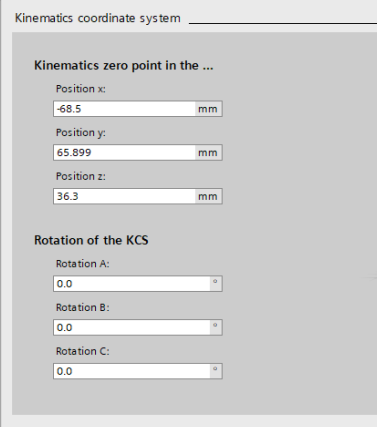
**POCFA documentation – internship Pieter**

Internship August-September-October 2021, Siemens NV/AG Huizingen - Pieter Van den Bossche

1. **Translation coordinate systems due to slip - IMPORTANT**

When implementing the new disassembly cycle, the machine went out of its kinematic zones, bumped into its hardware limits and the servos which manage the x/y direction slipped.

* This means the encoders are not aware that the position value they give to the TIA program is in fact wrong. Due to this fact, the KCS had to be translated in TIA by some centimeters, as shown in the picture below:



* Because the OCS’s are normally referenced with respect to the KCS, nothing else has to be changed.

|  |
| --- |
| **IMPORTANT NOTE #1:**  If one opens/downloads an older TIA project version into the physical POCFA machine, **CHECK IF THIS EXACT TRANSLATION (x –68.5 mm, y + 65.899 mm, z +36.3mm) IS SET IN THE PROJECT. IF THIS IS NOT DONE, SET THE VALUES SHOWN IN THE PICTURE AND BE SURE TO TEST IT FIRST VERY CAUTIOUSLY AT REDUCED SPEED** (e.g. take master control and set to 10% speed). **OTHERWISE MECHANICAL SLIP MAY AGAIN OCCUR!** |

|  |
| --- |
| **IMPORTANT NOTE #2:**  **Another translation has to be done when testing on the digital twin, see section 4.2 of this document (MCD – Offset in z-axis)**. Make sure to use the proper TIA versions for both! I made a separate folder structure and some README files in the CC4I pc to underline this. I also underlined this change in the **change log of the TIA backups, which can be found in the Teams group created for my internship.** |

1. **Assembly cycle - repairing and finetuning**

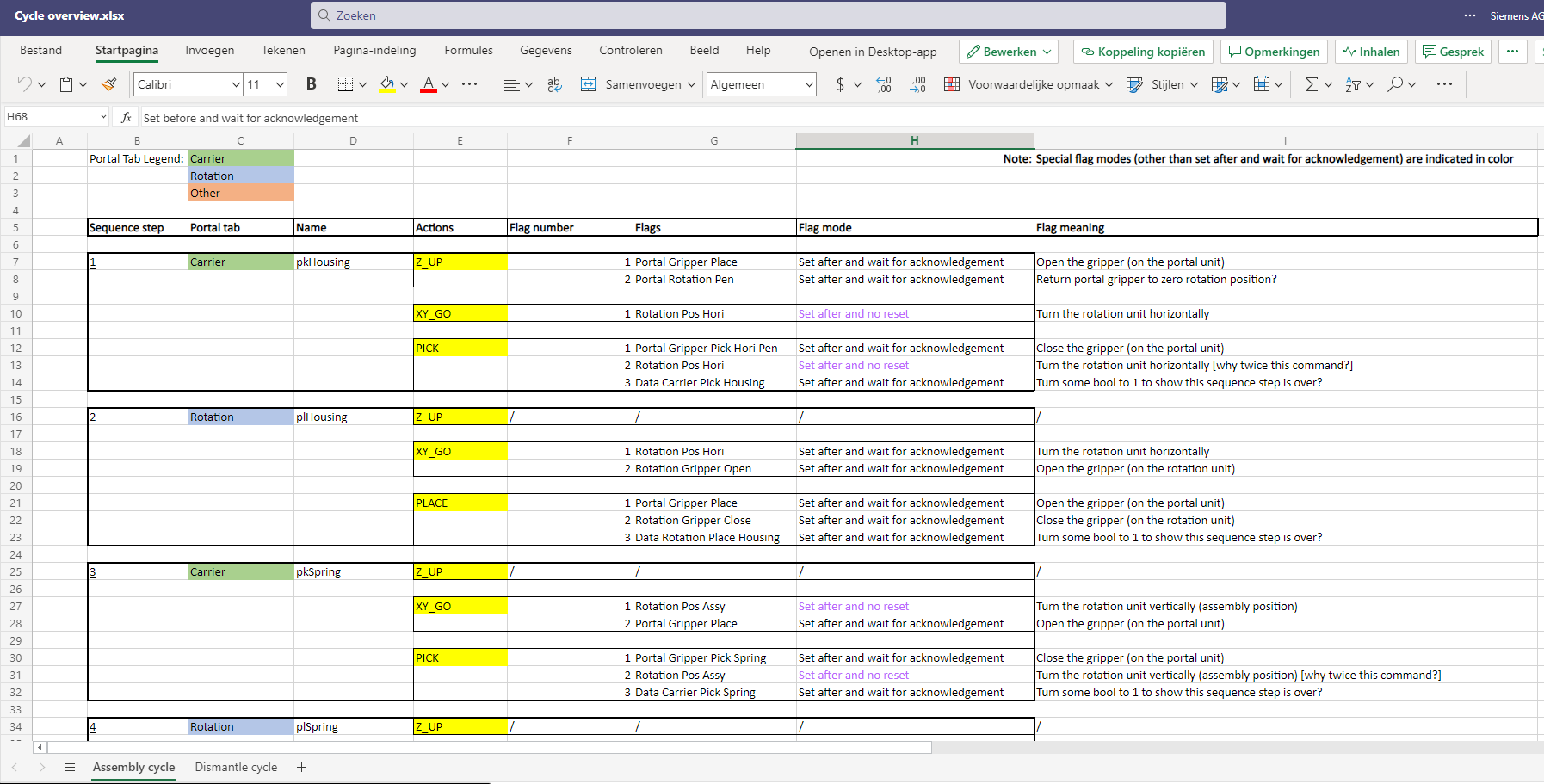
By fixing the mechanical slip problem mentioned before, we accidentally lost part of the assembly cycle position values, no idea how. Some of the actions were missing and position values were off by some mm’s. The missing steps were added again (loaded older snapshot) the positions were finetuned.

|  |
| --- |
| **NOTE:**  **TO AVOID THIS, MAKE SURE TO MAKE SNAPSHOTS OF THE POSITIONS AND SEQUENCE OFTEN ENOUGH, AND ARCHIVE THESE FREQUENTLY!** |

Both the EM01-HL00 - PickPlace / DB / Portal and EM01-HL00 - PickPlace / DB / SequenceData data blocks were snapshotted and **the program was archived, both on the PC in the CC4I, as well as on the PG. These were also uploaded in the back up folder on the Teams group.**

1. **Dismantle cycle - buildup**
   1. **Flags**

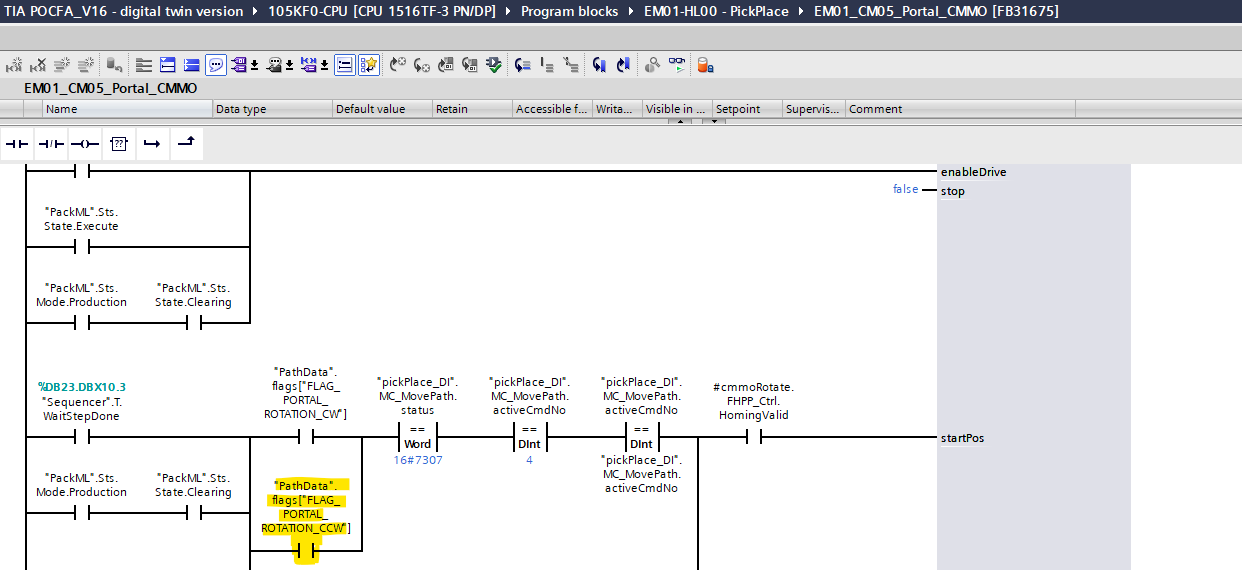
* The first problem encountered when building the dismantle cycle was the naming of flags: I didn’t know what exactly was done by the machine by setting the flags, the naming was unclear to me.
  + I tested a bit and looked in the program which actuator was linked to which flag. This is a tedious way of finding out a very simple thing
* **I made an overview of the Assembly cycle in Excel**. It includes a list of all steps of the Assembly sequence, the actions that are done in each step and the flags used in each of the actions. From this overview, I tried to identify what the flags do. This file can be found on the Teams group as well, it looks like this:



* 1. **Counterclockwise rotation**

To unscrew the cap, a counterclockwise rotation has to be performed, as opposed to the clockwise rotation performed when fastening the cap. When setting this flag, this rotation did not happen.

* The most obvious reason could be that the FESTO tool didnt have enough pressure, because it has to feel a certain pressure to perform this rotation: it has to be lowered sufficiently to activate the rotation. This was done, but still no rotation happened.
* The actual cause was that the **rotation flag was not wired to the actuator in the TIA program**. A normally open contact was added in NW43 of Program blocks / EM01-HL00-PickPlace / EM01\_CM05\_Portal\_CMMO to fix this, this is shown and indicated in yellow on the image below:



* 1. **Physical/mechanical problems**

A few physical/mechanical problems were encountered when building up this cycle, which made the cycle not robust.

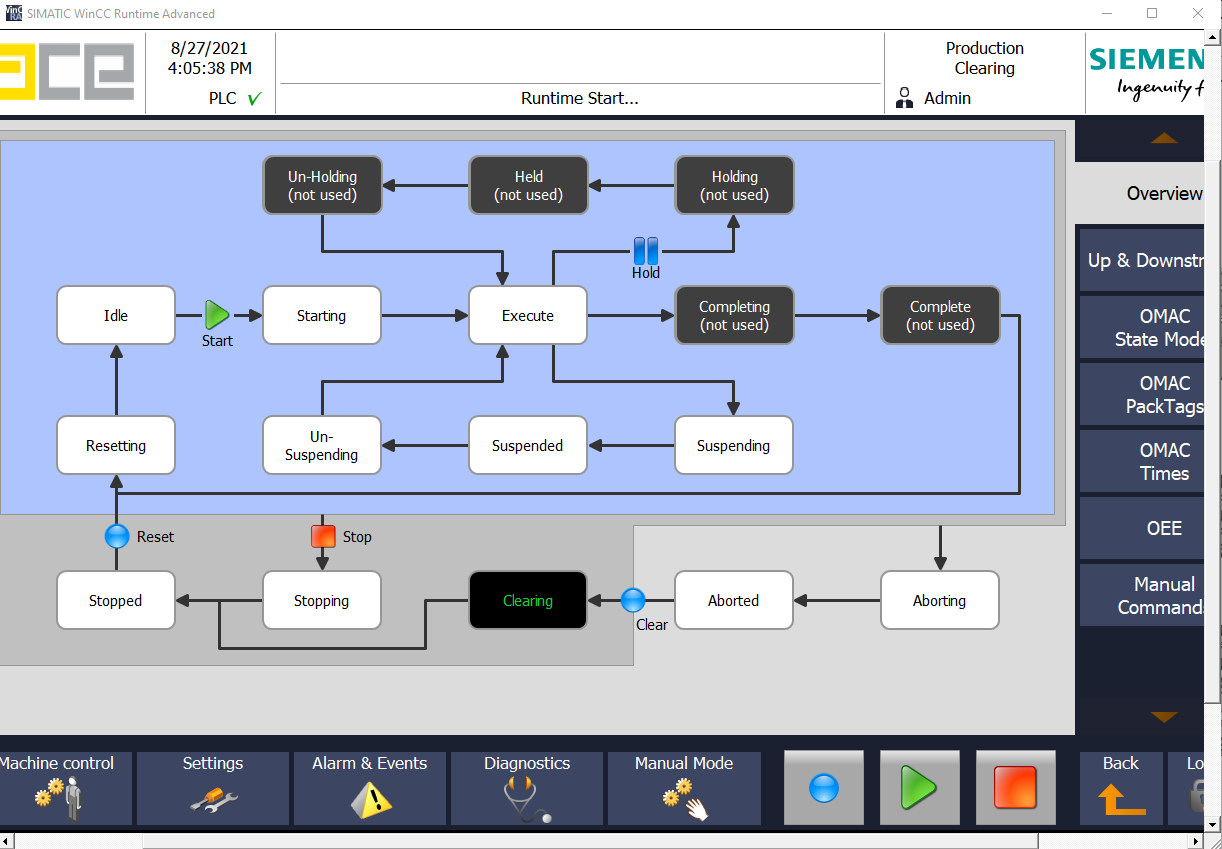
* The spring gets clamped by the inside of the cap when the pen gets assembled. This causes the spring to stay attached to the cap when unscrewing and lifting the cap in the disassembly cycle, which is a problem.
  + A possible mitigation is to drill a hole in the inside of the cap with the diameter of the spring + 1-2 mm, such that the spring doesn’t get clamped by the narrow inside.  
    **==> This was done and it solved the problem**
* The cap does not always get unscrewed, because the gripper does not always have enough grip on the cap. Mitigations:
  + Gripping the cap AFTER lowering the portal gripper to the proper height instead of before
  + Also, heightening the inscrewing position during assembly such that the cap does not get stuck too hard

**==> This was done, works**

* When picking up the spring from the disassembled pen, the tool cannot go low enough without bumping into kinematic zones
  + We tried moving the sensor, but apparently this was the hardware limit of the machine. ==> **The solution is to put the pen into the gripper not in the middle, but at the side. This way, the pen is positioned a bit higher when the gripper is in assembly position, such that the portal can reach all positions.**

1. **Digital twin - MCD**
   1. **Staying in ‘clearing’ state - IMPORTANT**

When trying to run the TIA project on the digital twin, a lot of complications arose. When using the MCD, the machine stays in the clearing state, while staying stationary (normally the machine moves up and down the z axis once when clearing).



I suspect that what is going on inside is the following:

* The MC\_GroupPower block enables its axes, but goes into error state (probably because the MovePath block wants to begin a movement before the axes are enabled or after the axes are disabled) and gets resetted immediately by an MC\_GroupReset block, this happens every cycle.
* This means the MovePath block cannot be executed, as the axes are not enabled. It goes also into error state.

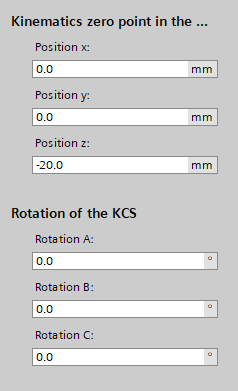
**==> This happens inside of the block Supportingblocks / kinCtrl, a block made by ACE**

* The **solution** to this problem is not very rigorous nor clean:When stuck in the clearing state, put the PLC in STOP, then in RUN AGAIN. If this doesn’t work, restart SIMIT / NX / TIA, or even your pc itself or try the previous steps again until it works.

|  |
| --- |
| **NOTE:**  **This issue does not appear when running the program on the real POCFA. However, when rebooting the real POCFA, it also acts weird (the rotation unit turns a few times, in a very jerky manner) and three resets have to be carried out until the machine goes to its initial position. One can also go into manual mode and home the Z rotation gripper. When going back to production mode, the clearing will work again. (This was documented in some small movies I made)** |

* 1. **Offset in z axis - IMPORTANT**

When running the program on MCD, an offset of 20mm in the z direction of the KCS has to be given:

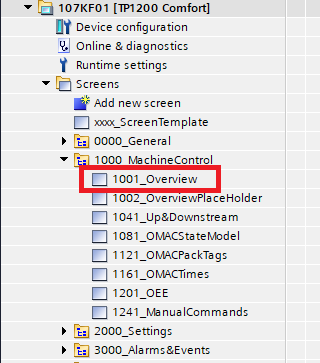


This is very strange, even more so because in earlier versions of the POCFA code, the positions were 20mm higher (in z), but when extracting the current code from the machine, the positions were 20mm lower. This resulted in having to translate the KCS in z to get the correct positions on the digital twin.

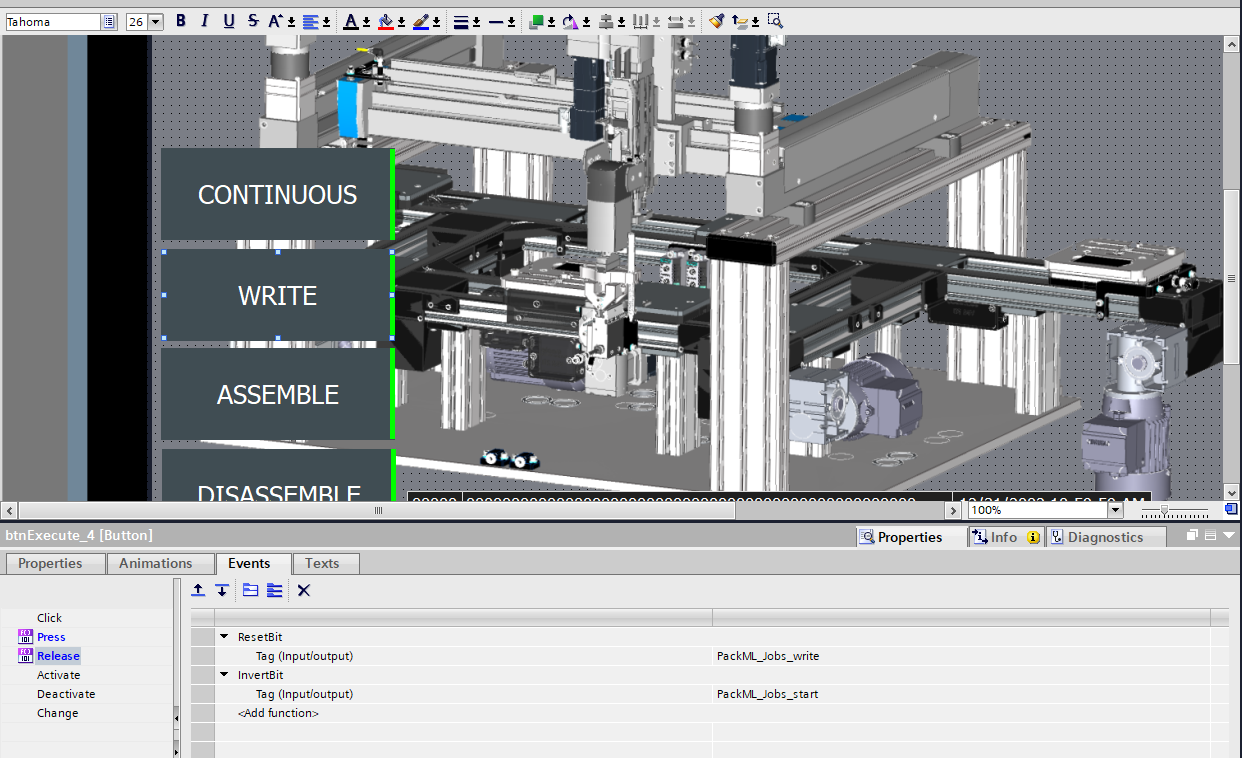
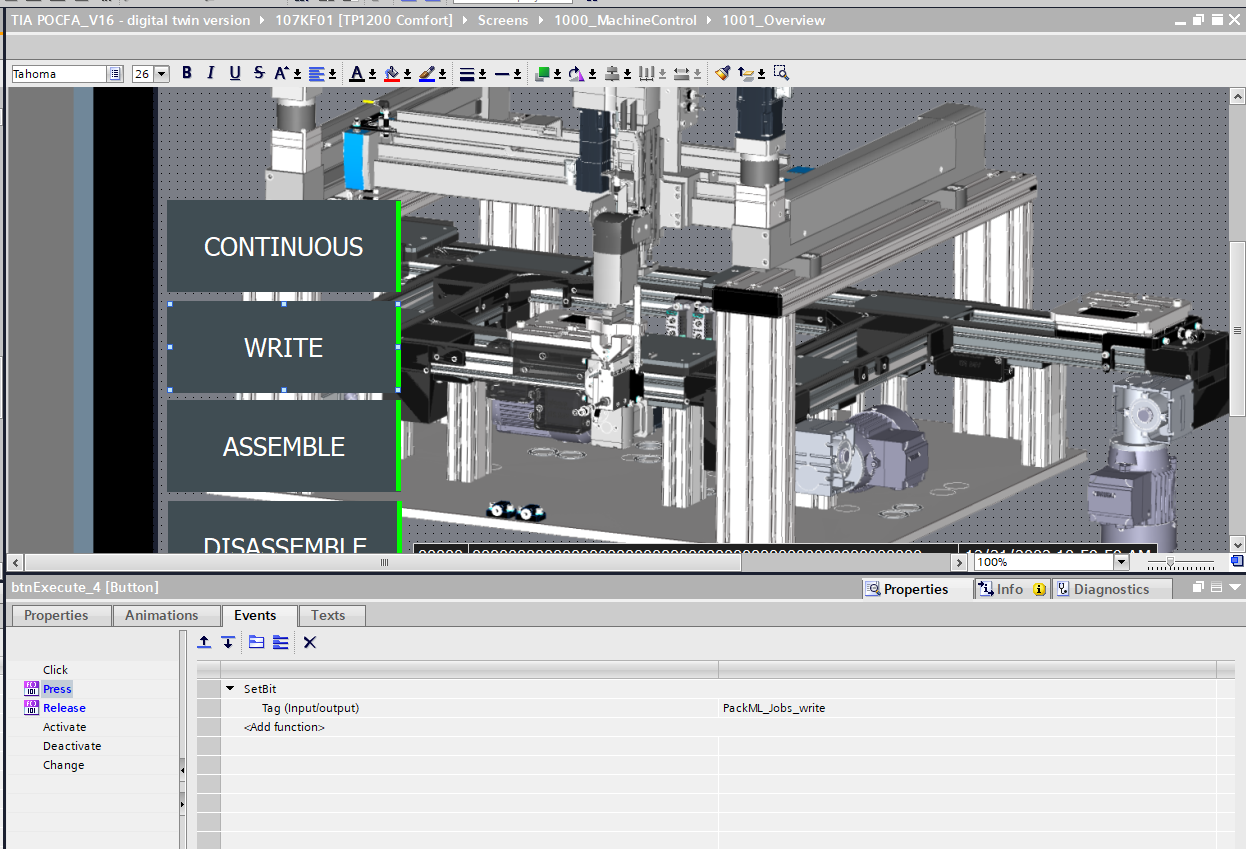
1. **Writing with the Portal**  
   1. **Wiring of new sequence**

The first step in implementing the writing function is being able to define and execute a separate cycle for the writing. This cycle is already defined in the HMI, and contains a few test steps, because the real writing cycle has yet to be made.

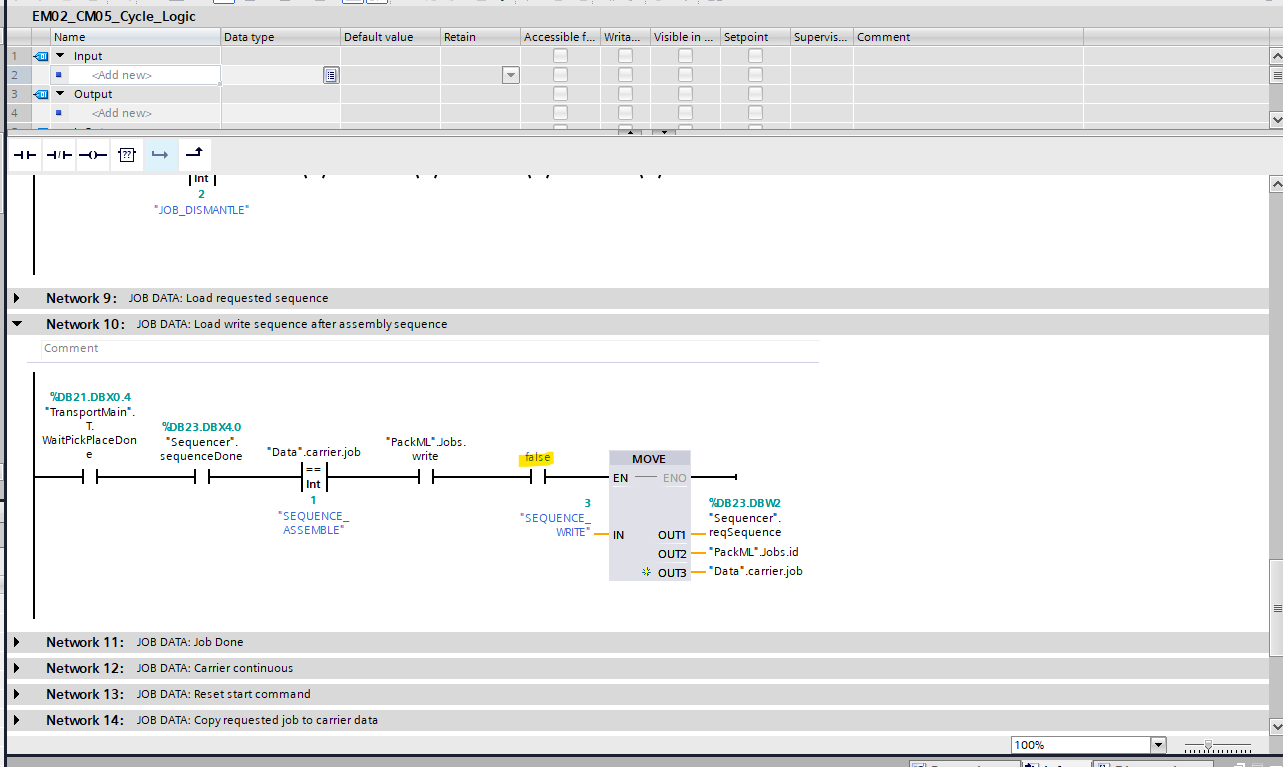
* First, a ‘Write’ button was added to the HMI begin screen. To go to the begin screen, open the HMI ‘Screens’ folder, open the first subfolder and double click on the first screen, this is indicated in the image below:



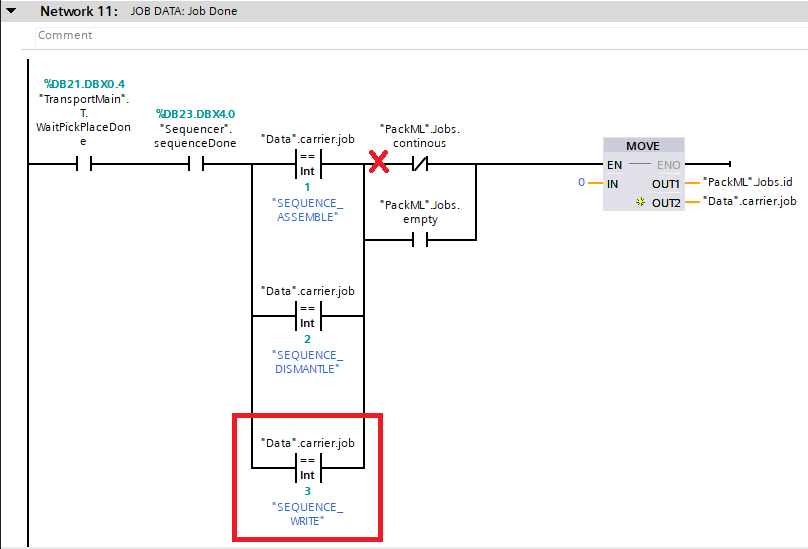
The button was then wired to the PackML\_jobs\_write tag in the HMI, as illustrated in the images below:



* After this, the network 10 in the function block EM02\_CM05\_Cycle\_Logic was disabled, because it automatically loaded the ‘Write’ sequence after the ‘Assembly’ sequence, which did not work... This was done by placing a permanent ‘false’ on a normally open contact, indicated in yellow on image below:

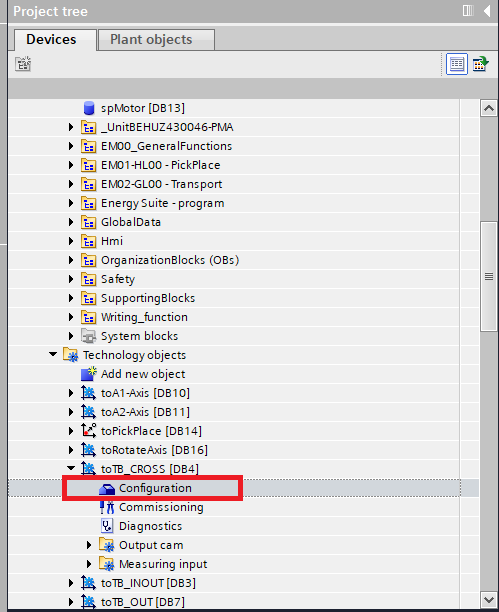


* The last thing to do was making sure that after the ‘Write’ sequence ended, the job id was set to 0, such that the carrier gets transported to the end of the line by the conveyor system. To do so, a condition in NW 11 of the function block EM02\_CM05\_Cycle\_Logic was added and one was deleted. The changes are illustrated in the image below, where a normally open contact was added (circled in red) and a normally closed contact referring to the boolean “PackML”.Jobs.write was deleted (indicated by the red cross).

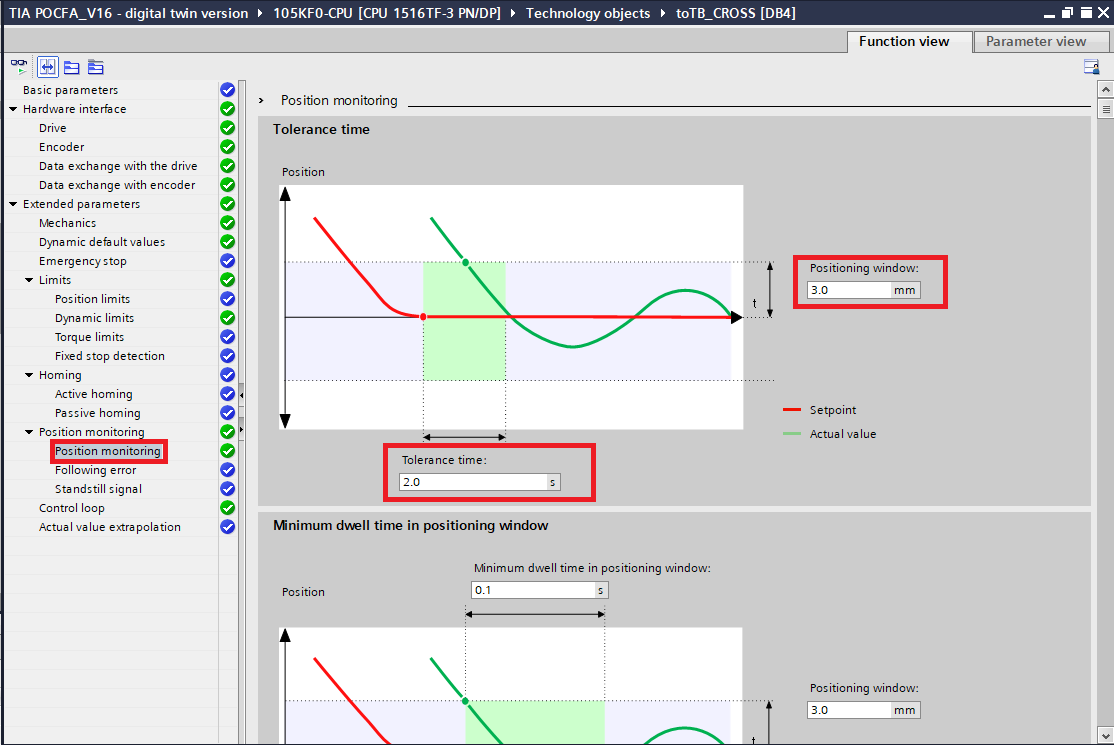


* 1. **Position monitoring bug (MCD)**

When working with the digital twin, the cycle always got interrupted by a position monitoring error thrown by the conveyor belts. Therefore, the constraints on the axes in the ‘Configuration’ tab of the conveyor axes had to be relaxed (see image below).



The ‘Positioning Window’ and ‘Tolerance Time’ values in the ‘Position Monitoring’ tab were adapted, this is indicated on the image below.



The original values were 1s and 1mm, which were okay for the real POCFA but not for the digital twin. This modification was done to all conveyor axes, then the cycle ran smoothly on MCD.

|  |
| --- |
| **NOTE:**  **Keep the position monitoring settings the way they were on the real POCFA (positioning window = 1.0mm; tolerance time = 1.0s)** |

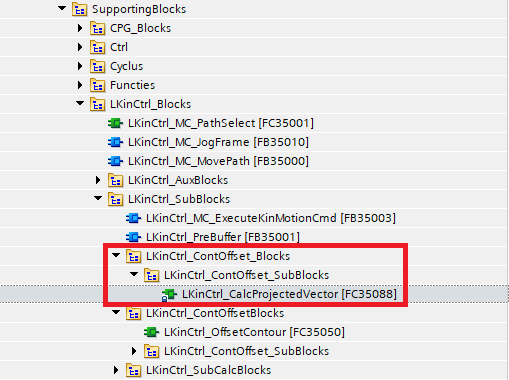
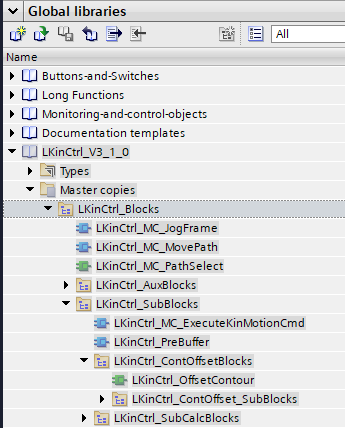
* 1. **Importing and updating libraries**

The version of the library LKinCtrl inside of the POCFA TIA program was V2.1, but the LKinLang library is not compatible with this version. This meant that the LKinCtrl library had to be updated to the newest version, V3.1.

* + 1. **Data types**

Due to the change in library, some types used by the LKinCtrl library blocks were updated and/or duplicated. This had to be cleaned up: the input type of some blocks was changed and had to be adapted manually, or some extra input types were needed. I will spare you the details, it was a tedious job.

* Also, one write protected block kept throwing incompatibility errors when replacing the old library with the new one: it did not need to be there, it was just not automatically deleted when replacing the library:

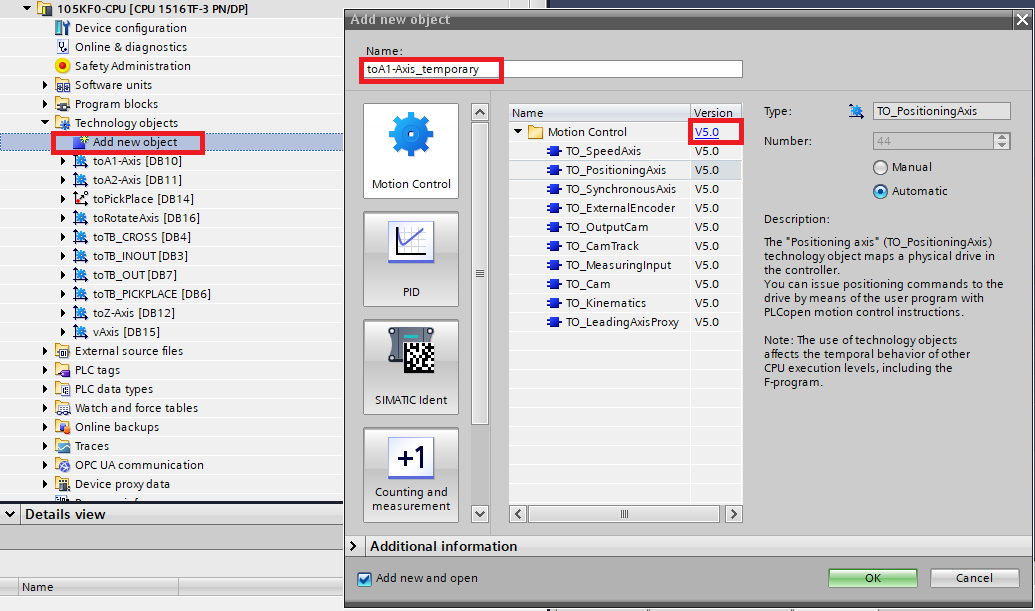
 

In the pictures above, you can see that in the new version of the LKinCtrl library, the name of the folder ‘LKinCtrl\_ContOffset\_Blocks’ is now ‘LKinCtrl\_ContOffsetBlocks’. This is probably the reason that the LKinCtrl\_CalcProjectedVector did not get deleted, because this folder is not present in the new library (see right image). Delete the folder manually to solve the errors thrown by this block. Make sure to always crosscheck the blocks you imported with the ones inside the ‘Master copies’ tab of the library.

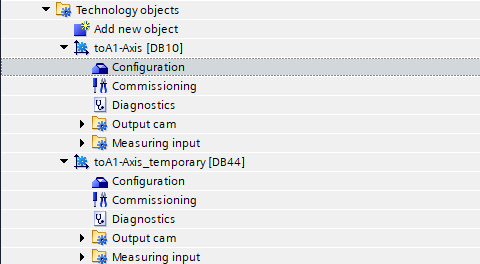
* + 1. **Technology objects incompatibility**

Because some data types were updated, they were no longer compatible with the version of the technology objects in the project (V4.0), so they had to be updated too, to V5.0. This was a tedious job, as there is apparently no way of doing this automatically. The way to do it is:

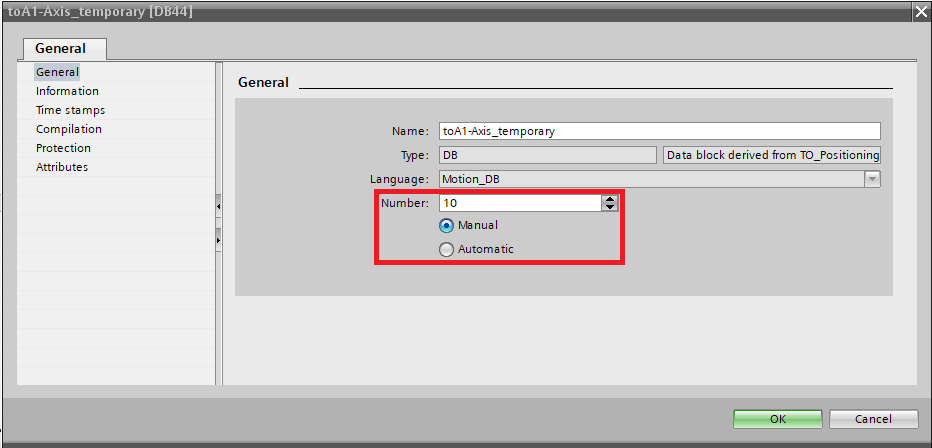
1. Add a new technology object of V5.0 and name it the same as the TO you’re trying to replace + ‘\_temporary’:



1. Open the ‘Configuration’ tab of both the TO’s and copy all the settings manually and very precisely to the new TO.



1. When this is done, delete the old TO
2. Set the DB number manually to the same number as the old TO (in this case: replace 44 with 10) by opening the ‘Properties’ tab of the TO (right click on the TO)



1. Rename the new TO to the name of the old TO (delete the ‘\_temporary’ in the name)
   * 1. **PLC tags adaptations**

Because a new version of the LKinCtrl library was imported, some of its user constants were adapted and/or duplicated. This had to be cleaned up: all duplicated user constants were deleted, the values of all tags were double checked with a previous version of the POCFA project and new tags were wired to the right blocks. Finally, everything was put into two tag tables:



* + 1. **HMI tags – PLC types accessibility adaptation**

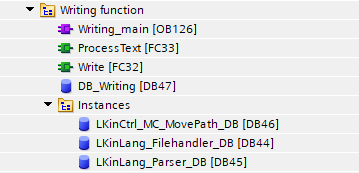
Because a new version of the LKinCtrl library was imported, the HMI accessibility settings of its types were changed. Usually, one does not need to be able to edit for example the flags of a LKinCtrl\_PathDataElement\_reduced type inside of the HMI, so by default the visibility/writability/... is turned off. However, in this HMI this is the case: one can add/change positions and flags in the ‘Settings/Portal’ tab of the HMI. This meant that the HMI visibility/writability/... settings had to be turned on for a few types, otherwise errors appeared inside of the HMI tag tables when compiling the HMI.

* 1. **First implementation – digital twin**

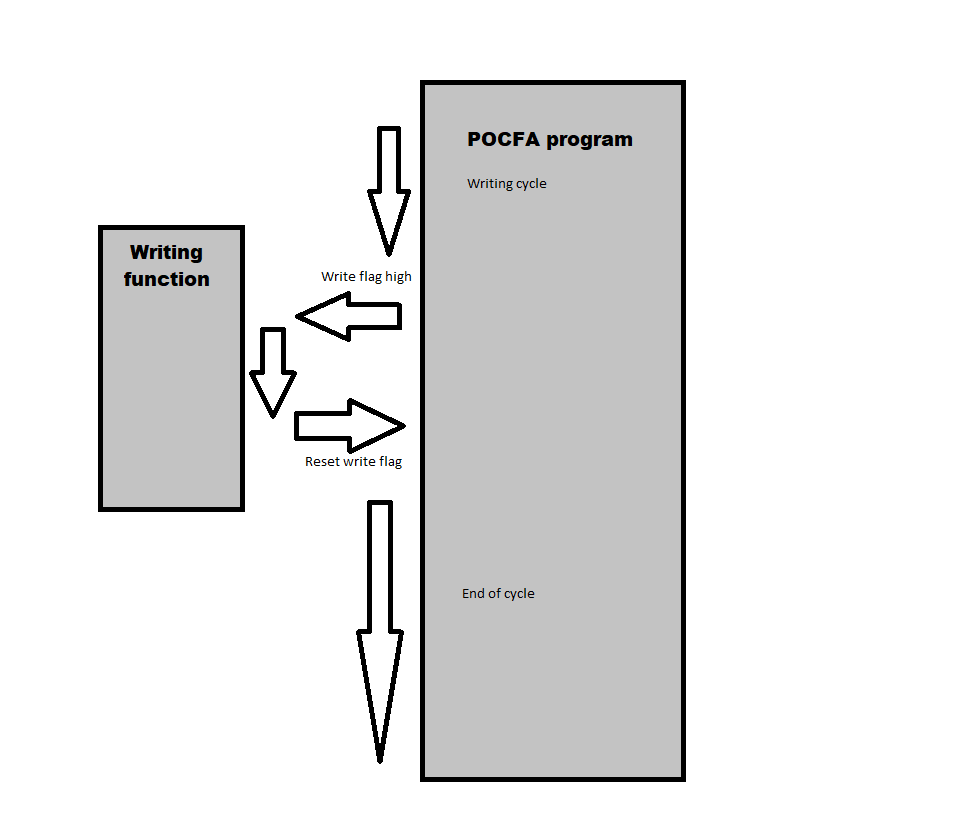
At first, the writing function was implemented and tested on the digital twin.

* + 1. **Main idea**

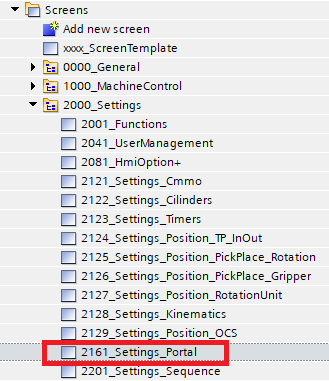
To keep things simple, the writing code was kept separately from the bulk of the POCFA program, all necessary blocks were put inside of a folder:



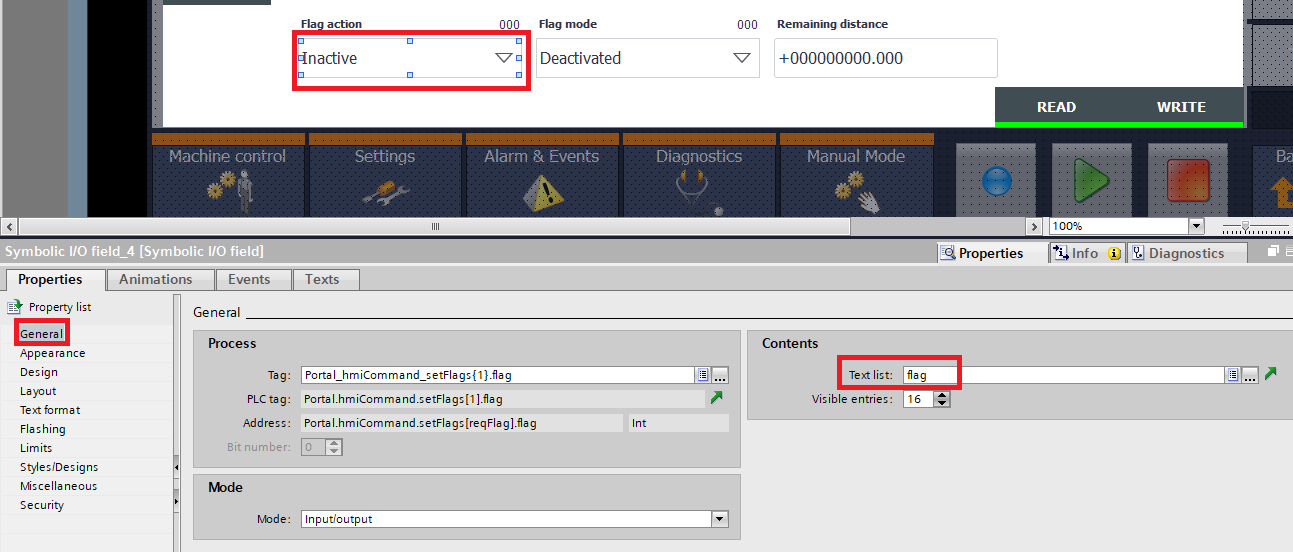
The idea is to use a ‘Write’ flag to initiate the writing function at the right moment in the writing cycle, after which the flag has to be reset in order to continue the cycle. Otherwise, the machine will wait in its current position for the reset (the flag mode is set to ‘Set After And Wait For Acknowledgement’). This means that the writing function can operate independent from the POCFA program, it just needs the text input from the HMI and the flag set to high at the right moment in the cycle. The program logic is shown in the diagram below:



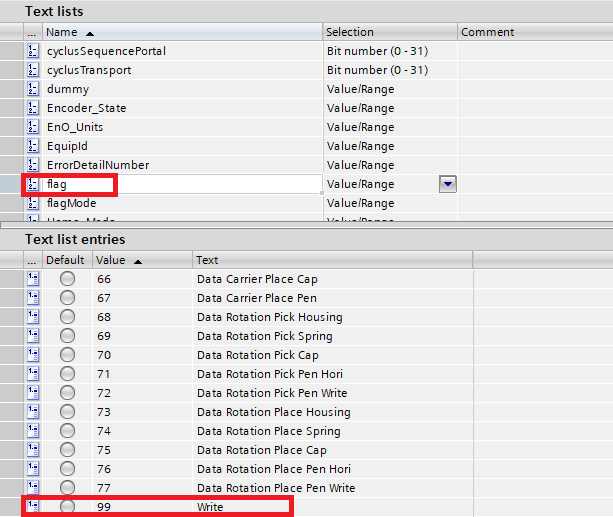
* + 1. **Adding a flag**
* First, the tag has to be visible in the list of the dropdown menu of the HMI. In the ‘Screens’ folder, open the ‘Portal’ screen:



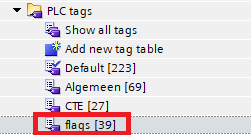
* Then, click on the dropdown menu of the flags and open its properties. In the ‘General’ tab, one can see that this HMI instance uses the text list ‘flag’:



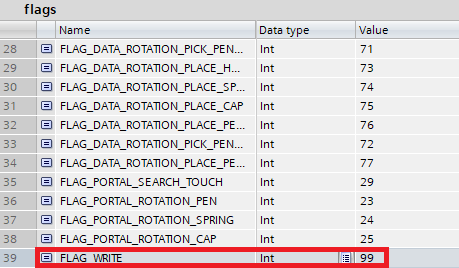
* Then, open this text list and add the new flag, using a unique but valid identification value:



* The next step is to define a user constant in the PLC, such that the flag can be easily identified. Open the ‘flags’ PLC tag table:



* Then, open the ‘User constants’ tab and define the constant, using the same id value as in the correspondent HMI text list entry:

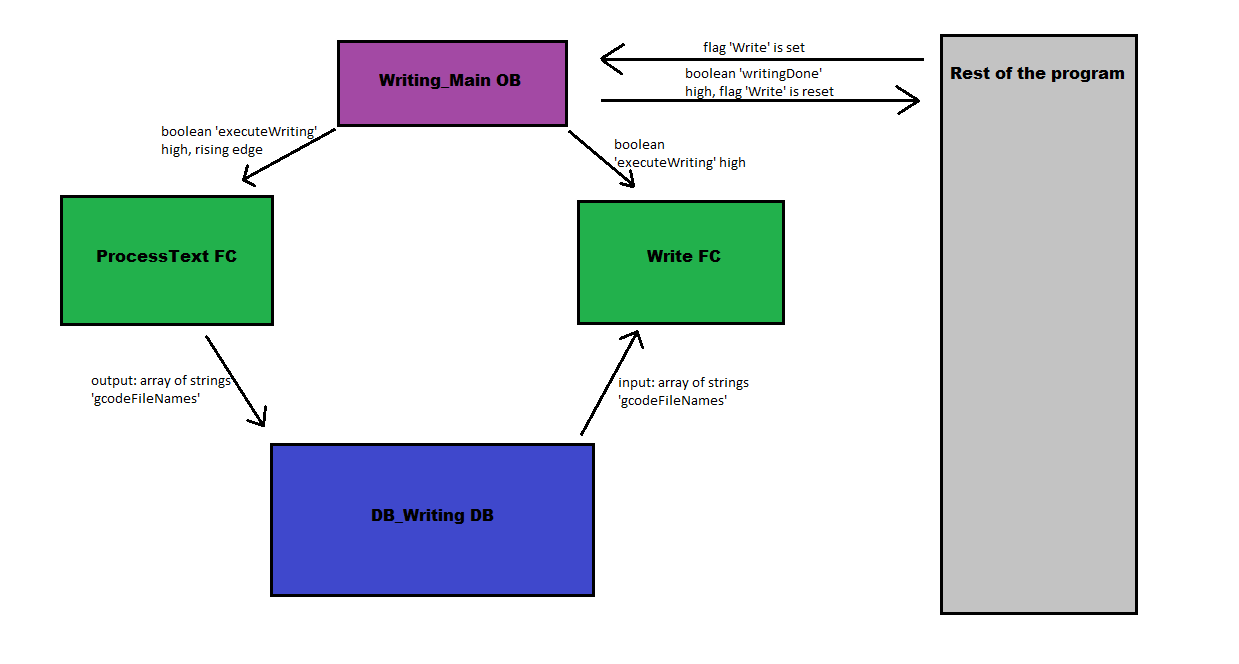


Now, the (boolean) flag can be used (as a boolean) and referenced when it is set in the HMI:



* + 1. **Program logic – general**

Without going in detail, an overview of the program is given: the OB ‘Writing\_Main’ checks if the flag ‘Write’ is set, only then it gives the commands to the writing function blocks. The boolean ‘executeWriting’ is set to high and when this is done, the ‘ProcessText’ FC will be activated (on a rising edge of the boolean). It reads the text input from the HMI and returns an array of strings, with each string being the name of a .nc command file. This array of file names is read by the ‘Write’ FC, which will execute the movement by first loading each file by searching in the memory of the PLC for the file name (FileHandler block), parsing each command file into a pathdata structure (Parser block) and then finally using the pathdata structure to execute the movement (MovePath block). The final step is to set the boolean ‘writingDone’ to high, after which the ‘Write’ flag is reset and the rest of the program can be executed. This explanation is illustrated in the diagram below:



* + 1. **Adding PackML logic for robustness**

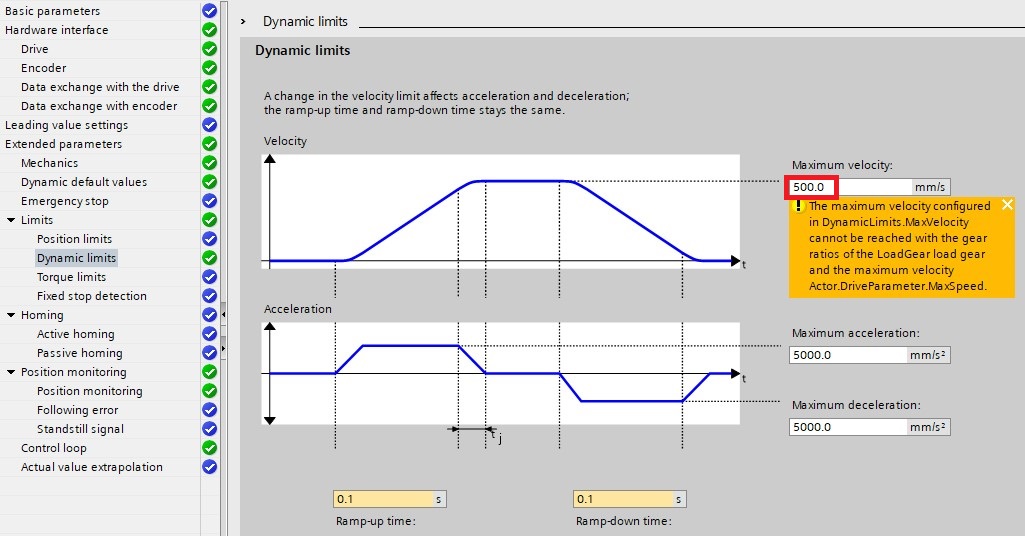
To make sure the program still functions correctly after aborting or stopping during the writing, some PackML logic had to be added into the writing function. When aborting during the writing, the axes get disabled so the movement is stopped in any case, but when going back into the execute state, the writing didn’t work anymore, a reset of the MovePath block is needed. This was implemented, along with a boolean that disables the MovePath block when the writing flag is not set, just to be sure.

* 1. **Final implementation - real POCFA**

When getting the program to work on the real POCFA, some issues came up.

* + 1. **Downloading the program to the PLC**

When downloading the program into the PLC, some weird behaviour was observed. The PLC and conveyor drives went into error state due to ‘faulty drives’. This was due to the **dynamic limits** set in the configuration of the conveyor axes, which were set too high to ever be reached by the conveyor drives. This was **changed from 500mm/s to 237.5 mm/s:**



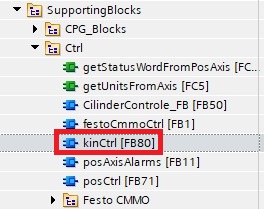
There were still internal communiciation errors going on, this might be due to the TO version update. After rebooting the POCFA, these errors were gone.

* + 1. **Downloading the program to the HMI**

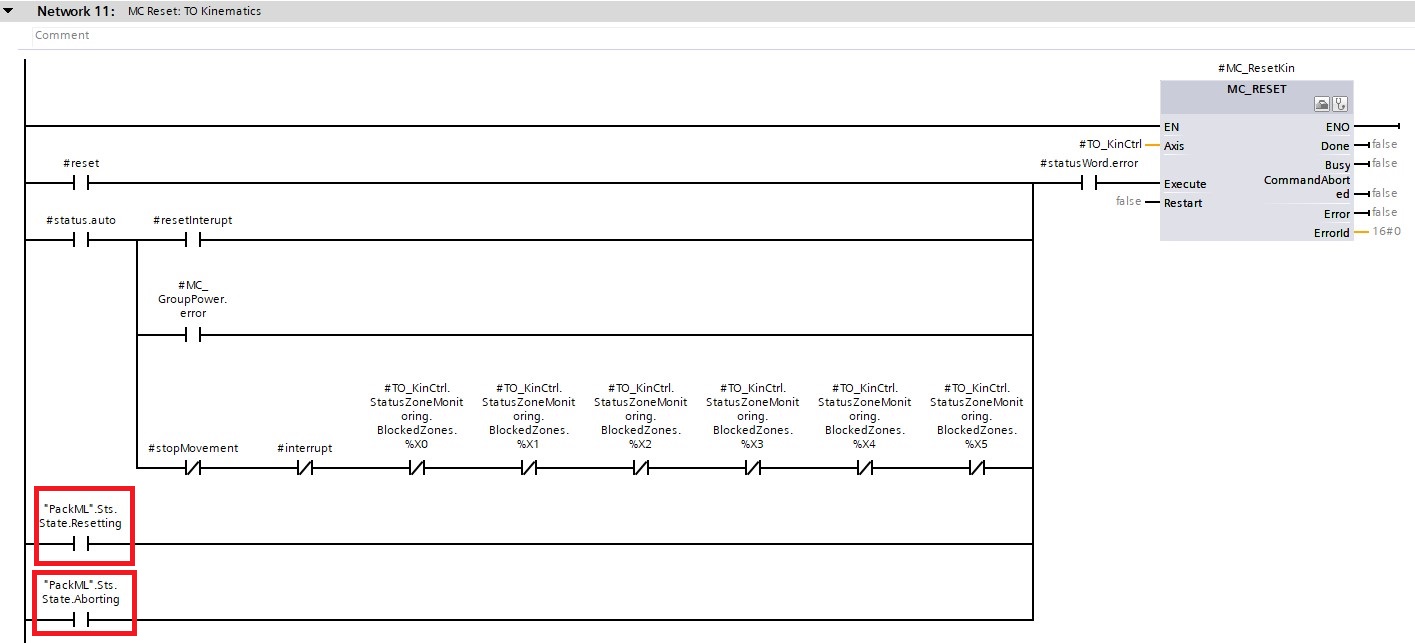
Because some things were added to the HMI, the HMI program had to be downloaded as well. After doing so, the wiring of some tags was strangely enough changed offline, INSIDE OF THE TIA PROGRAM, even though the HMI program had the wiring correctly before downloading, I cannot explain how this happened... **Make sure to check the HMI program before and after downloading!**

* + 1. **TO Alarm 801 for the positioning axes**

The last error to be solved is that every time when the machine is in ‘Resetting’ and ‘Aborting’ state, the TO alarm 801 gets thrown for all positioning axes: the axes are not ready to be used by the MovePath block. This is probably due to the timing of the enabling and disabling of the axes. The quick fix was to reset the TO kinematics once when in the ‘Resetting’ and ‘Aborting’ state, because playing with the timing of enabling and disabling blocks was complicated. The reset was done inside of the kinCtrl block:

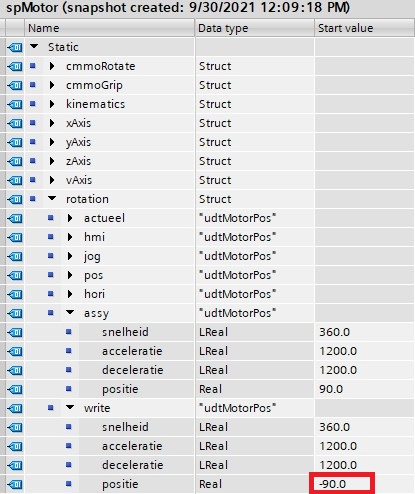


There, in network 11, the reset of the TO was already present, and some normally open contacts were added, so the reset would be done during the ‘Resetting’ and ‘Aborting’ states of the machine:



* + 1. **Getting the rotation gripper to turn left (write position)**

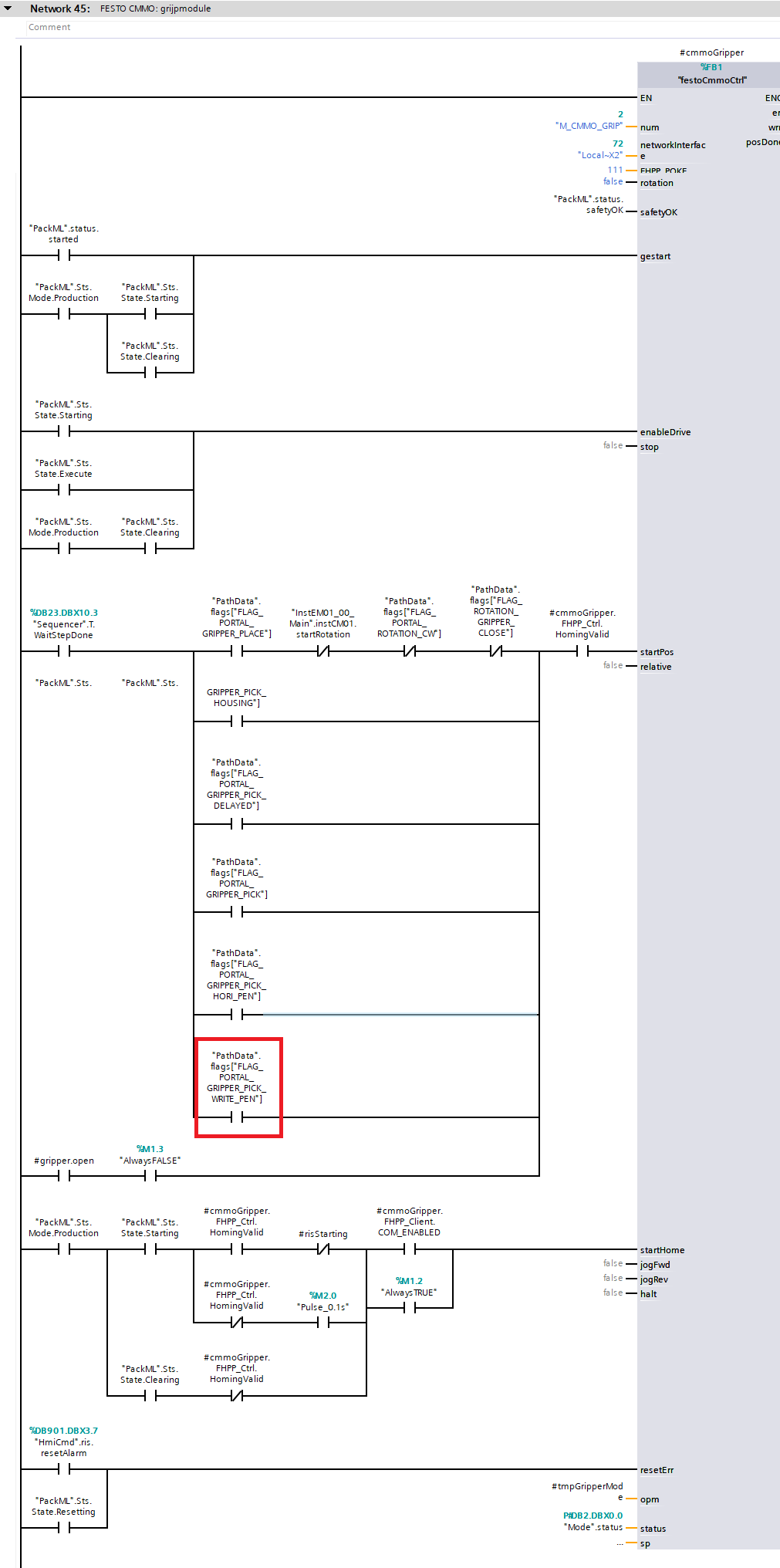
The flag ‘Rotation Pos Write’ is foreseen to get the rotation gripper in write position, but the position itself was set wrong in the DB ‘spMotor’. It was set as 270 degrees, but the gripper bumps into its SW limits when doing this, so this was set as –90, so it turns left instead of right (assy):



**However, the ‘Retain’ option must be turned off and on again to put the actual value equal to the start value, because the machine will retain its actual value (270) even when rebooting.**

* + 1. **Flag wiring bug**

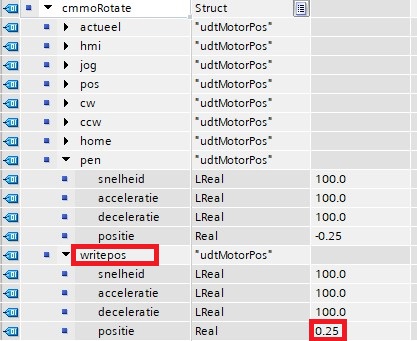
The flag ‘Portal Gripper Pick Write Pen’ was not wired, a normally open contact was added in NW 45 of the FB EM01\_CM05\_Portal\_CMMO:



* + 1. **Getting the portal gripper to turn 180 degrees (writing position)**

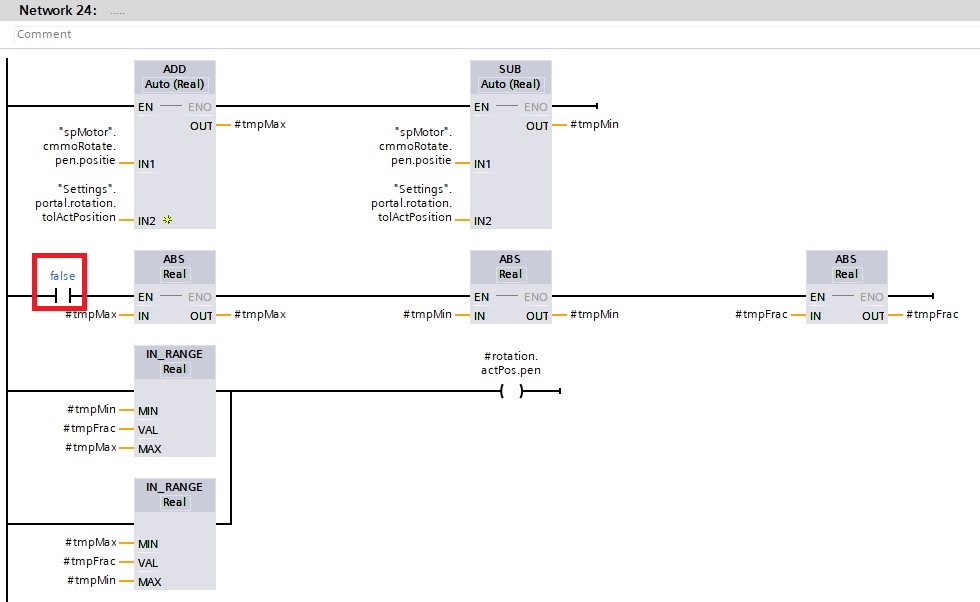
The portal has to be able to turn its gripper 180 degrees in order to pick up the pen from the rotation gripper, when the latter is in ‘Write’ position (-90 degrees, turned to the left, see 5.5.4).

* First, one of the rotation positions inside the ‘spMotor’ DB was adapted and renamed in order to distinguish the write portal rotation position:



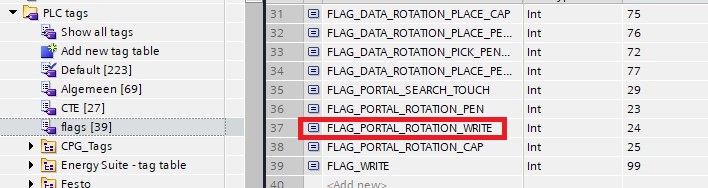
==> **the rotation position ‘Spring’ was renamed to ‘writepos’ and the position was set to 0.25 (corresponds to the rotation write position) instead of –0.25 (corresponds to the standard rotation pen position). The corresponding flag will be used to activate this position.**

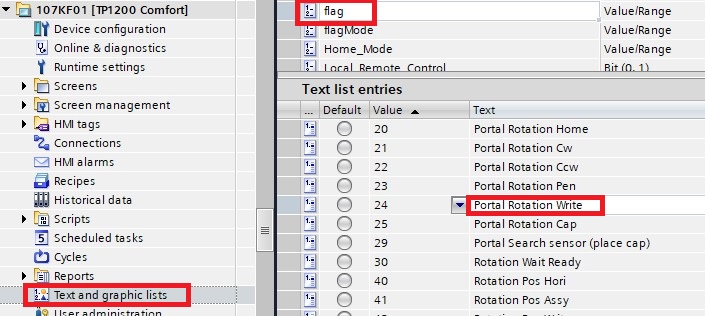
* Then, a weird section inside the network that determinates the actual position of the portal gripper rotation was disabled in NW 24 of the FB EM01\_CM05\_Portal\_CMMO:



As one can see, this section determined the absolute value of the tolerated positions, such that -0.25 (rotation pen position) as well as its absolute value 0.25 (which corresponds to the rotation write position) were accepted as being the pen rotation position. In other words, if the portal rotation position is at 0.25 and has to be rotated until –0.25, this didn’t happen because no distinction can be made between the two. Thus, it had to be disabled.

* Finally, the ‘Portal Rotation Spring’ flag (in the PLC tag table ‘flags’) and the corresponding text list entry in the HMI had to be renamed to ‘Portal Rotation Write’ to make a clear distinction:



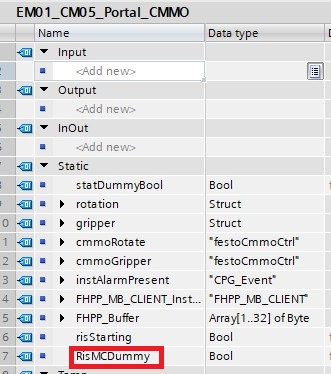


|  |
| --- |
| **IMPORTANT NOTE:**  **In the cycles, the portal rotation flags had to be adapted :**   * **IN ALL POSITIONS WHERE THE PORTAL ROTATION HAS TO BE SET TO ‘NORMAL’, THE ‘PORTAL ROTATION PEN’ FLAG IS USED** * **IN ALL POSITIONS WHERE THE PORTAL ROTATION HAS TO BE SET TO 180 DEGREES WITH RESPECT TO THE ‘PORTAL ROTATION PEN’, THE ‘PORTAL ROTATION WRITE’ FLAG IS USED** |

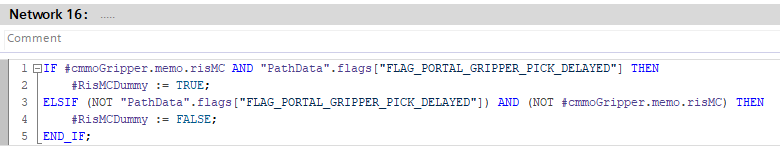
* + 1. **Picking with delay**

Because the CCW rotation happened already during the closing of the portal gripper (in the Rotation/pkCap position), the cap did not get unscrewed every time: the dismantle cycle lost some robustness in this step. This was solved by adding a pick flag with a delay after the picking motion:

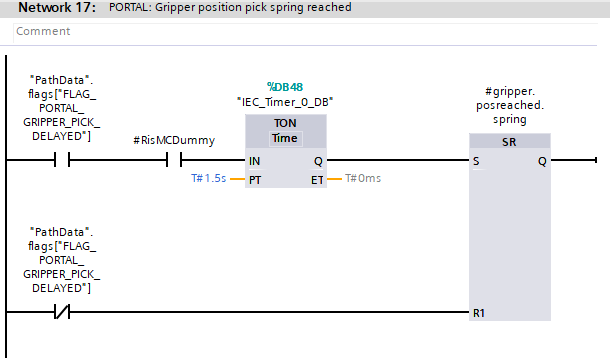
* First a static was added in the FB EM01\_CM05\_Portal\_CMMO



* This static will be set when the rising edge ‘#cmmoGripper.memo.risMC’ appears, and reset when the ‘Portal Gripper Pick Delayed’ flag is reset by the following network that was added in EM01\_CM05\_Portal\_CMMO:

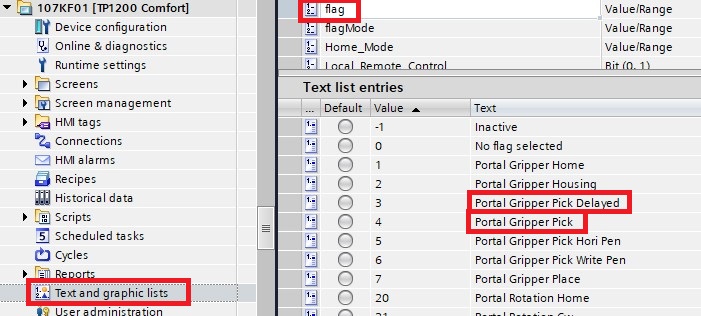


* The static will be used to activate the timer, added in NW 17 of EM01\_CM05\_Portal\_CMMO:



* Finally, some flag renaming was done to distinguish the pick flags. The flag ‘Portal Gripper Pick Cap’ was renamed to ‘Portal Gripper Pick’, this is now the standard pick flag. The flag ‘Portal Gripper Pick Spring’ was renamed to ‘Portal Gripper Pick Delayed’, which is now the pick action with delay. Also, the corresponding tekst list entries were renamed accordingly:





|  |
| --- |
| **IMPORTANT NOTE:**  **In the cycles, the portal gripper flags had to be adapted :**   * **IN ALL POSITIONS WHERE THE PORTAL GRIPPER HAS TO BE CLOSED WITHOUT DELAY AFTER, THE ‘PORTAL GRIPPER PICK’ FLAG IS USED** * **IN ALL POSITIONS WHERE THE PORTAL ROTATION HAS TO BE CLOSED WITH DELAY AFTER, THE ‘PORTAL GRIPPER PICK DELAYED’ FLAG IS USED** |

* + 1. **Sensor displacement - IMPORTANT**

At a certain point, the rotation gripper did not want to turn anymore. This was due to the sensor 322BG2 (in PLC tags/IO/BEWEEG), that did not go high anymore. This sensor HAS to be high when the portal is at z=325 (in WCS), such that the program is certain that rotating the rotation gripper will not result in a collision with the portal. The sensor was displaced only a few mm upwards, so it was shifted back again a few mm downwards.

* + 1. **Centering the text**

To center the text horizontally, the x of the begin position (and the x position after an enter to go to a new line) has to be dependent on the amount of letters that will be written on one line. Since the letters themselves are defined relative to the begin position, only the x of the begin position will have to be changed in function of the amount of letters in a line. Because 8 letters were needed in a line, 8 different (in x) begin positions were made, each shifted with half of the width of a letter.

To center the text vertically, the begin position has to be dependent on the amount of lines that have to be written. The same logic was followed.

* + 1. **Easter eggs**

I wanted the machine to be able to draw the Siemens logo when typing a certain password. When typing ‘siemens1’ or ‘siemens2’, the machine will draw the logo. The first password will result in the machine drawing the logo with the *Ingenuity for life*-slogan, the other logo will be without this slogan.

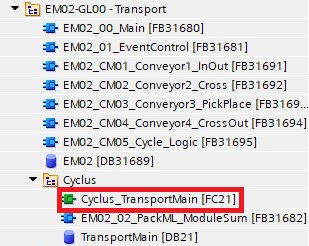
1. **Continuous cycle**
   1. **Main idea**

The goal is to alternate between the assembly and the disassembly cycle, but to switch to the next cycle the current cycle has to be finished. To finish the current cycle, some conditions must be met (for example: the inductive sensor at the end of the conveyor track which has to be set). Therefore, **a few bypasses of these sensors must be added in the right places**. All necessary code was put into one single OB and all needed tags were put into a single DB:

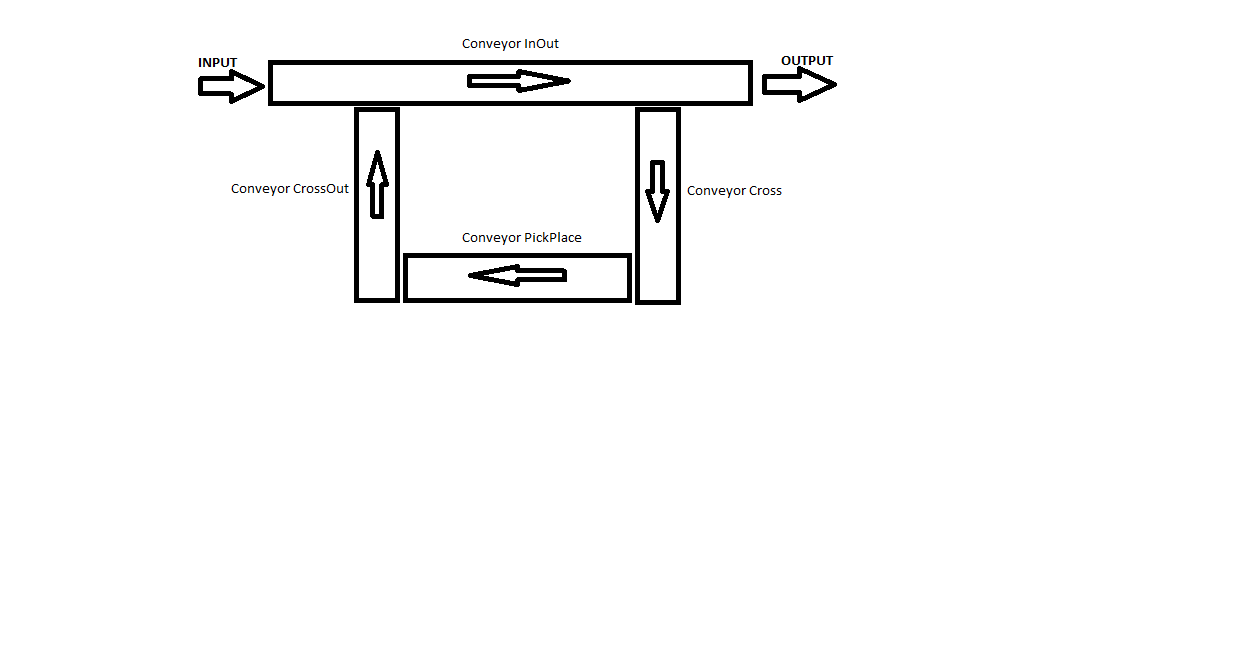


* 1. **Conveyor programming**

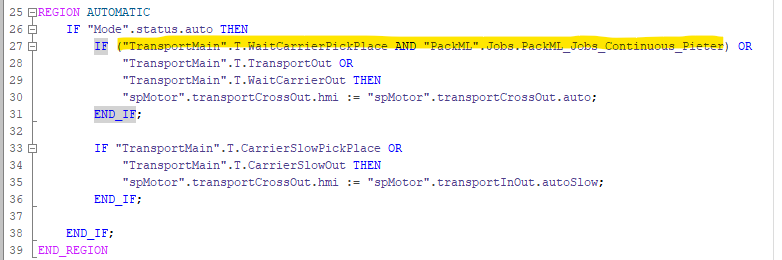
The transport cycle is programmed in the FC block Cyclus\_TransportMain, there one can find all (intermediate) statuses of the cycle. In this block, some bypasses for the input and output inductive sensors will be added.



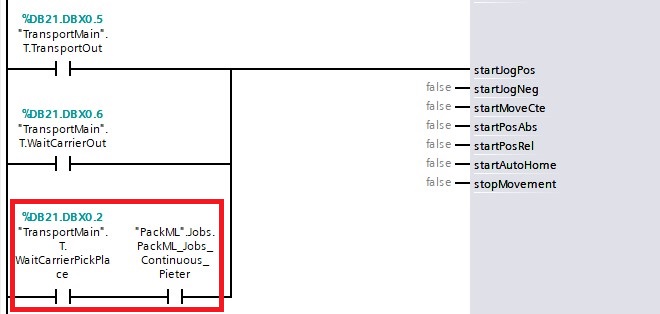
The conveyor system and naming is shown in the diagram below:



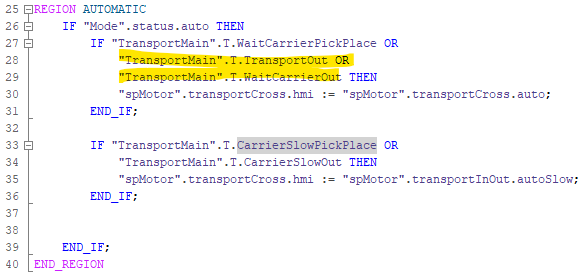
* The first problem was that the conveyor CrossOut did not run when the transport cycle was in status ‘WaitCarrierPickPlace’. This was needed because right after the portal has finished the assembly or disassembly, the new job is assigned and the carrier needs to get around the conveyor system. It has to be able to get over the CrossOut conveyor to arrive at the PickPlace conveyor again. This was enabled in the FB EM02\_CM04\_Conveyor4\_CrossOut:
  + NW 5:



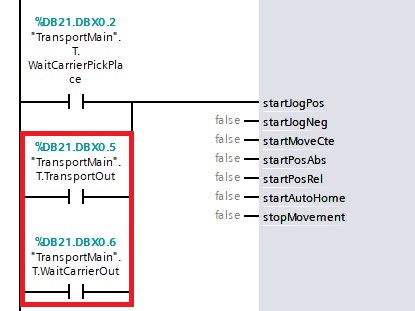
* + NW 6:



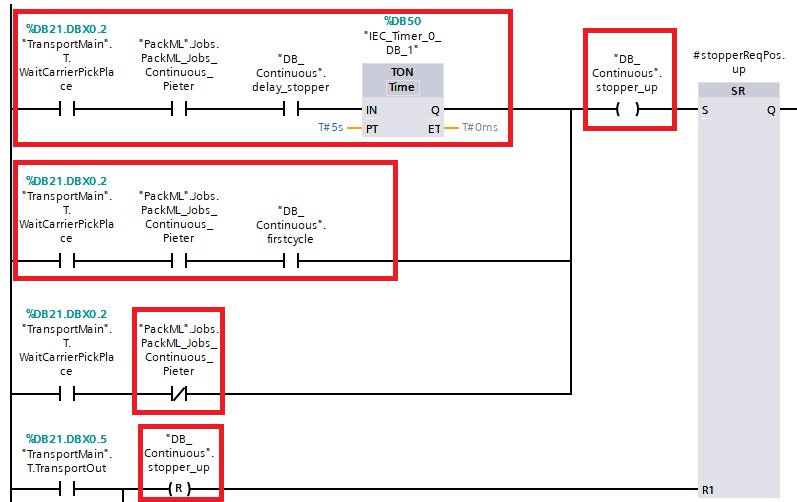
* The second issue was that the conveyor Cross did not run when an ejection cycle (after aborting, when getting back to the ‘execute’ state) was taking place. This is needed because if one aborts when the carrier is still on the conveyor Cross, it gets stuck there and all other conveyors keep running forever (carrier has to set inductive sensor at the end of the InOut conveyor). This was solved in the FB EM02\_CM02\_Conveyor2\_Cross:
  + NW 7:



* + NW 8:



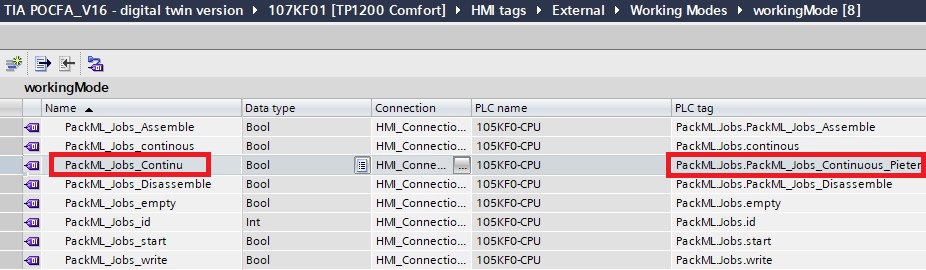
* The final problem was that the stopper on the PickPlace Conveyor went up too quickly (the cycle is ended right after the falling edge of the PickPlace inductive sensor), when the carrier is still right on top of the stopper. Therefore, a delay of 5s was introduced such that the stopper only went up after the carrier was already away. This was done in NW 9 of the FB EM02\_CM03\_Conveyor3\_PickPlace:



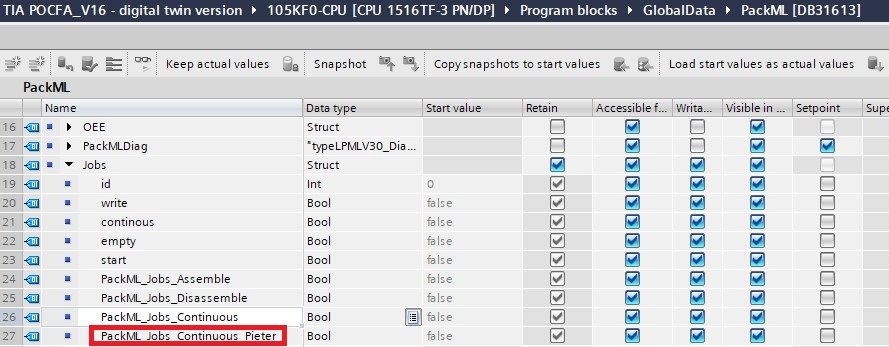
* 1. **HMI programming**

The HMI button was already wired to a HMI and PLC tag. However, there were two PLC tags for this functionality and the wiring was in some places done with one tag and sometimes with the other. To be clear, the continuous cycle did NOT WORK, even though a lot of wiring was already done. Probably this is due to the many people who worked on the machine before.

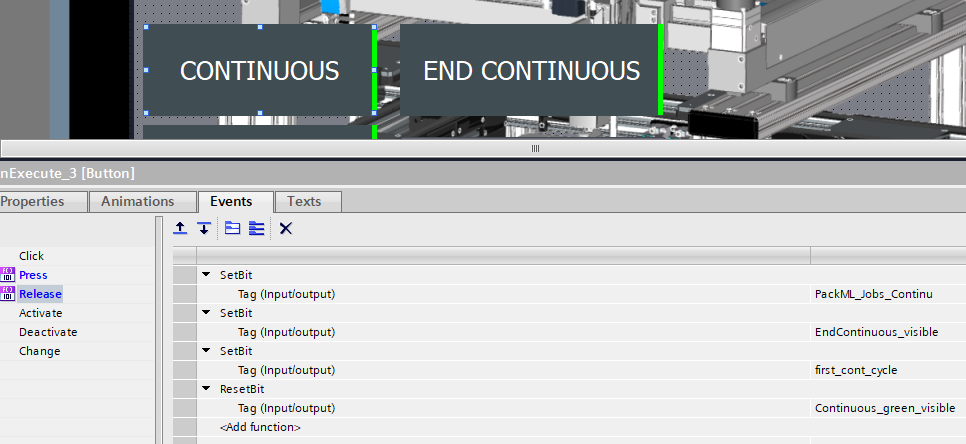
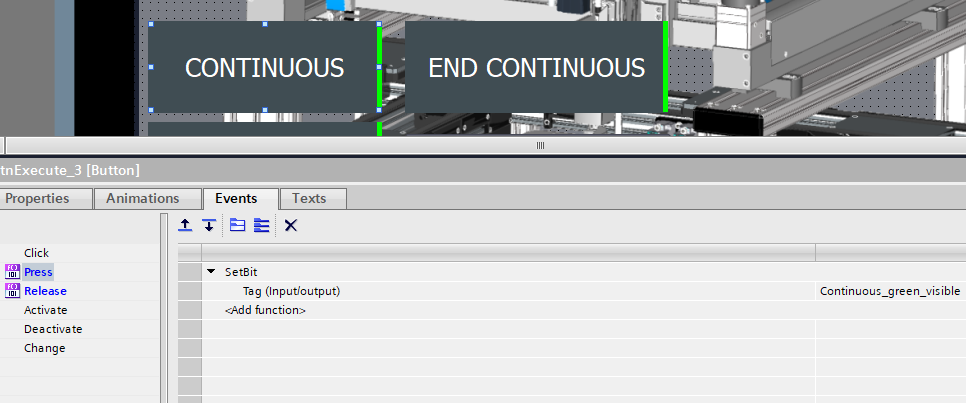
* To make things more clear (and to avoid having to rewire and delete all the faulty wiring of the two tags), I added my own tag for this functionality and coupled it to the button.
  + Firstly, in the HMI tags (in the HMI’s External/Working Modes/workingMode tag table):



* + Then, in the PackML DB (under the GlobalData folder in the PLC):



* Then, the coupling on the button was done as follows:



* The PLC tag PackML\_Jobs\_Continuous\_Pieter will be set and remain high until the ‘END CONTINUOUS’ button is pressed:

