MATRIX | SMART FACTORY

PLC Control of Industrial Robots and Industry 4.0

MATRIX

CP5413

www.matrixtsl.com

Contents

Introduction

This guide describes how to set up a Siemens S7 PLC to control the Smart Factory and the All-Code Robot Arm. This document is product code CP5413.

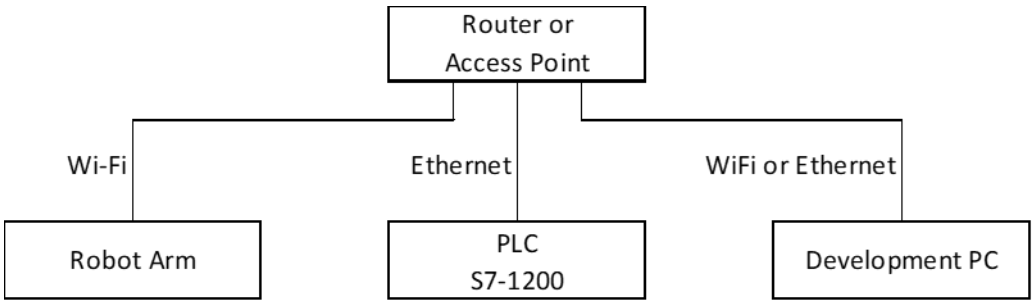
A framework project is available that includes function blocks to simplify command of the robot arm. The example programs include a web interface that allows the Smart Factory to controlled via a web browser. The accompanying download with the examples and Motion control technology object software is product code AU0611: Siemens S7 smart factory / arm examples.

You should already be familiar with the basics of PLC programming. For an introduction to ladder logic programming see https://www.solisplc.com/tutorials/siemens-tia-portal-programming-example-basic-ladder-logic-instructions

Some of the example programs that are provided use a programming construct called a state machine. For a description of how state machines are implemented in PLCs see: - https://www.solisplc.com/tutorials/programming-a-state-machine-in-ladder-logic

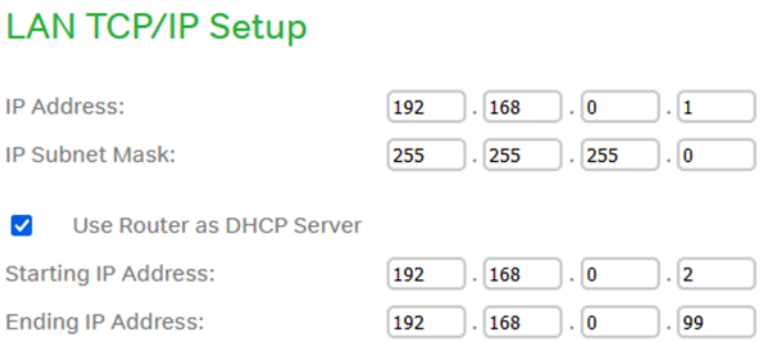
Connecting the Robot Arm to the S7 using Wi-Fi

To get the Siemens PLC to work with the Robot Arm they must be on the same network. The Robot Arm has a Wi-Fi connection and the S7-1200 PLC has an Ethernet connection. To connect the two we need a wireless router. It is convenient if this is the same office or home router that is being used with the PC that will be used for programming.
In the absence of a home or office network an ad-hoc network could be created using a compact router such as TP-Link TL-WR802N



Static IP Address

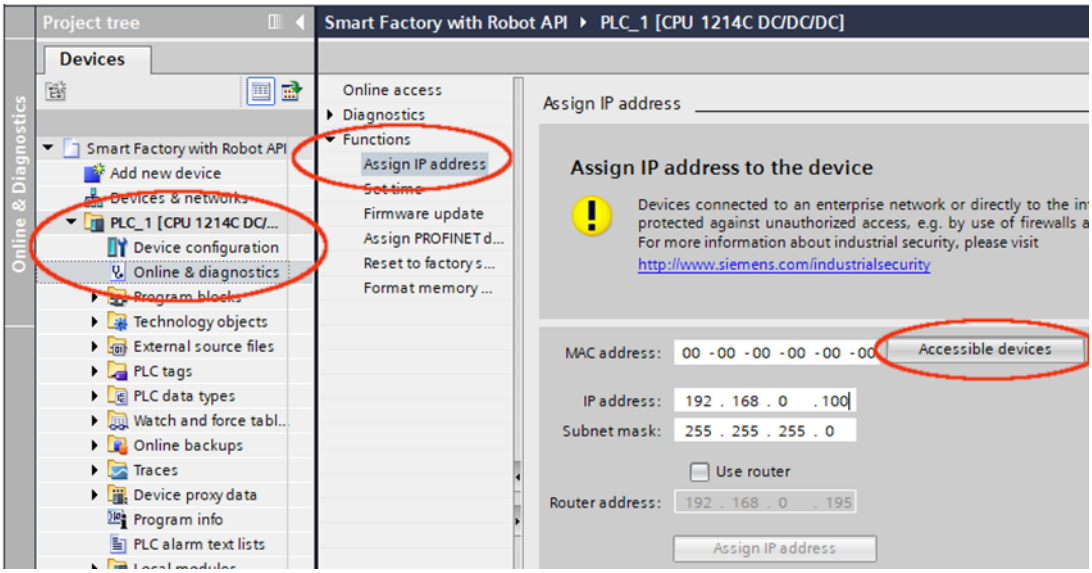
The PLC requires a static IP address. This can be configured in your router or provided by your network administrator. Generally this is just a case of restricting the range of addresses available for DHCP. The image below shows the LAN setup page a typical home router. The ending IP address of the DHCP server has been set to 192.168.0.99. Any addresses above that value are available for use as static addresses.



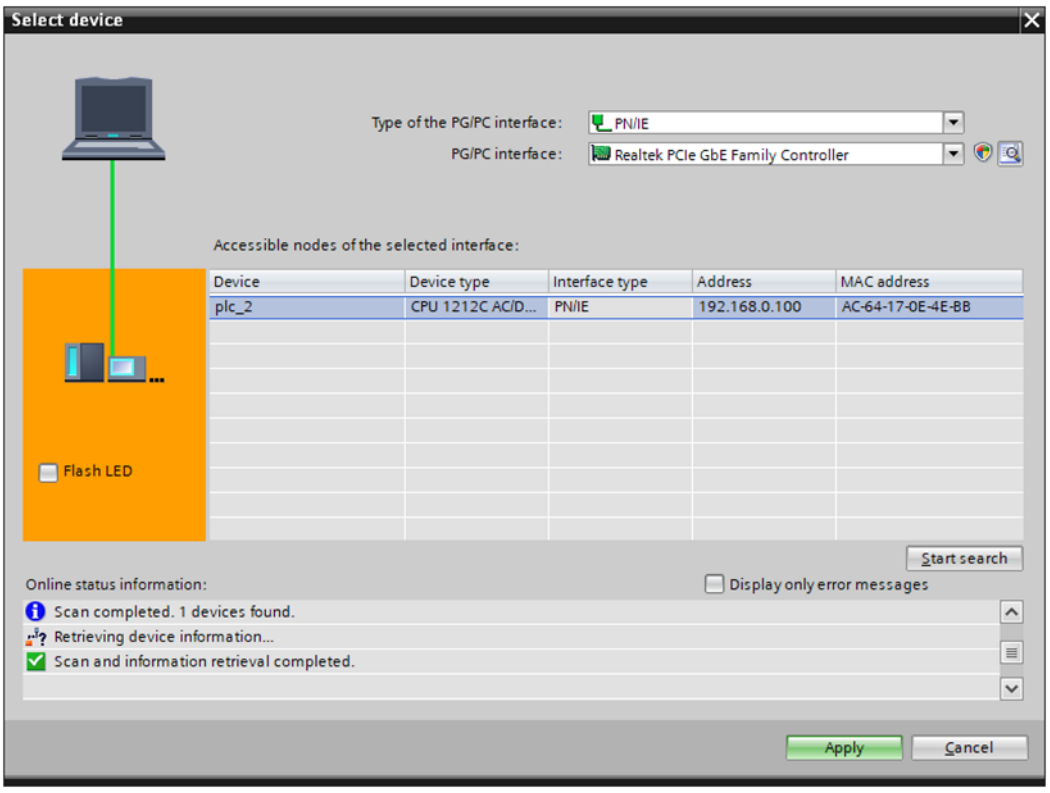
Network Address of PLC

Begin by downloading the project framework or one of the example projects and opening it in the Siemens TIA Portal software.

To assign an address to the PLC, locate the PLC in the Devices tree and select Online and diagnostics. From the online panel choose Assign IP address. Before the IP address can be assigned, the software must find the PLC. Click on the Accessible devices button to scan the local network for the PLC.



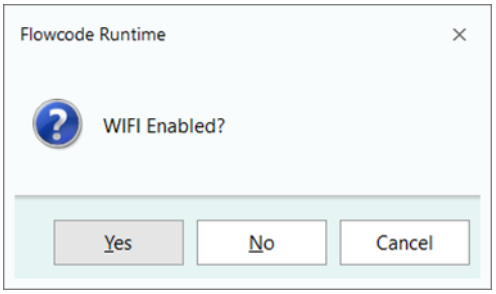
While scanning for accessible devices check that the Type of the interface is “PN/IE” and that the interface is the one on the PC that is physically connected to the network. Click on the Start Search button to find the PLC. Once it is found, clicking the Flash LED check box will make the RUN/STOP ERROR and MAINT LEDs on the PLC flash. This can be helpful if there is more than one PLC on the network as it ensures that we are connecting to the right one.



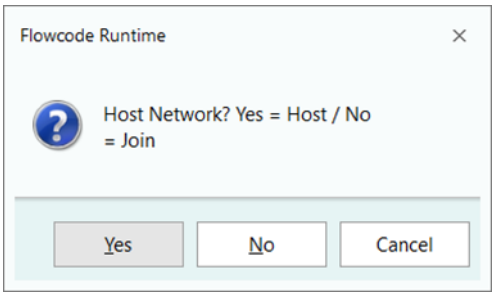
After finding the PLC, click on the Apply button to go back to the Assign IP address screen and click on the Assign IP address button to set the address.

Network Address of Robot Arm

The Wi-Fi networking of the robot arm can be configured using the Robot Arm V2 Control Apps that are available from the Matrix website. First connect the robot arm to a PC using a USB cable. Open the Configure App and select USB as the Comms Method. Click the ‘Go’ button in the top-left corner to start the app and click the Configure WIFI button. A series of dialog boxes guide the user through the configuration process.



Choose ‘Yes’.

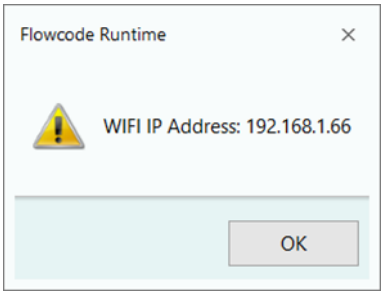


Choose ‘No’ to join the existing Wi-Fi network.

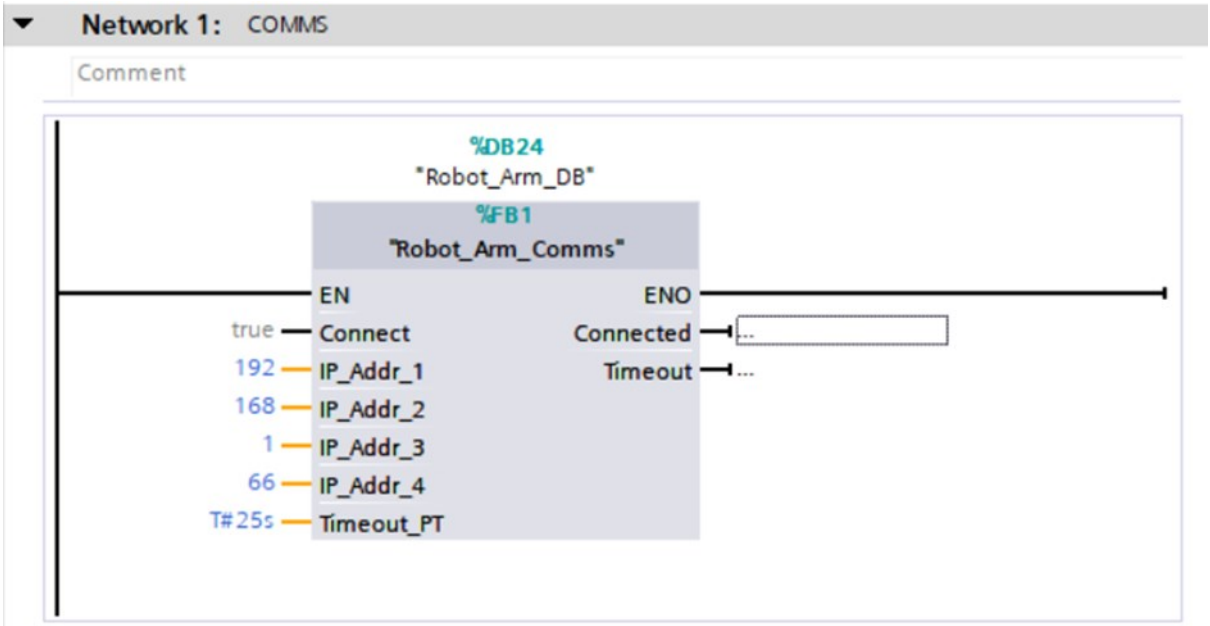
Enter the name of the Wi-Fi network that you wish to join.

Enter 1245 as the port number.

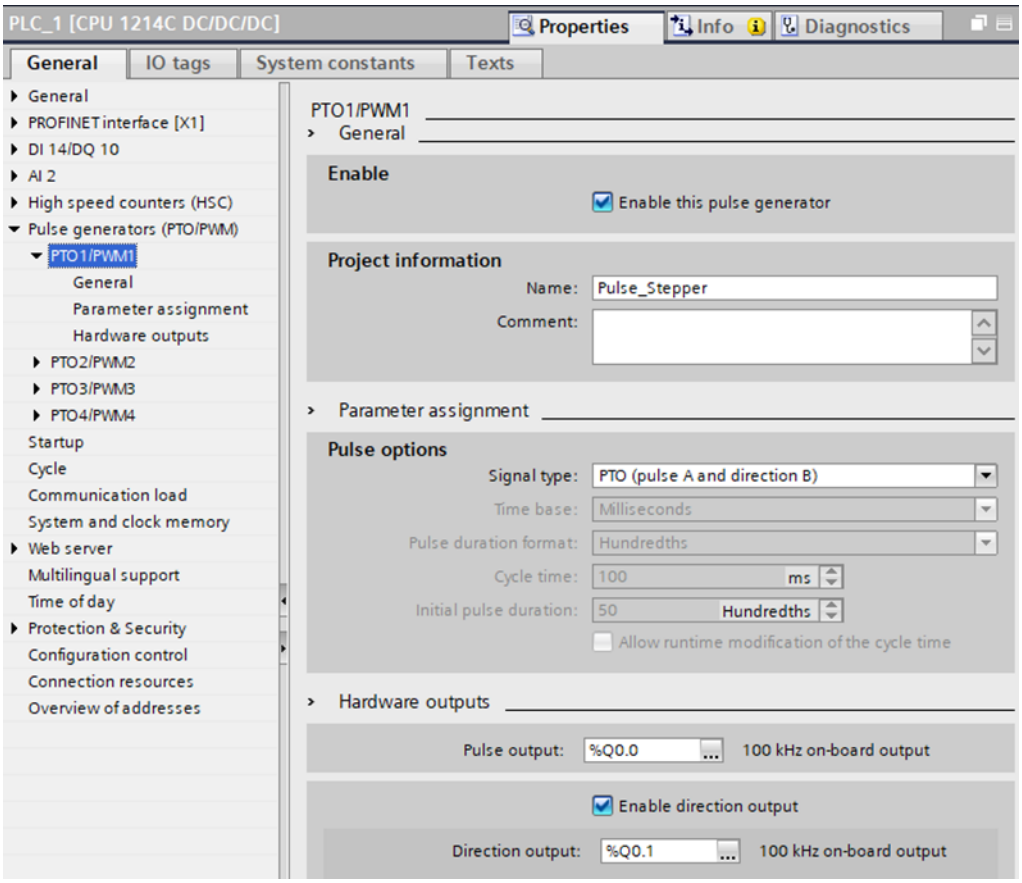
After entering these settings, stop the program and power cycle the arm. When the arm powers up, start the program again. Click on the WIFI Get IP button to learn the IP address that the Wi-Fi router assigned to the robot arm.



Make a note of this address. The function block Robot_Arm_Comms is described in the section on function blocks for the robot arm. This block maintains the connection with the robot and will require its IP_Addr inputs to match the address of the robot arm. The image below shows where the octets 192,168,1 and 66 are input to this block. These values should be replaced according to the address that is found in the step above.



The project framework encapsulates the stepper motor in the Smart Factory gantry as a Motion Control Technology Object. The pulse output for driving the stepper is assigned to output %Q0.0 (DQa.0) and the direction control is assigned to output %Q0.1 (DQa.1). The home position limit switch is assigned to input %I0.4 (DIa.0). When starting from the project framework this object has already been configured and the function blocks in the Motion Control portion of the instruction set can be used directly.
The fast output pulse is configured in the 'Device configuration' tab of the PLC. The pulse generator PTO1 is assigned to output %Q0.0 with %Q0.1 as a direction output. This configuration has already been done in the example programs.



The Motion Control Technology object has the type 'TO_PositioningAxis'. It builds on the pulse generator and encapsulates many of the operations that are needed for driving the gantry.
The parameters can be input using the configuration screen. The Function view screen guides the process of configuring the object. It is assigned to the pulse generator described above and millimeters are chosen as the unit of measurement. Mechanical parameters and the homing procedure are configured in this section.

Basic parameters

Axis name: Axis_Stepper
Drive: PTO (Pulse Train Output)
Position Unit: mm
Pulse Output: %Q0.0
Direction Output: %Q0.1

Extended parameters

Pulses per revolution: 200
Load movement per revolution: 40mm

Dynamics

Maximum velocity: 2000 mm/s
Start/stop velocity: 1.0 mm/s
Acceleration & Deceleration: 2000 mm/s^2

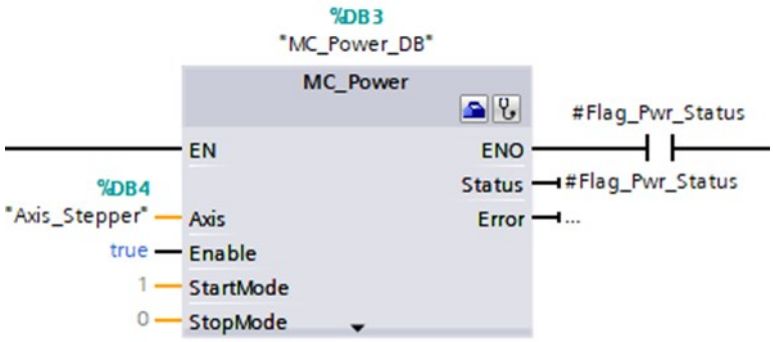
Active Homing

In active homing, the program calls a home function and the axis will hunt for the homing microswitch automatically. If passive homing were selected then the program must start the axis moving, stop when it detects the switch and then set the home position.

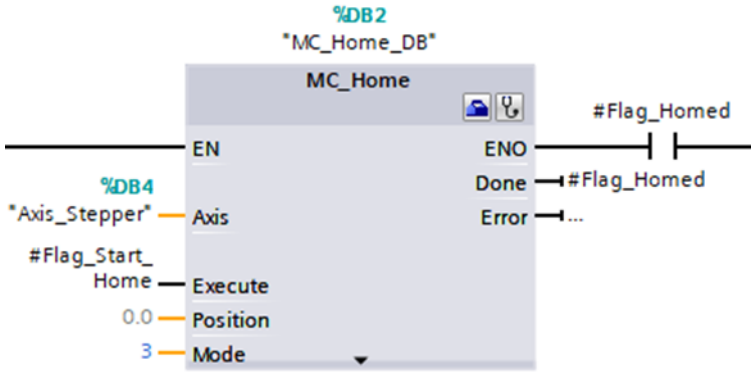
Input homing switch: %I0.4
Homing direction: Negative
Side of homing switch: Top side
Approach velocity: 20.0 mm/s
Homing velocity: 10.0 mm/s

Once configured, a Commissioning panel is available that allows the programmer to test the axis to determine if the motion is correct before writing the program.

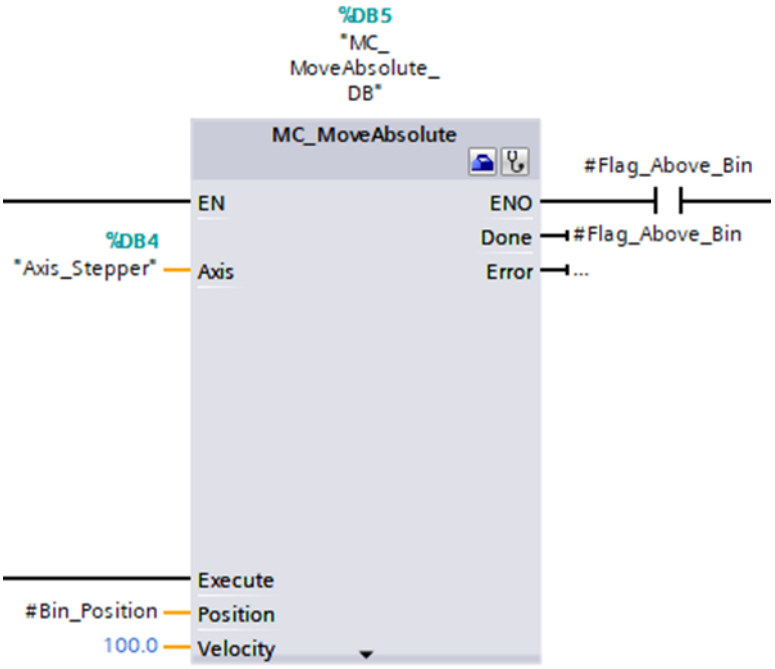
The MC_Power block must be called before any other function block. The Axis input is “Axis_Stepper”, Enable is true, StartMode = 1 and StopMode = 0. Once the object is ready the Status output becomes true and the rest of the program can continue.



At the beginning of an operation, it is a good idea to home the stepper. Calling the MC_Home block will cause the gantry to automatically drive into the limit switch and learn its home position. The homing procedure starts the Execute input changes from false to true. When the Done output becomes true, the rest of the operation can continue. The Position input should be set as 0.0 and the Mode input as 3.



The MC_MoveAbsolute block can be used to send the gantry tool to any position relative to its home position. The move will start when the Execute input goes from false to true. The Position input is the desired position measured in millimetres from the home position. The Velocity is measured in mm per second, 100 mm/s is a suitable velocity to use. The technology object will automatically generate a smooth acceleration ramp up to the target velocity and decelerate as the end of the move approaches. When the move is completed, the Done output will become true to signal that the next part of the program can continue.



The robot arm is shipped with an Application Programming Interface that allows it to be controlled via Wi-Fi.
The project framework contains function blocks that package the API of the AllCode Robot Arm. This makes it easy to use the Robot Arm with an S7 PLC.

Robot_Arm_Comms

This block attempts to establish a connection with a robot arm at the specified IP address. It must be run in a cycle before any of the API function blocks are used. If the connection to the arm is lost then it will attempt to re-establish the connection as soon as it becomes possible.

Connection	Direction	Data Type	Description
Connect	Input	Bool	Set to true to attempt to establish and maintain a connection.
IP_Addr_1 ~ 4	Input	Byte	The IP address of the robot arm, this must be on the same subnet as the PLC.
Timeout_PT	Input	Time	This is the maximum time that the API function blocks will wait for the arm to re-spond.
Connected	Output	Bool	Becomes true to indicate that the connection is established.
Timeout	Output	Bool	Becomes true when one of the function blocks has waited the maximum time without a response from the robot arm.

RA2_Get_API_Version

This block sends a query to the robot arm to check the API version in the arm firmware. This library is compatible with version 1. This block is useful as a check that the communication has been established and is working correctly.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the query.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the query is sent or the block is waiting for a response.
Error	Output	Bool	True for one cycle if the query fails.
API_Version	Output	Word	If successful, this output will hold the API version number. If it fails then the output will be 16#FF.

RA2_Home

This block sends the robot arm to its home position. This is necessary at least once because all the movements are relative to the homing microswitches. Depending upon where the arm is when this block is called, it can take up to 20 seconds to complete the operation.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Done	Output	Bool	This output is true for one cycle at the successful completion of the operation.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Home_Axis

This block sends one single axis of the arm to its home position.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Axis	Input	Byte	Identifies the axis to be homed: 0 = Base, 1 = Shoulder, 2 = Elbow.
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Get_Position

This block gets the current position in steps of each axis of the arm. It should give zero for each axis after a home operation and a positive number of steps if the arm is away from the home position.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the query.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the query is sent or the block is waiting for a response.
Error	Output	Bool	True for one cycle if the query fails.
Axis_A, Axis_B, Axis_C	Output	int	If successful, these outputs will hold the step counts for each axis. If it fails then the outputs will all be -1.

RA2_Goto_Position

This block will cause the arm to move to a previously stored position. These positions would typically have been stored while controlling the arm with a pendant. The arm will take the shortest path from its current position. In the factory there are likely to be obstacles around the arm. The programmer should allow for this by programming a series of positions that navigate around the obstacles.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Pos_Index	Input	Byte	Identifies the index of the stored location.
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Set_Motor

This block sets the position in steps of a single motor.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Motor	Input	Byte	Identifies the motor to be set: 0 = Base, 1 = Shoulder, 2 = Elbow.
Coord	Input	int	This is the required position of the motor in steps.
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Set_All_Motors

This block sets the positions of all three motors in steps.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Coord_0 ~ 2	Input	int	These are the required positions of the motors in steps.
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Get_ToolXYZ

This block gets the current position in x,y,z coordinates of the tool at the end of the arm. The units are in millimetres with the pivot point of the base as the zero position.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the query.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the query is sent or the block is waiting for a response.
Error	Output	Bool	True for one cycle if the query fails.
Tool_X, Tool_Y, Tool_Z	Output	int	If successful, these outputs will hold x,y,z coordinates of the tool. If it fails then the outputs will all be -1.

RA2_MoveToXYZ

This block moves the arm to position the tool at the specified x,y,z coordinates. The units are in millimetres with the pivot point of the base as the zero position.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Tool_X, Tool_Y, Tool_Z	Input	int	These are the required hold x,y,z coordinates of the tool.
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Set_Gripper

This block sets the position of the gripper tool at the end of the arm. The position can be set between 0 = fully open and 255 = fully closed. The jaw are sprung so that the servo motor can be driven to the fully closed position even if an object is between the jaws.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Gripper	Input	Byte	The required position of the gripper tool. 0 = fully open and 255 = fully closed
Done	Output	Bool	This output is true for one cycle at the successful completion of the move.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Get_Colour

This block reads an RGB colour sensor connected to the arm. It is necessary to ensure that the colour sensor is attached before calling this block since attempting to read a sensor that is not there will cause the arm to stop responding until the next power cycle. The output is a structure of type Colour that contains one byte for each of the three colours detected by the sensor.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.
Colour	Output	Struct	A structure of three bytes: Red, Green and Blue that contain the sensor values that are read.

RA2_Disable_Motors

This block cuts power to all three motors allowing the arm to be moved by hand. If this is used then a home instruction must be used to re-enable the motors and allow the arm to learn its position.

Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

RA2_Set_LED

This block sets state of one of the five LEDs on the arm to on or off. This overrides the standard functions of the LEDs.

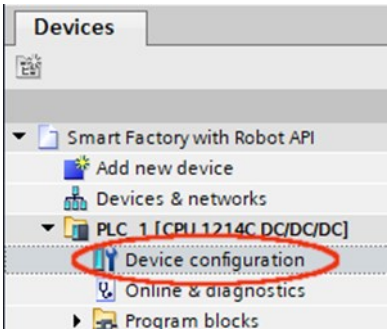
Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
LED	Input	Byte	Selects which LED to set. 0 ~4
LED_On	Input	Bool	True to switch the LED on, false to switch it off.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

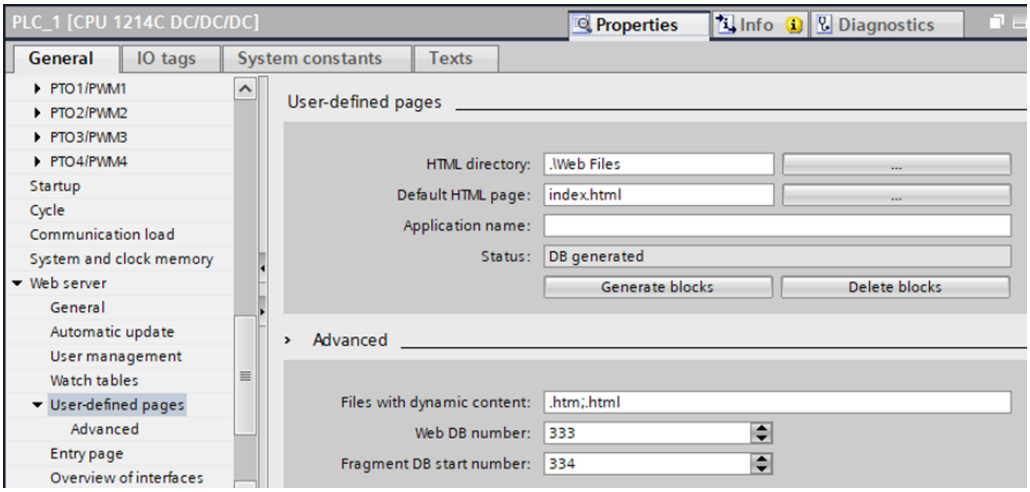
RA2_Auto_LEDs

This block cancels the effect of RA2_Set_LED and returns LEDs to their standard functions.

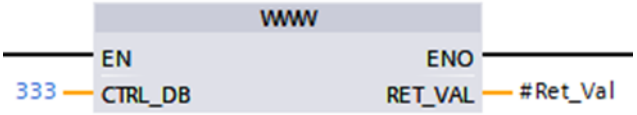
Connection	Direction	Data Type	Description
Req	Input	Bool	A rising edge on this input starts the operation.
Done	Output	Bool	This output is true for one cycle at the successful completion of the query.
Busy	Output	Bool	True while the operation is in progress.
Error	Output	Bool	True for one cycle if the query fails.

The S7 PLC has a built in web server that can be used to control and application and to report its status. The webserver can in effect be used as an HMI. The project framework has been configured to include user defined pages within its webserver. The pages to be included are contained in a folder named ‘Web Files’ within the project folder and the first page to be displayed will be ‘index.html’.
If these pages are edited then the data block in the PLC must be updated. Locate the PLC in the devices tree and select Device configuration. In the Properties panel for the PLC, locate the User -defined pages part of the Web server section and click the Generate blocks button. This will overwrite the existing data block with the newly edited pages.





Once the blocks are generated, the WWW function block handles updating and serving the web pages. This block is already included in the project framework. There is nothing for the programmer to do with this block apart from including it in the program so that it runs in the main cycle. The CTRL_DB input is set to 333, the default block number that is generated in the step above.



Once the webserver is enabled and configured, make a note of the IP address of the PLC (see section 1) and type it into the address bar of a web-browser. It may be necessary to prefix the address with https:// for example:

```
https://192.168.1.150
```

Live Variables in Web Pages

The easiest way to include a live variable in a web page is to write the variable name in the html file as it would appear in a program and put ':=' in front of it and ':' after it. For example; if the program includes a data block named "Web_Data" with a variable inside it named "Count_Fe" then this variable would appear in the program as:

```
"Web_Data".Count_Fe
```

If the web page includes a piece of html code of the form:

```
<div>:="Web_Data".Count_Fe:</div>
```

Then, when the page is served, the variable name will be replaced with the current value of the variable.

The variable is only updated when the page is served, if the program writes a new value to the variable it will not be updated on the page. The simplest way to fix this is to include the line

```
<meta http-equiv="refresh" content="1">
```

at the top of the page to force the page to periodically reload. This is easy to do but the page will be seen to flicker as it re-loads. A more elegant approach is to place the variables in a separate page and use jQuery and AJAX to update the variables on the main page. An example of this approach is included in the project framework. For more information on jQuery and AJAX see online resources such as https://www.w3schools.com/js/js_ajax_intro.asp

If the web page is expected to write to the variable as well as read it then an html comment declaring AWP_In_Variable must be included at the top of each page that references that variable. If the program includes a variable named "Web_Data".Trigger_1 then the comment

```
<!-- AWP_In_Variable Name='"Web_Data".Trigger_1'-->
```

must be included. The variable can then be updated by posting a new value. The web pages included in the project framework contain an example of this.

The examples are product code AU6011 which can be downloaded from the Matrix web site. The following example programs are provided to demonstrate the use of Smart Factory with the Siemens S7-1214 PLC: -

“Smart Factory Siemens S7-1200.zap15”

This program runs the gantry and the conveyor only. Aluminium and steel counters are sorted by the reject booms and plastic counters run off the end of the conveyor.

“Smart Factory with Robot Demo.zap15”

This program builds on the previous one. When a plastic counter is detected, the conveyor stops and a robot arm is triggered to pick up the counter. The robot holds the counter against a colour sensor and places it in a bin depending upon its colour. This program includes a web interface that presents the user with a count of the different types of counter that have been sorted and enables picking from any of the three hoppers.

“Smart Factory Framework.zap15”

The project includes all the necessary configuration for the various technologies that have been used in the previous programs. It also includes the function blocks for control of the robot arm. It does not include the higher level programming of the components. Use this project as a starting point for developing your own programs.

Opening the Example Program

1. Download the project archive.
2. Open TIA Portal version 15 or later.
3. In the Project View, select “Retrieve…” from the Project menu.
4. Select the project archive file and choose a directory to store the project.

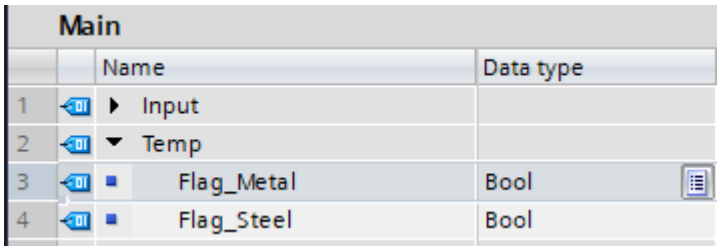
Sorting Counters – Sensor Gates and Reject Booms

The smart factory is built as shown in CP7329, Worksheet 4. The S7-1214 PLC is connected as follows: -

Connection	Device	PLC Tag Name
Inputs Block: 0	First light gate	%I0.0 Sensor_First_Light
Inputs Block: 1	Inductive sensor	%I0.1 Sensor_Inductive
Inputs Block: 2	Capacitive sensor	%I0.2 Sensor Capacitive
Inputs Block: 3	Final light gate	%I0.3 Sensor_Final_Light
Motor Outputs: 0	Steel reject boom	%Q1.0 Reject_Steel
Motor Outputs: 1	Aluminium	%Q1.1 Reject_Al

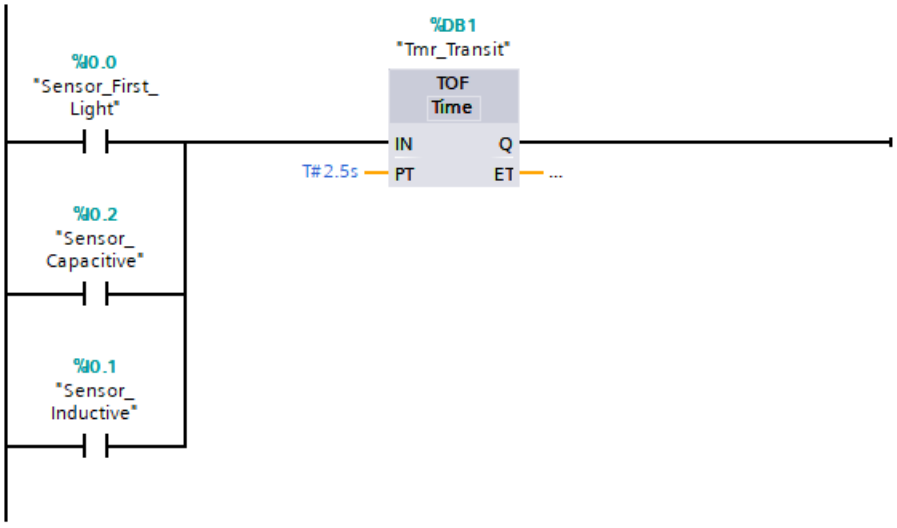
The PLC Program is named Sorting and runs as the Program cycle. It will run continuously, starting again as soon as it finishes.

Two local variables are declared, Flag_Metal and Flag_Steel, both of type Bool.



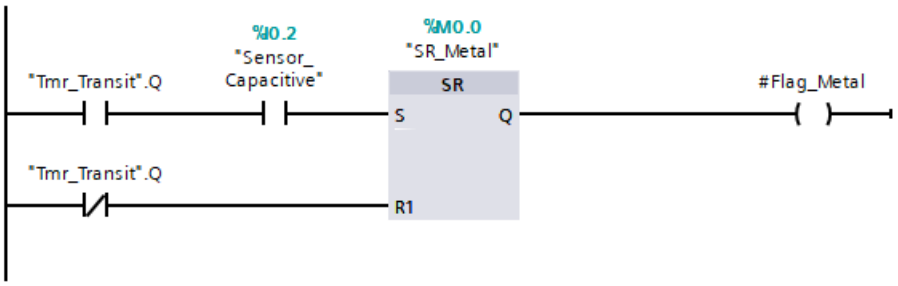
Network 1: TRANSIT TIMER

The first light gate or either of the following sensors will trigger the transit timer. Once triggered, the timer will stay on for the length of time taken to transit from one end of the conveyor to the other. The purpose of this timer is to keep the reject booms out until the counter has completed its transit.



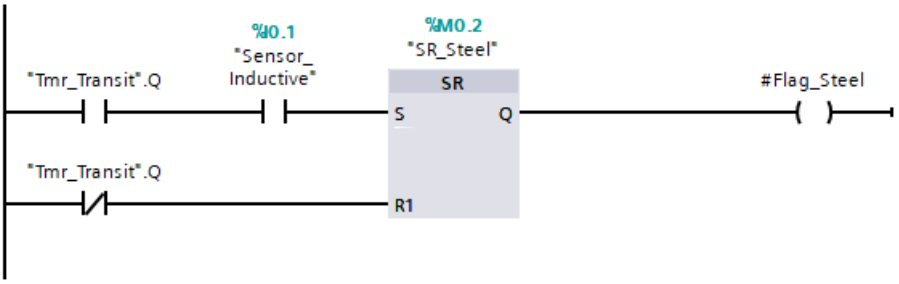
Network 2: LATCH METAL FLAG

This flag is latched when the capacitive sensor tells us that we have a metal counter. It will stay latched until the transit timer times out (the counter has completed its transit).



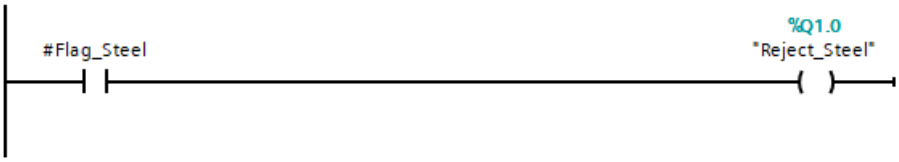
Network 3: LATCH STEEL FLAG

This flag is latched when the inductive sensor tells us that we have a steel counter. It will stay latched until the transit timer times out.



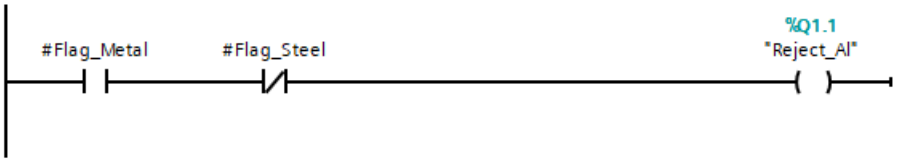
Network 4: SORT STEEL

As long as we have a steel counter, the reject boom is operated. This will be from the inductive sensor pulse until the transit timer times out.



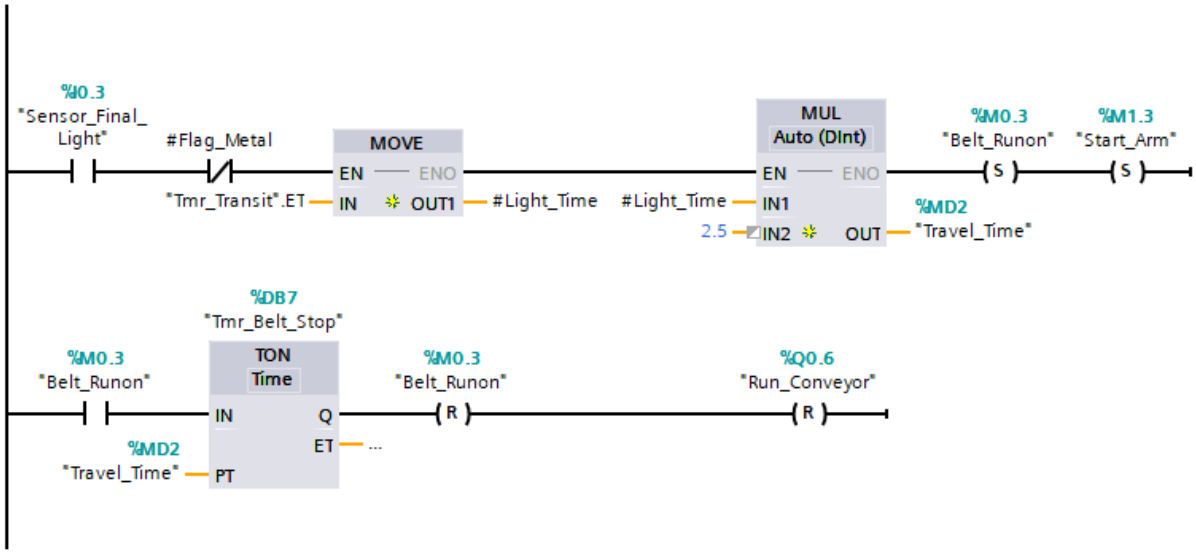
Network 5: SORT ALUMINIUM

As long as we have a metal counter that is not steel, the reject boom is operated. This will be from the inductive sensor pulse until the transit timer times out.



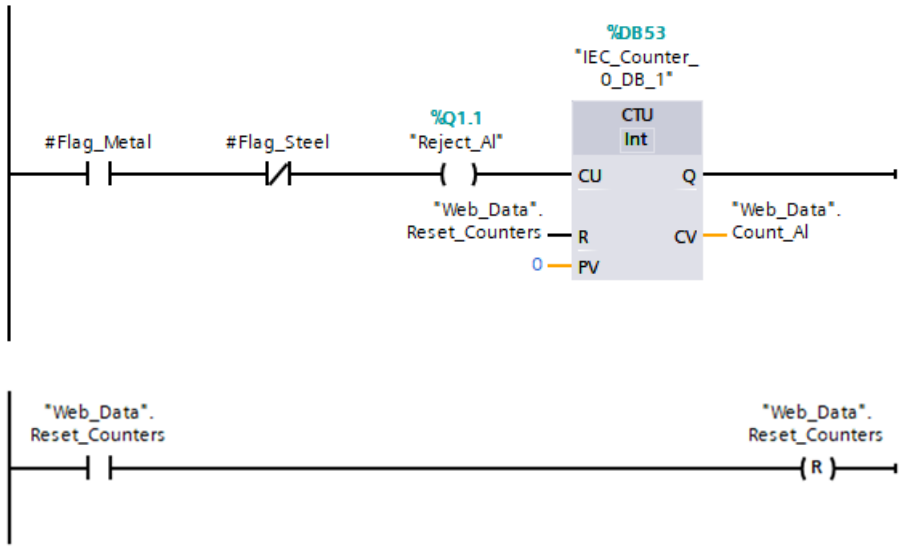
Network 6: TRIGGER ROBOT ARM

This network is only included in the “Smart Factory with Robot Demo” program. If the counter is not metal then the final light gate is used to get the conveyor speed. The conveyor is stopped after 2.5x the time between the two light gates and then the robot arm is triggered.



Network 7: RESET COUNTERS

This network is only included in the “Smart Factory with Robot Demo” program. Networks 4, 5 and 6 include CTU blocks that count up every time their CU input changes from false to true. The current values of these counters are written the count variables that are displayed on the web interface. The web interface incudes a flag that is used to reset the counters. This flag is read by each of the counters, triggering a reset and then it is itself reset in network 7 so that the counters are only reset once.



Picking Counters – Gantry and Plunger

With the sorting routine working, the program is extended to deliver counters to the conveyor for sorting. The following connections are added: -

Connection	Device	PLC Tag Name
Inputs Block: 4	Gantry home microswitch	%I0.4 Axis_Home_Position
Inputs Block: 5	Trigger pick from bin 1	%I0.5 Pick_Bin_1
Inputs Block: 6	Trigger pick from bin 2	%I0.6 Pick_Bin_2
Inputs Block: 7	Trigger pick from bin 3	%I0.7 Pick_Bin_7
Outputs: 0	Stepper motor pulse	%Q0.0 Stepper_Pulse
Outputs: 1	Stepper motor direction	%Q0.1 Stepper_Direction
Outputs: 3	Plunger down valve	%Q0.3 Pneu_Plunger_Down
Outputs: 4	Plunger up valve	%Q0.4 Pneu_Plunger_Up
Outputs: 5	Power vacuum generator	%Q0.5 Pneu_Vacuum
Outputs: 6	Power conveyor	%Q0.6 Run_Conveyor

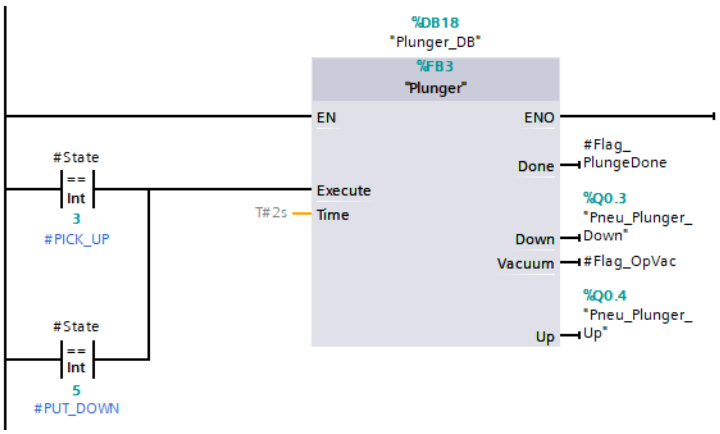
GANTRY PROGRAM

The Gantry program is added to the main cycle and calls the “Run_Gantry” function block. Along with the sorting program, this will be run to completion and then restarted.

The function is implemented as a state machine. The variable #State runs through a sequence with different parts of the program being run depending upon the sequence. The state is incremented when each stage is completed.

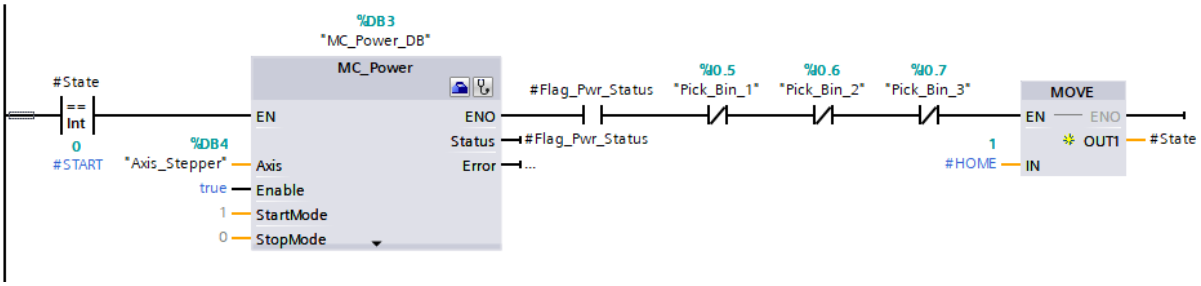
Network 1: Plunger

Two states can trigger the plunger function block; picking up from the bins or putting down on the conveyor. The plunger begins to operate as soon as either state is reached. The up and down valves are driven directly but the vacuum output is a flag. This flag is used to switch the vacuum on or off depending upon the state.



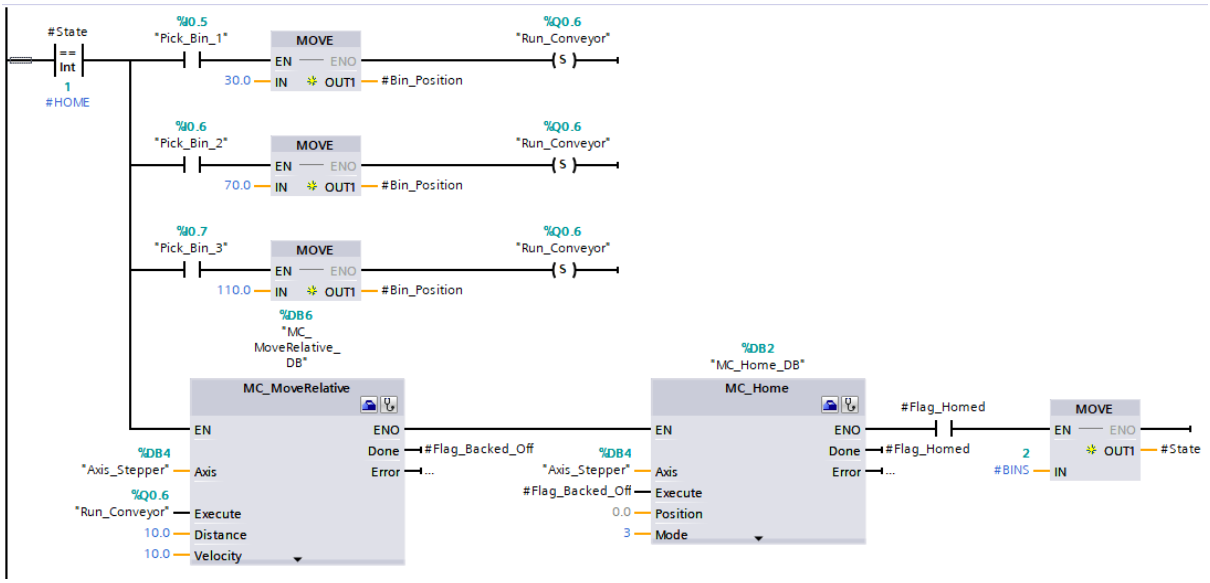
Network 2: Start

When the PLC starts up the MC_Power block is called to initialize the motion control object. Once the axis is ready and no pick inputs are energized then the state progresses to the #HOME state.



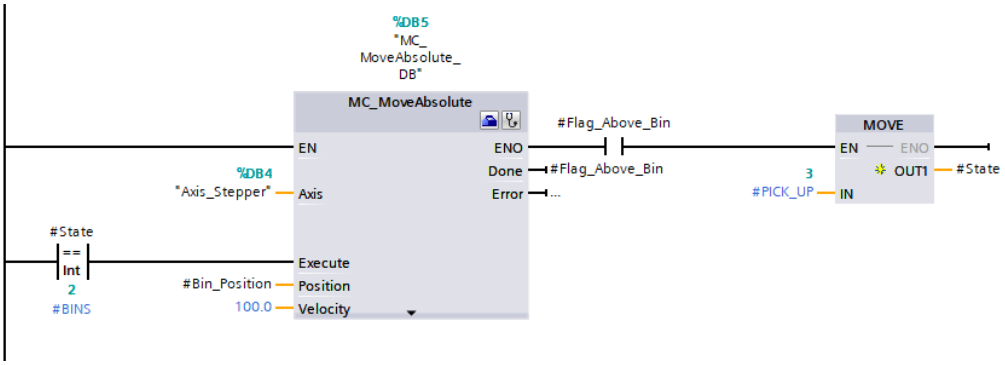
Network 3: Home

In this state the function waits for one of the pick inputs to be energized. A constant is copied to the #Bin_Position variable depending upon which input and the conveyor is started. Starting the conveyor triggers the stepper to back of by 10mm and then run its homing procedure. Once the homing procedure is complete, the state progresses to the #BINS state.



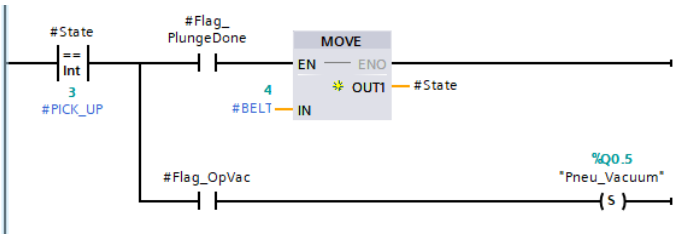
Network 4: Move to counter bins

In this state, the stepper motor is moved to position the plunger above one of the counter bins. The position at the end of this move depends on the bin that was chosen in the previous state.



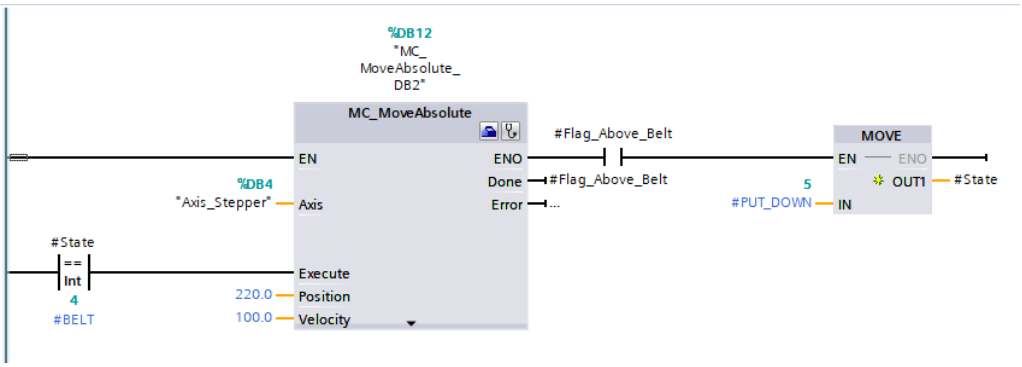
Network 5: Pick up counter

In this state the program waits for two signals from the plunger function block that was called in network 1. When the signal is given to operate the vacuum, the vacuum generator is switched on. When the cycle of the plunger is complete, the state progresses.



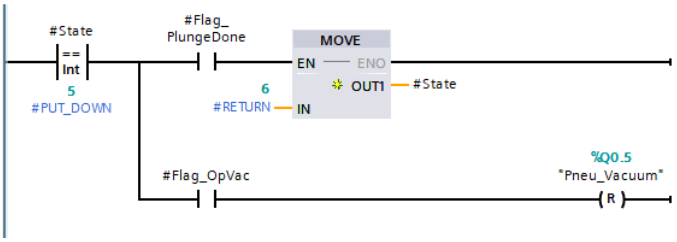
Network 6: Move to the belt

In this state the stepper moves to position the plunger above the conveyor belt.



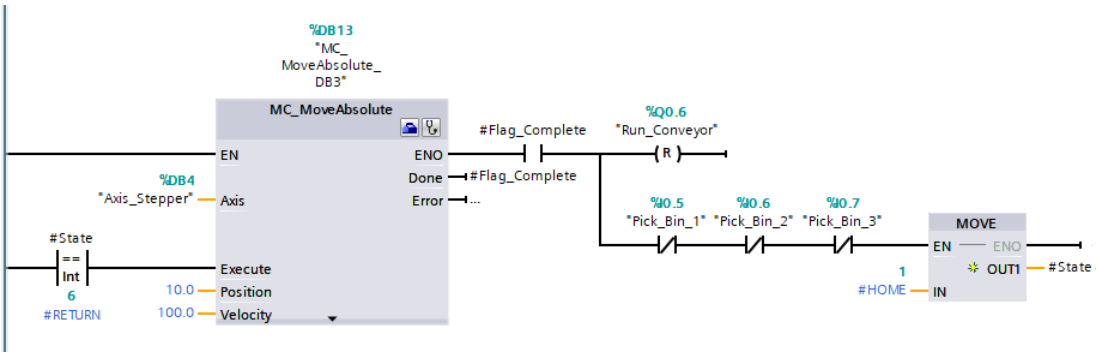
Network 7: Put down the counter

This state is very similar to network 5 where the counter was picked up. In this state however, the signal to operate the vacuum results in the vacuum generator being switched off and the counter dropped onto the conveyor.



Network 8: Returning to the park position

In this state, the stepper returns the gantry to its home position and stops the conveyor. The state becomes ready for the next cycle when all the pick inputs are off. This means that a new input signal is needed to trigger the next operation.

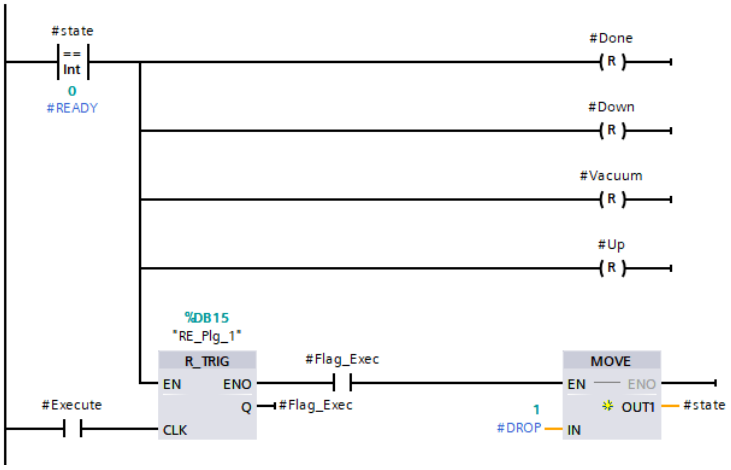


PLUNGER FUNCTION BLOCK

The plunger function block is implemented as a state machine and is called by the gantry program when it is in a position to operate the plunger.

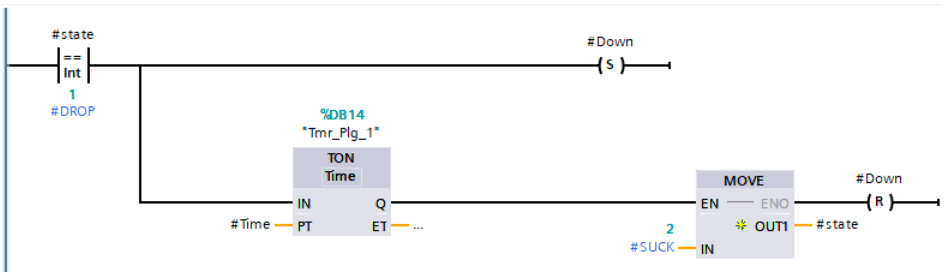
Network 1: Ready

In this state the function switches off all the outputs and waits for the rising edge of the execute input.



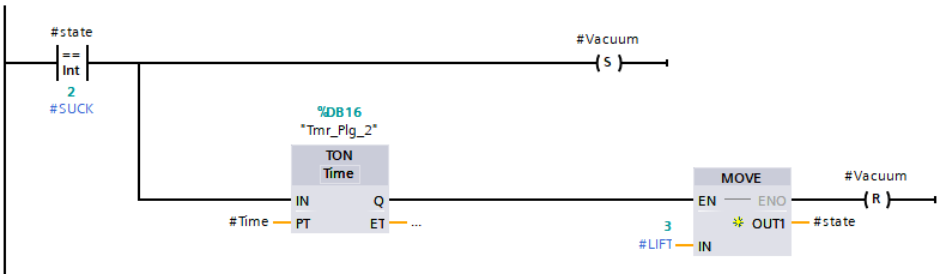
Network 2: Driving down

The valve is set to drive the plunger down and a timer is started. When the timer completes, the valve is switched off and the state progresses.



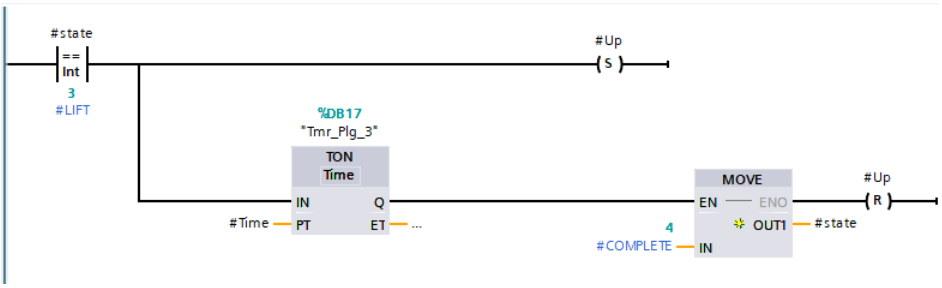
Network 3: Operating the Vacuum

In this state, the vacuum flag is signaled until a timer completes and the state progresses. The gantry program that calls this function will use the vacuum signal to turn the vacuum on or off depending upon where in the cycle it is.



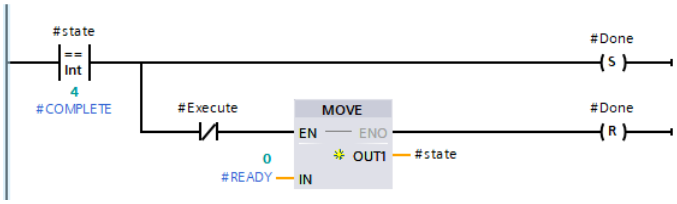
Network 4: Driving up

In this state the plunger is driven up until a timer completes.



Network 5: Complete

In this state a flag is set to signal to the calling program that the sequence is complete. When the execute signal is removed, the state resets to the beginning.

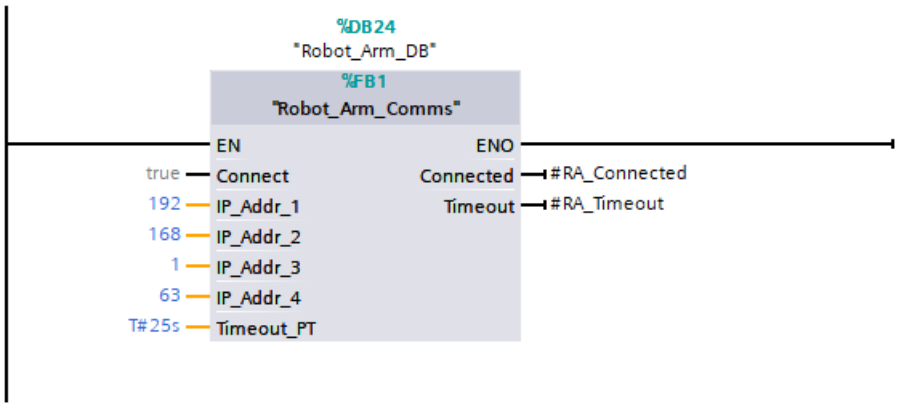


Control of the Robot Arm

Two networks are included in the Robot program. The first handles the communication to the robot arm and the second contains the entire sequence of picking up, checking and sorting a counter.

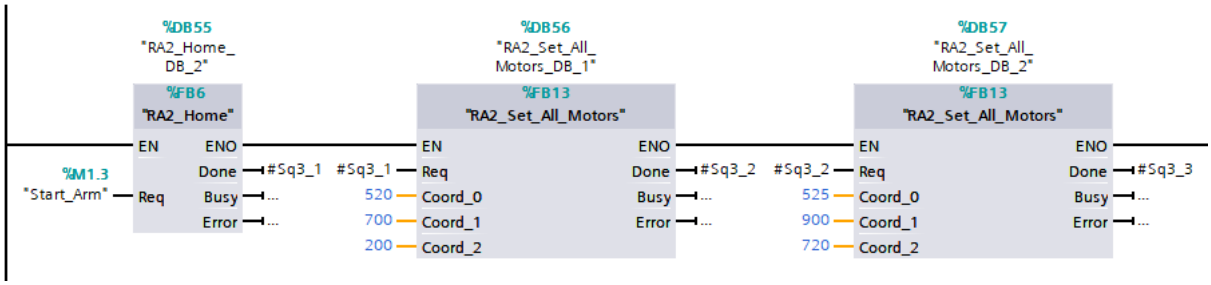
Network 1 COMMS

This network contains the block that handles communication. It must be run at the beginning of each program cycle. The inputs and outputs are described in section 3.

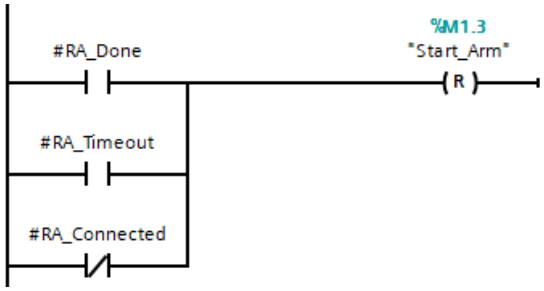


Network 2 Complete pick, check colour and sort routine

Each of the blocks is triggered by a transition from false to true on its Req input and sets its Done output to true when the operation is completed. This means that blocks can be chained together with the Done output of one operation triggering the Req input of the next.

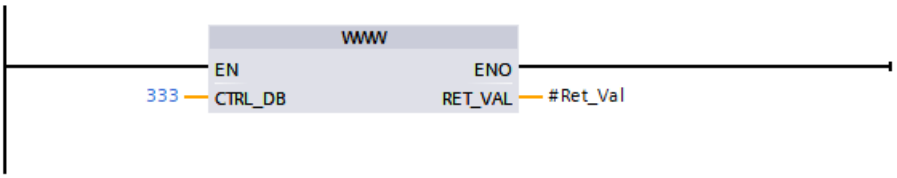


The whole sequence when the Sorting program detects a plastic counter and sets the “Start_Arm” flag. This flag is reset ant the end of the sequence. The Run_Gantry function block uses this flag to hold off dropping a new counter until the robot arm has finished its operation. If the robot arm is not connected or times out due to an error then this flag must also be reset so that the gantry is not left hanging.

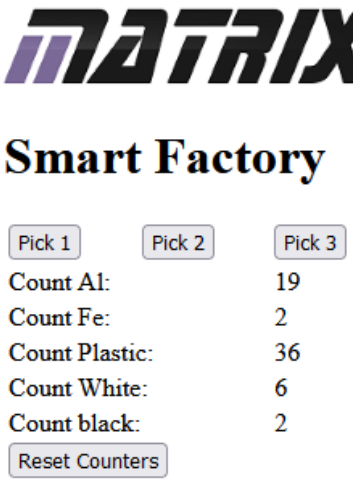


Web Server Interface

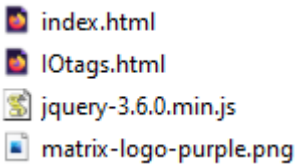
The “Smart Factory with Robot Demo” program includes a web server interface. The Web_Server program simply calls the WWW function block.



This block handles serving the user web pages and the live variables. The variables that are referenced by the web pages are all included in the “Web_Data” data block. This is done for convenience, the web pages could access any global variables in the PLC.
The web page includes buttons to trigger the gantry to pick from one of the three hoppers, counts of each type of counter sorted and a button to reset the counters.



The Web Files directory includes four files that are required for the web interface. These are the jQuery library, a logo and two html pages.



IOtags.html

This file contains declarations of all the variables that can be written and a JSON structure containing all the variables in the web interface. The names of the variables are replaced by the current values when the page is served.

```
<!-- AWP_In_Variable Name='"Web_Data".Trigger_1'-->
<!-- AWP_In_Variable Name='"Web_Data".Trigger_2'-->
<!-- AWP_In_Variable Name='"Web_Data".Trigger_3'-->
<!-- AWP_In_Variable Name='"Web_Data".Reset_Counters'-->
<!-- AWP_In_Variable Name='"Web_Data".Robot_Present'-->
{
"tg1":":="Web_Data".Trigger_1:",
"tg2":":="Web_Data".Trigger_2:",
"tg3":":="Web_Data".Trigger_3:",
"rct":":="Web_Data".Reset_Counters:",
"rap":":="Web_Data".Robot_Present:",
"cal":":="Web_Data".Count_Al:",
"cfe":":="Web_Data".Count_Fe:",
"cps":":="Web_Data".Count_Pls:",
"cbk":":="Web_Data".Count_Bk:",
"cwt":":="Web_Data".Count_Wt:",
"sts":":="Web_Data".Status_Str:"
}
```

Index.html

This file contains the html code to display the web page and the JavaScript code to continually load IOtags.html and use its data to update the live variables displayed in the web page. It contains JavaScript functions for each of the buttons that posts new values for selected variables back to the PLC.

```
<!-- AWP_In_Variable Name='"Web_Data".Trigger_1'-->
<!-- AWP_In_Variable Name='"Web_Data".Trigger_2'-->
<!-- AWP_In_Variable Name='"Web_Data".Trigger_3'-->
<!-- AWP_In_Variable Name='"Web_Data".Reset_Counters'-->
<!-- AWP_In_Variable Name='"Web_Data".Robot_Present'-->
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Matrix Smart Factory</title>
    <script src="jquery-3.6.0.min.js"></script>
    </head>
    <body>
      <img src="matrix-logo-purple.png">
      <h1>Smart Factory</h1>
      <table style="width:250px">
      <tr><td><button id='pk1'>Pick 1</button></td>
          <td><button id='pk2'>Pick 2</button></td>
          <td><button id='pk3'>Pick 3</button></td>
```

```
    </tr>
    <tr><td colspan="2">Count Al:</td><td id='Ct_Al'></td></tr>
    <tr><td colspan="2">Count Fe:</td><td id='Ct_Fe'></td></tr>
    <tr><td colspan="2">Count Plastic:</td><td id='Ct_Pls'></td></tr>
    <tr><td colspan="2">Count Black:</td><td id='Ct_Bk'></td></tr>
    <tr><td colspan="2">Count white:</td><td id='Ct_Wt'></td></tr>
    <tr><td colspan="3"><button id='rst_cnt'>Reset Counters</button></td></tr>
    </table>

</body>

<script type="text/javascript">
    $(document).ready(function(){
        $.ajaxSetup({ cache: false });
       setInterval(function() {
        $.getJSON("IOtags.html", function(result){
            $('#Ct_Al').text(result.cal);
            $('#Ct_Fe').text(result.cfe);
            $('#Ct_Pls').text(result.cps);
            $('#Ct_Bk').text(result.cbk);
            $('#Ct_Wt').text(result.cwt);
        });
       },1000);
       // Operate the buttons
       $('#rst_cnt').click(function(){
       url="IOtags.html";
       name='"Web_Data".Reset_Counters';
       val=1;
       sdata=escape(name)+'='+val;
       $.post(url,sdata,function(result){});
       });
       $('#pk1').click(function(){
       url="IOtags.html";
       name='"Web_Data".Trigger_1';
       val=1;
       sdata=escape(name)+'='+val;
       $.post(url,sdata,function(result){});
       });
       $('#pk2').click(function(){
       url="IOtags.html";
       name='"Web_Data".Trigger_2';
       val=1;
       sdata=escape(name)+'='+val;
       $.post(url,sdata,function(result){});
       });
       $('#pk3').click(function(){
       url="IOtags.html";
       name='"Web_Data".Trigger_3';
       val=1;
       sdata=escape(name)+'='+val;
       $.post(url,sdata,function(result){});
       });
    });
</script>

</html>
```