UNIVERSITY OF VICTORIA

Department of Mechanical Engineering

MECH 458 – Mechatronics

Final Design Proposal For:

UVIC Inspection/Sorting System

Submitted to:

Bingxian Mu

Submitted by:

Amber MacNeill (V00827818)
Rachel Brennan (V00817799)

Submitted on:

December 10, 2017

Executive Summary

This project involved creating a control system to sort items of four different material types via the apparatus shown in figure 1. The apparatus used included a conveyor belt driven by a dc motor, a sorting tray driven by a stepper motor, an AVR microcontroller, and several sensors. The sensors used are connected to the microcontroller as shown in figure 5. Performance criteria consisted of a speed test, where 48 items had to be sorted in under 60 seconds with minimal errors. It was also required that the system have two additional functionalities, one to pause (and resume) the system and one to ramp down the system so that it can't be started up again. Both functions also had to display the number of sorted and unsorted items on LEDs.

Several methods were used in the design, which was interruption based such that the system could react instantly to changes in state. A linked queue was used to manage the incoming object's type and order dynamically. The interrupts used are listed below:

- Hall effect sensor - Used to configure the stepper.
- Exit sensor - Used to note when an item has reached the end of the belt. This interrupt was used within a finite state machine to handle the dropping of the items.
- Optical/ADC reflective - Used to classify the item's type and add it to the queue.
- Pause button - Sets a flag such that the pause function will start after the stepper has finished rotation.
- Ramp down button - Disables ramp down interrupt and enables timer interrupt.
- Timer interrupt - Used to delay system shutdown so that the items remaining on the belt can be sorted

To optimize the stepper, it's excitation mode was chosen to be full step with two phases on, as shown in figure 2, which creates more torque than using single phase. The stepper was also sped up by reducing the time delay between steps to 7ms, and accelerating to that speed non-linearly (see section II - C). Unlike most components of the system, the stepper motor was coupled with the main loop for this project.

The algorithms used in this control system are shown in flowcharts in section III-B. Falling and rising edge interrupts were used in several algorithms to sense the position of the item. Flags were also used in several functions and interrupt service routines to notify that an event had taken place, which were checked in the main routine. A state machine was used at the exit sensor, and main loop, to manage the various states at that point, see figures 9 and 13.
The maximum performance this system achieved was 35 seconds with zero errors during the demonstration. This performance is possible with moderate stepper speed (to avoid stalling), a relatively high belt speed, and item loading that maintains adequate spacing between the pieces.

This system successfully met the performance criteria. If more time was available the state machine would be made to handle more cases and the stepper method would be decoupled from the main loop. A faster clock cycle would also have been beneficial to the system, allowing for more interrupts to be handled simultaneously.

Abstract

This project involved creating a control system to sort 48 pieces of four different material types into a compartmentalized tray within 60 seconds with minimal errors. It was also required that separate functions be created to pause and shutdown the system while displaying a count of sorted/unsorted items. The system used consisted of a conveyor belt driven by a dc motor, a sorting tray driven by a stepper motor, an AVR microcontroller, and several sensors. The sensors acted as interrupts in the code such that state changes could be handled in ‘real time’. Several methods were used in the design including dynamic memory control, full step two phase stepper excitation and acceleration, timer interrupts, finite state machine/flowcharts, analog to digital conversion, and a specific item loading procedure. The best performance recorded with this system was 35 seconds and zero errors. To achieve this, it was necessary to limit the stepper speed (to avoid stalling), use a relatively high belt speed, and load pieces with appropriate spacing between them. This system successfully fulfilled the performance requirements, but given more time it would be beneficial to have the state machine handle more cases,implement a faster clock cycle, and decouple the stepper motor from the main loop.

Acknowledgments

We would like to express our gratitude to Bingxian Mu, and the Department of Mechanical Engineering, for providing us with the opportunity to complete the Mechatronics Project “UVic Sorting/Inspection System”. Thanks to Bingxian Mu’s lectures, we gained adequate knowledge that served as a basis for this project. We would also like to thank Patrick Chang for his guidance, advice and support, without which this project would not have been successful. The challenges faced throughout this project has provided us with many learning opportunities, for which we are very grateful. It is our pleasure to extend our thanks to all faculty, and friends, for their dedicated support that helped make this project possible.

Amber MacNeill
Rachel Brennan

Table of Contents

List of Figures

I. Introduction

A - Problem Statement

A control system is required to inspect, classify and sort various objects using the sorting apparatus shown below in Figure 1. The objects are small cylindrical pieces with concentric holes cut through the centre, and come in four types: black plastic, white plastic, steel and aluminum. A set of 48 pieces (12 of each type) is required to be sorted within 60 seconds, in an unknown order. The items are to be sorted into bins at the end of the conveyor belt after undergoing inspection and classification. The classification of each object is to be done using a reflective sensor and/or a magnetic sensor. The reflective sensor uses Analog to Digital Conversion (ADC) to identify differences between materials. The reflective sensing station and ferromagnetic sensing station are approximately halfway down the conveyor. The exit sensor for the apparatus is shown in the figure below at the end of the conveyor, and is to be used to stop the belt if the tray is not in position.

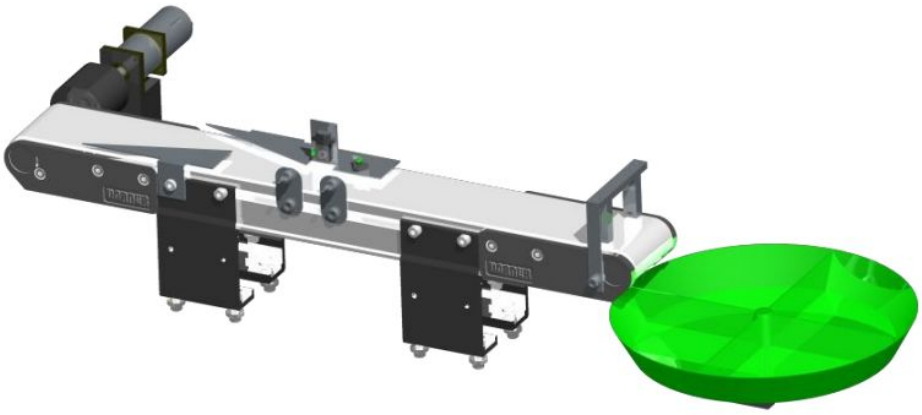

Figure 1: Sorting Apparatus [1]

The system must incorporate a pause function that will halt inspections and display counts of each item type that has been sorted, as well as a count of all unsorted items. Additionally, a system ramp down must also be included. The ramp down function is to sort the remaining items on the conveyor belt, shut down the system, and display continuously the counts of each item that has been sorted. After system ramp down, the system must not start up again. Efficiency and accuracy of sorting is an important system consideration. The microcontroller used for this project is the AT90USB1287 Atmel Microcontroller, and the corresponding microcontroller manual was used thoroughly throughout the design process [2].

B - Purpose of Design and Design Overview

The design of this system uses various interrupts in order to decouple processes from the main loop. This allows state changes in the inspection/sorting system to be handled immediately by an interrupt service routine (ISR). The main components of the system are the stepper motor operation, the conveyor belt operation, item classification, pause, and ramp down. These components are handled within the code in the following places:

- Stepper motor operation:
    - Hall effect sensor interrupt
    - Main loop
    - Update stepper function
    - Configure stepper function
- Belt conveyor operation:
    - Exit sensor interrupt
    - Main loop
- Item Classification:
    - Optical sensor interrupt
    - Reflective ADC sensor interrupt
- Pause:
    - Pause button interrupt
    - Pause/ramp down routine
    - Main loop
- Ramp down:
    - Ramp down button interrupt
    - Timer interrupt
    - Pause/ramp down routine

The configuration of the stepper motor is completed using both the hall effect sensor and the main loop. Beyond stepper configuration, stepper motor control is handled by the update stepper function called in main loop whenever an update in position is required.

The belt conveyor operation is controlled by the exit sensor to stop the belt as required, and by the main loop to start the belt. A state machine was implemented to control belt operation, and dropping, at the end of the conveyor. The state of the first two items in a queue contribute to the decisions made for belt control.

Item classification is completed by the reflective sensor's optical interrupt and ADC interrupt. Upon entering the optical sensor, ADC conversion begins and continues until the item leaves the optical sensor. At this point, the item is classified based on the lowest ADC value taken during the conversions.

Pause and ramp down are controlled using two buttons connected to interrupt ports. Pause uses the main loop, after a flag has been set, to allow a second button press to start the system again. Ramp down calls the pause/ramp down routine inside of the timer interrupt, which results in an infinite routine as no other interrupts can be handled during this ISR.

II. Methods and Design Approach

The first design method employed for this system was a comprehensive flowchart (see Appendix A), which was used as a foundation for the code developed. This foundation was modified to meet the needs and time constraints of the project. In order to incorporate all of the various components into the code (eg. item classification, stepper motor acceleration, etc.) several different methods were used.

A- Linked Queue

A linked queue was used to manage the items on the belt such that the memory could be handled dynamically. The code for this, LinkedQueue.c and LinkedQueue.h (Appendix D) was provided through coursespaces [3]. A bug in the 'dequeue' function was fixed (pointed tail to NULL if queue is empty), and a 'nextValue' function was added to return the material type of the second item in the queue. Items are added to the queue in the reflective optical sensor's ISR.

B- Debounce

An issue was encountered when the pause interrupt fired while the stepper was moving, causing the stepper to get out of sync. To avoid this error, a method was employed to allow the stepper to finish it's rotation before pausing the system. To do this, a flag is set within the pause ISR and the belt is stopped. This flag is checked within the main loop of the function, and if the flag is set, the pause routine function will be called. Therefore, if the pause interrupt fires while the stepper is updating the pause routine will not be called until the stepper finishes moving.

C- Stepper Motor Control

Stepper Speed

In order to optimize the stepper motor speed we reduced the delay between steps, reaching a ‘top speed’ at a 7 ms delay between each step of 1.8°. A method of acceleration needed to be introduced to reach this speed. After much trial and error it was found that setting the delay between steps to 20 ms, then 16 ms, 12 ms, 9 ms, and 8 ms before jumping to 7 ms, and decelerating in reverse order, was fast and effective.

Stepper Excitation Mode

The stepper motor excitation mode was chosen to be full step, with two phases on, as shown below in Figure 2. This mode operates by energizing both coils at the same time, and allows the stepper to move with more torque than full step single phase, and thus improves performance.

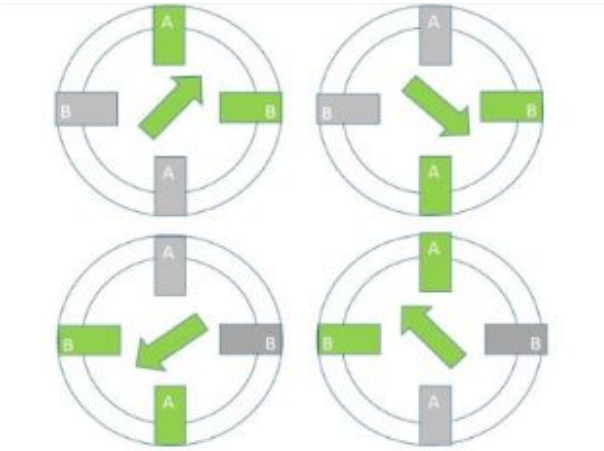

Figure 2: Stepper Motor Excitation Mode [4]

D- Ramp Down

The ramp down function shuts down the system and displays the item count after all the items on the belt have been sorted. To achieve this a timer interrupt was used. When the ramp down button is pushed, the ramp down ISR disables the ramp down interrupt, enables the timer interrupt and sets the timer so that it will interrupt shortly after. In the timer ISR, the size of the queue is checked. If the queue is empty it sets the interrupt to fire again in two seconds and sets a flag (in case there are items before the first set of sensors). If the queue is still empty when the interrupt is fired a second time the pause routine will be called. If the queue is not empty during any of the checks the timer will be set to two seconds.

E- Item Classification

Item classification is completed when an item leaves the reflective optical sensor, using the lowest value from the ADC conversions. For accurate classifications, the ADC values for the parts are calibrated before system start-up, using a separate calibration program (Appendix C). The ranges for the values are hard-coded, with added tolerances, into the code for the sorting/inspection system. It was found that item's can be accurately classified without the use of ferromagnetic sensor. The use of the ferromagnetic sensor would allow the program to be more robust, however, given time constraints and the risk of loss of accuracy for the main program, it was decided to classify the item's using only the reflective sensing station.

F- Exit Sensor State Machine

An issue was encountered with the exit sensor at the end of the belt, as some possible item states were not being handled properly. To manage this a state machine was used, with the following possible item states:

- No state
- Falling
- Mid Exit
- After Exit

When the exit sensor interrupt fires the system first checks whether the item is 'entering' or 'leaving' the sensor (first or second edge), and then checks whether or not an item is falling. If the interrupt fires on the first edge and an item is falling, the state of the entering item is checked (in order to control whether the belt should stop to allow the stepper to move or not). Figure 9 and 13 show the state machine in the form of a flow chart.

G- Belt Loading

It was found that loading six pieces at a time, then waiting for them to reach the exit sensor before loading the next six, was an effective loading method. This ensured that items weren't passing through the mid-belt sensors while other items passed through the exit sensor, which would overload the system with interrupts. It also allowed for good item handling (ie. proper spacing between pieces while the belt stopped to allow for stepper rotation).

III. Results

A- Technical Description of Inspection/Sorting System

A high level block diagram of the inspection/sorting system is shown below in Figure 3. This system accepts input from six sensors that the system uses to control the output to the LEDs, stepper motor driver, and conveyor motor driver. This system runs at a clock cycle frequency of 1 MHz, allowing the system to respond in time to the various interrupts.

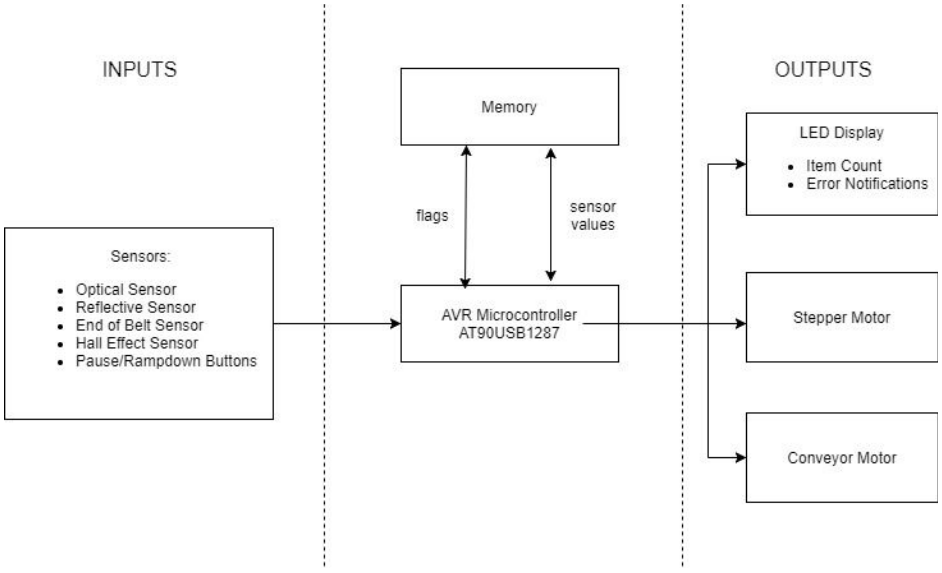

Figure 3: Block Diagram of Sorting System

Circuit Diagram

The final layout of our board is shown below in Figure 4. Working from the top of the image down, the eight LEDs can be seen on the left connected through resistors to the breadboard, while the pause and ramp down are situated on the right side of the board (ramp down closest to right edge). Power is connected through the second row from the top and through the bottom row, while ground is through the top row and second row from the bottom. Two low pass filters can be seen on the board that limit the input signal for the hall effect and exit sensors. Wire connections for the stepper motor driver, DC belt motor driver, pulse width modulation (PWM), sensors and buttons are shown below as well .

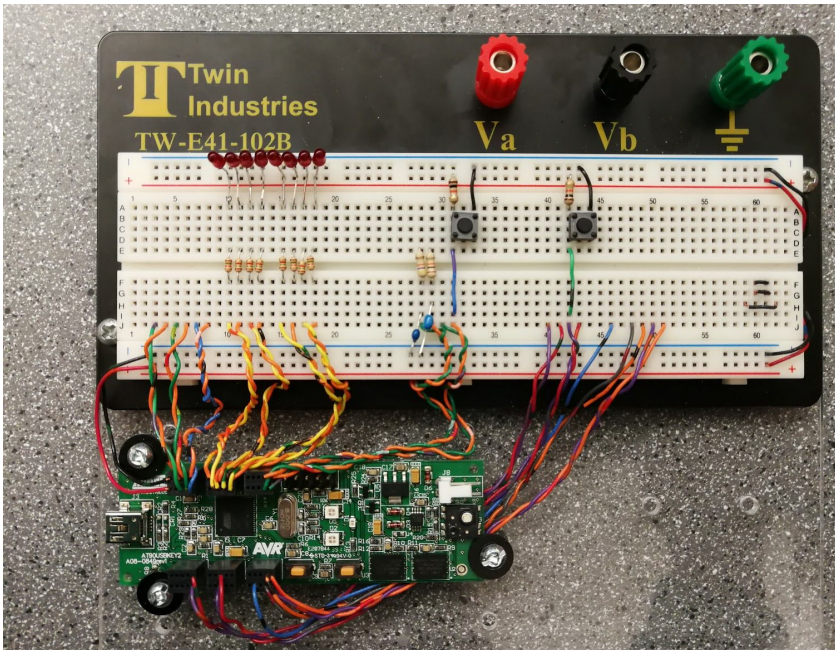

Figure 4: Circuit Layout

The final pin connection diagram is shown below in Figure 5. It can be seen that the stepper's integrated dc motor driver is connected to the microcontroller through pins PA0 to PA5, and that the belt's dc motor driver is connected through pins PB0 to PB3. Eight LEDs are connected to pins PC0 to PC7 to display various information (ie. number of items sorted). The LEDs were also used throughout the design process for debugging purposes. The interrupts corresponding to the pause function, hall effect sensor (HE), slotted optical sensor (EX), optical sensor (OR), and the Ramp down function were allocated pins that allow for interrupt operations. These sensors/buttons were connected through pins PD0 to PD3 and to PE5 in respective order. The reflective sensor, RL, is connected to PF1 such that the ADC may convert the values read by the sensor. It should be noted that the low-pass filters were added on the board for the hall effect and exit sensors, but are not shown in circuit diagram.

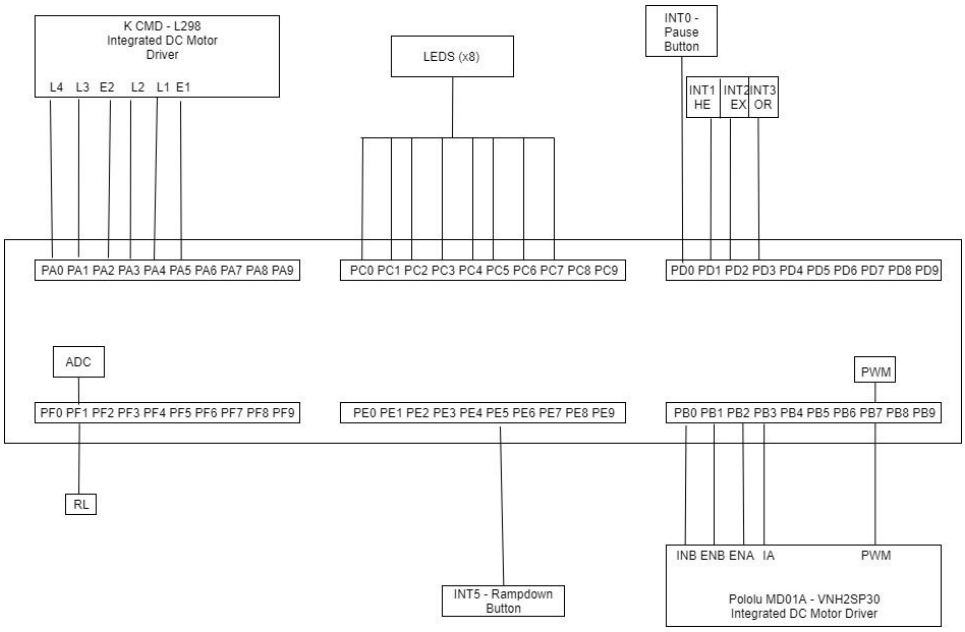

Figure 5: Pin Connection Diagram

B- System Algorithm

The algorithms for this project are described using various flowcharts for the interrupts, the main loop, and other major routines.

Hall Effect Sensor Interrupt

The purpose of the hall effect sensor interrupt is to signal when the stepper motor has reached a known position. In the case of this system, the hall effect sensor will fire an interrupt when it reaches black. The flowchart for this interrupt is shown in Figure 6 below. As shown in the flowchart, once the interrupt fires, it will set a flag to signal that stepper configuration is complete, and it will disable future occurrences of this interrupt.

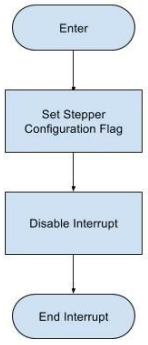

Figure 6: Hall Effect Sensor Interrupt

Optical Sensor Interrupt

The optical sensor at the reflective sensing station will fire an interrupt as an item enters the sensor, and as an item leaves the sensor. When an item enters, ADC conversion will start and an ADC flag is set. At this point, the reflective sensor's ADC interrupt will take over until the ADC flag is reset. Upon leaving the optical sensor, the ADC flag will be reset and item will be classified based on the lowest ADC value that was determined in the ADC interrupt. In the case that an item is out of range, this system will classify the item as black and display an error. The reason for this is to minimize the risk of accumulating more errors. The flowchart for the procedure of the optical sensor is shown in the Figure 7 below.

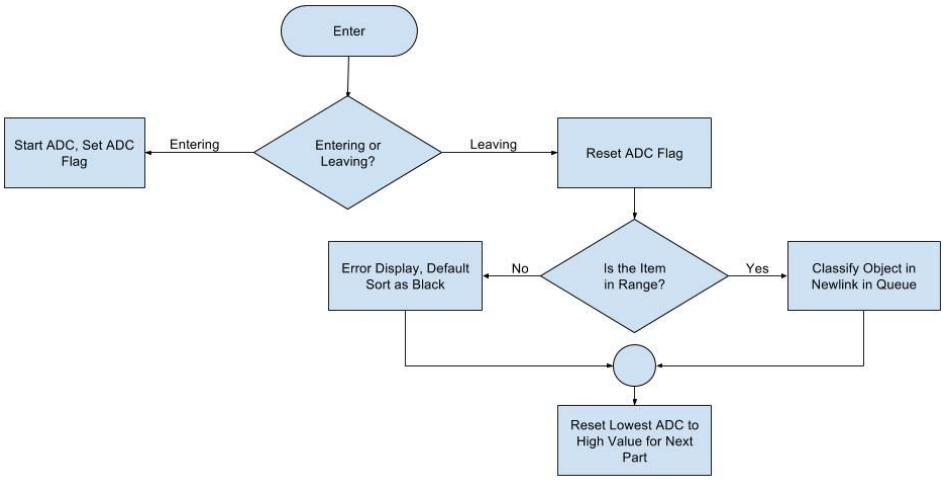

Figure 7: Optical Sensor Interrupt Flowchart

Reflective Sensor Analog-To-Digital Converter Interrupt

As shown in the flowchart below (Figure 8), the ADC interrupt for the reflective sensor is responsible for keeping the lowest ADC value for each part. To do this, the ADC value is read and if it is lower than the previous ADC value, the lowest value will be updated and the ADC conversion will start again. Once the ADC flag is reset by the optical sensor's interrupt, the ADC interrupt will terminate for that part.

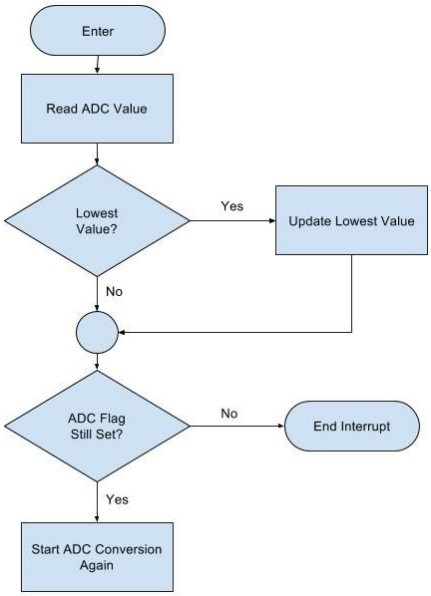

Figure 8: Reflective ADC Interrupt Flowchart

Exit Sensor Interrupt

The purpose of the exit sensor interrupt is to stop the belt as required, and to set the state of the current item and the next item. The exit sensor interrupt is shown in Figure 9 below, and is fired when a part enters and leaves the sensor.

Upon entering, the state of the current item is checked. If the item state is ‘not falling’ and the stepper is not in position, the belt is stopped to allow the stepper to get into position. If the item state is ‘falling’, and the next item is the same the exit sensor will allow the item to pass right through into the correct bin. If the item is ‘falling’, but the next item is different the belt is stopped and the next item’s state is set to ‘mid-exit’. The item will remain in this position and state until the stepper is in position.

Upon leaving, the state of the current item is checked again. If the item state is ‘not falling’, it is set to ‘falling’ and the item type is added to a count of items that have been sorted. If the item state is ‘falling’, and the next item is the same a timer flag is reset, the first item is dequeued, and the item type is added to a count of items that have been sorted. If the item state is ‘falling’, but the next item is different, this next item’s state is set to ‘after exit’.

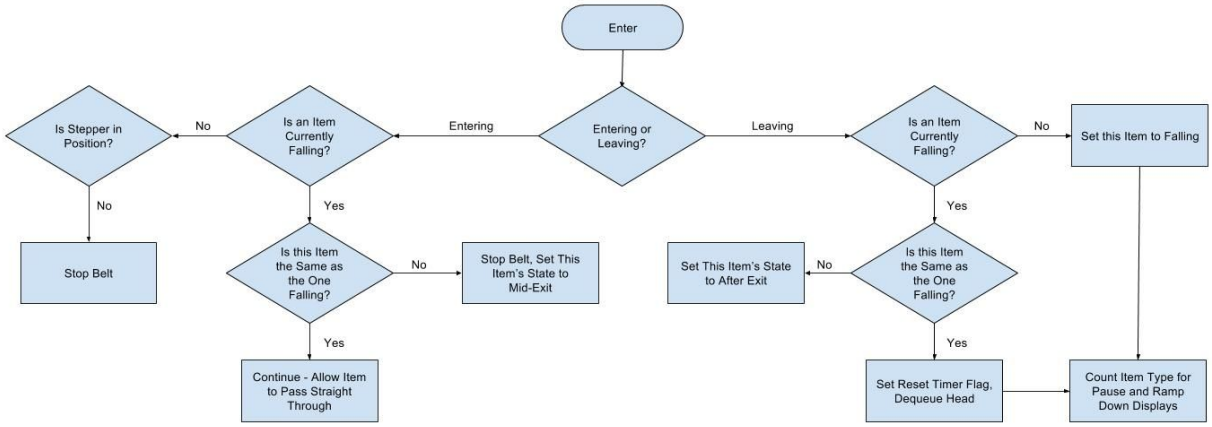

Figure 9: Exit Sensor Interrupt Flowchart

Pause Button Interrupt

The pause button interrupt works with the pause routine function, in order to allow the stepper motor to complete its' movement to the desired position before pausing the system. The flowchart for the pause button interrupt is shown below in Figure 10. Upon firing, the interrupt will determine if it is resuming or pausing by checking a flag set by the pause routine function. On system start-up, the first button press will allow the interrupt to go into pause mode. For pause, this interrupt will set a pause button flag, stop the belt, and reset the LEDs for the pause display. For resume, this interrupt will reset the pause button flag, start the belt and reset the LEDs.

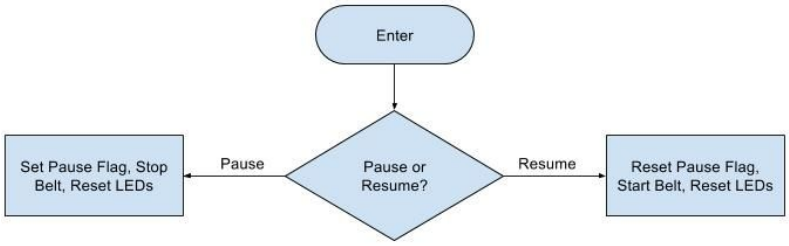

Figure 10: Pause Button Interrupt Flowchart

Ramp Down Button Interrupt

The ramp down button interrupt works with Timer interrupt, and pause/ramp down routine function to perform a system shut-down that will allow sorted item counts to be displayed on the LEDs repeatedly. Upon firing, the interrupt disables the ramp down button interrupt and enables the timer interrupt, as shown in Figure 11 below.

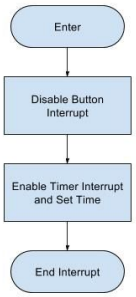

Figure 11: Ramp Down Button Interrupt Flowchart

Timer Interrupt (For Ramp Down)

The timer interrupt, as shown in Figure 12, is used to determine whether the system is ready for shut down. Once the interrupt is triggered, it will check if the queue is empty and whether or not this is the first check. If the queue is empty and it is the first check, it will set the interrupt timer to approximately 2 seconds, and will set a flag that the first check has been completed. After the first check, the ISR will check if the queue is empty again. If the queue is not empty, the interrupt timer is set for another two seconds, if the queue is empty the belt is stopped, a flag is set of end of system routine, and the routine is called. At this point, the system will not start up again (until a manual reset) and will display LEDs continuously after entering the final pause/ramp down routine.

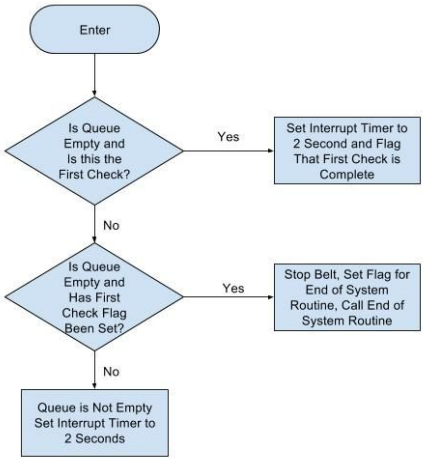

Figure 12: Ramp Down Timer Interrupt Flowchart

Main Loop

The flowchart for the main loop is shown below in Figure 13. Upon system start up, the main loop initializes all i/o pins required, enables PWM for the conveyor belt, and enables all necessary interrupts. This is followed with configuring the stepper motor, setting up the linked queue, and starting the conveyor belt. If there are no items in the queue, the system will wait and

check if the pause or ramp down buttons have been pressed. If the queue is not empty, the desired position is updated and the state of items in the queue are checked. If the item state is ‘mid-exit’, the stepper will be updated, the item’s state and the next item’s state will be set to ‘no state’, and the belt will start. If the item’s state is ‘after exit’, the stepper will be updated, the item state will be set to ‘falling’, the next item’s state will be set to ‘no state’, and the belt will start. If the item state is ‘falling’, a timer is set before dequeuing. The timer is resettable and will be reset if another item of the same type comes through the exit sensor. Once the timer is expired, the item will be dequeued, the desired position will be updated, the item’s state will be set to the next item’s state, and the next item’s state will be reset to ‘no state’. If the item does not have a state, the stepper is updated, the pause/ramp down flag is checked, and the initial position is set to the current position. It should be noted that the update stepper routine will only proceed if the stepper is not already in position. The pause flag is only checked after the stepper has been updated in order to allow the system to resume without misaligning the stepper motor.

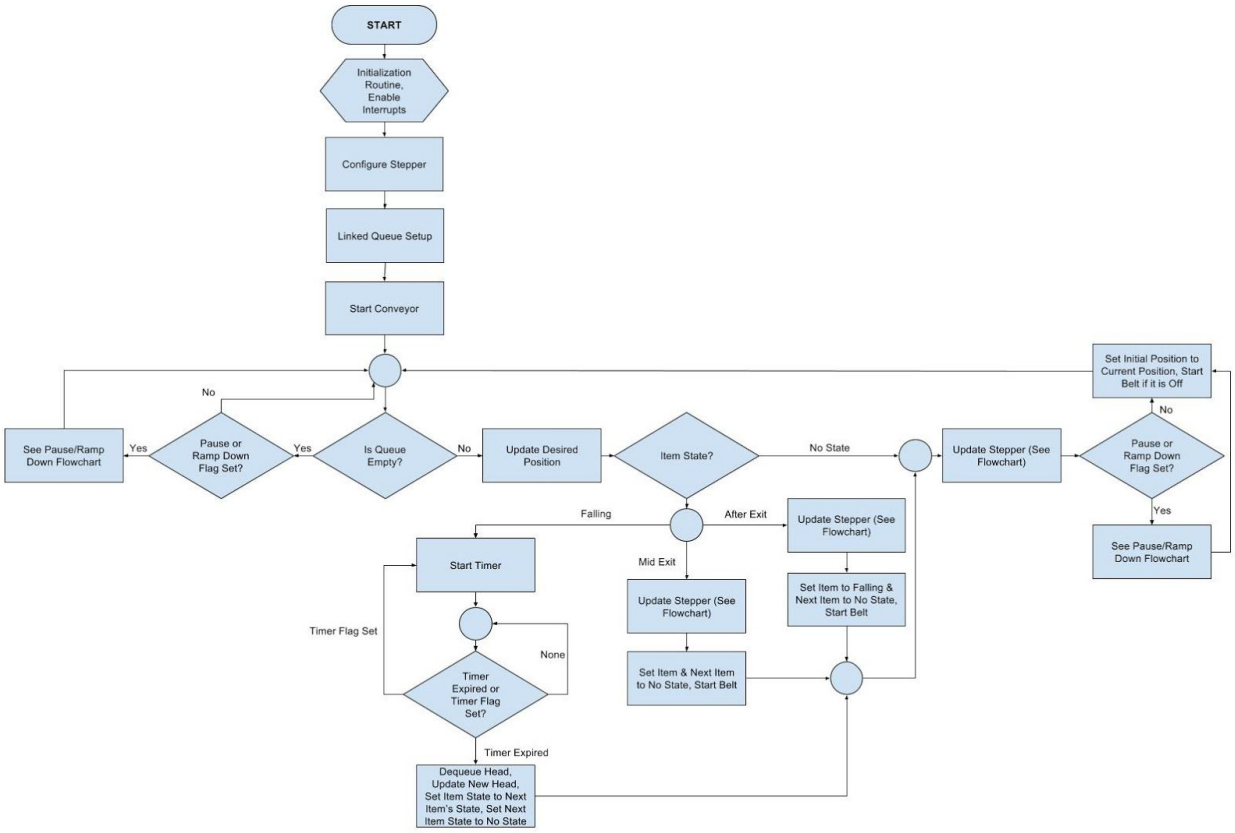

Figure 13: Main Loop Flowchart

Pause/Ramp Down Routine

The pause/ramp down routine is called in the main loop when a flag is set by either the pause button interrupt or the timer interrupt used for ramp down. The flowchart for the pause/ramp down routine is shown in Figure 14 below. Debouncing for the buttons occur in the pause/ramp

down routine, rather than the interrupt, so that the stepper motor is not affected until it has come to a complete stop. In the case that the routine was called (and flag set) in the timer interrupt, the LEDs will continuously display each item count on repeat until a manual reset has taken place.

If the flag was set in the pause button interrupt, the button is debounced and a flag is set to allow the system to resume on the next button press. At this point, the LEDs continuously display each item count until the pause button interrupt fires again to resume. Before exiting the routine, upon signal to resume, the unsorted item count is reset, the resume button is debounced, and a flag is set to allow the system to pause on the next button press.

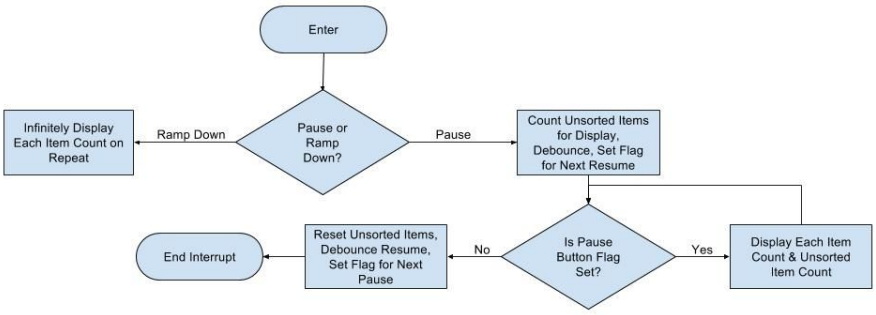

Figure 14: Pause/Ramp Down Routine Flowchart

Update Stepper

The update stepper routine is responsible for moving the stepper into position so that items can be sorted. The flowchart for this routine is shown in Figure 15 below. As soon as the stepper is in position, the routine is exited. If the stepper is not in position, the direction is determined by calculating the difference between the current position and the desired position and choosing the shorted path. Next, the stepper moves one step, updates the current position and then sets the time before taking the step (ie. the speed of the stepper is determined). To determine the speed of the stepper, the position on the acceleration/deceleration ramp is found and the time between steps is set accordingly.

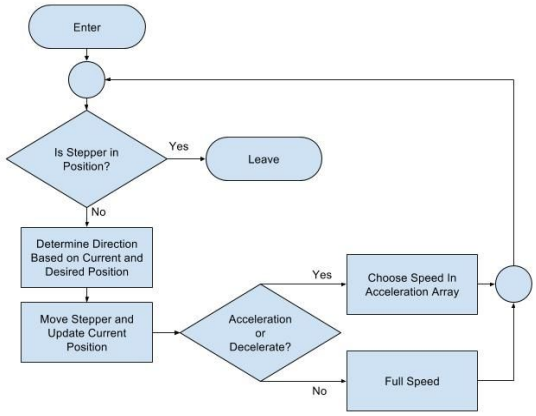

Figure 15: Update Stepper Flowchart

C- System Performance Specifications

System performance is primarily based on sorting 48 pieces as quickly as possible with minimal errors. The maximum performance seen by this system was a sorting time of 35 seconds with zero errors, which occurred during the demonstration of the system. There were issues with the stepper motor's acceleration and top speed (ie. there wasn't enough torque), so the speed and acceleration had to be lowered. In order to achieve the maximum performance mentioned above the belt speed had to be relatively high, and the item loading had to be very accurate (if the spacing between items was too small the system would mis-sort).

To achieve better performance the belt could be stacked. That is, adding pieces to the belt before the previous set of pieces have reached the end. This increases the risk of the system mis-sorting due to limited spacing between pieces, while also introducing the risk of overloading the system with interrupts. Performance could also be increased by making the stepper acceleration and speed more aggressive, but this is likely to cause errors if items pile up on the tray.

D- Operation

The operation of the system, depends on a variety of factors. First of all, the stepper speed is moderate and ranges from 20 ms per step to 7 ms per step. Higher speeds were found to not work 100% of the time, and as such were not chosen for this system. The conveyor belt speed was originally set using 50% PWM duty cycle, but was found to be too quick for placing objects. A stable and reliable belt speed was found at about ~41% PWM duty cycle, but it should be noted that this value changes from station to station, and should be checked and calibrated prior to sorting. Another factor in stable operation of the system, is the spacing between parts. For the project demonstration, the spacing between the objects was kept to approximately one half of the width of each part. This spacing reduces the risk of mis-sorting at the end of the conveyor.

E- Testing and Calibration

Testing and calibration of the system involved creating another program that determines a range of reflective values for each part (Appendix C). These values are then hard-coded into the main inspection/sorting system code. In order to obtain the object part ranges, the program makes use of the reflective sensor's ADC interrupt, optical sensor interrupt, and an interrupt for one button. The ADC interrupt works in the same manner as in the main system. The optical reflective sensor will start the ADC conversion when the item enters, and when the item leaves the lowest ADC value for that part will be added to the queue. To calibrate the system, multiple of the same object type will go through the sensor and the lowest ADC value for each will be held in the

queue. Upon button press, the ISR for the button will loop through the linked queue and hold the minimum and maximum values in the queue, and will display the minimum value on the LEDs. Since the ADC value is 10 bits, the two most significant bits are displayed on one of the onboard LEDs (green for the first, red for the second, orange for both, and off for none). After a second button press, the maximum value in the queue is displayed. After one object type is calibrated, the program is reset to do the next. This was tested by hard-coding the values into the main program, and checking that the items would properly sort.

In addition to testing and calibrating the objects, stepper speed, conveyor belt speed and positioning of the end of travel sensor were also tested and calibrated. Stepper speed was tested through trial and error using various acceleration and deceleration methods. Once the final stepper speed and acceleration method was determined, two 9V batteries we placed on the tray to check that it can withstand weight and still move reliably. This test was conducted to ensure that the objects falling into the trays would not greatly affect the stepper's movement. The conveyor belt speed and exit sensor position were calibrated together to ensure that items would not stay on the belt after leaving the sensor, but would also not fall into the tray too early. Another factor that contributed to the belt speed, was ensuring that the belt was slow enough to still be able to place the objects, without too much difficulty, while it is moving.

F - Limitations and System Tradeoffs

Some common errors encountered during debugging that limited the system are listed below:

- Incorrect item classification due to reflective sensor misread. This was most prevalent with the plastic pieces because they had similar reflective values.
- Dequeuing an item too early. This would result in the stepper moving the tray before the item being sorted had time to fall into the correct tray.
- Dequeuing an item too late. This would result in the next queued item falling into the previous items bin.

To avoid these issues the items needed sufficient space between them when loaded. This limited our system, since we could only load up to six pieces at a time while maintaining the necessary spacing. To handle the dequeue timing the belt speed, position of the exit sensor, the stepper speed, and the delay before dequeue needed to be adjusted with respect to each other.

Another issue occurred if several interrupts fired simultaneously. To avoid this, the pieces had to be loaded such that there were not pieces at the end of belt sensor and mid-belt sensor at the same time. It was found that loading six pieces at a time, then waiting for them to reach the end

of belt sensor before loading the next six, was an effective way to avoid this error. However, this did limit our sorting speed.

The main tradeoff in this system was between sorting speed and accuracy. If the belt was moving too fast loading issues would occur, which could result in mis-sorting, whereas if the stepper was rotating too fast it could stall.

IV. Conclusions

Overall, this project was successful in meeting the system requirements. As learned later into the project, implementing a state machine for the various object states resulted in a more reliable sorting system. Given more time, the state machine would be added to and revised to handle more cases and allow the system to run more efficiently. Additionally, a re-design of the stepper method to decouple the process from the main loop would have allowed the system to work faster and more reliably. Another improvement that would have been implemented if there was time would be a faster clock cycle (ie. 8 MHz instead of 1 MHz). This, with some major changes to the code, would allow for more interrupts to be handled simultaneously, so that more items could be stacked on the belt.

An improvement to the apparatus/system design that we were unable to alter would be to cover the stepper motor driver with a clear plastic covering, rather than a paper cup. This would allow users to view the LEDs while the system is running without the risk of having objects crash into it. Another recommendation would be to allow users to pre-load the system. This would reduce sorting time significantly, and reduce the risk of having items fall or become poorly spaced. Also, if there was more clearance between the belt and tray the probability of pieces getting jammed in between could be reduced.

To improve lab experience, adding more material regarding interrupt handling and state machines in the labs leading up to the project would give the teams a stronger footing before starting the project. This would allow teams to become more innovative and creative during the design process by decreasing the learning curve that the project presents. Additionally, including more lectures on common programming techniques and practices would be beneficial.

VI. References

[1] B. Mu. MECH 458. Class Lecture, Topic: “Project.” Department of Mechanical Engineering, University of Victoria, British Columbia, CA, Oct. 30, 2017.

[2] Atmel, *8-Bit AVR Microcontroller with 64/128K Bytes of ISP Flash and USB Controller*, Sept. 2012.

[3] B. Mu. MECH 458. *Linked Queue Code.* Department of Mechanical Engineering, University of Victoria, British Columbia, CA. Fall 2017.

[4] Design Spark, “Stepper motors and drives, what is full step, half step and microstepping?” [Online]. Available: https://www.rs-online.com/designspark [Accessed Dec. 02, 2017].

Appendix A - Original Statement of Preliminary System Design

Appendix B - Detailed Technical Documentation: Final Project Code Document

Appendix C - Detailed Technical Documentation: ADC Calibration Code

Appendix D - Detailed Technical Documentation: Linked Queue Code