MANF 486: Mechatronic Systems Laboratory

Lab #3 - PLC organization blocks (OBs)

School of Engineering

Introduction

Organizational blocks are widely referred to as the interface between the plc user program and the operating system. Tasks are written into the OBs using various programming languages like SCL or SFC from which point they can be called by the operating system to execute. OBs can also be used to call other Function Blocks(FBs) or Functions(FCs) allowing for the implementation of more complex algorithms. Siemens provides a variety of OBs each serving a unique purpose and differing in their calling conditions. In this lab, we will be exploring three OBs namely the StartupOB, the Hardware Interrupt OB and the MainOB.

Each OB has a priority ranging from 1 up to 26. An OB with a higher priority is able to interrupt the execution of an OB having a lower priority. Upon the completion of the higher priority OB's task, the CPU will return control to the lower priority OB. This feature allows for error handling and interrupt-driven programming.

The StartupOB, as the name implies, is only executed when the PLC is switched into its run state. It is used for any tasks involved in initialization and cannot be interrupted. It has the highest priority of all OBs, being 26.All interrupts and cyclic processes are executed after the startupOB has completed its execution.

The MainOB is executed on a cyclic basis with a minimum cyclic time of 1 microsec and a maximum of 6 sec. It has a fixed priority of 1 and can thus be interrupted easily by higher-priority OBs. The user program is usually called from OB1

The HardwareInterruptOB has a default priority of 16, but this can be changed depending on the needs of the program. It is triggered by an event such as a change in a digital or analog I/O module.

Solution Description

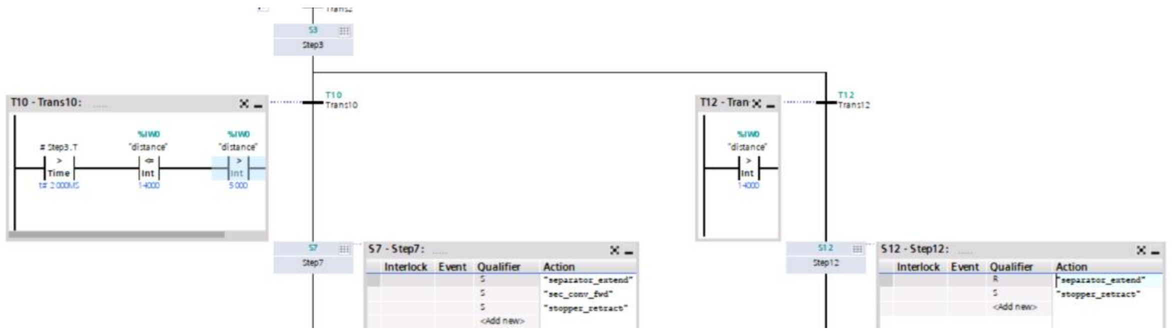
Initialization Task

This task was implemented using a simple step by step algorithm written in Graph. When the PLC start button is pressed, the system transitions to a step where the main conveyor runs, the stopper retracts and the separator arm extends. A timer is used to start up the secondary conveyor 0.5s later and the two conveyor belts are left to run for a total of 5s. During this time any workpiece will be diverted by the separator arm into the secondary conveyor. Given the size of the MPS station, this would allow enough time for any workpiece that was already on the main conveyor to move into the secondary conveyor section.

We did not experience any motor driver failure using this approach. We believe the 0.5s delay in between turning on the motors prevented any large current draw.

Auto-Mode Task

For this task, the threshold must first be determined. After experimenting, we found that a distance value of 14000 corresponded to lidded workpieces whereas values between 5000 and 14000 indicated that the workpiece had no lid. Values below 5000 meant that there was no workpiece under the distance sensor. From here, a branching condition was implemented into a Graph program.



S7 - Step7:

Interlock	Event	Qualifier	Action
		S	"separator_extend"
		S	"sec_conv_fwd"
		S	"stopper_retract"
		<Add new>	

S12 - Step12:

Interlock	Event	Qualifier	Action
		R	"separator_extend"
		S	"stopper_retract"
		<Add new>	

Fig 1. Auto-Mode Task

After measuring the height of the workpiece, the code proceeds into the left branch if there is no lid. In this case, the separator arm is extended to move it into the secondary conveyor. If however there is a lid, the code proceeds into the right branch where the separator arm is not extended thus the workpiece moves to the end of the main conveyor.

Auto-mode should only run when the key is in the horizontal position. A normally open contact is used to implement this as shown in Fig 2. If the key is in the vertical position, the auto-mode block cannot be called in MainOB.

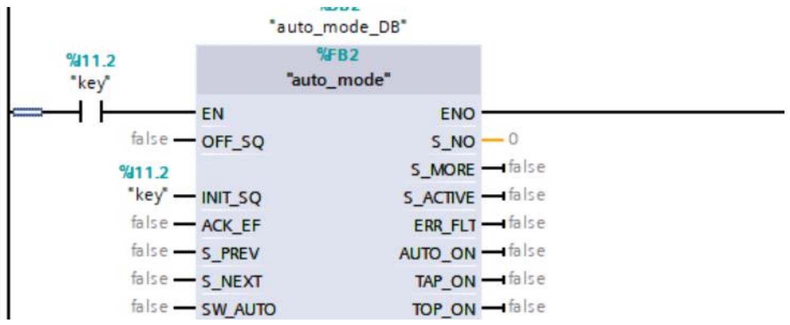


Fig. 2 Auto-mode control using control panel key

Manual Mode Task

There are 4 actuators involved in the MPS separating task. Using the HMI in Fig 3 we can control these actuators by entering either a “1” or “0” into the corresponding HMI input field.

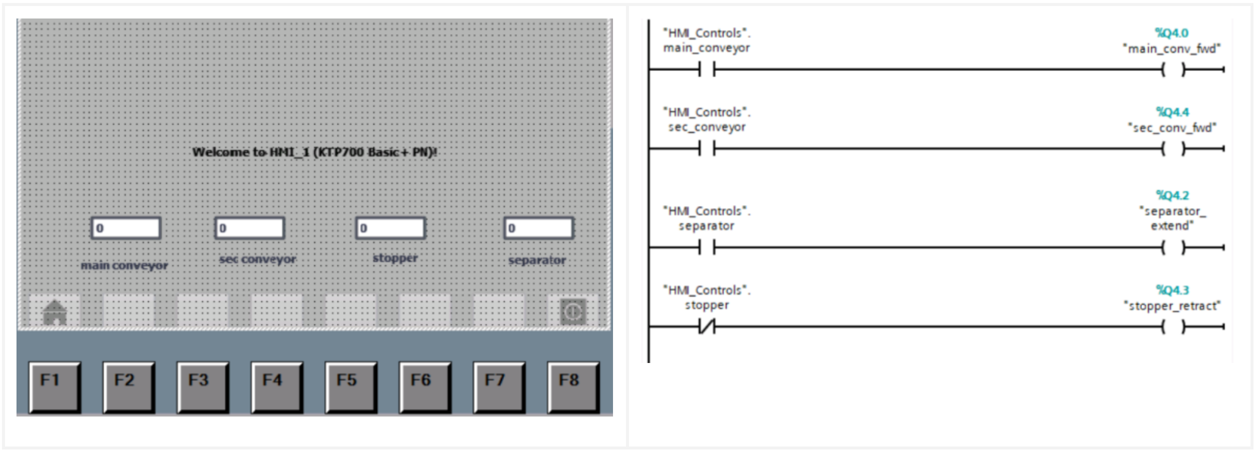


Fig 3. HMI View(left) and HMI controls to PLC controls mapping network(right)

Similarly to the auto mode block, the manual block is only called when the key is in the manual mode(vertical) and this is implemented using the code rung below(Fig 4) within MainOB.

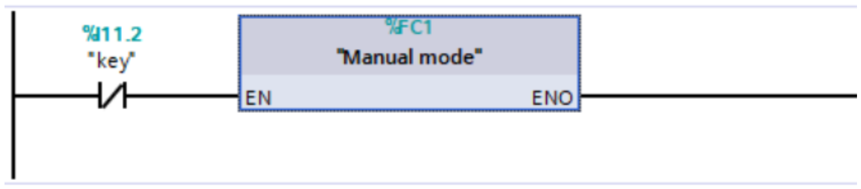


Fig 4. Manual-mode control using control panel key

StartupOB task

To store the number of times the PLC has been switched from STOP to RUN mode, an ADD black is used in the startupOB as shown in Fig 5. The variable “Startup_DB”.START_Cnt is set to “retained” in the datablock to preserve its value between switching cycles.

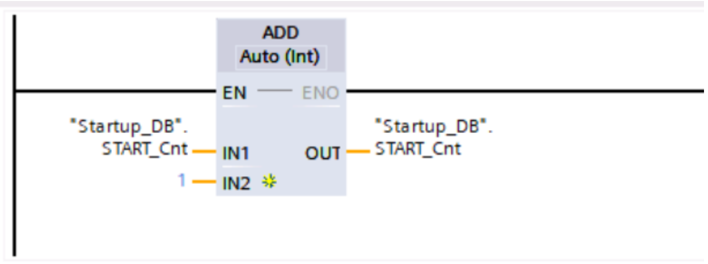


Fig 5. PLC start counter

To keep track of the time since the last startup, the code in Fig 6 was incorporated into startupOB. Each time it ran, the time from the previous call would be moved into the "time_last" variable. The current system time would then be read and placed into "time_Current". Finally, the T_DIFF block was used to calculate the difference between "time_Current" and "time_last", ultimately outputting the result into the "time_Since_Last" variable. "time_Current" was set to retain its value between switching calls.

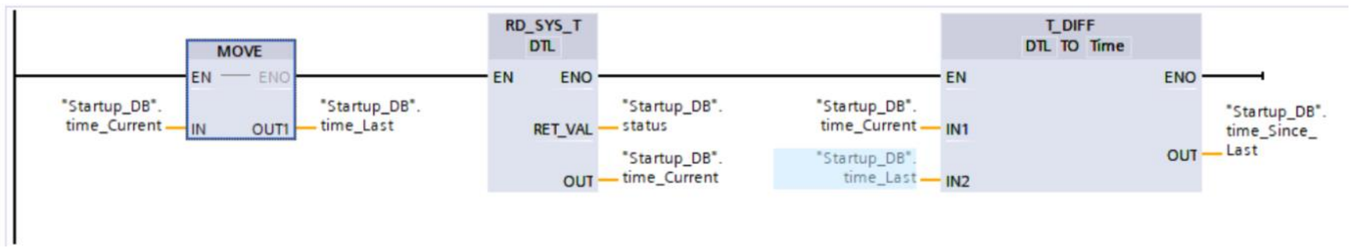
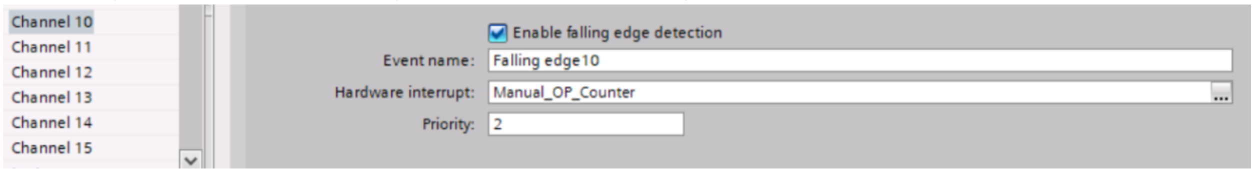


Fig 6. Implementation code for calculating the time since last startup operation

Number of Manual Operations

A hardware interrupt was set to be triggered by a falling edge when the key is switched from auto("key" = 1) to manual("key" = 0) as shown in Fig7.



✓ Enable falling edge detection

Event name: Falling edge10

Hardware interrupt: Manual_OP_Counter

Priority: 2

Fig 7. Hardware interrupt setup

This would call the hardware interrupt block "Manual_OP_Counter" which uses an ADD block to increment a counter variable. This "Manual_OP_Counter" is also set to be retained in the datablock.

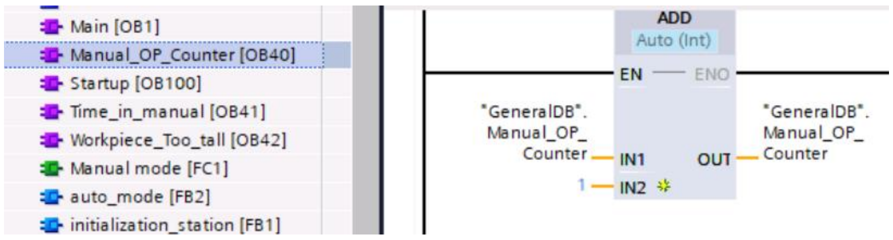
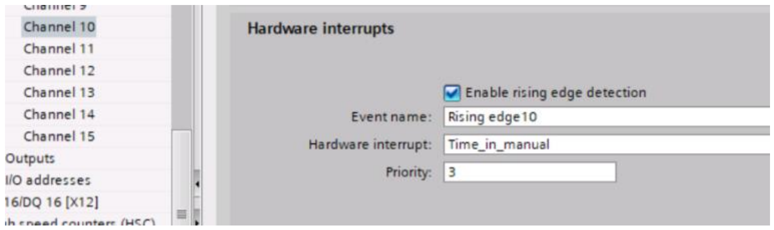


Fig 8. Manual Mode counter using HWI

Calculating the total time in manual mode

A new HWI was set up, triggered by a rising edge on the PLC "key" tag(see Fig 9). This means the MPS station would transition into auto mode. The code keeps track of how long it has been since the station was in auto mode. This time equates to time in manual mode. It then adds these time intervals to get the total time in manual mode.(see Fig10)



Hardware interrupts

☑ Enable rising edge detection

Event name: Rising edge10

Hardware interrupt: Time_in_manual

Priority: 3

Fig 9. Hardware interrupt setup for total time in manual mode

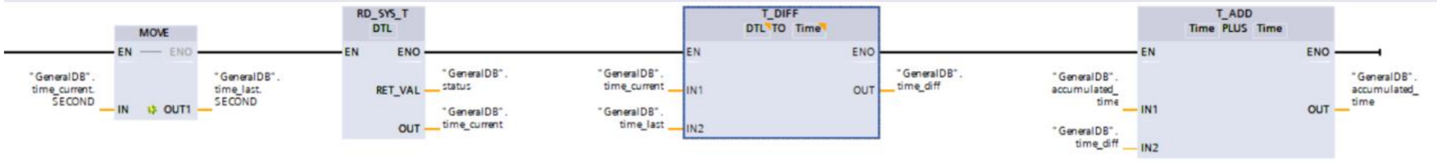


Fig 10. Time in manual mode using HWI

To reinitialize the station, a rising edge detector was used on the key tag. This would set "switch_to_auto" to true, thus resetting the auto_mode block and initializing the "initialization task".

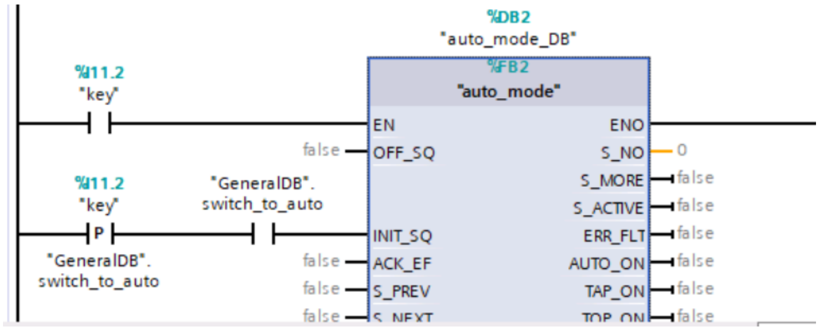


Fig 11. Resetting auto_mode block

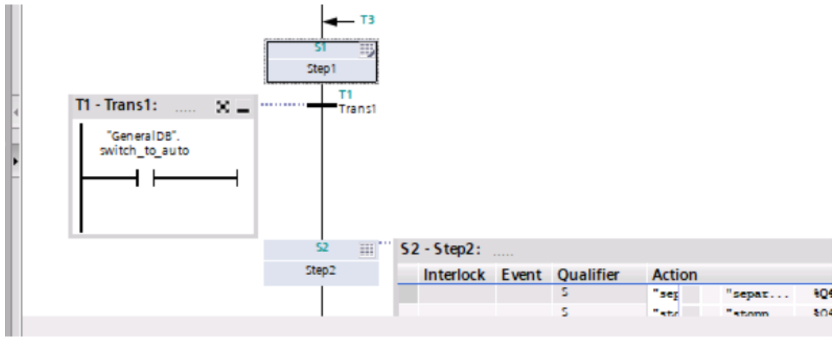


Fig 12. Starting the initialization task using "switch_to_auto" tag

<u>Solution Assessment</u>

In order to test our code, we conducted testing of the requirements given to us at each part of the lab. That is for the initialization program, auto-mode, manual mode, etc. These tests were outlined in the lab manual in the following steps:

1. Put the PLC in RUN mode with workpiece(s) already on the station. Ensure that the workpiece(s) are discarded and the station is clear after initialization.
2. Put the station in Auto mode, press the start button, and send a combination of workpieces with/without lids through the station. Ensure that the separating function works as expected.
3. Put the station in Man mode and use the HMI to turn on/off the various Actuators.
4. Put the station back in Auto mode and make sure that the initialization sequence is run again and Auto mode behaves as expected after the switch.
5. Simulate a large object being detected by the distance sensor (with your finger). Ensure that the large object is backed out of the station and that the station re-initializes after removing the object. Ensure that Auto and Man mode work still.
6. Simulate a large object being detected mid-way through an Auto cycle and ensure that everything works as expected.

All of the following tests were passed by our program. This tells us that the code was able to conduct its key functions properly, however it does not mean that the code is as efficient or bug proof as it could be. With any coding problem, there are a number of different ways to get the desired result. This can be done with different methods, strategies, and code structure.

The first example in this lab where a different method could have been used is in our HMI code. We were tasked to create an HMI which enables manual control of the station's actuators when in manual mode. In the implementation of our code, we chose to change the state of each actuator by entering either a 1 or 0 in the HMI. Although this solution to the task does work, a different and cleaner approach to this problem could have been implementing on/off buttons on the HMI.

The second example of where we could have used a different strategy of coding the station is in the initialization portion of the lab. In order to initialize the machine, we must run both conveyor belts to ensure that no workpieces are present on the station before we proceed with the operation. To do this, we decided to utilize timers to turn the conveyors on for a set amount of time to allow for any workpieces to be cleared. An alternative to this would be to utilize the sensors that are present on the station that are located at the ends of each conveyor. We could monitor the state of the sensors and use them to turn off the conveyors. Although the timer approach worked in this case, the sensor approach does have some advantages such as not having the conveyors run longer than needed. This would be important if it were applied at a larger scale where energy savings are more important.

Another important thing to note when assessing the solution we came up with for the lab, is the serviceability of the code. In industry, there are always going to be different people viewing, editing, and altering the code of an automation system. Therefore, it is very important that we design code to be clear, understandable, and organized. We did our best to adhere to these principals while designing our code however it could be improved in a few ways. First, we could have labeled all of our blocks and variables in a more clear way that can easily be tracked back to the hardware on the physical station. Next, we could have created separate networks for some of the station functionality rather than combining some of them. This would make the code easier to follow and allow edits to be maybe easily and in an organized way.

Conclusion

In this lab, we have successfully explored, implemented, and evaluated the functionality of StartupOB, MainOB, and Hardware Interrupt OB in a PLC program designed for an MPS separating station. The lab provided a comprehensive understanding of the application of organizational blocks in PLC programming and their role in creating efficient and effective automation systems. We were able to demonstrate the ability to properly initialize the station, handle auto-mode and manual-mode tasks, and utilize hardware interrupts for counting manual operations and calculating time spent in manual mode. Our program was able to meet the lab requirements, successfully passing all outlined tests.

Although our program has been effective in meeting the primary objectives, it is important to recognize that there is always room for improvement in code design, structure, and serviceability. Throughout the lab report, we have discussed alternative approaches to some tasks and suggested ways to enhance the code's readability and maintainability. By considering these improvements, future automation projects can benefit from more efficient, scalable, and serviceable code that meets industry standards and expectations.

The knowledge and skills gained during this lab have broad applications in the field of industrial automation and control systems. Understanding the nuances of PLC programming and the use of organizational blocks is essential for designing and implementing complex automation systems in various industries such as manufacturing, logistics, process control, and more. The lab experience has provided valuable insights into the practical application of PLC programming concepts, which can be leveraged to create more efficient, reliable, and scalable automation solutions in the future