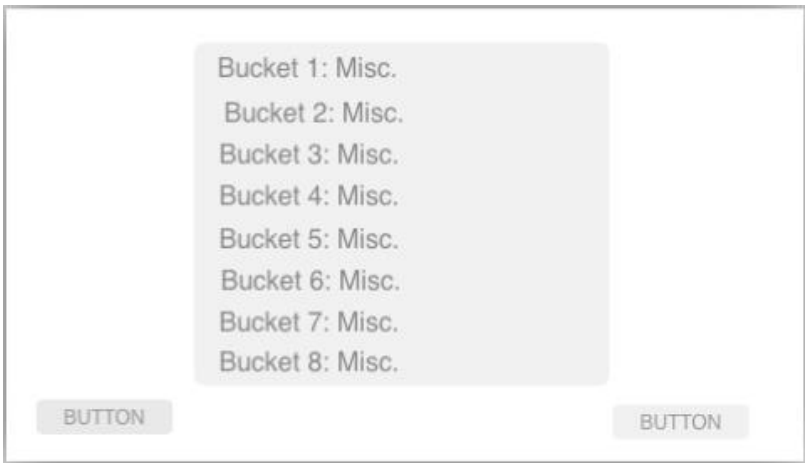


Figure 4.3.1 - D Example Confirmation screen

Pause - By the time the user hits this screen the sorter has begun. Here the user can pause or resume the sorting process. When paused, the user can decide to quit the current sorting process or resume at another time. If the user selects quit then the user will be sent back to the start screen and the sorter will stop completely.

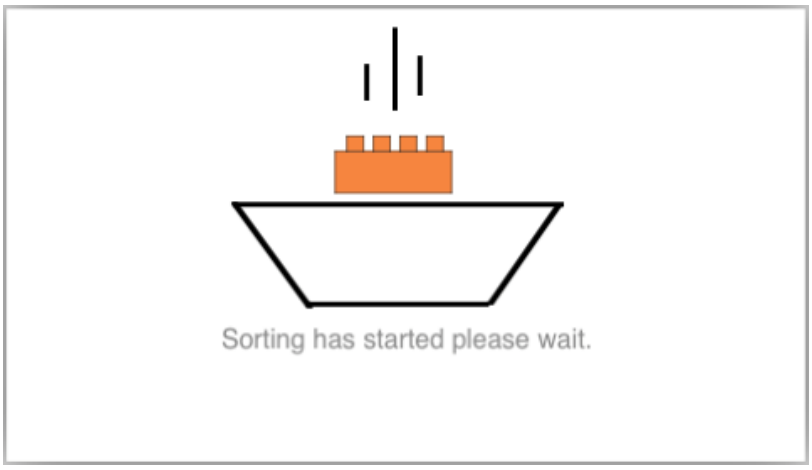


Figure 4.3.1 - E Example Status screen

By the time the user hits this screen the sorter has begun. Here the user can pause or resume the sorting process. When paused, the user can decide to quit the current sorting process or resume at another time. If the user selects quit then the user will be sent back to the start screen and the sorter will stop completely.

4.3.2 Image Processing

This section of software programming is debatably the most important in the entire project. This is the part of the system that actually determines what the Legos in the system are, and determines the receptacle that they belong in.

The first portion of the design for this process is that of determining when an image on the camera is ready to be pulled for use in sorting. This was done by analyzing frames in 1 second increments. In software, an image is simply represented by a two dimensional array of integers that all correspond to a different color. Ideally, an image of the background used in the image processing chamber will be represented by an array of all of the same number. A simple pixel analysis algorithm will look for changes in this number as the webcam in the system continues to process the images that it sees. If an unusual spike occurs within the array, this almost certainly means that a Lego has entered the area and the algorithm is able to process that. The algorithm will have a specific threshold however that determines when a spike is significant enough to deserve attention. This will eliminate errors that could be caused by smaller jumps in the array. For example, two cells in the array may only differ in value by one. This is almost certainly due simply to changes in the lighting, and thus can be ignored by the edge detection algorithm.

After the Lego has been detected initially, it performs a full analysis immediately after. This is to account for the possibility that a Lego was captured in frame while falling into the chamber. So the analysis is then done when the Lego is at rest in the chamber. Was the Lego is at rest, the algorithm initially detects color. If it is a color recognized by the program, it will send the proper signal to the MCU. If it does not recognize the color, or if the user is sorting by shape, then the process moves on to the shape recognition algorithm.

In the case of determining the color of the Lego, the process is incredibly simple. By searching through the two dimensional array of the image, it will determine what color the Lego is simply based off of the number that represents that color. Through testing, ranges were determined that represent the multiple different colors that the project will be handling. It is important that these colors are represented by a small range of numbers instead of just a singular number. This accounts for any possible discrepancies caused by the lighting. The most difficult color to determine will of course be white, seeing as this is going to be the same color as the background for the image processing chamber.

In the case of sorting by shape and size, processing the images will of course be slightly more involved. The two images that will be gathered from the camera will be able to gather all of the dimensions of the more basic Legos. Using the images gathered and some very basic math applied to that information, it was incredibly easy to gather the dimensions of the brick. So the approach that will be used will be to hardcode a library of the parts that will be sorted by the algorithm directly into the Beaglebone. This was a simpler algorithm and will most likely take less time than a sort of image comparison algorithm, but that will translate into more time consumed by simple data entry. This will require inputting details for every single Lego, and in order to gather those details it was extensive testing with the image processing chamber.

The math that the algorithm uses is to determine the length and width of the Lego in question, and then the side view of the Lego gathered from the camera is able to provide the width. The algorithm first locates the first colored pixel in the image and then it begins looking for the left-most and right-most corners of the Lego. By gathering the location in the array of these points and using the Pythagorean theorem, the algorithm gathers the length and width of the brick. These measurements are then compared to the data entries to determine if the Lego recognizes such a brick. If it does, it places the brick in the correct bucket, and if it doesn't, the brick will be placed in the miscellaneous bucket by default.

After the analysis has been completed in full on the Lego, the Beaglebone will send the appropriate signal to the MCU in order to determine specifically which bucket the brick belongs in, move the rotating arm to the appropriate position, and then activate the sweeper arm to push the brick into its bucket.

Another important part of the image processing algorithm is that it is able to determine whether or not the system needs to shut down if no more Legos are passing through. A

predetermined value decides how long the system will need to wait until it is ok to shut the system down completely.

So in summary, the software design of the image processing portion of the Lego sorter will run in a series of steps that are repeated until no more Legos are being presented into the image processing chamber. The steps are as follows:

1. Determine that a Lego has entered the chamber.
2. Activate a small delay to give the Lego time to come to rest within the chamber.
3. Perform any color or shape analysis necessary by calculating length, width, and height of the Lego.
4. Compare the results to the data in the Beaglebone.
5. Output results to rotating arm unit to dispense Lego.
6. Determine exit of the Lego from the chamber.
7. Repeat process until Legos have stopped entering the chamber.

4.3.3 Conveyor System

In the case of the conveyor system, the software handles a couple of different details. First and foremost is the speed of the two conveyors. The first conveyor moves at a much slower pace than the second. This is to ensure that Legos get spilled onto the second conveyor belt only one at a time. This allows them to be spaced out as much as possible on the second conveyor belt. The second conveyor belt is then able to move the Legos one at a time into the image processing chamber. The space between the Legos ensures as few errors as possible in the image processing.

The other detail that is accounted for is simply whether or not the conveyors are running. While a Lego is being processed for sorting, this causes a delay in the system requiring the conveyor belts to stop for small periods of time. The software is able to communicate this with the conveyor system so that it can start and stop the belts freely.

Simply put, the software communicates to the conveyor belts a static speed value, and the on/off state of the conveyor belt are handled as a Boolean variable.

4.3.4 Lift Arm System

In the case of the lift arm system, one detail that the software handles is the intervals in which the lift arm activates. In the case of the activity variable, if the lift arm activates too often, it would end up overloading the conveyor belt with too many Legos causing errors in the system. A value is set so that there are intervals between lift arm activations.

The second variable in the lift arm system is its speed. If the lift arm moves too quickly, it could throw Legos past the conveyor belts and off of the system, but if it moves too slowly, the Legos may not move off of the lift arm.

The speed of the lift arm is a static variable that was determined in hardware testing.

4.3.5 Errors

With main moving parts that make this whole system function there are many things that can go wrong if not corrected. There should be methods that can be used to monitor and treat the problem as it appears, report the problem to the user or just maintenance. Here are some problems that could surface and solutions to those problems.

Conveyor belt jam - With many small pieces constantly spilling into the conveyor belt there is a chance that some like an axle can fall in the spaces or somewhere it shouldn't be like where the motors are. It could end wedge in with the motor and could stop the system or break a vital part or perhaps the belt itself could get caught in the motors seizing it up or some other outside force that managed to get wedged in the conveyor belt motors. To prevent that walls were built on the final belts to prevent the LEGOs from falling anywhere near the motor or wheels.

Jammed Chute - This is similar to the motors in the conveyor belt some outside force could get wedged into the motor. The motor is concealed in a box in order to prevent any small parts from falling into it incase of a small chance of overflow. Also the chute was made wide enough to catch any brick and reorient itself as it slides to the appropriate bucket.

Lift Jam - Similar to the problem with the motor jam it is possible for the smaller LEGO® parts to become wedge into the lift components or some other form of outside force. However because of how it is built it is now not a problem

More Than One Piece Entering the Processor - The system that we have designed cannot handle processing two objects at the same time. So there needs to be a protocol that handles what to do if more than one part has entered the image processor. The image processor should pick up on the number of objects on screen. (There should be two. One for the actual part and the other is the reflected image of the part) If there is more parts then the sorter should be able to check is they happen to be the same color or same part and sort normally else all pieces goes into the error bucket.

Memory - Because image processing is being used to determine what each LEGO® part is according to what is stored in the library it is important to take into account the amount

of memory needed to perform this process. This was prevented by having to put both copies of the list of parts in both the Beagle bone and the AtMega.

Camera - The camera must be stationary as it must be at the correct angle to view the incoming LEGO® parts and the mirror. The camera is stored tightly in a box made of legos affixed in place without worry of moving.

4.3.6 Code Integration

The entirety of the Lego sorter was planned around building smaller subsystems separately, and then integrating all of them seamlessly together in the final stages of preparation. The software for the sorter is no exception. Each subsystem was coded and tested on its own to ensure that each part works properly, and then all of the parts were ultimately combined in the final stages.

A major advantage of taking this particular approach to the coding of the Lego sorter is that it makes things much easier to debug in the final stages. If all of the parts were put together from the beginning and then immediately tested, it creates a scenario where the bug could be in any number of locations, whether that be the subsystem, the main portion of code, or simply in the communications between the systems. By testing the subsystems before integrating everything together, this eliminates the possibility of the subsystem itself being at fault, and makes it easier to locate the problem.

A small downside to designing the software in this fashion was the need to keep track of the variables across different sections of code before they are integrated with each other. When the subsystems begin to be integrated with one another, became imperative that all of the variables were accounted for so that the communications between each of the systems works properly. It was a minor detail that proved to be tedious, but if it wasn't properly handled it could have caused major frustration in building the project in the long run.

Once each of the individual subsystems were working on their own, the only concern was the communications between them. The method that was used to code up the software allowed major focus in specific areas of the software one part at a time, rather than dealing with an overwhelming amount content areas simultaneously. The plan promoted good organization in the software, and it also made it easier to plan out time to ensure the best outcome for the project.

5.0 Interfacing Beaglebone Black & Atmega32u4

BBB & Atmega32u4 UART Interface- The Beaglebone Black Rev C was interfaced with the Atmega32u4 using UART transmit and receive pins. When the Atmega32u4 is

transmitting data the BBB is receiving data. A logic level shifter was implemented to transmit and receive data between the two MCU's since the BBB operates at 3.3V and the Atmega32u4 operates at 5V.

Baud Rate- In order for proper communications between the Atmega and the Beaglebone Black, the baud rate needs to be set to the same value on both ends. In the case of this project, the baud rate is set to 9600.

LCD Interface- The Atmega32u4 was interfaced to the RA8875 LCD controller using 4 Wire SPI interface as shown in Fig.5.0-A

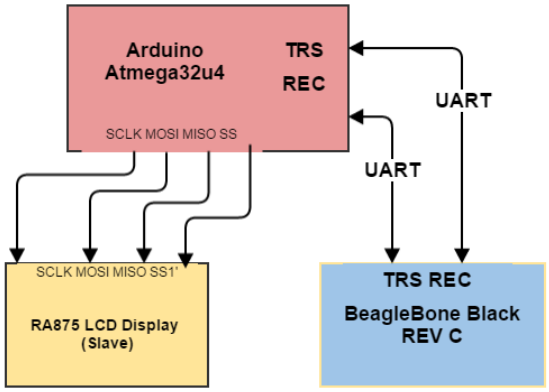


Figure 5.0-A: Interface Between the Beaglebone Black, Atmega32u4, and LCD RA8875 Controller

6.0 Prototype Construction

Section six will outline the process that will go into assembling the final prototype. This will cover both the hardware portion of the project as well as the software portion.

6.1 Materials List and Parts Acquisition

Our team decided to go without a sponsor. The parts were paid for by the team members.

Prototype Final Budget			
Part	**Quantity**	**Price**	**Total**
Beaglebone Black	1	59.99	$59.99
Atmega32u4 Chip	1	$7.00	$7.00
Power Supply PCB	4	$33	$132

Wooden Dowels	2	1.65	$3.30
L293D H- Bridge	1	$1.39	$1.39
RS-455 PA DC Motor	1	$9.20	9.2
TFT RA8875 LCD	1	$30.76	30.76
MicroSwitch	4	$1.50	$6.00
PCB	2	$33	$66
Stepper Motor 28BYJ-48 + UNL2003 Driver	1	$7.99	$7.99
Mirrors	2	$1	$2
Threaded Rod	1	$1.70	$1.70
Coupler for Threaded Rod	1	$0.90	$0.90
Wood	1	$15	$15
Photo Paper	2	$5 / ft^2	$10
Buckets	10	$0.50	$5
Legos	N/A	Donated	$0
Conveyor Belt DC Motor	2	$12	$24
Webcam Logitech C110	1	$19.50	$19.50
QRE1113 Sensor	1	$2.95	$2.95
Power Supply BOM	1	$33	$33
Embedded PCB Components	1	$30	$30
SPST Switch	1	$.69	0.69
Balsa Wood Glue	1	$4.50	4.50
Parallax S148 Servo Motor	1	$20	20.00
Balsa Wood	4	$4	20.00
Mean Well Power Supply	1	$18	18.00
Wire & Connectors	1	$10	10.00
Zip Ties	1	$2.50	2.50
LED	1	$3	$3
	Total		$546.38

6.2 PCB Vendor

As suggested there are two vendors to be considered for ordering the PCB for the project, OSH park and 4PCB.

Osh Park – OSH park has a lot of resources for PCB design. The website offers a lot of resources about common file formatting issues and plenty of links on how to solve these issues including 13 files on general "FAQ" and 7 files on CAD Package specific help. The price is more than fair as can be seen in Figure 6.3-A. OSH Park supports Eagle Cad

"Eagle BRD" files is only compatible with version 6.6 up to 7.1. The only real issue is the processing time and fabrication time. Time will need to be managed efficiently to allow 1 business day process time, 12 day fabrication time, and shipping time. It will take a minimum of 2 weeks for the PCB to arrive.

OSHPark	Price	# of Copies	Order Process Time	Fabrication Time
Standard 2 Layer Board	$\$5/in^2$	3	1 Business Day	12 Days
Standard 4 Layer Board	$\$10/in^2$	3	Weekly	14 Days

Figure 6.3-A PCB vendor OSH Park

4PCB – 4PCB offers a free design tool "PCB Artist" that can be used to develop the design needed. 4PCB offers a free file review which is reviewed by engineers so if there are any errors in design they will be found immediately with no surprises later. Figure 6.3-B shows that the turnaround time is very quick. It should take no longer than 1.5 weeks for the PCB to arrive.

4PCB	Price	# of Copies	Order Process Time	Fabrication Time
Standard 2 Layer Board	$33	4	Same Day	3 Days
Standard 4 Layer Board	$66	4	1 Business Day	5 Days

Figure 6.3-B PCB Vendor 4PCB

Summary- We decided to use 4PCB since it has a much faster turnaround time. We knew that the PCB's would take some time to design since we have no prior experience with eagle or soldering.

6.3 PCB Software

The PCB software used for both the power supply as well as the embedded Atmega32u4 PCB was Eagle. We chose to use Eagle because there is a lot more information on the web and plenty of youtube tutorials to get us started. Sparkfun, element14, and Webench all have eagle libraries, schematics, and board files that can be downloaded for reference. PCB Artist was experimented with to some extent. It is much more user friendly, but it was not used for this project simply because there is not as much information or tutorial videos as with PCB Artist.

6.4 Final Coding Plan

This section will outline the plan that was used on coding the system. The major parts to this include the image processing portion, the user interface, the moving portions of the system, and error handling. In order to do proper testing for the final prototype, all of these portions needed to be able to work individually, then they had to be able to accurately communicate between each other.

6.4.1 User Interface

Since the AtMega used to control the LCD screen R8875 library was had to be used. All the code and function provided are already written in C and is designed to easy to implement.

Initialization – First the object R8875 is called to a variable in order to used functions for the LCD. Four pins for the touch screen are assigned to the SPI pins (CS, CLK, MOSI, MISO) in the ATMega chip and 2 pins used to control the touch panel on the screen. They can technically be assigned to any pin but in this case it was pin 23 and pin 24 to INT and RESET on the LCD screen. The both the display and touch panel are activated separately in the set up function. Now the program is ready to draw on the screen and detect touch from the resistive touch panel.

Start screen – The start screen will just contain text displaying the name of our system, version number, and a push button. An example of the screen is shown in Figure 4.X.X – a . In order to implement this screen first the menu color is set for the background in this case white was used, then the Name and version of software are displayed by first changing from GRAPHICS mode to TEXT mode. Note that when first initialized, it is automatically set to GRAPHICS mode. Net set the cursor is set to the appropriate coordinates and text set to the appropriate color. Then call print through the touch screen library.

Since there is no function that handles events like in Java a function must be made to let the user move on to the next page. Because buttons are going to be used throughout this program a void function called roundButton is created to draw out the button states for when it is touched or untouched. The function takes in the x and y coordinates for where the button will be drawn from the top left corner, a string which is what is printed on the button, and a Boolean indicating if the button is being pressed or not. It will draw a white box with a black outline using the function call drawRect with black text centered in the button. If the button has been touched then the button is instead colored black using fillRect with white text. The size of the button is program to change if the string is too long.

For the touch it is done using a combination of the function touchDectect and touchReadX, where X is either Pixel or Analog. In this case touchReadPixel was used out of preference. touchDetect is used to sense any change in the analog values of the touch panel and returns true if it senses something. It also has a secondary use of clearing touch data when true is entered as an argument. Without a true argument touchReadPixel must be called immediately after it. touchReadPixel takes in the address of two variables to store the coordinates of the touch as coordinates in pixel. Those stored values are then checked to see if they within the pixel values of the button using and if statement. If true then the button will call roundButton and redraw it as being touched to indicate to the user that they touched the button. Then the next page is drawn on screen.

Setup Screen – This screen is used to assign sorting type and where each lego will go. On screen there will be two buttons that will decide sorting type. Eight numbered circles that will represent the buckets and eight labels under them that lets the user know what that bucket has been assigned to hold. An example of this screen is shown in Figure 6.4.1 – B. First a call to fillScreen to erase the screen. Then a call to drawRect is made to make a container for all the buckets. Then two more calls to drawRect is made to draw the tabs for when the user will switch between sorting by color or sorting by shape which which is determined by the variable isColorMode where true is "sort by color" and false is "sort by shape". A white line is draw on the bottom border of one of the tabs depending on the state of isColorMode. Finally a call to roundButton is made to draw an untouched "next" button. The buckets, bucket numbers, and labels are all drawn at the same time using a for loop and adjusting the spacing as needed. The labels are values stored and in array that is initialized to the value "Misc."

When the tab for Shape or Color is touched a call to redraw the setup screen by redrawing the tabs to indicate which one is active if not already selected and reinitializes the labels to "Misc.". Each of the buckets when pressed becomes highlighted and the choice window opens letting the user assign the buckets. When the next button is touched a call to the confirmation screen is made if and only if at least one of the buckets is assigned otherwise a function is called for a pop up window to notify the user that there must a "Misc." assigned to a bucket.

Choice window – The creation of this window is the most complicated because of the amount of interactive parts that are close together in one area and the fact that the touch screen has no means of detaching pressure on the touch panel. In order to lessen the amount of false positives the user must double tap their choices.

First the area where the window will redraw in cleared by calling rectFill using whatever color for the background. Then the shapes for the buttons, title space, choice space using the draw functions. Once the basic shape of the window is created print the title of the window in the title space. The title used come form the menu structure.

For choices in this code it is set to hold four items at the time by adjusting the spaces and size of the window it can be changed. As for the items depending on the level of the menu (Level 0 being the lowest) it will either print the items from the list of items when

the menu is a level 0 menu or print the title of the lower levels from the list of menu structures. A for loop is used to print the items on the window. The starting point from where on the list to start looking at is through the variable idx as when the user is scrolling through the list the view of the list shift every four blocks.

For touch a while loop is used to keep polling for a touch. If one of the for choices are touched the choice is highlighted blue. This done by drawing a blue rectangle with fillRect and reprint the string in white. The term idx which is an integer based on the array values of the menu list is loaded into a temporary variable called tempsel which is then saved in the variable selected when the user lifts their finger to remember the highlighted value. While it is highlighted if the current menu is level 0 then the choice is stored in an array that will update the setup screen otherwise a recursive call to itself is made with the new argument for the menu struct being based of the title printed on the list of choices. If the size of the menu list is greater than four the bottom left button is pressed the button first is highlighted and a flag for the previous button is set to true. The next button, and the top left button are similar to the previous button only the next flag and prev_p flags are set to true respectively. When the finger is lifter for a certain amount of time determined by the counter the selected button is set. If the next flag is true the idx value is updated by incrementing it by four or resetting it to zero is the value goes past the list size. If the previous flag is true then the value is decremented by four or goes the last available pointer value for the menu list. Then the window is cleared and updated with the next set of values. If prev_p is true if there exists a higher-level menu based on the current menu then a recursive call is made to that menu otherwise it returns a null string.

Confirmation screen – This screen is the simplest of the screens as it is just a text box showing what the user choices from the setup screen and two buttons with one leading back to the set up screen and the other leading to the status screen. Basically a call to drawRect is used to draw a container for the text. A for loop is used to print the bucket assignment into the container and two calls to roundButton for a Next and previous button.

For touch it is checked if the touch value is within one of the two buttons. If the “prev” is detected to be touched then the screen changes to the Setup Screen. If the other is touched then the pause screen is drawn.

Pause screen – The pause screen gives the user to pause, resume, and quit the current job along with a fun little image to fill in white space that uses a hand full of the R8875’s drawing functions such as drawLine, drawHLine, drawVLine, and other drawing functions. Text is printed below the image stating the state of the sorter. The buttons that appear depends on if the system is paused or not. If the system is not paused then one call to round button is made to make a pause button. When the button is pressed the pause state it updated and the pause screen is redrawn to reflect the change in state. When redrawn the text prints the new state of the system and two calls to roundButton are made making a resume and quit button. When resume is touched the pause screen updates to

it’s previous state of the system running else when the quit button is touched the screen a pop up window opens asking the user if they want to quit.

Popup windows – When going through the screens there are only two pop up window the user will encounter. A pop up for a missing “Misc.” assigned bucket and the quit pop up. Both are made using similar methods. First and area for the window is cleared using fillRect. Then the window is drawn using drawRect. Next the text for the pop up is printed in the box. Finally one call to roundButton to draw and “Ok” button for the warning pop-up and two calls to roundButton for the “Yes” and “No” buttons for the quit box.

For touch when the “Ok” button is pressed in the warning pop-up or “No” is pressed in the quit pop-up the pop-up disappears. If “Yes” is pressed in the quit pop-up then the screen is brought back to the start screen.

6.4.2 Error Processing

Error processing is designed to handle any problems that occur in the hardware and software. With the amount of moving parts in the sorter, most of the errors will come from some sort of outside force. Sensors are used for vital parts of the sorter in or to catch any problem that appears. As for the software the vital areas are the image processor and the library as with out them there will be nothing to compare the Lego parts with. When these problems arise depending on the severity of the problem the system needs to either pause the sorting and state the error on the LCD screen or dynamical correct the problem as it occurs and continue running. The coding language used to run this operation will be C++ language as that is the language the microcontrollers use.

Motor Speed – Because of how the conveyor belt is built and how it function the speed of the motors are critical as the conveyor belt is what helps separate the Lego parts before they enter the image processor. Therefore the speed of the motors of the first and second conveyor belt must be run at specific speeds, the first belt running at a slower speed and the second conveyer belt running at a faster speed. The motor drivers will constantly report the speed of the motor as it is running. If for some reason the motors run at a faster or slower speed than it is allowed the drivers should correct themselves otherwise the Conveyor Belt Jam error process is performed.

Power Surge – This is a very important error to catch as many of the components running this machine are low power and a sudden change in power will damage the sorter or lead to a fire. To avoid that the power supply should shut itself off.

Camera – Before the system can even run the camera must be set in place. The camera is set when it sees certain marks in the area were the image processing takes place. If at any time the camera is moved out of place the system will pause and state message on the

LCD screen that the camera is off center. Once back in place the image processor will rerun if there is a part that needs processing and continue operation from there.

Weight – When the containers become full that cannot be used anymore because the parts will just over flow. With the load sensors under the containers the system can keep track on the amount of parts in the container. Once the load has reached a certain threshold the rotating arm will skip that container and instead rotate to the next assigned container otherwise to the error bucket. If the Error bucket is full the system will pause and alert the user that there are full containers.

6.4.3 Image Processing

The process of coding up the image processing portion of the system was one that had to be methodical and fairly meticulous. With it being the main brain behind the sorting of the Legos, it was absolutely imperative that the image processing code works EXACTLY how it's supposed to in every aspect. If it was even slightly incorrect, it could have resulted in more errors in the final project than successful sorts, and that was unacceptable for the project.

The important thing to understand when it comes to the image processing is the two dimensional array that represents the image. Figure 6.4.3-A shows a rough visual representation of what a two dimensional array looks like with an image from the image processing chamber.

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	[illegible]	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 6.4.3-A

As shown in the figure, 255 is the number that is representing the background of the conveyor belt, while in this hypothetical case, 1 is representing the color of the Lego currently in the chamber. This is the representation of all of the images that are processed throughout this entire program. To reach this result the code moves through a number of steps.

Step 1

The first portion of coding that was done was the code required to read in an image from the webcam. While it appears to be fairly trivial, a good deal of research was required to ultimately reach the point where the camera was able to gather and print a full image.

Step 2

The second step was to write the simple algorithm that detects when a Lego has entered the image processing chamber. This algorithm quickly goes through the color analysis algorithm to determine if a Lego has appeared in the chamber. The algorithm goes through the two dimensional array that represents the image given by the camera and if it finds a color significantly different than white (the background) it will know that a Lego has indeed entered the area. After a small delay, the code then begins its full analysis of the Lego that has entered the chamber.

The code for this portion is fairly simple and very active. It works in intervals, checking the chamber every second for significant changes in the chamber to determine if it's time to run a full analysis.

Step 3

The next part of coding will be to test color detection. This portion of coding won't be so much a contribution to the algorithm as it will be testing to determine thresholds for the different Lego colors. With all of the lighting in the image processing chamber, the chances of a single Lego showing the same color code for all of its pixels is EXTREMELY low. There is a small range of numbers that each color will represent and testing helped to determine these ranges. These are set in the code so that when the program goes to detect colors, it returns every color represented within the image. Ideally, there will be two colors returned each time: white (the background color) and the color of the Lego. In some cases it may only return white if the Lego is white as well. As shown in figure 6.5.2-B, this Lego is representing most of its pixels as a value of 2, with some discrepancies where the value is one. These are very likely caused just by the lighting in the image processing chamber, and this could very well be the exact same Lego that was shown in figure 6.4.3-B

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	1	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	1	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	1	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	1	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 6.4.3-B

The coding for this portion is fairly simple. Through testing, predetermined ranges of values were set for the different colors. So to determine what colors are in the image currently being processed, the program simply goes through the image and determines all of the different colors represented. A simple brute force algorithm using a double "for" loop is all that is needed to do the job.

The code creates an array that will contain the colors detected within the image so far, but will not contain duplicates. Ideally this array should only have data in two cells at the end of the process and then this is used to ultimately determine which bucket the Lego will go in when sorting by color.

Step 4
This was the most tedious and time consuming part of the code. It required extensive testing, a good deal of data entry, and a good deal of plain effort.

From what research and preliminary observation has been done, the easiest and most efficient way to perform the shape recognition was with some basic measurements. The part of the image that captures the top view of the Lego is able to gather the length and the width of the brick in question. The other part of the image (the side view) may will not have that advantage. But it still is able to gather the height of the Lego accurately. These two pieces of data are able to narrow down the Lego selection to the particular Lego in question. These details can be measured simply in array cells. This is where the

extensive testing and data entry was done. Each individual Lego shape was individually tested multiple times to determine the accurate measurements of the shape. After determining the small range of values for the Lego in question, the data was hard coded into the Beaglebone Black, allowing it to accurately determine the shape of the Lego. This also takes into account the different possible orientations of the Lego (IE if the Lego is on its side) and has those measurements entered into the library as well.

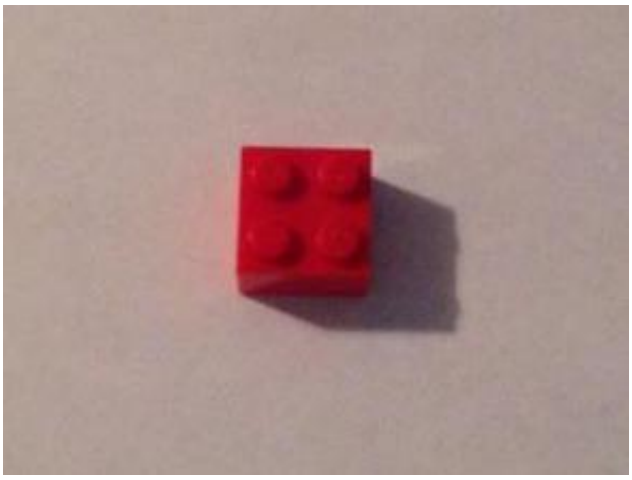


Figure 6.4.3-C

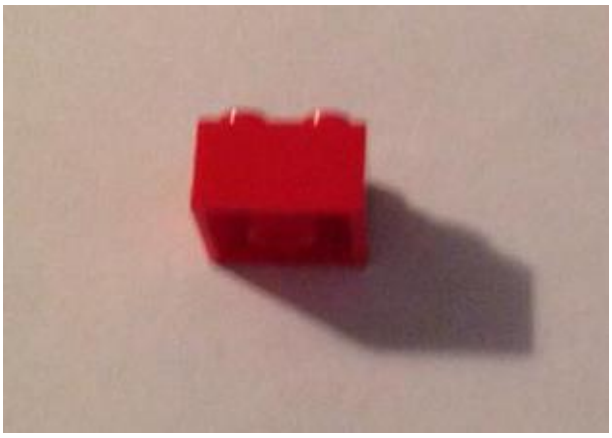


Figure 6.4.3-D

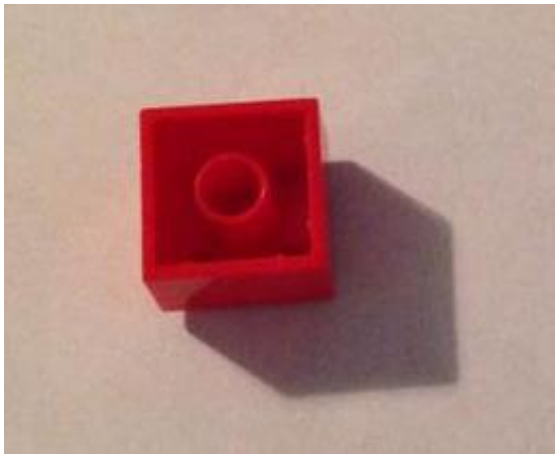


Figure 6.5.2-E

So the coding to read in the image is fairly extremely simple. It consists of two separate double for loops (one for each portion of the image). The first set is for the topside view. This view locates the position of the pixels at the highest, furthest left, and furthest right positions. By using the Pythagorean theorem the length and width of the brick is able to be determined. For the second portion of the image, the algorithm runs through each column searching for whether or not that COLUMN contains the color it is looking for. The code then simply keeps track of how many columns that were counted, and this is the value that is used for the height of the Lego.

These steps combined together ultimately make up the whole of the image processing algorithms.

6.4.4 Other Subsystems

The coding plan for the subsystems is fairly straightforward. There doesn't need to be excessive coding, as the other subsystems don't have a massive amount of variables to handle. The only exception to this was the rotating arm system, which had to deal with each of the individual buckets.

Lift Arm System – The lift arm system is treated as its own object, containing a few very basic variables. The variables account for the speed that the lift arm moves, the interval at which the lift arm moves, and then a variable to denote whether or not the lift arm is active. There are also variables to account for the switches above and below the lift arm, indicating when it is time to change direction.

Variables:

- int Lift_Speed – this variable accounts for the speed at which the lift moves up and down

- int Lift_Interval – this variable accounts for the time interval in between lift activations, to make sure there is proper space between loads of Legos
- boolean Lift_Status – this variable is simply to denote whether the lift is active or inactive, with the default being inactive
- boolean upperSwitch – this variable accounts for the switch that tells the lift arm to stop moving upward
- boolean lowerSwitch – this variable accounts for the switch that tells the lift arm to stop moving downward

Conveyor Belt System - The conveyor belt system is only slightly more complex to represent in the code than the lift arm system, only because there are two conveyor belts to account for in the code. The necessary variables for the conveyor have to simply account for whether or not each belt is active, and the speed at which each belt is moving.

Variables:

- int BeltOne_Speed – this variable accounts for the speed of the first belt in the conveyor system
- boolean BeltOne_Status – this variable determines the current status of the first belt in the conveyor system
- int BeltTwo_Speed – this variable accounts for the speed of the second conveyor belt, which will definitely be greater than that of BeltOne_Speed
- boolean BeltTwo_Status – this variable determines the current status of the second belt in the conveyor system

Rotating Arm System - Of the mechanical subsystems in the Lego sorter, this was the most complex with the software. It will required variables to account for each of the buckets, as well as the rotating arm itself. Also there needed to be variables for each of the sensors that the rotating arm is reading to know it has reached its destination. The buckets are simply stored in an array in the code. Another array contains the values of the Legos corresponding to each bucket. So the code uses this array to determine the position of the bucket that the Lego needs to go into. The rotating arm is also be its own object, containing variables for speed, whether it's active, and location.

Variables:

- Buckets[8] – the code contains an array of characters corresponding to Legos that are called in order to identify each of the Legos, and these Lego types can be assigned to each of the individual buckets, which determines which bucket the Lego will be sorted into

- int Arm_Location – this variable simply keeps track of the current location of the rotating arm as it continues to travel around the buckets to find its final destination
- int Arm_Speed – this variable accounts for the speed that the rotating arm will be moving
- boolean Arm_Active – this variable simply accounts for whether or not the arm is currently moving

Sweeper Arm System – A later addition to the project, the sweeper arm system is the means by which a Lego brick makes its way to the rotating arm. It's location directly under the conveyor belts, as well as its position flush against the base of the image processing chamber, make it the perfect addition to simplify Lego detection compared to the initial plans of a direct disposal from the conveyor to the rotating arm. The sweeper arm uses switches just like the lift system, and a few variables to determine its speed and direction.

Variables:

- int speed – this variable determines both the speed AND the direction of the arm; the motor's code dictates that a certain range of values move the arm forward and another range moves it backwards, which we have set to 180 and 5 respectively
- boolean forwardSwitch – this variable accounts for the switch that is placed out in front of the sweeper arm to tell it to begin moving backward
- boolean backSwitch – this variable accounts for the switch that is placed underneath the conveyor, towards the back of the sweeper arm to signal when the sweeper arm should stop moving

After these various subsystems were taken care of, the next step was to integrate all of these parts into the main portion of code with the image processing chamber. The integration was fairly simple at this point, given that each subsystem is properly working on its own. Each of the variables listed were assigned by the main portion of code and then changed based on the status of the image processing chamber.

Status Changes:

- Conveyor Belt System – The dual conveyor system moves continuously, stopping when a Lego has entered the image processing chamber. This gives the image processing component time to do its work, and it gives the sweeper arm and the rotating arm ample time to do their job as well.
- Lift Arm System – The lift arm is set in static intervals, moving up and down based off of the speed of the conveyors.

- Rotating Arm System – The rotating arm system is only active as soon as the analysis on a current image is complete and the result has determined the final location of the Lego being analyzed. So the status will change to active when it receives the signal to move, and inactive as soon as it reaches its location.
- Sweeper Arm – After the rotating arm has finished its job, the sweeper arm activates, pushing the current brick off the edge and down into the rotating arm.

7.0 Prototype Testing

This section outlines methods used to test our prototypes before integration.

7.1 Hardware Test Environment

Much of the initial testing of hardware was done in the Senior design lab. Towards the end of senior design, there was no longer any room to set up our project with dimensions of 5 x 3ft. This made integrating all of the moving parts of our project very difficult. We ended up moving our project to a team mate's home to finish the project. We used 12V and 5V AC adapters to preserve our power supply in case of a short. We used a 14 function multi-meter to test voltages, currents, continuity, capacitance, resistance, etc. of our circuitry. Once we got everything assembled and moving together, our hefty project was transferred back to the senior design lab to get ready for the demo.

7.2 Hardware Specific Testing

This section will outline the hardware testing of the three main mechanical subsytems: conveyor belts, rotating arm, lift system, and sweeper system.

7.2.1 Conveyor Testing

The testing for the conveyors was mainly done using the MSP430g2553 PWM pins and adjusting the speed of the belts until the Top belt moved as slow as possible while still having enough torque to turn the conveyor belt.

7.2.2 Rotating Arm Testing

The stepper motor requires a specific sequence of signals to operate as intended. Testing will begin once it is completely hooked up as designed. After hooking up the stepper motor and driver circuit to the Atmega32u4 microcontroller and power supply as shown in Figure 4.2.4-B, the code will be written. Testing will make sure the timing in the code allows for smooth operation of the stepper motor. Care will be taken to make sure delays are put in the code to accommodate for the slower electro-mechanical processes that occur in the motor. Being that the motor has 1.8° steps, it should not be a problem to get the rotating arm slide to line up with each LEGO bin. Testing will make sure it does anyway. The color sensor on the rotating arm will also be tested. It will be connected via I2C with the microcontroller before testing. The sensor has a register and some integrated features that will be read into to properly test its functionality.

7.2.3 Lift System Testing

The lift system was initially tested without a load. The MSP430g2553 was used initially to test the hardware. The Switches were hooked up to interrupt pins and the motor was hooked up to the L293D and the MSP430G2553 as shown in figure 7.2.3-A

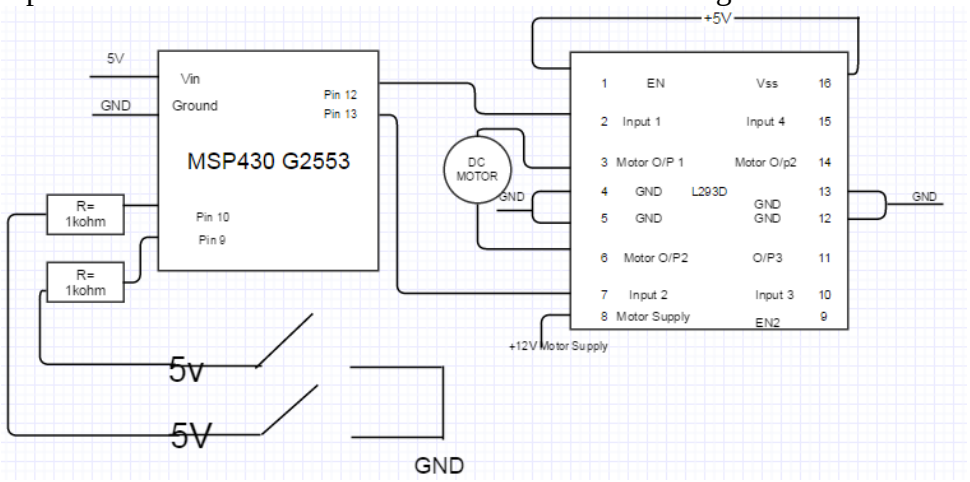


Figure 7.2.3- A: Lift System Testing Before Integration

For the final integration the switches were hooked up to digital pins. One switch was set High and the other was set low. The Atmega32u4 used polling through serial interface to constantly poll the switches to see if they were pressed. When the switch is pressed, the motor reverses direction until it hits the other boundary switch, and the motor reverses direction repeating the process.

7.2.4 Sweeper Arm Testing

Setting the Internal Potentiometer of the Servo -In order to test the sweeper arm, the Parallax S148 Servo motor's internal potentiometer must be tuned the neutral position. The datasheet specifies that the neutral position of the Parallax S148 is at 1.5ms with a period of T=20ms. The servo was connected to 5V using a triple power supply in the senior design lab, and connected to ground. The function generator was set to a pulse width of 1.5ms and a period of 20ms. The output of the function generator was connected to the oscilloscope and grounded. Once the square wave was measured on the scope to the appropriate neutral pulse width of 1.5ms and a period T=20ms, the function generator was connected to the control line of the servo. In order to adjust the set point of the servo's internal potentiometer, the screw connected to the outside casing of the motor was slowly turned until the servo stopped rotating.

Configuration – The Sweeper arm switches were tested in a similar manner to the lift system. The Parallax148 only has 3 wires for control: +Vcc = 5V, GND, and a control line. The control line was wired to an analog pin on the Atmega32u4. The servo was initially programmed using the Arduino "servo sweep" library. The only thing the servo needed to do was go CW until it hit one boundary switch and then reverse direction and go CCW until it hits the other switch. The maximum speed was used with 180 degrees CW, and 5 degrees CCW.

7.3 Software Test Environment

An extremely important portion of testing is creating an accurate environment in which to test the software. In order to make sure that the testing done accurately reflects how the project will respond in the final prototype, it is of utmost importance that the testing be done in the same kind of environment. Therefore, since all of the software will be coming from the Atmega and the Beaglebone Black, those will be the testing environments for all of the software. The software will be written onto the microcontrollers via a Windows 8 computer with standard specifications, and this computer will also be used to help debug the code during active testing.

7.4 Software Specific Testing

This section covers the user interface testing, image processing testing, error handling tests, hardware communication testing.

7.4.1 User Interface Testing

It's important the user interface functions behaves properly without many issues as it the only way the user can communicate with the LEGO® sorter. That why it important to test the responsiveness of the LCD screen and GUI such that the screens flow in the correct order, the bucket screen is assigned to the correct bucket, test that the interface communicated to the controller the sorting method, reading the data from the sensors, and calibrating the screen in order to interact with the GUI properly. Before any of these tests can be done first make sure the LCD screen is properly calibrated for use. The RA8875 library comes with code used to calibrate the touch screen panel.

Screen flow – Testing the screen progression is simple enough. It's mainly to see if all the screens are displaying in the right resolution and if the buttons go to the correct pages. During this stage there is no other functionality like reading sensor data or sending the bucket assignments.

Supplies:

- 5" RA8875 Resistive touch LCD Screen
- ATMega
- Computer

Preparation:

Attach the LCD screen to the AtMega. Upload the user interface program from the computer to the AtMega to begin Debugging.

Procedure:

1. Run the first screen on the display. The entire screen must fit on the screen and must not leave any space or get cut off screen. If it is not fitted properly then go back into the code and readjust the GUI window size.
2. Once the first screen is displaying properly press the start button to go to the next page. The page showing should be the set up screen where the user picks how to sort and what each bucket is going to hold. If it is not got back to the code and correct the where the button should be pointing to.
3. Then check to see if the page is the right resolution.
4. Set the sorting type and assign any value to the buckets.
5. Once everything is set up properly press the next button. The next screen should be the confirmation screen. There will be an un-editable text box that displays how the blocks will be sorted and what each bucket is going to hold.
6. Again check if the resolution of the window and adjust accordingly in the code if needed.
7. Next press the back button. This should lead back to set up screen.
8. Make some changes in the setup screen and press next.

9. The confirmation screen should change according to the new settings. Press the next button. The next screen should be the pause screen and the final screen. Here the screen shows and image, some text, and a pause button.
10. Check the resolution and adjust if needed.
11. Press the pause button. The text should change and the resume or quit buttons should appear.
12. Press resume. The screen should go back to its previous state.
13. Press pause again.
14. Press the quit button. A new window should pop-up asking for a confirmation.
15. Press no and the window should disappear returning to the status screen again.
16. Return to the confirmation window and press yes. The screen should return to the start window concluding the test.

With all the screens progressing in the right order it is ready to be integrated with system.

Setting Up – Here is where is where we assign what will go into witch bucket. Here we just need to test if the information is loaded into the array properly.

Procedure:

1. Go the set up screen
2. Select the setting for the seven buckets and compare it to the information in the debugger.
3. When everything is set correctly go to the confirmation screen. All the information in the array should be displayed on the confirmation screen's text box. If it's not then that means the Library is not organized properly and some data does not have all it's information.

Once it is know that all the data selected is going into the array next is to test communication of the LCD screen to the image processor.

Procedure:

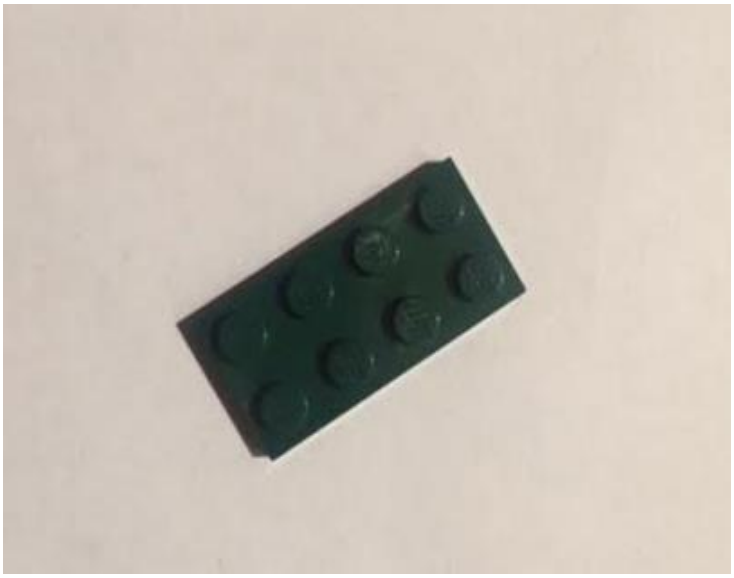
1. First set all the data for all the buckets on the set up screen.
2. Then hit next to go to the confirmation screen. Make sure to check the data in the array with a debugger.
3. Press the next button on the confirmation screen. This will send a byte to the image processor indicating how the LEGOs will be sorted.
4. Check the image processor with a debugger to see if the byte has been received. If that is not the case then there is something wrong in communication. Check the code to see if any port name has been misspelled or if the wiring of the microcontrollers is wrong.

7.4.2 Image Processing Testing

One of the most important parts of testing is that of the image processing component. This consists of a number of different tests that determine how the system handles different scenarios that may occur within the system, as well as testing the very basic situations that should be occurring regularly.

The simplest version of this test is by simply running through Legos one at a time in the normal fashion to ensure that they are sorted into the correct buckets. These tests were run for every available color that is going to be used in the project. The supplies and preparation for these tests were all the same.

Color Test Number 1: Basic Situation Test



Supplies:

- Image Processing Chamber
- BeagleBone Black
- Computer

Preparation:

Connect and power up the BeagleBone via the computer, and then connect the BeagleBoard to the Image Processing Chamber.

Procedure:

1. Place singular Lego within the parameters of the Image Processing Chamber.

2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.
3. Check result of algorithm to confirm that it matches the color of the Lego.
4. Repeat 1-3 for other singular Legos to test all of the different colors and a wide variety of shapes.

There are multiple other situations, however, that may arise that should be accounted for as well when it comes to color testing. One such situation involves having multiple Legos on frame. In this case, the test was run to see if all of the Legos on screen coincidentally are the same color. In this case, the multiple Legos can be dispensed into the correct bucket as normal, however if the two are different colors then they should be put into the error bucket instead.

Color Test Number 2: Multiple Legos

Procedure:

1. Place multiple Legos within the Image Processing Chamber either all of the same color, or with different colors.
2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.
3. Check the result of the algorithm. In the case where the Legos are all the same color, confirm that this is the result that was gathered by the algorithm. In the case where the Legos used were of various colors, confirm that the algorithm resulted in an error.
4. Repeat steps 1-3 for various different combinations of Legos. This includes overlapping Legos, Legos in different positions, and Legos that represent each of the colors used in the algorithm.

Another situation that could occur would be multi-colored Legos. If there are multiple Legos in the chamber at the same time, and their colors are not the same, this will be processed as an error and the Legos will be placed in the miscellaneous bucket.

Color Test Number 3: Multi-Colored Legos

Procedure:

1. Place Lego(s) with multiple colors in the Image Processing Chamber.
2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.
3. Check the result of the algorithm to ensure that it is an error every time.
4. Repeat steps 1-3 for a wide variety of Legos to ensure a consistent result.

These three tests will account for virtually all of the situations that the system may encounter. With multiple tests in a wide variety of areas involving color it will help to ensure that the sorter is able to successfully separate the Legos properly based off of their color. Testing the system for its ability to recognize shape was a different set of tests however.

In the case of shape, there is a set library of shapes that the system will be able to recognize. In the case of a shape that is not recognized, it will be sent to the error bucket.

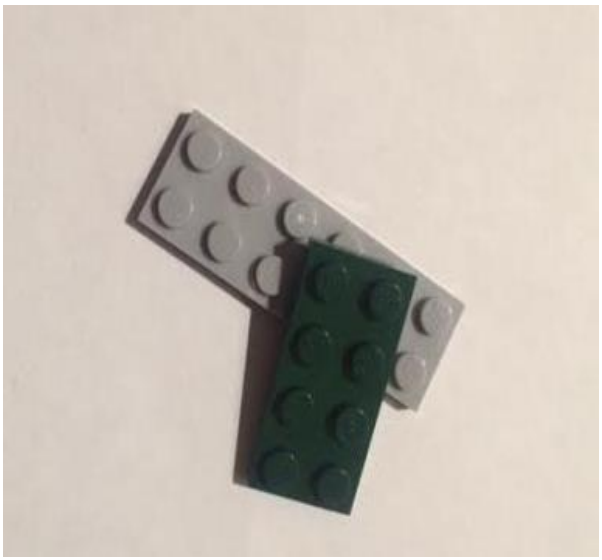
Shape Test Number 1: Basic Shape Recognition

Procedure:

1. Place a single Lego into the Image Processing Chamber.
2. Run the algorithm on the BeagleBone for shape recognition on the Lego.
3. Check the result of the algorithm to ensure that it recognizes the shape if it is in the library, or if it processes it as an error if it is not.
4. Repeat steps 1-3 for all the Legos in the library as well as a variety of those that are not in the library.

One of the additional situations that could arise is that of overlapping Legos in the image processing chamber. If two Legos overlap within the chamber, the system will likely recognize this as a single Lego. While this is certainly not an ideal situation, the algorithm should be able to simply register this situation and drop the two Legos into the error bucket.

Shape Test Number 2: Overlapping Legos

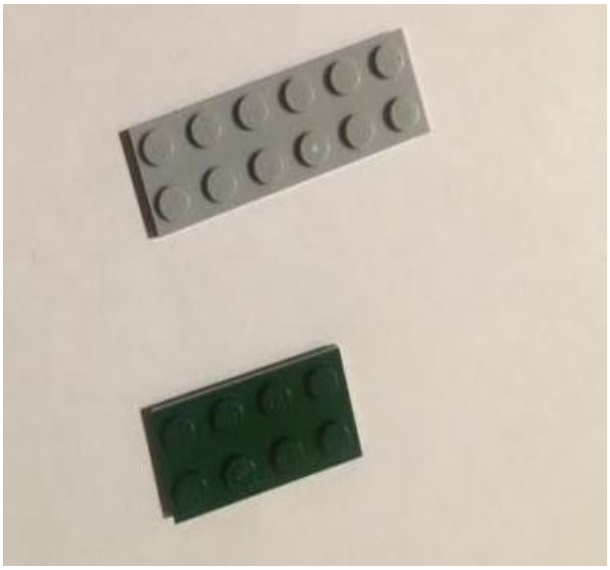


Procedure:

1. Place multiple Legos within the Image Processing Chamber, making sure that they overlap each other.
2. Run the algorithm on the BeagleBone for shape recognition on the Lego.
3. Check the result of the algorithm to ensure that it process the Legos as an error, rather than mistaking it for an actual Lego.
4. Repeat steps 1-3 multiple times to ensure consistency of results.

That covers the basic situations that could complicate the simple color and shape recognition abilities of the system, but there is also the possibility that multiple shapes could appear on the image WITHOUT overlapping. This isn't so much of a test of the shape or color recognition algorithm, but the ability of the algorithm to know WHEN to process what is on the image, or possibly how much of the image to process. In this case, the first Lego passed through will have already been processed, but the second Lego will be the one that creates the trouble. The algorithm should be able to recognize this and be able to handle it. The difficulty with this test though, is that it relies on a lot of different parts to be finished and working to properly test it. This simply means that this will be one of the last bits of testing that the project will be subjected to.

Multiple Legos (Separated) Test



Supplies:

- Image Processing Chamber
- Atmega
- BeagleBone Black

- Conveyor Belt
- Rotating Arm System
- Computer

Preparation:
Set up the conveyor belt, Image Processing Chamber and Rotating Arm System as it will be in the final prototype. Power up the BeagleBone, connect it to the Atmega and display it via the computer.

Procedure:

1. Place singular Legos along the conveyor belt. Ensure that the Legos are close enough that there will be more than one in the Image Processing Chamber at a time.
2. Begin running the system.
3. When the first Lego has reached the Image Processing Chamber, ensure that it is processed properly and is ready to be dispensed into the rotating arm system.
4. Continue to run the system as the second Lego begins to enter the chamber. Ensure that the algorithm DOES NOT begin to process the second Lego until it has received a signal from the sensor on the rotating arm system that the first Lego has left.
5. If another Lego has gotten into the chamber before the second Lego has been processed, ensure that the algorithm works correctly in ONLY processing the second Lego.
6. Repeat this process multiple times. Space the Legos out at different intervals and use multiple different types of Legos to ensure that the test occurs under multiple situations.

This series of tests should be able to effective test every aspect of the image processing algorithm that was used in the project. This will ensure the best possible results when the project is finished to make the Lego sorter the best it can be.

7.4.3 Error Handling Testing

Majority of test plans involve similar procedure, as all that is needed are the sensors and the LCD. The following tests can only be done when all testing on the LCD screen pass. Attach motors, load sensors, and power sensors to the main controller in order to communicate to the LCD screen. Have the LCD be set to the status screen. Here were the sensor error window will appear.

Supplies:

- LCD Screen
- Atmega32u4
- Beagle Bone Black
- Camera
- DC Motors
- Servo Motors
- Stepper Motors
- Motor Sensors
- Motor Drivers
- Power Sensor
- Power Supply

Motor Sensors – It is important that all the motors run at the appropriate speed. Especially the motors driving the conveyor belt at they serve as a mean to separate the LEGO® parts. This applies to all

Procedures:

This part of the procedure applies only to DC motors.

1. Run the motors. Check the read the data of the motor that is running on the LCD is showing the correct reading.
2. Increment speed of motor while monitoring it vial the LCD screen.
3. Continue until speed suddenly drops to initial speed. The monitor should reflect this change in speed.
4. Repeat this procedure except decrease the speed instead.

This part of the procedure applies to all motors

5. Now carefully hinder the motor spinning to simulate an obstruction.
6. The motor should come to a halt. Check the LCD screen. A window should appear displaying a message that there is a jam in a motor.
7. Remove the obstruction.
8. Press resume on the LCD screen. The motor should start running again.

Load Sensors – If more LEGO®s are being loaded into the sorter as it is still running then is possible that containers to soon become full. Therefore the system need to determine when a container is full and notify the user through the LCD screen that the Error bucket is full meaning that there are other buckets that are full. In this test we check if the system can recognize a full error bucket.

Procedure:

1. Hard code senor to represent an error bucket.
2. Slowly increment the amount of weight put on the sensor.
3. Monitor the LCD screen to keep track of the amount of weight put on the sensor.
4. Continue incrementing weight until window appears on screen indicating a bucket is full.

Camera – The camera must be on center in order to properly view the LEGO® part and the mirror in order to see the details of the part otherwise mistakes in comparison are more likely to happen.

Procedure:

1. Set camera to view the marker.
2. Check if LCD screen is seeing the same marker by pressing the camera tab.
3. Slowly move the camera off marker while monitoring the LCD screen.
4. Continue until an error window appears stating that the camera is off its mark.
5. Move the camera back in place. The pop up should then disappear.

Power Sensors – Because of the amount of low power devices it is important to monitor the power coming out of the power supply. If the output is too high the power supply must shut off to prevent any damage to the rest of the system.

Procedure:

1. Set a safe maximum level to test with.
2. Increase power slowly.
3. Monitor activity on LCD screen.
4. Continue until entire system shuts off.

7.4.4 Hardware Communication Testing

One of the most important things to test within the project is how well the algorithm on the BeagleBone communicates with the various hardware that will be used. Multiple test need to be done to make sure that the BeagleBone is sending signals to the hardware properly, but also that those are the RIGHT signals.

The first set of tests that will be run will be that of the conveyor belts. The conveyor belts will be fairly necessary in other parts of the testing, including portions of testing the image processing algorithm (see 7.4.2, Multiple Legos (Separated) Test). The initial tests of the conveyor belt will start off in an extremely basic manner. These first tests are simply to confirm that the signals are being sent correctly to the conveyor belt, so for these tests an extremely simple test program can be run that sets the speed of the conveyor belt and can alternate it turning on and off.

Conveyor Belt Test

Supplies:

- BeagleBone Black
- MSP430 Microcontroller
- Conveyor Belt System
- Computer

Preparation:

Connect the BeagleBone Black to the MSP430 Microcontroller in the same manner as the final prototype, and connect the MSP430 to the conveyor belt system. Power up and display the BeagleBone via the computer. Prepare a simple test program for the conveyor system and run it with the BeagleBone.

Procedure:

1. Run the sample program on the BeagleBone that starts the conveyor belts.
2. Confirm that the conveyors move at proper speeds and turn off and on correctly when instructed to.
3. Repeat this process until the conveyor belts are all behaving properly and until proper settings for the final prototype have been determined.

Another simple part that will need to be tested will be that of the lift arm. The lift arm will be running at intervals throughout the process of the Lego sorter and it needs to be tested as such. This will require another very simple test program that simply sends the required signals to the lift arm.

Lift Arm Testing

Supplies:

- BeagleBone Black

- MSP430 Microcontroller
- Lift Arm System
- Computer

Preparation:
Connect the BeagleBone Black to the MSP430 Microcontroller in the same manner as the final prototype, and connect the MSP430 to the lift arm system. Power up and display the BeagleBone via the computer. Prepare a simple test program for the conveyor system and run it with the BeagleBone.

Procedure:

1. Run the sample program on the BeagleBone that starts the conveyor belts.
2. Confirm that the lift arm is moving at an appropriate speed and at the correct intervals.
3. Repeat this process until the lift arm is behaving accurately and until a desired setting for the final prototype is arrived at.

The final hardware communications testing that needs to be done involves the rotating arm system. This will likely be the most difficult of the testing that needs to be done. It will involve communicating with the rotating arm system and keeping track of where the rotating arm is at and where it needs to go. This will require more extensive testing than that of the conveyor belts and the lift arm.

Rotating Arm System

Supplies:

- Rotating arm system
- BeagleBone Black
- MSP430 Microcontroller
- Computer

Preparation:
Connect the BeagleBone Black to both the computer and the microcontroller, and then the microcontroller to the rotating arm system as it will be in the final prototype. Prepare the BeagleBone with a sample code that will test the various necessities of the rotating arm system.

Procedure:

1. Begin running the sample code to test the rotating arm system.
2. Ensure that the rotating arm system stops at the appropriate positions, moves at an appropriate speed and starts when instructed.

3. Repeat this process for a multiple of different scenarios to test the various situations that the system may encounter in the final prototype.

These tests cover the three major hardware components in the system. Before any of these test are done the hardware tests will have been performed to ensure that the parts are already working, these tests are to confirm that the software portion of the process is communicating in the proper manner and that the signals are being communicated correctly between the BeagleBone, the microcontroller, and the hardware itself. After all of these tests have been performed on the system, all that's left will be to test everything together in the final prototype.

Final Prototype Testing

Supplies:

- All subsystems, including the Lift Arm System, the Conveyor Belt System, the Rotating Arm System, the Image Processing Chamber and the User Interface.
- Tiva-C and BeagleBone Black

Preparation:

Prepare the final prototype by piecing everything together in the same manner that the final project will be prepared. Feed conveyor belt 1 into conveyor belt 2, feed conveyor belt 2 through the image processing chamber and over the rotating arm system. Connect the BeagleBone Black and the Tiva-C to each of the subsystems and power on the machine.

Procedure:

1. Test each aspect of the User Interfaces options to ensure that they are all working at first glance. Make sure displays are working properly, that kill switch is functional, and that the interface is assigning the buckets the proper Legos.
2. Begin running the Lego sorter.
3. Pore over the sorters process in search of any sign of errors or needed changes including:
    a. Lift Arm speeds and intervals
    b. Conveyor Belt speeds and stops
    c. Rotating Arm timing
    d. Image Processing error/success ratio
4. Repeat this process THOROUGHLY to check that every aspect of the sorter is working properly.

The goal of the Lego sorter is to create a project that is highly accurate, and as efficient as possible given the constraints of its accuracy. In order to ensure this outcome, testing is

of utmost importance with the project. The sorter needs to be tested thoroughly and repetitively so that no stone is unturned in making sure that the project is the best it can be. As long as these tests are adhered to, it should help to produce a superior project.

8.0 Milestone

Figure 8.0 –a shows the milestone chart for the project including deadlines for subsystem specific research, testing, and final prototype plans.

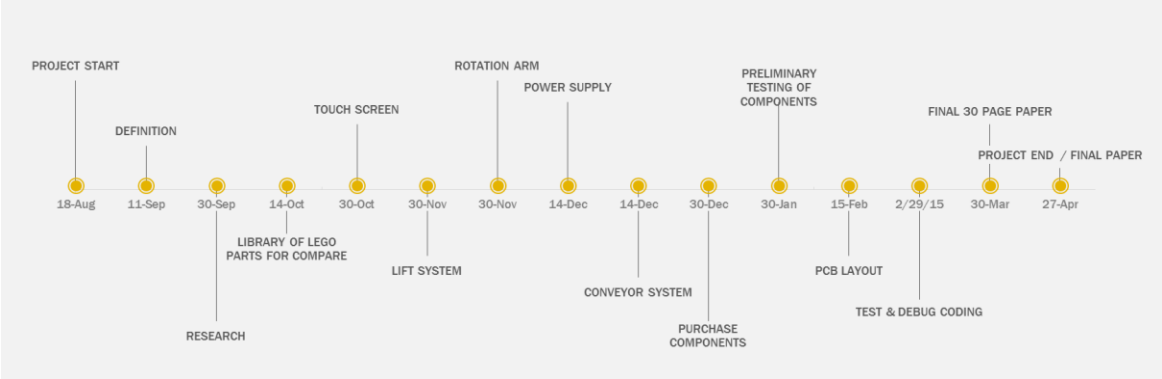


Figure 8.0 – Milestone timeline for Senior Design 1 & 2

Appendix A – Copyright

[1] http://www.ti.com/lit/ds/symlink/tps55340.pdf

[2] http://www.ti.com/lit/ds/symlink/tps54525.pdf