FLUID AUTOMATION TASK 1 CONVEYORS

Politecnico di Torino

Students:

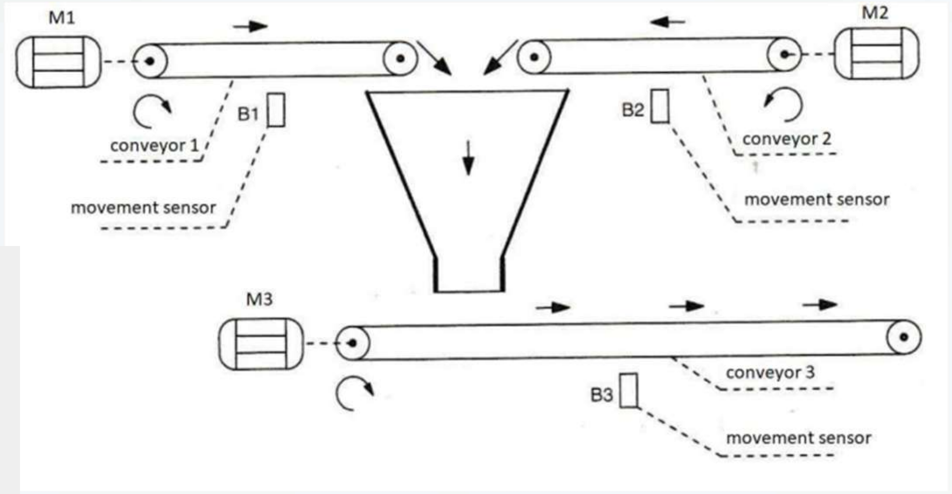
Muratore Luigi	s333098
Akbarov Iskandar	s329650
Muhammad Fatir Noshab	s331898

Professor:

Luigi Mazza

CONTENT

01 Requirements



Inputs:

- Start button **S1** to start conveyor 1 (pulse signal)
- Start button **S2** to start conveyor 2 (pulse signal)
- Stop button **S3** to stop conveyor 1 (pulse signal)
- Stop button **S4** to stop conveyor 2 (pulse signal)
- Emergency button **S5** to stop all conveyors (selector)
- Sensor **B1** (detects movement of conveyor 1)
- Sensor **B2** (detects movement of conveyor 2)

Outputs:

- Motor **M1** of conveyor 1
- Motor **M2** of conveyor 2
- Motor **M3** of conveyor 3
- Lamp **H1** (indicates the status of conveyor 1)
- Lamp **H2** (indicates the status of conveyor 2)
- Lamp **H3** (indicates the status of conveyor 3)

Goals and Objectives

Activation of conveyors

- **S1** commands **M1**
- **S2** commands **M2**
- Conveyors 1 and 2 cannot move contemporary
- Conveyor 3 is activated when conveyors 1 or 2 are active
- Lamps H1 and H2 are switched on when motors M1 and M2 are active respectively

Deactivation of conveyors

- S3 and S4 are pressed to stop conveyors 1 and 2 respectively,
- they must be arrested after 20 s
- Conveyor 3 is arrested 60 s after pressing either S3 or S4

System operations

- Sensors **B1** and **B2** detects the movement of conveyors 1 and 2 respectively:
  - during the start-up of the conveyors, the signal of the sensors is not considered for the first 3 s

- In case, If motor M1 or M2 fails during the normal operation of conveyors 1 and 2:
  - the motor must be deactivated and lamp H1 (or H2) must begin to flash.
  - conveyor 3 must continue to move for 40 s before it stops and red lamp H3 must be enlighted

CHANGES

We made some changes to the requirements in order to have more reactive and understandable system. All the changes do not affect either the system operations or the goals and the objectives.

1. Regarding the **time delay of the timers** in order to have a more interactive simulations:

- Timer of 20 seconds -> 2 seconds
- Timer of 60 seconds -> 6 seconds
- Timer of 40 seconds -> 4 seconds

2. Regarding the **lights in the HMI**:

- In situation of failure scenarios, instead of having the same lights with different colours or behaviours to show different failures in case of malfunction we employ different lights in order to highlight the specific type of failure.

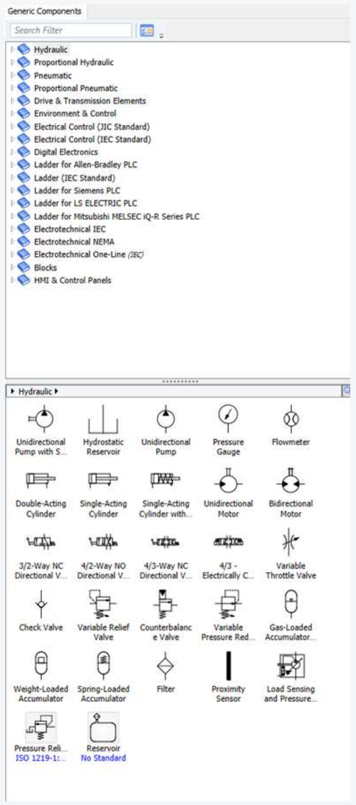
02 Automation Studio

- Develop the automatic control to implement the task in **Automation Studio programming**, using:
  - Ladder Code with the Step-Transition method
  - Grafcet code (subroutines in parallel, to manage different parts of the system)

- The code must be **well-organized**, with the use of **subroutines**:
  - FC
  - Partial GRAFCETs with various functionalities

- Moreover, develop a clear and understandable **HMI interface** in Automation Studio with:
  - Start
  - Stop
  - Emergency
  - Alarms
  - Conveyors
  - Objects
  - Cylinders and whole system animation (movements and visibility)
  - Sensors and pilot lights needed to simulate the real plant

02 Automation Studio



- We utilized the **Library Explorer** in Automation Studio to gather components and all necessary items, making use of different sections, in particular:

  - Pneumatic -> Actuators
  - Pneumatic –> Directional Valves
  - Pneumatic -> Flow lines & Connections
  - Pneumatic -> Sensors

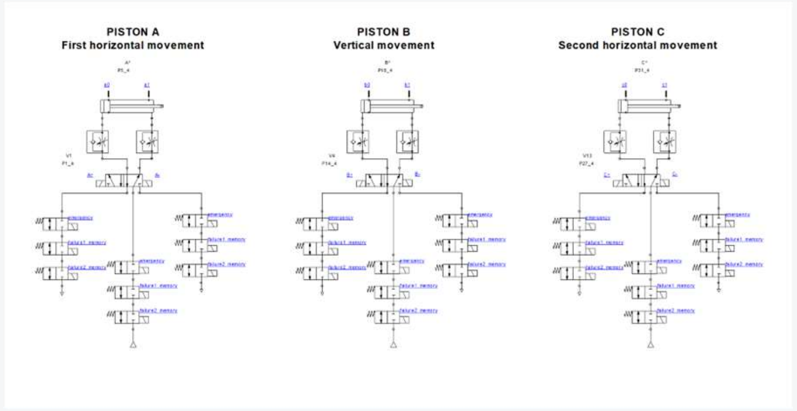
  - Electrical Control (JIC Standard) -> Power sources
  - Electrical Control (JIC Standard) -> Output components
  - Electrical Control (JIC Standard) -> Contact
  - Electrical Control (JIC Standard) -> Switches

  - Ladder for Siemens PLC -> Rung
  - Ladder for Siemens PLC -> Bit Logic
  - Ladder for Siemens PLC -> Timers
  - Ladder for Siemens PLC -> Program Management

  - HMI & Control Panels -> Control

03 Pneumatic

- We used <u>pneumatic pistons</u> to simulate the movements of the boxes, we linked each box to the linear movement of the respective piston:
  - The first piston simulates the first horizontal movement.
  - The second piston simulates the vertical movement.
  - The third one simulate the second horizontal movement.



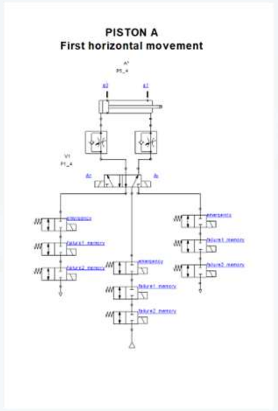
- For the pneumatic circuit we used:
  - 3x – **double acting cylinders**
  - 3x – **5/2 directional valve with 2 stable positions** (5 ways 2 position directional valves)
  - 27x – **2/2 directional valve with 1 stable position** (2 ways 2 position directional valves)
  - 6x – **sensors** (sensor ref. Bidirectional)
  - 6x – **restrictors**

- We controlled each cylinder by means of three 5 ways 2 stable position directional valves (5/2 directional valve) electrically switched by **solenoids** on both sides.

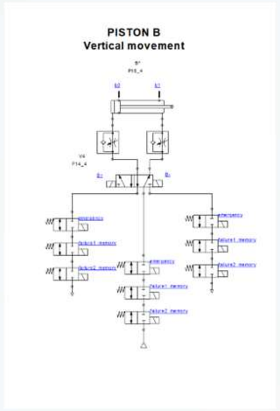
03 Pneumatic

Since we needed to follow a specific path, composed of 3 linear movements, specifically 2 horizontal and 1 vertical, we could link the translation animations of the box to the linear position of the respective piston.

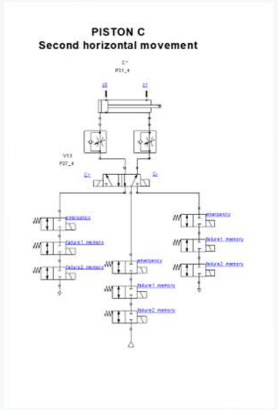
- First horizontal movement



- Vertical movement



- Second horizontal movement

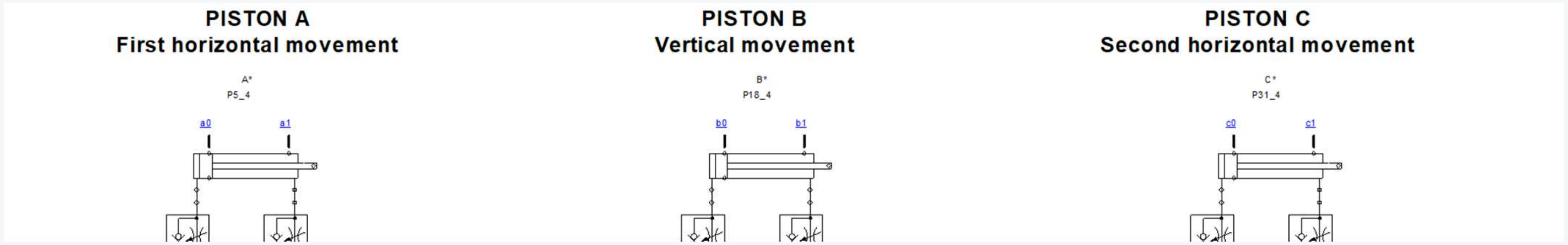


At the end we had 3 boxes:

- the first one will follow the linear displacement of the first piston
- the second one will do the same with the second piston
- same as the third one with the third piston.

We will handle the animation mixing the visibilities of the three boxes.

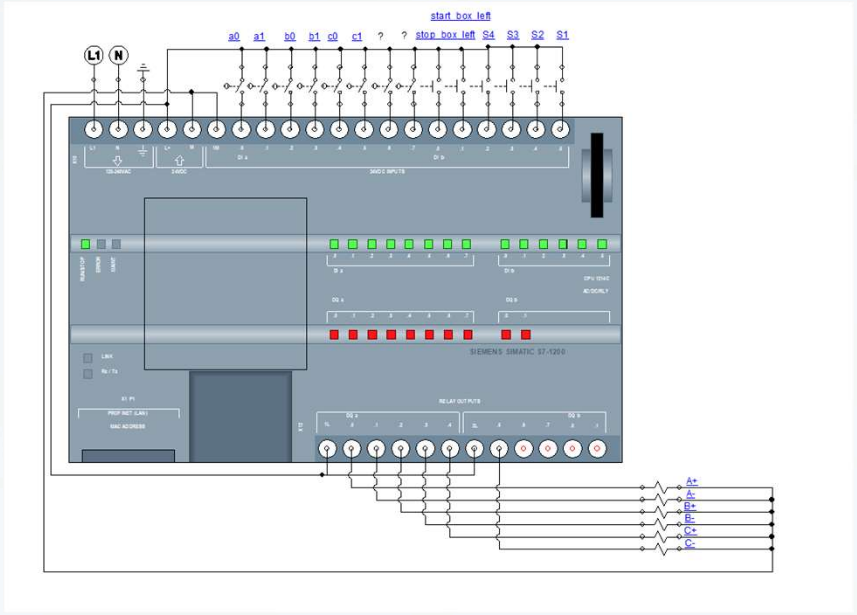
03 Pneumatic



Sensors:

• **a0**: detect the fully instroke piston A (A-)	-> it is activated when the piston A is in a rest position and fully retracted.
• **a1**: detect the fully outstroke piston A (A+)	-> it is activated when the valve of the piston A is switched and A is fully extended.
• **b0**: detect the fully instroke piston B (B-)	-> it is activated when the piston B is in a rest position and fully retracted.
• **b1**: detect the fully outstroke piston B (B+)	-> it is activated when the valve of the piston B is switched and B is fully extended.
• **c0**: detect the fully instroke piston C (C-)	-> it is activated when the piston C is in a rest position and fully retracted.
• **c1**: detect the fully outstroke piston C (C+)	-> it is activated when the valve of the piston C is switched and C is fully extended.

04 PLC



We used **Programmable logic controller (PLC)** to handle all the information regarding inputs and outputs and program actions related to a specific situation.

In particular, we used a **Siemens S7-1200** PLC with its library in Automation Studio.

To complete the connections, we used:

- **Normally-open proximity switch**
- **Push-Button Auxil contact**
- **Solenoids**

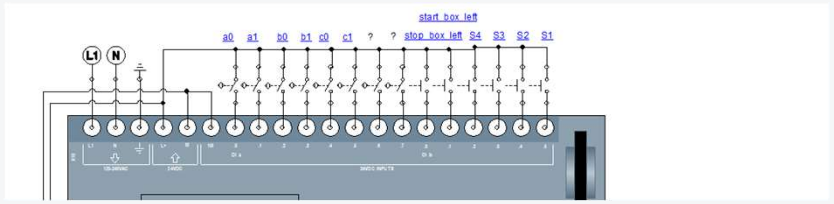
We connected the normally-open proximity switches to the sensors mounted in the pistons' rod: a0, a1, b0, b1, c0, and c1.
So whenever one of the sensors is activated, an electrical signal comes to the respective PLC's port and we can handle a specific task knowing where the signal is.

The Push-Buttons are linked to the HMI buttons and they command the start of a routine depending on different conditions.

The solenoids are the ones that take the signal to the valve to allow them to commutate and handle the outstroke or the instroke of a cylinder.

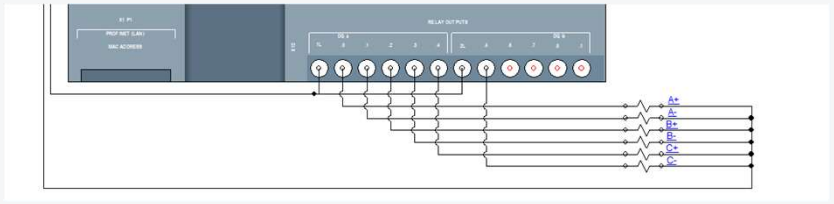
04 PLC

INPUTS



- **Sensor** inputs:
  - a0
  - a1
  - b0
  - b1
  - c0
  - c1
- **Button** inputs:
  - S1
  - S2
  - S3
  - S4
  - Start box left or right
  - Stop box left or right

OUTPUTS



- **Valve** outputs:

A+ :	A outstroke
A– :	A instroke
B+ :	B outstroke
B– :	B instroke
C+ :	C outstroke
C– :	C instroke

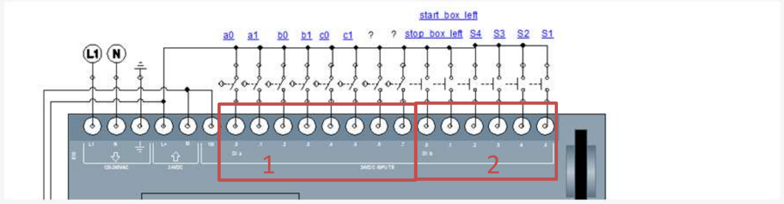
Once we have defined all the inputs in the upper part of the PLC and all the outputs in the bottom one, we moved these data into ladder Code to be able to manage all the system macro actions, routines and subroutines.

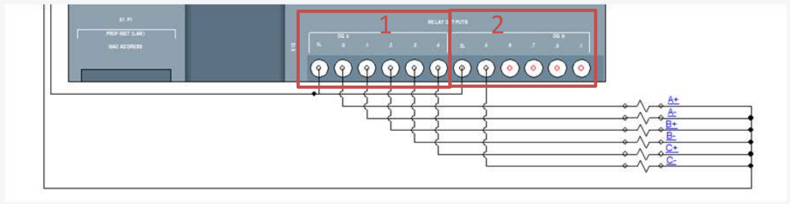
To do this we set the ladder code with normally-open/normally-closed contacts associated to the relative input in the PLC port.

04 PLC

The ports are structured in terms of **bytes**, **bits**, and **port names**:

- **Digital Inputs and Outputs (I/O)**
  - **Digital Inputs (I)**
    - Addressing:
      - Digital inputs are addressed by bytes and bits.
      - Each byte consists of 8 bits.
    - Port Naming:
      - Inputs are named starting with "I" followed by the byte and bit number.
      - Example: I0.0 (Input byte 0, bit 0), I1.3 (Input byte 1, bit 3).
  - **Digital Outputs (Q)**
    - Addressing:
      - Digital outputs are similarly addressed by bytes and bits.
    - Port Naming:
      - Outputs are named starting with "Q" followed by the byte and bit number.
      - Example: Q0.0 (Output byte 0, bit 0), Q1.7 (Output byte 1, bit 7).
- **Memory Areas**
  - **Memory Bits (M)**
    - Addressing:
      - Memory bits are addressed by bytes and bits.
    - Port Naming:
      - Memory bits are named starting with "M" followed by the byte and bit number.
      - Example: M0.0 (Memory byte 0, bit 0), M1.5 (Memory byte 1, bit 5).





Summary

Digital I/O:

Inputs: I0.0 to I1.7

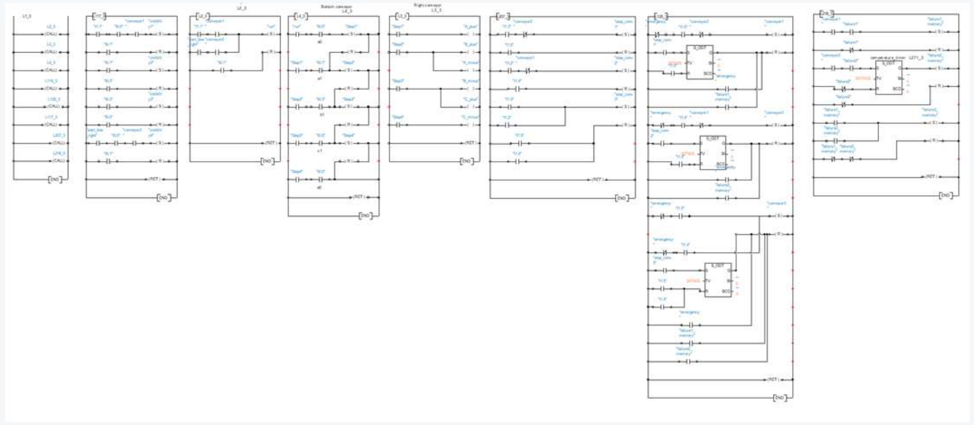
Outputs: Q0.0 to Q1.7

Memory:

Bits: M0.0, M1.0, etc.

<ins>This structured approach allows for precise control and monitoring of various processes.</ins>

05 Ladder Code



Ladder Diagram is a graphical programming language, it is used to develop software for programmable logic controllers (PLCs). It is one of the standard languages specifies for use with PLCs.

A program in ladder diagram notation is a circuit diagram that emulates circuits of relay logic hardware.

As already said in the PLC section, we moved all the information in Ladder Code.

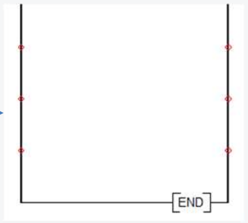
Here, we built different subsystem and subroutines to handle different scenarios:

- **Main**
    - Visibility
    - Run
    - Step-Transition
    - Piston movements
    - Stop
    - Emergency
    - Failures

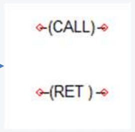
05 Ladder Code

To complete all the required actions and scenarios we used different components in the Ladder Code, in particular:

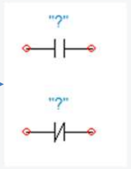
- **Rung** -> to create a horizontal flow handling different conditions.



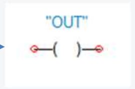
- **Call** –> to call a subsystem to start a subroutine
- **Return** -> to end a subroutine



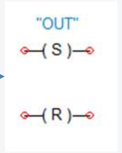
- **Normally open contact** -> associated to a variable, it closes the circuit if it is True
- **Normally closed contact** -> associated to a variable, it opens the circuit if it is True



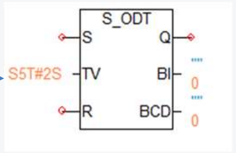
- **Coil** -> to activate a coil, for example a solenoid in a valve



- **Memory coil** -> to set or reset a memory



- **Timers** -> to delay actions, it is possible to define the delay time



05 Ladder Code

The first block (1) is the main one, it is responsible for calling the subsystems.

The first subsystem (1.1) is about the visibility of the boxes.
By means of normally-open switches and set-reset memories it is responsible for enabling the visibility of the boxes in the animation section.
For example, in each cycle only one box can be visible, so the others should be properly programmed to be invisible.
There are different situations where the boxes are moving simultaneously, so we need to create a smooth trajectory pretending to have only one box.

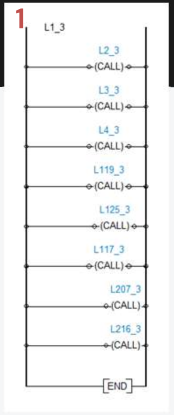
As the subsystem 1.1 shows we implemented three set-reset memories:

- Visibility1 -> can be set only if conveyor 2 is not active.
- Visibility2 -> can be set only if conveyor 1 is not active.
- Visibility3 -> is after either visibility1 or visibility2 but not at the same time.
- Visibility4 -> is the last one for the most bottom conveyor and it is after visibility3.

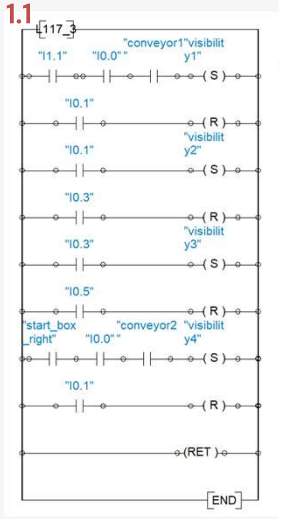
Each of them are reset after the fully extension of the associated piston.

The second subsystem (1.2) is about the start and stop of the three conveyors by means of three set-reset memories.
The start of conveyor 1 (conveyor 2) is related to both a push-button S1 (S2) and the not coexistence of the opposite conveyor 2 (conveyor 1).

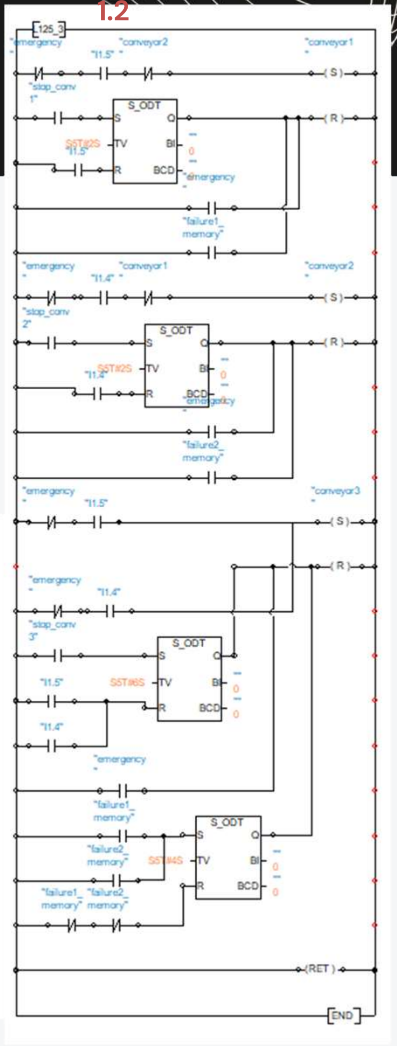
1



1.1



1.2


L125_3
"emergency" "I1.5" "conveyor2" "conveyor1" (S)
"stop_conv1" S_ODT S5T#2S (R)
"failure1_memory"
"emergency" "I1.4" "conveyor1" "conveyor2" (S)
"stop_conv2" S_ODT S5T#2S (R)
"failure2_memory"
"emergency" "I1.5" "conveyor3" (S)
"emergency" "I1.4" (R)
"stop_conv3" S_ODT S5T#6S
"failure1_memory" "failure2_memory" S_ODT S5T#4S
(RET)
END

05 Ladder Code

Here we have **the Step-Transition method** to compute the routine of the conveyors and the boxes.

First of all, we have the **RUN** memory, implemented in the <u>subsystem 1.3</u>, that allows to start the loop composed by:

- Step1
- Step2
- Step3
- Step4

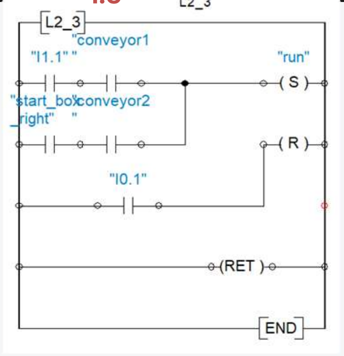
Run is set only if both the push-button is pressed and the conveyor associated to that side is already turned on.

After that we associated the steps to the variable of the PLC output that will move the respective valve and consequently the piston (<u>subsystem 1.4</u>).

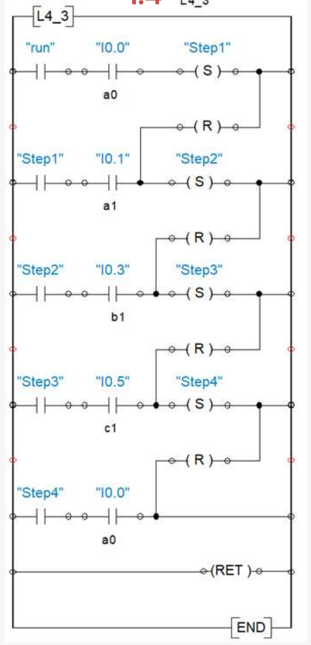
<u>Subsystem 1.5:</u>

Step1:	A+	-> outstroke of piston A
Step2:	A-/B+	-> instroke of piston A and outstroke of piston B
Step3:	B-/C+	-> instroke of piston B and outstroke of piston C
Step4:	C-	-> instroke of piston C

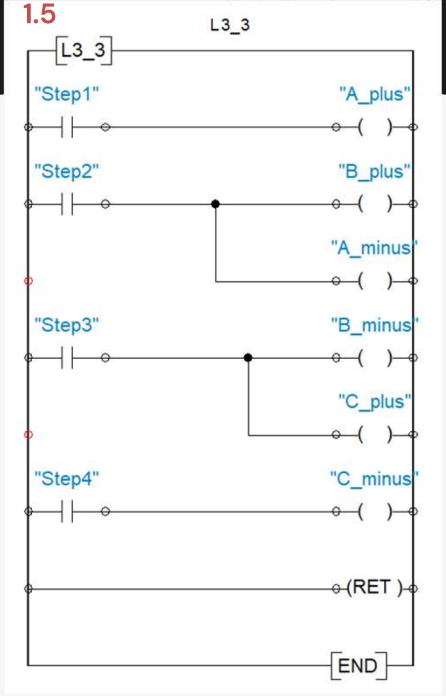
1.3



1.4



1.5



05 Ladder Code

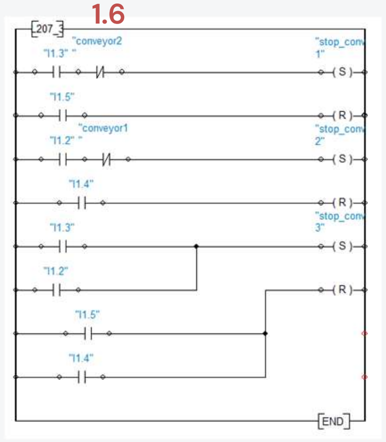
The stop and emergency phases are handled by the last two subroutines 1.6 and 1.7.

The stop of conveyor 1 (conveyor 2) is related to the push-button S3 (S4) only if the opposite conveyor 2 (conveyor 1) is turned off (subsystem 1.6).
After that, a memory is set and take the signal to the last subroutine, where it starts a timer with a predefined time to reset the memory associated to the start of the conveyor (subsystem 1.7).

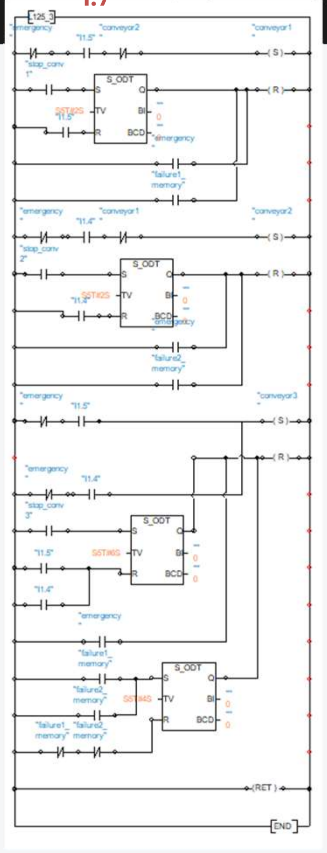
The **emergency** is handled in every scenario but in different ways:

- In the set memory side:
  putting a **normally-closed contact** before every action, so whenever the emergency selector is activated the normally-closed opens the circuit preventing the signal pass, for example activating memory.
- In the reset memory side:
  putting a **normally-open contact** in parallel with the other conditions the reset the memory, so whenever the emergency selector is activated the normally-open closes the circuit making the signal pass to reset the memory, for example to stop the conveyors

1.6



1.7



05 Ladder Code

The failure scenarios are handling in the last subsystem (subsystem 1.8).

We defined to types of failure:

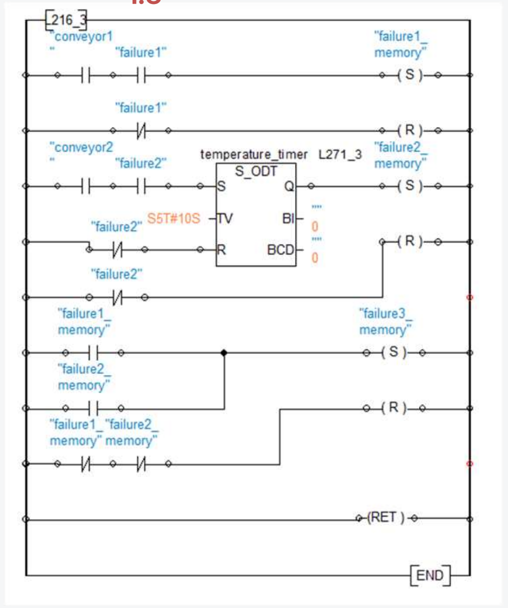
- **First failure:**
  Generic instantaneously failure of conveyor 1 and conveyor 3.
  The **sensor B1** detects a generic failure and stops conveyor 1 instantaneously and conveyor 3 after 4 seconds.

- **Second failure:**
  Failure of conveyor 2 and conveyor 3 are related to the over temperature.
  The **sensor B2** detects an over temperature failure and stops conveyor 2 when the temperature reaches 80 °C and conveyor 3 after 4 seconds.

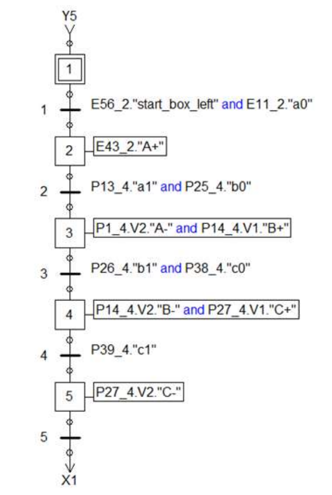
Everything is clearly visible and understandable from the HMI.

Each failure can be simulated by switching one of the two selectors in the failure section of the HMI.

1.8



06 GRAFCET


Y5
1
1
E56_2."start_box_left" and E11_2."a0"
2
E43_2."A+"
2
P13_4."a1" and P25_4."b0"
3
P1_4.V2."A-" and P14_4.V1."B+"
3
P26_4."b1" and P38_4."c0"
4
P14_4.V2."B-" and P27_4.V1."C+"
4
P39_4."c1"
5
P27_4.V2."C-"
5
X1

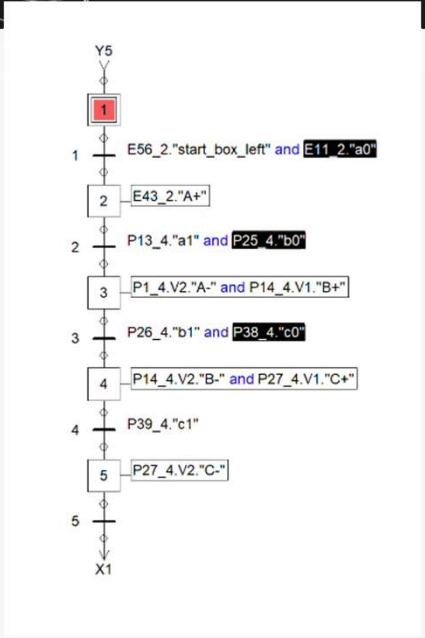
GRAFCET (GRAphe Fonctionnel de Commande Etape Transition) is a graphical programming language used for designing and implementing sequential control systems in Programmable Logic Controllers (PLCs).
It visually represents the states and transitions of an automated process, making it easier to understand and debug complex sequences.

It is very important, especially for:

- **Clarity and Structure**: GRAFCET provides a clear, step-by-step representation of the process, enhancing comprehension and communication among team members.
- **Ease of Troubleshooting**: The graphical format simplifies identifying and resolving issues in the control logic.
- **Standardization**: As an IEC 60848 standard, GRAFCET ensures uniformity in programming across different systems and industries.

Using GRAFCET in our projects helped streamline the development process, improved reliability, and reduced downtime due to more efficient maintenance and troubleshooting.

06 GRAFCET



Y5
1
1
E56_2."start_box_left" and E11_2."a0"
2
E43_2."A+"
2
P13_4."a1" and P25_4."b0"
3
P1_4.V2."A-" and P14_4.V1."B+"
3
P26_4."b1" and P38_4."c0"
4
P14_4.V2."B-" and P27_4.V1."C+"
4
P39_4."c1"
5
P27_4.V2."C-"
5
X1

After building every scenarios in the ladder code we moved the pistons' movements part in the GRAFCET. We built the GRAFCET of the conveyors and box together.

Structure of GRAFCET:

1. Steps (Etapes) and Actions (Actions)

- We represented the different states in the control process, each step is represented by a square or rectangle, from 1 to 6.
- We defined the operations to be performed when a specific step is active:

• 2:	A+
• 3:	A-/B+
• 4:	B-/C+
• 5:	C-

2. Transitions (Transitions) and Conditions (Conditions)

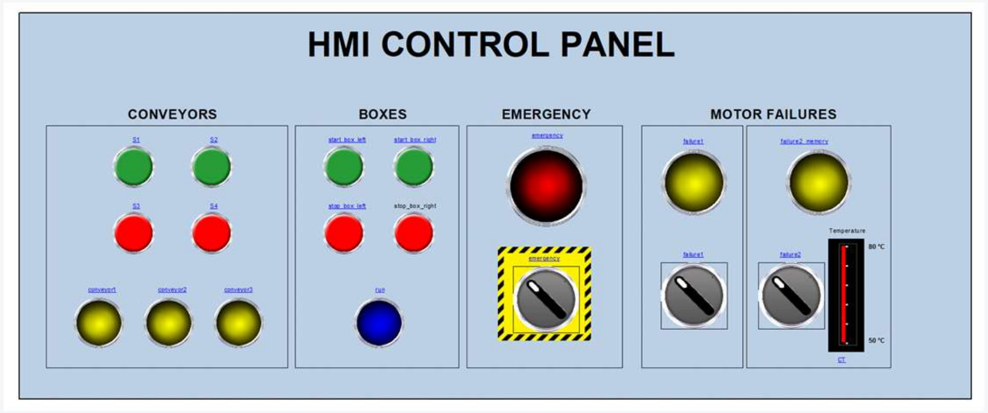
- We indicated the conditions under which the process moves from one step to another, they are placed between steps to show the flow from one state to another, represented by horizontal lines with a condition or event that must be satisfied for the transition to occur:

• 1:	start_box_left + a0
• 2:	a1 + b0
• 3:	b1 + c0
• 4:	c1

3. Initial Step (Initialisation)

- The starting point of the GRAFCET diagram, it is represented by a double square and it indicates where the control process begins.

07 HMI



As far as the **Human-Machine Interface (HMI)** is concerned, we built a simple panel composed of **four sections** from where we can control the whole system.

The four sections are:
1. Conveyors
2. Boxes
3. Emergency
4. Motor failures

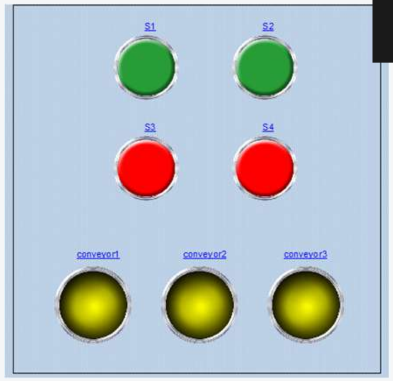
We used:
- Push-buttons
- Selectors
- Lamps

Human-Machine Interface (HMI) is very important in order to make a connection between programming side and animations.

Thanks to that we can have a representation, physically simulated, to what the system should do in the real world if everything was in these conditions.

07 HMI

CONVEYORS



According to the requirements the push-buttons are responsible to start the conveyor 1 (**S1**), start the conveyor 2 (**S2**), stop the conveyor 1 (**S3**), stop the conveyor 2 (**S4**).

The lamps indicate the status of the conveyors (yellow lamp: **H1**, **H2**, **H3**).

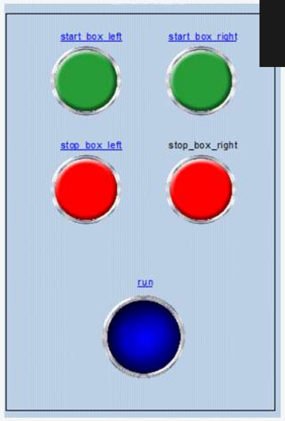
When a push-button is pressed it sends a signal to the PLC port where it is defined, then we can move that signal in the ladder code end generate actions.

When S1 is pressed the motor 1 is activated and the conveyor 1 starts to move. Simultaneously, the motor 3 with the conveyor 3 also starts to move. The lamp H1 and H3 turned on to indicates that they are moving.

The same happens with S2, but here instead of motor 1 there is motor 2 with conveyor 2. The motor 3 with conveyor 3 is activated at the same way. The lamp H2 and H3 turned on to indicates that they are moving.

S3 and S4, as already said, are used to stop respectively conveyor 1 and conveyor 2.
When one of them is pressed, a timer starts in the ladder and stops the motor and the conveyor after 20 seconds.
Subsequently, motor 3 and conveyor 3 turn off automatically after 60 seconds.
The lamps are turned off if anything is moving.

07 HMI



BOXES

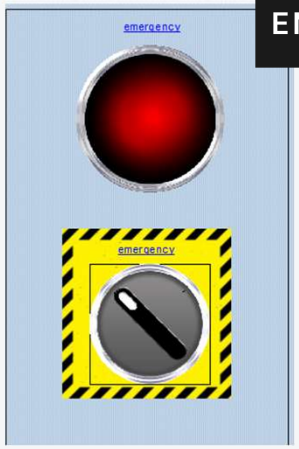
We wanted to add a section regarding the movements of boxes on the conveyors to have a more realistic system.

In this section there are **start_box_left** and **start_box_right** to start the cycle with a box and the corresponding buttons to stop them.

So, we have four push-button and a lamp that allows us to know the status of the box cycle, if it is active or not (blue lamp: **run**).

The green push-button can be pressed only if the conveyor associated is already turned on.
For example, we can start the cycle with the box form the left only if the conveyor 1 is active.
The same is for the other side but with the conveyor 2.

07 HMI



EMERGENCY

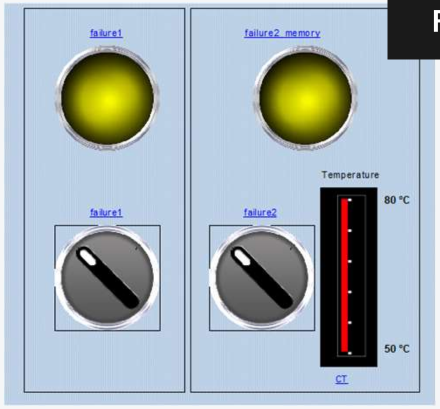
The **emergency selector** is used in emergency scenario to stop all the conveyors instantaneously by means of a selector.

We also have a lamp that allows us to check the emergency status (red lamp: **emergency**).

The emergency is essential and it has the highest priority.

Whenever the selector is switched the whole system is stopped, all the conveyors are stopped, all the boxes are stopped. We can then activate again the conveyors and start again the cycles with the boxes only if the switch is placed in the original position.

07 HMI

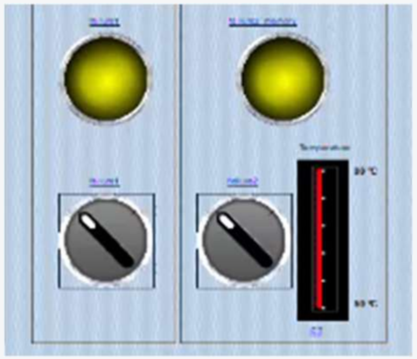


FAILURES

The **FAILURES** are handled by the last section of the HMI.

We have <u>two selectors</u> that simulate two different failure scenarios.

- **Failure 1:**
  - Unknown cause, generic failure.
  - It stops the conveyor 1 instantaneously and the conveyor 3 after 4 seconds.
  - The red lamps on the conveyors are turned on.
  - The yellow lamp in the HMI starts blinking.



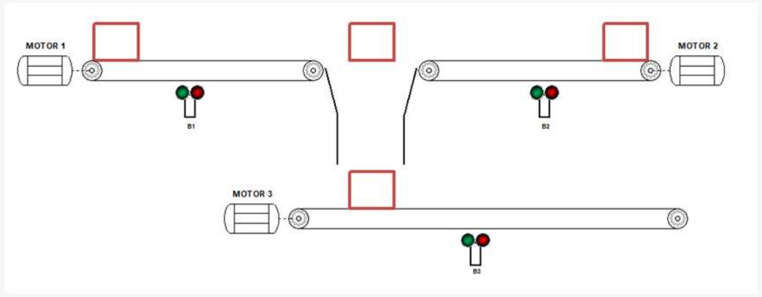
- **Failure 2:**
  - Temperature limit is reached.
  - After 10 seconds the temperature reaches 80 °C, so to avoid motor damages it stops the conveyor 1 instantaneously and the conveyor 3 after 4 seconds.
  - The red lamps on the conveyors are turned on.
  - The yellow lamp in the HMI starts blinking.

08 Animations

As last step we built the model graphically to be able to create animations and test the system.

We created everything form scratch, with lines, round and square shapes, in particular we drew:

- 4 boxes
- 3 motors
- 6 gear wheels
- 3 sensors
- 3 buttonhole
- 1 funnel



After drawing everything we set the line width and the colors.

Then we put everything together in order to make a clear system, visible and understandable.

To be able to control the animations with the routine programmed with Ladder code and GRAFCET we had to connect the objects with variables pre-defined.

We worked with two types of animations:
- Visibility
- Translation

08 Animations

We have four boxes in the workspace always together, we had to work on their visibility in order to show one box at time.

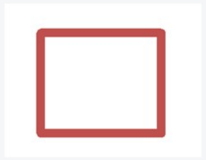
The translation is linked to the linear movement of the pistons, as already said, the only movement difference between the two cycle (box from the right and box from the left) is the first one, where the linear movement is positive for the one from the left and negative for the one from the right.

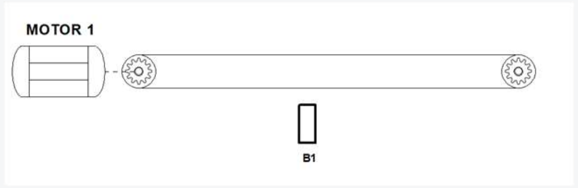
The visibility is related to the sensors on the pistons:

- a0:
  visibility box 1 ON
  visibility box 2 OFF
  visibility box 3 OFF
- a1:
  visibility box 1 OFF
  visibility box 2 ON
  visibility box 3 OFF
- b1:
  visibility box 1 OFF
  visibility box 2 OFF
  visibility box 3 ON
- c1:
  visibility box 1 OFF
  visibility box 2 OFF
  visibility box 3 OFF

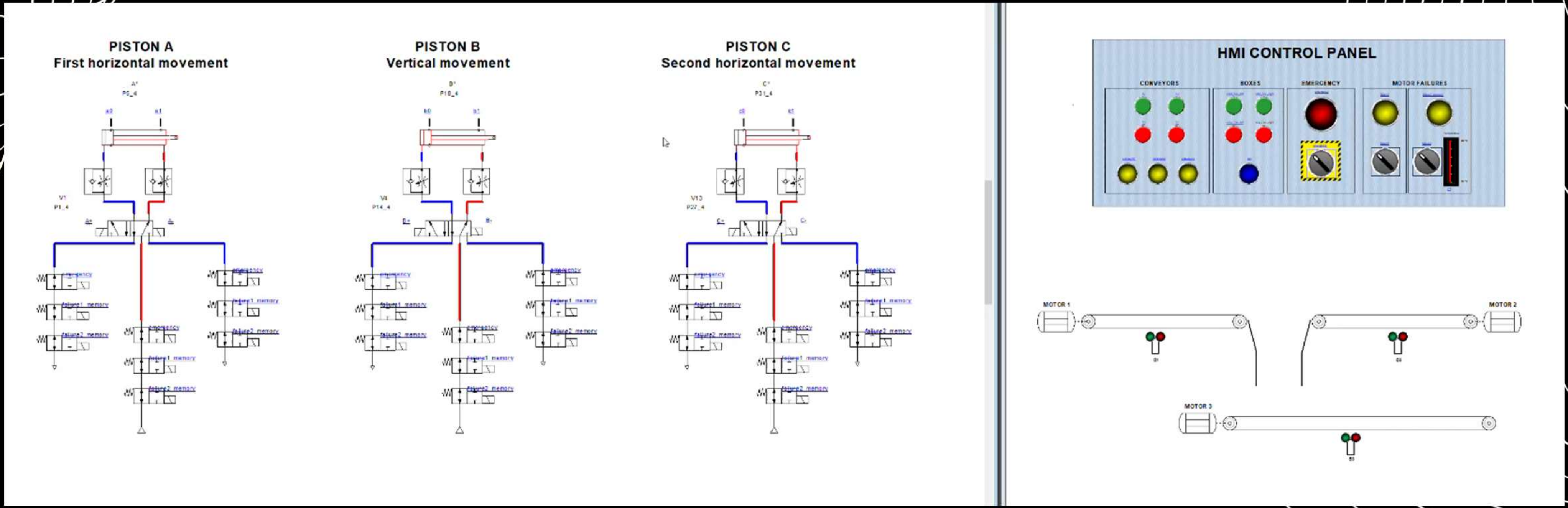
The translation is related to the linear movement of the pistons:

- A+:
  Horizontal movement of box 1
- B+:
  Vertical movement of box 2
- C+:
  Horizontal movement of box 3

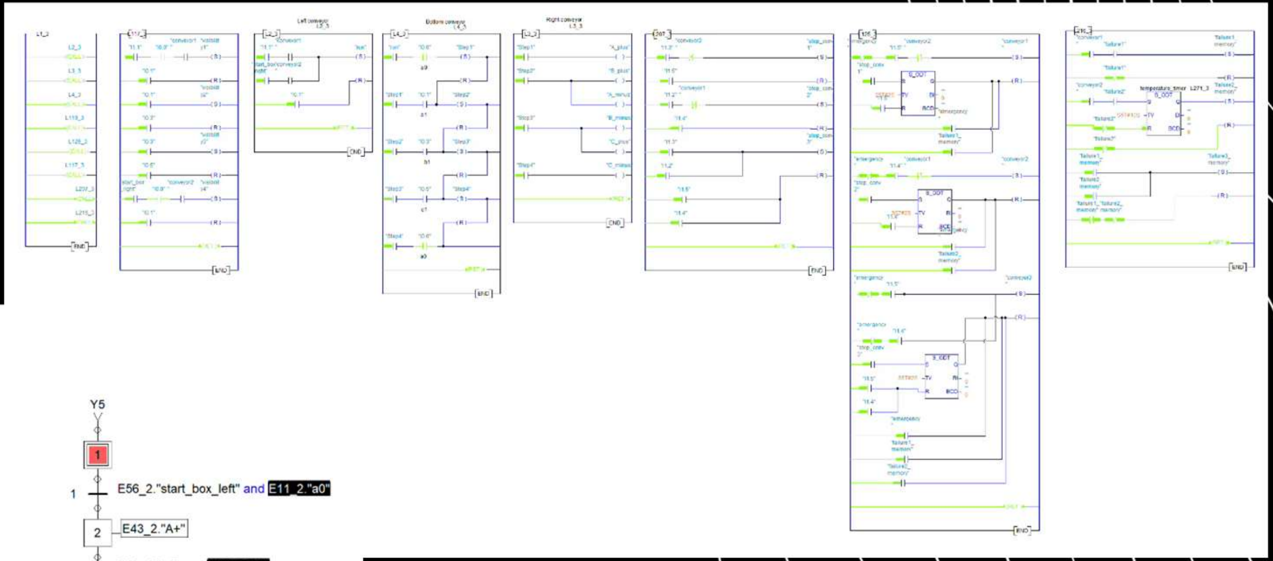


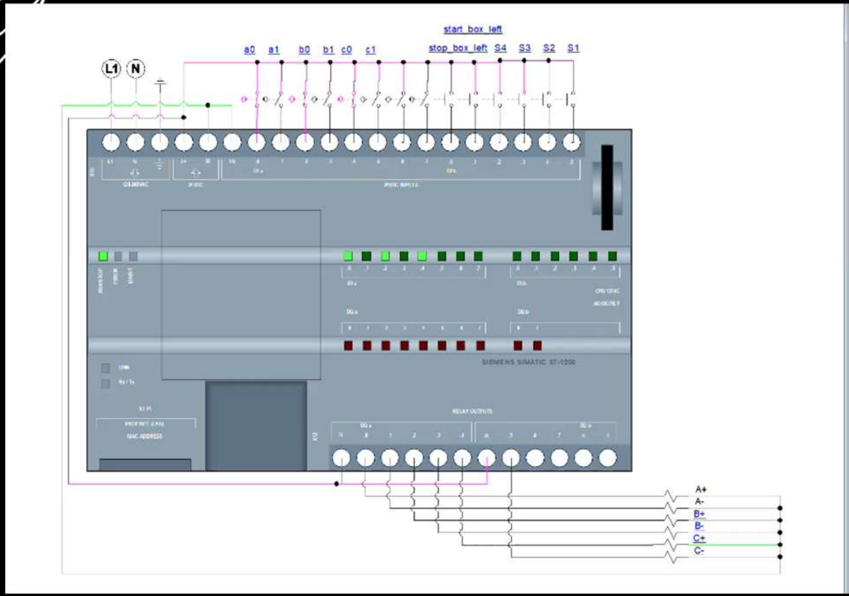


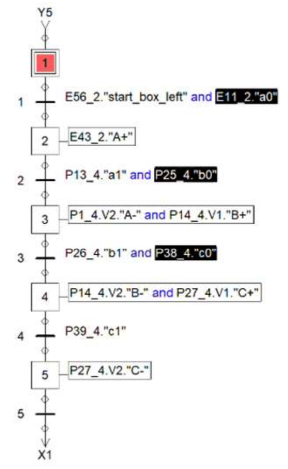
FINAL RESULT



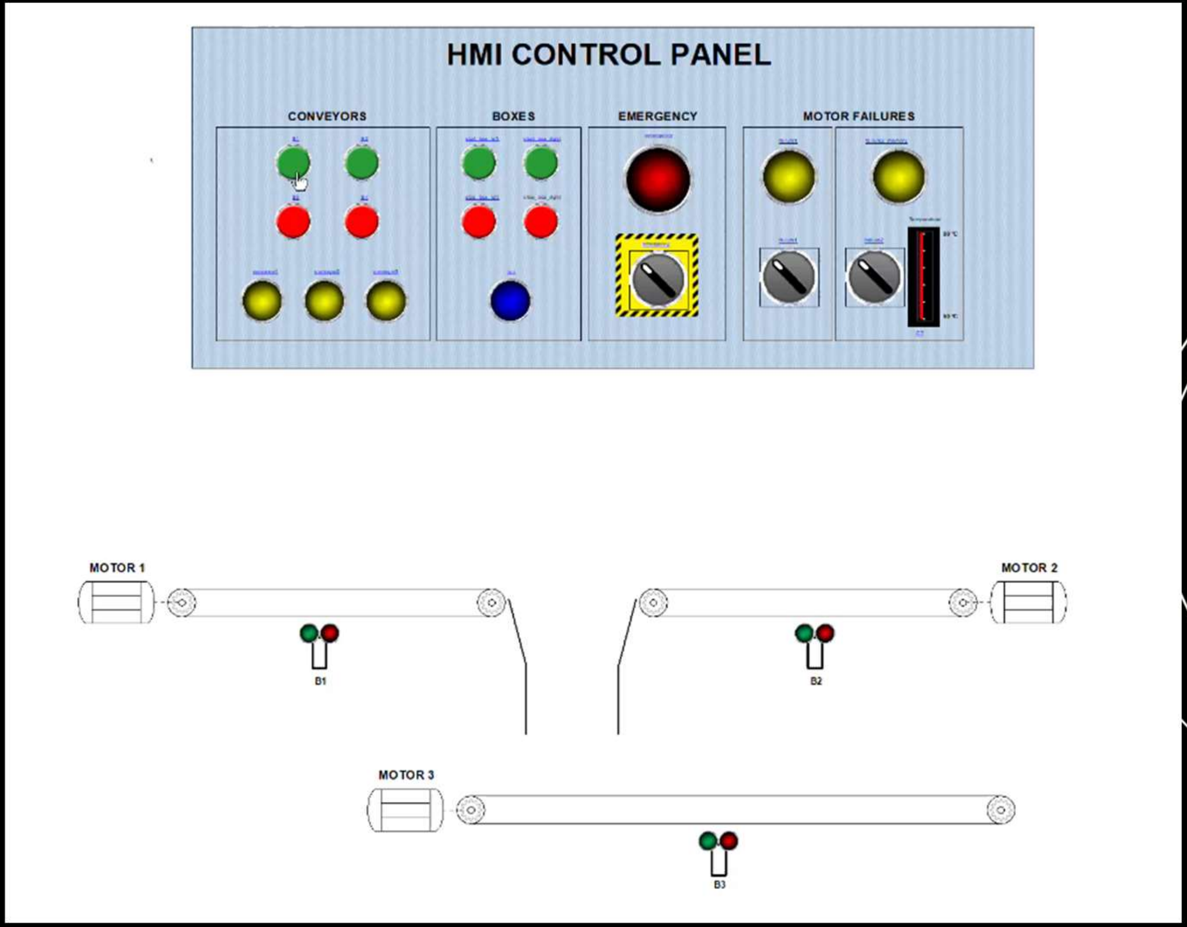
FINAL RESULT






Y5
1
E56_2."start_box_left" and E11_2."a0"
2
E43_2."A+"
P13_4."a1" and P25_4."b0"
3
P1_4.V2."A-" and P14_4.V1."B+"
P26_4."b1" and P38_4."c0"
4
P14_4.V2."B-" and P27_4.V1."C+"
P39_4."c1"
5
P27_4.V2."C-"
X1

FINAL RESULT



OUR TEAM

Luigi Muratore

s333098

Muhammad Fatir Noshab

s331898

Iskandar Akbarov

s329650

THANKS FOR WATCHING

Politecnico di Torino