Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DISEÑO DE UNA RED LOCAL CON PLCs ALLEN BRADLEY Y HMIs PANELVIEW PARA ESTABLECER LA COMUNICACIÓN ENTRE DISPOSITIVOS

Alumno/Alumna	*Ruiz, Martinez, Mikel*
Director/Directora	*Pinto, Charles*
Departamento	**Relaciones Internacionales**
Curso académico	*2018/2019*

Bilbao, 29, Octubre, 2019

Capstone Project

Final Report

Team Members:

Mikel Ruiz

Austin Schroder

Vivek Kashyap

Jonan Gomez

Miranj Kansara

Project Advisor:

Dr. Max Rabiee

Additional Advisors:

Dr. Carla Purdy

Nathan Huber

April 15, 2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

Introduction

During the last decades a lot of industries have transitioned into a more automated way of completing their duties. Some of the reasons that can explain this change are the benefits that this practice has brought into industry: cost reduction, productivity, availability, reliability and performance.

Since people became aware of the benefits, they have found the way to automate more and more processes in industry that were previously done manually. This has made the automation device providers to also upgrade their products, improving their computing capacity, the amount of inputs and outputs they can handle and the way these devices can be connected to others to create sophisticated networks that can automate a whole factory.

With developments in industrial automation comes the need to protect PLC (Programmable Logic Controller) systems from hacking attacks. Since all devices in a plant are typically connected on an ethernet network, they can be vulnerable from attacks if the network is not secured. Malware developers usually target the SCADA (Supervisory Control and Data Acquisition) system as a point of entry. This problem with cybersecurity must be handled in modern systems. This project can act as a testbed for the protection of PLC and HMI (Human Machine Interface) systems from hacking attacks. This has been examined in limited scope, so further developments will be necessary in future projects.

In this report some of the latest technology released by Rockwell Automation, a well known automation product distributor, will be examined, proved and used to create a network that could be similar to one you could find in industry. This network is a joint effort. One half of the network, consisting of 1 PLC and 1 HMI, is developed by this team, referred to as “Team 1” in the paper. The other half also consists of 1 PLC and 1 HMI, and is developed by the other team, referred to as “Team 2” in the paper. These systems are set up identically, and experiments with Master/Slave networking are carried out, utilizing both halves of the system. Both teams share the network switch and the main workstation to accomplish the end goal.

Used Equipment

In the following lines the different software and hardware used to carry out the setup will be listed with a brief description of them.

Hardware

- 2 units of Allen-Bradley ControlLogix 1756-L81ES PLC: This controller is powered by its power supply, which takes 120VAC. The chassis has 7 slots, though only slots 0-4 are used. Slot 0 is the processor. Slot 1 is the Logix 55L8SP safety processor. Slot 2 is the 16-point DC input. Slot 3 is the 36-point analog input. Slot 4 is the 16-point DC output. Slots 5 and 6 are blank. The PLC is shown in Figure 1.

Figure 1: Allen-Bradley ControlLogix 1756-L81ES PLC

- 2 units of PanelView™ 5310 terminals: Human Machine Interfaces able to monitor and control devices attached to PLCs. The HMI is shown in Figure 2.

Figure 2: PanelView™ 5310

- Linksys Etherfast 10/100 8-port switch: Indispensable devices to create a network with multiple devices. It is shown in Figure 3.

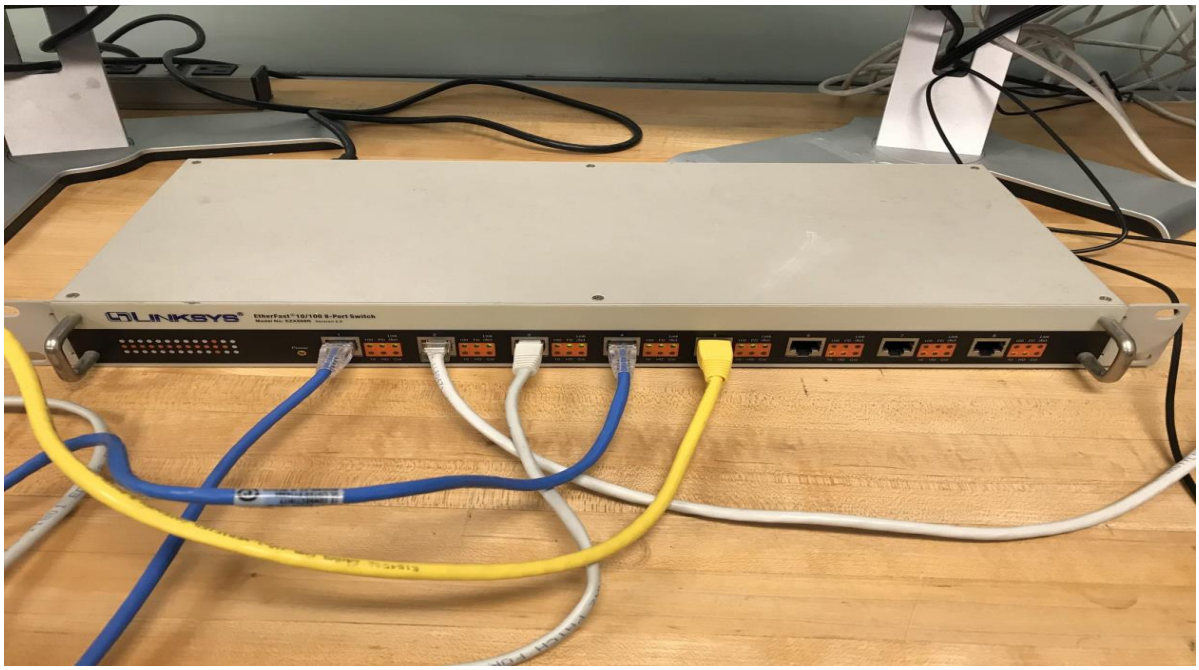


Figure 3: Linksys Etherfast 10/100 8-port switch

- Ethernet cables: 5 ethernet cable are used, one for connecting each of 5 devices to the network switch. The devices are the 2 PLCs, 2 HMIs, and programming workstation.

Software

- RSLinx: The software being used to enable communication between all Allen-Bradley devices. Ethernet and serial drivers are configured here, and all network addresses are listed in tree format, with associated IP addresses.

- Studio 5000: This software has integrated into the same framework the design of HMI applications and the programming of PLCs, whereas before two different softwares needed to be purchased (RSLogix and FactoryTalk View).

    - Studio 5000 Logix Designer: This software allows the user to configure and program the ladder logic diagram in the Allen-Bradley Logix 5000 family.

    - Studio 5000 View Designer: This software allows the user design environments for the PanelView 5000 family operator terminals.

3. Part I: Configuring a single PLC with the computer

In this part the steps to make the Allen-Bradley PLC communicate with the PC will be explained, as well as the problems that were encountered during the process. After this part

the user should be able to download his programs into the PLC and make them run in a successful way.

There are 2 possible connections from the PLC to the computer, using the USB or the Ethernet port. As the intention is to create a network, the communications will be carried out by the Ethernet port, however, the communication using the USB port was also tested in order to check its functionality. These are the steps that were taken to carry out the connection:

1) Connection was made via ethernet cable into the port on slot 0.
2) The driver for Ethernet/IP was created in RSLinx, under *communications* > *configure drivers* > *add new*. Ethernet/IP driver was selected and added. Browsing the local subnet confirms that the driver is running and available.
3) In order to connect to the Ethernet/IP network, the static IP address of the PLC module was set as 192.168.1.81.
4) After connecting via Ethernet/IP, the controller firmware was updated, which allowed the USB driver to be configured. Communication is possible through Ethernet/IP and USB. The RSLinx communication tree is shown in Figure 4.

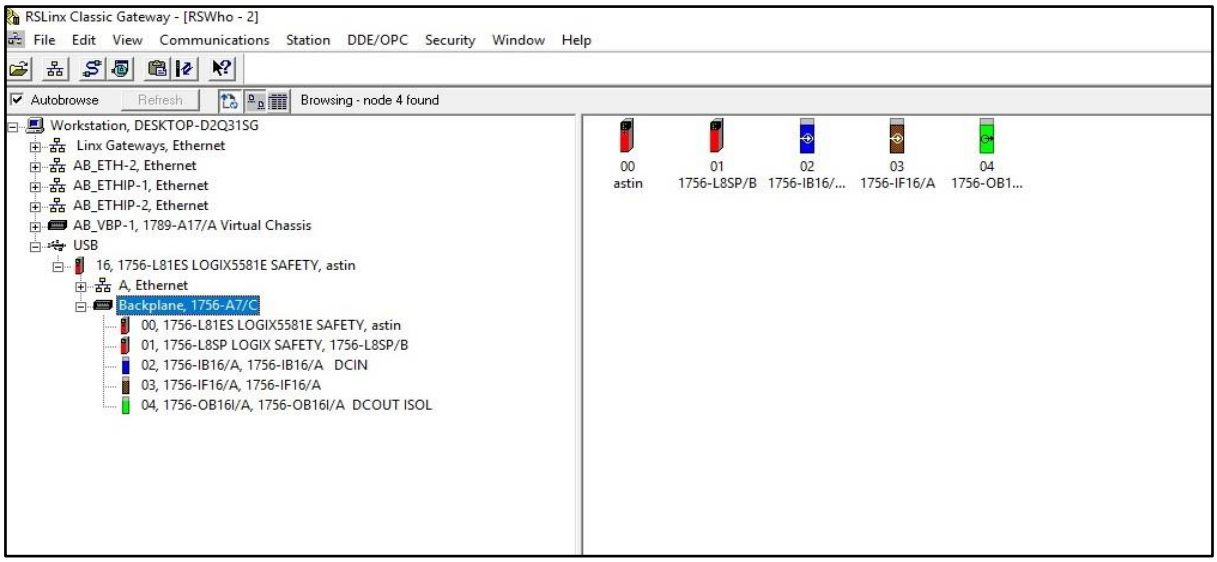


Figure 4: RSLinx displaying the PLC modules

Creating Project File

After communication was established, the project file was created in Studio 5000 Logix Designer. The revision, chassis, and slot properties of the controller were defined. In Studio 5000, RSWho was selected, and the communication path set in RSLinx was found, allowing the ability to go online with the processor through the established path. After this, the hardware was configured in the project tree defining processor and I/O's in the project area. The Who Active Dialog box is shown in Figure 5.

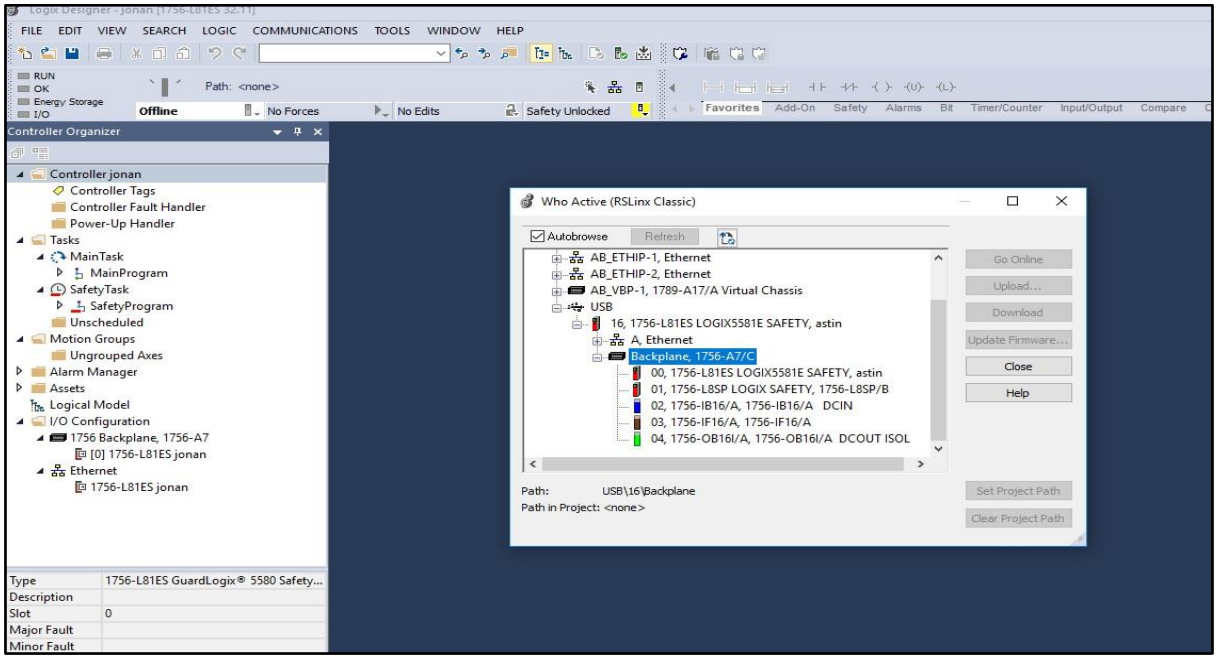


Figure 5: Selecting path to processor in Studio 5000

Logic design

The Studio 5000 Logix Designer is used to design the logic that will be implemented on the PLC. In the Controller Organizer panel, under *Tasks > Main Tasks > Main Program > Main Routine* is where the main routine will be programmed. In an industrial control the main routine will call a lot of subroutines, which will make it possible to accomplish the control. However, a simple logic will be programmed this time, just to download and test if the communication with the PLC has been established succssesfully. This logic is shown in Figure 6.

Figure 6: Test routine

Creating the tags

The tags will be the tool used to connect the I/O (inputs/outputs) in the software with the ones in the hardware (the PLC) and to create variables that can store data inside the program. Once the different I/O have been set, the intention is to match that I/O with ones in the PLC. This will be done by right clicking on the question mark and selecting New Tag… (*?>New Tag…*). A New Parameter Tag window will be opened, which will look like the one shown in Figure 7.

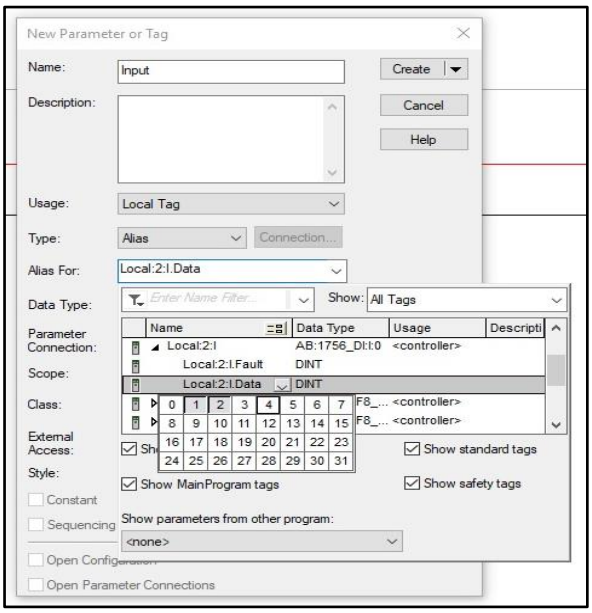


Figure 7: Creating new tag

In this New Parameter Tag window the name of the tag will be defined, which should be unique for each tag. There are quite a few options, and the user could choose between base, consumed, produced or alias tags.

In this simple test program some alias tags have been created, which will stand for the real outputs in the device. In the *Alias for* section the port connected to the different I/O will be chosen, as it is shown in Figure 8.

Scope: MainProgram Show: All Tags Enter Name Filter...

Name	Usage	Alias For	Base Tag	Data Type	Class	Description	External Access	Constant	Style
Output1	Local	Local:4:O.Data.2(C)	Local:4:O.Data.2(C)	BOOL	Standard		Read/Write	☐	Decimal
Output	Local	Local:4:O.Data.1(C)	Local:4:O.Data.1(C)	BOOL	Standard		Read/Write	☐	Decimal
Input1	Local	Local:2:I.Data.1(C)	Local:2:I.Data.1(C)	BOOL	Standard		Read/Write	☐	Decimal
Input	Local	Local:2:I.Data.2(C)	Local:2:I.Data.2(C)	BOOL	Standard		Read/Write	☐	Decimal
								☐	

Figure 8: Tags in test routine

The user can view all tags in the project by going to the *Controller Tags* section on the project explorer.

Downloading the logic to the PLC

After the program has been created together with the corresponding tags and the path has been selected, it is time to download the program into the PLC. It is important to have the PLC

in program mode when you try to download something into it. However, at this point we encountered an error concerning compatibility issues between software in PC and firmware in PLC.

Firmware Revisions

One of the issues encountered when trying to set the communication channel between the Studio 5000 software and the PLC was the incompatibility between versions. The software was on its latest version (32.0), and the firmware was in the 31.011 version.
Rockwell Automation website offers a tool to compare the compatibility between different software and firmware, and the option to download the firmware you want to install on the PLC. This tool is shown in Table 1.

Table 1: Checking the compatibility

	1756-L81ES	RSLogix 5000	Studio 5000 Logix Designer	1756-L81ES
Series	series B			series B
Version	31.011	20.05.00	32.00.00	32.011
Downloads				
Information				
Compatibility				
1756-L81ES B 31.011	✓	○	✗	○
RSLogix 5000 20.05.00	○	✓	✓	○
Studio 5000 Logix Designer 32.00.00	✗	✓	✓	✓
1756-L81ES B 32.011	○	○	✓	✓

After identifying the problem the firmware was updated to the 32.011 version, which is compatible with the latest software version, as it is shown in Table 1. Once the update was made, the incompatibility problems were solved.

Download and Force Function

Once the program has successfully been downloaded to the device, it's time to check if the logic works. Since the PLC is not connected to any physical I/O, the force function is used to turn inputs on.
The program was downloaded to controller in program mode, as it has been said before, and then was placed in Run Mode. The option to enable all forces in the Studio 5000 was selected. Then the individual input was forced on, causing the XIC (Examine if Closed) and OTE (Output Energize) instructions to become true. The input and output indicators on the PLC cards confirm that the logic is working as intended. The ladder rung with the forces enabled is shown in Figure 9. The input and output indicators on the PLC are shown in Figure 10.

Figure 9: Test routine Online

Figure 10: PLC showing output from test routine

4. Part II: Dishwasher program and PanelView application Overview

To try out this new equipment a new program more complicated than the previous one will be created. This program will intend to emulate the way an industrial dishwasher is controlled, powering on or off its different actuators depending on the washing cycle this machines have usually preprogrammed.

The way this control has been designed and the steps taken are explained below.

Dishwasher ladder logic

The following steps are the ones a dishwasher machine is supposed to follow to perform its main task:

1. Energize the soap solenoid for 4 seconds: Green_PLT ON.
2. Open the input valve for hot water for 5 seconds: White_PLT ON.
3. Operate the washer impeller for 12 seconds: M1 ON.
4. Open the rainwater valve for 1 second: Ring Bell ON.
5. Turn on the drain pump for 3 seconds: M2 ON.
6. Turn on the heater for 6 seconds: Red_PLT ON.

A machine that performs this task can be modelled as a state machine, where the outputs will be defined by the current state of the machine. In Table 2. the different states and the outputs value for each of them are shown.

Table 2: State machine for dishwasher

Step Number	Time in step	Motor 1	Motor 2	Green_PLT	Red_PLT	White_PLT	Bell
1	4	off	off	ON	off	off	off
2	5	off	off	off	off	ON	off
3	12	ON	off	off	off	off	off
4	1	off	off	off	off	off	ON
5	3	off	ON	off	off	off	off
6	6	off	off	off	ON	off	off

Defining the I/O

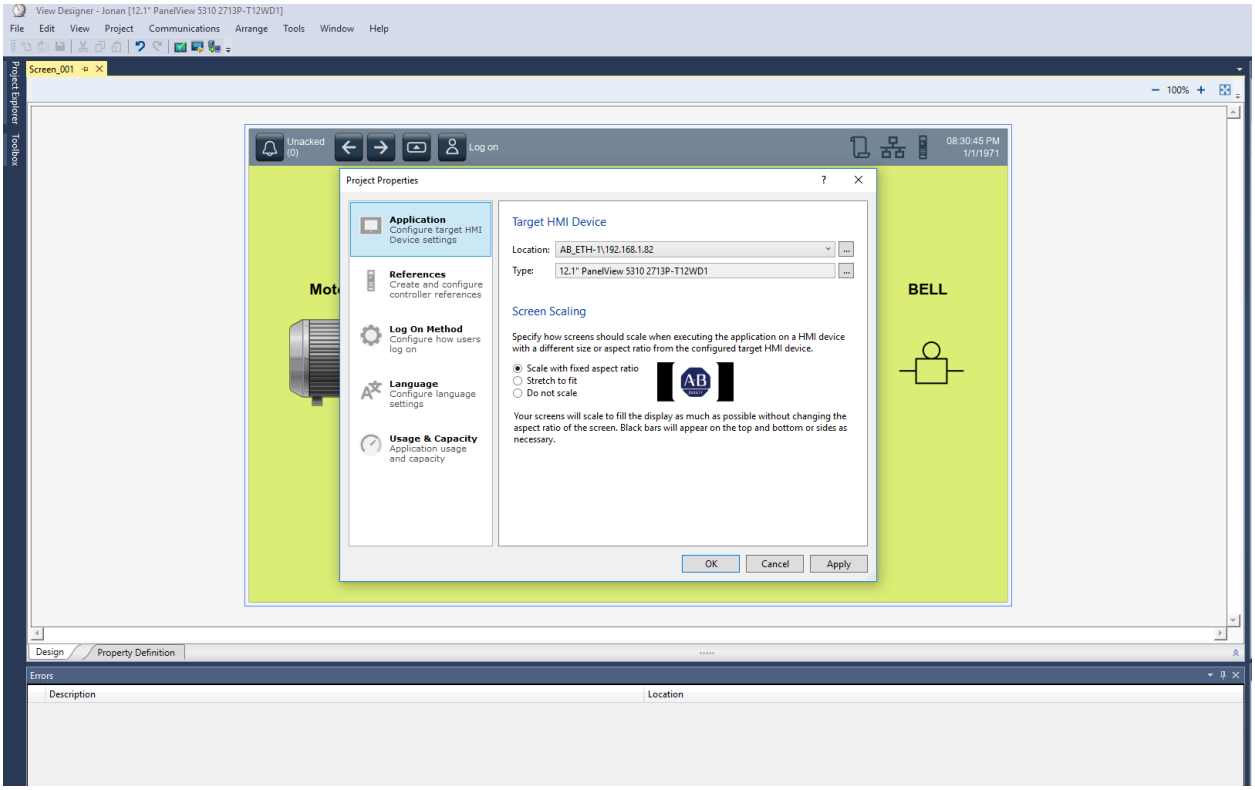
First of all, it is very important before designing the ladder logic or any other program to know how many inputs and outputs the program will need.

The number of outputs is clear from Table 2. The control system will have 6 elements to control, therefore, the program will need to have 6 outputs. To decide the number of inputs we assumed that there will be a start push button that gives the machine a signal to start. As well as a master reset, which is a security button that all machines have in case you want to stop it immediately.

In the program the I/O will be linked to the following variables from the Studio 5000, shown in Table 3.

Table 3: Linking I/O to variables or outputs

I/O	Tags (TYPE)
Start Push Button	Start_Bit (BOOL)
Master Reset	Reset_Bit (BOOL)
M1	Local:4:O:Data.0
M2	Local:4:O:Data.1
Green_PLT	Local:4:O:Data.2
Red_PLT	Local:4:O:Data.3
White_PLT	Local:4:O:Data.4
Bell	Local:4:O:Data.5



Ladder Logic

The main functions used for this logic was the sequencer output function (SQO) and the timer function. The purpose of this function in this program will be explained below:

-Non retentive Timer on Delay

The timer function will de-energize and energize the rung containing the sequencers each time the *Accum* reaches the *Preset* value. This action will make the sequencer change the value of *Position*, making the state machine go to the next step.

As the time step for each step is variable the *Preset* won't be defined, this value will come straight from *Time_Sequencer_Array*, which will be explained later on. TON block is shown in Figure 11.

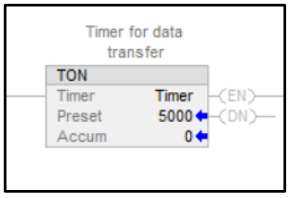


Figure 11. TON in Studio 5000

-Sequencer (SQO)

The main part of this ladder logic diagram is the sequencer output function. The sequencer function will make it possible to change the states for one to the next one and to change the waiting time for each of the states. SQO block is shown in Figure 12.

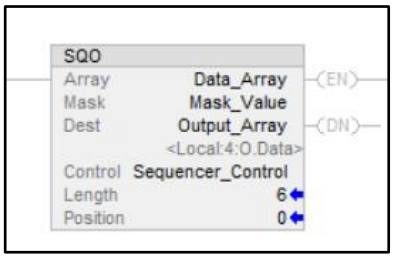


Figure 12: SQO in Studio 5000

But, how does the sequencer work? Each time the sequencer energizes it passes a number (32 bit binary number in Studio 5000) from *Data_Array[Position]* to *Output_Array*, masking it through *Mask*. To store different information that makes this function possible, a control variable has to be set in *Control*. Finally, the length of the array has also need to be defined in the SQO, this length is used to reset the *Position* once it has reached the value of *Length*.

When the step time varies from one to another 2 sequencers will need to be used, one to pass the state values (the outputs), and other one to pass the time to the timer.

-Output sequencer

The output sequencer is the one is shown in Figure 13. *Data_Array* will be an array which saves all the different states, and the process will be done as it is shown in Figure 13.

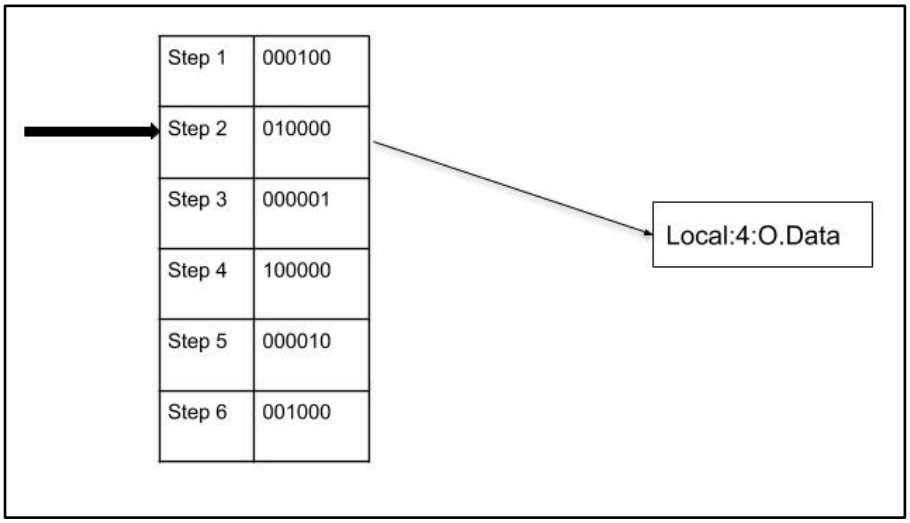


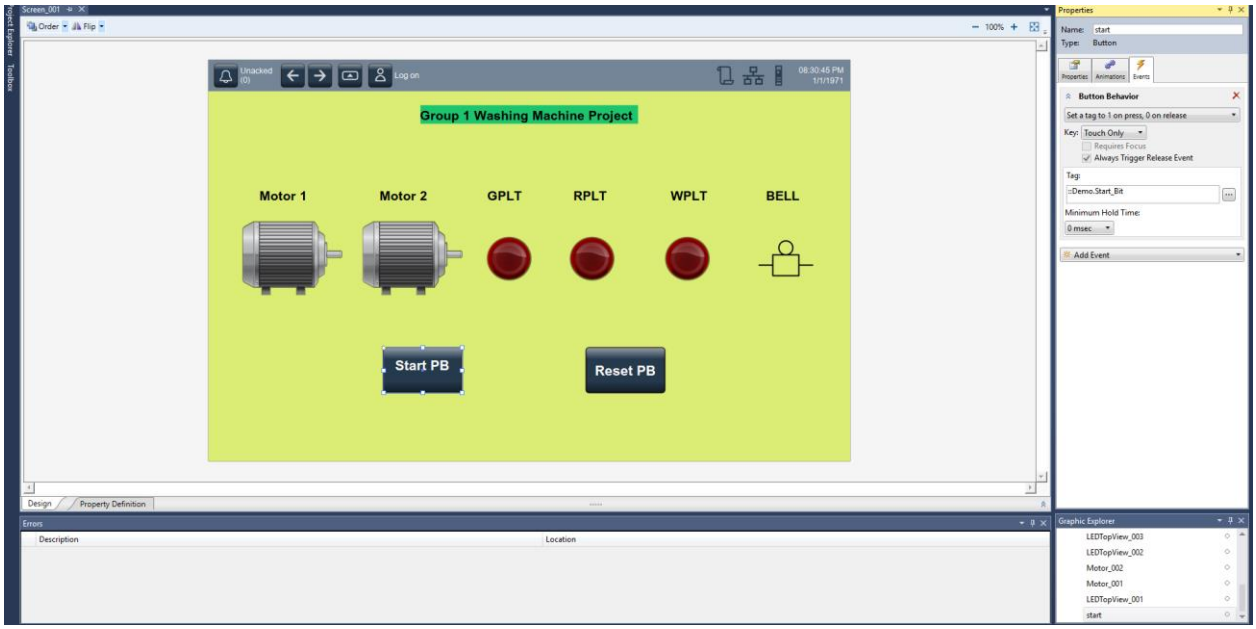
Figure 13: Output sequencer scheme

-Time sequencer

The time sequencer will work in the same way as the output sequencer, however, in this sequencer the time for each step will be stored in each position of the array. And instead of sending that data to an output it will be sent to the timer's preset (*Timer.PRE*).

Main Program

Figure 14 shows the ladder logic that holds the main program.
When the *Start_Bit* goes high it activates the timer and the sequencer, defining the output state and the time the program will stay in the position.



Once the *Accum* reaches the *Preset* value, *Timer.TT* will go low and *Timer.DN* will go high. These both events will make the timer and the sequencer rungs de-energize and energize again, making the current state and the *Timer.PRE* change.
In the diagram the reset rungs are also shown. These 2 rungs will reset the *Control* variables of the sequencers and the timer, as well as setting all the outputs to 0 using the MOV function, which passes a null value (000000) to the output tags.

Figure 14: Main routine for dishwasher

Tags

The tags for this program are shown in Table 4. The *Data_Array* is used to hold the states of each step of the machine. 7 elements of this array are used, one for each state, and also one for a null state. In each array element, 6 bits are used, to specify which outputs are turned on and off for each state. This data is used by the SQO function, passing each state to *Output_Array* (alias for *Local:4:O.Data*) at each timestep.

In a similar way, *Timer_Sequencer_Array* is used to hold the time the machine should stay at each step by passing timer values to *Timer.PRE*. In this case also 7 elements of the array will be used to store the 6 values and the blank element. What we are passing now are decimal numbers written in memory, so we cannot really now the number of bits that each of the number will be passing. To avoid problems when passing this numbers to the timer's preset, *Mask_Value* can be turned to an all ones 32 bit number.

Input is provided to this program in two ways. The first way is hard input from the input card *2:I.* The program was initially run by forcing the input *Start_PB* on. However, hard input tags cannot be used with HMI, since those are taken directly from the input card. In order to fix this two BOOL tags *Start_Bit* and *Reset_Bit* are defined and take input from the HMI.

Table 4: Tags in dishwasher program

Name	Value	Force Mask	Style	Data Type
Allow_control_PB	0		Decimal	BOOL
Data_Array	{...}	{...}	Decimal	DINT[10]
Destination	0		Decimal	DINT
Latch_Bit	0		Decimal	BOOL
Local:2:C	{...}	{...}		AB:1756_DI:C:1
Local:2:I	{...}	{...}		AB:1756_DI:I:0
Local:3:C	{...}	{...}		AB:1756_IF8_Float:C:0
Local:3:I	{...}	{...}		AB:1756_IF8_Float:I:0
Local:4:C	{...}	{...}		AB:1756_DO:C:0
Local:4:I	{...}	{...}		AB:1756_DO:I:0
Local:4:O	{...}	{...}		AB:1756_DO:O:0
M1_analog	0		Decimal	INT
M1_FROM1_TO2	0		Decimal	DINT
M1_speed	0		Decimal	DINT
M1_speed_slave	0		Decimal	DINT
M2_analog	0		Decimal	INT
M2_FROM1_TO2	0		Decimal	DINT
M2_speed	0		Decimal	DINT
M2_speed_slave	0		Decimal	DINT
Mask_Value	1048575		Decimal	DINT
Master_Reset	0		Decimal	BOOL
One_shot_1	0		Decimal	BOOL
One_shot_2	0		Decimal	BOOL
one_to_two_master	0		Decimal	DINT
Output_Array	0		Decimal	DINT
Reset_Bit	0		Decimal	BOOL
Sequencer_Control	{...}	{...}		CONTROL
Start_Bit	0		Decimal	BOOL
Start_PB	0		Decimal	BOOL
Time_En	0		Decimal	BOOL
Timer	{...}	{...}		TIMER
Timer_Sequencer_Array	{...}	{...}	Decimal	DINT[7]
two_to_one_slave	0		Decimal	DINT

Creating the HMI application

One of the good things of Studio 5000, as it was mentioned before, was the integration of Logix Designer and the View Designer in the same program.

To start creating the application the user needs to create a new project in the View Designer section and choose the hardware, the size of the screen and the PanelView number, in which the project will be downloaded. This screen is shown in Figure 15.

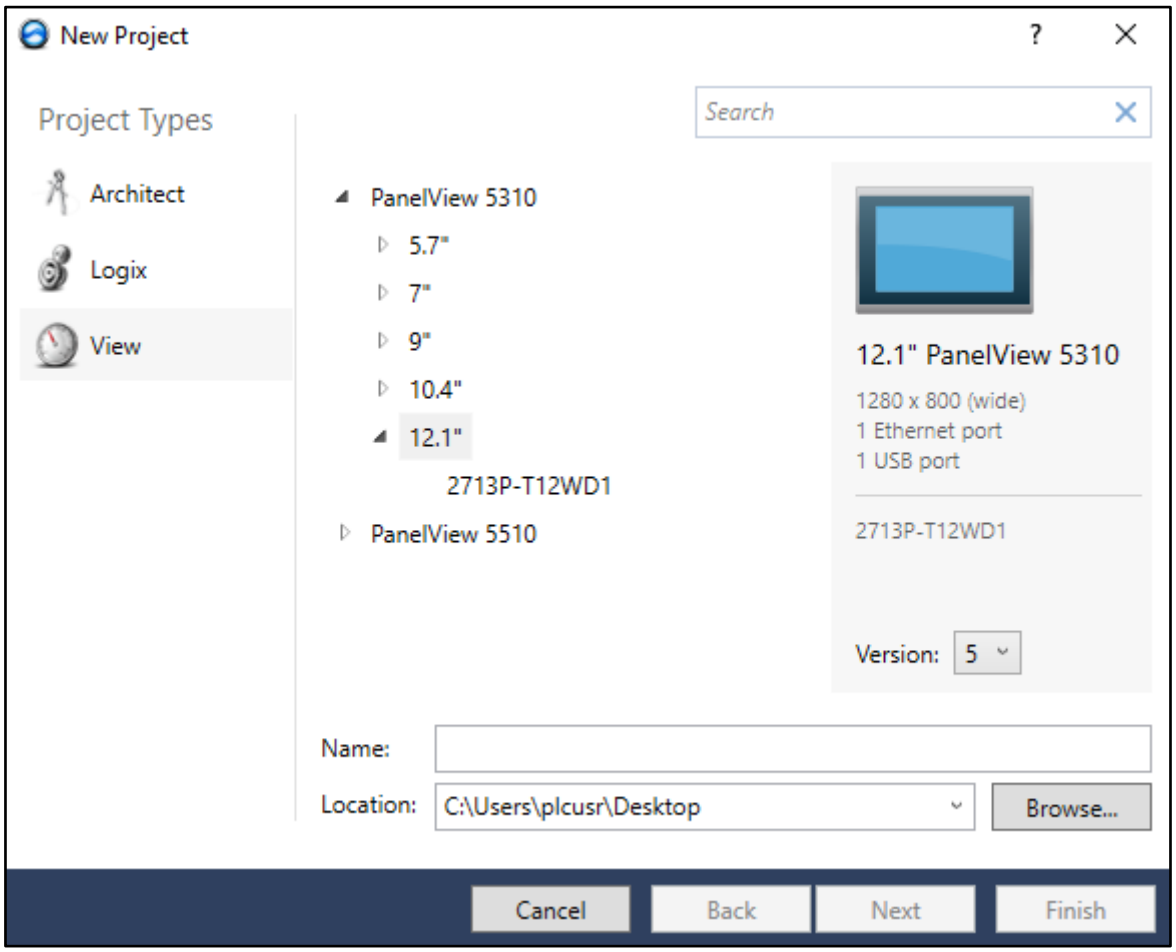


Figure 15: Starting a View design program

Before stepping ahead to start your application the different project properties will be defined in the *Communication>Project Properties* section. In the *References* section this important information needs to be declared to the program, a screenshot of this window is shown in Figure 16.

The information to be listed is:

a. Controller Reference Name
b. ACD project file path to be given from the PLC in order for data to be read by HMI. In this case, the path where the previously designed program is located will be written.
c. PLC IP address to be given with processor slot number.

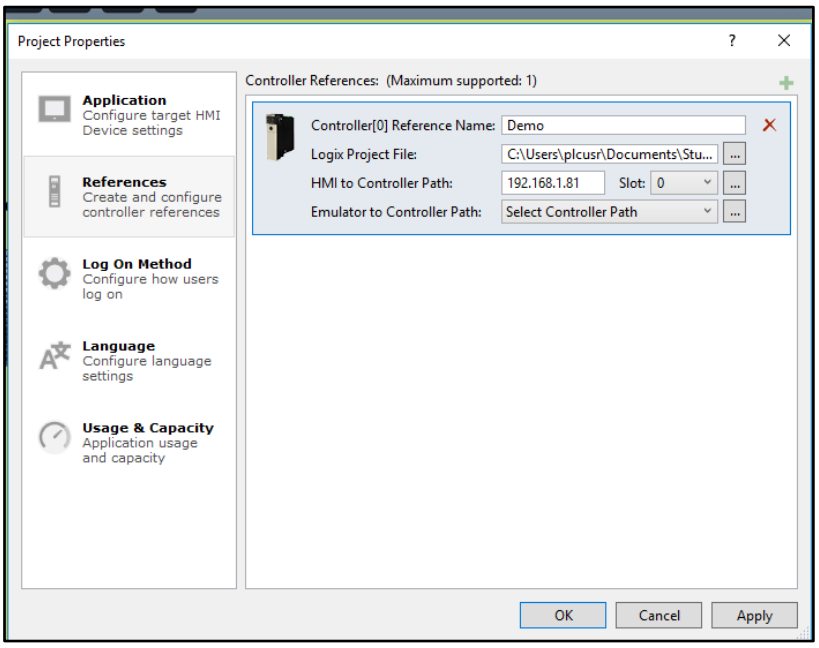


Figure 16: Selecting references for the View design program

In the same *Project Properties* window, in the *Application* section, there is also important information that needs to be input into the program. A screenshot of this window is shown in Figure 17.

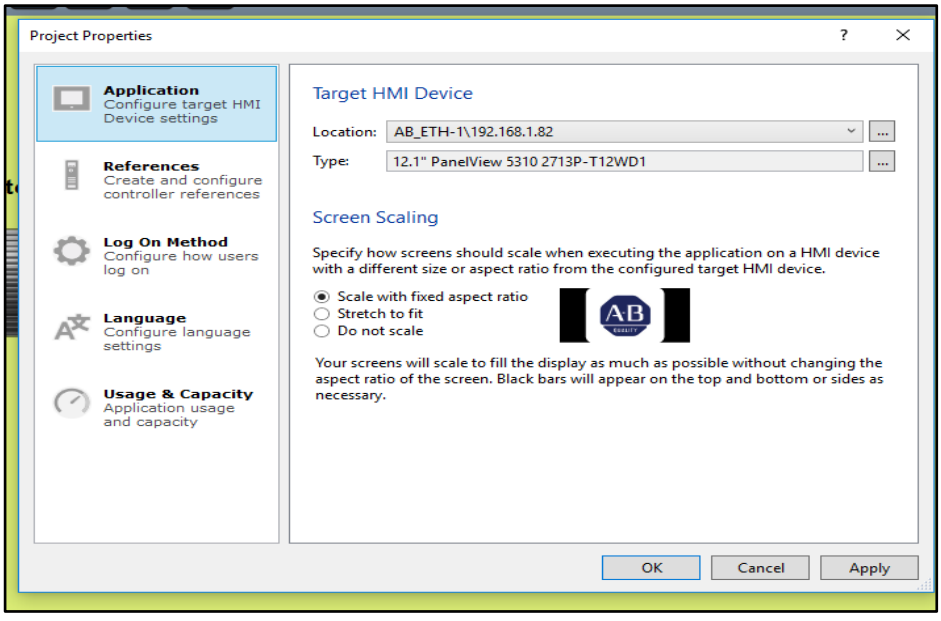


Figure 17: Selecting applications for the View design program

The information to be listed is:

a. The path for the HMI through RSlinx communication network.

b. The model of PanelView you are using, in case you want to change it from the one you specified in the beginning to download the program in another device.

Design of the HMI screen

The design of the HMI screen was done after proving the connection between the HMI and the PLC worked correctly by creating a pushbutton in the HMI that would activate the process. The HMI screen is shown in Figure 18.

The steps to design an intuitive and simple display were the following:

1. Pushbutton creation as virtual inputs to the PLC. They were added as toggle buttons and related to the start and reset bit tags.
2. Motors, different color lights, and an alarm were created as outputs from the PLC. Two states were defined in each of the outputs, 0 or 1. The change from one state to the other was displayed with a change of color in the motors and the lights and a change from invisible to visible in the case of the bell.
3. Tagging of the outputs: Motor 1, Motor 2, GPLT, RPLT, WPLT and BELL.
4. Change of the background color and title.

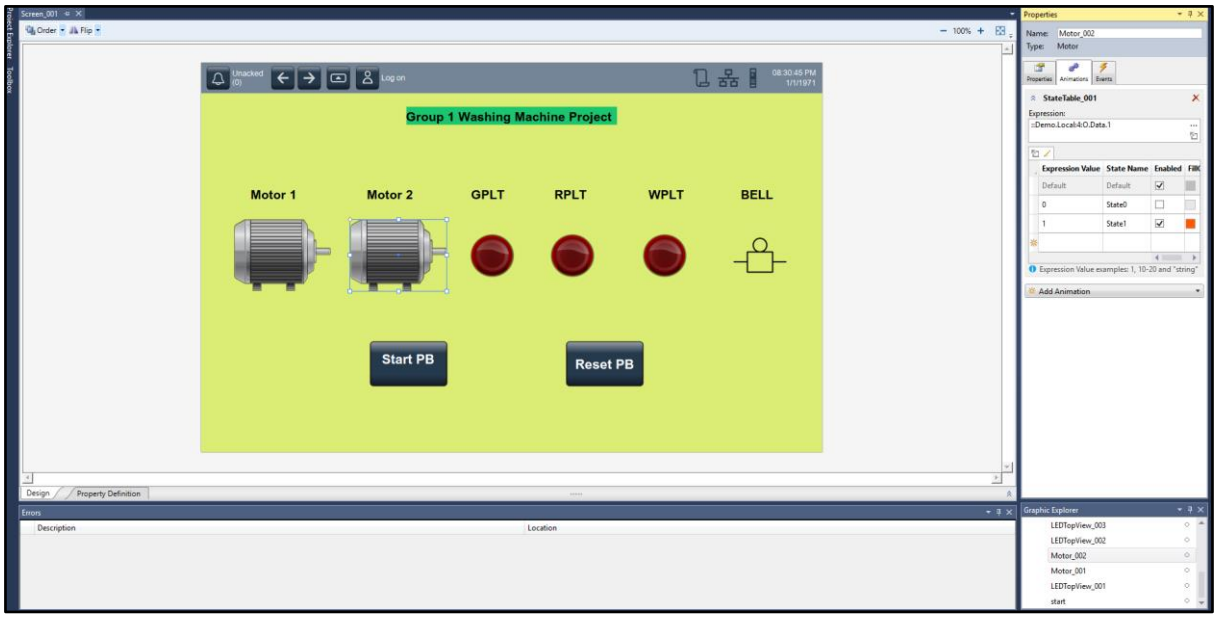


Figure 18: Designing the HMI screen

Assigning Tags in Studio 5000 View Designer

Figure 19 shows the properties window used to assign program tags from Logix Designer to graphic objects in View Designer. *Motor_002* changes color from grey to red to signify it is in either off or on states, respectively. The tag used to convey this information is *Demo.Local:4:O.Data.1*, or location 1 in the data array output by the SQO function in the main program. When this tag is 0, motor is off. When this tag is 1, motor is on. The start button is controlled by the tag *Demo.Start_Bit*, and the button behavior is set to "Set a tag to 1 on press, 0 on release." This makes the start button a momentary pushbutton, and the process can execute due to the latch bit in the program.

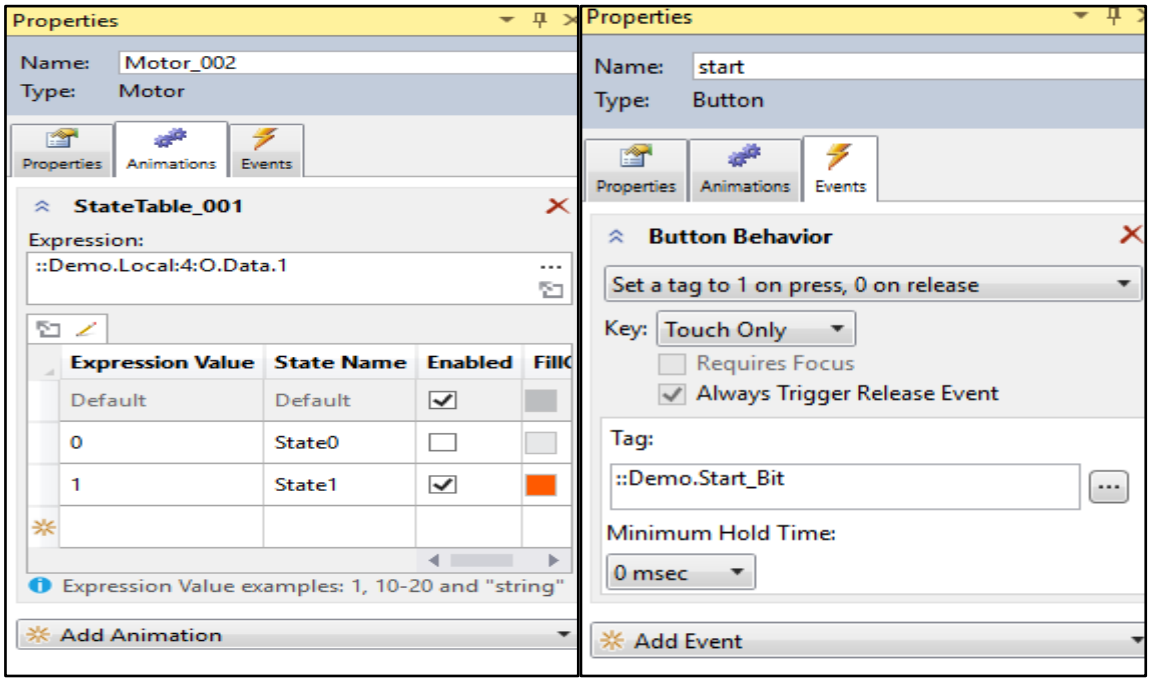


Figure 19: Linking objects in View Designer to tags in Logix program

5. Part III: Creating the network

For this section of the Capstone Project, the PLC/HMI system from Team 1 will be networked with the system built by Team 2. Team 1 is this team, and Team 2 is the opposite team, creating an identical PLC/HMI network. This group has built an identical system, which controls a washing machine that is represented by the HMI. In order to network the two projects together, it is necessary to make use of a network switch. The switch used is the Linksys Etherfast 10/100 8-port switch. This device with connected ethernet ports is shown in Figure 20. 5 ports are used: one each for Team 1 PLC, Team 1 HMI, Team 2 PLC, Team 2 HMI, and connection to the computer. The IP addresses for each device was set as a static IP, with different addresses for each of them:

- Team 1 PLC is 192.168.1.81.
- Team 1 HMI is 192.168.1.82.

- Team 2 PLC is 192.168.1.91.
- Team 2 HMI is 192.168.1.92.

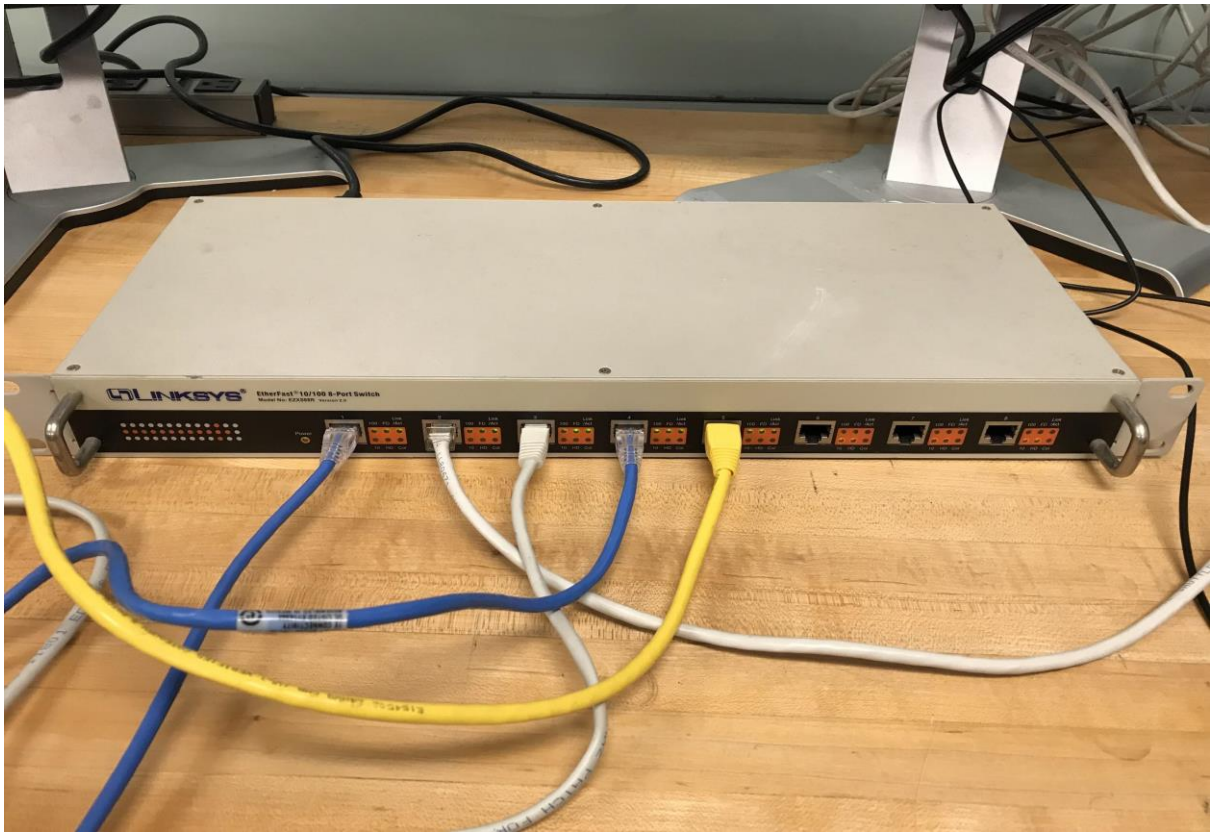


Figure 20: Linksys Etherfast 10/100 8-port switch

Updated HMI

The updated HMI screen is shown in Figure 21.

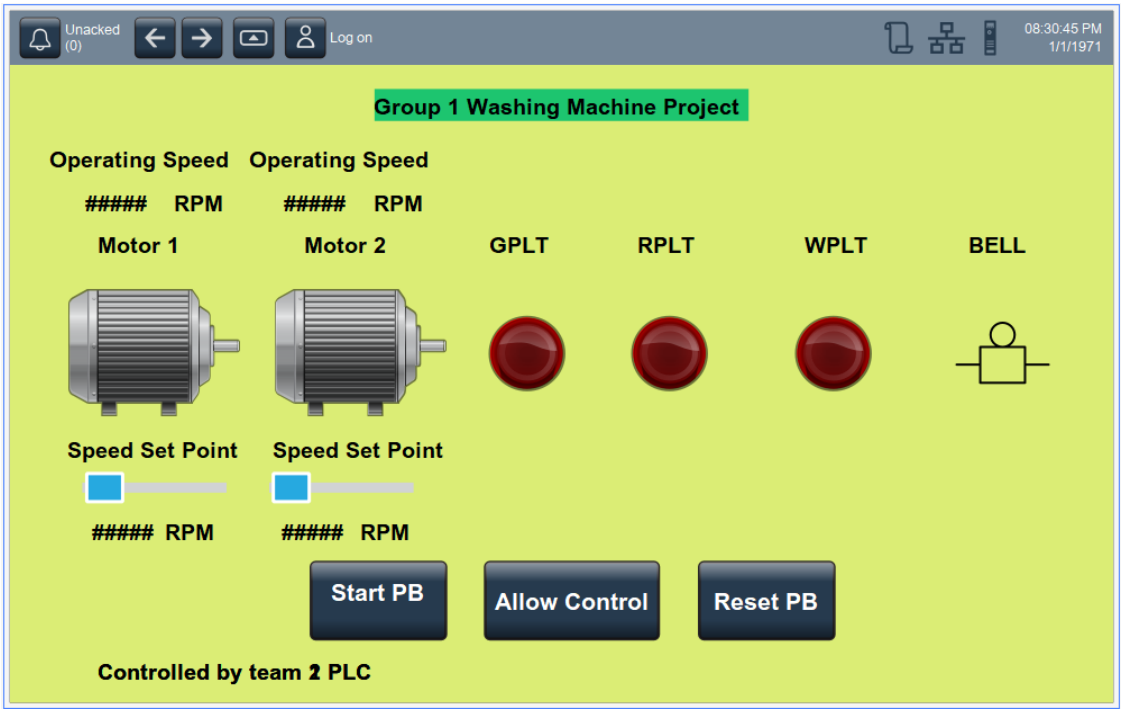


Figure 21: Updated design of HMI screen

For this revision, new graphics objects are made use of. These are sliders for changing the motor speeds, readouts for the motor speeds, a button for allowing control from Team 2 controller, and a readout for the current controller of the HMI. Properties for Motor 1 and Motor 2 sliders are shown in Figure 22. The slider objects update tags *M1_analog* and *M2_analog*, respectively. The min and max values are 0 and 650, respectively. These are arbitrary values, and they represent the speed of a virtual motor, in RPM.

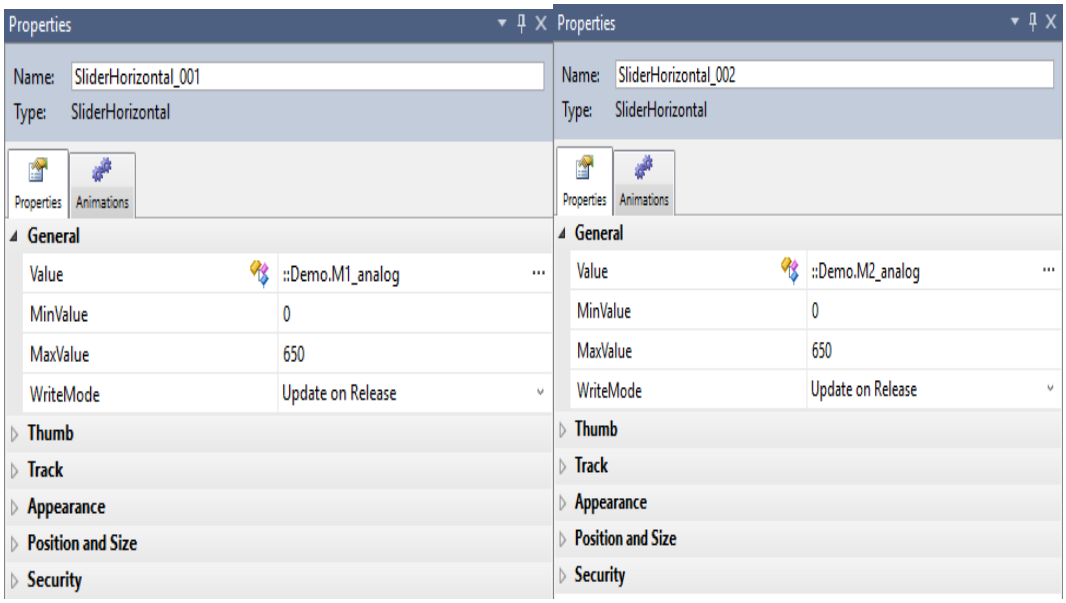


Figure 22: Tags in sliders

The properties for the speed setpoints are shown in Figure 23. These numeric displays simply display the values of the tags *M1_analog* and *M2_analog*, giving information about the set speed of each motor to the operator.

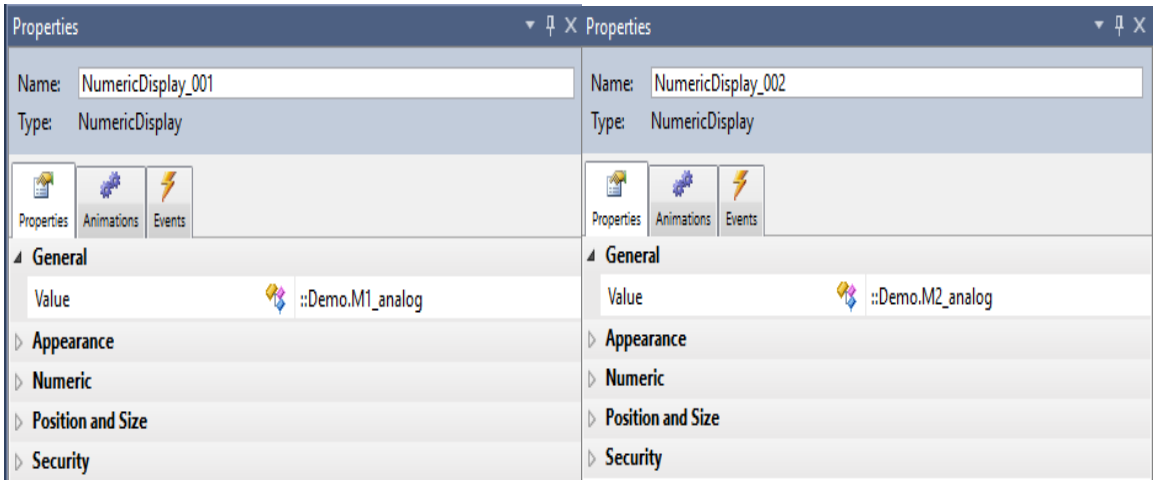


Figure 23: Linking sliders to tags in Logix program

A button, entitled “Allow Control”, was created which is used to toggle control of the HMI between Team 1 PLC and Team 2 PLC. The tag used by this button is utilized in the logic in order to enable Master/Slave networking. The properties for this button and its associated text display is shown in Figure 24. The button is set to toggle the tag *Allow_control_PB* every time the button is touched. The text display reads “Controlled by Team 2 PLC”, and is only visible when tag *Allow_control_PB* is 1, which indicates that Team 2 has control of the HMI. The text reads “Controlled by Team 1 PLC” when this tag is 0.

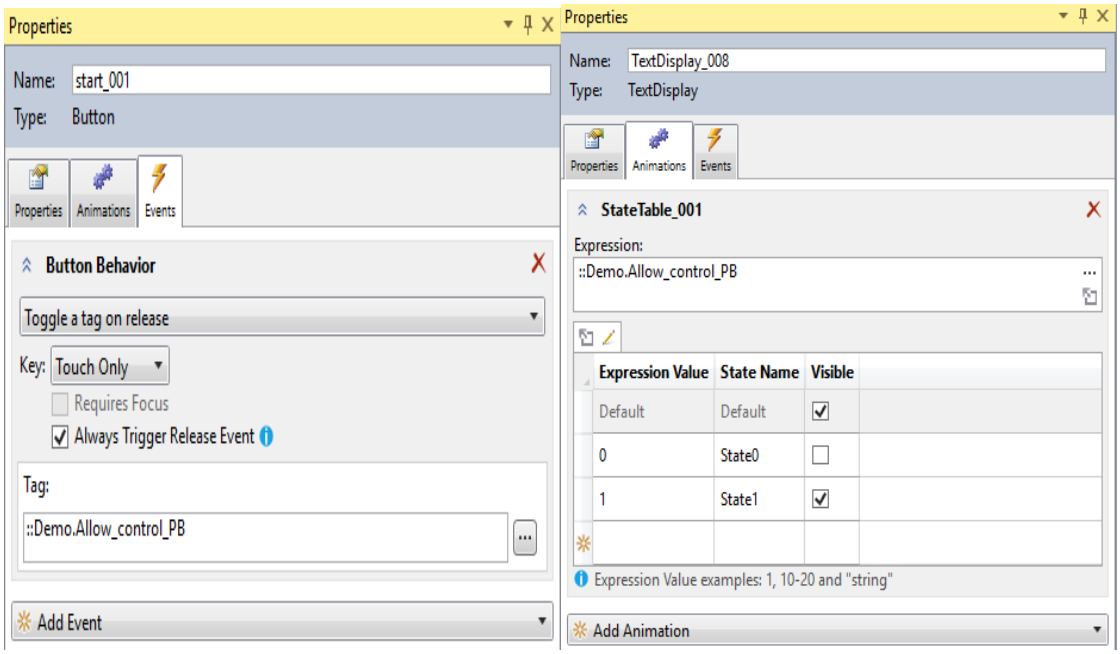


Figure 24: Allow control button properties

Tags

New tags were created in order to implement Master/Slave networking of the two team projects. The properties for tags *M1_analog* and *M2_analog* are shown in Figure 25. These tags are used by the HMI motor slider and motor display objects, for Motor 1 and Motor 2. These tags are within the scope of the controller used by Team 1, *Meng_Capstone_Activity_1,* and are of data type INT.

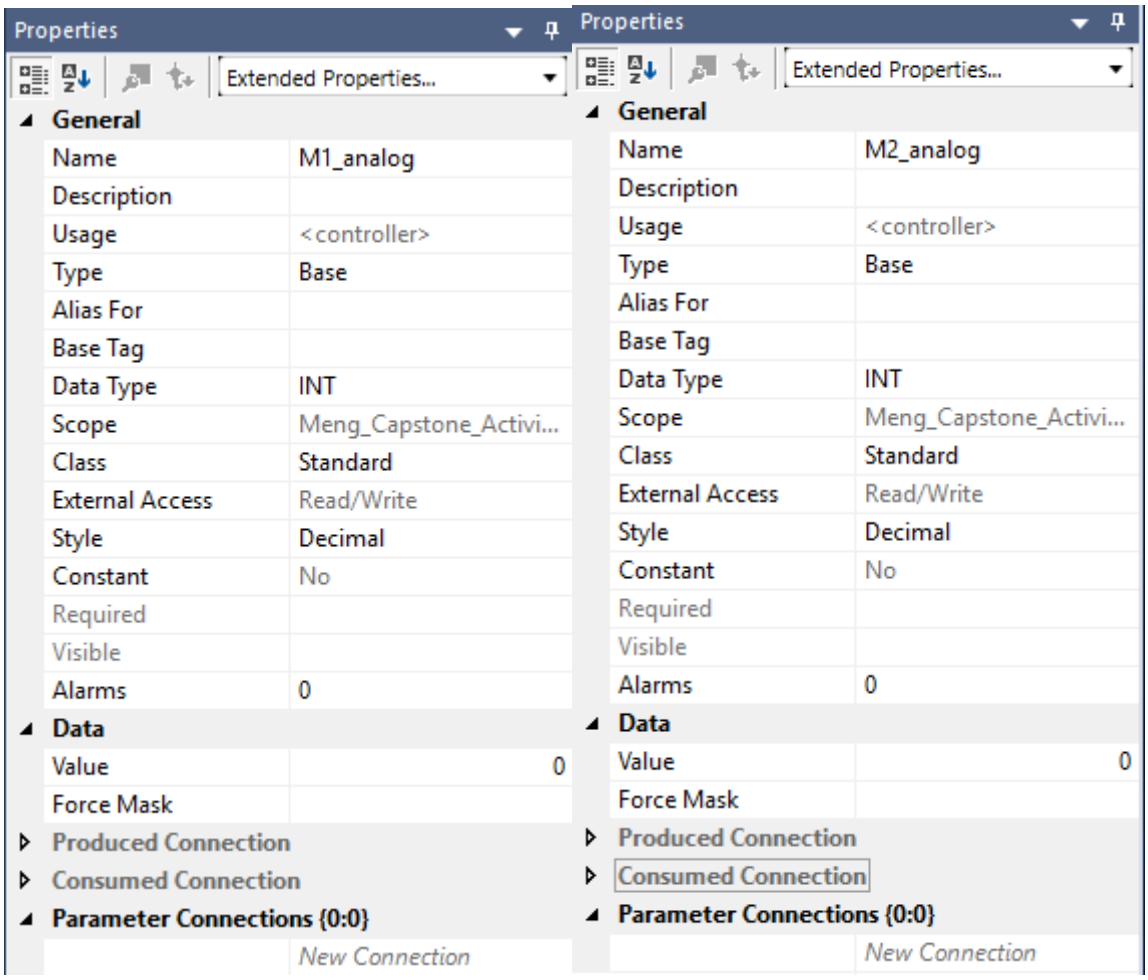


Figure 25: Tags in sliders

In order to designate one controller as the master, and one controller as the slave, produce and consume tags are utilized. These tags are able to be accessed by all PLCs within the I/O Configuration tree, therefore it was necessary to add *Team_2_PLC* to this tree within Team 1's project, so that each project has access to the same produce and consume tags. In addition, Team 2 took the same step of adding Team 1 PLC to their I/O Configuration tree. The properties for tags produced by Team 1, and consumed by Team 2, are shown in Figure 26. A Produced Tag is created for both Motor 1 and Motor 2, since these are the HMI outputs which will change control between PLC 1 and PLC 2. The usage of the tags *M1_FROM1_TO2,* and *M2_FROM1_TO2* will be discussed during the discussion of the ladder rungs.

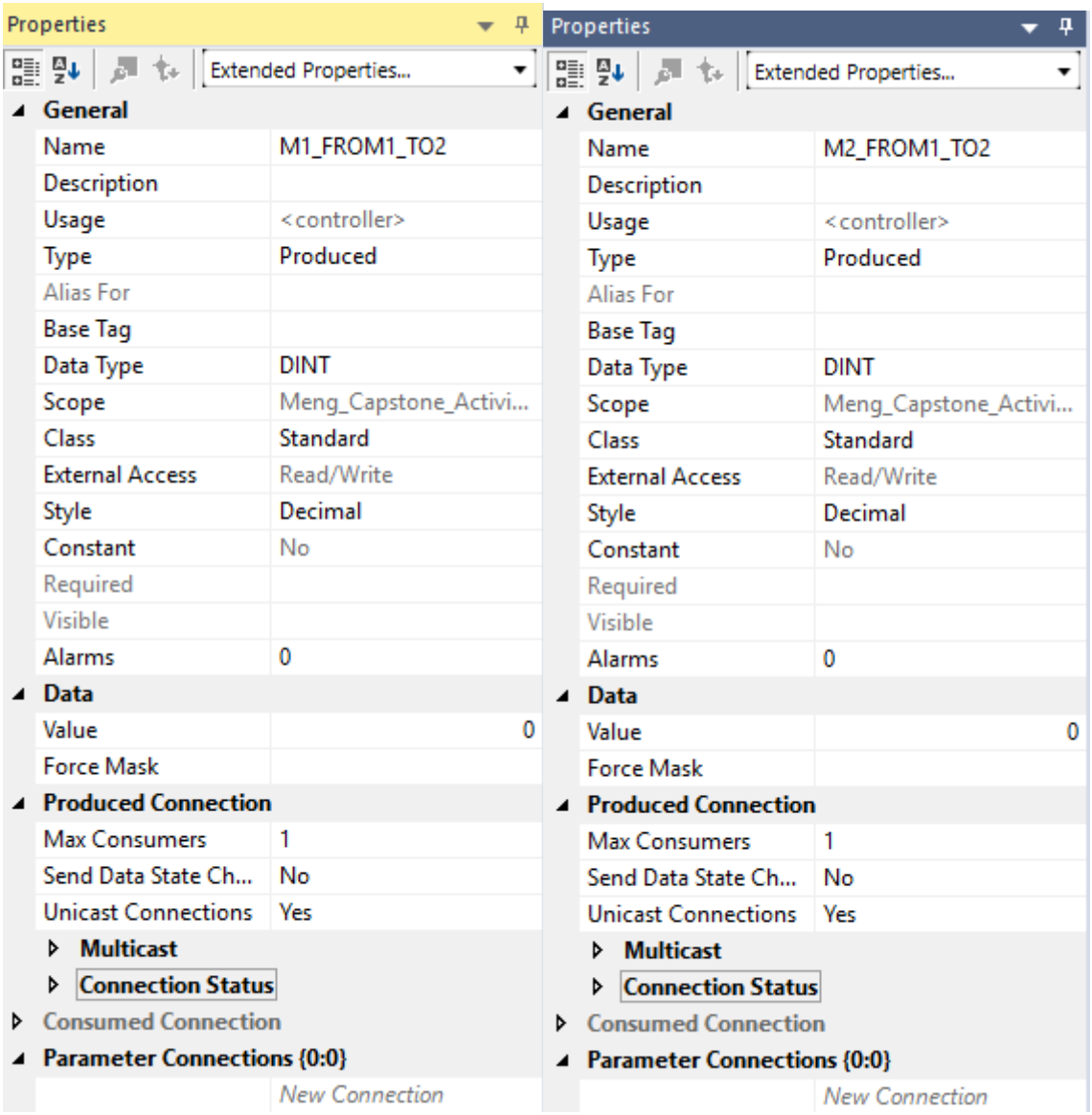


Figure 26: Produced/Consumed tag couples for RPM values

Finally, two additional tags are created for this project: *one_to_two_master,* and *two_to_one_slave.* Tag properties for these are shown in Figure 27. Tag *one_to_two_master* is produced by Team 1 PLC, and energizing this bit indicates that Team 1 is the master, and that Team 2 will use tag values from Team 1's project. Under "Produced Connection", it is shown that this tag can have a maximum of 1 consumer. The tag *two_to_one_slave* is a Consumed tag. The tag is produced by *Team_2_PLC* as *two_to_one_master*, and is used to enable the control of Team 1 HMI by Team 2. Therefore, Team 2 is the master and Team 1 is the slave, in this case. The program uses these tags to transfer control from 1 to 2, and vice versa.

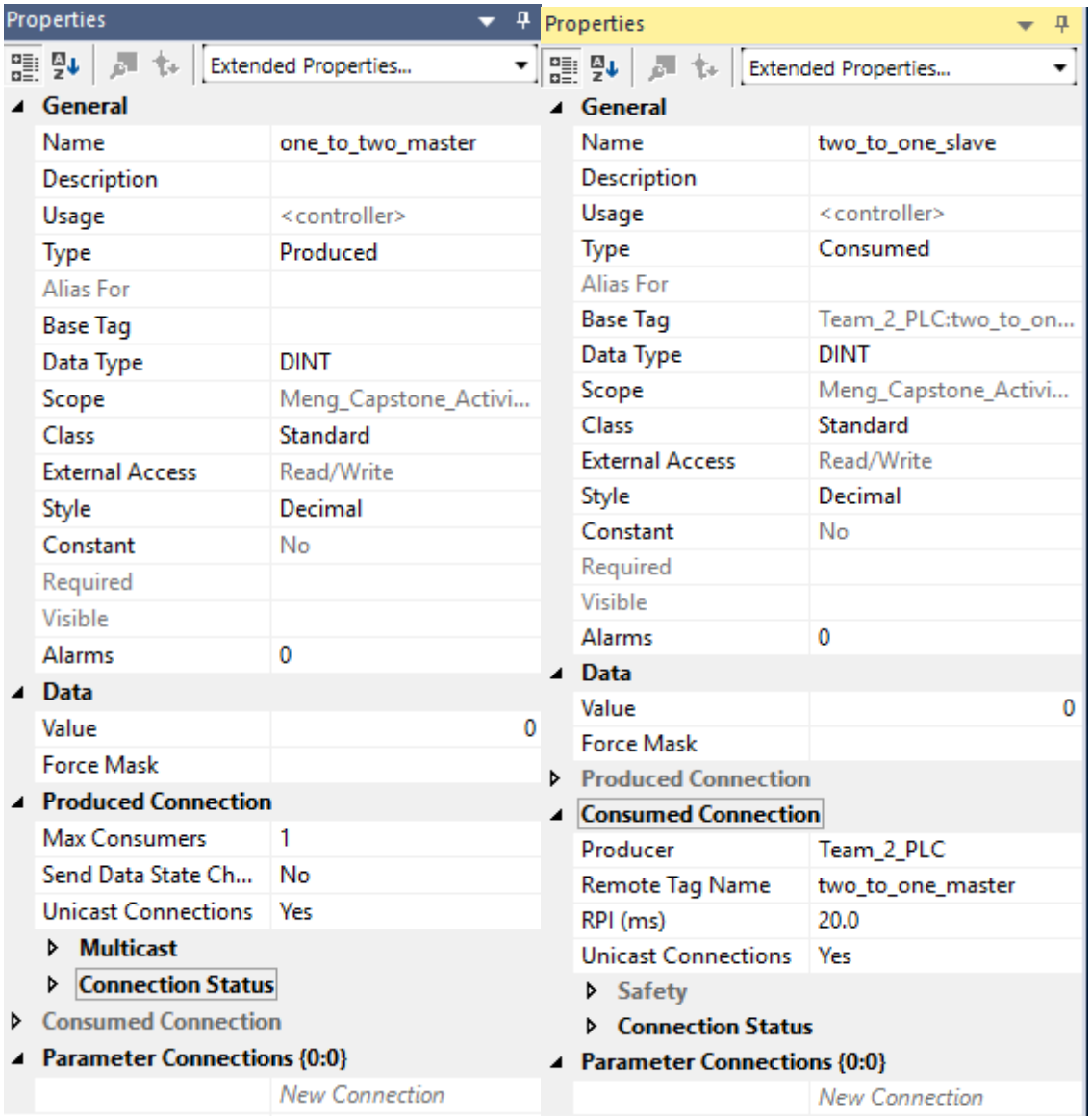


Figure 27: Produced/Consumed tag couples for output values

Logic

Additions were made to the main routine, and two new subroutine files were created, called *Control_from_Team1,* and *Control_from_Team2.* Figure 28 shows the changes made to *Main Routine.*

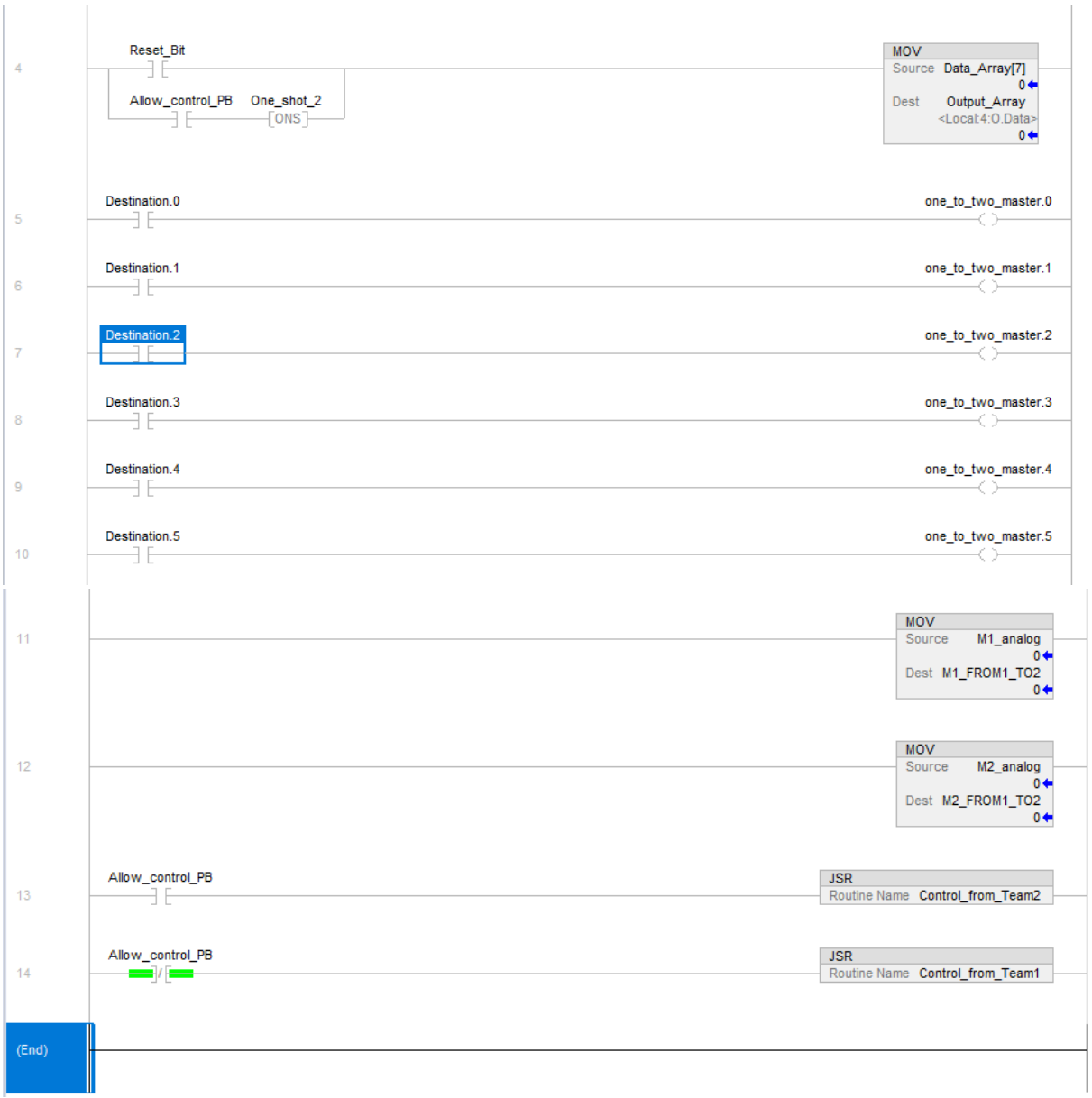


Figure 28: Changes in main routine to send the tags

Rung 4 uses *Allow_control_PB,* which is the tag that is toggled by the HMI button. A branch around *Reset_Bit* was added such that all values in *Output_Array* will become 0 initially, after control is transferred from Team 1 to Team 2. This is to signify a stop in the sequence that was originally running, so that the sequence programmed by Team 2 can be displayed. A One Shot instruction is used to ensure that the MOV instruction is only executed for one scan. Rungs 5 through 10 show the mappings of the sequencer output array elements, held in *Destination,* to the produced tags *one_to_two_master.0* through *one_to_two_master.5.* Through this produced tag, which is consumed by Team 2's controller, the sequencer outputs are continuously sent to the consuming controller. It is up to Team 2 to use these values to control their output sequence by pressing "Allow Control" on their HMI. Rungs 11 and 12 operate similarly. The MOV instruction is used to continuously move the analog motor input from Team 1 HMI into a produced tag called *M1_FROM1_TO2*, and likewise for Motor 2. These tags, which represent motor speeds, are continuously sent to Team 2 controller, which consumes

the tags. Pressing “Allow Control” on Team 2 HMI allows that set of motor speeds to be controlled by Team 1 HMI.

The first added subroutine is called *Control_from_Team2,* and the JSR instruction for it is shown in rung 13 of Figure 28. This subroutine is for the condition in which Team 1 enables control by Team 2, by pressing the “Allow Control” button on the HMI. This action energizes the *Allow_control_PB* bit, allowing the subroutine to be executed. Whenever this bit is open (checked by XIO instruction), the subroutine *Control_from_Team1* is executed. This subroutine executes logic without input from Team 2.

The subroutine file *Control_from_Team2* is shown in Figure 29. The input side of the rungs uses the *two_to_one_slave* tag, which is an array of DINTs, produced by Team 2 controller and consumed by Team 1 controller. This file is simply a mapping of Team 2 sequencer outputs to Team 1 outputs, through this consumed tag. The order is reversed, simply because Team 2 chose to use different elements of the output array for their outputs. For example, Team 1 used Local:4:O.Data.0 for Motor 1, and Team 2 used Local:4:O.Data.5 for Motor 1, which is represented here as *two_to_one_slave.5.* Rungs 6 and 7 move the consumed tags for Motor 1 speed and Motor 2 speed into the tags used by HMI, only if the output is currently on. This ensures that the speed is only displayed when the output is on. Otherwise, the speed will be 0 rpm, carried out by rungs 8 and 9.

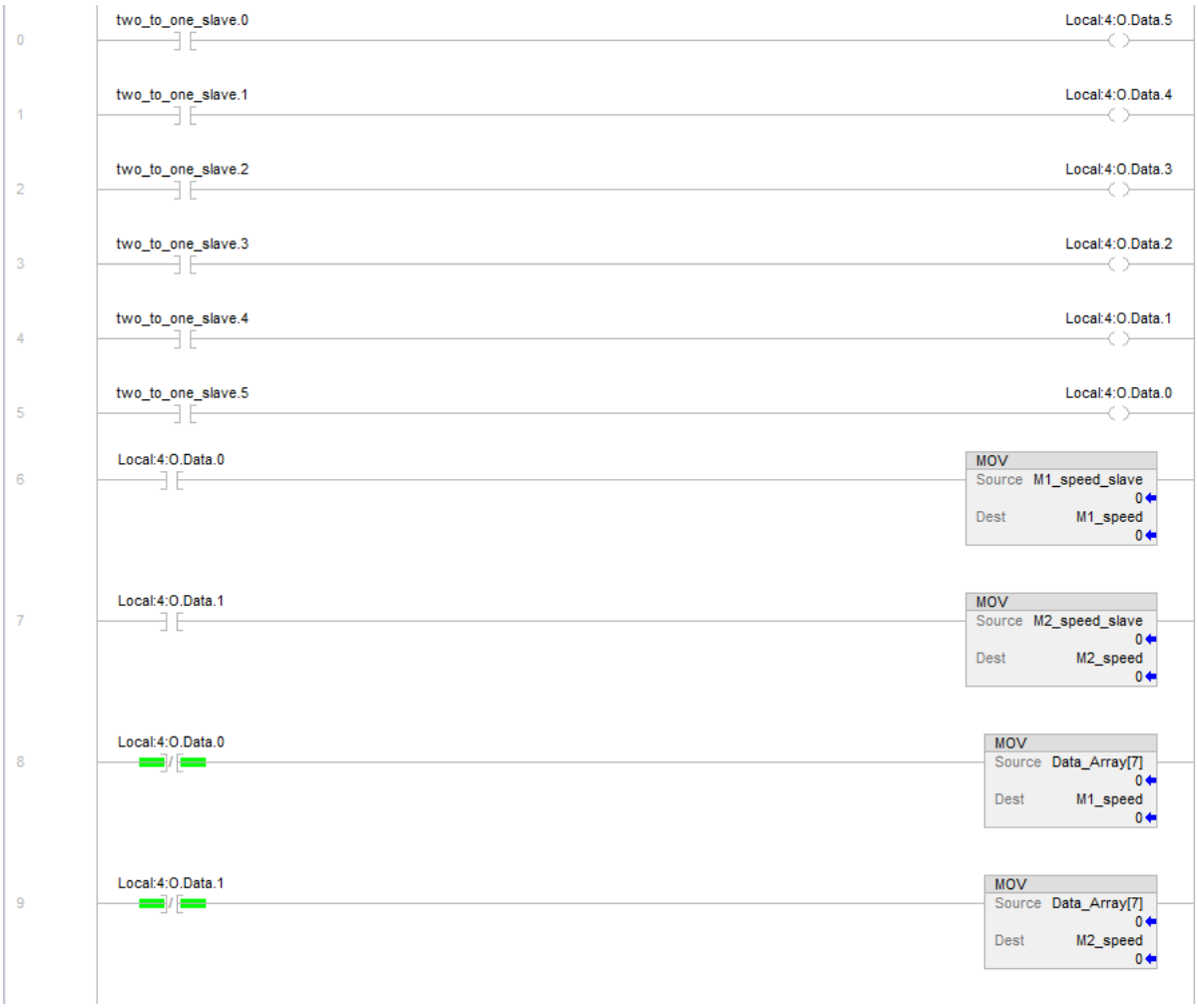


Figure 29: *Control_from_Team2* subroutine

Figure 30 shows the subroutine *Control_from_Team1.* This file is also simply a mapping of Team 1 sequencer outputs, in *Destination,* to the actual output pins in the Team 1 PLC output card. It was necessary to place the outputs in an intermediary array, *Destination,* in the main routine, because these outputs may or may not be used by the local project, depending on which controller is actually in control. Rungs 6 through 9 move the speed input into the output tag for the motor display on Team 1 HMI. Finally, this logic has all been mirrored in the project by Team 2, such that the master/slave operation is identical to both projects.

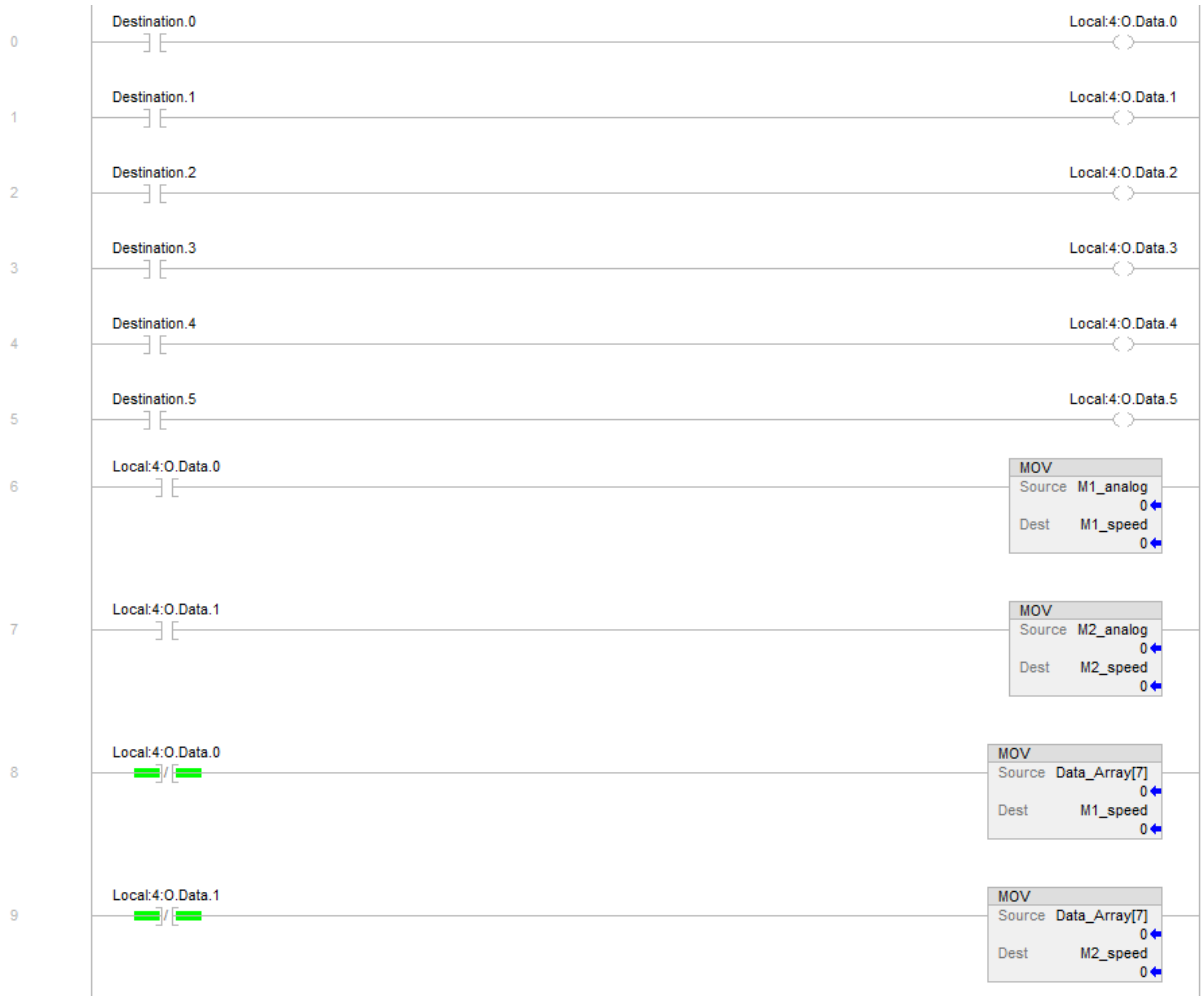


Figure 30: *Control_from_Team1* subroutine

Test Setup

Figure 31 shows the final test setup. In the picture, the HMI on top is Team 2, below that is Team 1. The PLC to the left is Team 1, and to the right is Team 2. They are all connected to the network switch, which is not pictured.

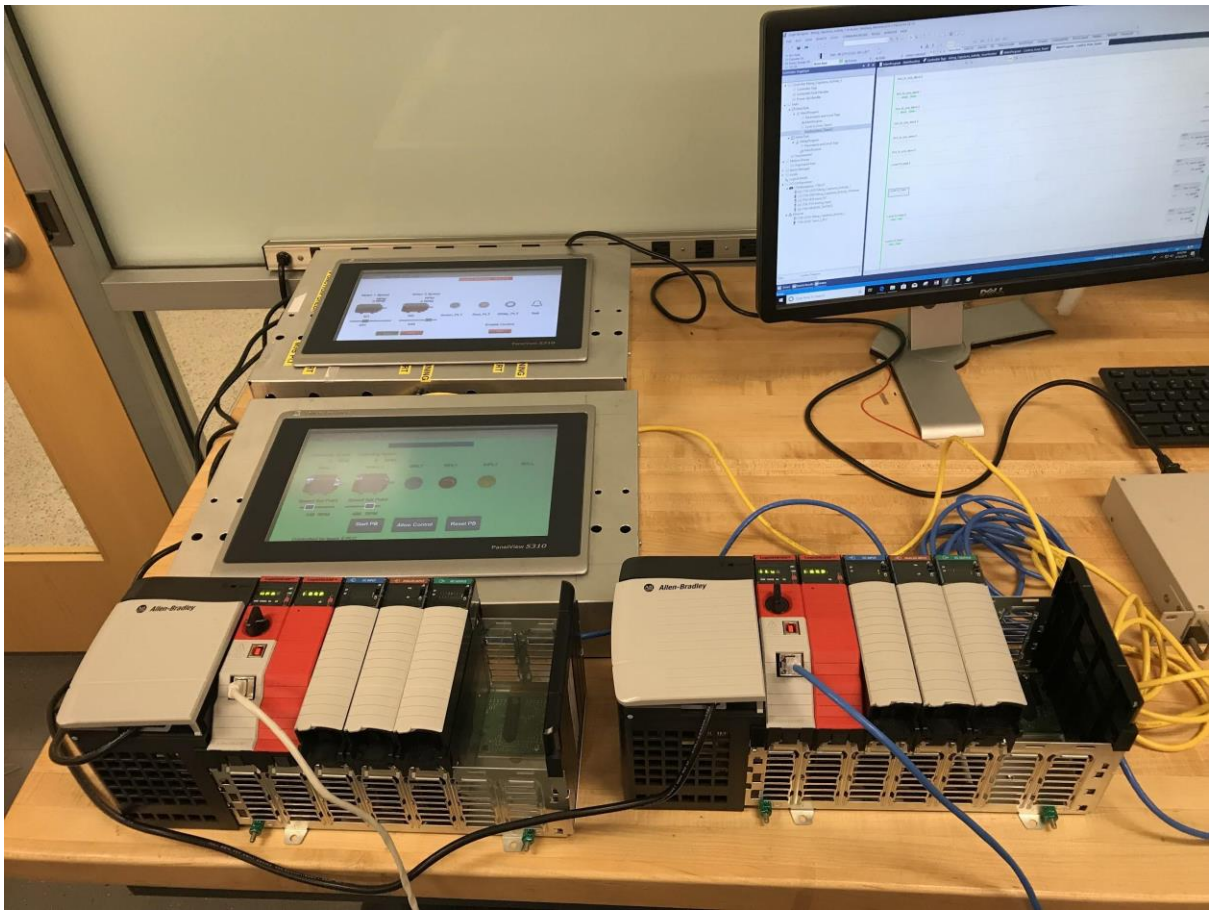


Figure 31: Final setup

The above logic was tested by both teams with PLC and HMI setup. Firstly, the case of Team 1 controlling Team 2's output is considered. Figures 32 and 33 shows Team 1 HMI on the left, and Team 2 HMI on the right. The *Enable Control* button is pressed on Team 2 HMI, as is evident by the text "Controlled by Team 1 PLC". To clarify, an on state of a motor is represented by red in Team 1, and off state is represented by grey. For Team 2, on state is represented by green, and off state is represented by red. In figure 32, Motor 1 is being energized in both HMI by the code in Team 1 PLC. The RPM value follows the set point in Team 1 HMI. So, even though Team 2 is trying to set their speed as 231 RPM, it is actually running at 135 RPM, as indicated by the text display. Similarly, in Figure 33, both Motor 2 are running at the speed set by Team 1 HMI.

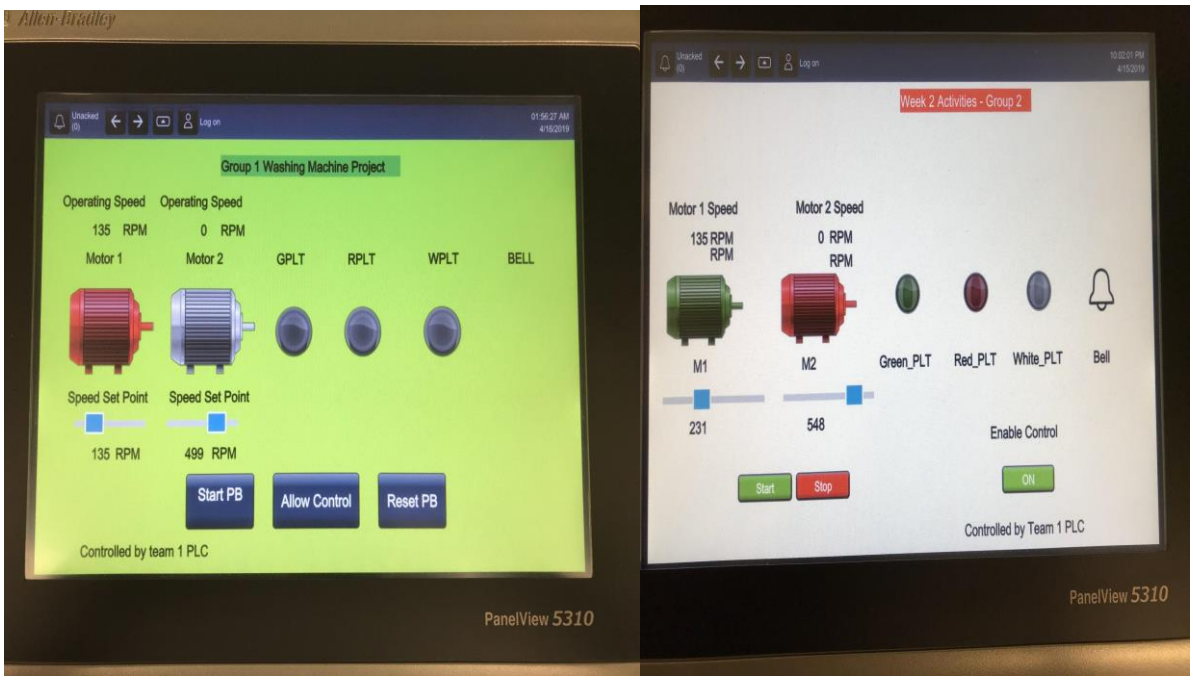


Figure 32: Team 1 PLC controlling Team 2 PLC (M1 ON)

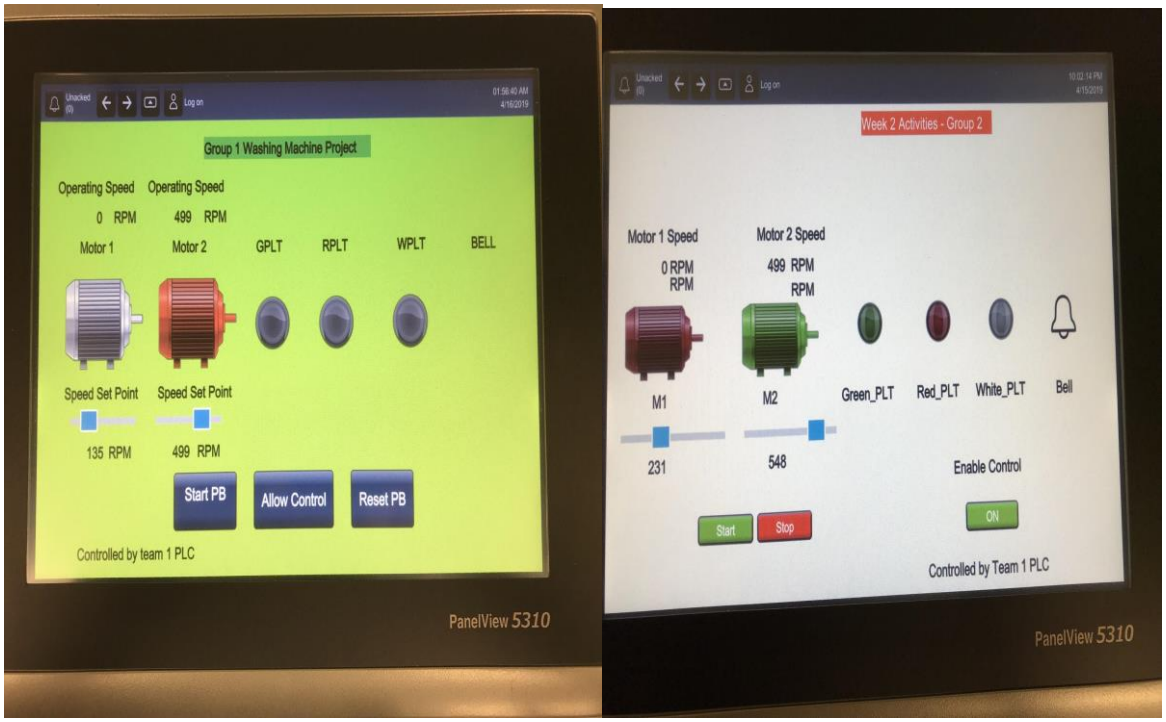


Figure 33:Team 1 PLC controlling Team 2 PLC (M2 ON)

The next test to be considered is the controlling of Team 1 HMI by Team 2. Figure 34 shows a comparison of the two HMI. “Allow Control” is pressed on Team 1 HMI, so both HMI are

running the sequence in Team 2 PLC. In the pictured state, both motors are energized, and the speeds match those specified by the operator of Team 2 HMI.

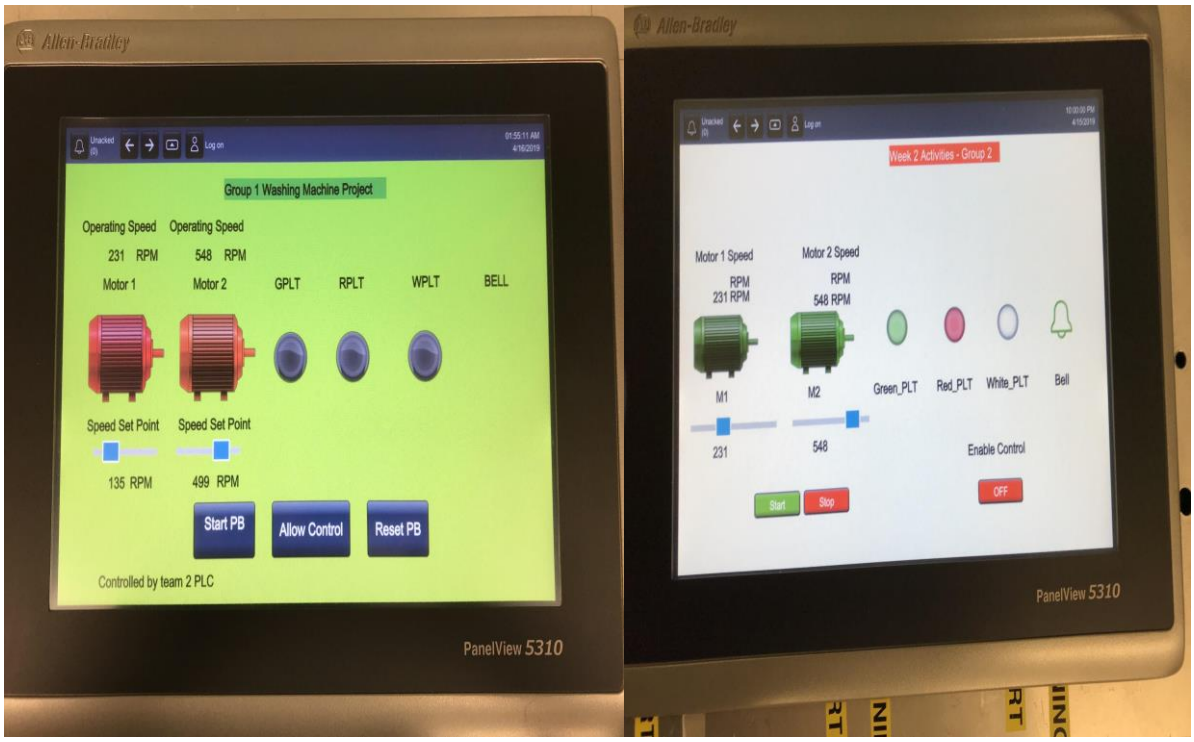


Figure 34.Team 2 PLC controlling Team 1 PLC

Through these tests, it has been confirmed that the master/slave network of the PLC/HMI system is functional. It is possible to control one HMI from another, and to write values into tags using the produce and consume functionality. This network operates both ways, as both HMI have the opportunity to control the other.

6. Conclusion and Future Work

The main goal of this project was to establish a local network between two Allen Bradley PLCs and be able to control each of them from the other PLC. Moreover, each of the PLCs should be connected to an HMI to show the animation of an automated process. The necessary steps to create the local network included configuring a single PLC, connecting an HMI and creating an intuitive interface and finally, establishing a connection between the two PLCs.

This project can also be seen as a setup configuration to develop a testbed for PLC cybersecurity research. There is still many work to be done in this field to reduce vulnerabilities and make industrial automation systems safer. Dr. Carla Purdy and her students will continue working on CyberSecurity methods for PLCs and the work done in this project will serve them as a platform to test the effectiveness of their solutions.

7. References

[1] Allen-Bradley. Logix 5000 Controllers I/O and Tag Data. Rockwell Automation, 2018.

[2] Allen-Bradley. ControlLogix 5580 and GuardLogix 5580 Controllers. Rockwell Automation, 2018.

[3] Allen-Bradley. Panelview 5310 Terminals. Rockwell Automation, 2018.

[4] Allen-Bradley. Studio 5000 View Designer User Manual. Rockwell Automation, 2018.

[5] Allen-Bradley. RSLogix 5000 Controllers General Instructions Reference Manual . Rockwell Automation, 2018.

[6] Fundamentals of Logix 5000 Systems. Rockwell Automation, 2003.

[7] Jack, Hugh. Automating Manufacturing Systems with PLCs. 2007.

[8] J. McDonald, “Developing and Defining Basic SCADA System Concepts,” IEEE Conference Paper on Advanced Control Systems, 1993, pp. B3-1 - B3-5.

[9] Queiroz, Carlos, et al. “Building a SCADA Security Testbed.” 2009 Third International Conference on Network and System Security, 2009, doi:10.1109/nss.2009.82.

[10] R. Langner, “Stuxnet: Dissecting a Cyberwarfare Weapon”, IEEE Security and Privacy, May/June, pp. 49-51, 2011.

[11] Ten, Chee-Wooi, et al., “Vulnerability Assessment of Cybersecurity for SCADA Systems”, IEEE Transactions on Power Systems, November, Vol. 23, No. 4, pp. 1836-1846, 2008.