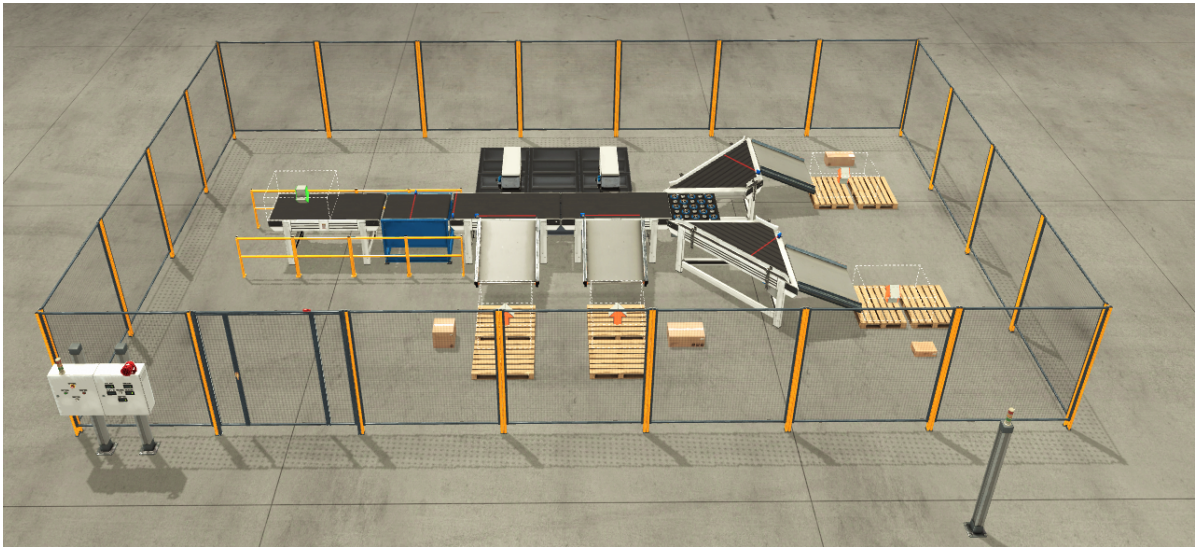


Applied Programmable Logic Controllers – ACDA6000 Factory Control



Jayden Fox – 2316329

Table of Contents

Abstract	4
1.0 Introduction.....	5
2.0 Task 1: Literature Review of PLCs	6
2.1 Introduction to PLCs	6
2.2 PLCs commonly used in industry	7
3.0 Task 2: Factory Design using Factory I/O	9
3.1 Items used in Factory Design.....	9
3.2 Factory Build Breakdown.....	10
4.0 Task 3: Ladder Logic Program	13
4.1 Program Structure	13
4.2 System Operation	14
4.3 Program Breakdown	15
4.3.1 Sorting Line Operation.....	15
4.3.2 Box Tracking	23
4.3.3 Alarms Errors Warnings.....	26
4.3.4 HMI Signals	32
5.0 Task 4: HMI Integration	33
6.0 Discussion/Evaluation.....	37
6.1 Recommendations for future work	38
7.0 Conclusion	39
8.0 References	40
Appendix.....	42
Appendix A – Program/Factory IO information.....	42
Appendix B – Full program breakdown	45

Table of Figures

Figure 1 – Factory IO Sorting Line Design.	10
Figure 2 – Close up of box emitter and conveyor scale.	11
Figure 3 – Close up of pushers (1&2).	11
Figure 4 – Close up of pop-up wheel sorter.	12
Figure 5 – Control Panels.	12
Figure 6 - Program Structure/Directory Tree.	13
Figure 7 – Flowchart of system operation	14
Figure 8 – ‘BoxWeightValues’ data block	15
Figure 9 – ‘SortingDB’ data block.	15
Figure 10 – Network 1 of ‘SortingFC’	16
Figure 11 – Network 3 of ‘SortingFC’	17
Figure 12 – Networks 5&6 of ‘SortingFC’	18
Figure 13 – Network 7 of ‘SortingFC’	19
Figure 14 – Network 8 of ‘SortingFC’	19
Figure 15 – Network 9 of ‘Sorting FC’	20
Figure 16 – Network 10 & 11 of ‘SortingFC’	21
Figure 17 – Networks 12 & 13 of ‘SortingFC’	22
Figure 18 – ‘CountingDB’ Data block.	23
Figure 19 – Network 1 of ‘BoxTrackingFC’	23
Figure 20 – Networks 2 & 3 of ‘BoxTrackingFC’	24
Figure 21 – Networks 4 to 7 in ‘BoxTrackingFC’	25
Figure 22 – Box not arrived error ladder logic.	28
Figure 23 – Pusher not extending error logic	29
Figure 24 – Pusher extended error	30
Figure 25 – Box area full error	31
Figure 26 – ‘HMI Signals’ data block.	32
Figure 27 – HMI Main Screen	33
Figure 28 – Error present and line stopped, user prompt to check errors page	34
Figure 29 – HMI Errors Screen.	35
Figure 30 – Login required to access engineer’s screen.	36
Figure 31 – Engineer’s Screen.	36
Figure 32 – Factory IO box weight values [22].	42

Abstract

This report presents the design, development, and implementation of a fully automated parcel sorting system using Factory and the Siemens TIA Portal V16. A literature review is carried out to give an overview of programmable logic controllers (PLCs) and to give a comparison of leading PLC brands, which ends with the selection of a Siemens S7-1200 PLC for this factory control application. A complete factory layout was then designed to sort four parcels of different weights and sizes through the use of conveyors, a pop-up wheel sorter, pushers, sensors and a control panel. A structured ladder logic program was developed to manage the operation of the system, including a staged conveyor start up, weight-based routing of packages, box tracking and error detection. An HMI was then created to provide real-time monitoring of the system, along with operator control and a password protected screen for engineer level configuration. The system created successfully demonstrates a reliable sorting, robust fault handling and clear operator feedback from the HMI. The report concludes with an evaluation of the system performance and recommendations for future work to improve the accuracy of the system and its similarity to real world industrial practices.

1.0 Introduction

The purpose of this report is to discuss the stages involved to create an automated package sorting system simulation, using Factory I/O and the Siemens TIA Portal. PLCs are seen as the brain of an automated system, and many modern industries count on PLCs for reliable and safe automated control of process operations.

This report begins with a literature review of the current PLC technologies used in industrial applications worldwide. The literature review gives an overview of PLCs, and compares the features and benefits of the different PLC brands in different scenarios and applications. Following the literature review, the report describes the design of the factory I/O layout that is capable of sorting different types of packages through the use of conveyors, pushers and sensors. The ladder logic programming language is then used to effectively manage the operations of the factory.

The main objective of this project is to develop a functioning simulation that can automatically sort four types of parcels of different sizes and weights (palletizing, small, medium and large) into dedicated areas. This system should work with minimal intervention from operators, to show how PLC programming, simulation and the integration of an HMI can be combined to create realistic automated package sorting system simulation.

2.0 Task 1: Literature Review of PLCs

This section of the report focuses on a literature review of Programmable Logic Controllers (PLCs) and the several available types currently used in industrial applications.

2.1 Introduction to PLCs

PLCs are specialised industrial computers designed to control machinery and processes in real time [1, 2]. They were first introduced in the late 1960s to replace large, complex and expensive panels of relays, timers and counters [3, 4, 5]. Since PLCs were introduced, they have become the foundation of the majority of modern automation systems due to their robustness, flexibility and ability to connect with many other devices such as Internet of Things (IoT) sensors, actuators and Human-Machine Interfaces (HMIs) [4].

PLC programming is standardised under IEC 61131-3, which defines the main languages used across industrial controllers [6]. According to the existing literature, ladder logic is the most commonly used as it is a graphical programming language, and is seen as the easiest to read and maintain [7]. Additionally, structured text is a high-level language which is more suited to complex calculations and data processing [8]. Function block diagrams is another graphical language that is frequently used due to its easy-to-understand block-based structure [9]. These languages are supported by all of the major PLC manufacturers, such as Siemens, Allen-Bradley and Hitachi.

Despite providing an effective solution to automation applications, challenges remain in PLCs regarding cyber security as they are frequent targets of industrial cyberattacks [10]. As an example, most PLCs contain an embedded real-time operating system, usually stored in the PLC's internal memory. If the operating system of a PLC is compromised by a hacker, any devices controlled by the PLC can be completely taken over [11]. There are also concerns with malware and ransomware from external devices and networks [12, 13].

Furthermore, the existing literature showed that the integration of artificial intelligence (AI) is changing industrial automation. AI-enabled PLCs have shown to improve decision-making, support predictive maintenance and also hold the ability to detect anomalies in real-time, making control systems more flexible and autonomous [14].

Overall, PLCs in smart industrial control systems have evolved from isolated controllers to networked, intelligent nodes and the discussions surrounding AI show they are still improving and will continue to be relevant in automation activities and applications.

2.2 PLCs commonly used in industry

Regarding PLCs currently used in the automation industry, Siemens is a global leader in automation and their S7 series is widely used in manufacturing and process industries. These PLCs provide integrated safety features, strong networking capabilities and compatibility with the TIA portal software [15]. In addition to these features, Siemens PLCs are highly reliable, whilst also offering advanced diagnostics and strong global support [16]. They also support a wide range of industrial communication protocols such as PROFINET, PROFIBUS and MODBUS TCP, and are an extremely competent automation control system option [16].

Secondly, Allen Bradley PLCs are also shown to be a popular automation control choice. The ControlLogix and CompactLogix series PLCs are tightly integrated with the Studio 5000 development environment, and are known for their reliability, modular design and strong support for large, distributed control systems [17, 18]. Allen Bradley PLCs use Ethernet/IP as their primary communication protocol, but are quite limited in the communications protocols they support [19].

Hitachi PLCs, such as the Micro-EHV+ Series, are commonly used in smaller automation applications. They offer reliable performance and support the programming languages mentioned under IEC 61131-3 [20]. Although Hitachi PLCs are seen as cost-effective, they do not offer the advanced diagnostic features, integrated development environments or worldwide support networks seen from Siemens and Allen-Bradley. As a result of this, they are less common in industrial automation.

Table 1 has been created to display the advantages and disadvantages of the PLC brands mentioned above. Based on the review of existing literature, a Siemens S7-1200 PLC will be used as its standout features made it best suited to the factory control application.

Table 1 – Advantages and disadvantages of mentioned PLC brands.

PLC Brand	Pros	Cons
Siemens	• Highly reliable and scalable. • Integrates with high quality development environment. • Strong global support and documentation. • Advanced diagnostics. • Wide communication protocol support. • Suited to majority of automation applications.	• Harder to use for beginners. • High initial cost.
Allen-Bradley	• Reliable hardware • Excellent support for EtherNet/IP networks. • Studio 5000 development environment is powerful for large systems.	• Limited protocol support compared to Siemens. • Costly licensing for hardware and software.
Hitachi	• Cost-effective. • Better for smaller automation applications. • Supports IEC 61131-3 languages. • Simple and reliable hardware.	• Limited advanced features. • Smaller support network. • Less suitable for large/complex automation. • Fewer diagnostics capabilities.

3.0 Task 2: Factory Design using Factory I/O

To aid this project, a factory simulation was created in Factory I/O that would be able to sort boxes of different sizes and weights.

3.1 Items used in Factory Design

This section will contain a list of the parts used in Factory I/O to create the factory simulation.

Sorting Line Process

- 1 x Emitter
- 4 x Remover
- 3 x Belt Conveyer (2m)
- 1 x Conveyor Scale
- 6 x Diffuse Sensor
- 2 x Pusher
- 4 x Chute Conveyor Low
- 1 x Pop Up Wheel Sorter
- 2 x Straight Spur Conveyer
- 2 x Stack Light (start and end of line)

Control Panels (start of line)

- 2 x Electric Switchboard
- 1 x Emergency Stop
- 1 x Start Push Button
- 1 x Stop Push Button
- 2 x Reset Push Button (1 for errors on sorting line, 1 for digital displays/counts)
- 1 x Stack Light
- 5 x Digital Display
- 1 x Alarm Siren

Other

- 8 x Pallet (after chute conveyors)
- 1 x Stack Light (end of sorting line).
- 3 x Column (2 for electric switchboards, 1 for end of line stack light)
- 3 x Platform (for pushers)
- 4 x Handrail
- 21 x Safeguard (L)
- 1 x Safety Door

3.2 Factory Build Breakdown

The system, shown in Figure 1, can sort four types of parcels of different sizes and weights, and works from left to right. The emitter (green arrow) will automatically drop parcels on to the sorting line, and removers (orange arrow) have also been placed on the end of each exit chute. The conveyor scale (Figure 2) at the start of the line weighs the parcel and the program will decide how to sort the parcel based on the weight value received. The system is able to deal with any type of parcel randomly and can alternate its sorting sequence to correctly organise the parcels. In Table 2, the different Factory IO box types and their weights are presented.

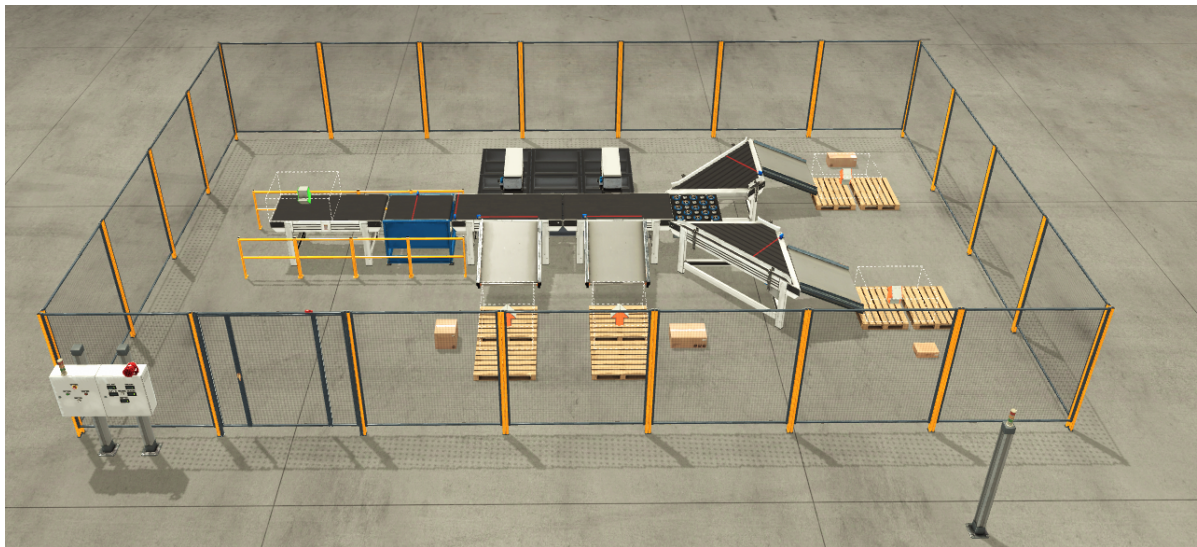


Figure 1 – Factory IO Sorting Line Design.

Table 2 – Parcel box types and their weights.

Parcel Box Type	Weight (kg)
Palletizing Box	3
Small Box	8
Medium Box	10
Large Box	15

The system sorts large parcels first using pusher 1, and medium sized parcels being sorted by pusher 2 (Figure 3). The small and palletizing boxes are sent to the pop-up wheel sorter, with the small boxes being directed left and the palletizing boxes being directed to the right (Figure 4). Each sorting area is equipped with a diffuse sensor to allow for the ability to count the number of sorted parcels in each area.

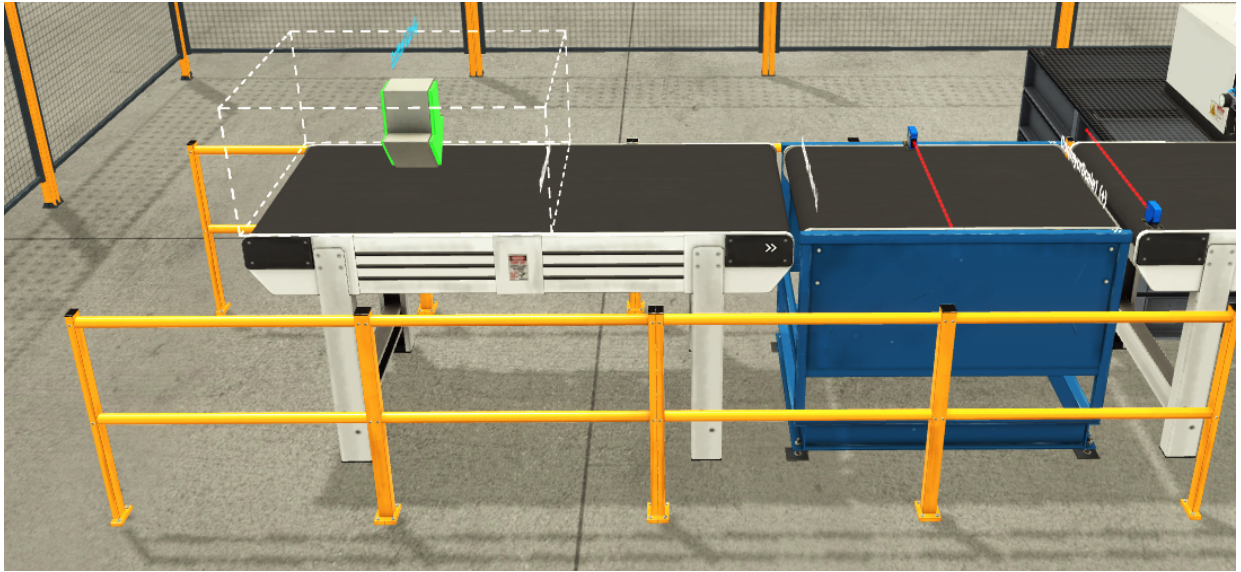


Figure 2 – Close up of box emitter and conveyor scale.

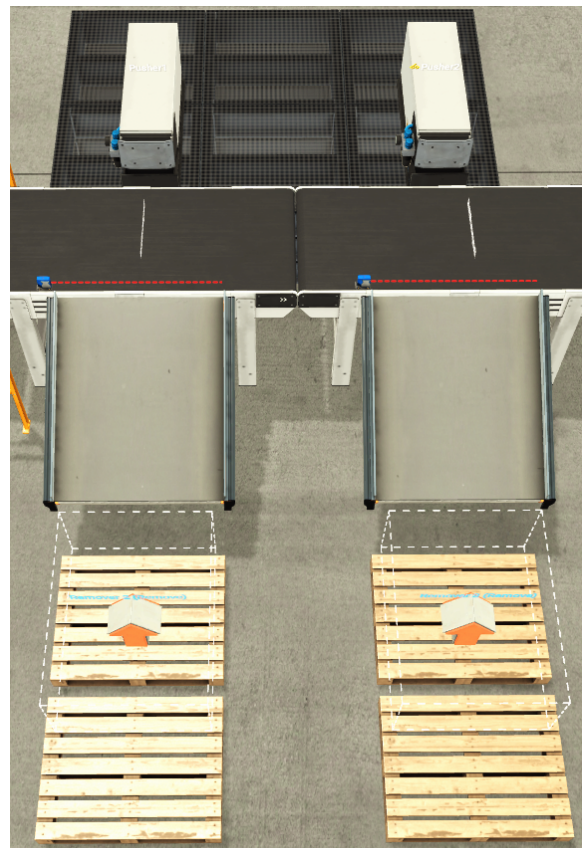


Figure 3 – Close up of pushers (1&2).

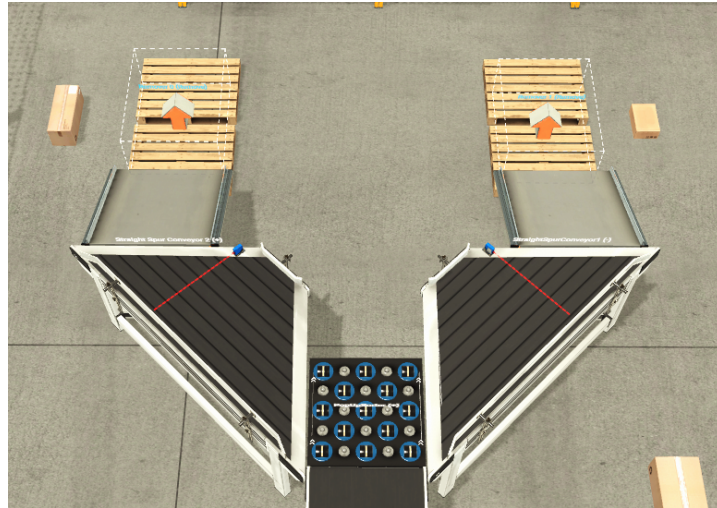


Figure 4 – Close up of pop-up wheel sorter.

The system can be operated using the two control panels at the start of the line (Figure 5). The first panel offers a start button (green), a stop button (red), an emergency stop button (red/yellow) and also a reset button (white) after errors on the line have been corrected. The second panel has five digital displays, which display the weight of the box currently on the line and the amount of each box type that has been transported to the specific sorting area. This panel also has a white reset button to reset the count of the sorted boxes.



Figure 5 – Control Panels.

4.0 Task 3: Ladder Logic Program

A ladder logic program for the Factory IO setup was created in TIA Portal V16. The S7-PLCSIM V16 was used to connect the TIA program with the factory IO simulation. All tags used in the program can be found in Table 6 and Table 7 of Appendix A.

4.1 Program Structure

All data blocks and function blocks containing ladder logic programs were organised into program groups to keep the system easy to navigate through (Figure). There is a main folder for all of these program groups called 'Sorting Line', which then had four subfolders named:

- 1 – SortingLineOperation
- 2 – BoxTracking
- 3 – AlarmsErrorsWarnings
- 4 – HMISignals

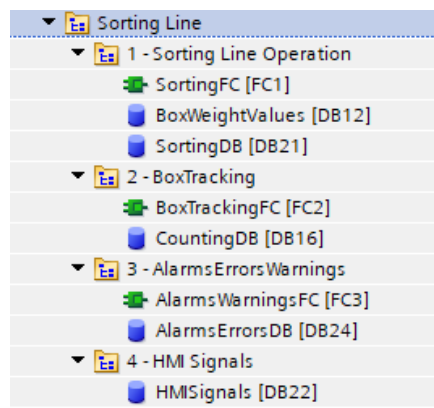


Figure 6 - Program Structure/Directory Tree.

Data blocks are used throughout the program for data storage, as opposed to M memory/tables. M memory is also known as bit memory and is a global, internal memory area used for temporary storage and status flags [15]. M memory tables are a flexible memory space where bits, bytes, and words can be defined. An example of where M memory could be used is to show whether a motor is currently running, where it will have an M address like M10.0 and a unique tag like 'Motor_Running'. On the other hand, data blocks offer a similar way to store data however they allow tags to be more structured and organised. A global data block can be set up for any instance and used throughout the code [21]. As an example, any data related to the sorting line's operation is stored in the 'SortingDB' data block as a part of the Sorting Line Operation program group. An example of how a tag would be display if it is a part of a data block is "SortingDB".BoxInTransit. The use of a data block allows related data to be stored in the same place and makes programs easier to understand and navigate through. Overall, M memory tables offer great features for data storage however, they can make tags difficult and harder to find. Whereas data blocks can be named and organised to eradicate this issue whilst still offering the data storage abilities of the M memory. Therefore, data blocks are preferred in this program.

4.2 System Operation

The following flowchart has been created to illustrate the system's operation. The flowchart considers the sorting and counting of each box type, along with describing how the system will deal with alarms and errors.

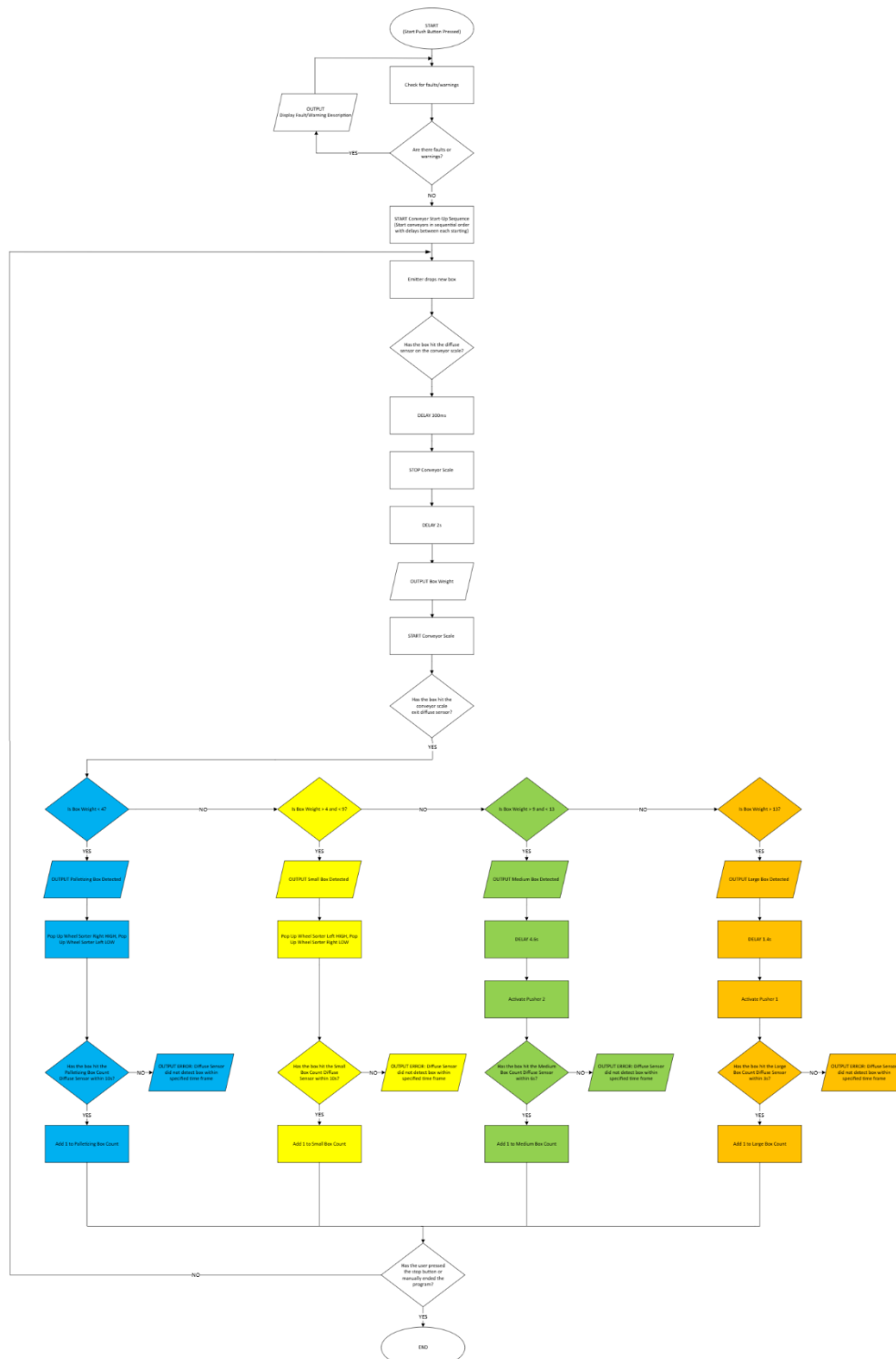


Figure 7 – Flowchart of system operation

4.3 Program Breakdown

This section will breakdown each part of the program to show the key logic, tags and data blocks used to run the system. A brief explanation of the key networks in this program will be provided, with an in-depth description of the whole program provided in Appendix B.

4.3.1 Sorting Line Operation

The '1-SortingLineOperation' folder holds the general program for operating the sorting line, and also contains the comparison code to correctly sort the boxes based on the weight. This folder also contains two data blocks to support the ladder logic code, called 'BoxWeightValues' and 'SortingDB'.

The 'BoxWeightValues' data block (Figure 8) contains tags to allow for scaling the weight of the box and to show which type of box has been detected based on the weight received from the conveyor scale.

BoxWeightValues									
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static								
2	RawWeight	Real	0.0		☒	☒	☒		Raw value from conveyer scale
3	ScaledWeight	Real	0.0		☒	☒	☒		Multiplied by 2 to give actual box weight
4	RoundedWeight	DInt	0		☒	☒	☒		For Weight LCD
5	PalletizingBoxDetected	Bool	false		☒	☒	☒		For code to sort palletizing box
6	SmallBoxDetected	Bool	false		☒	☒	☒		For code to sort small box
7	MediumBoxDetected	Bool	false		☒	☒	☒		For code to sort medium box
8	LargeBoxDetected	Bool	false		☒	☒	☒		For code to sort large box

Figure 8 – 'BoxWeightValues' data block

The 'SortingDB' data block (Figure 9) contains general tags to operate the sorting line and also contains tags to support the comparison code to correctly sort the boxes.

SortingDB									
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static								
2	Operating	Bool	false		☒	☒	☒		
3	ScaleControl	Bool	false		☒	☒	☒		
4	BoxLoaded	Bool	false		☒	☒	☒		
5	Ready	Bool	false		☒	☒	☒		
6	ErrorLight	Bool	false		☒	☒	☒		
7	PBoxInTransit	Bool	false		☒	☒	☒		
8	SBoxInTransit	Bool	false		☒	☒	☒		
9	MBoxInTransit	Bool	false		☒	☒	☒		
10	LBoxInTransit	Bool	false		☒	☒	☒		
11	PBoxArrived	Bool	false		☒	☒	☒		
12	SBoxArrived	Bool	false		☒	☒	☒		
13	MBoxArrived	Bool	false		☒	☒	☒		
14	LBoxArrived	Bool	false		☒	☒	☒		

Figure 9 – 'SortingDB' data block.

The 'SortingFC' function block in this folder holds the ladder logic to operate the sorting line. Network 1 (Figure 10) controls the 'Operating' status and uses a latch so the line keeps running without the need to hold the start button down. The program will only run if the pre-run conditions of the factory are met. This means there must be no errors on the line (such as pushers starting fully extended etc.) and the safety door to access the sorting line must also be closed. A description of each tag used in network 1 has been included in Table 3.

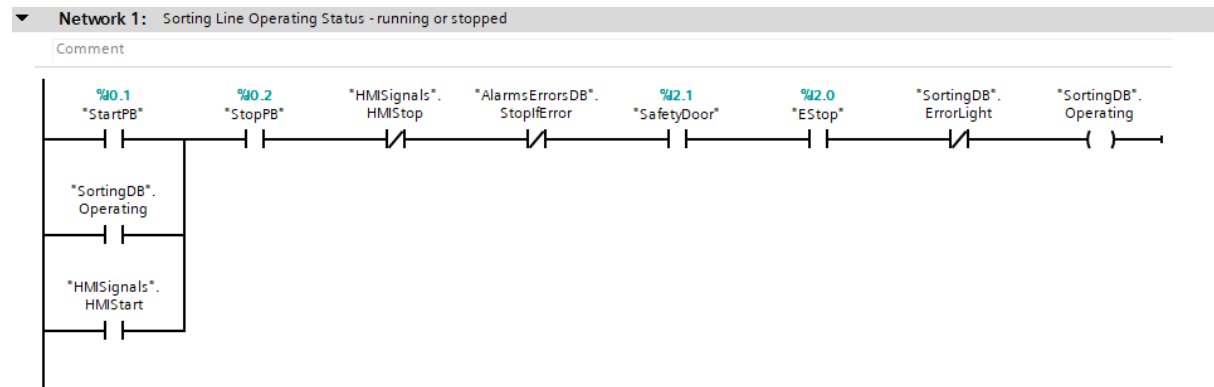


Figure 10 – Network 1 of 'SortingFC'

Table 3 – Network 1 tags description

Tag Name	Purpose
StartPB	Linked to the start push button on the control panels
StopPB	Linked to the stop push button on the control panels
HMStart	Start Button on the HMI
HMStop	Stop Button on the HMI
"AlarmsErrorsDB".StopIfError	Does not allow the line to operate if errors are present
SafetyDoor	Does not allow the line to operate if the safety door is open
EStop	Does not allow the line to operate if the E-Stop has been pressed in.

Network 3 of SortingFC – Staged conveyor start

Network 3 (shown in Figure 11) handles the logic to limit the current draw when the sorting line starts up.

- Purpose: Start belts and spurs sequentially using 1 second TON delays.
- Inputs: SortingDB.Operating = 1.
- Outputs: BeltConveyor1, BeltConveyor2, BeltConveyor3, PopUpSorter, StraightSpurConveyor1(-) and StraightSpurConveyor2(+) enabled after completion of prior TON timer.
- Further comments: 1 second TON chain to limit the current draw at start up.

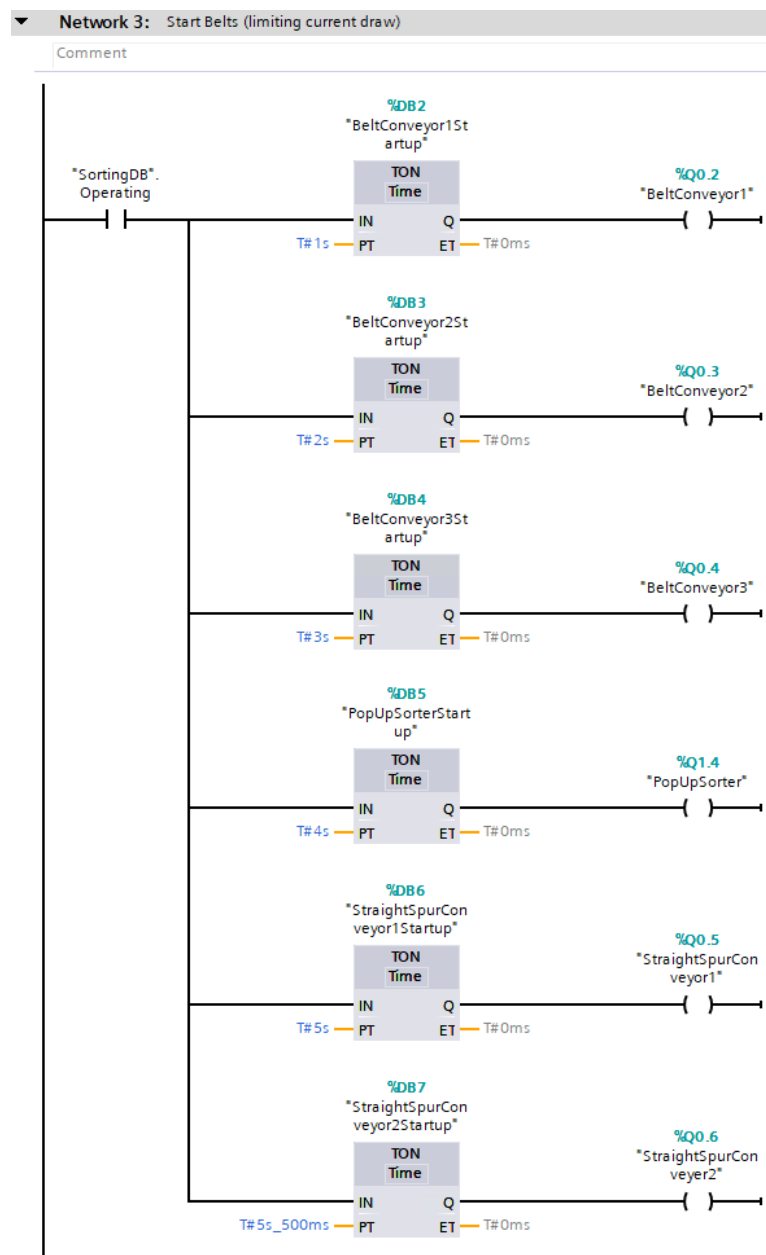


Figure 11 – Network 3 of 'SortingFC'

Network 5 of SortingFC – Conveyor Scale and weighing window

Network 5 of the 'SortingFC' function block (shown in Figure 12) handles the box-weighing sequence and includes two TON timers (i.e. StabiliseBoxTimer (DB8) & BoxWeighingTimer (DB9)).

- Purpose: Provide a stable time window to read the weight of a box.
- Inputs: SortingDB.BoxLoaded = 1, SortingDB.ScaleControl = 1.
- Output: SortingDB.Ready = 1 during window.
- Further comments: StabiliseBoxTimer (0.3s) raises 'Ready' tag, BoxWeighingTimer (2s) then resets 'Ready' and 'ScaleControl' tags.

Network 6 of SortingFC – Box detection hold

Network 6 (shown in Figure 12) detects when a box enters in the conveyor scale through the use of a diffuse sensor.

- Purpose: Set SortingDB.BoxLoaded while boxers travels along sorting line, used to prevent errors from potential sensor flickers.
- Inputs: CSDiffuseSensor (on centre of conveyor scale).
- Outputs: SortingDB.BoxLoaded = 1 held by TOF timer.
- Further comments: BoxDetectionHoldTimer (TOF) keeps BoxLoaded TRUE for 5s after diffuse sensor drops.

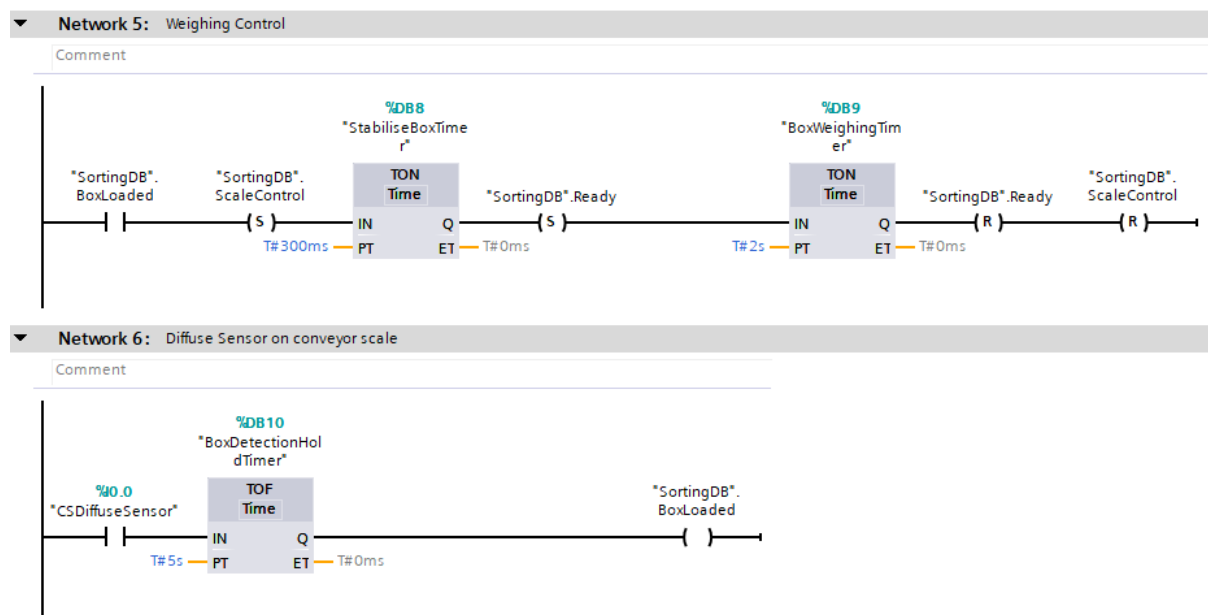


Figure 12 – Networks 5&6 of 'SortingFC'

Network 7 of SortingFC – Weight scaling

Network 7 of 'SortingFC' (shown in Figure 13) handles the logic to scale the initial value received from the conveyor scale to reflect the weight values of each box (seen in Figure 32).

- Purpose: Convert raw scale value to ScaledWeight
- Inputs: SortingDB.Ready = 1.
- Outputs: BoxWeightValues.RawWeight multiplied by 2 to give a scaled weight.
- Further comments: Scaling aligns with Factory I/O parcel weights.

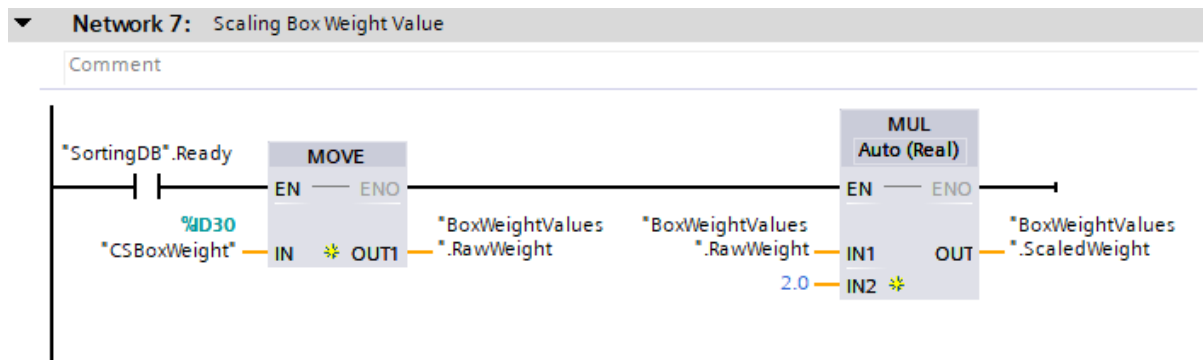


Figure 13 – Network 7 of 'SortingFC'

Network 8 of SortingFC – Sorting of palletizing boxes

Network 8 holds the logic used to sort the palletizing boxes (shown in Figure 14).

- Purpose: Detect palletizing weight boxes and send to the right end area through use of pop-up sorter.
- Inputs: CSExitDiffuseSensor = 1, comparator ScaledWeight $\leq$ 4.0.
- Outputs: PalletizingBoxDetected = 1, PBoxInTransit = 1, PopUpSorterRight = 1.
- Further comments: Transit flag supports arrival-timeout error handling.

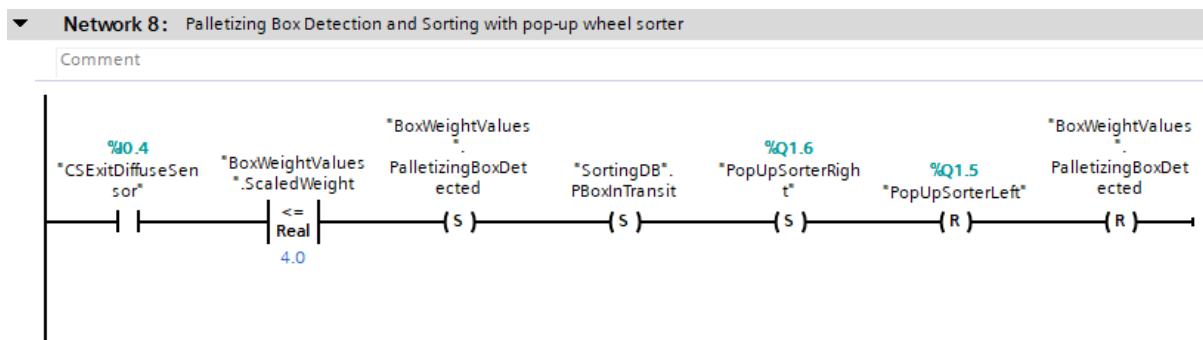


Figure 14 – Network 8 of 'SortingFC'

Network 9 of Sorting FC – Sorting of small boxes

Network 9 contains the logic to sort small boxes (shown in Figure 15).

- Purpose: Detect small weight boxes and send to the left end area through pop-up sorter.
- Inputs: CSExitDiffuseSensor = 1, ScaledWeight > 4.0 and < 9.0.
- Outputs: SmallBoxDetected = 1, SBoxInTransit = 1, PopUpSorterLeft = 1.
- Further comments: Comparison range selected to separate small boxes appropriately.

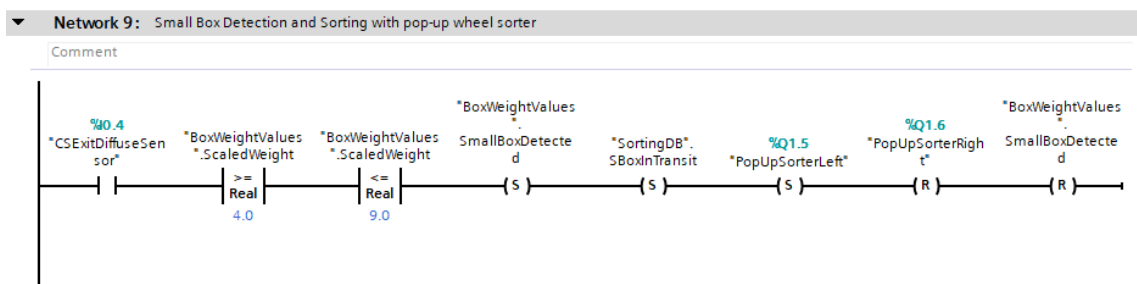


Figure 15 – Network 9 of 'Sorting FC'

Networks 10 to 11 of SortingFC – Sorting of medium boxes

In regard to the sorting of medium boxes, networks 10 and 11 (shown in Figure 16) handle the logic to do this.

- Purpose: Detect medium weight boxes and time pusher activation window.
- Inputs: ScaledWeight ≥ 9.0 and < 13.0 .
- Outputs: MediumBoxDetected = 1, MBoxInTransit = 1, Pusher 2 = 1 after MBoxPusherTimer.
- Further comments: MBoxPusherTimer activates pusher when the box is expected to be exactly in front of it, MediumBoxDetected gets reset after pushing ready for next box.

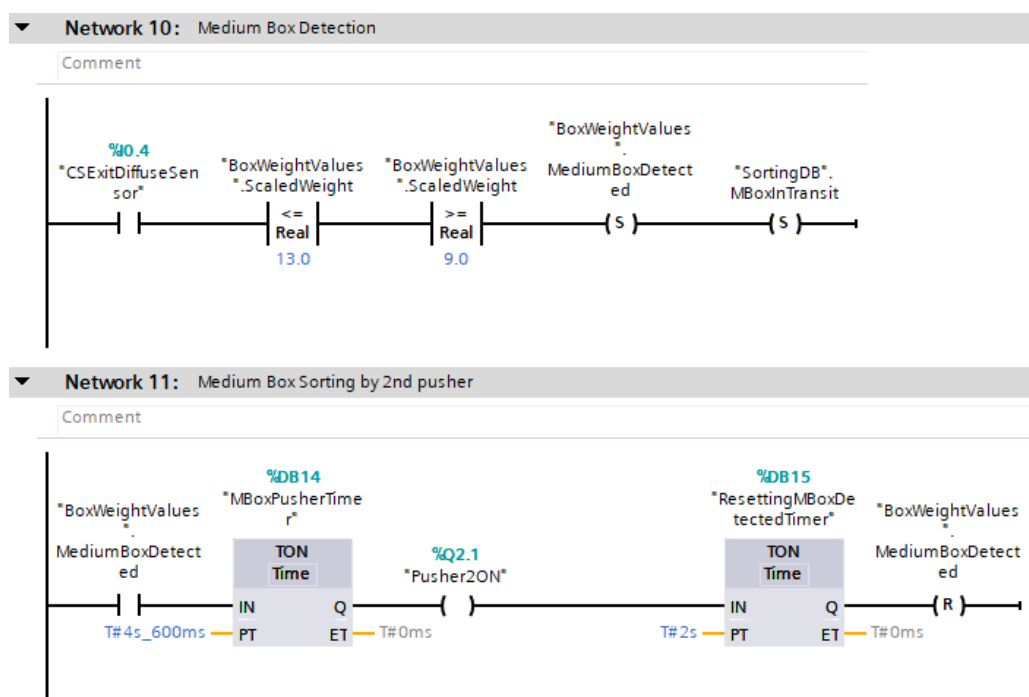


Figure 16 – Network 10 & 11 of 'SortingFC'

Networks 12 to 13 of SortingFC – Sorting of large boxes

Networks 12 and 13 handle the logic for sorting large boxes, shown in Figure 17.

- Purpose: Detect large weight boxes and time pusher activation window.
- Inputs: ScaledWeight > 13.0.
- Outputs: LargeBoxDetected = 1, LBoxInTransit = 1 after completion of LBoxPusherTimer.
- Further comments: LBoxPusherTimer activates pusher when the box is expected to be exactly in front of it, LargeBoxDetected gets reset after push, to be ready for next box.

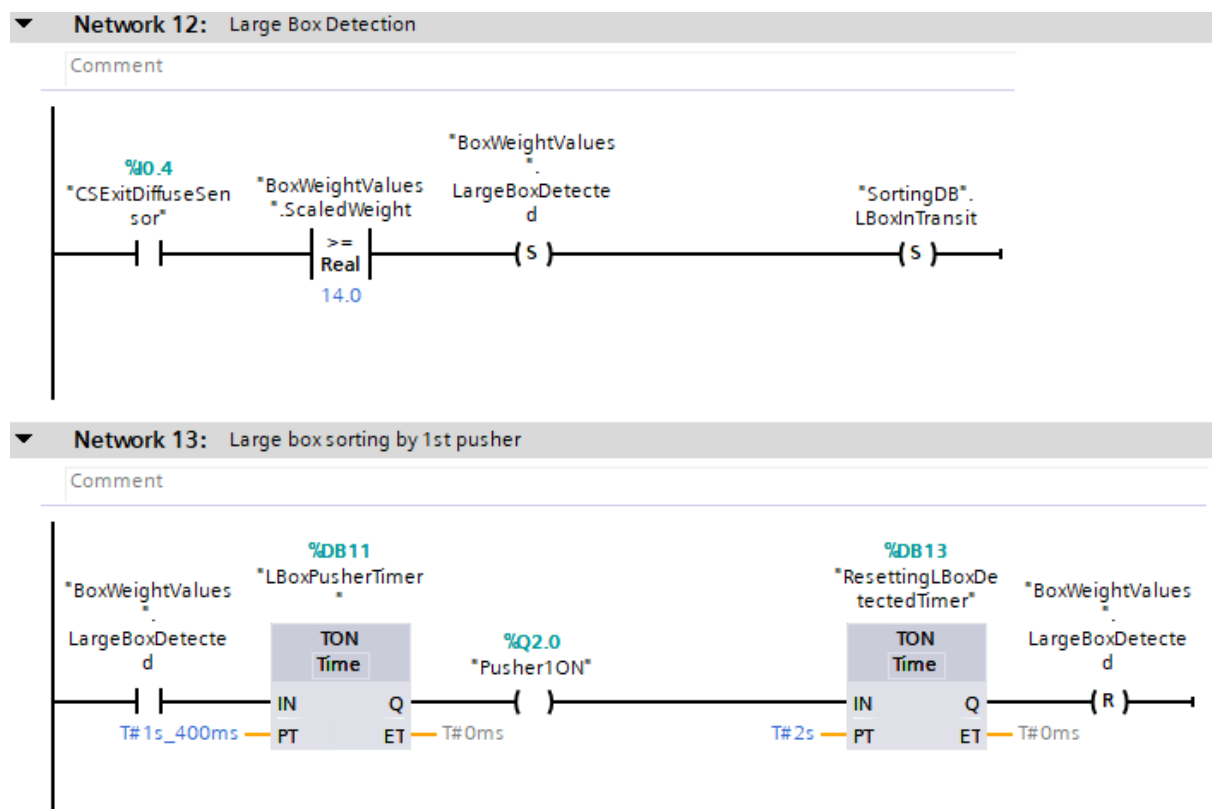


Figure 17 – Networks 12 & 13 of 'SortingFC'

4.3.2 Box Tracking

Next, the '2 – BoxTracking' folder contains code in the 'BoxTrackingFC' function block to track the weight of the box currently on the line and the count of the number of sorted parcels in each area. It also has a data block called 'CountingDB' (Figure 18), which contains tags to supporting the box counting features.

CountingDB									
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static								
2	PBoxCount	DInt	0		☒	☒	☒		Used to track amount of boxes in P Box sorting area
3	SBoxCount	DInt	0		☒	☒	☒		Used to track amount of boxes in S Box sorting area
4	MBoxCount	DInt	0		☒	☒	☒		Used to track amount of boxes in M Box sorting area
5	LBoxCount	DInt	0		☒	☒	☒		Used to track amount of boxes in L Box sorting area
6	ResetCount	Bool	false		☒	☒	☒		Used to reset count when button pressed

Figure 18 – 'CountingDB' Data block.

Network 1 of Box Tracking FC – Resetting all counts

The first network of the Box Tracking FC handles the count resetting, as shown in Figure 19.

- Purpose: Clear parcel counts from control panel or HMI.
- Inputs: ResetCountPB and HMIRestCount.
- Outputs: CountingDB. (P, S, M, L) all made equal to 0.
- Notes: Single press of either reset counts of all sorted boxes.

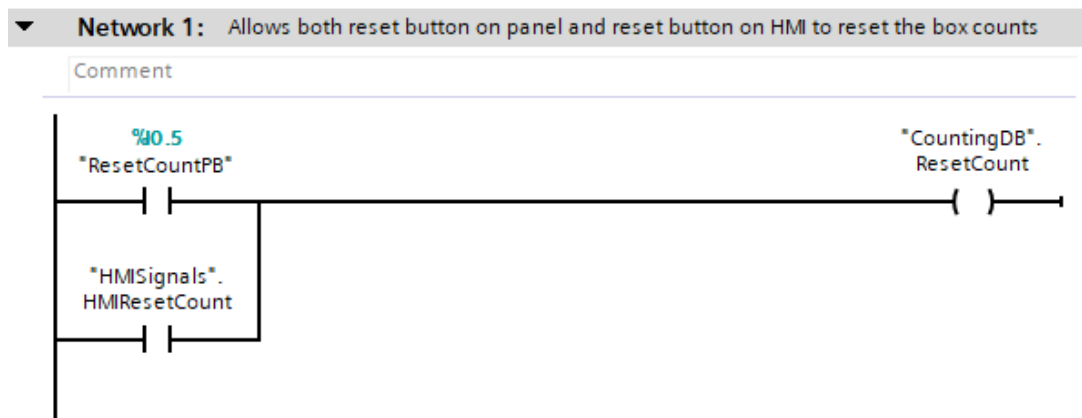


Figure 19 – Network 1 of 'BoxTrackingFC'

Networks 2 to 3 – Conversion for weight display

The weight received from the conveyor scale needed to be converted to an integer and moved to the appropriate LCD to display the weight of the box, as shown in Figure 20.

- Purpose: Show parcel weight of box currently on scale on an LCD as integer.
- Inputs: BoxWeightValues.ScaledWeight
- Outputs: BoxWeightValues.RoundedWeight, then moved to WeightLCD.
- Notes: Real to Integer conversion needed to display weight.

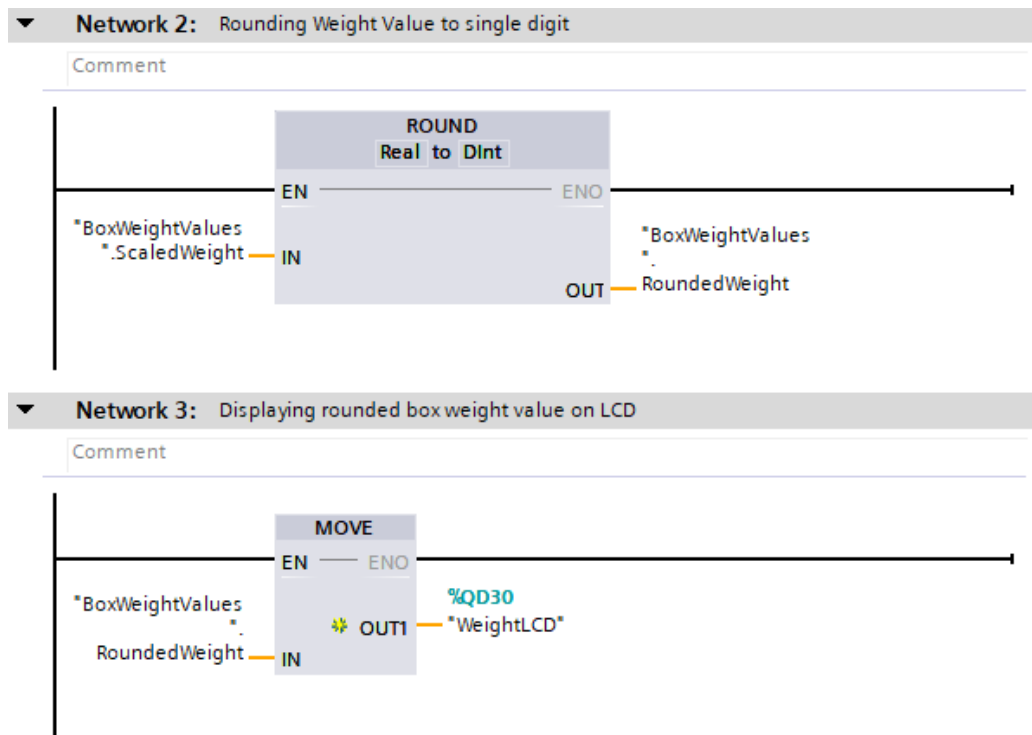


Figure 20 – Networks 2 & 3 of 'BoxTrackingFC'

Networks 4 to 7 – Count of sorted boxes per box size

The counting of each sorted box (through a sensor on its sorting area) is handled in networks 4 to 7, as shown in Figure 21.

- Purpose: Increment count when parcel enters final sorting area.
- Inputs: (P, S, M, L) BoxDiffuseCountSensor per chute.
- Outputs: CTU block increments CountingDB.(P, S, M, L)Count, which is then moved to an appropriate count LCD.
- Further comments: Same structure used to count each box type, with different tags used for each box type.

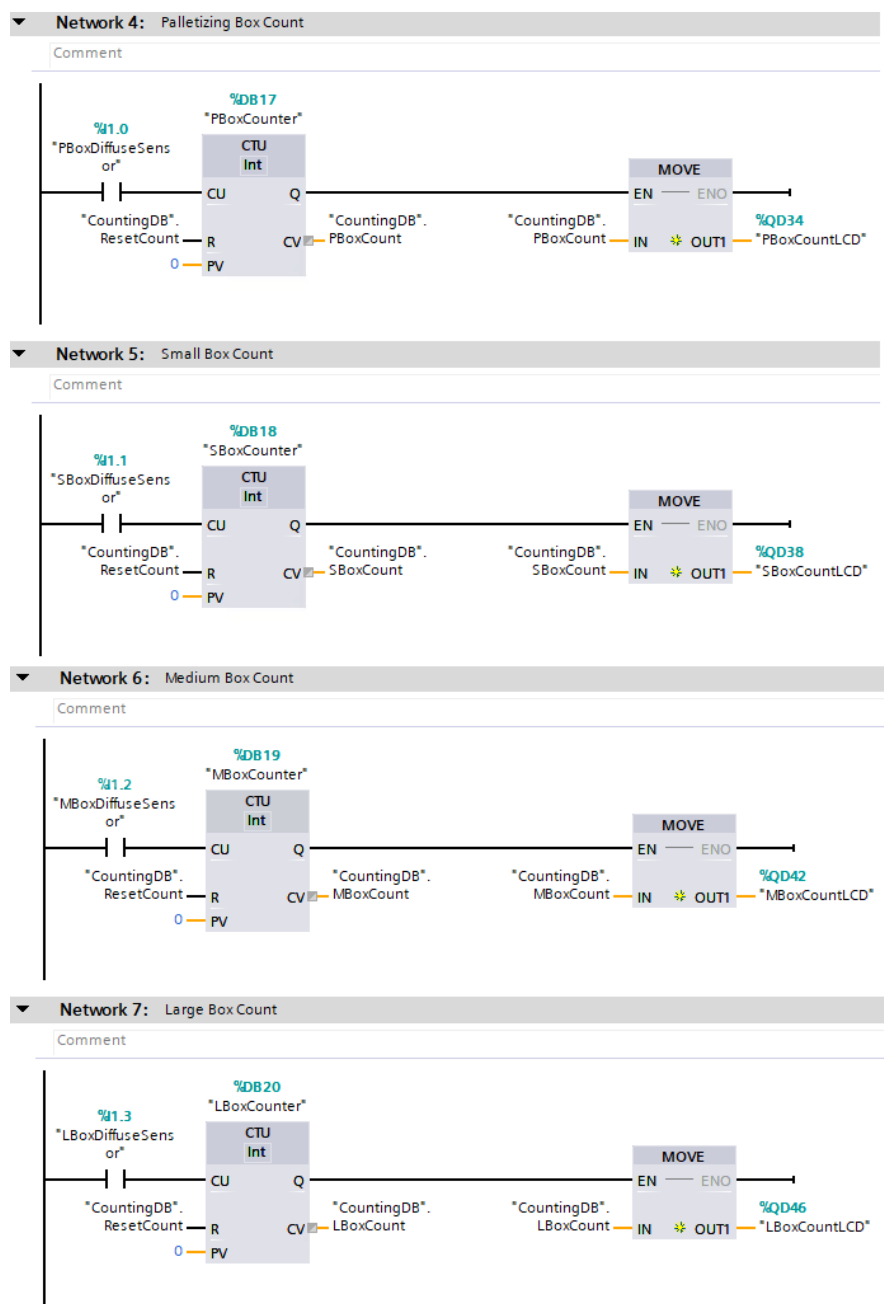


Figure 21 – Networks 4 to 7 in 'BoxTrackingFC'

4.3.3 Alarms Errors Warnings

To keep the system safe and reliable, ladder logic has been created in this program group to show the status of the sorting line and also consider a number of potential errors and warnings that could occur. The system can deal with:

Operation Critical Errors

These errors will halt the operation of the line if they are raised.

- Error A: Boxes not arriving in their specified sorting area in the time frame they should (all four box types).
- Error B: Pushers not extending when they should.
- Error C: Pushers staying extended/not returning when they should.

Warnings

- Warning A: Box sorting areas becoming too full (over 10 in an area).

In regard to the showing the status of the sorting line, stack lights consisting of green, yellow and red segments are used as shown in Table 4. When the system is running perfectly with no errors or warnings, the green segment of the stack light will be lit. When there is a warning on the line, but it is not critical (e.g. a box area filling up), both the green and yellow segments of the stack light will be lit to indicate the line is still operating but there is a warning present. When operation critical errors occur such as pushers not extending when they should, the line will be stopped. This means both the red and yellow segments of the stack light will be lit. These operation critical errors will need to be manually resolved and cleared to be able to start the sorting line again.

Table 4 – Line status and corresponding stack light status.

Line Status	Stack Light Status		
Operating	Green		
Operating with a warning	Green	Yellow	
Error present – cannot operate until cleared		Yellow	Red
Manually stopped			Red

Error A: Box arrival timeout

This error is raised when boxes do not arrive in the expected time frame. The same four-network template was used for each type of box. The example shown in Figure 22 is taken from the small box version, but the same logic with different tags is applied for the other box types.

- Purpose: Stop line operation if box fails to arrive within expected time.
- Inputs: BoxInTransit = 1 (from routing rungs in SortingFC) and BoxArrived tags (from chute diffuse sensors).
- Outputs: If timeout is reached and box hasn't arrived, BoxNotArrivedError = 1, AlarmsErrorsDB.StopIfError (seen in Network 1 of Sorting FC) = 1, Stack Lights display Yellow + Red to indicate line is stopped due to error (as seen in Table 4).
- Further comments: The time each box should take to arrive is presented in Table 5.

Table 5 – Time taken for each box to arrive at its final sorting area.

Box Type	Time taken to arrive at final sorting area
Palletizing	10 seconds
Small	10 seconds
Medium	6 seconds
Large	3 seconds

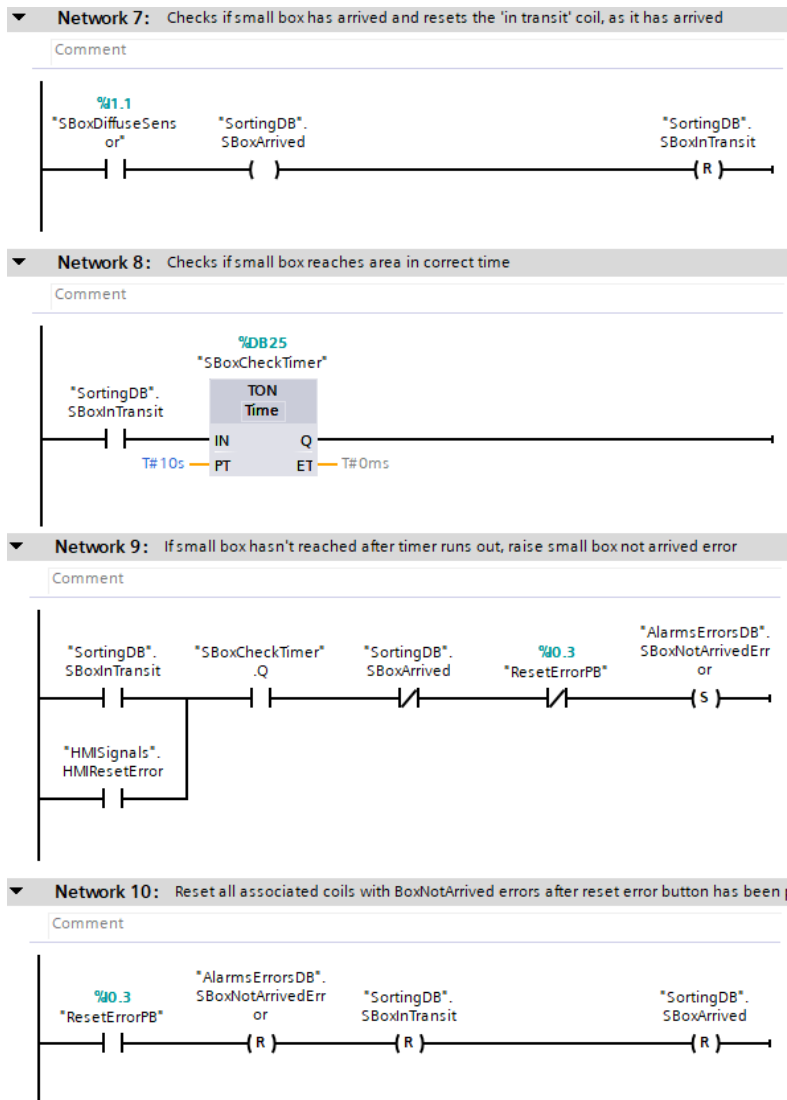


Figure 22 – Box not arrived error ladder logic

Error B: Pusher not extending

This error checks if the front limit of the pusher (active when pusher is fully extended) has been hit. The logic used for this error can be seen in Figure 23.

- Purpose: Detect pusher failing to extend after command.
- Inputs: Pusher (1|2) ON command = 1 and Pusher(1|2)FrontLimit = 0
- Outputs: After TON = 1.2s has completed, pusher has not extended in time so Pusher (1|2)NotExtendedError = 1, AlarmsErrorsDB.StopIfError (seen in Network 1 of Sorting FC) = 1 to stop line operation. Stack Lights display Yellow + Red to indicate line is stopped due to error (as seen in Table 4).
- Further comments: Error can be reset on ResetPB or HMIRreset and line operation can restart once fault has been cleared.

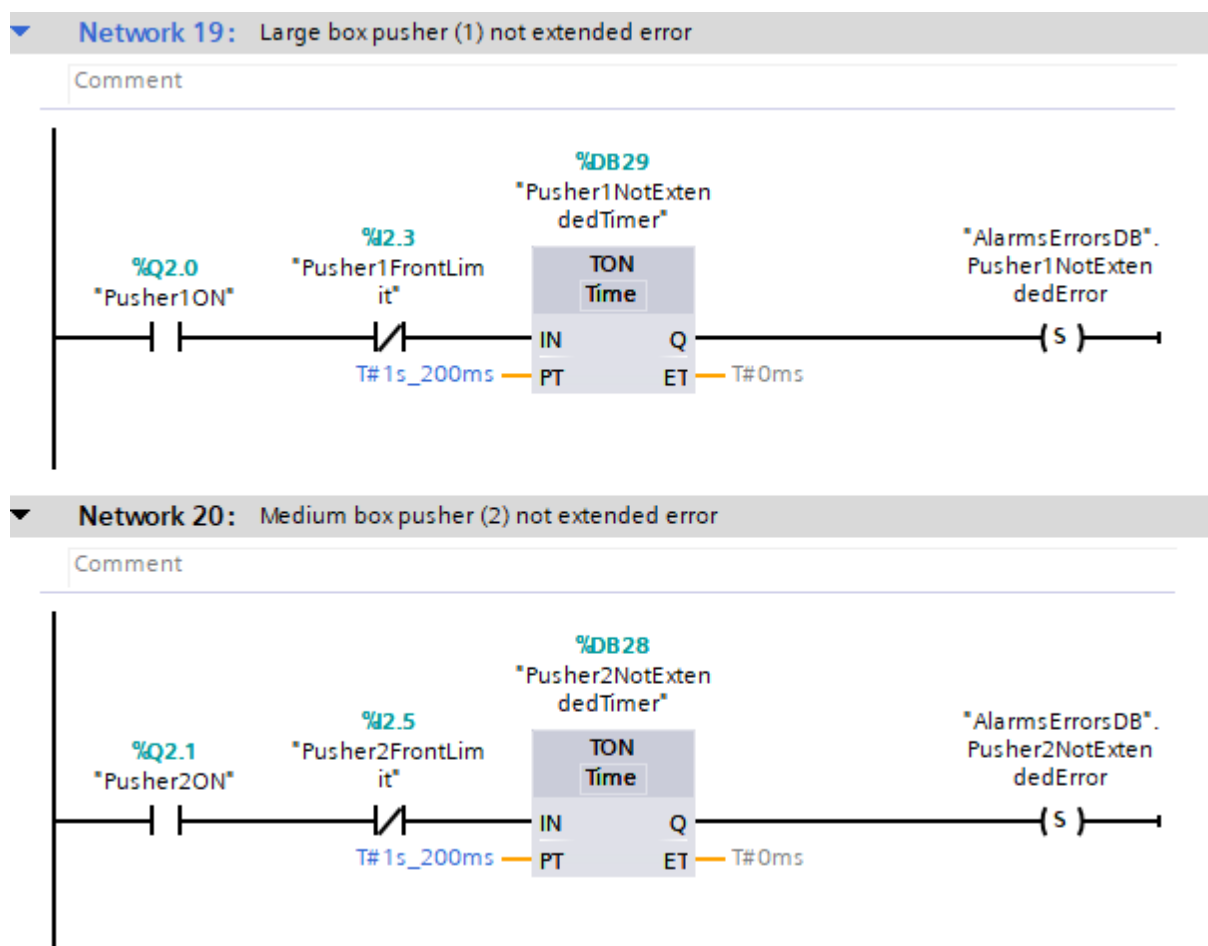


Figure 23 – Pusher not extending error logic

Error C: Pusher stuck extended

This error detects if a pusher is extended longer than it should be, as shown in Figure 24.

- Purpose: Detect pusher failure to return.
- Inputs: Pusher(1|2)FrontLimit = 1 longer than expected after 1.5s TON.
- Outputs: Pusher(1|2)NotReturnedError = 1. Stack Lights display Yellow + Red to indicate line is stopped due to error (as seen in Table 4).
- Further comments: Error raised whenever pusher is extended longer than it should be.

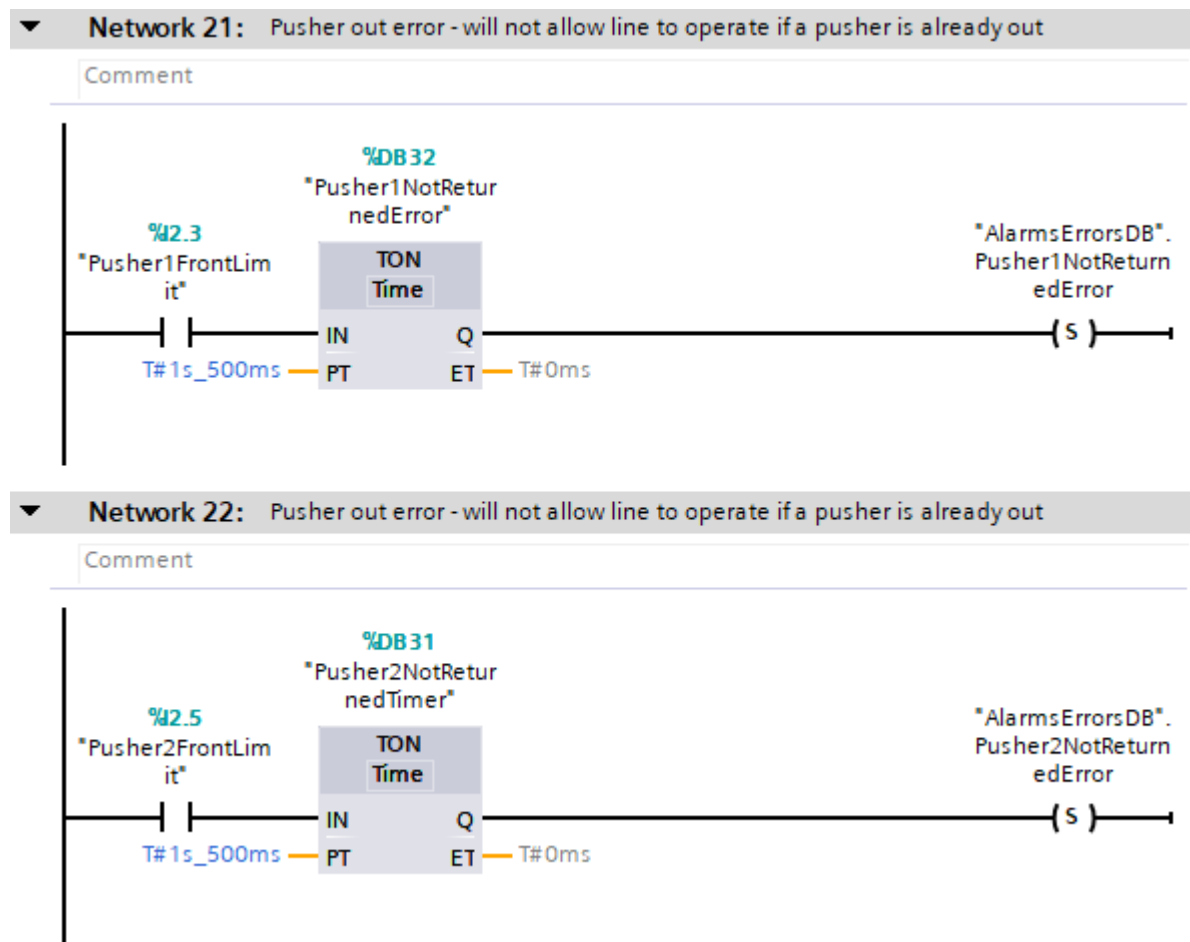


Figure 24 – Pusher extended error

Warning A: Box area overfill warning

This warning indicates when sorted box areas are filling up and the logic for this can be seen in Figure 25.

- Purpose: Indicate when a box area has reached full capacity
- Inputs: CountingDB.(P|S|M|L) ≥ 10
- Outputs: (P|S|M|L) BoxAreaFullWarning = 1. Stack Lights display Yellow + Green to indicate warning present, but operation can continue (as seen in Table 4).
- Further comments: Warning will disappear when boxes have been cleared and the ResetCountPB has been pressed so counts reset.

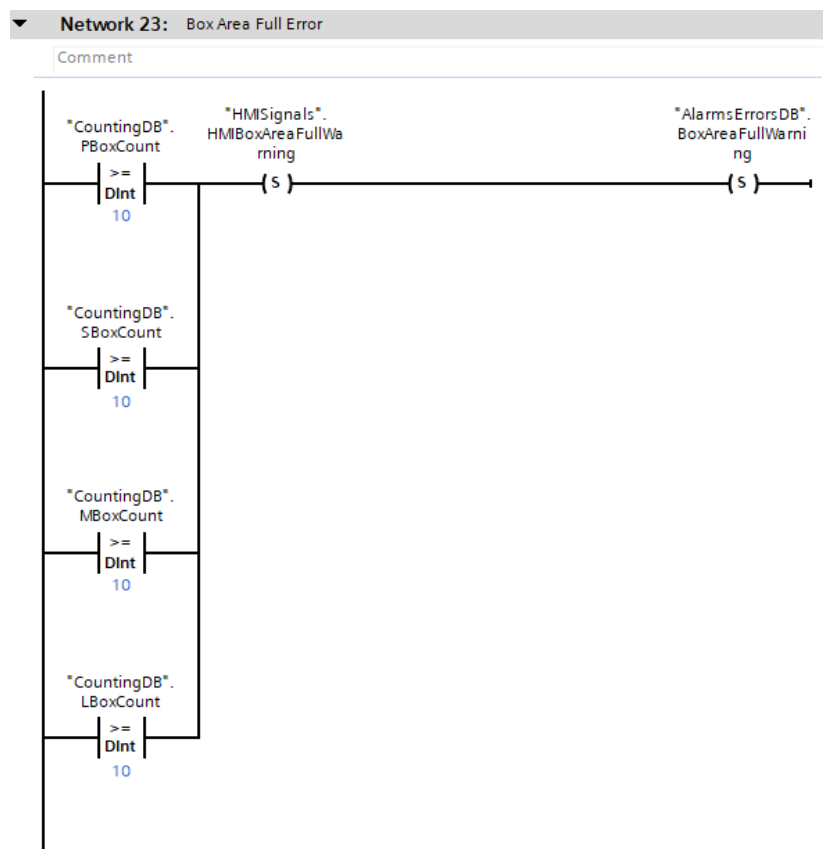


Figure 25 – Box area full error

4.3.4 HMI Signals

The HMI signals program group contains tags used to display all information associated with the sorting line onto the HMI display, as shown in Figure 26. There is no ladder logic in this program group, and just contains these signals that act as a bridge between the sorting line and the HMI. The HMI implementation is further discussed in section 4.0.

HMSignals								
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static			☐	☐	☐	☐	☐
2	HMIStart	Bool	false	☐	☒	☒	☒	☐
3	HMIStop	Bool	false	☐	☒	☒	☒	☐
4	HMIResetError	Bool	false	☐	☒	☒	☒	☐
5	HMIResetCount	Bool	false	☐	☒	☒	☒	☐
6	HMIErrorLight	Bool	false	☐	☒	☒	☒	☐
7	HMIBoxMaxCount	Int	0	☐	☒	☒	☒	☐
8	HMIBoxMaxCount	Int	0	☐	☒	☒	☒	☐
9	HMIBoxMaxCount	Int	0	☐	☒	☒	☒	☐
10	HMIBoxMaxCount	Int	0	☐	☒	☒	☒	☐
11	HMIBoxAreaFullWarni...	Bool	false	☐	☒	☒	☒	☐

Figure 26 – ‘HMI Signals’ data block

5.0 Task 4: HMI Integration

To incorporate a Human-Machine Interface (HMI) into the program, the HMI simulation feature in TIA Portal V16 was utilised. The HMI used (KTP700 Basic PN) for this factory control system utilises three pages (main screen, errors screen and engineer's screen).

Main Screen

The HMI main screen (shown in Figure 27) contains buttons to operate the sorting line, a stack light to display the system's operation, the current count of each sorted box and the weight of the box currently on the line. It also contains a way to reset the counts of the sorted boxes in each area and a button to navigate to the errors screen. When an error is present, the stack light will change to give the user a prompt to open the errors page, as shown in Figure 28. The engineer's screen can also be accessed from this page, by pressing F3 at the bottom of the screen. This is the most commonly used screen and has no access limitations.

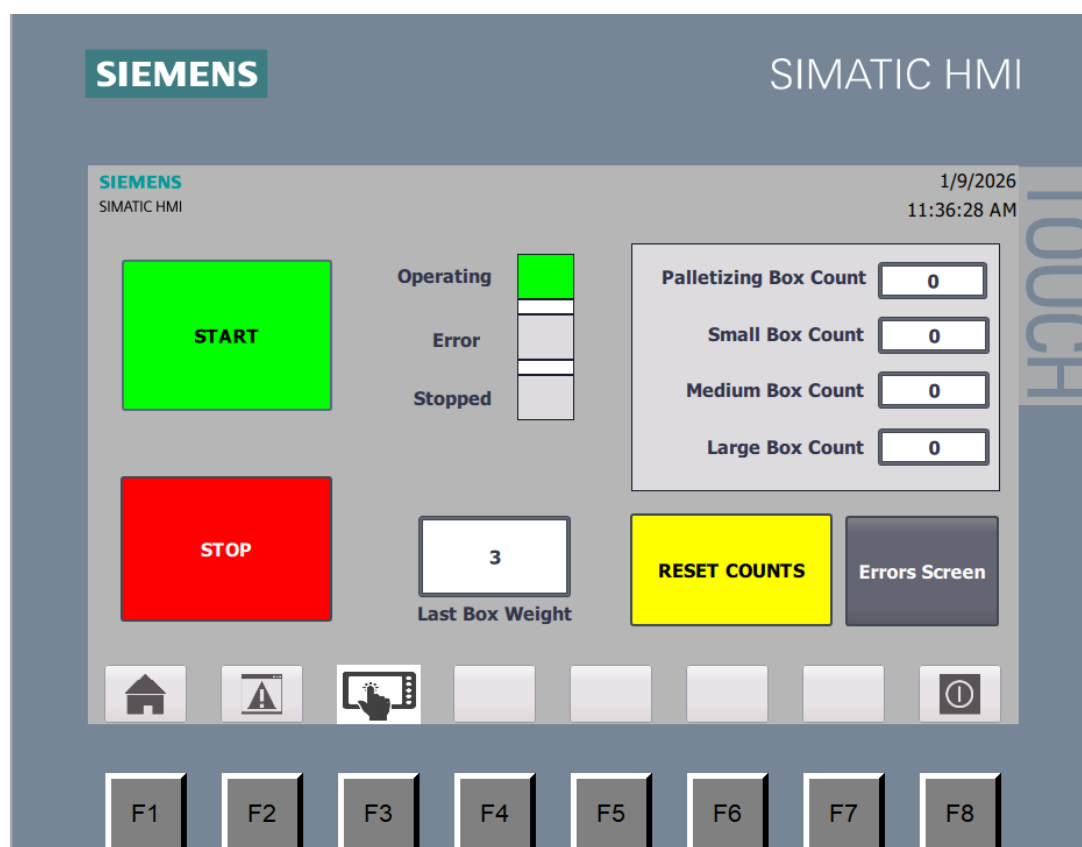


Figure 27 – HMI Main Screen

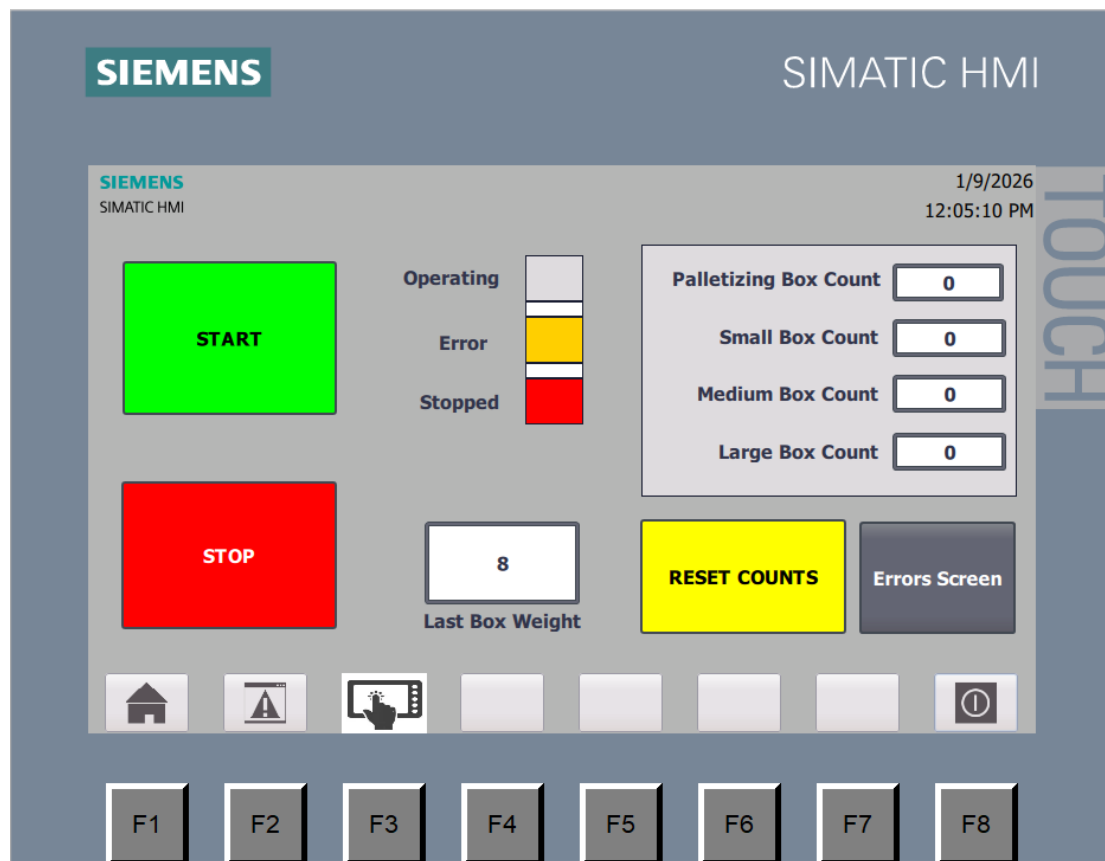


Figure 28 – Error present and line stopped, user prompt to check errors page

Errors Screen

When there is an error on the line and operation has been halted, the errors page (shown in Figure 29) can be opened to identify the fault. When an error is raised, it will be highlighted (as shown with the Pusher 2 Not Returned error below). When the fault has been dealt with, the user can reset the errors using the button and can resume the line operation by navigating back to the main menu. Additionally, any warnings that are present will be highlighted (as shown with the 'Box Area Full' error).

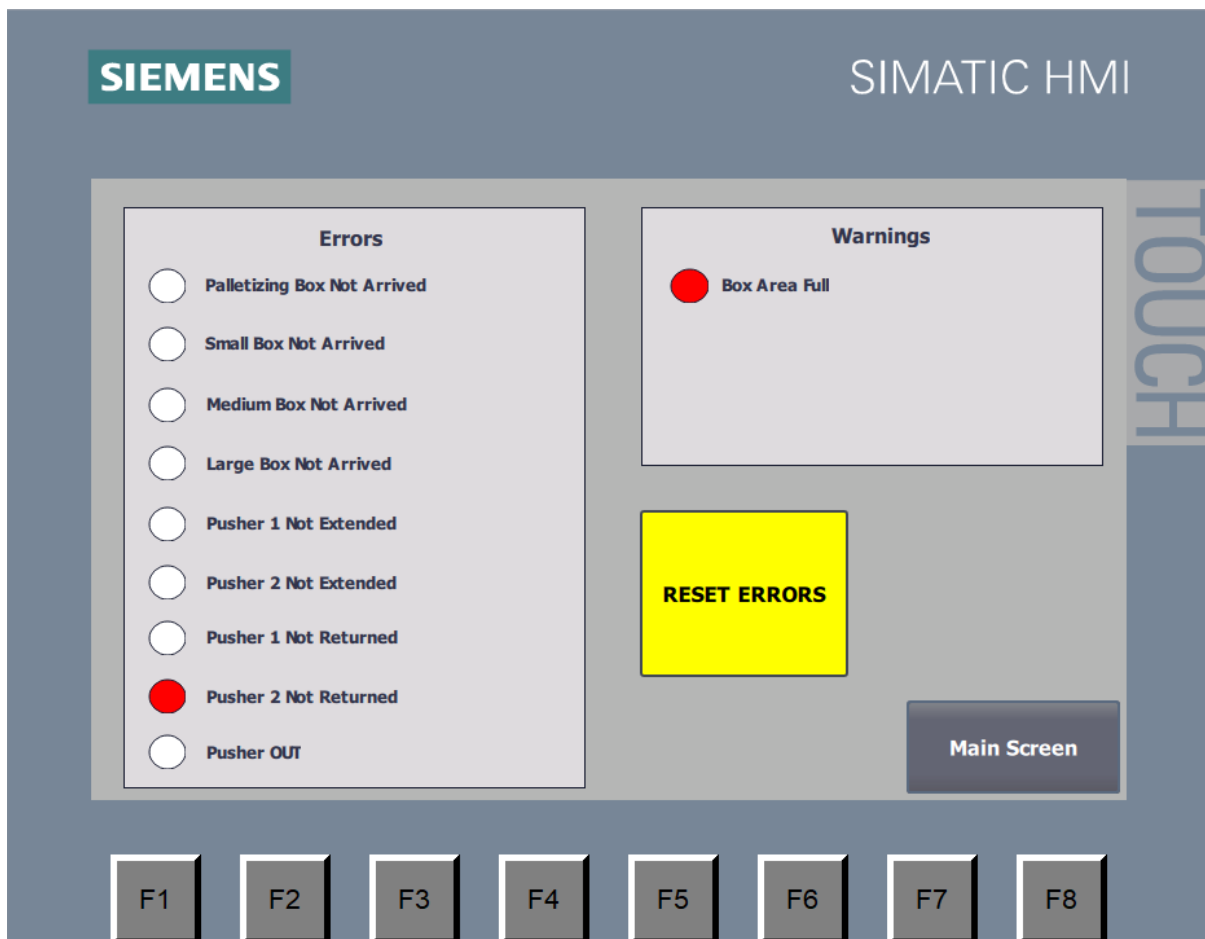


Figure 29 – HMI Errors Screen

Engineer's Screen

The engineer's screen is only available to authorised personnel and requires a login to be accessed, as shown in Figure 30. The login should only be shared to authorised engineers to ensure those who are not competent cannot access it. The engineer's screen will allow the user to update key information of the sorting line, such as the limits of sorted boxes in each area (shown in Figure 31).

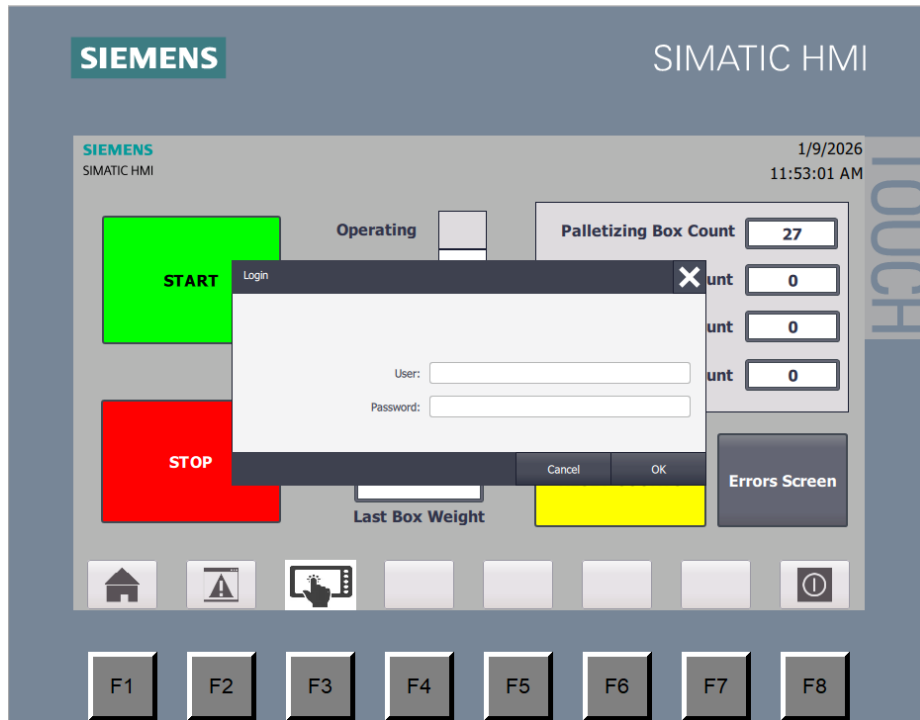


Figure 30 – Login required to access engineer's screen.

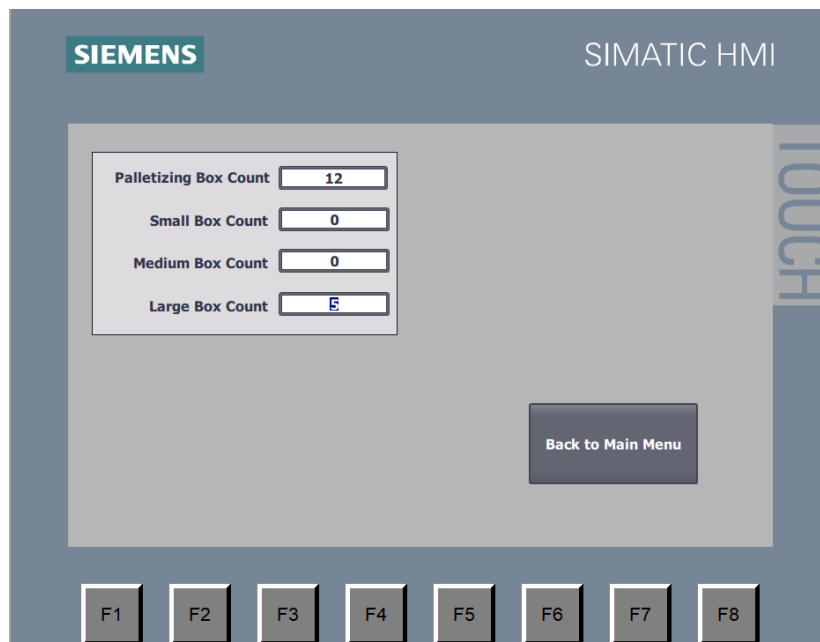


Figure 31 – Engineer's Screen

6.0 Discussion/Evaluation

The development of the automated package-sorting system provided an opportunity to apply PLC programming and the design of an appropriate system in a controlled environment. The project was shown to meet the main objective of being able to sort four different package types, but opportunities for further improvements have been highlighted.

One of the key strengths of the project was the system design. Conveyors, sensors and pushers were used in the Factory I/O layout to meet the requirement of sorting four different package types. A thorough understanding of industrial automation principles and the requirements of the system was shown by including a sequential start-up of conveyors to limit current draw, as well as the use of diffuse sensors across the sorting line and including a conveyor scale to weigh the packages. The system operated reliably under normal conditions, and the logic implemented for package detection and sorting performed consistently during operation.

The ladder logic program created in the TIA portal also proved to be effective in controlling the operation of the system. The program was commented throughout to ensure it was of a high standard with rungs, data blocks, and named timers and counters also being used to ensure that the control strategy was manageable and easy to follow.

The HMI design also successfully provided the required operator controls and gave real-time information of the sorting line. Users could successfully start and stop the line, as well as being able to reset counts and errors on the system through the HMI. The HMI gave a good demonstration on how operators interact with real systems, however, there was room for improvement with a clearer grouping of tags, navigation between pages and additional information on alarms when they occur.

Overall, the project showed that the combination of a competent factory I/O design, a complete ladder logic program and an appropriate HMI design can lead to a functioning package sorting system. The final system met all the requirements of the brief, however, some improvements were identified that could further enhance the operation of the system.

6.1 Recommendations for future work

In terms of recommendations for future work, it was seen that there are key points that would improve the system. As an example, using dedicated Siemens safety modules (like the ET200SP [22]) and certified safety components would bring the system closer to real-world standards. The use of safety components would ensure that emergency stops, and guard monitoring (e.g. safety door breaches) are handled with the appropriate redundancy and reliability.

Currently, the system can successfully sort four different packages however, the speed and frequency at which the packages can be sorted could be improved. Increasing the speed of the incoming packages would allow the system to be stress-tested to see if the current sensing and logic could withstand a more realistic sorting environment. Another recommendation for further work is investigating the inclusion of AI in the system. AI driven tools could be used to provide enhanced package identification (e.g. a vision-based detection system) and predictive maintenance, which could significantly benefit the system's capability and also reduce downtime as potential faults would be identified before they occur. Overall, these recommendations would strengthen the system and prepare it for operation under more demanding conditions.

7.0 Conclusion

In conclusion, this project successfully developed an effective and reliable package sorting system that met all given requirements. The integration of the Factory I/O design, the development of ladder logic using the Siemens TIA Portal, and the implementation of an HMI demonstrated a thorough understanding of industrial automation principles. Additionally, the use of global data blocks also ensured tags were easy to find and monitor for future use. The system operated reliably, accurately sorting four package types whilst having a structured program and the necessary safety features. The alarms and errors routines included in the program also ensured that the system would not operate if an error was present, meaning the system stayed safe and reliable throughout use.

Although the system performed as expected, the evaluation identified areas for improvement such as including enhanced HMI screen navigation, increased sorting speed and frequency, and the potential integration of AI for advanced box detection and predictive maintenance. These additions would further improve the performance of the system and allow it to be more similar to systems that run under real-world industrial standards.

Overall, the project provided a practical demonstration of how PLCs, simulation tools, and HMIs can be combined to create a realistic and efficient automated sorting system.

8.0 References

- [1] M. Walker, C. Bissell and J. Monk, "The PLC: A Logical Development," *Measurement and Control*, vol. 43, no. 9, pp. 280-284, 2010.
- [2] W. Bolton, *Programmable Logic Controllers*, Newnes, 2015.
- [3] K. T. Erickson, "Programmable logic controllers," *IEEE Potentials*, vol. 15, no. 1, pp. 14-17, 1996.
- [4] E. R. Alphonsus and M. O. Abdullah, "A review on the applications of programmable logic controllers (PLCs)," *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185-1205, 2016.
- [5] R. Netto and A. Bagri, "Programmable Logic Controllers," *International Journal of Computer Applications*, vol. 77, no. 11, pp. 27-31, 2013.
- [6] K.-H. John and M. Tiegelkamp, *IEC 61131–3: Programming Industrial Automation Systems*, Berlin: Springer, 2001.
- [7] H. Jack, *Automating Manufacturing Systems with PLCs*, Lulu Press, Inc, 2009.
- [8] V. V. Godase, "Comparative Study of Ladder Logic and Structured Text Programming for PLC," *Journal of Electronics Design and Technology*, vol. 2, no. 2, pp. 34-44, 1015.
- [9] R. Ramanathan, "The IEC 61131-3 programming languages features for industrial control systems," in *2014 World Automation Congress*, Hawaii, USA, 2014.
- [10] G. P. Sandaruwan, P. S. Ranaweera and V. A. Oleshchuk, "PLC security and critical infrastructure protection," in *2013 IEEE 8th International Conference on Industrial and Information Systems*, Peradeniya, Sri Lanka, 2013.
- [11] A. Serhane, M. Raad, R. Raad and W. Susilo, "Programmable logic controllers based systems (PLC-BS): vulnerabilities and threats," *SN Applied Sciences*, Vols. 1, 924, 2019.
- [12] Y. Zhang, Z. Sun, L. Yang, Z. Li, Q. Zeng and Y. He, "All Your PLCs Belong to Me: ICS Ransomware Is Realistic.," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Guangzhou, China, 2020.
- [13] X. Pan, Z. Wang and Y. Sun, "Review of PLC Security Issues in Industrial Control System," *Journal of Cybersecurity*, vol. 2, no. 2, pp. 69-83, 2020.
- [14] N. Dhameliya, "Revolutionizing PLC Systems with AI: A New Era of Industrial Automation," *American Digits: Journal of Computing and Digital Technologies*, vol. 1, no. 1, pp. 33-48, 2023.

- [15] Siemens, "S7-1200 G2 Programmable Logic Controller," [Online]. Available: https://docs.tia.siemens.cloud/r/simatic_s7_1200_g2_manual_collection_enus_20/introduction. [Accessed 15th December 2025].
- [16] D. Beresford, "Exploiting Siemens Simatic S7 PLCs," in *Black Hat USA+2011*, Las Vegas, 2011.
- [17] Lucy Heather, "ControlLogix vs. CompactLogix: Allen-Bradley PLC Comparison," Mizen Automation, 23rd December 2024. [Online]. Available: https://mizenautomation.com/controllogix-compactlogix-comparison?utm_source=copilot.com. [Accessed 6th January 2026].
- [18] Allen-Bradley, "ControlLogix 5570 and 5560 Controllers," [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1756-um001_en-p.pdf. [Accessed 6th January 2026].
- [19] Rockwell Automation, "Allen-Bradley Communications Modules," August 2017. [Online]. Available: https://literature.rockwellautomation.com/idc/groups/literature/documents/pp/logix-pp002_en-p.pdf. [Accessed 7th January 2026].
- [20] Hitachi, "Hitachi Industrial Equipment Systems Europe," Hitachi, [Online]. Available: <https://hitachi-industrial.eu/products/drivesautomation/micro-ehv-series/>. [Accessed 7th January 2026].
- [21] Siemens, "Programming basics," [Online]. Available: <https://docs.tia.siemens.cloud/r/enus/v20/programming-basics>. [Accessed 15th December 2025].
- [22] Siemens, "Fail-safe distributed I/O systems with SIMATIC ET 200," Siemens, [Online]. Available: <https://www.siemens.com/global/en/products/automation/topic-areas/safety-integrated/factory-automation/offering/simatic-safety/fail-safe-io-systems.html>. [Accessed 7th January 2026].
- [23] F. I/O, "Items," Factory I/O, [Online]. Available: <https://docs.factoryio.com/manual/parts/items/>. [Accessed 15th December 2025].

Appendix

Appendix A – Program/Factory IO information

Small 8 Kg	Medium 10 Kg	Large 15 Kg	Palletizing Box 3 Kg
			

Figure 32 – Factory IO box weight values [22].

Table 6 – Assigned **Input** Tags in Factory IO/TIA Portal.

Tag Name	Tag Address	Comment
CSDiffuseSensor	I0.0	Diffuse sensor on conveyor scale
StartPB	I0.1	Start Push Button
StopPB	I0.2	Stop Push Button
ResetPB	I0.3	Reset Push Button
CSExitDiffuseSensor	I0.4	Diffuse sensor on the exit of conveyor scale
ResetCountPB	I0.5	Reset Count Push Button
SPARE	I0.6	
SPARE	I0.7	
PBoxCountDiffuseSensor	I1.0	Diffuse sensor to count palletizing boxes
SBoxCountDiffuseSensor	I1.1	Diffuse sensor to count sorted small boxes
MBoxCountDiffuseSensor	I1.2	Diffuse sensor to count sorted medium boxes
LBoxCountDiffuseSensor	I1.3	Diffuse sensor to count sorted large boxes
SPARE	I1.4	
SPARE	I1.5	
SPARE	I1.6	
SPARE	I1.7	
EStopPB	I2.0	Emergency Stop Push Button
SafetyDoor	I2.1	Safety Door to sorting line
Pusher1BackLimit	I2.2	Pusher 1 in OFF position
Pusher1FrontLimit	I2.3	Pusher 1 fully extended
Pusher2BackLimit	I2.4	Pusher 2 in OFF position
Pusher2FrontLimit	I2.5	Pusher 2 fully extended
SPARE	I2.6	
SPARE	I2.7	
CSBoxWeight	ID30	Value received from conveyor scale (V).
SPARE	ID34	
SPARE	ID38	
SPARE	ID42	
SPARE	ID46	
SPARE	ID50	
SPARE	ID54	
SPARE	ID58	

Table 7 – Assigned **Output** Tags in FactoryIO/TIA Portal.

Tag Name	Tag Address	Comment
AlarmSiren	Q0.0	Siren to indicate line is ON
SPARE	Q0.1	
BeltConveyor 1	Q0.2	2m
BeltConveyor2	Q0.3	2m
BeltConveyor3	Q0.4	2m
StraightSpurConveyor1 (-)	Q0.5	Moves palletizing boxes after pop-up sorter (- direction)
StraightSpurConveyor2 (+)	Q0.6	Moves small boxes after pop-up sorter (+ direction)
SPARE	Q0.7	
ConveyorScale1 (+)	Q1.0	Starts conveyor scale
SPARE	Q1.1	
SPARE	Q1.2	
SPARE	Q1.3	
PopUpSorter	Q1.4	Used to start pop up sorter
PopUpSorterLeft	Q1.5	Used to sort small boxes
PopUpSorterRight	Q1.6	Used to sort palletizing boxes
SPARE	Q1.7	
Pusher1	Q2.0	Used to sort large boxes
Pusher2	Q2.1	Used to sort medium boxes
StackLight1Red	Q2.2	Red part of stack light 1
StackLight1Yellow	Q2.3	Yellow part of stack light 1
StackLight1Green	Q2.4	Green part of stack light 1
StackLight2Red	Q2.5	Red part of stack light 2
StackLight2Yellow	Q2.6	Yellow part of stack light 2
StackLight2Green	Q2.7	Green part of stack light 2
WeightLCD	QD30	Displays weight of box currently on sorting line
PBoxCountLCD	QD34	LCD to show Palletizing Box Count
SBoxCountLCD	QD38	LCD to show Small Box Count
MBoxCountLCD	QD42	LCD to show Medium Box Count
LBoxCountLCD	QD46	LCD to show Large Box Count
SPARE	QD50	
SPARE	QD54	
SPARE	QD58	

Appendix B – Full program breakdown

SortingFC

The 'SortingFC' function block in this folder holds the ladder logic to operate the sorting line. Network 1 (Figure 10) controls the 'Operating' status and uses a latch so the line keeps running without the need to hold the start button down. The program will only run if the pre-run conditions of the factory are met. This means there must be no errors on the line (such as pushers starting fully extended etc.) and the safety door to access the sorting line must also be closed. A description of each tag used in network 1 has been included in Table 3.

To limit the current draw when the sorting line starts up, each conveyor is started sequentially. Network 3 (Figure 11) handles the logic to do this and uses multiple timers of 1 second intervals between starting each conveyor. The ordering of this rung starts the first conveyor belt and moves throughout the line by finally starting the straight spur conveyors at the end of the line.

Network 5 of the 'SortingFC' function block (shown in Figure 12) handles the box-weighing sequence and includes two TON timers. The first TON timer 'StabiliseBoxTimer' (DB8) gives the box time to settle on the scale before taking a weight reading. The timer starts when the 'BoxLoaded' tag is true and after the 'ScaleControl' tag has been set to true. After 300ms of stable load, the output of the timer become true, which sets the 'Ready' tag to indicate that a weight value can now be taken for the box. The 'Ready' tag starts the 'BoxWeighingTimer' (DB9) and after two seconds have passed, the 'Ready' and 'ScaleControl' tags are reset.

Network 6 (shown in Figure 12) detects when a box enters in the conveyor scale through the use of a diffuse sensor. When the diffuse sensor is hit by a box, the off-delay timer 'BoxDetectionHoldTimer' is held on and sets the 'BoxLoaded' tag. When the box has been weighed and has passed the diffuse sensor, the timer keeps the 'BoxLoaded' tag on for 5 seconds. This is included to prevent any errors from possible sensor flickers and ensure the correct weight value of the box is received.

The initial value received from the conveyor scale needed to be scaled to reflect the weight values of each box (seen in Figure 32). Network 7 of 'SortingFC' handles the ladder logic created to do this (Figure 13). When the 'Ready' tag (set in Network 5) is true, the raw value is taken from the conveyor scale and moved to the 'RawWeight' tag in the 'BoxWeightValues' data block using a 'MOVE' block. After this, the value in the 'RawWeight' tag is multiplied by 2 using a multiply block, and the output of this is moved to the 'ScaledWeight' tag in the 'BoxWeightValues' data block.

The following rungs deal with the sorting of each box depending on the scaled weight received from the conveyor scale. When a type of box is detected, a tag to say the type of box that has been detected is in transit (e.g. PBoxInTransit) and this tag is used in the error handling program discussed later in the report.

The 'palletizing' boxes are sorted through the logic in Network 8 (Figure 14). When the box leaves the conveyor scale, a diffuse sensor on the exit of it will be hit and if the weight of the box is 3 (or ≤ 4.0), the 'PalletizingBoxDetected' and 'PBoxInTransit' tags will be set. The pop-up sorter's right function will then be activated to sort the boxes in the correct area. The comparison blocks are taken from the comparator operations folder, which is readily available in TIA Portal.

Similarly to the 'palletizing' box sorting, small boxes are sorted by the pop-up wheel sorter at the end of the line (Figure 15). As the small boxes weigh 8kg, the program needs to use the comparator operations to ensure the box is sorted correctly. When the conveyor scale exit diffuse sensor is hit and the weight of the box is greater than 4kg but less than 9kg, the 'SmallBoxDetected' and 'SBoxInTransit' tags are set to true and the pop-up sorter's left function is activated to sort the box into the small boxes area.

In regard to the sorting of medium boxes, networks 10 and 11 (Figure 16) handle the logic to do this. The medium box weighs 10kg and is sorted by the 2nd pusher. Similarly to the previous sorting logic of the other boxes, comparator operations are used and if the weight of the box is above 9kg and below 13kg, the 'MediumBoxDetected' and 'MBoxInTransit' tags will be set to true. However, as a pusher is needed to sort medium boxes, a timer (MBoxPusherTimer) is required to activate the pusher exactly when the box is in front of it. After the pusher has been activated, a 2 second timer runs to then reset the 'MediumBoxDetected' tag. This allows the sorting line to be ready for the next box to be sorted.

The large boxes weigh 15kg and are sorted by the first pusher. Networks 12 and 13 handle the logic for sorting large boxes, as seen in Figure 17. Similarly to the other boxes, a comparator operation is used so that if the weight of the box is over 13kg, the 'LargeBoxDetected' and 'LBoxInTransit' tags are set to true. This will then start the 'LBoxPusherTimer' and will activate pusher 1 when the box is in front of it. After this, a 2 second timer takes place and will reset the 'LargeBoxDetected' tag so following boxes can be sorted correctly.

Box Tracking FC

The program in the 'BoxTrackingFC' function block starts with network 1, which allows both the reset count push button on the factory control panel and the reset count button on the HMI to reset all box counts (seen in Figure 19).

As the weight value received from the conveyor scale is currently a real (e.g. 15.0), it needs to be converted to an integer so it can be displayed on a digital display. A 'round' block is used to do this, where the scaled weight value from the conveyor scale is converted to an integer and stored in the 'RoundedWeight' tag. This tag is then displayed onto the appropriate digital display using a 'move' block, as shown in Figure 20.

To count the amount of each box type that has been sorted, a diffuse sensor is placed on the chute leading to each sorting area. Networks 4 to 7 of the 'BoxTrackingFC' block follow the same template to track the count of sorted boxes. As a box is about to reach its final destination, it will hit the diffuse sensor on the chute and add '1' to the current count. This is done using a count up block, and the count is then moved to the appropriate digital display as shown in Figure 21.

Alarms Errors Warnings

When creating the program to raise an error for the boxes not arriving in the specified time period, the same four-network template was used for each type of box. The example included below is taken from the small box version, but the same logic with different tags is applied for the other box types.

The first network in the template checks to see if the box has hit the diffuse sensor on the sorting area. If it has, the box has been sorted correctly and 'BoxArrived' tag is set to true. As previously mentioned, when a box of a certain type is detected based on the weight comparisons in the 'SortingFC' program, a tag stating the box is in transit is set to true. This tag gets reset when the box has arrived, as shown in network 3 (Figure 22 – Box not arrived error ladder logicFigure 22). The next rung uses the 'BoxInTransit' tag to activate a timer based on how long the box should take to arrive in its sorting area. This timer value varies based on the box type, and the time each box type should take to arrive in its sorting area is presented in Table 5.

If this timer completes and the box still hasn't arrived, an error will be raised to indicate this. This error will stop the operating line and prompt operators to solve faults on the line before sorting operations can resume. In the next network, any tags related to the box in transit will be reset if the reset button is pressed.

Another error that can be detected on the sorting line is the pushers not extending when they should. When the pusher is activated and it hasn't fully extended, a 1.2 second timer will begin. If this timer is able to be completed, this means the pusher did not extend in time or at all. An error called 'Pusher1NotExtendedError' or 'Pusher2NotExtendedError' will be set, as shown in Figure 23.

On the other hand, it can also be detected when a pusher has been extended for longer than it should be (suggesting it is stuck), as shown in Figure 24. When the front limit of a pusher is hit, it should start retracting approximately 1 second after. A 1.5s timer has been included so that if this timer completes, the pusher has taken too long to return, and it will set a 'PusherNotReturned' error. This is seen as a critical error, and the line will stop until the error is solved and manually reset.