

Group members: Drew, Chris,Alex H,Alex K

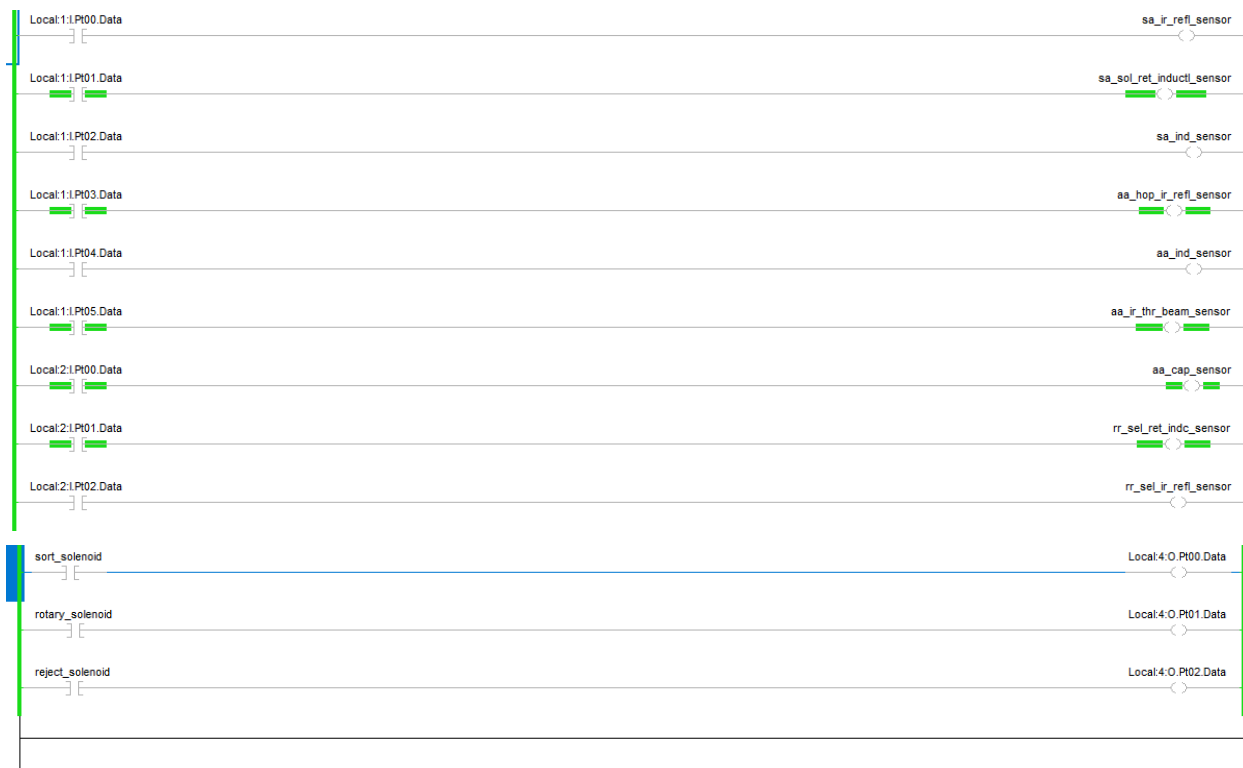
Introduction:

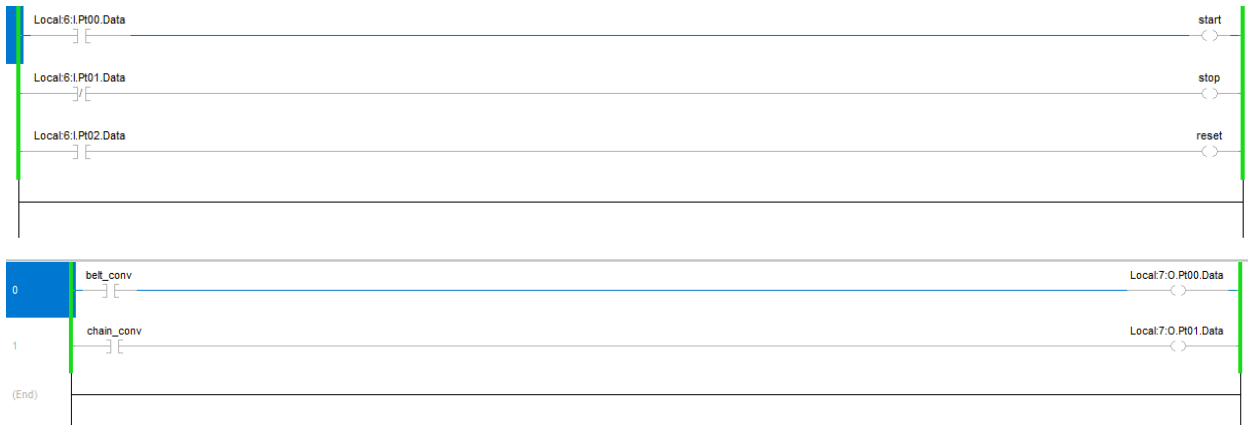
This lab is the first of three to deal with Programmable Logic Controllers, based around the Industrial Control 5069-L320ERS2 COMPACT GUARDLOGIX PLC. We will familiarize ourselves with the software necessary in order to program the PLC, and more broadly the ladder logic on which the software is built. We will first map all the necessary inputs to their respective outputs, then we will build on that by adding sorting logic, kicking plastic pieces off a conveyor belt into a queue while allowing metal pieces safely through. By the end of this lab we will have built a strong foundation, both in understanding and in code, for the next lab.

Mapping of Inputs and Outputs:

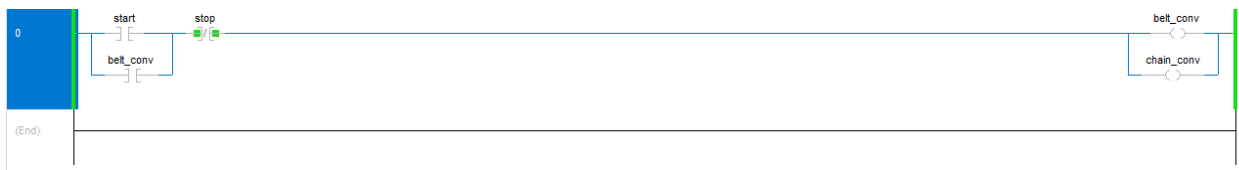
To begin the lab, the team had to first map all of the inputs and outputs to the PLC. The inputs and outputs modeled were in four categories: Standard Inputs, Standard Outputs, Safety Inputs, and Safety Outputs. There were 17 combined inputs and outputs configured in this section. Below are all of the inputs and outputs mapped in the PLC program:

I/O

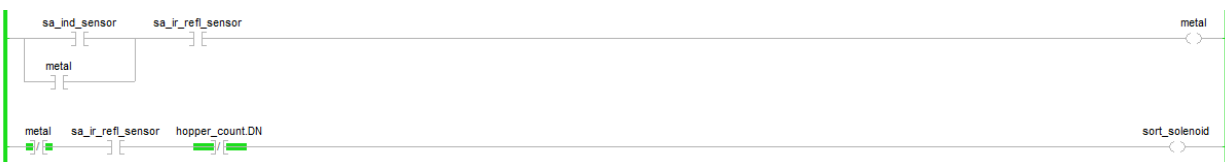




To implement the code that starts and stops the conveyors, we used a similar setup to the latch that was demonstrated in class. We put the start button in parallel with the belt conveyor output, and then put the stop button, which is normally open, in series with those. This means that when the start button is pressed, both conveyors start. When the stop button is pressed, the connection is broken and both of the conveyors will stop running. The code that we used is shown below.



To implement the sorting logic, we again made use of latching. We set the inductive sensor (the sensor that notices metal parts) in parallel with a variable that we defined as metal. The metal variable essentially keeps track of whether or not the part currently passing through the sort area is metal. In series with that is the IR reflective sensor. The IR reflective sensor senses if any part is in the sort area. In function, this makes it so that when a metal part goes through, at the point when the inductive and the IR reflective sensors are both high, the metal variable is latched at high for the duration of the IR reflective sensor signal. On the next rung, an XIO gate for the metal variable is in series with the IR reflective sensor and an XIO gate for the hopper count full bit. This makes it so that if the metal sensor is low (meaning the part in the sort area is not metal), the IR reflective sensor is high (meaning that there is a part in the sort area), and the hopper is not full, the sort solenoid will activate, kicking the part into the hopper chute. Otherwise, the part will continue on and slide down the ramp past the sorting area. Our code implementing our sorting logic is included below.



Overall, through this lab we successfully gained an understanding of Programmable Logic Controllers by working with 5069-L320ERS2 COMPACT GUARDLOGIX PLC and

implementing ladder logic in Studio 5000. We mapped all required inputs and outputs, ensuring proper communication between sensors and actuators, and designed controller logic to operate the conveyor system using latching for start and stop functionality. Additionally, we implemented sorting logic that accurately distinguishes between metal and plastic pieces using inductive and IR sensors, allowing the plastic parts to be diverted while the metal parts continue through the conveyor system. This lab established a solid base in PLC programming, ladder logic design, and system integration, which will be essential for more advanced automation tasks in the future.

Names: Alex H, Drew, Alex K, Chris

Introduction:

After the previous week's lab of mapping the inputs & outputs, stop & starting the chain/belt conveyers, and sorting plastic and metal pegs, the team built onto the control logic. This week in the lab, the team was tasked with implementing two new functions in the control logic. The first was to add on to the plastic and metal sorting from the previous week by implementing a queue size of 5 in the plastic ring holder. Then also, there is a loading area at the end of the holding queue where one plastic ring is set allowing it to assemble to a metal peg once a peg passes through. A ring goes to the loading/assemble area once a rotary solenoid is triggered, the team implements rotary control and activates it only if there are plastic rings available in the queue. The newest topics we utilized for this weeks lab were count up and count down counters, this was used for keeping track of the amount of plastic rings in the holding queue and ensuring the queue doesn't surpass five plastic rings. In order to account for the delays of the placement of various sensors, time off delay timers were utilized too. Our detailed implementation and logic can be seen in the sections below.

Sort Logic:

In this portion of the lab we updated the sorting code that we made in the previous lab by adding an up counter when the sort_solenoid (the solenoid that kicks plastic rings into the hopper) activates. We also added a down counter with the same counter variable to the rotary solenoid. Because this means that there is an up counter at the entrance of the hopper and a down counter at the exit of the hopper, in function this keeps track of how many plastic rings are in the hopper. When we first implemented this, we found that the solenoid would activate, incrementing the hopper_count variable before the solenoid actually kicked. The solenoid would then see that the hopper count was full and not activate, meaning that the hopper maxed out at 4 parts and then the solenoid wouldn't activate again. To fix this, we added a timer to delay the hopper_count increasing until one second after the sort_solenoid activates. This fixes the issue and we tested to make sure the one second delay was short enough that it didn't add any further issues. An image of our completed code with the implemented counters is included below.

Assembly of Pieces and Rotary Solenoid Logic:

In this part of the lab we built off the counter and sort logic devised in the previous section with the intent to implement the counter in our rotary solenoid logic. We want the rotary solenoid to activate only when two simultaneous facts are true: there is no ring currently held in the hopper and there is at least one ring remaining in the queue. The first of these conditions is easy enough to program into the ladder logic, a problem resolved just by adding an XIO condition for the hopper sensor just before the rotary solenoid output. The second condition posed a larger problem, mostly due to the timing issues detailed in the previous section. With an initial condition of 0 rings in queue and no ring in the hopper, we needed the rotary solenoid output to only trigger after that first ring made its way physically to the edge of the hopper. Luckily this issue was resolved when the above-detailed timer was added to the counter. As a

result of our queue timer, the queue was not updated until after the ring had made its way to the hopper physically, and so the queue statement attached to the rotary solenoid output was not true until it was appropriate. Again, the completed code with the output implemented is pictured below with the counter logic.



Overall, through this lab, we successfully expanded on the previous week's control logic by integrating a counter and a timer to automate the sorting and assembly of plastic rings and metal pegs. Up-and-down counters were used effectively to monitor the plastic ring holding queue, ensuring it did not exceed its maximum capacity of 5 rings. During the sorting implementation, we encountered a timing discrepancy in which the hopper incremented before the sort solenoid physically moved the part, but this was resolved by introducing a 1-second time-off delay timer. This timer proved essential during the assembly phase by cascading timing issues to ensure the rotary solenoid activated only when the assembly hopper was completely empty and at least one plastic ring was physically available in the queue. Finally, our strategic implementation of counters, delays, timers, and sensor conditions yielded a fully functional control system that met all requirements.

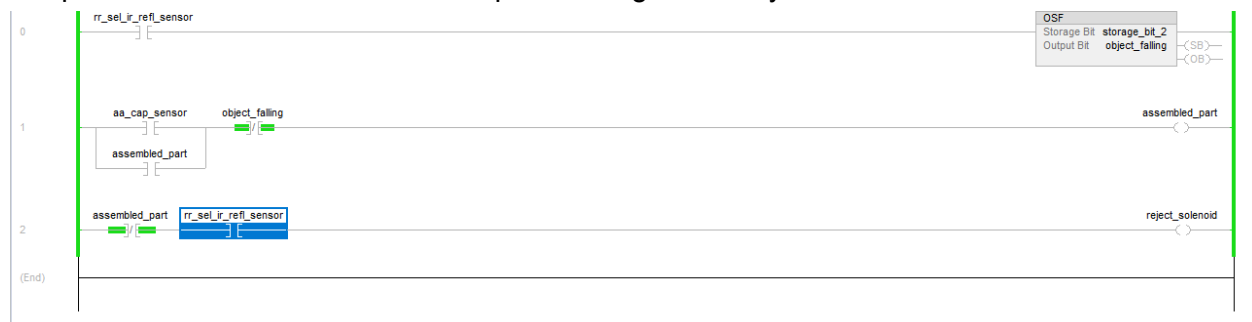
Names: Alex K, Chris C, Alex H, Drew

Introduction:

In this lab we will build off of the work of the previous two labs, and finish off the code for our PLC. Once this lab is finished, we will have a machine that can sort metal and plastic pieces off of the top conveyor; take the plastic pieces from that conveyor and count and load them into the hopper accurately; combine the plastic piece in the hopper with a metal piece; and at the end be able to sense what kind of piece passes by the gate, assembled, metal, or plastic, and dispose of the lone metal and plastic pieces. We will also implement an emergency stop function at the end of the lab, adding logic to our PLC that will immediately turn off the system when the E-Stop button is pressed and not allow it to start back up again until the reset button is pushed.

Reject:

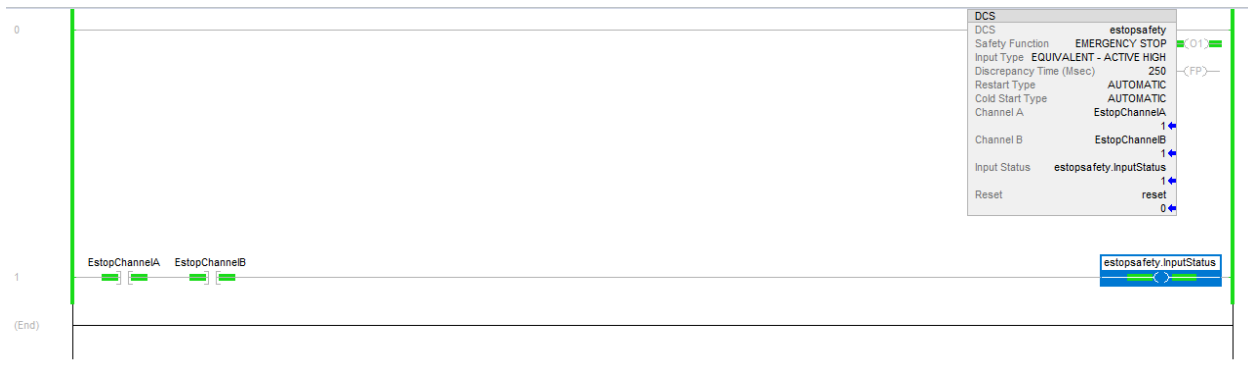
For the reject section of the lab, our group needed to determine whether the part at the end of the conveyor was assembled or unassembled. The main goal was to use a system with an assembly-area sensor to identify whether the object was an unassembled metal peg, an unassembled plastic ring, or a completed metal-and-plastic assembly. Once the object reached the discharge area, the reject-area IR sensor was used to determine when the reject logic should activate. The reject solenoid is supposed to fire only for unassembled pieces, while completed assemblies are allowed to pass through normally.



In our implementation, we used latch-based logic instead of a full FIFO block to keep track of whether the part was assembled. The first rung used a reject-area reflective sensor with an OSF instruction to generate a one-shot signal when an object passed through the reject sensor. This one-shot helped ensure the logic responded only once per part. The next rung used a capacitive sensor and the object detection signal to set the “assembled_part” bit when a completed assembly was detected. Finally, the reject solenoid rung used the reject area sensor in series with the assembled_part condition. If the object reached the reject area and the assembled_part bit was not active, the PLC activated the reject solenoid to push the metal or plastic pieces away. If the assembled_part bit was active, the reject solenoid stayed off, allowing the complete assembly to exit straight.

Emergency Stop:

For the final task of lab 11, the team was tasked with implementing an emergency stop button using a DCS (Dual Channel Stop) block. The DCS block has two input channels and one reset channel, therefore, we had to implement this in the IO setup which can be seen in the figure below. Now that the team was able to implement its IO setup, we used the FP (Fault Present) bit of the E-Stop's DCS block in the conveyor belt start section implemented in a previous lab as a XIO gate. The E-Stop button was latching therefore when it stayed closed, the system can only be started again once the E-Stop button has been reset and the reset button was hit, then the user can hit run and the system will start again. Our team demonstrated this to the TA and all worked well.



All in all, this lab gave very good practice at solving issues related to PLC manufacturing processes using latching and DCS for emergency stops. Our lab group got great practice with coming up with innovative solutions to problems by finding a way to use latching to handle the reject code instead of using FIFO. Furthermore, we got practice ensuring that our ideas will work as expected by testing the different parts to ensure that there was no possible way for using latching instead of using a FIFO block for the reject code to cause a problem with the manufacturing process. When working with the emergency stop, we found that using a DCS block simplified the code greatly, and made it very easy to have the emergency stop button stop whatever functions you need it to stop (in our case, the conveyers) by introducing the FP (fault present) bit in an XIO contact in series with the rest of the conveyer rung.