FLUID AUTOMATION

TASK 2

Automatic conveyor system

Students:

Muratore Luigi s333098
Akbarov Iskandar s329650
Muhammad Fatir Noshab s331898

Professor:

Luigi Mazza

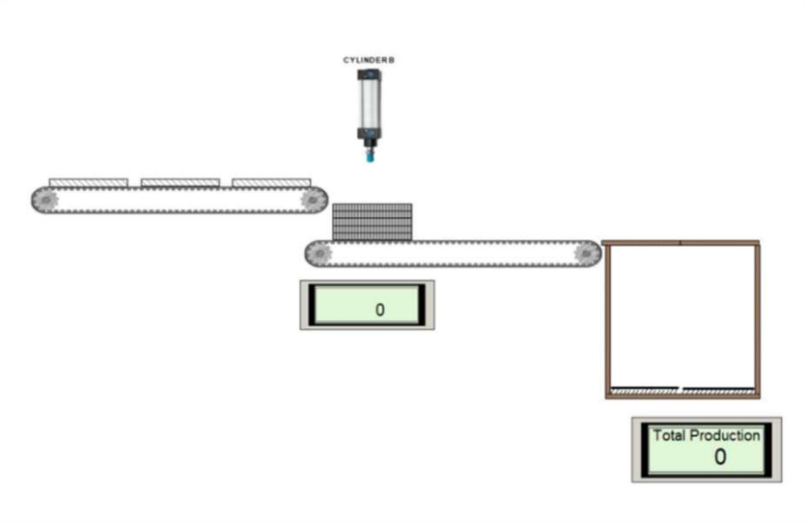
Politecnico di Torino

CONTENT

01 Requirements

Develop an automatic conveyor system to process and store steel plates coming from a steel mill.

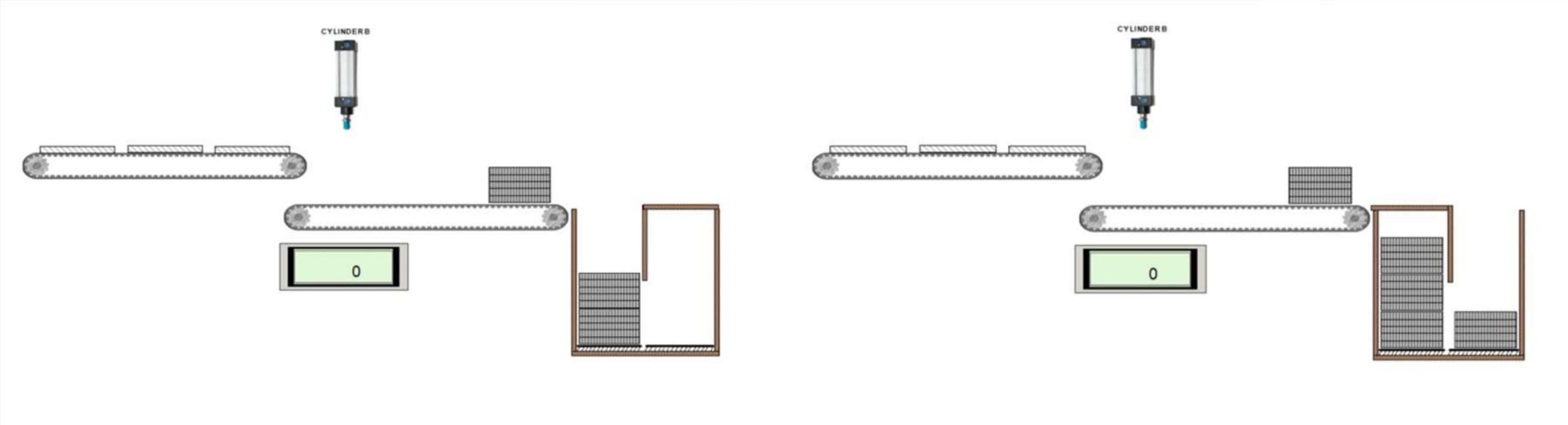
The top conveyor can transport three steel plates simultaneously, which are dropped onto a second conveyor to form a stack.
The operator shall be able to change the number of plates that each stack is made of.
As the stack is complete, the top conveyor stops to allow for the extension of a pneumatic cylinder.
Such cylinder presses on the stack for 3 seconds before retracting.
The stack is then transported to a wooden crate for shipping, while the top conveyor is reactivated to build the next stack.



The incoming plates must be loaded on the first conveyor according to a clock signal generated by a Pulse Timer.
No more than three plates can be loaded at the same time due to weight limits.
Develop an automatic conveyor system to process and store steel plates coming from a steel mill.

01 Requirements

The wooden crate at the end can store the stacks of plates in **two columns**, with a maximum of **three stacks each**.
Two openings at the top allow for the stacks to be lowered in the correct column: as soon as the first column is full, the second opening is activated to direct the incoming stacks to the second column, while the first opening is closed off.
When the whole crate is full, it is ready to be shipped so it must be transported to the designated storage area of the plant.



Goals and Objectives

Develop the automatic control to implement the task with:

- CODESYS (programming)
- Fluid Sim (plant)

Using:

- Ladder
- ST languages (in proper well-organized FB– ladder and ST, and FUN– in ST).

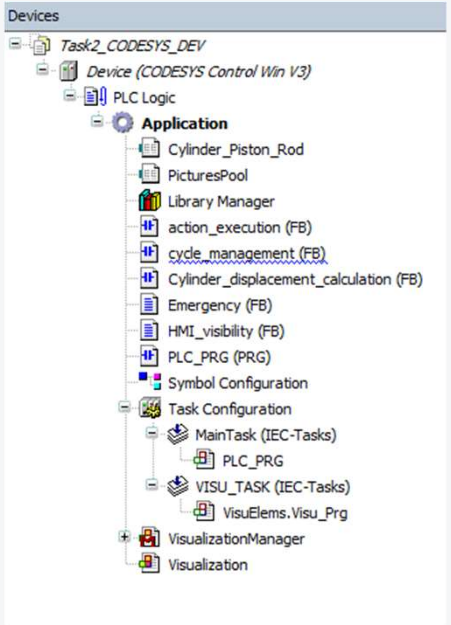
Moreover, develop a clear and understandable HMI interface CODESYS with:

- Start
- Stop
- Emergency
- Alarms
- Conveyors
- Storage station
- Cylinders
- Sensors
- Pilot lights

Goals and Objectives

In particular, develop a well-organized code that uses POUs in LD (ladder) and structured text (ST) with batch method including:

- Main POU (Start, Stop, general commands)
- FBs with I/O communications signals for:
    - Cycle management (LD)
    - Actions execution (LD)
    - Emergency (ST)
    - HMI visibility (ST)
- FUN in ST for cylinders and objects animation



02 FluidSIM

FluidSIM *is a simulation software developed by* ***Festo*** *Didactic, designed for the creation and analysis of pneumatics, hydraulics, and electrical circuits.*
It is widely used in education and training, as well as in industry, to simulate and study fluid and control systems.

Key Features:

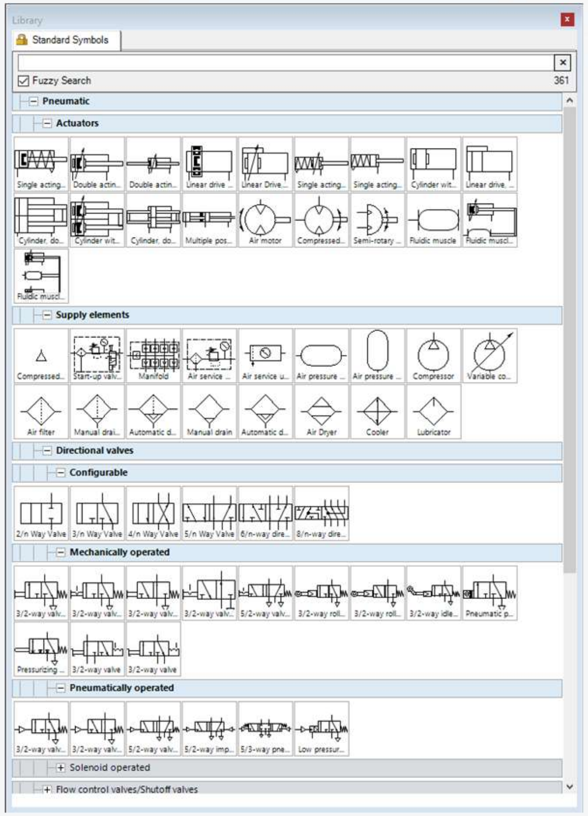
- **Simulation Environment**: Allows users to design and simulate fluid power and electrical circuits.
- **Interactive Circuit Design**: Provides an intuitive drag-and-drop interface for building complex systems.
- **Real-Time Simulation**: Simulates the dynamic behavior of components in real-time, giving feedback on system performance.
- **Component Library**: Offers an extensive library of pre-built components, including valves, cylinders, sensors, and motors.
- **Detailed Analysis**: Allows users to analyze the behavior of individual components and overall system performance.
- **Reduced Testing Costs**: By simulating systems before physical testing, it minimizes the need for costly prototypes and hardware.
- **Integration with PLCs**: Can be connected to external PLCs for hardware-in-the-loop (HIL) simulations.

02 FluidSIM - Library

- We utilized the **Library Explorer** in FluidSIM to gather components and all necessary items, making use of different sections, in particular:
  - Pneumatic -> Actuators
  - Pneumatic –> Directional Valves
  - Pneumatic -> Flow control valves
  - Pneumatic -> Sensors

  - Electrical Controls (IEC Standard) -> Power supply
  - Electrical Controls (IEC Standard) -> Output components
  - Electrical Controls (IEC Standard) -> Switches and Contact

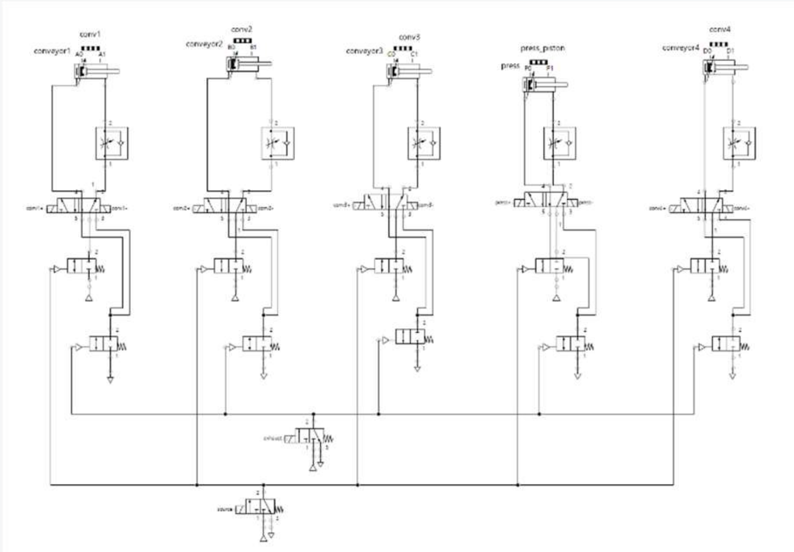
  - EasyPort/OPC/DDE



03 FluidSIM - Pneumatic

- We used pneumatic pistons 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 second horizontal movement.
  - The third piston simulates the third horizontal movement.

  - The fourth piston simulates the vertical movement of the press.
  - The fifth one simulate the fourth horizontal movement.

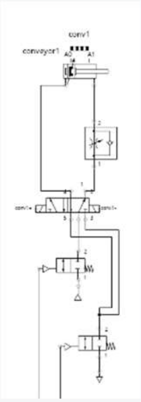


- For the pneumatic circuit we used:
  - 5x – **double acting cylinders**
  - 5x – **5/2 directional valve with 2 stable positions**
  - 10x – **2/2 directional valve with 1 stable position**
  - 2x – **3/2 directional valve with 1 stable position**
  - 10x – **sensors** (sensor ref. Bidirectional)
  - 5x – **restrictors**
- We used 5/2 directional valves electrically switched by **solenoids** on both sides to control each cylinder.
- We used 3/2 and 2/2 directional valves to manage the emergency and stop mode.

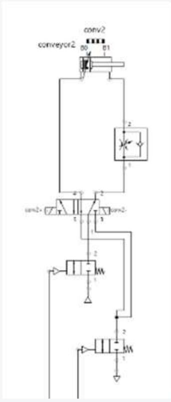
03 FluidSIM - Pneumatic

Since we needed to follow a specific path, composed of 5 linear movements, specifically 4 horizontal and 1 vertical, we could link the translation animations of each box to the linear position of the respective piston.

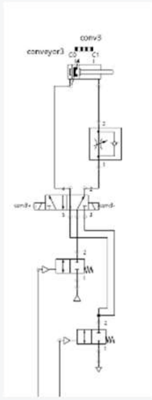
- 1 horizontal



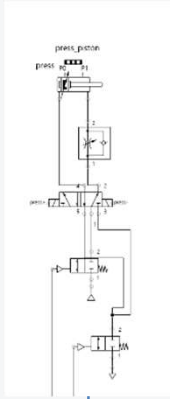
- 2 horizontal



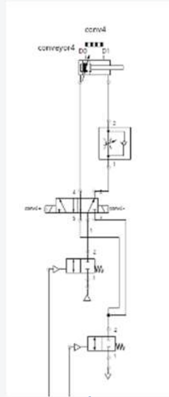
- 3 horizontal

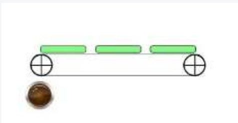


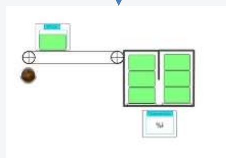
- 4 Vertical



- 5 horizontal

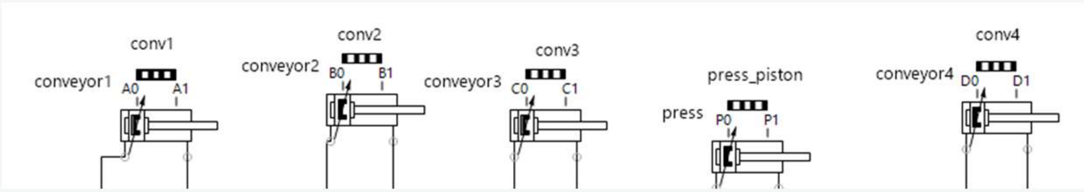






We will handle the animation mixing the visibilities of the four boxes in CODESYS.

03 FluidSIM - Sensors



Limit Switch:

- **A0**: detect the fully instroke piston *conveyor1* -> it is activated when the piston is in a rest position and fully retracted.
- **A1**: detect the fully outstroke piston *conveyor1* -> it is activated when the valve of the piston is switched and it is fully extended.

- **B0**: detect the fully instroke piston *conveyor2* -> it is activated when the piston is in a rest position and fully retracted.
- **B1**: detect the fully outstroke piston *conveyor2* -> it is activated when the valve of the piston is switched and it is fully extended.

- **C0**: detect the fully instroke piston *conveyor3* -> it is activated when the piston is in a rest position and fully retracted.
- **C1**: detect the fully outstroke piston *conveyor3* -> it is activated when the valve of the piston is switched and it is fully extended.

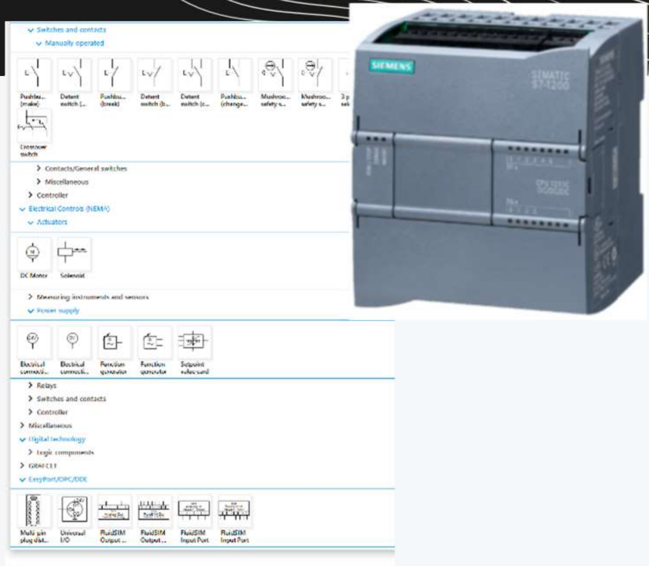
- **P0**: detect the fully instroke piston *press* -> it is activated when the piston is in a rest position and fully retracted.
- **P1**: detect the fully outstroke piston *press* -> it is activated when the valve of the piston is switched and it is fully extended.

- **D0**: detect the fully instroke piston *conveyor4* -> it is activated when the piston is in a rest position and fully retracted.
- **D1**: detect the fully outstroke piston *conveyor4* -> it is activated when the valve of the piston is switched and it is fully extended.

Displacement encoders:

- **conv1**: provide motion feedback for linear measurement of the movement of piston *conveyor1*
- **conv2**: provide motion feedback for linear measurement of the movement of piston *conveyor2*
- **conv3**: provide motion feedback for linear measurement of the movement of piston *conveyor3*
- **press_piston**: provide motion feedback for linear measurement of the movement of piston *press*
- **conv4**: provide motion feedback for linear measurement of the movement of piston *conveyor4*

04 FluidSIM - PLC



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

To complete the connections in FluidSIM, we used:

- **Electric connections and lines**
- **Make switch**
- **Push-Button**
- **Solenoids**
- **Displacement encoder**
- **FluidSIM Input Port**
- **FluidSIM Output Port**

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

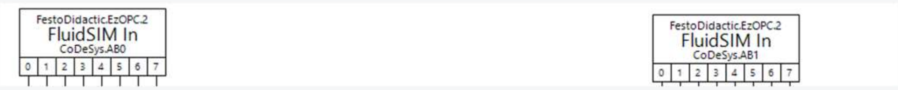
Digital Inputs (AB)

Addressing:

- Digital inputs are addressed by bytes and bits.
- Each byte consists of 8 bits.

Port Naming:

- Starting with “AB" followed by the byte and bit number.
- Example: AB1.3 (Input byte 1, bit 3).



Digital Outputs (EB)

Addressing:

- Digital outputs are also addressed by bytes and bits.

Port Naming:

- Starting with “EB" followed by the byte and bit number.
- Example: EB1.7 (Output byte 1, bit 7).

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

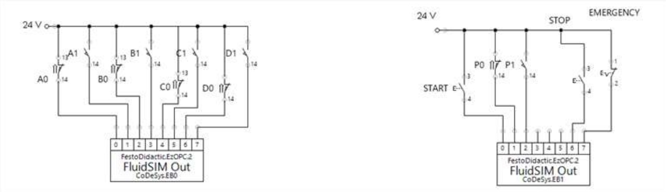
04 FluidSIM - PLC

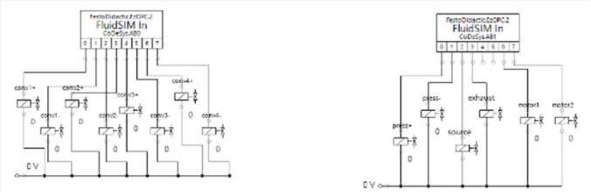
We connected the normally-open make switches to the sensors mounted on the pistons' rod: *A0, A1, B0, B1, C0, C1, P0, P1, D0* and *D1*.
So whenever one of the sensors is activated, an electrical signal comes to the respective PLC's port and we can handle a specific task knowing where the signal is.

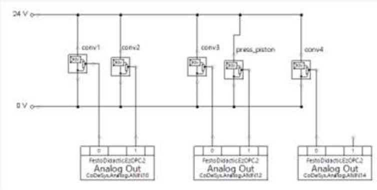
The solenoids are the ones that take the signal to the valve to allow them to commutate and handle the outstroke or the instroke of a cylinder.
They allow us to also control the valve at the beginning of the circuit to let the air enter into the circuit or the block it in the emergency condition.

The displacement encoder take us data related to the position of the piston and so allows us to move the boxes in the simulated environment.

Once we have defined all the inputs and all the outputs, we passed these data to CODESYS to be able to manage all the system macro actions, routines and subroutines.







- **FluidSIM OUT:**
  - **Push button and limit switch**
    - A0/A1
    - B0/B1
    - C0/C1
    - D0/D1
    - P0/P1
    - START
    - STOP
    - EMERGENCY

- **FluidSIM IN:**
  - **Solenoids:**
    - conv1+ / conv1-
    - conv2+ / conv2-
    - conv3+ / conv3-
    - conv4+ / conv4-
    - press+ / press-
    - source
    - exhaust
    - Motor 1 / Motor 2

- **FluidSIM Analog OUT:**
  - **Displacement encoder:**
    - conv1
    - conv2
    - conv3
    - press_piston
    - conv4

05 CODESYS

CODESYS *(Controller Development System) is a powerful, IEC 61131-3 compliant software for industrial automation.*
It is widely used in industries such as manufacturing, process automation, energy, and building automation.

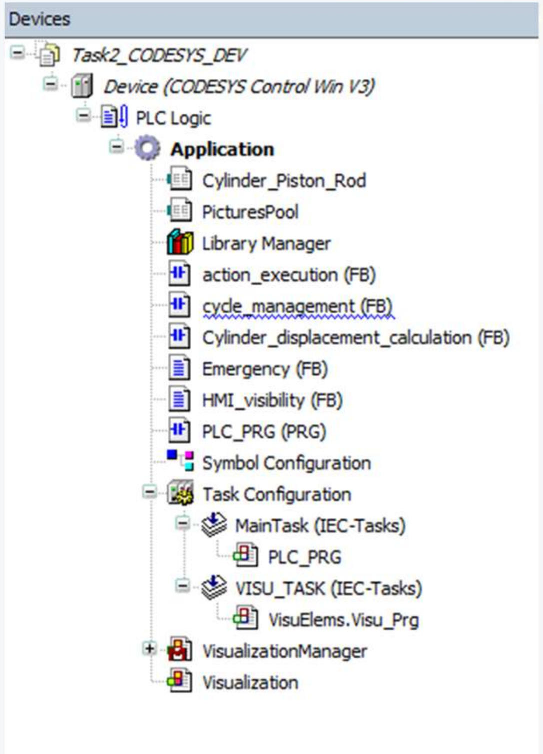
Key Features

- **Multi-language Support:**
    - Ladder Diagram (LD)
    - Structured Text (ST)
    - Function Block Diagram (FBD)
    - Sequential Function Chart (SFC)
    - Instruction List (IL)

- **Cross-Platform Compatibility**: Supports a wide range of hardware platforms, including PLCs, HMIs, and soft PLCs.
- **Real-Time Control**: Enables real-time, deterministic control of industrial processes.
- **Integrated Development Environment (IDE):** Offers tools for coding, simulation, and debugging in one unified environment.
- **Flexibility**: Adaptable to various hardware and application types.
- **Scalability**: Suitable for small to large-scale automation projects.

06 CODESYS-POUs

*In CODESYS, a **POU (Program Organization Unit)** is a core element used to organize and structure the program. It refers to individual programming units that include Programs, Functions, and Function Blocks.*



Each POU can be written in any of the IEC 61131-3 languages and serves different purposes:

- **Program**: Main unit where code execution starts; defines the primary control logic.
- **Function**: A reusable block that returns a single output based on given inputs, like a mathematical operation.
- **Function Block**: A reusable unit with internal memory, used for complex operations or control logic, such as managing timers or counters.

POUs help modularize the code, making it easier to develop, test, and maintain complex automation projects.

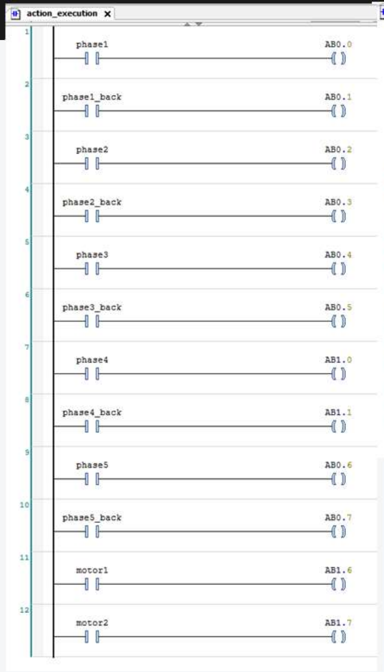
We created six different POUs based on the requirements of our projects:

- ***Main PLC_PRG***
- *Action execution*
- *Cycle management*
- *Cylinder displacement calculation*
- *Emergency*
- *HMI visibility*

06 CODESYS – Action execution (LD)

We started creating the ***Action execution POU in Ladder Code***, it is responsible of taking the data from the PLC on FLuidSIM and convert them into actions according to a sequence.

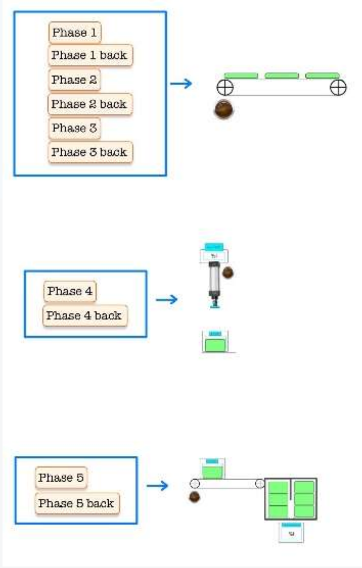
Ladder Diagram is a graphical programming language, it is used to develop software for programmable logic controllers (PLCs).
It is one of the standard languages specifies for use with PLCs.



action_execution

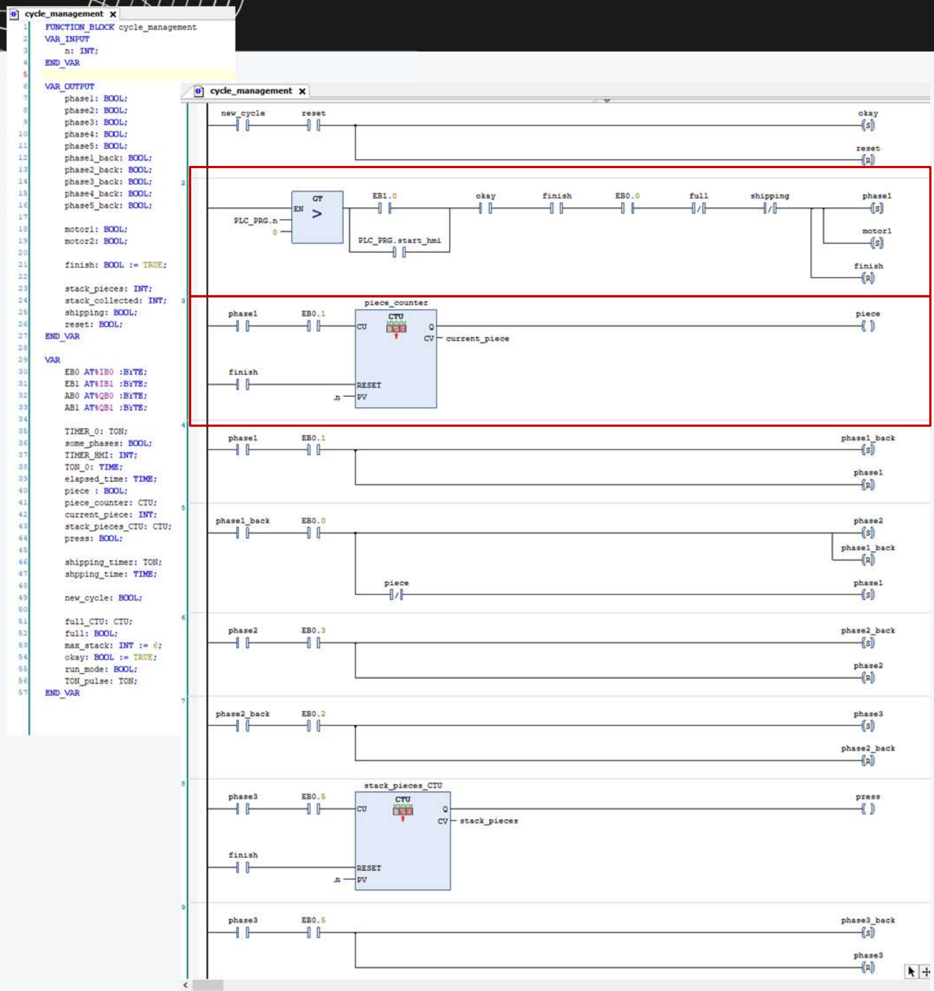
```
FUNCTION_BLOCK action_execution
VAR_INPUT
    phase1: BOOL;
    phase2: BOOL;
    phase3: BOOL;
    phase4: BOOL;
    phase5: BOOL;

    phase1_back: BOOL;
    phase2_back: BOOL;
    phase3_back: BOOL;
    phase4_back: BOOL;
    phase5_back: BOOL;
END_VAR

VAR_OUTPUT
END_VAR

VAR
    EB0 AT%IB0 :BYTE;
    EB1 AT%IB1 :BYTE;
    AB0 AT%QB0 :BYTE;
    AB1 AT%QB1 :BYTE;
    motor1: BOOL;
    motor2: BOOL;
END_VAR
```

In CODESYS we first defined all the input and output variables and then we divided all the operations of our cycles into different phases to handle them in an easier way:



- AB0.0 (conv1+) -> Phase 1: conveyor1 outstroke -> first box goes forward
- AB0.1 (conv1-) -> Phase 1 back: conveyor1 instroke -> first box goes backward
- AB0.2 (conv2+) -> Phase 2: conveyor2 outstroke -> second box goes forward
- AB0.3 (conv2-) -> Phase 2 back: conveyor2 instroke -> second box goes backward
- AB0.4 (conv3+) -> Phase 3: conveyor3 outstroke -> third box goes forward
- AB0.5 (conv3-) -> Phase 3 back: conveyor3 instroke -> third box goes backward
- AB1.0 (press+) -> Phase 4: press outstroke -> press goes forward
- AB1.1 (press-) -> Phase 4 back: press instroke -> press goes backward
- AB0.6 (conv4+) -> Phase 5: conveyor4 outstroke -> pressed box goes forward
- AB0.7 (conv4-) -> Phase 5 back: conveyor4 instroke -> pressed box goes backward

06 CODESYS – Cycle management (LD)



Then we created the ***Cycle management POU*** in ***Ladder Code***, it:

- is responsible for managing the performing of the loop cycles.
- defines the position and the timing of each phases
- performs the pulse signal for the loading of the plates on the first conveyor
- performs all the sequence phases
- manages all the counters and timers

We used all the information related to the limit switch mounted on the pistons on FLuidSIM to better supervise all the operations. Here we handled all the variable:

- new cycle
- full
- shipping
- finish
- okay
- reset

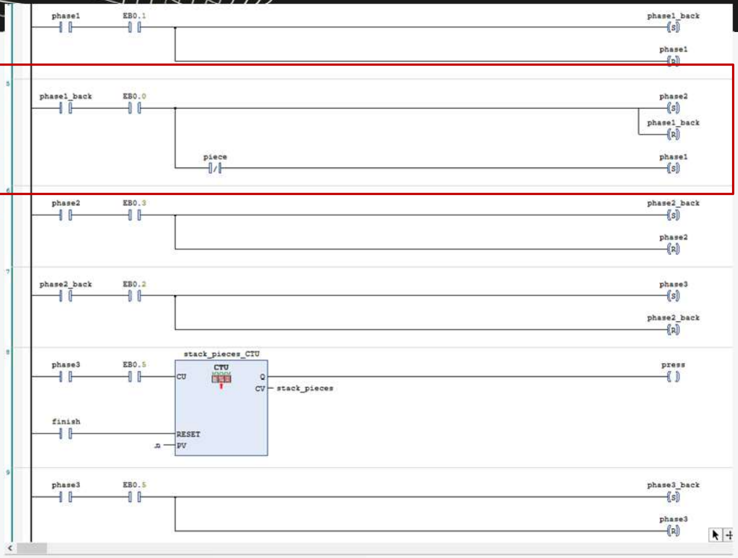
The **second network** starts effectively the loop.
If :
- n > 0
- start is pressed (either from the HMI or from the PLC)
- all the variables are checked

Then:
- phase1 is performed
- motor1 is turned on
- reset of the variable finish

The **third network** is used to count the pieces loaded on the first conveyor.
If:
- n = the quantity of pieces loaded on the conveyor

Then:
- stop the loading

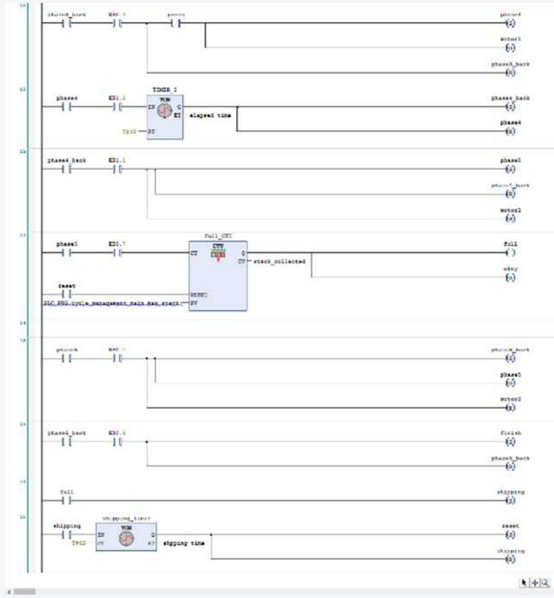
06 CODESYS – Cycle management (LD)

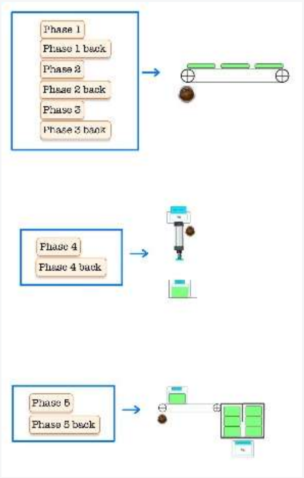


The most important condition is the **network 5:**

- the phase1-back set the execution of phase2
- in parallel it sets also the phase1 again if the variable piece is not set.

Finish variable -> set at every ending loop of stack-cycle and reset at every start.



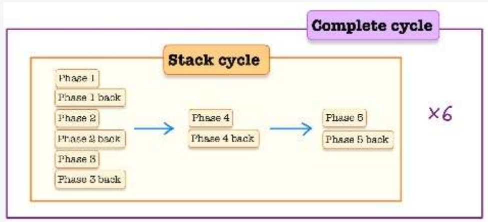


From phase1 to phase3-back = boxes on the first conveyor.

From phase4 to phase4-back = press -> 3 seconds -> timer

From phase5 to phase5-back = stack from the press station to the stacking area

Stacking area -> max 6 stacks -> counter
If counter = 6 -> full -> shipping -> delivering -> new-cycle



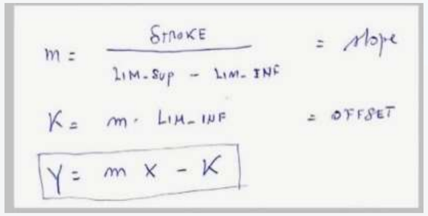
06 CODESYS – Cylinder displacement (LD)

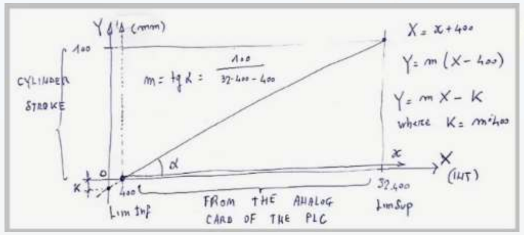
The ***cylinder displacement calculation*** is a POU written in ***Ladder Code***, necessary to convert the linear position of the pistons in a linear function and then in a scale from 0 to 100 to better handle the animations.

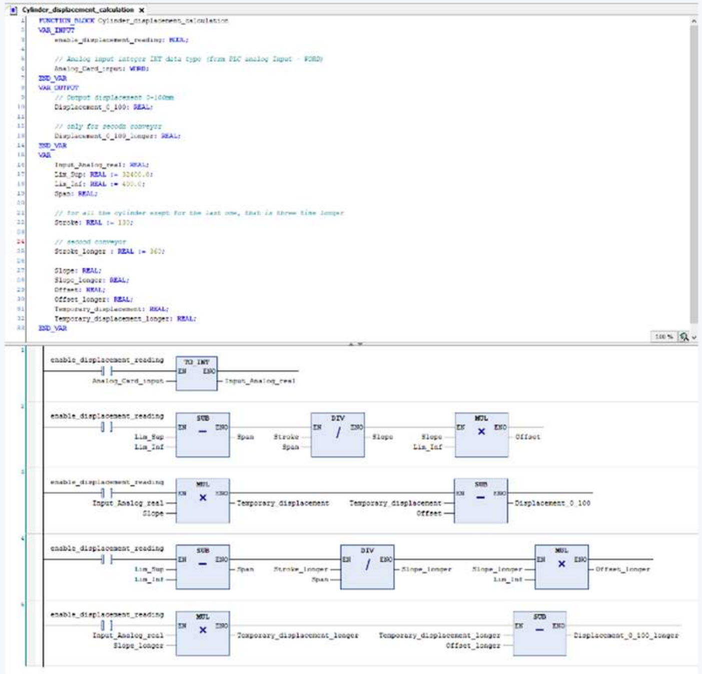
The displacement encoder reads the value of the position of the piston. They send these data from the analog card of the PLC to CODESYS.

By means of some **simple formulas**, we are able to have a linear scale of the position. To then perform animations, as translation, in a better way.

We firstly convert the data from the analog card of the PLC to an integer number, then we use **math operators blocks** to perform calculations, as divider or subtractor and so on, to have the displacement as output.







06 CODESYS – Emergency (ST)

Emergency

```
FUNCTION_BLOCK Emergency

VAR_INPUT
    start_hmi: BOOL;
END_VAR

VAR_OUTPUT
    stop_state: BOOL;
    emergency_state: BOOL;
    normal_mode : BOOL;
END_VAR

VAR
    EB0 AT%IB0 :BYTE;
    EB1 AT%IB1 :BYTE;
    AB0 AT%QB0 :BYTE;
    AB1 AT%QB1 :BYTE;

    emergency : BOOL;
    stop : BOOL;
END_VAR
```

```

IF EB1.0 OR start_hmi THEN
    emergency_state := 0;
    stop_state := 0;
END_IF


IF emergency OR NOT EB1.7 THEN
    emergency_state := 1;
END_IF

IF emergency_state THEN
    AB1.2 := 0;
    AB1.3 := 0;
END_IF


IF stop OR EB1.6 THEN
    stop_state := 1;
END_IF

IF stop_state THEN
    AB1.2 := 0;
    AB1.3 := 1;
END_IF
```

The *Emergency* POU, written in ***Structure Text (ST)***, is necessary to prevent dangerous situation and handle failures and unexpected results.

We defined three states:

- normal-mode
- stop-state
- emergency-state.

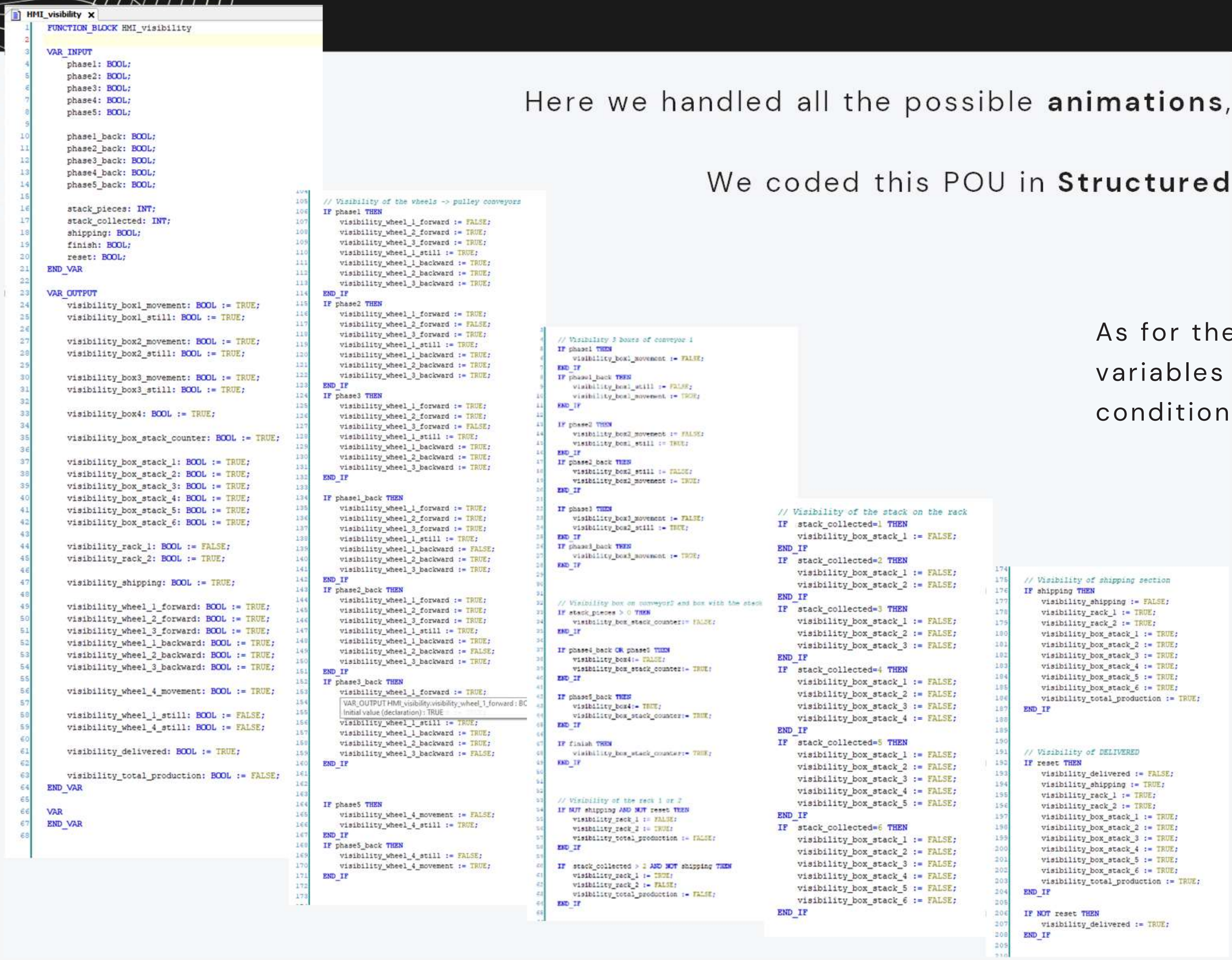
At the beginning of the cycle	During the cycles
If: • start button is clicked (either from the HMI or from the PLC) Then: • *stop-state* and *emergency-state* -> set to *FALSE* or *0*.	If: • emergency button is clicked (either from the HMI or from the PLC) Then: • *emergency-state* -> set to *TRUE* or *1*. If: • stop button is clicked (either from the HMI or from the PLC) Then: • *stop-state* -> set to *TRUE* or *1*.

Here we handled these two states directly on the pneumatic circuit, placing two 3/2 valve with 1 stable position:

- *Emergency-state* -> both of the valve are de-energized -> stop the system instantaneously.
- *Stop-state* -> blocks only the source of air -> let the exhaust open to unload the air -> end in the next phase.

We did it sending commands directly to the PLC port where the solenoids of the valves are connected.

06 CODESYS – HMI visibility (ST)



Here we handled all the possible **animations**, in terms of *translation and visibilities*.

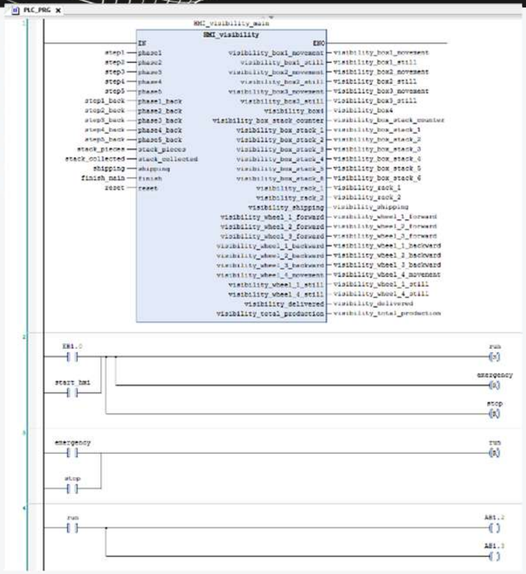
We coded this POU in **Structured Text**.

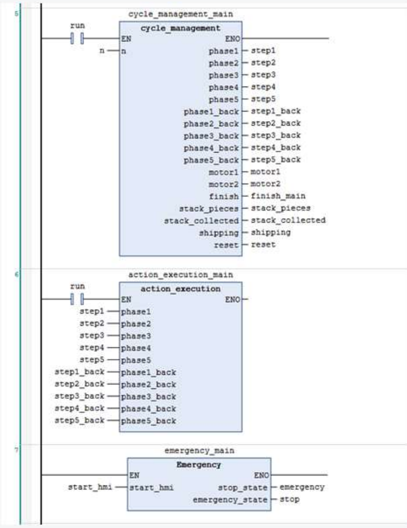
As for the other POUs we firstly set the input variables that were needed to activate a particular condition and then, we coded them in if-cycle.

We dealt with *visibility* of:

- three plates on the first conveyor
- the big box that contains the plates coming from the first conveyor
- the storage area
- the shipping
- the delivering station.

06 CODESYS - MAIN PLC_PRG



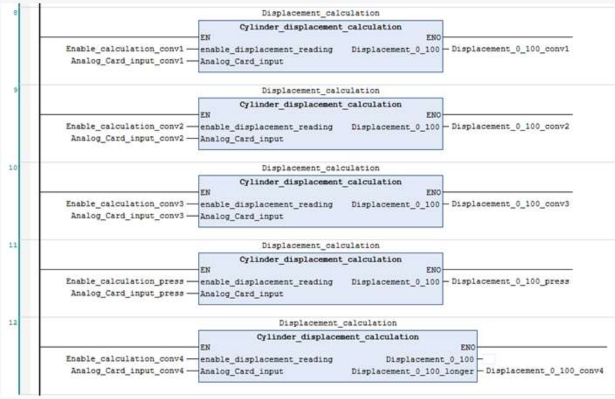


The ***MAIN POU*** is the most important POU of the whole project, because it takes all the information from the other POUs, and it organizes them to let them work together.

Firstly, we have that the **start button (**either from the HMI or from the PLC):

- set the variable run that -> open the valves source and exhaust of the pneumatic circuit.
- reset stop and emergency

Stop and emergency reset run.



The main POU is responsible for linking the input and output of POU-blocks:

- HMI_visibility
- cycle_management
- action_execution
- Emergency
- cylinder_displacement_calculation

On the left we have the **inputs** that enter to the block of the POU and on the right the **output** that exit from it.

07 HMI

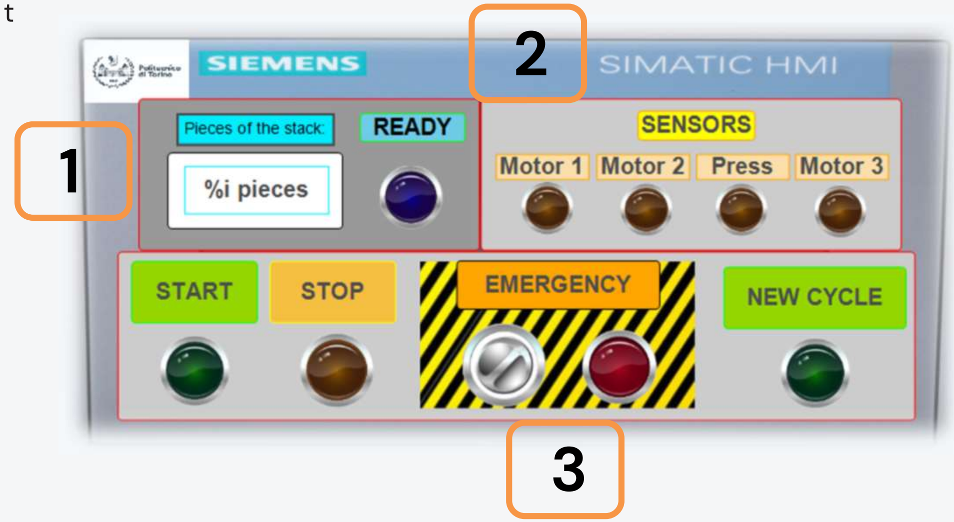
As far as the ***Human-Machine Interface (HMI)*** is concerned, we built a simple panel composed of three sections from where we can control the whole system.

The three sections are:

1. **Input**
2. **Sensors**
3. **Modes**

We used:

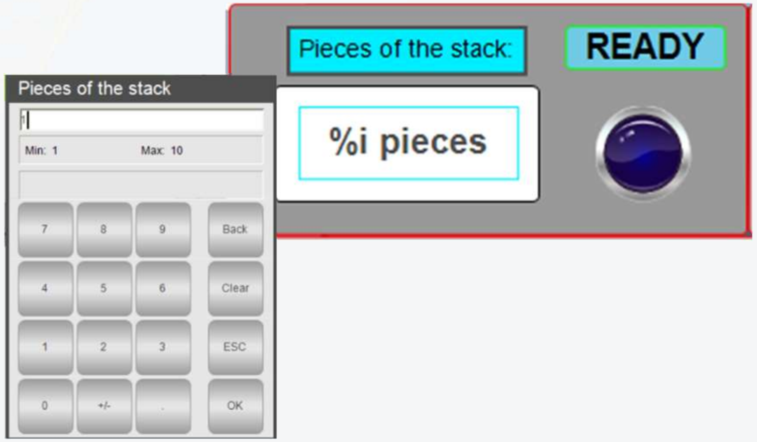
- **Push-buttons**
- **Selectors**
- **Lamps**
- **Input numpad**



Human-Machine Interface (HMI) is very important 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 – Input & Sensors



The first section is related to the **input**:

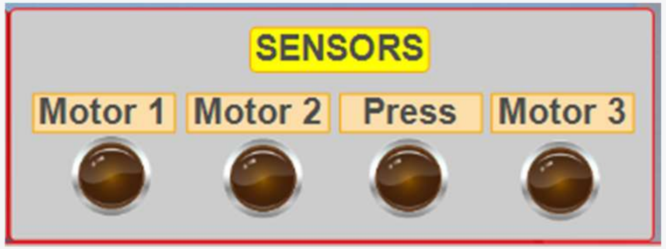
- at the beginning of each stack-cycle *the operator is required to insert the number of pieces* that the stack will be formed by.

A stack-cycle is made of the number of pieces selected by the operator that are going to be pressed and stocked.

Six stack-cycles compose a complete-cycle.

Clicking on the box *pieces* a **numeric pad** appears and the operator can insert a number (minimum 1 and maximum 10 to make the simulation more dynamic)

At the end of every stack-cycle the **blue lamp** will turn on indicating that the machine is ready to perform a *new cycle*.



The **sensors** are simulated by the **yellow lamps** and indicate the status of the motors, they turned on to indicates that motors are moving.

They are also present in correspondence of the motor in the plant.

07 HMI – Modes

According to the requirements the push-buttons are responsible to:

- **start** the cycle
- **stop** the cycle in the first available ending phase
- **stop** the cycle immediately in case of emergency and start a new cycle

START:

- motor 1 is activated
- conveyor 1 starts to move
- the phases related to the boxes are activated

we will see as much boxes as we set on the input appear on the first conveyor once at time and max 3 at the same time.

STOP:

- air in the pneumatics circuit is stopped

the cycle will end in the next available phase.

EMERGENCY:

- the whole system is stopped
- all the conveyors are stopped
- all the boxes are stopped

The emergency is essential, and it has the highest priority. There is a red lamp that allows us to check the emergency status.
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.

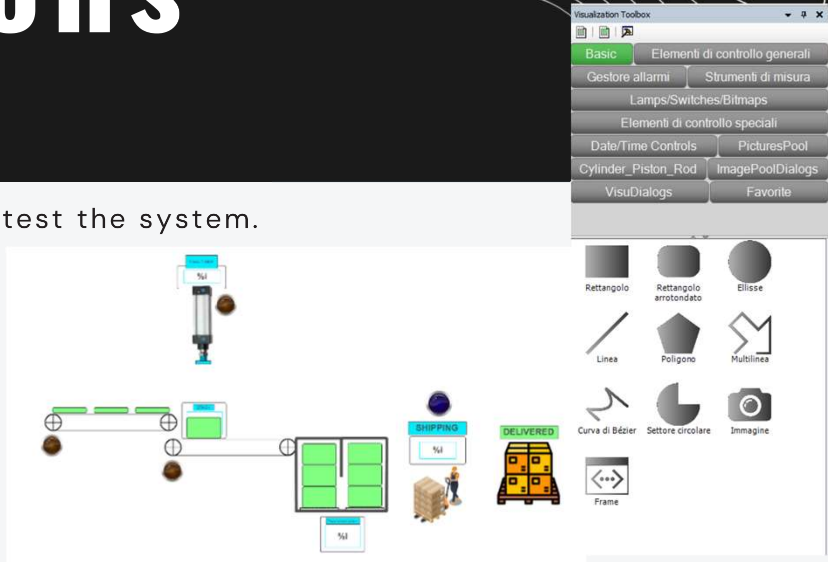
NEW CYCLE:

- reset all the variables
- make available a new cycle

is used when we have 6 stacks ready, shipped and delivered.

08 Animations

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

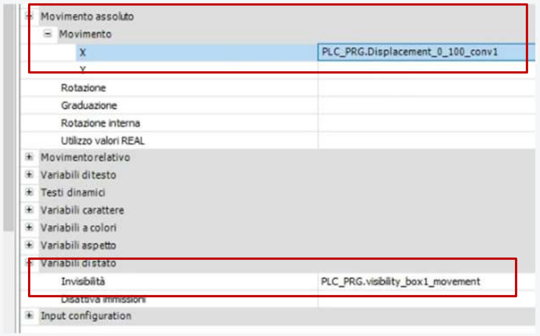


We created everything form scratch, with lines, round and square shapes, using the ***Visualization Toolbox*** *in CODESY*, in particular, we drew:

- 10 boxes
- 2 conveyors
- 1 piston
- 1 shipping station
- 1 delivered station
- 2 timers
- 1 counter

We also added pictures to make the simulation more real:
- for the emergency area
- for the HMI panel
- for the shipping and delivered station.

We embedded them into the projects thanks to the section *Pictures*



To be able to control the animations with the loops programmed with *Ladder code* and *Structured Text,* we had to **connect the objects with pre-defined variables**.

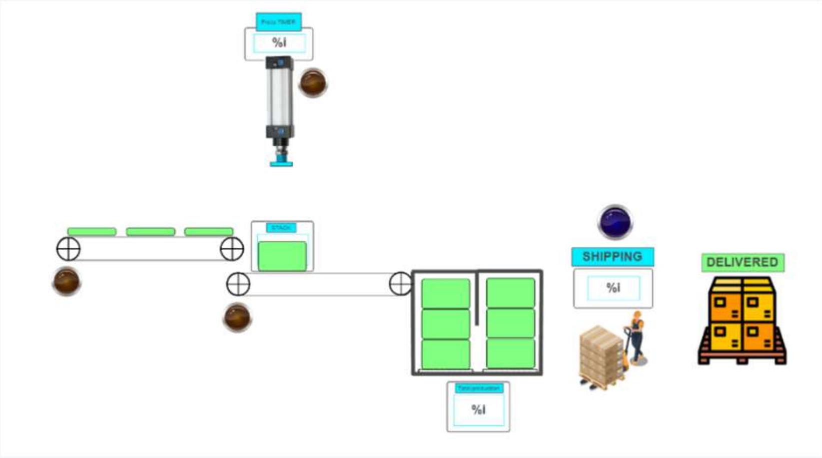
We worked with *two types of animations*:
- **Visibility**
- **Translation**

08 Animations - Pictures

To better visualize our plant, we built the whole system from scratch using the visualization toolbox of CODESYS.

We used rectangles, ellipsoids, lines, counters, lamps, buttons and so on.

To make a complete simulation we added animations, as for the boxes but also for the stoking, the shipping and the delivering stations.
We had timers and counters, and we mixed everything programming visibilities and linking variables.



For example, as far the lamps are concerned, they are simply linked to a variables of the MAIN POU.
While the plates are linked both to the visibility variable to be visible only when it is necessary and to the displacement variable to be able to move and simulate the conveyor.

FINAL RESULT

Single stack-cycle with 4 pieces

FINAL RESULT

Complete cycle

FINAL RESULT

Stop & Emergency

OUR TEAM

Luigi
Muratore
s333098

Muhammad Fatir
Noshab
s331898

Iskandar
Akbarov
s329650

THANKS FOR WATCHING

Politecnico di Torino

1859