Fluid*Mechatronix*™

Process Automation System

Curriculum Sample

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

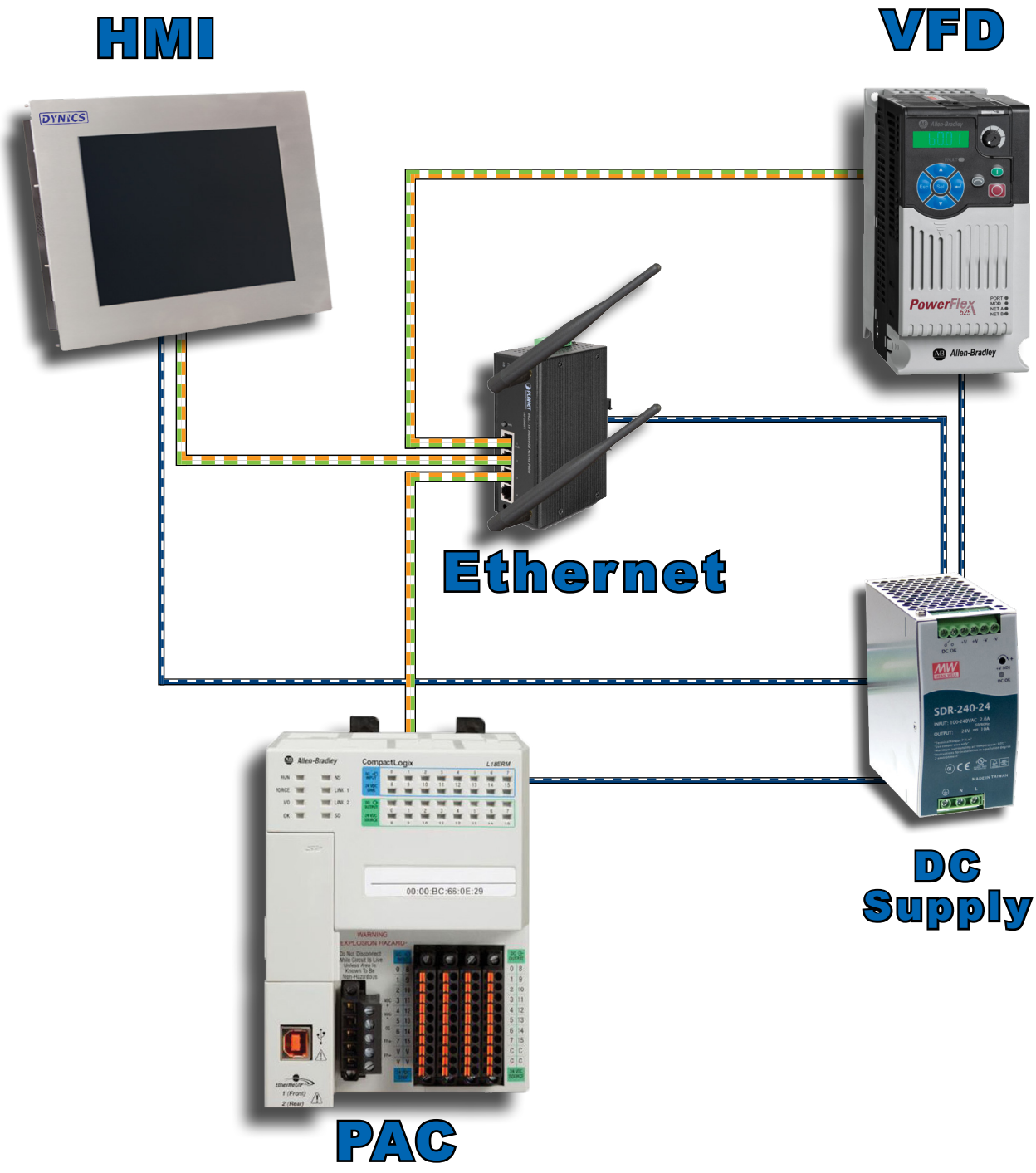
Table of Contents

Table of Contents

Table of Contents

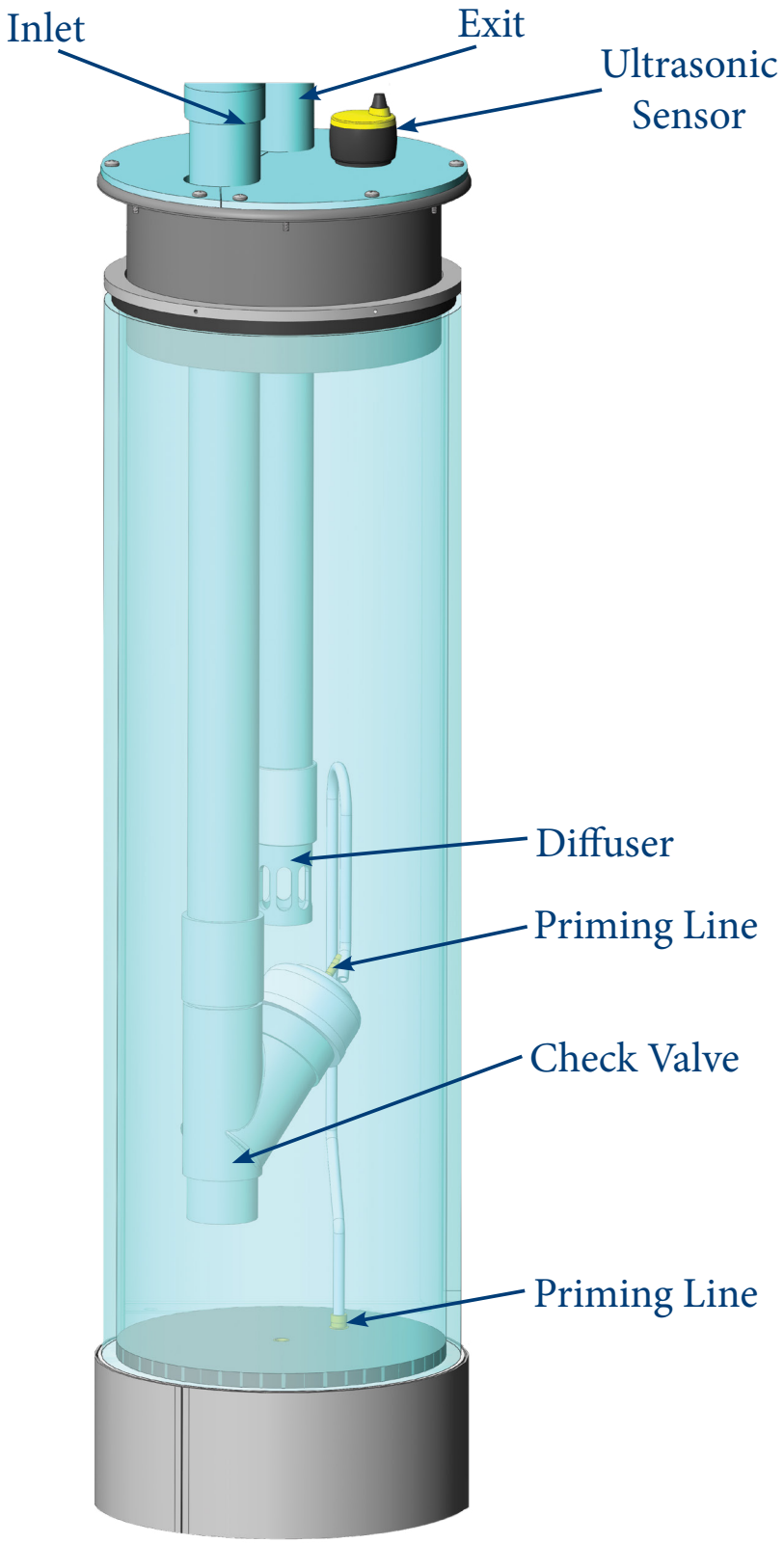
Table of Contents

Unit 1 - General Overview



Section 3: Tank Controls

Primary Tank Components

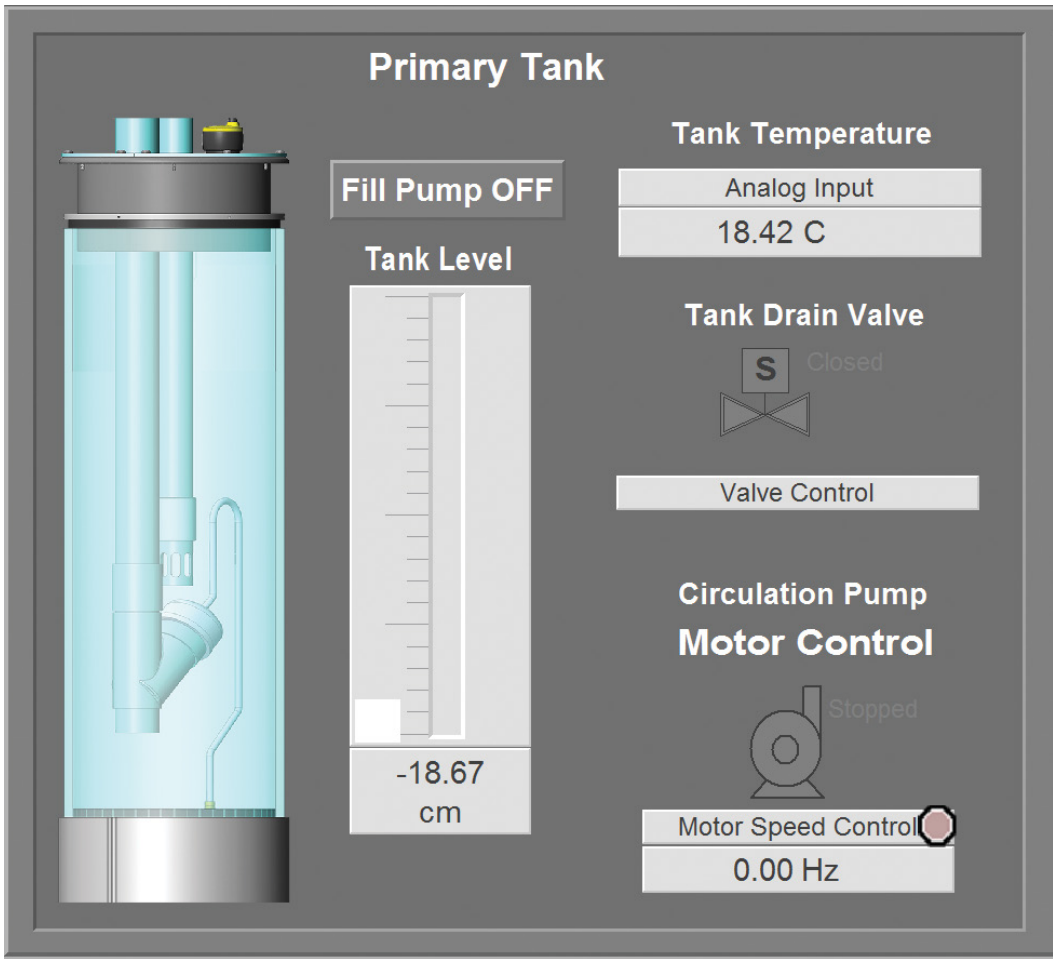


Section 3: Tank Controls

Focus 1 - Primary Tank Controls

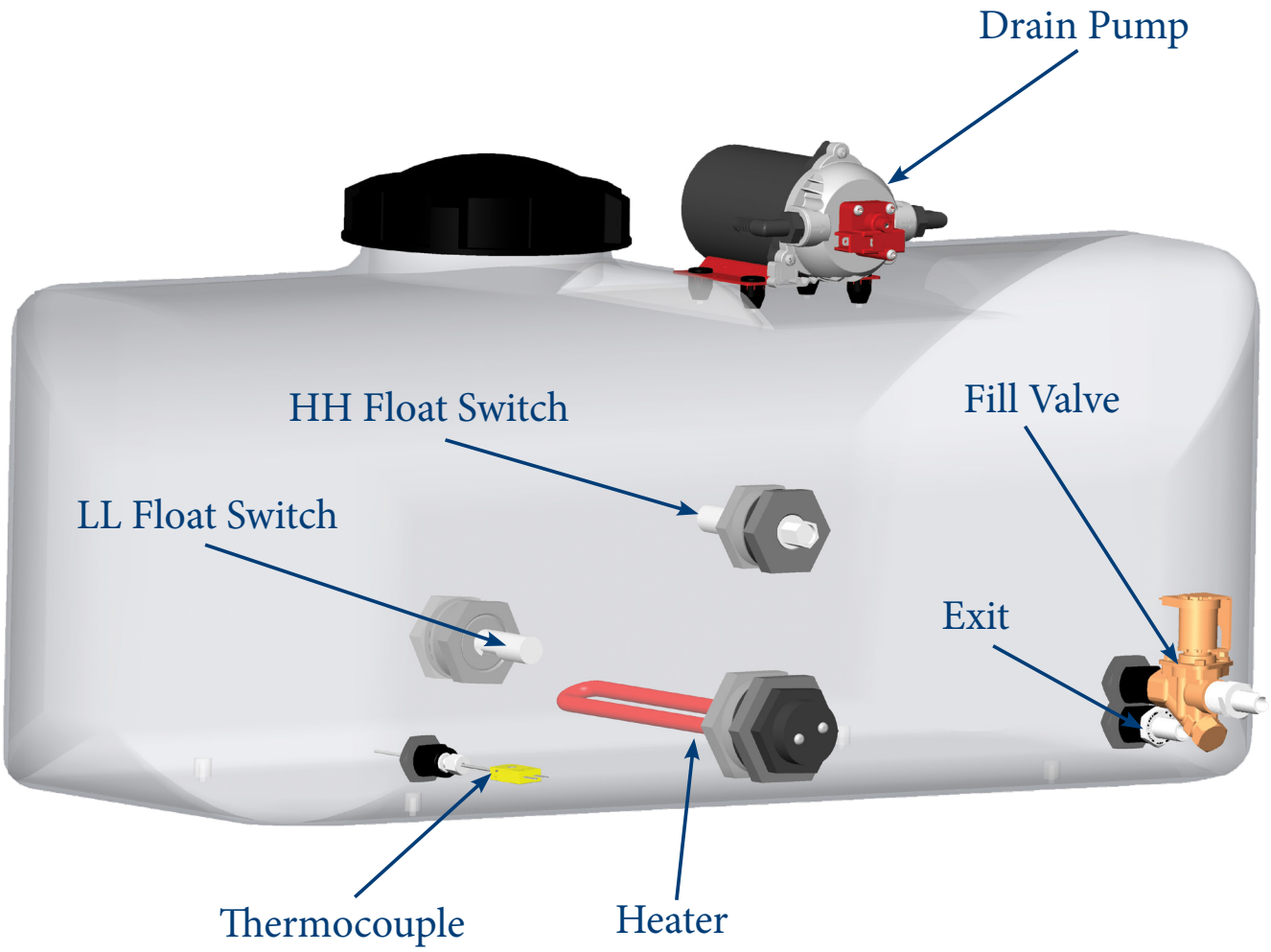
Fluid*Mechatronix*™ is equipped with two different fluid storage tanks. The main tank is the visible tank on the corner of the unit. The secondary tank is inside the unit. The main tank represents the main fluid and system monitoring. The main tank has a check valve near the bottom of the tank fluid inlet. This helps hold the prime of the fluid circuit on the top of the unit. The tank exit has a diffuser installed on the end. This changes the direction of fluid flow when re-entering the tank and will reduce tank agitation and air bubbles. The small line coming in above the check valve is for priming the flow system. This forces water into the top flow circuit so the main pump can be operated.

On top of the tank there is an ultrasonic sensor to indicate main tank fluid levels. This sensor monitors the level of the fluid in the tank with high accuracy. This is discussed further in Section 3 > Focus 4.



Section 3: Tank Controls

Secondary Tank Components

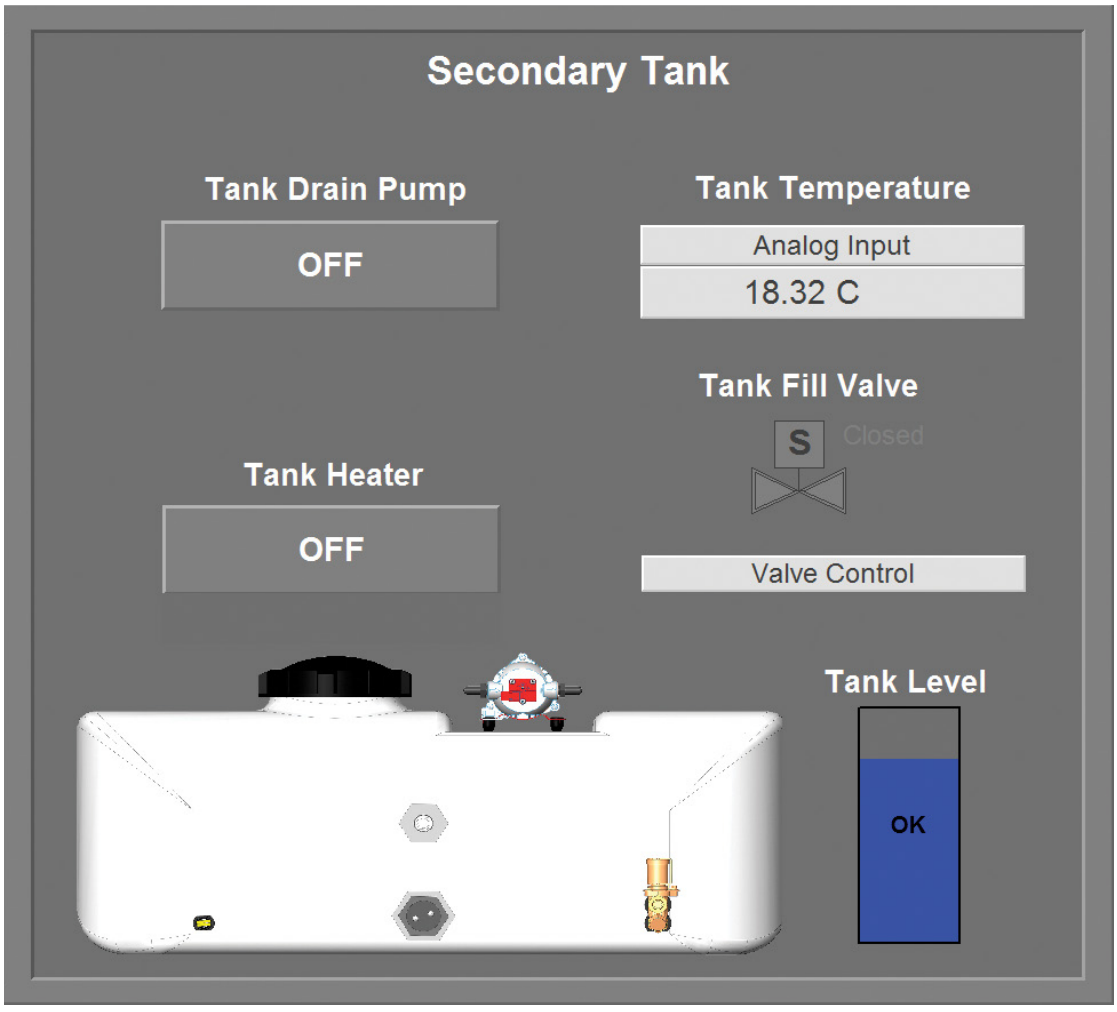


Section 3: Tank Controls

Focus 2 - Secondary Tank Controls

The second tank provides more flexibility of the system. The second tank contains a heater, a valve for level control, and a pump for moving fluid from the secondary tank to the main tank. There are also high and low float switches for over and under filling.

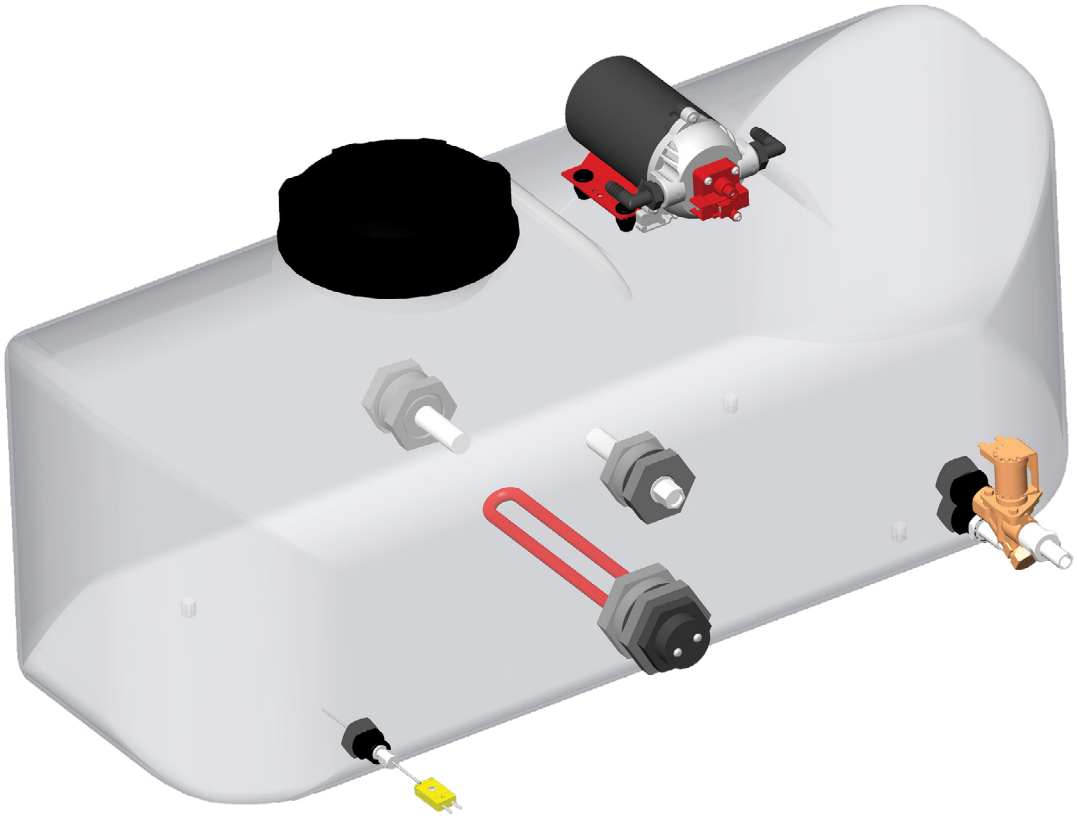
The tank has an inlet from the bottom of the main tank, and has an exit that is pumped back into the main tank. At the inlet there is a solenoid valve that allows controlled entrance of water into the tank. The valve is small, which increases the drain time of the main tank. This helps simulate a large main tank with slow drain times.



Focus 3 - Temperature Control

The second tank has a heating circuit that allows the user to heat the water for 10 minutes. This tank gives Fluid*Mechatronix*™ the ablility to mix fluids with different temperatures in one system. There is a thermocouple mounted in the tank for temperature monitoring. The heater is programmed to operate between 10° and 50° C. If the thermocouple detects a temperature that is out of range, the heating circuit relay will not engage. The heating circuit raises the temperature in the tank about 5 degrees. There is a timing circuit on the heater, so the heater automatically shuts off after 10 minutes. This is a safety measure to prevent heating element damage if the tank runs low on water.

The tank exit pump is wired with the LL float switch. The float switch is located above the heating element in the back of the tank. This prevents the pump from removing too much water, and exposing the heating element.



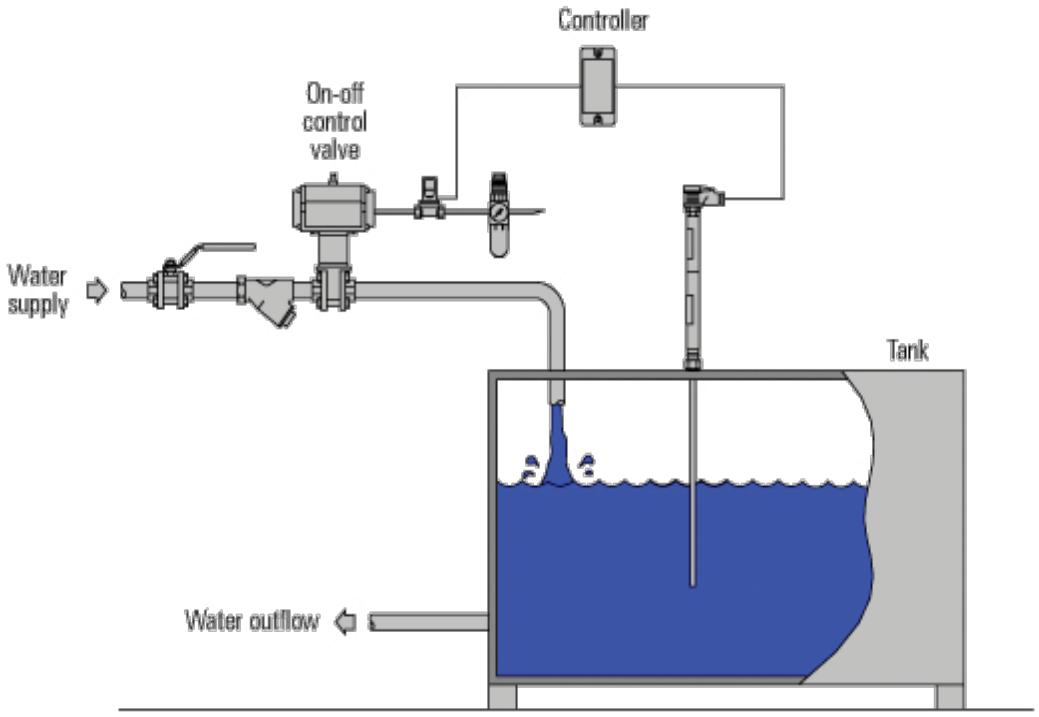
Section 3: Tank Controls

Focus 4 - Fluid Level Control with Pumps and Valves

Pumps and solenoids perform 2 functions. First, they can removes water from a tank to decrease the fluid levels. This would be useful when studying the setpoints and alarms of a tank system. Second, they can increase the fluid levels in a tank. For a solenoid, water flows from a tank to the secondary tank when this solenoid is open. Head pressure or water pressure is the only thing moving the fluid, so the process can be a little slow. When using head pressure alone, the increase or decrease in the tank can be difficult to see visually.

Fluid*Mechatronix*™ has fluid level control in 4 variations:

- Drain a tank with a solenoid valve
- Fill a tank with a solenoid valve
- Drain a tank with a pump
- Fill a tank witha pump



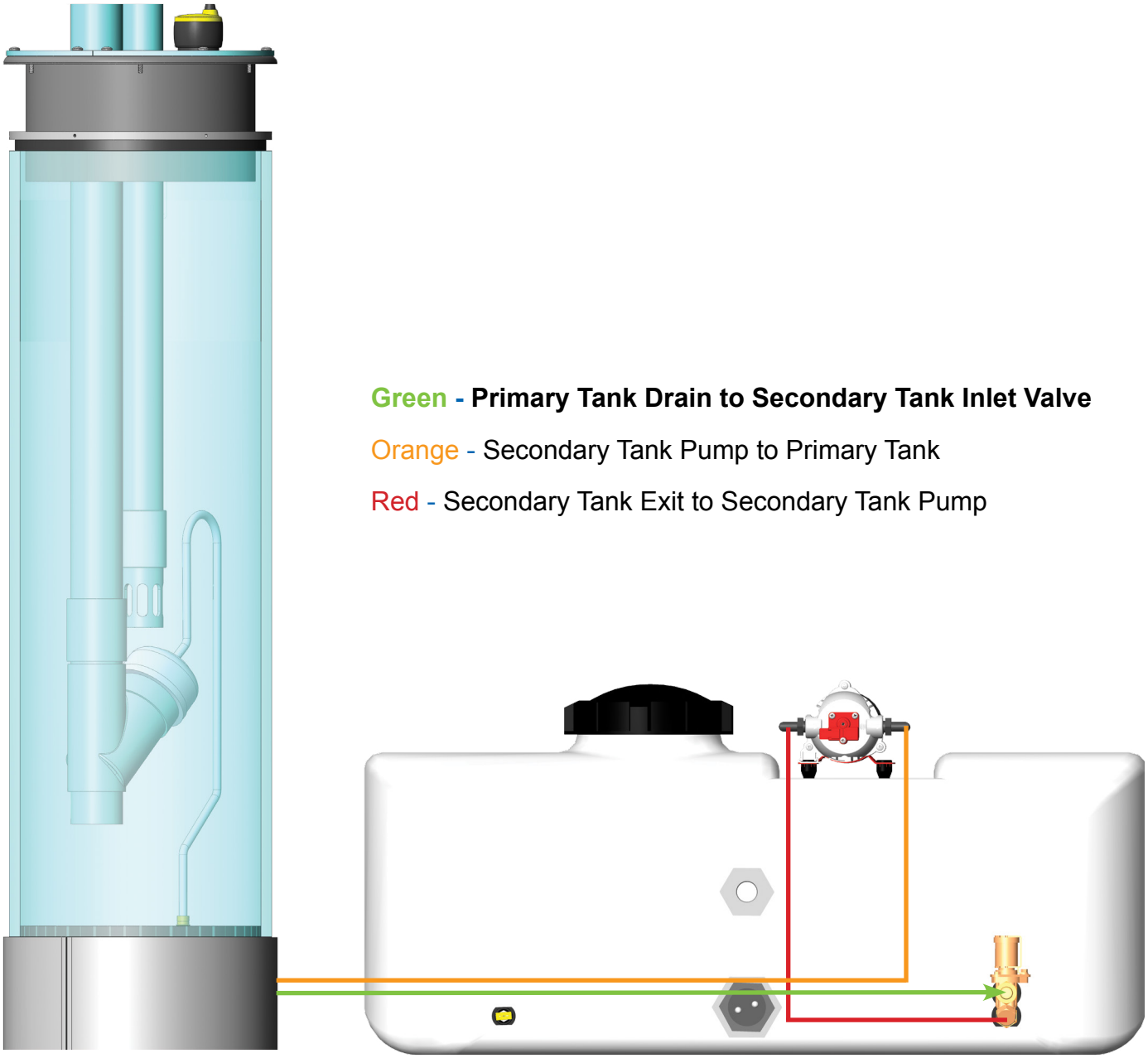
Basic Tank Level Control System

Section 3: Tank Controls

Focus 5 - Solenoid Valve Control

Opening the secondary tank solenoid affects the levels of the main tank. When the solenoid is open, fluid drains from the primary tank and fills the secondary tank. This would be very similar to a system with a large tank, and a relatively small inlet and exit. The decreasing fluid level of the main tank is difficult to see, but the software can graphically track the process and give the operator a visual indication of the tanks level and progress.

Fluid will fill the secondary tank until the HH float limit switch is reached. When fluid reaches the float and lifts it up, the solenoids electrical circuit is broken and it returns to the default state of normally closed. See diagram below for the plumbing of the Primary and Secondary Tank:



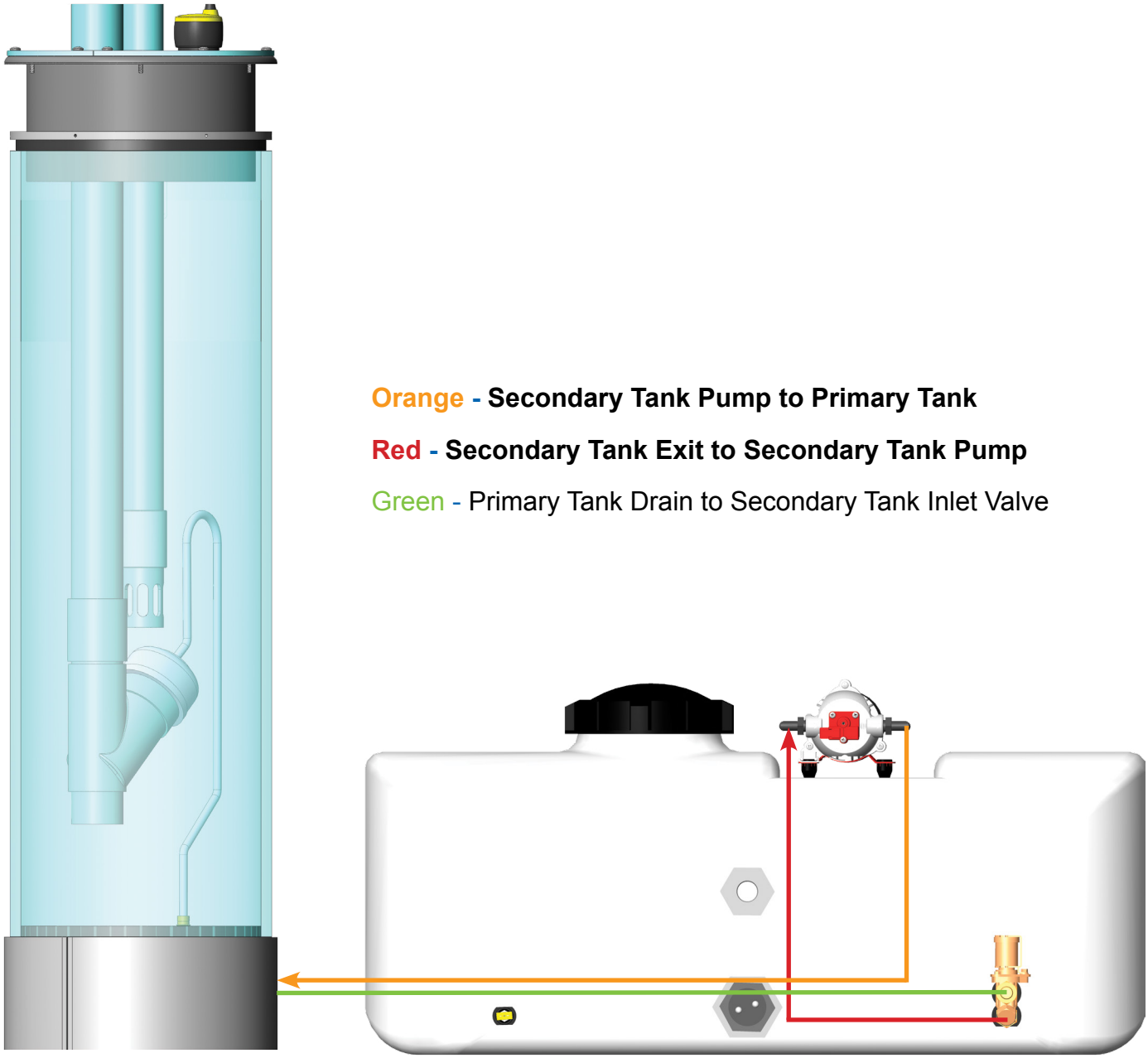
Green - **Primary Tank Drain to Secondary Tank Inlet Valve**

Orange - Secondary Tank Pump to Primary Tank

Red - Secondary Tank Exit to Secondary Tank Pump

Focus 6 - Pump Control

The secondary tank pump also affects the levels of the main tank. When the secondary tank is full of fluid, the pump can be enabled to pump fluid back to the primary tank. The pump runs continuously until it is shut off, or until it reaches the LL float limit switch. If the LL float limit switch falls to its open position due to lack of water, the prime pump will turn off. In reality, the float switch stops passing electricity to the pump relay. This causes the relay to turn off, and the relays output drives the pump, therefore the pump will turn off. The pump will not be able to turn back on until the fluid levels in the secondary tank increase.



Orange - Secondary Tank Pump to Primary Tank

Red - Secondary Tank Exit to Secondary Tank Pump

Green - Primary Tank Drain to Secondary Tank Inlet Valve

Section 3: Tank Controls

Skill Builder Section 1, Focus 1-6

1. What are some of the Primary Tank Control features?

2. Describe a similar situation where fluid level control is used.

3. What does a PLR typically show on the LCD display?

Section 3: Power Supply Requirements

Focus 3: Choosing a Power Supply

Most power supplies are designed to handle higher inrush currents for a brief period of time (like 1 second). After that, the electrical load of the system needs to be below the power supplies ratings. It is a good practice to avoid constant operation of a power supply near its limits. Typically the power supply needs to be about 20% larger than the maximum load conditions of the electrical devices connected to it. What would the proper power supply size be for the 5A motor?

To be safe, we want a supply 20% larger than normal load so we need to multiply by 120%

$$P = IV = 5A \cdot 24V = 120W \cdot 120\% = 144W$$

In this case, it would be best to purchase a supply that can handle 144 watts of power continuously and can handle an inrush of 216 watts.

Skill Builder

Section 3, Focus 1

1. A 12V motor has a maximum current draw of 7A, with a normal operating current of 5A. Calculate the proper size for the power supply.

2. If a 2W PAC and a 100W HMI are connected to the motor above, what size power supply would be required to operate the system?

Unit 2 - Centrifugal Pumping

Chapter 1 - Introduction to Centrifugal Pumping

Student will understand what a centrifugal pump is, what the major components of a centrifugal pump are, what pumping head is, and how to size a pump impeller.

Section 1: Define Centrifugal Pumping

Centrifugal pumps are used in many applications including water, sewage, petroleum and petrochemical pumping. They have become a primary component of many industries. A centrifugal pump is a machine that gives energy to a fluid. This energy infusion can cause a liquid to flow, rise to a higher level, or both.

The centrifugal pump is a very simple rotary machine which consists of two major parts: 1) impeller (rotating component) and 2) volute (the stationary structure or casing). The figure shows these two parts (along with the eye, which is the fluid inlet into the volute).

The centrifugal pump's function is as simple as its design. It is filled with liquid and the impeller is rotated (usually with an electric motor). The impeller vanes grab the liquid and "fling" it, giving it energy, which causes it to exit the impeller's vanes at a greater velocity than it possessed when it entered.

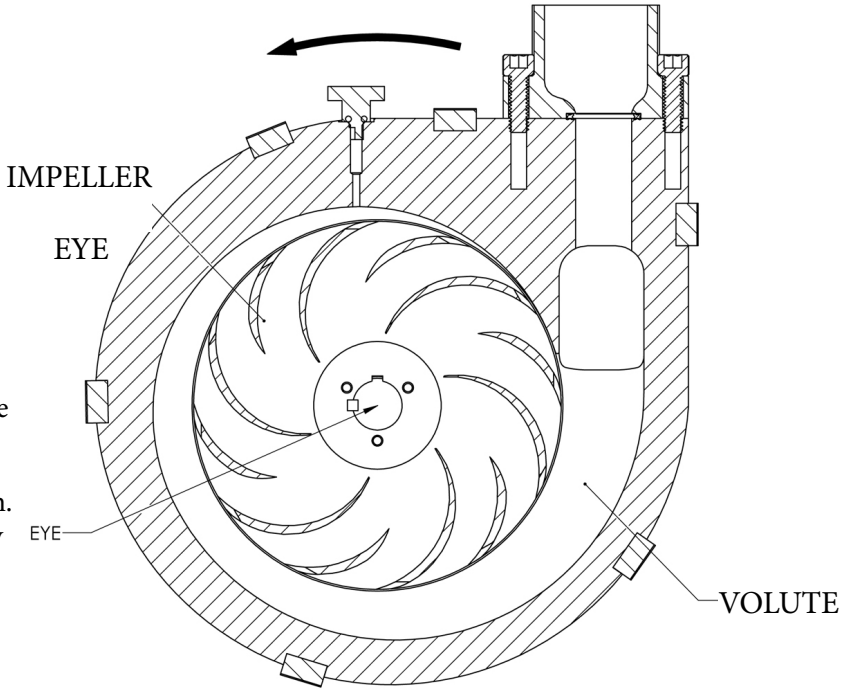
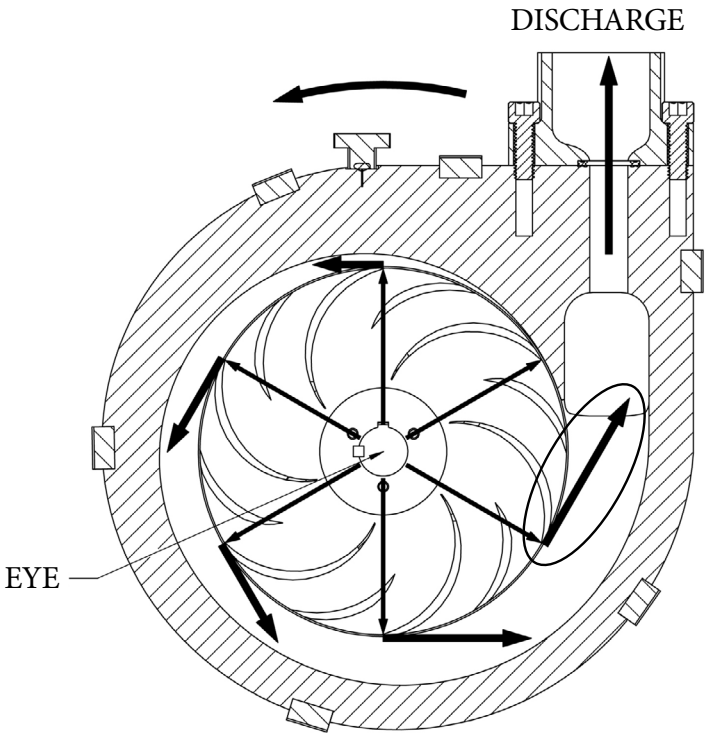


Figure 1.1: Pump Primary Components

Section 1: Define Centrifugal Pumping



This outward flow (flinging) reduces the pressure at the impeller eye (the center where the inlet piping allows the fluid to enter the casing), allowing more liquid to enter. The liquid exits the impeller at its edge (peripheral velocity) and is collected in the casing (volute) where its velocity is converted to pressure before it leaves the pump's discharge.

Figure 1.2: Impeller Outward Flow

What does a real centrifugal pump look like?

Figure 1.3: Commercial Centrifugal Pump

What does a real centrifugal impeller look like?

Figure 1.4: Centrifugal Pump Impeller

Section 1: Define Centrifugal Pumping

Focus 1: Let's Fling Stones!

(How gravity and velocity relate to centrifugal pumping)

Gravity is one of the more important forces that a centrifugal pump must overcome. You will find that the relationship between final velocity, due to gravity, and initial velocity, due to impeller speed, is a very useful one.

Let's use an example that's easy to visualize to better understand what we're talking about.

If a stone is dropped from the top of a building, its speed (velocity) will increase at a known rate of 32.2 feet per second for each second that it falls. This increase in velocity is known as acceleration due to gravity.

If we ignore the effect of air resistance on the falling stone, we can predict the velocity it will hit the ground based upon its starting height off the ground and the effect of acceleration due to gravity (that's 32.2 feet per second PER second).

Now, with help of mathematics and even Sir Isaac Newton himself (you remember that he discovered this thing called gravity), a convenient equation was developed which describes the relationship of velocity, height, and gravity as it applies to a falling body, like that stone as:

$$v^2 = 2gh$$

Where:

v = the velocity of the body in ft/sec

g = the acceleration due to gravity @ 32.2 ft/sec/sec (or ft/sec^2)

h = the distance through which the body falls

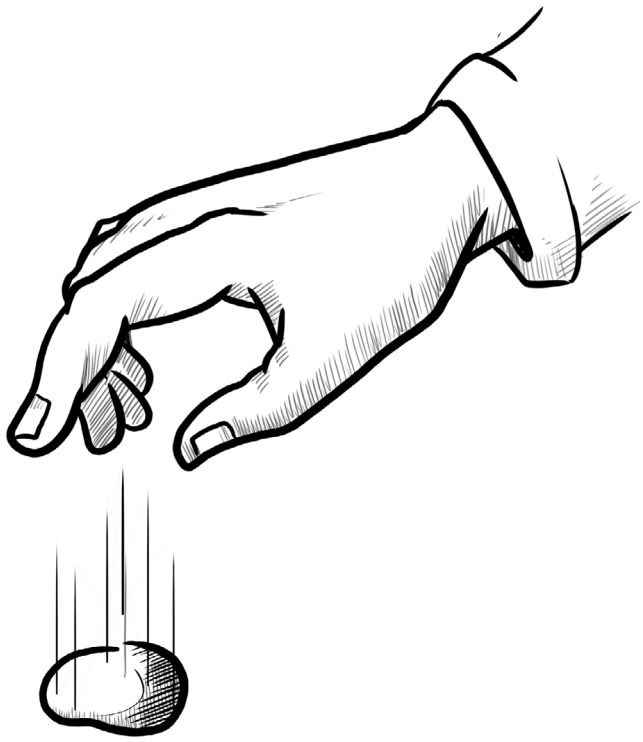
For example, if a stone is dropped from a building 100 feet high:

$v^2 = 2 \times 32.2 \frac{ft}{s^2} \times 100\,ft$, which gives us...

$$v^2 = 6440 \frac{ft^2}{s^2}$$

$$v = \sqrt{6440 \frac{ft^2}{s^2}} = 80.3 \frac{ft}{s}$$

The stone, therefore, will strike the ground at a velocity of 80.3 feet per second (about 55 miles per hour).



Section 1: Define Centrifugal Pumping

Now, let's reverse this scenario. That same equation allows us to figure out how fast we would have to throw that stone (initial velocity) upward from ground level for it to reach the top of the building (100 feet). This is true because the **final velocity** of a falling body **happens to be equal** to the **initial velocity** required to launch it to get it to that same height from which it fell (got that?). Using the example above, the initial velocity required to throw the stone to a height of 100 feet is 80.3 feet per second, the same as its final velocity.

So what does dropping and throwing stones have to do with pumping? Here's the cool part: the same equation we used for the stone applies when pumping water with a centrifugal pump. The velocity of the water as it leaves the impeller determines the head developed. In other words the water is "thrown" to a certain height. To reach this height it must start with the same velocity it would attain if it fell from that height.

So, if we rearrange the falling body equation we get:

$$h = \frac{v^2}{2g}$$

Now we can determine the height to which a body (or water) will rise given a particular initial velocity. For example, at 10 ft per sec:

$$h = \frac{\left(10\frac{ft}{s}\right)^2}{2 \times 32.2\frac{ft}{s^2}}$$

$$h = 1.55\,ft$$

Skill Builder Section 1, Focus 1

1. Calculate the velocity of a marble at impact if it is dropped from a 6 story building (each story is 12 ft. high). Disregard any air resistance.

2. At what speed would we need to launch the marble from the ground level to reach the top of the building?

3. If you were to try this with several different initial velocities, you would find out that there is an interesting relationship between the height achieved by a body and its initial velocity. **This relationship is one of the Affinity Laws of centrifugal pumps which can be studied independently.**

Section 1: Define Centrifugal Pumping

As a finale to this section, let's apply what we have learned to a practical application. Follow each step carefully and you will gain a good understanding of how this works (you'll feel so smart).

Problem: We have a centrifugal pump that is spinning at 1800 RPM. We need to figure out what impeller diameter would be needed to fling the water up 200 feet (a.k.a. Develop a head of 200 feet).

First we must calculate the initial velocity required to develop a head of 200 feet. Using our falling body equation from before:

$$v^2 = 2gh$$

$$v^2 = 2 \times 32.2 \frac{ft}{s^2} \times 200\,ft$$

$$v^2 = 12{,}880 \frac{ft^2}{s^2}$$

$$v = \sqrt{12{,}880 \frac{ft^2}{s^2}} = 113 \frac{ft}{s}$$

We also need to know the number of rotations the impeller undergoes each second:

1800 RPM / 60 sec = 30 RPS (Revolutions per Second).

Now we can compute the number of feet a point on the impellers rim travels in a single rotation: 113 ft/sec divided by 30 rotations/sec = 3.77 ft/rotation

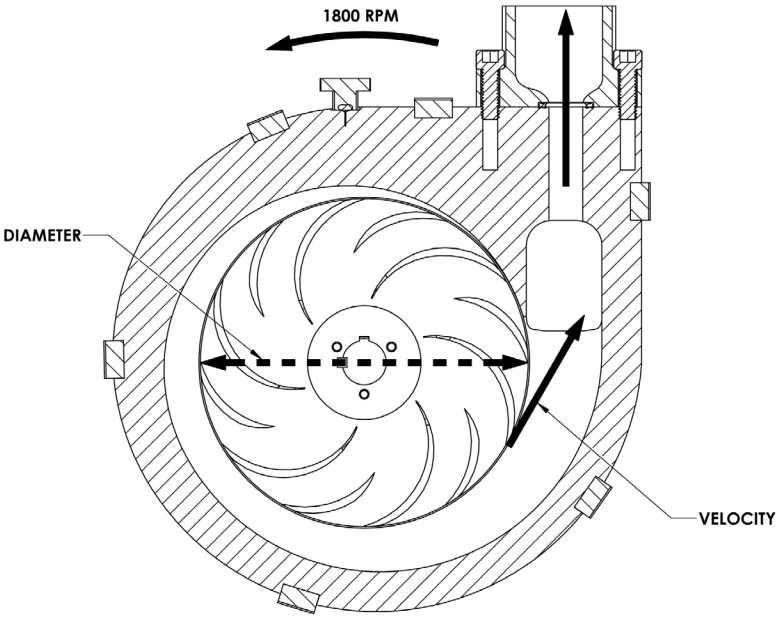
Since distance traveled per rotation is **the same** as the circumference of the impeller we can compute the diameter using the circular relation formula as follows:

$$C = \pi D \Rightarrow D = \frac{C}{\pi}$$

$$D = \frac{3.77\,ft}{\pi}$$

$$D = 1.2\,ft = 14.4\,in$$

Therefore an impeller with a diameter of approximately 14.4" turning at 1800 RPM will produce a head of (fling water up) 200 Feet.



Skill Builder

Section 1, Focus 2

We have a centrifugal pump that is spinning at 1800 RPM. What impeller diameter would be needed to fling the water up 122 feet (a.k.a. Develop a head of 122 feet). _____

Focus 3: The Ziegler-Nichols Method

The Ziegler-Nichols method is another popular method of tuning a PID controller. It is very similar to the trial and error method wherein I and D are set to zero and P is increased until the loop starts to oscillate. Once oscillation starts, the ultimate gain K_u and the period of oscillations T_u are noted. The P, I and D are then adjusted as per the tabular column shown below.

Ziegler-Nichols Method			
Control Type	K_p	K_i	K_d
PID	$0.6\ K_u$	$2/T_u$	$0.125\ T_u$

Table 7.1- Ziegler-Nichols tuning, using the oscillation method

Focus 4: PID Tuning Software

Modern process facilities, especially with complex processes, can also utilize PID tuning and loop optimization software to ensure consistent results. These software packages will gather the data, develop process models of the system and suggest optimal tuning. There are a variety of programs available, including custom application developments.

Section 7

1. Can you name three methods of applying gain to a PID process?
2. Can you just use P&I in a control system to get adequate control?

Knowledge Certification Quiz: Lesson 7

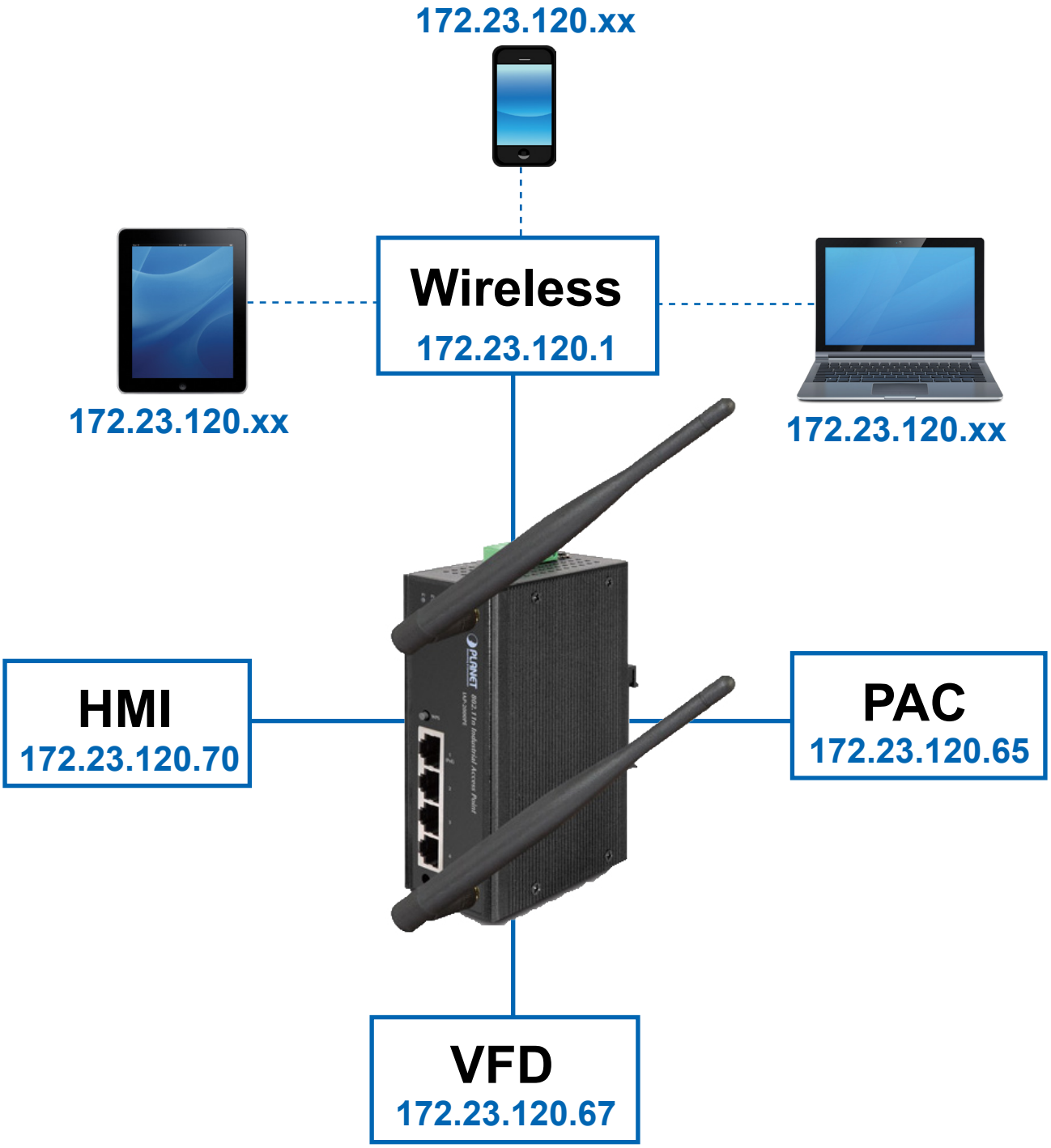
1. *What is gain?*
2. *What is the process called of setting optimal gains to get the desired control responses?*
3. *What does gain improve for each element of the control algorithm*

I certify that I have answered all certification quiz questions correctly and am ready for the next lesson.

_____ _____

Your Signature Date

Unit 3 - Ethernet Communication



Chapter 1 - Introduction to Industrial Ethernet

Section 1 – What is Ethernet?
- **IP**
- **TCP**
- **UDP**
- **IP Addressing**

Section 2 – Switched Ethernet
- **Basic Switches**
- **Managed Switches**
- **Intelligent Switches**

Manufacturing companies are increasingly expanding their global operations to address new opportunities and reduce operating costs. They are also seeking to continuously improve efficiency and drive down costs for existing facilities and processes. Achieving the goals of globalization and operations excellence requires improving connectivity between plant and business systems for real-time visibility to information and effective collaboration. This helps to assure consistent quality and performance across global operations, and to reduce the cost of design, deployment, and support of distributed manufacturing and IT systems. Manufacturers must be able to balance production with demand to optimize material usage and asset utilization, while continuing to meet increasingly exacting customer requirements and metrics for on-time delivery. Manufacturers also need to improve response to events that occur on the plant floor, regardless of location, while implementing more flexible and agile operations in order to react to rapidly changing market conditions.

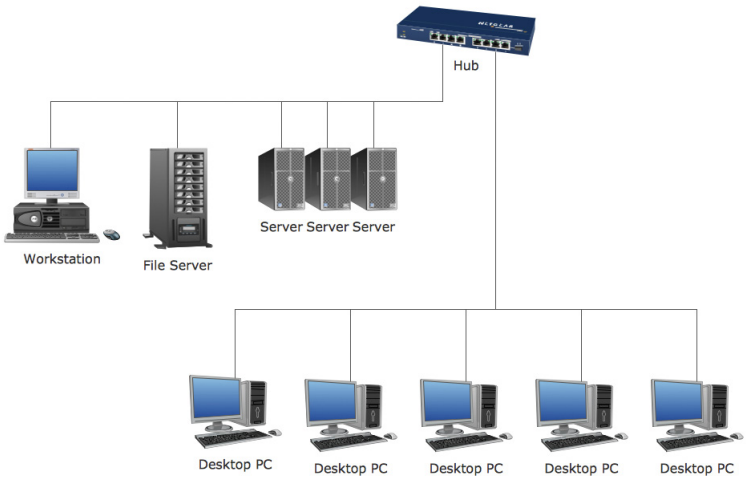
As manufacturers seek to improve processes many are turning to Ethernet technology on the factory floor. This migration is rapidly gaining momentum. To deploy this technology, engineers on the manufacturing floor should be familiar with some of the important concepts behind Industrial Ethernet.

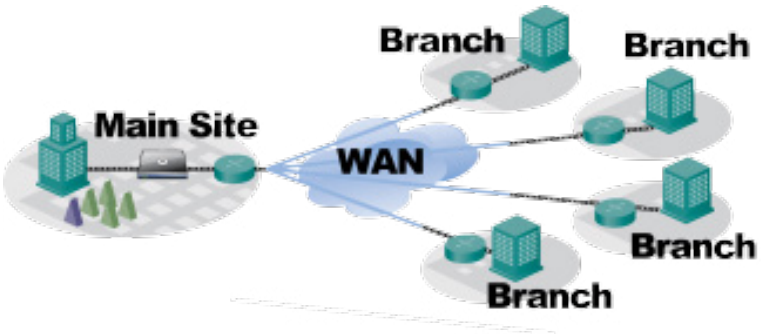
Section 1: What is Ethernet?

Focus 1: Types of Ethernet

Local Area Networks (LANs)

A network is any collection of independent computers that exchange information with each other over a shared communication medium. Local Area Networks or LANs are usually confined to a limited geographic area, such as a single building or a college campus. LANs can be small, linking as few as three computers, but can often link hundreds of computers used by thousands of people. The development of standard networking protocols and media has resulted in worldwide proliferation of LANs throughout business and educational organizations.





Wide Area Networks (WANs)

Often elements of a network are widely separated physically. Wide area networking combines multiple LANs that are geographically separate. This is accomplished by connecting the several LANs with dedicated leased lines such as a T1 or a T3, by dial-up phone lines (both synchronous and asynchronous), by satellite links and by data packet carrier services. WANs can be as simple as a modem and a remote access server for employees to dial into, or it can be as complex as hundreds of branch offices globally linked. Special routing protocols and filters minimize the expense of sending data over vast distances.

Wireless Local Area Networks (WLANs)

Wireless LANs, or WLANs, use radio frequency (RF) technology to transmit and receive data over the air. This minimizes the need for wired connections. WLANs give users mobility as they allow connection to a local area network without having to be physically connected by a cable. This freedom means users can access shared resources without looking for a place to plug in cables, provided that their terminals are mobile and within the designated network coverage area. With mobility, WLANs give flexibility and increased productivity, appealing to both entrepreneurs and to home users. WLANs may also enable network administrators to connect devices that may be physically difficult to reach with a cable.

Focus 2: Types of LAN Technology

Ethernet is by far the most widely used LAN technology today. It is estimated that more than 85 percent of the world's LAN connected PCs and workstations use Ethernet. Ethernet refers to the family of computer networking technologies covered by the IEEE 802.3 standard, and can run over both optical fiber and twisted-pair cables. Over the years, Ethernet has steadily evolved to provide additional performance and network intelligence. This continual improvement has made Ethernet an excellent solution for industrial applications. Today, the technology can provide four data rates.

- 10BASE-T Ethernet delivers performance of up to 10 Mbps over twisted-pair copper cable.

- Fast Ethernet delivers a speed increase of 10 times the 10BASE-T Ethernet specification (100 Mbps) while retaining many of Ethernet's technical specifications. These similarities enable organizations to use 10BASE-T applications and network management tools on Fast Ethernet networks.

- Gigabit Ethernet extends the Ethernet protocol even further, increasing speed tenfold over Fast Ethernet to 1000 Mbps, or 1 Gbps. Because it is based upon the current Ethernet standard and compatible with the installed base of Ethernet and Fast Ethernet switches and routers, network managers can support Gigabit Ethernet without needing to retrain or learn a new technology.

- 10 Gigabit Ethernet, ratified as a standard in June 2002, is an even faster version of Ethernet. Because 10 Gigabit Ethernet is a type of Ethernet, it can support intelligent Ethernet-based network services, inter-operate with existing architectures, and minimize users' learning curves.

To date, more than 300 million switched Ethernet ports have been installed worldwide. Ethernet technology has a wide acceptance because it is easy to understand, deploy, manage, and maintain. Ethernet is low-cost and flexible, and supports a variety of network topologies. Compared to traditional, non-Ethernet-based industrial solutions that have a data rate of between 500 Kbps to 12 Mbps, Ethernet technology can deliver substantially higher performance. It is based on industry standards, and it can connect over any Ethernet-compliant device from any vendor.

Focus 3: IP (Internet Protocol)

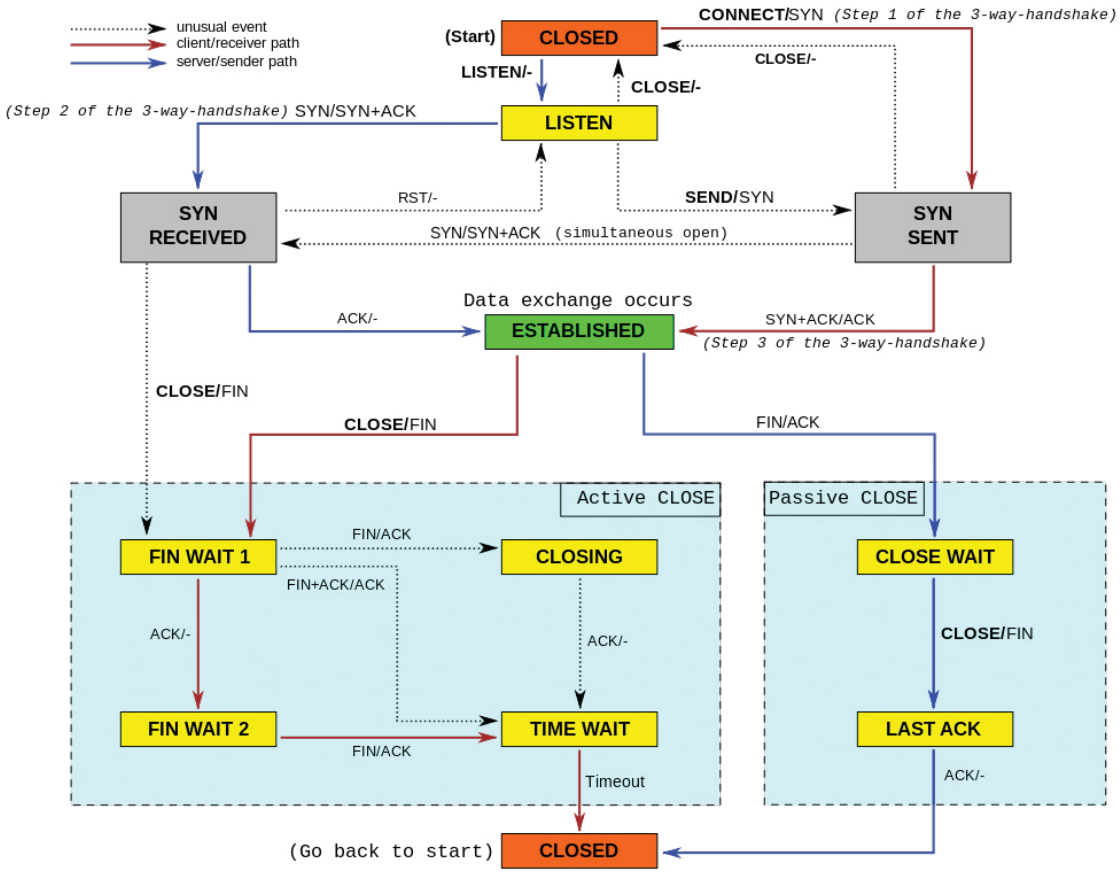
IP is often understood as the Internet Protocol suite: A suite of protocols and standards on which the Internet and most enterprise networks are based. The IP suite includes not only lower-level specifications, such as TCP and IP, but specifications for such common applications as e-mail, terminal emulation, and file transfer. The most relevant of these to the factory floor though are IP itself, TCP, and UDP. These three protocols communicate in collaboration to form what is known as the Internet. This technology is moving automation forward, giving operators the ability to access and control plants remotely.

IP is the common and primary network (Layer 3) protocol in the Internet suite. IP represents the core of the Internet Protocol suite. It defines an address by which the network can transmit the packet from source to destination—even across LANs. This is opposed to the MAC addresses, which are used to network locally, within a LAN. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams or packets through a network; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes. More on IP address allocation and administration is found in the network management section.

Focus 4: TCP (Transmission Control Protocol)

TCP provides reliable, in-order delivery of packets between two devices. It relies upon IP. TCP applications establish connections between one another over which they send packets. TCP is a stateful protocol; it maintains the state after the packet is sent. TCP checks whether all packets have arrived and can request re-transmission if a packet is dropped, lost, or corrupted during transmission. Due to this overhead though, TCP is not always ideal for realtime applications. TCP has emerged as the dominant protocol used for the bulk of Internet connectivity. Its services break large data sets into individual packets, checking for and resending lost packets and reassembling packets into the correct sequence. But these additional services come at a cost in terms of additional data overhead, and delays called latency.

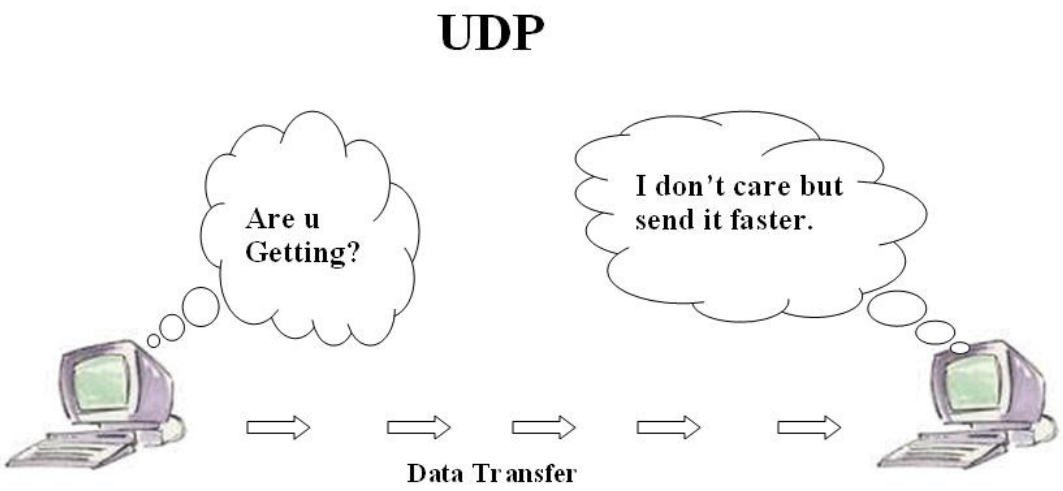
TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, and manages flow control. It is meant to provide error-free data transmission. If any data packets get dropped or garbled, they will re-transmit until acknowledgment that all packets arrived sucessfully. This error-free feature requires a complicated routine. Below is a diagram showing the TCP routine.



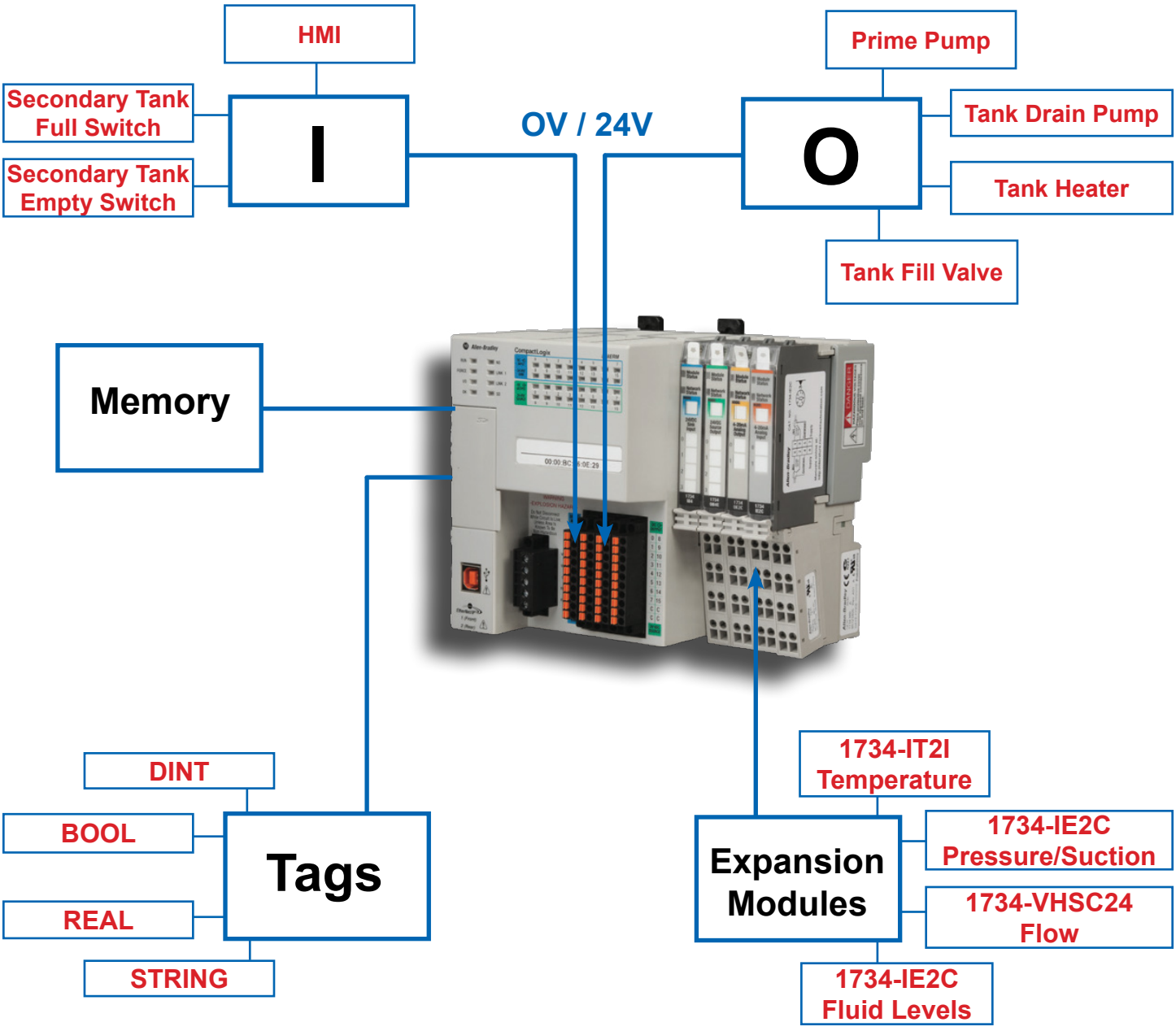
Focus 5: UDP (User Datagram Protocol)

UDP is often used for real-time communications such as voice and I/O traffic. UDP also relies on IP. UDP does not guarantee delivery or the order of the packets, thus simplifying the protocol. It has much lower bandwidth overhead and latency. The applications would rather drop a packet than receive it late. UDP is considered a “stateless” protocol. It is compatible with packet broadcast (sending to all on local network) and multicasting (sending to all subscribers). UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications, which can suffer some data loss without adversely affecting perceived quality. In some cases, forward error correction techniques are used to improve audio and video quality in spite of some loss.

UDP can also be used in applications that require lossless data transmission when the application is configured to manage the process of retransmitting lost packets and correctly arranging received packets. This approach can help to improve the data transfer rate of large files compared with TCP. The main goal for UDP transmission is send it as fast as possible. The diagram below summarizes UDP transmission:



Unit 4 - Programmable Automation Controller



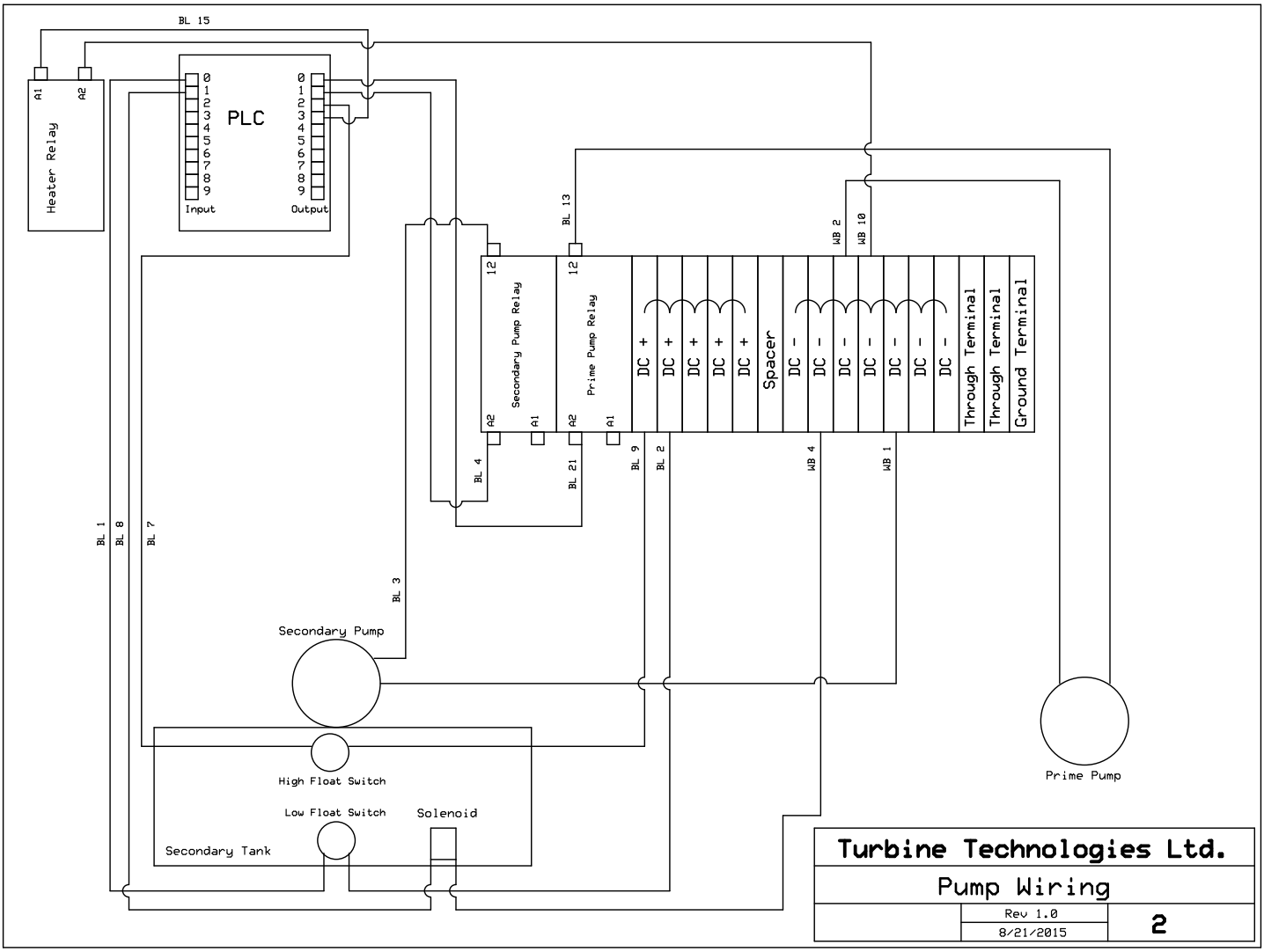
Chapter 2 - Embedded Input / Output

The PAC has 16 embedded inputs. Fluid*Mechatronix*™ uses 2 inputs for the following devices:

- Drain Pump Float Switch
- Solenoid Valve Float Switch

The PAC has 16 embedded outputs. Fluid*Mechatronix*™ uses 4 outputs for the following devices:

- Prime Pump
- Tank Heater
- Secondary Tank Drain Pump
- Secondary Tank Solenoid



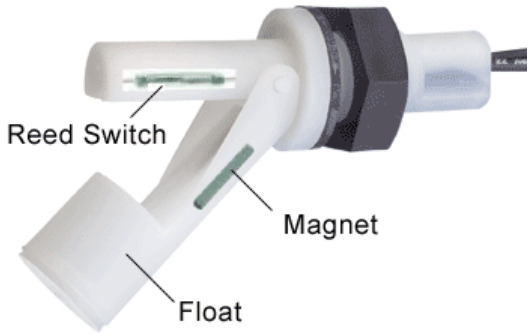
Section 1: Inputs

Focus 1: Drain Pump Float Switch

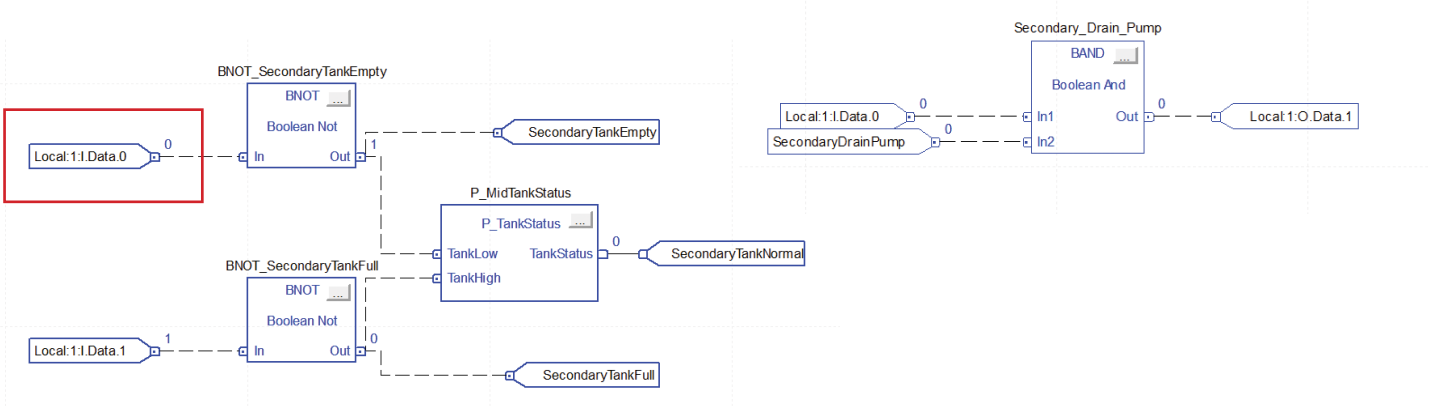
In order to fully understand the FluidMechatronix™ tank levels, we must first learn how a float switch operates. The purpose of a float switch is to open or close a circuit as the level of a liquid rises or falls. Most float switches are “normally closed,” meaning the two wires coming from the top of the switch complete a circuit when the float is at its low point, resting on its bottom clip (for example, when a tank is dry).

To complete a circuit, float switches utilize a magnetic reed switch, which consists of two contacts sealed in a glass tube. When a magnet comes close to the two contacts, they become attracted to each other and touch, allowing current to pass through. When the magnet moves away, the contacts demagnetize and separate (breaking the circuit), as shown below.

Horizontal Type:



FluidMechatronix™ monitors the status of the drain pump float switch on **Local:1:I.Data.0**. The float switch serves two purposes. First, to monitor the low level of the fluid in the tank. Second, to de-energize the drain pump coil in the event that the water levels cause the float to drop. The data is stored as a tag and is monitored through software. The data is stored as a boolean value (0 or 1) and is represented in the tag library by BOOL. Recall binary numbers from Chapter 3, the byte values for 0 and 1 would be 00000000 and 00000001 respectively. To fully understand the circuits below, we need to investigate the BNOT and BAND further.

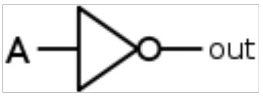


Section 1: Inputs

BNOT and BAND are special types of logic used in Studio 5000. This logic can be represented with software (in our case), or integrated circuit (IC) chips. Traditionally, these gates were represented as NOT and AND gates.

A NOT gate is the most basic type of gate. There are only 2 states for a NOT gate. When the input is 0 to a NOT gate, the output is NOT 0 so it has to be 1. Similalry, when the input to a NOT gate is 1, the output is NOT 1 so it is a 0. The table on the right shows the typical diagram and logic table for a NOT gate. The BNOT gate that is used in the Studio 5000 software is nothing more than a NOT gate that has the ability to use any data with a BOOL tag.

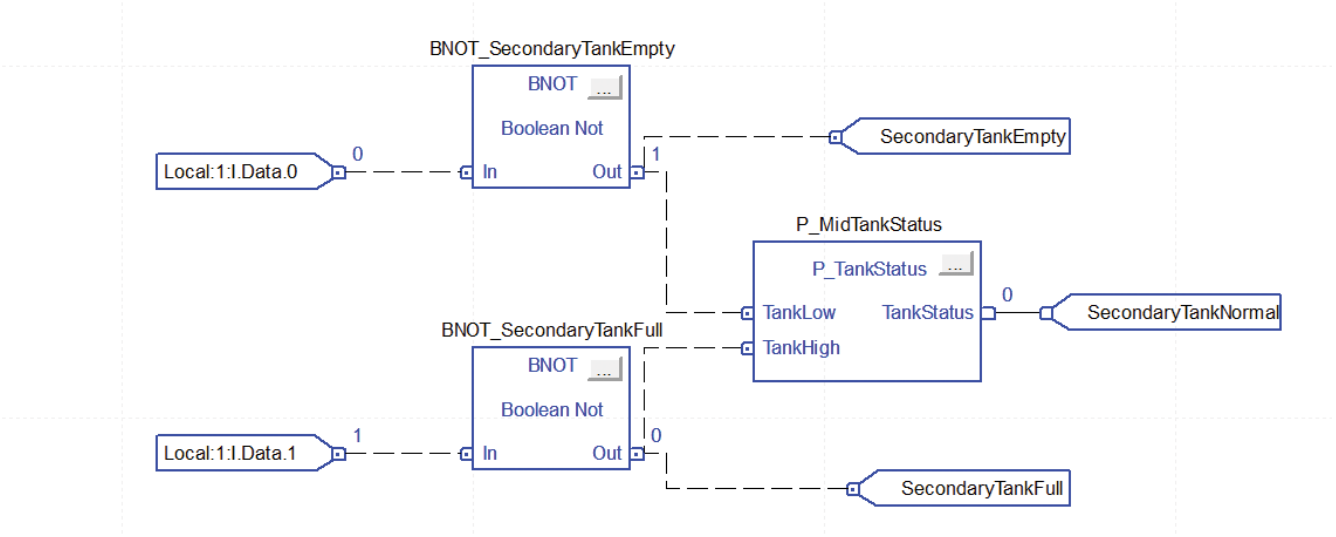
INPUT	OUTPUT
A	NOT A
0	1
1	0



An AND gate is another fairly simple gate that has 4 states. This type of gate is slightly different because it has 2 channels of input. When A and B are both zero, the output of the AND gate is 0. When A is 0 and B is 1, the output of the AND gate is 0. When A is 1 and B is 0, the output of the AND gate is 0. Finally, when A is 1 and B is 1, the output of the AND gate is 1. Both A and B have to be 1 in order for the AND gate to output a 1. Any combinations that have 0 at the input, the output will always be 0 on an AND gate. The BAND gate that is used in the Studio 5000 software is nothing more than a AND gate that has the ability to use any data with a BOOL tag.

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

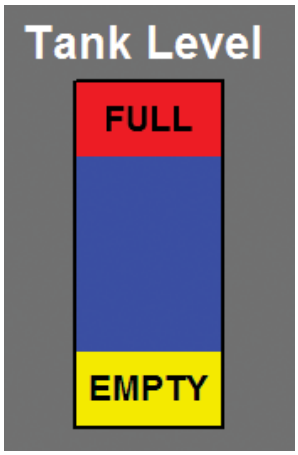
Section 1: Inputs



Now we have enough information to fully understand the Drain Pump Float schematic used by Studio 5000. There are two states to the float switch:

1. When the water level in the secondary tank is above the switch, the float switch is closed and the PAC will see the tag Local:1:I:Data.0 = 1. In this case the tank has enough water.

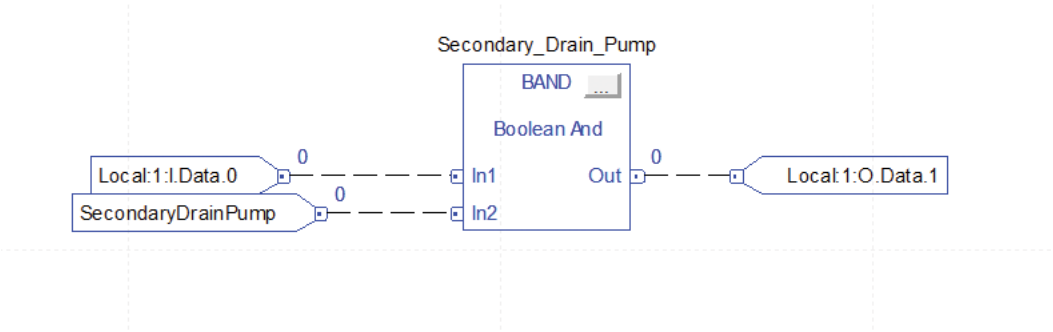
2. When the water level in the secondary tank is below the switch, the float switch is opened and the PAC will see the tag Local:1:I:Data.0 = 0. In this case the tank is empty.



We need a tank alarm to tell us when the secondary tank is empty. Looking at state number 2 above, the value is zero when the tank is empty. Since we want the alarm to be represented by a 1 (alarm on), we need to send the data through a BNOT to reverse the signals. The data is taken into the BNOT as a 0, so the output from the BNOT must be a 1. When SecondaryTankEmpty = 1 we can use this status to setup an alarm on the HMI screen.

When Local:1:I:Data.0 = 1, there is water in the tank and the output of the BNOT will read 0. So the tank empty alarm status will be 0 and the alarm is off. Alarm configurations with FactoryTalk will be covered in Unit 5. For now, we are only concerned about the data in Studio 5000.

Section 1: Inputs



Now we can analyze the Drain Pump circuit that controls the pump. We want the operator to have pump control when there is enough water in the secondary tank. If there is isn't enough water in the tank, we want the PAC to control the pump and shut it off. The BAND gate is perfect for this situation because we require two elements to turn the pump on. We have water level indicated by the float switch tag on Local:1:I.Data.0. We have an operator button controlling the status of the tag SecondaryDrainPump. There are four states for this dual input BAND gate:

1. The float switch is open (0) and the operator switch is off (0). The output of the BAND is 0 and the pump stays off.
2. The float switch is open (0) and the operator switch is on (1). The operator wants to turn the pump on, but there is not water. The output of the BAND remains 0 and the pump stays off.
3. The float switch is closed (1) and the operator switch is off (0). There is enough for the pump to run, but the operator doesn't want the pump on. The output of the BAND is still 0 and the pump stays off.
4. The float switch is closed (1) and the operator switch is on (1). There is enough water in the tank and the operator wants to turn the pump on. The output of the BAND is 1 and the pump will turn on.

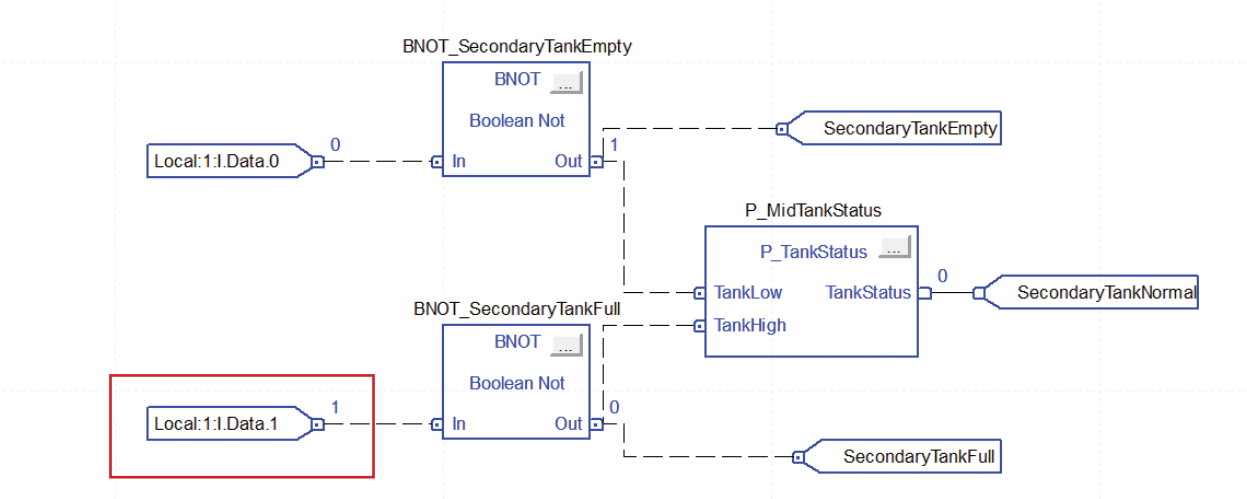
Section 1, Focus 1-3

1. What state is the lower float switch in when the tank has enough water?

2. What type of logic can invert binary data?

3. Describe a situation where a AND gate could be used?

Focus 2: Solenoid Valve Float Switch



FLUID*Mechatronix*™ monitors the status of the drain pump float switch on **Local:1:I.Data.1**. The float switch serves two purposes. First, to monitor the high level of the fluid in the tank. Second, to de-energize the solenoid valve coil in the event that the water level gets too high. The data is stored as a tag and is monitored through software as a BOOL (boolean). This float switch is oriented opposite the other switch, so when the float is up the switch is open. There are two states to the float switch:

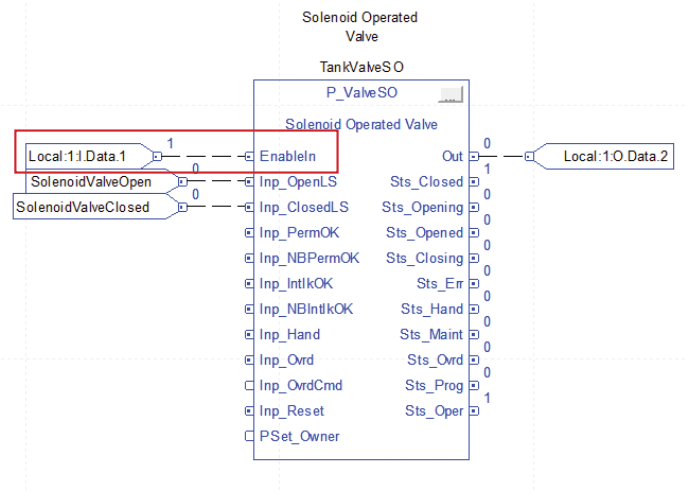
1. When the water level in the secondary tank is above the switch, the float switch is opened and the PAC will see the tag **Local:1:I:Data.1 = 0**. In this case the tank has too much water.

2. When the water level in the secondary tank is below the switch, the float switch is closed and the PAC will see the tag **Local:1:I:Data.0 = 1**. In this case the water in the tank is below the switch.

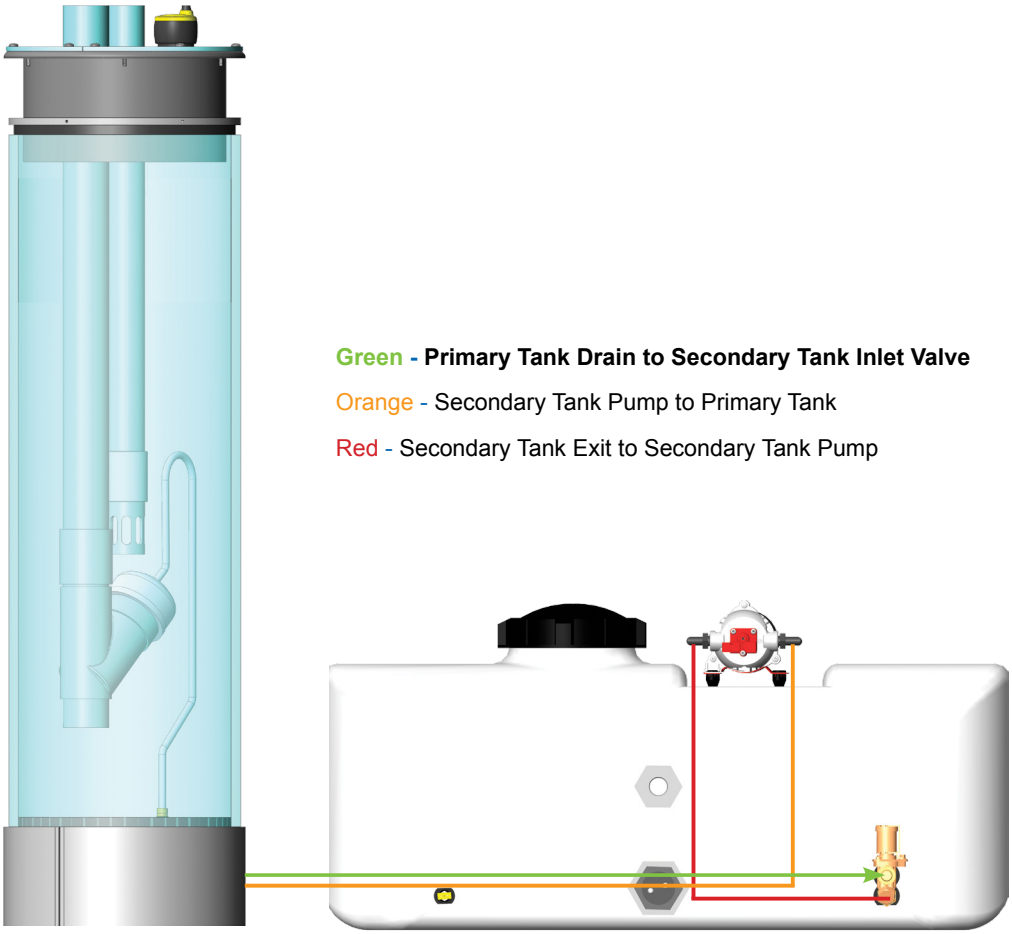
We need a tank alarm to tell us when the secondary tank has too much water. Looking at state number 1 above, the value is zero when the tank has too much water. Since we want the alarm to be represented by a 1 (alarm on), we need to send the data through a BNOT to reverse the signals. The data is taken into the BNOT as a 0, so the output from the BNOT must be a 1. When SecondaryTankFull = 1 we can use this status to setup an alarm on the HMI screen.

When Local:1:I:Data.0 = 1, there is water in the tank and the output of the BNOT will read 0. So the tank full alarm status will be 0 and the alarm is off. Alarm configurations with FactoryTalk will be covered in Unit 5. For now, we are only concerned about the data in Studio 5000.

Section 1: Inputs



Now we can analyze the Solenoid Valve circuit that controls the solenoid. In this case, we don't need to use a BAND gate because the valve is controller with an Add On Instruction (AOI). The AOI operates like a logic gate, but also offers many extra features like EnableIn. Recall the state where **Local:1:I:Data.0 = 1.** The water level in the tank is below the full switch, so the solenoid valve can operate. We can send this tag value directly into the EnableIn on the AOI. When the value is 1, it allows the AOI solenoid control. The operator will request control of the valve from the HMI and that control will be permitted because the feature is enabled. When there is too much water and the tank is full, the value will be 0. When the EnableIn status of the AOI is 0, the AOI is disabled and the instruction will not execute. The operator can request valve control with the HMI, but the control is not allowed because the AOI is disabled. The AOI EnableIn simplifies the programming to enable and disable the solenoid.

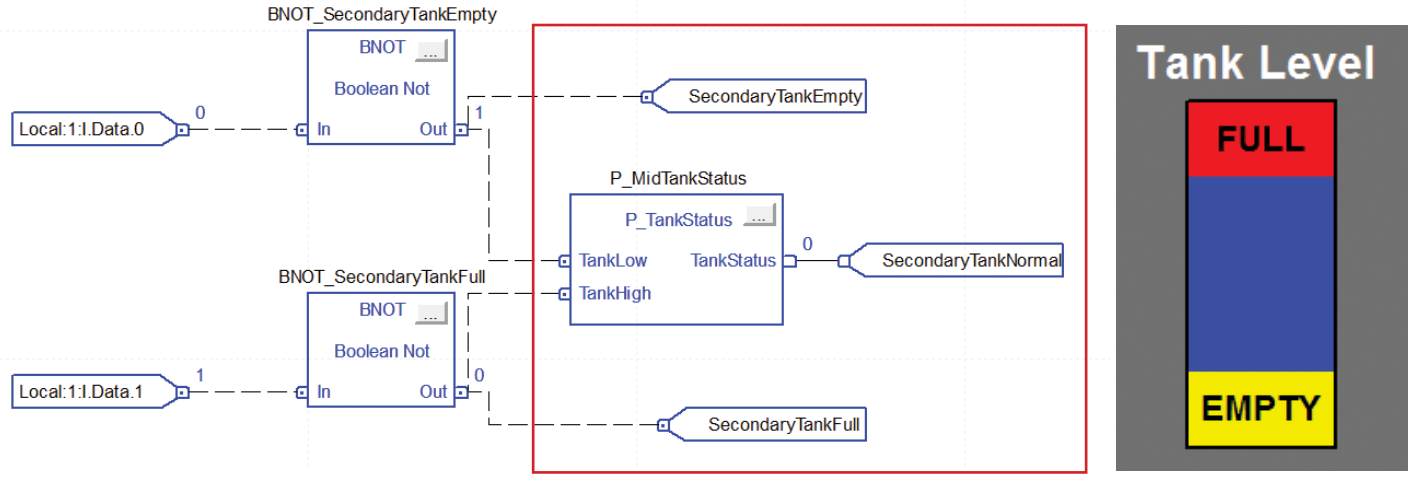


Green - **Primary Tank Drain to Secondary Tank Inlet Valve**

Orange - Secondary Tank Pump to Primary Tank

Red - Secondary Tank Exit to Secondary Tank Pump

Section 1: Inputs



Now we can analyze the P_TankStatus circuit that controls the status of the blue section on the tank level above. There is a custom AOI, built to control the fluid visibility on the HMI. This custom add-on was programmed with Structured Text. We created 2 boolean tags for low and high, and one INT tag for the tank status. The text below represents the actual programming that runs the mid tank status. The blue indicator for mid tank level has 3 different output states. The structured text code below shows the states of the TankStatus:

```
If TankLow & NOT TankHigh Then
    TankStatus := 0;

ElsIf NOT TankLow & TankHigh Then
    TankStatus := 2;

ElsIf NOT TankLow & NOT TankHigh Then
    TankStatus := 1;

End_IF;
```

For the first line of code, TankLow = 1 and TankHigh = 0. We want the output for this case to be 0. This result is used to remove the blue and red area of the tank level on the HMI. We don't want any blue or red to show because the tank is empty. We will cover the HMI alarm connections in Unit 5.

For the second line of code, TankLow = 0 and TankHigh = 1. We want the output for this case to be 2. This result will display the blue section on the HMI, but not show the text "OK" in that section. This is because we will also be displaying the full alarm at the same time.

For the third line of code, TankLow = 0 and Tank High = 0. We want the output for this case to show the blue section with the text "OK" on the screen. This is very similar to the second line of code, but we will not have the full alarm on the screen.

Section 1: Inputs

Section 1, Focus 1-3

1. Describe how a float switch can be used as a control?

2. How is the EnableIn feature useful?

Knowledge Certification Quiz: Section 1

1. Develop a situation where a BNOT and AND could be used in conjunction to create a result.

2. Describe the two states of the EnableIn and give an example of each.

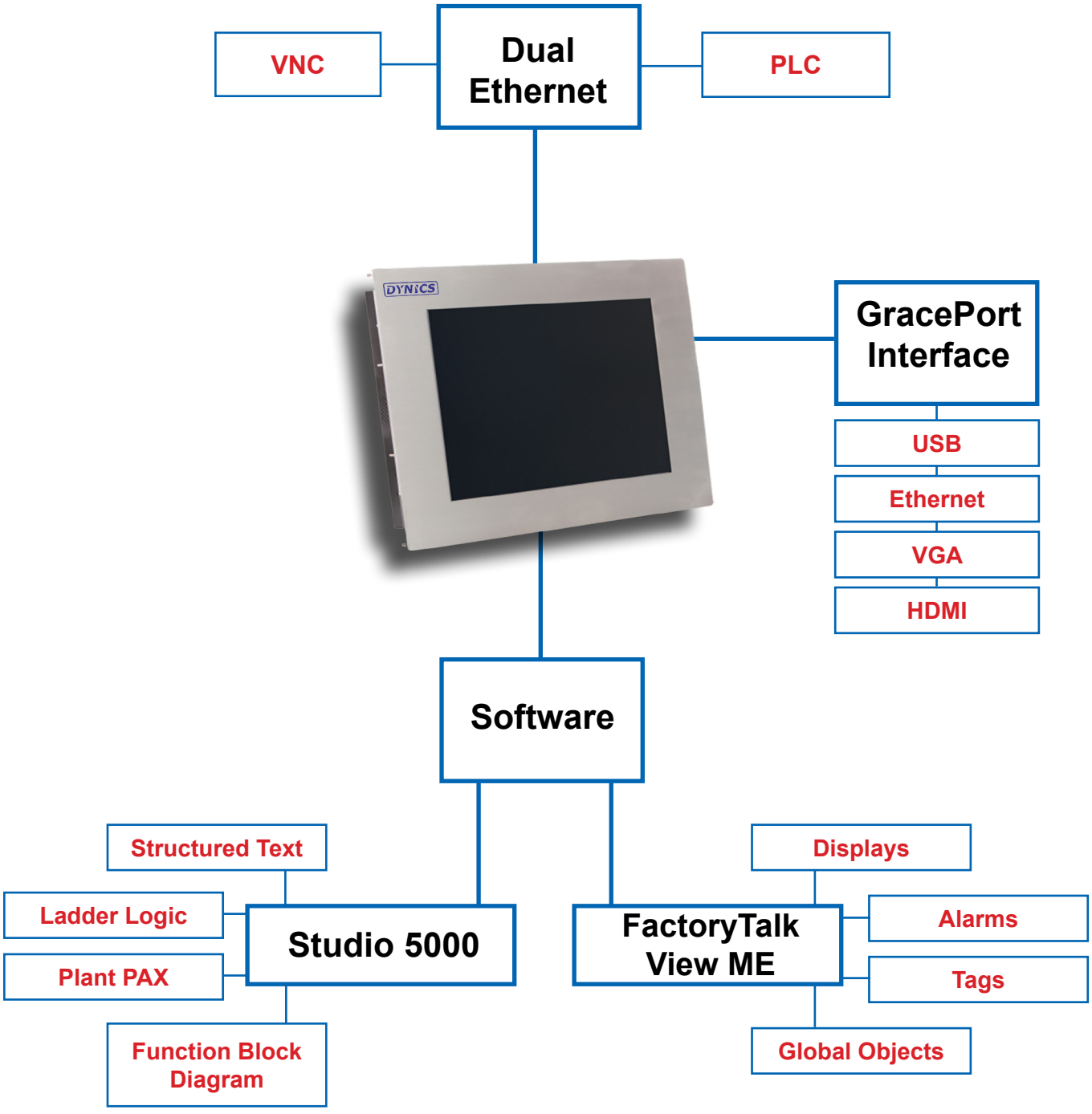
3. How could 4 float switches be used to show 4 different tank levels?

I certify that I have answered all certification quiz questions correctly and am ready for the next lesson.

_____ _____

Your Signature Date

Unit 5 - Human Machine Interface



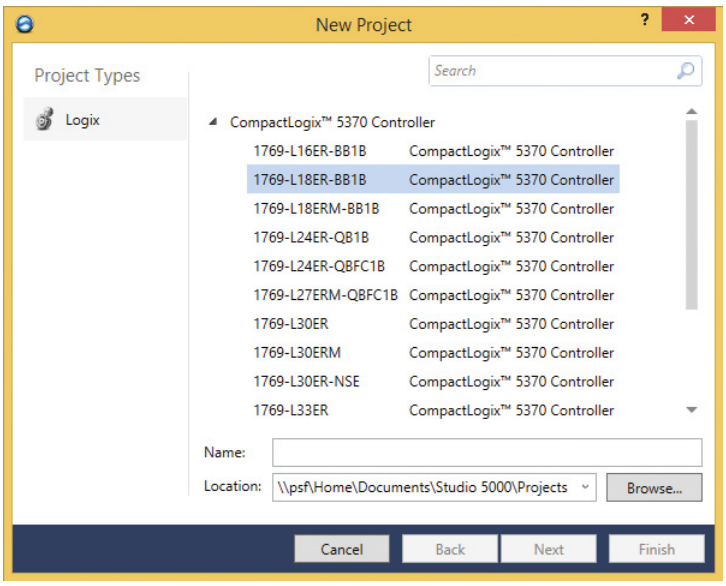
Section 1: Studio 5000 Logix Designer Programming Environment

Focus 1: Creating a New Project

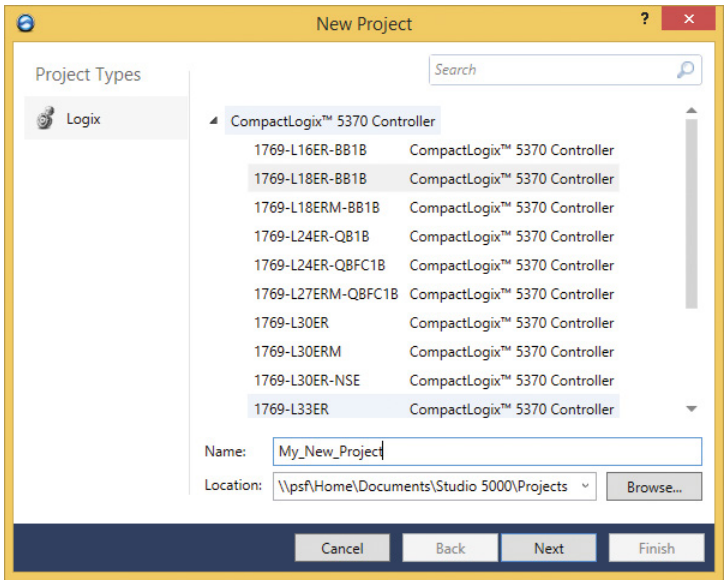
Double click the Studio 5000 icon in the dock

The Studio 5000 splash screen with open. You will see FLUID*Mechatronix*™ is a recent project. If you wish to open that project for viewing, click on it. Otherwise, click on **New Project**.

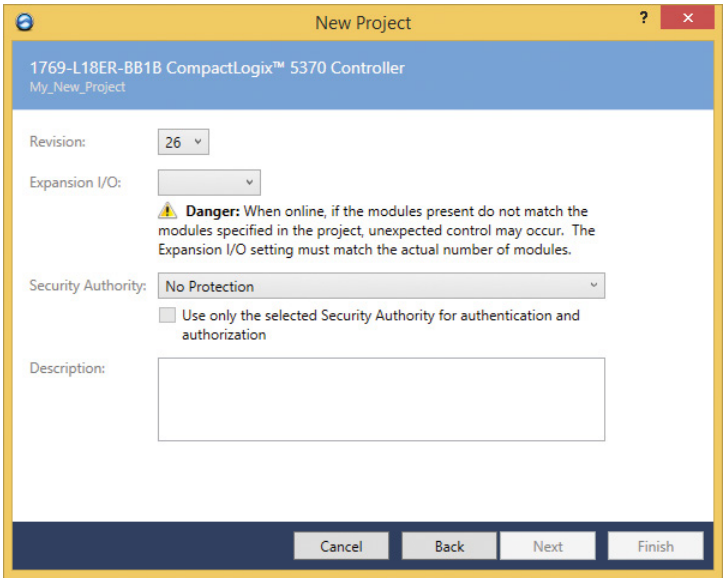
A new window will open. Click on Logix and navigate to the 1769-L18ER-BB1B controller.



Choose a name for the new project and click **Next**.

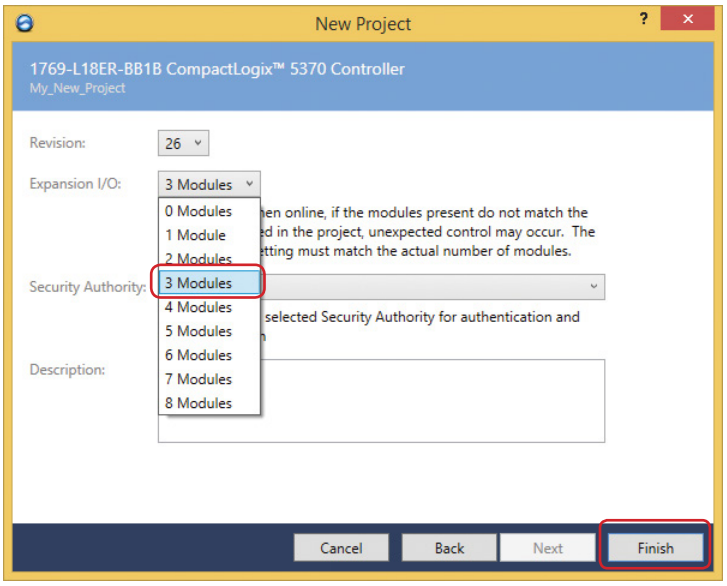


A new window will open with the New Project details. We can see the current software revision is v26.



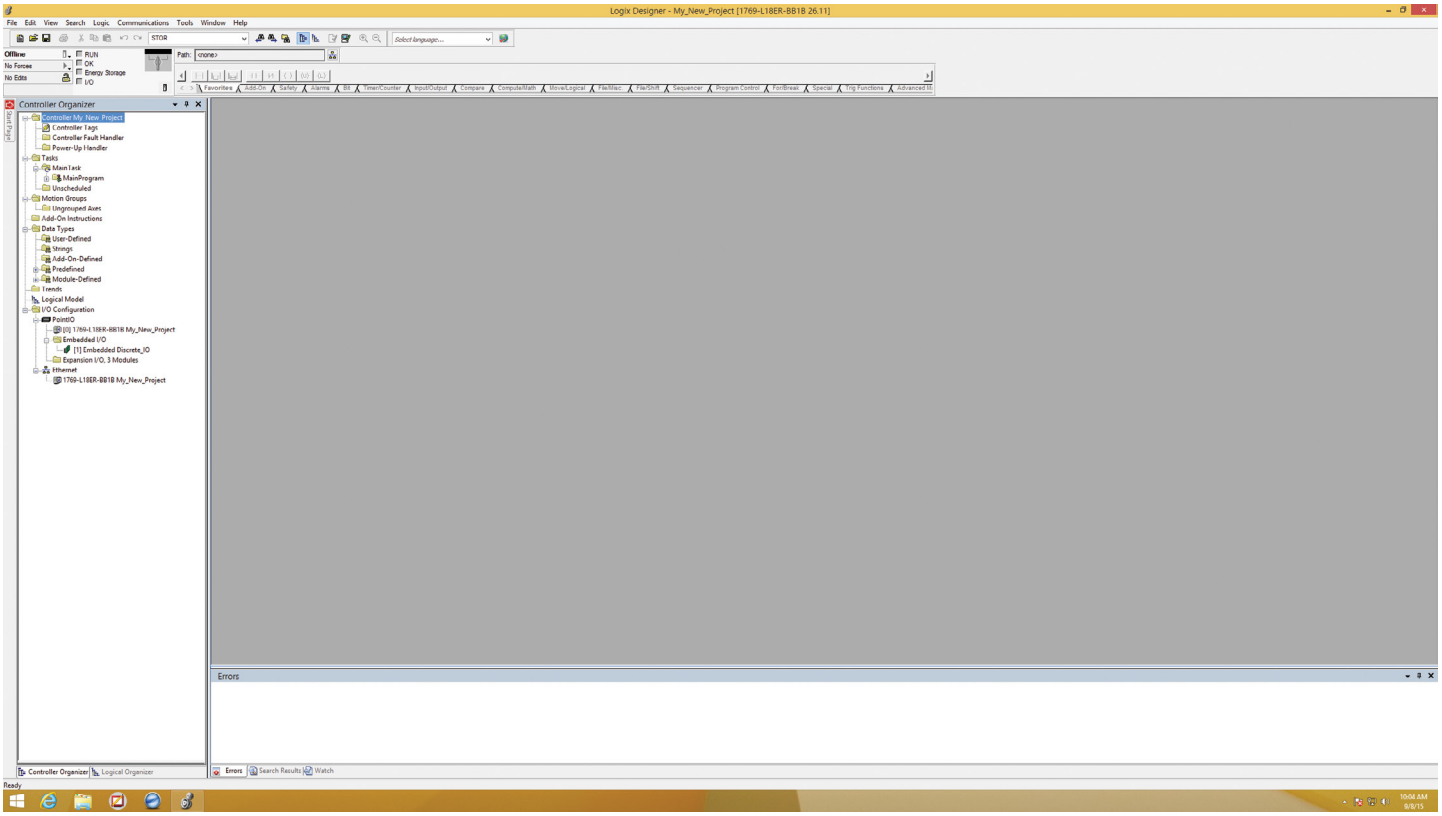
Section 1: Studio 5000 Logix Designer Programming Environment

Select the proper number of modules for the system. In this case, we have 3 expansion modules installed. Select **3 Modules**. If security is desired, select the type of security and click **Finish**. Note: the software already knows the total number of modules allowed for expansion is 8 modules.



The new project will begin loading and show the following screen.

The new project will open and display the main window.



The **Organizer Window** appears on the left side of the Studio 5000 window. There is a folder highlighted for Controller My_New_Project. There is no I/O, tag database, or logic associated with the controller.

Skill Builder

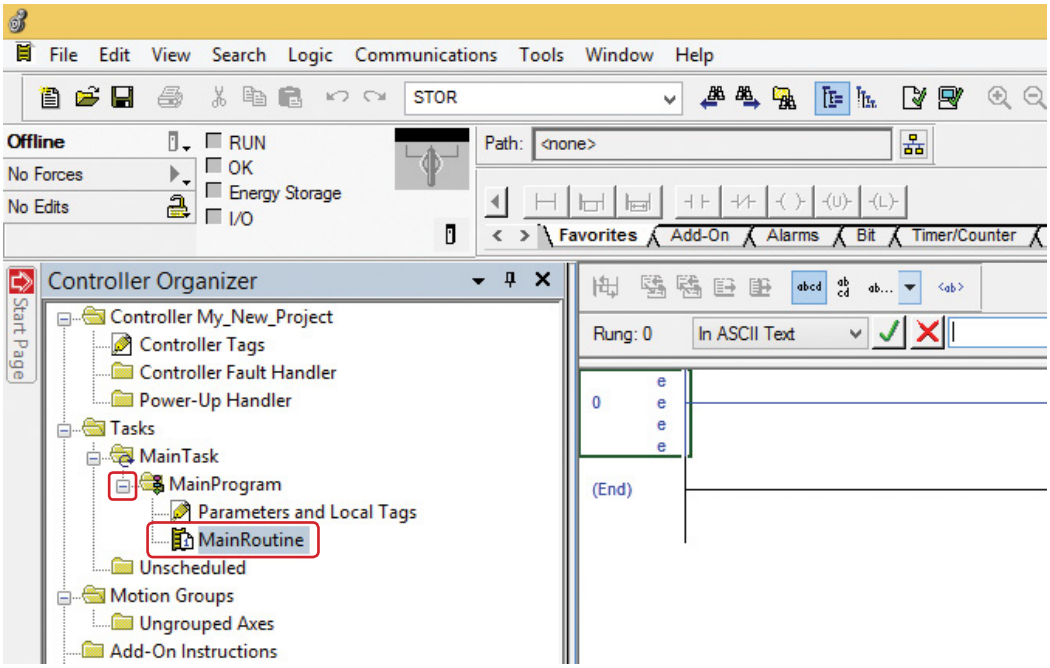
Section 1, Focus 1

1. Why is it critical to select the correct number of modules for a PAC?

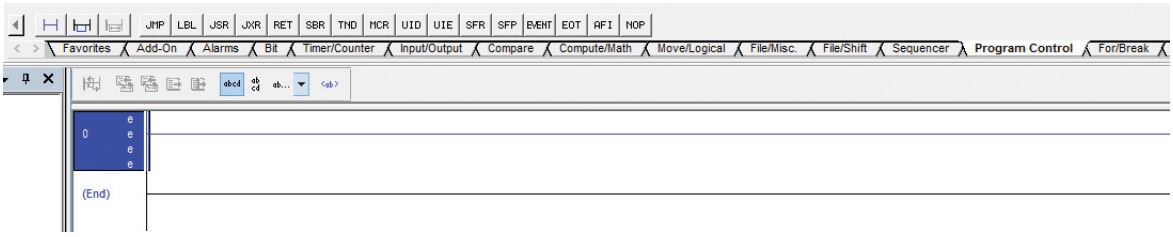
2. How do you find and select the proper PAC controller?

Focus 2: Ladder Logic

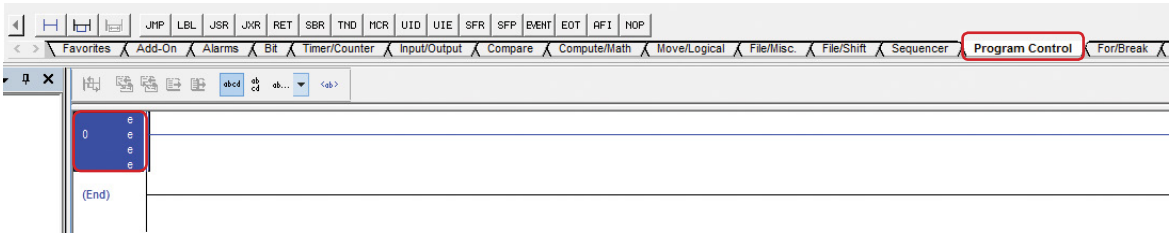
The next step in creating a new project is setting up the main routine. The main routine executes the ladder logic for the main ladder. Click the (+) sign to open the **MainProgram** and show the **MainRoutine.**



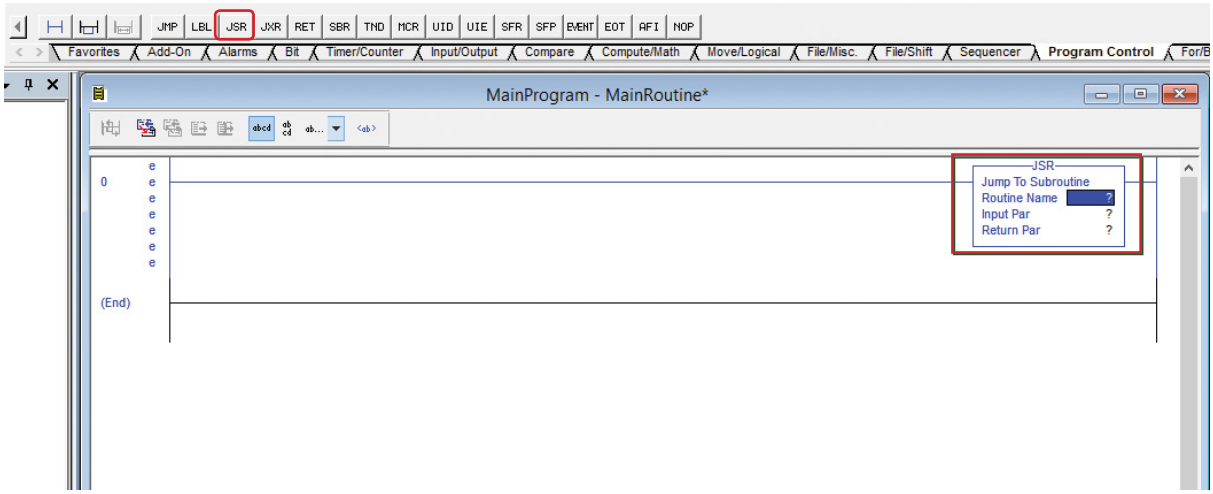
Find the tab near the top that says **Program Control**. Click on the tab and a list of program controls will show above the tabs.



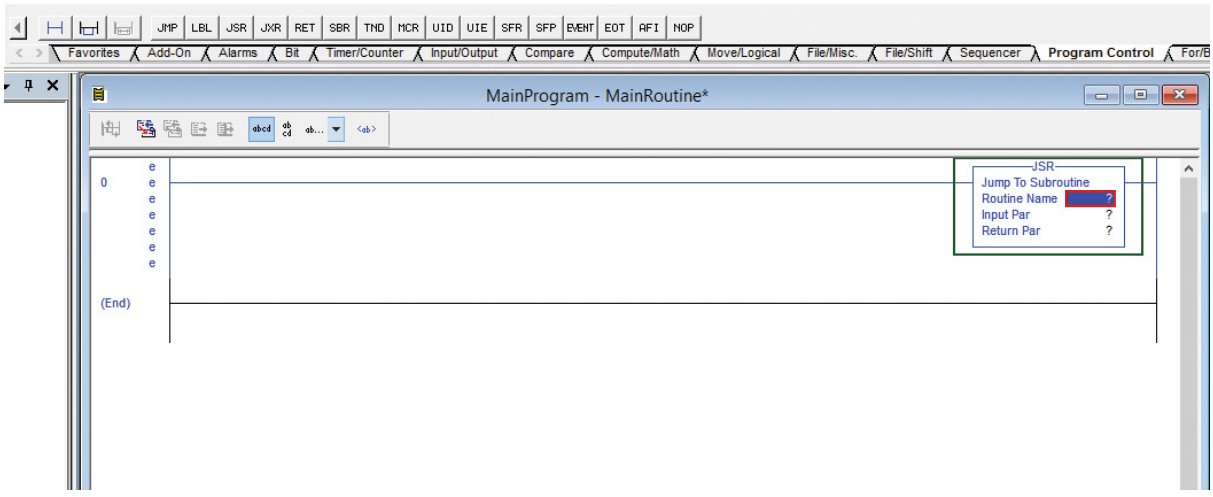
Find the tab near the top that says **Program Control**. Click on the tab, and a list of program controls will show above the tabs. We are



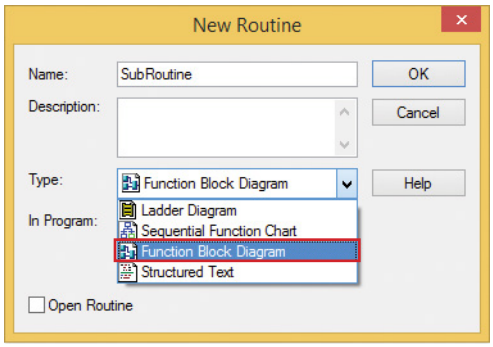
Click on the JSR (Jump To Subroutine) to create a sub routine in the main routine.



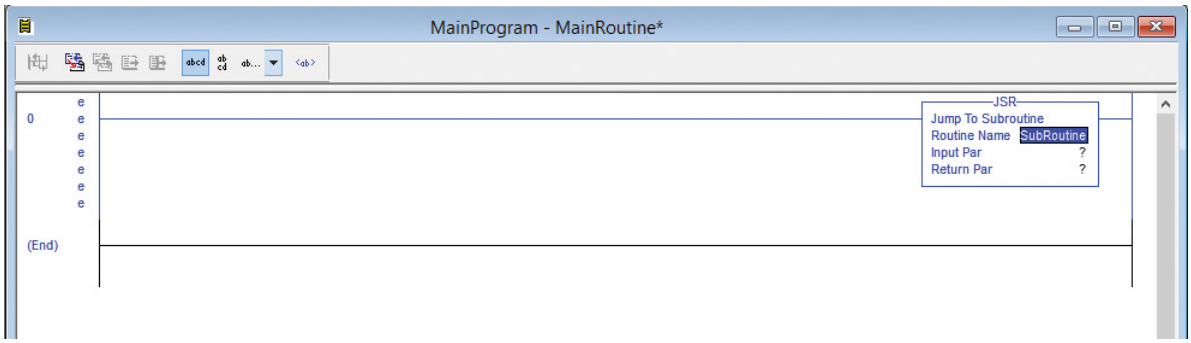
Right-click on the question mark for the Routine Name. A window will appear for creating a new sub routine.



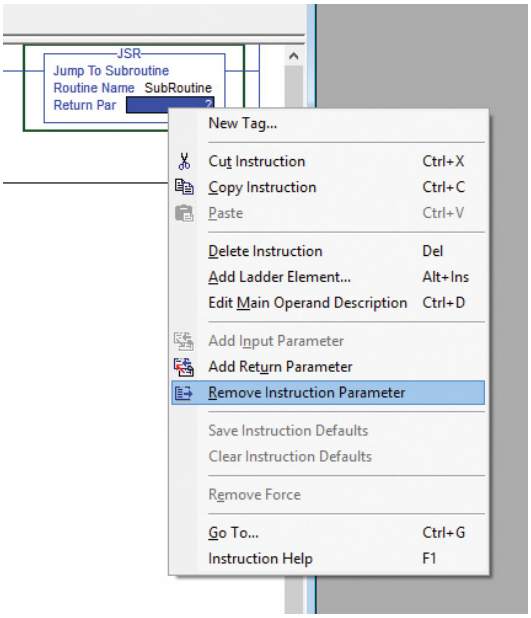
Name the new sub routine and select the drop-down for type. Select Function Block Diagram for the sub routine. Click OK twice to accept the settings.



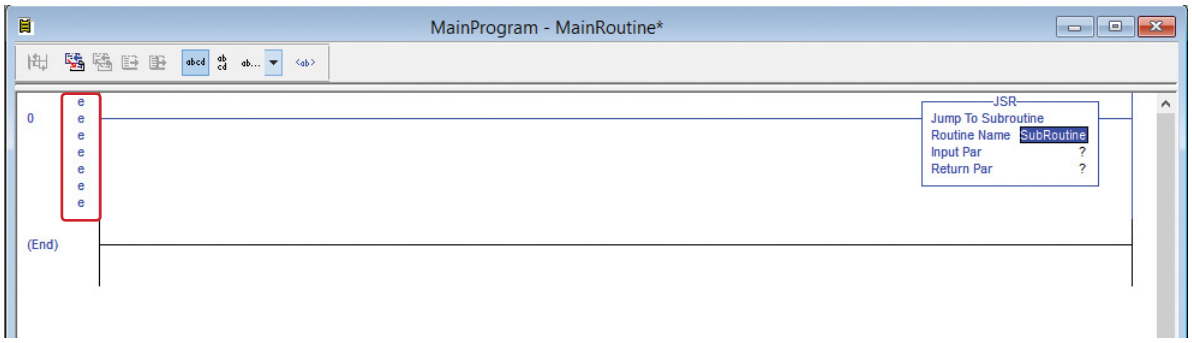
Click on the JSR (Jump To Subroutine) to create a sub routine in the main routine.



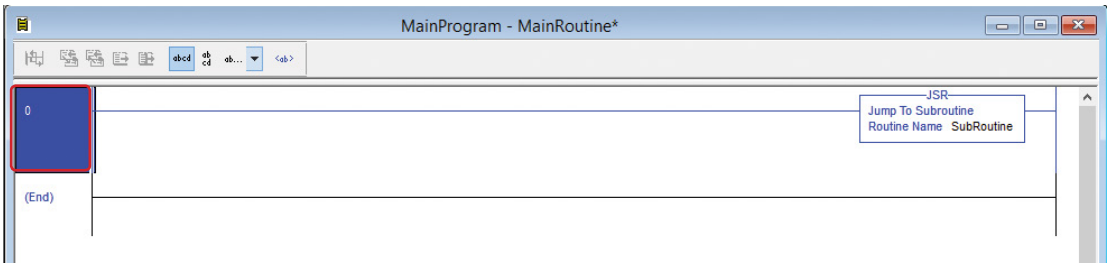
Right-click on the question mark for the Parameter. A window will appear for editing the parameter. Click remove instruction parameter. Do this for both parameters.



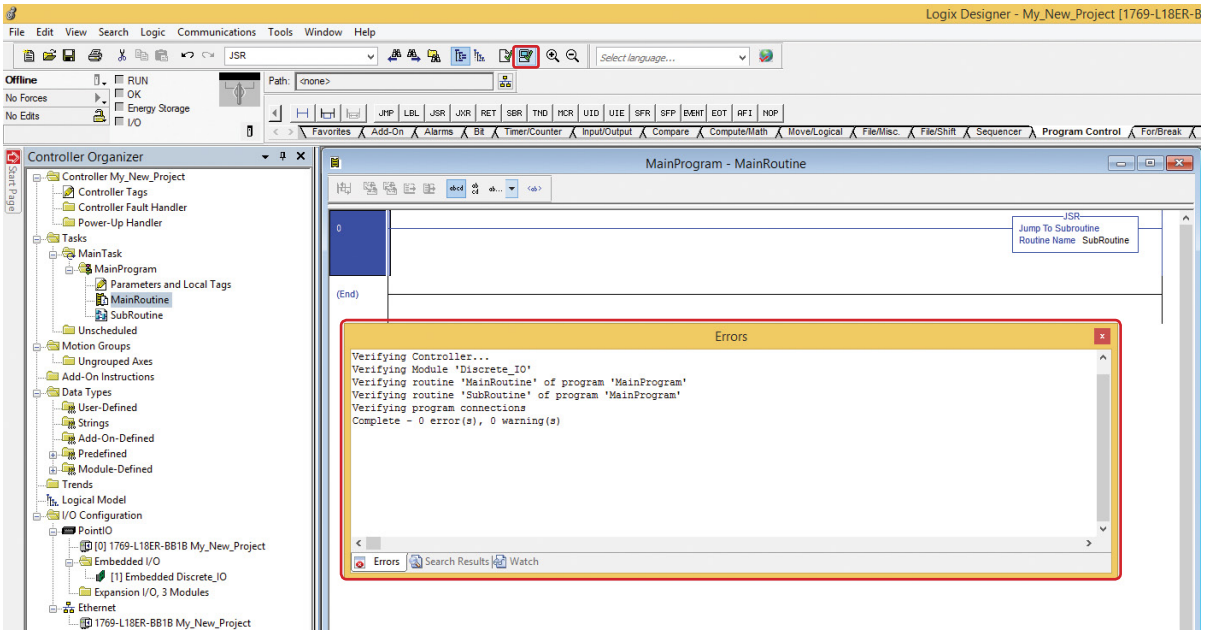
Before removing the unused parameters, the routine has rung errors. Errors are denoted by the letter “e” next to the rung. See below:



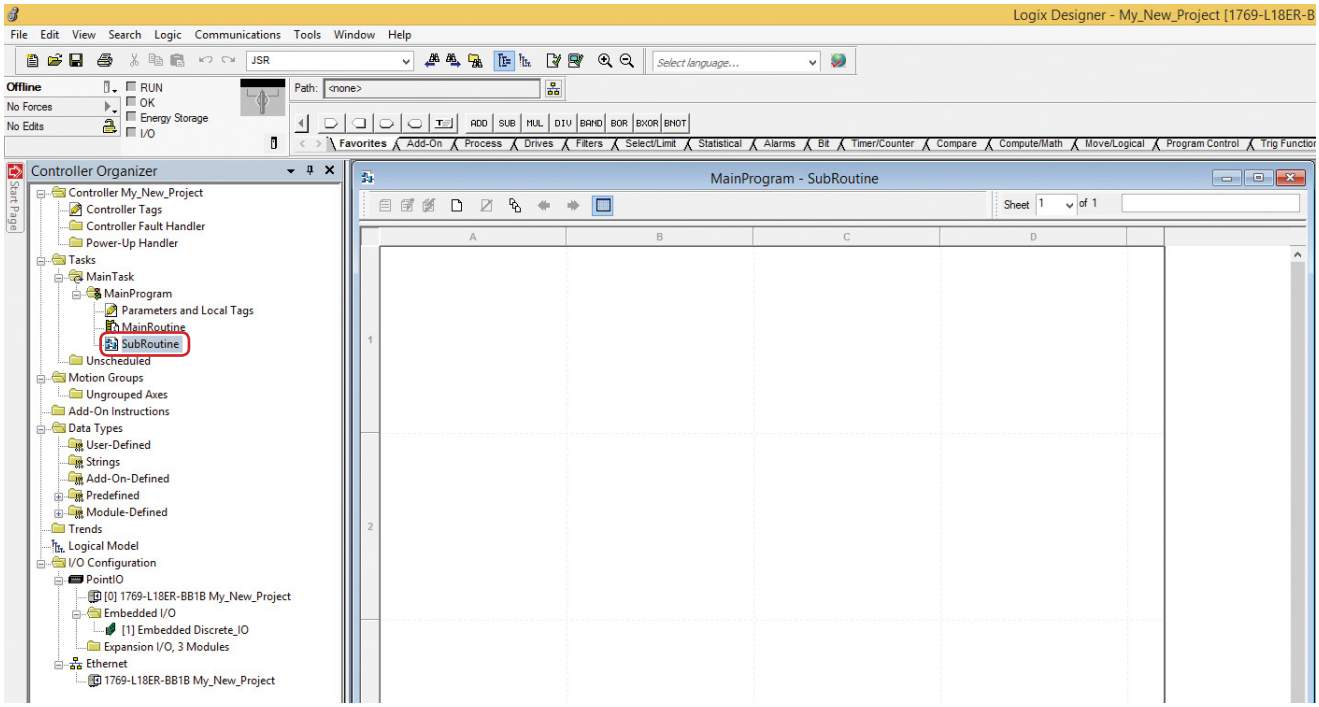
After removing the parameters the routine is error free.



To verify the programming the for errors, click the verify controller button as shown below. Monitor the errors in the **Errors** windows.



The new program is now set to run correctly under the new sub routine. This allows the use of the customized Function Block Programming for using Add-On Instructions (AOIs).



For detailed instruction on Ladder Logic, see Unit 8.

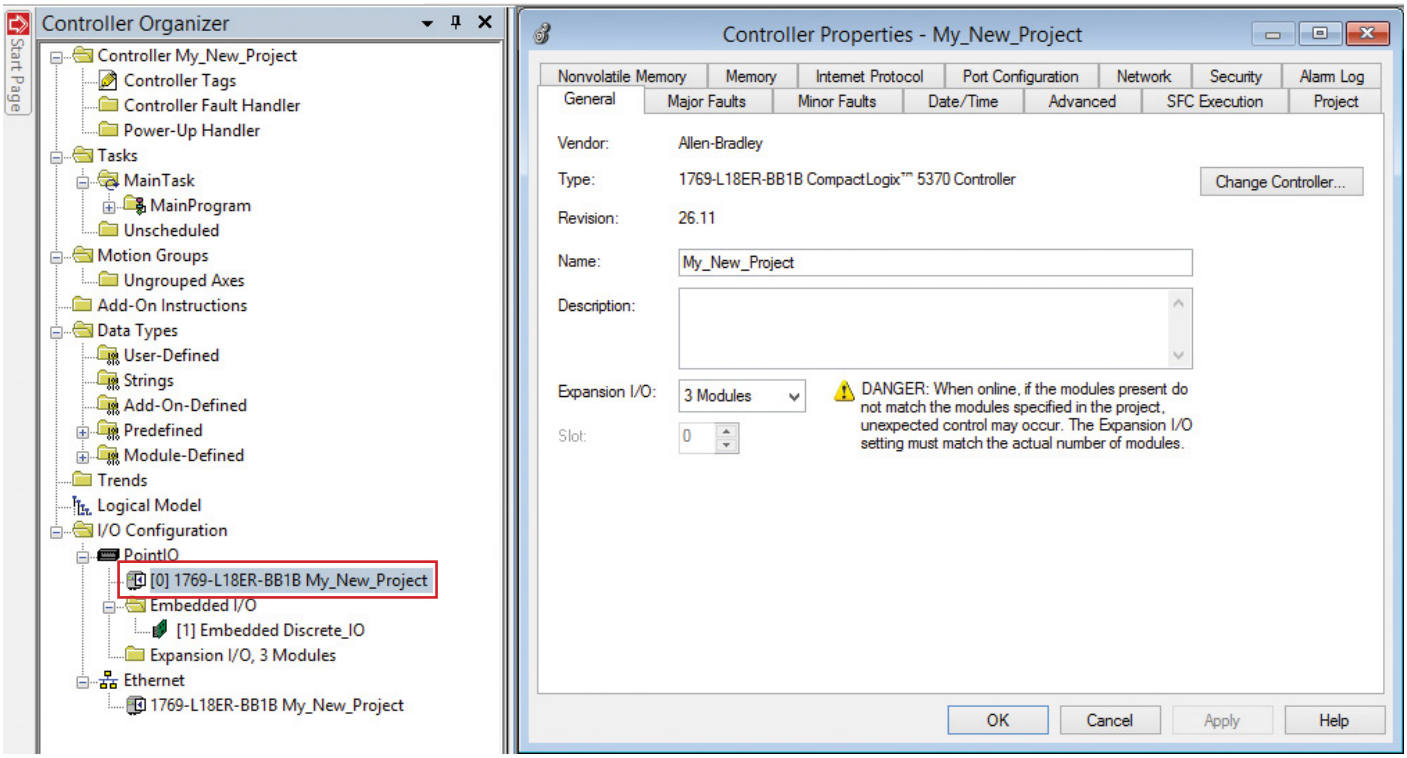
Skill Builder Section 1, Focus 2

1. How does the MainProgram utilize ladder logic?

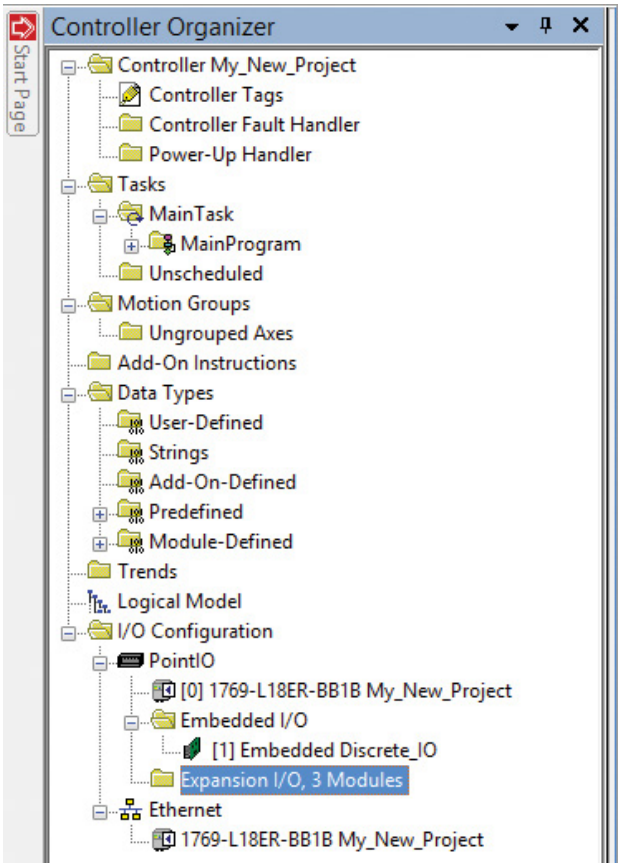
2. What is the benefit of using a SubRoutine?

Focus 3: Adding Expansion Modules

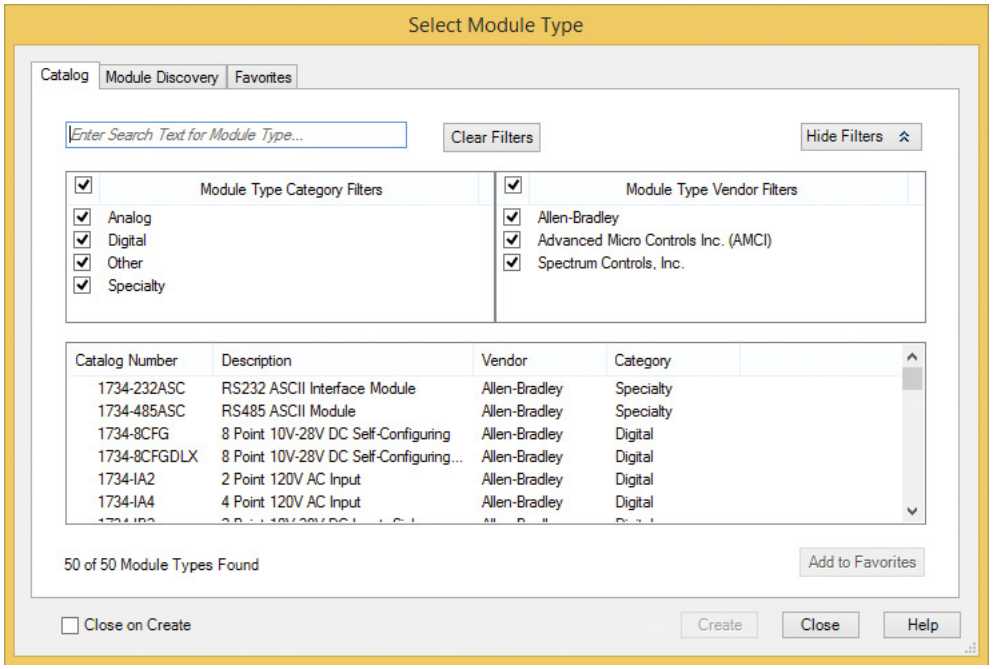
When creating a new project, the number of expansion modules was pre-determined. If that number is different than what the project shows, edit the properties of the controller. Right click on the controller and select properties. The controller properties window will show up. Update the correct number of expansion modules



Next, right-click on the **Expansion I/O** folder and select



A new window will be displayed for **Select Module Type**

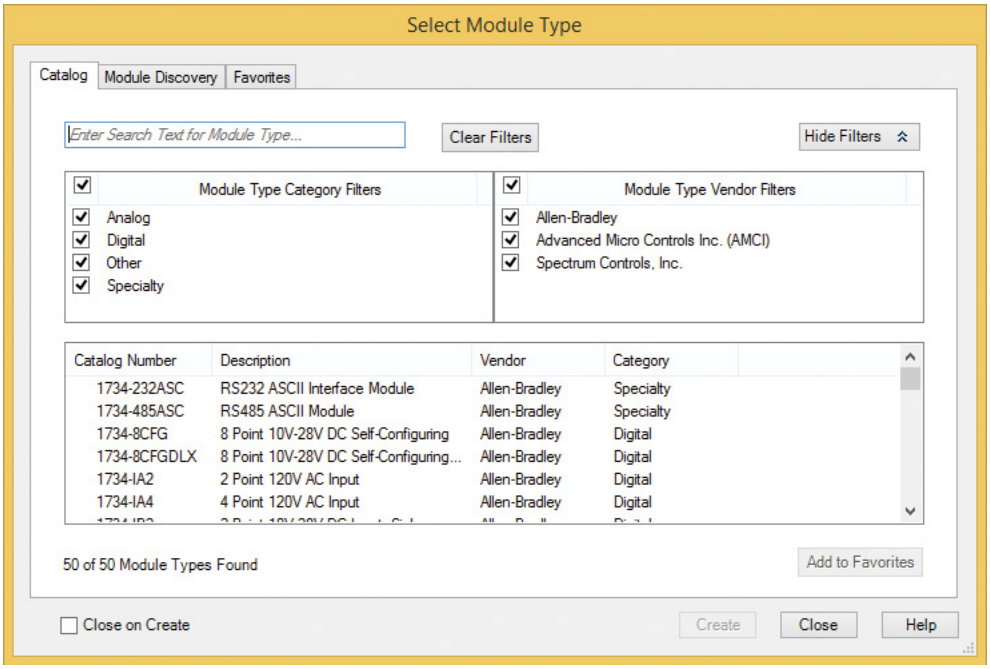


Section 1: Studio 5000 Logix Designer Programming Environment

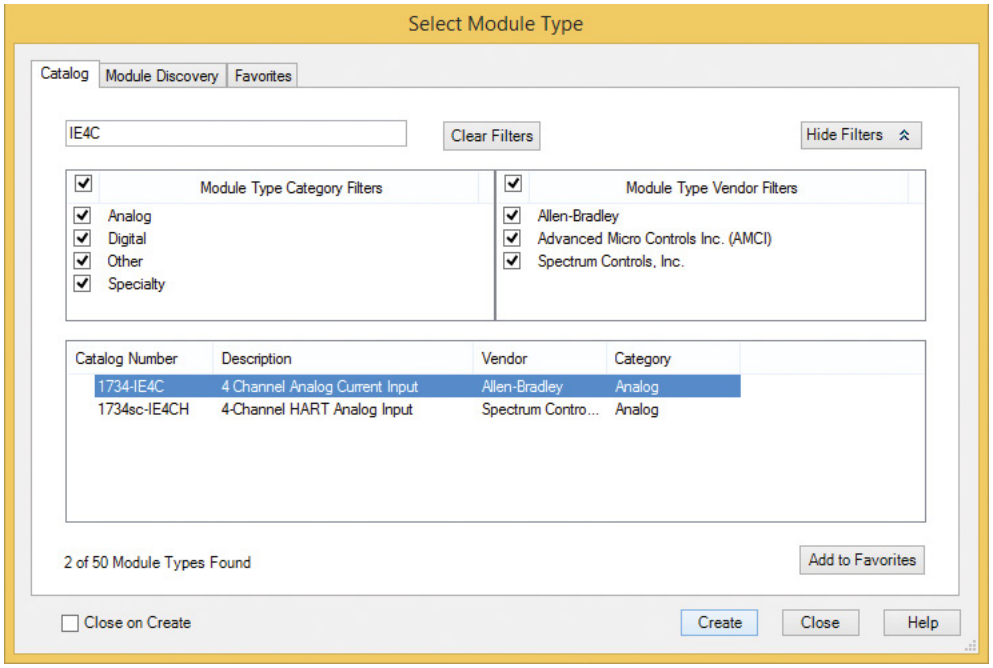
The FLUIDMechatronixTM system uses 3 expansion modules. Recall that the location of modules were the following:

Module 1: Embedded I/O
Module 2: IE4C
Module 3: IT2I
Module 4: VHSC24

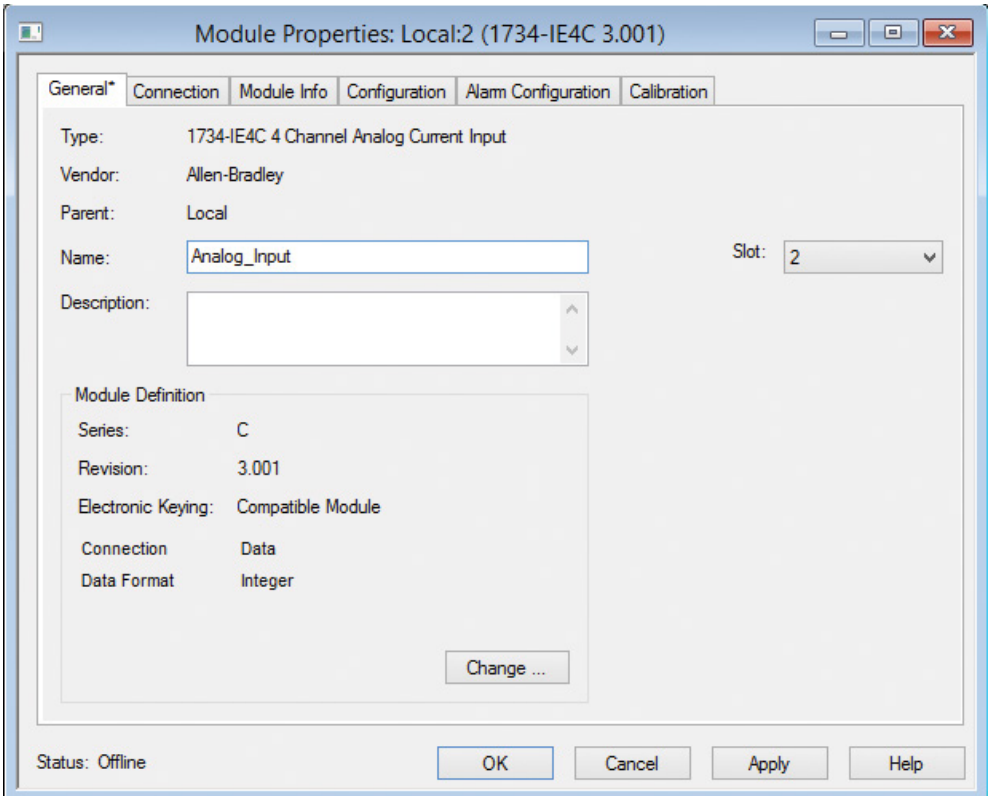
Use the Select Module Type window to search for modules 2-4. Start with module 2, by searchine IE4C.



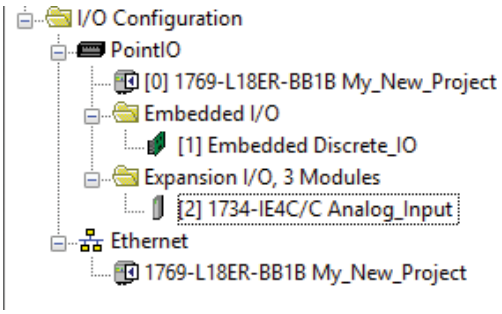
The search is narrowed down to 2 modules of this type. Select the correct module and click **create** to add it to the program.



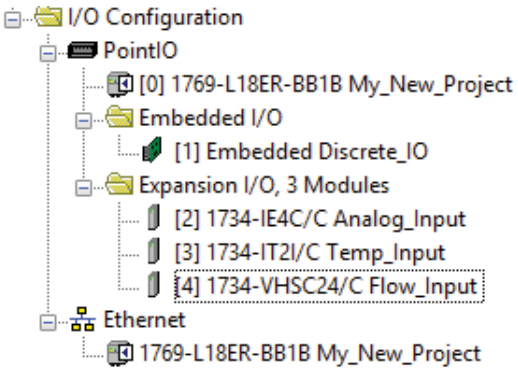
A new window will pop up allowing you to configure the module. At this point, all we need to do is give the module a name and click OK.



The module configuration should show the module in the correct location with proper naming.



Repeat the steps to add the IT2I and VHSC24 Module. Note: It is important that the module specified is in the proper location on the slot. Improper slot specification can lead to dangerous outcomes during runtime.The Module list should show the following:



Skill Builder Section 1, Focus 3

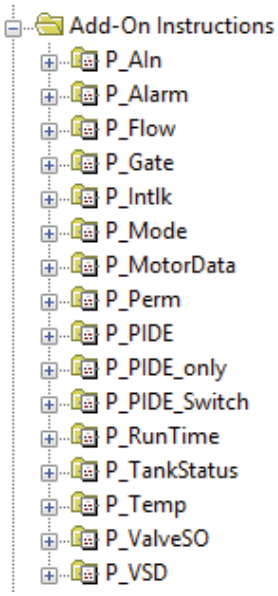
1. How can modules be added to the project?

2. Why is the first expansion module showing up as module # 2?

3. How do the modules communicate with the project?

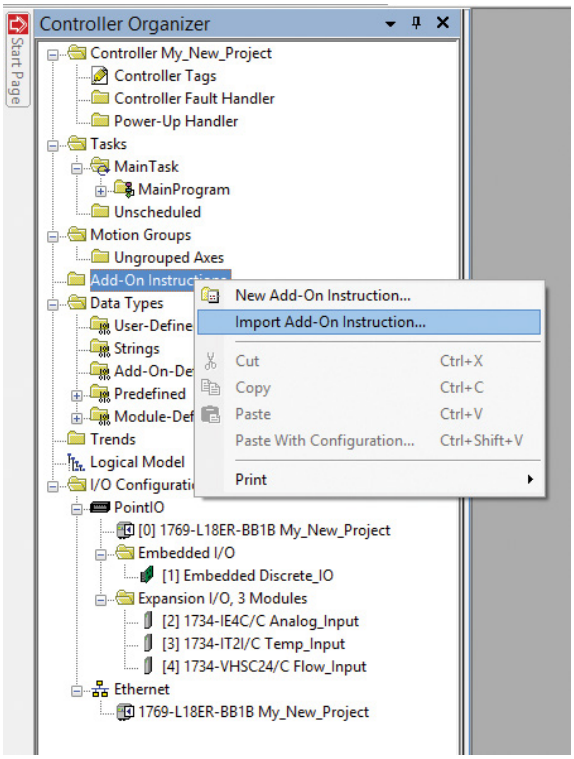
Focus 4: Function Block Diagram

The main purpose of the FBD programming is to simplify logic programming. FBDs not only add simplicity but also provide a graphical diagram of the circuit schematic. FBD instructions sometimes have both Structured Text and Ladder Logic programming. Many of the AOIs used in Studio 5000 are based on FBD programming. This makes complicated programming appear simple, saving companies money by not paying for programmers. FLUID*Mechatronix*™ utilizes the FBD instructions for most of the programming in the system.

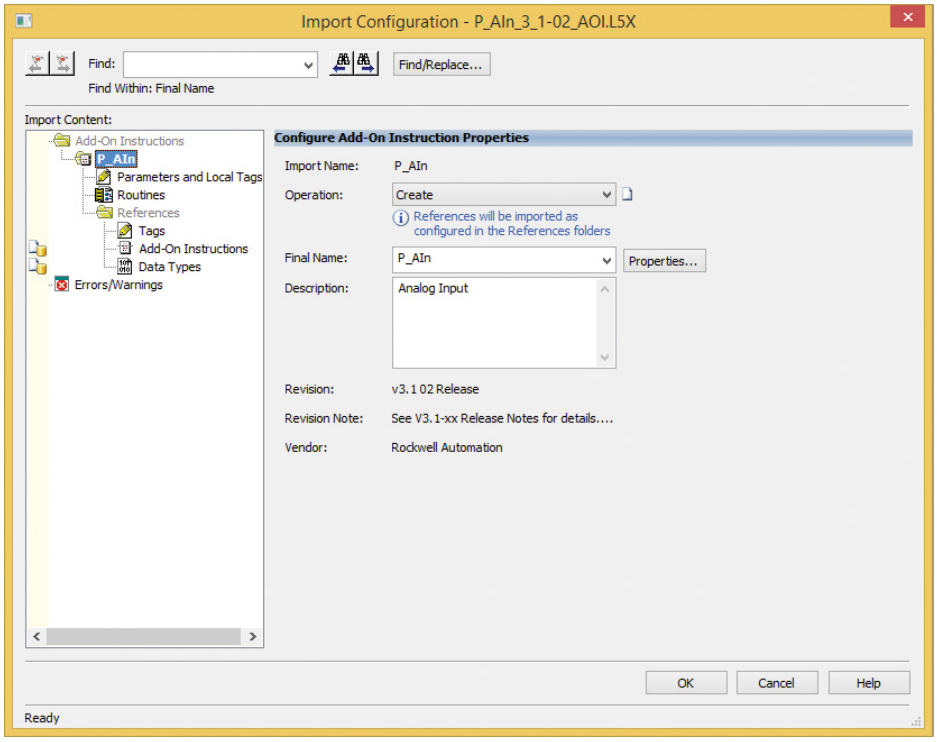


Recall My_New_Project where we created a sub routine under the main routine with FBD programming. Now we can begin populate the first page of My_New_Project with FBD instructions.

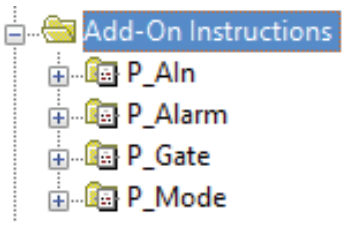
In order to use Add-On Instructions, they must be added to the project. Right click on the Add-On Instructions folder and select **Import Add-On Instruction.**



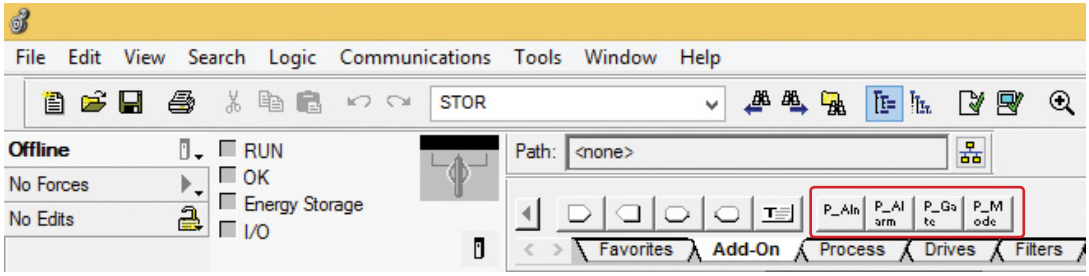
Locate the PlantPAX add-on instructions in the FLUID*Mechatronix*™ Support Documents. PlantPAX v3.1 is included with the system. The latest version can be downloaded from Rockwell Automation support. Once you locate the PlantPAX folder, browse to Files > Process Objects Library > Process Add-On Instructions > P_AIn_3_1-02_AOI.L5X. An import window will open, change settings if desired and click OK.



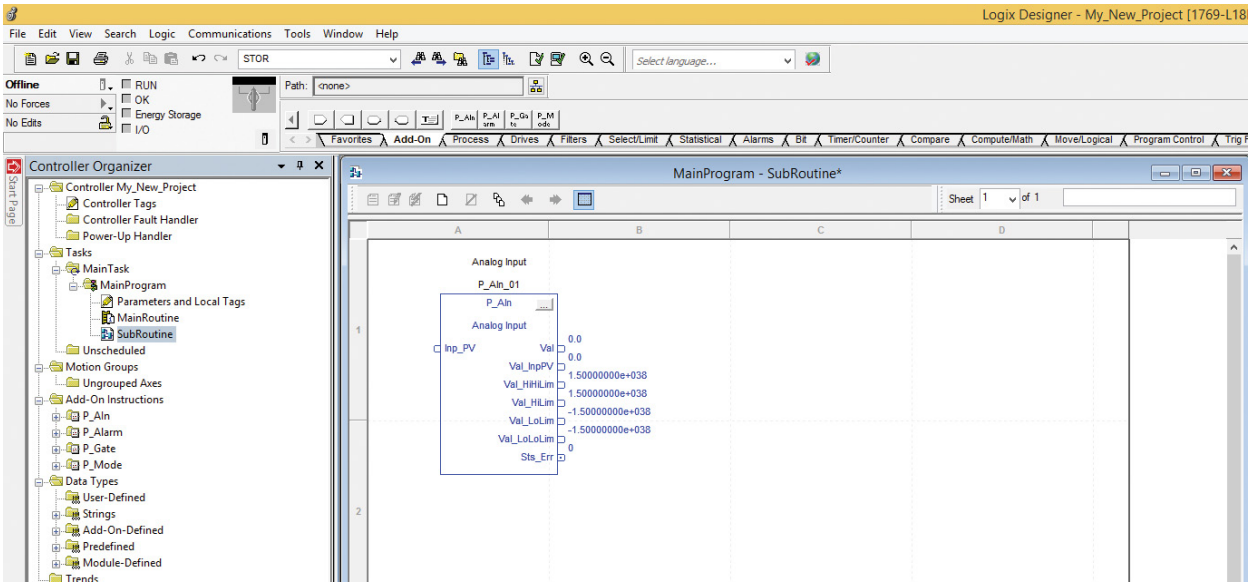
The AOI and other required components are imported. The new imports are shown below.



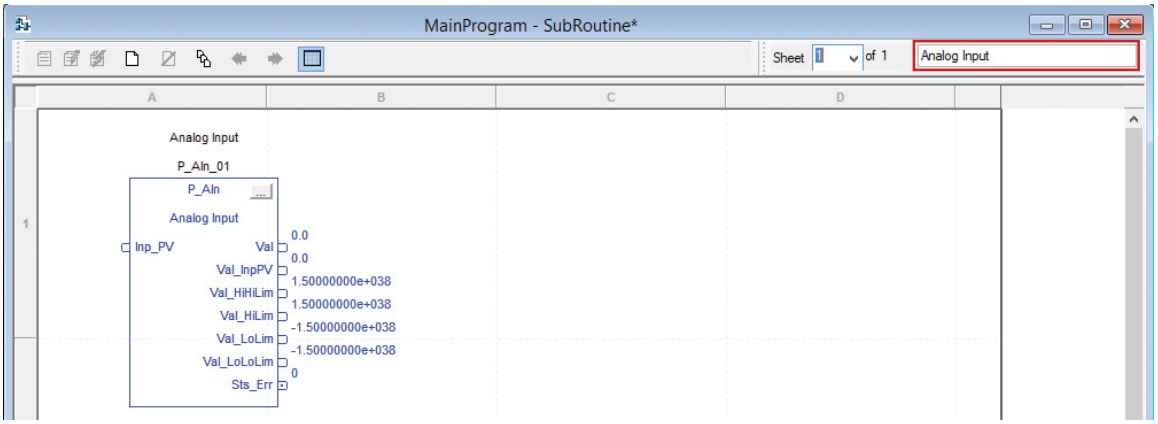
After importing the AOI, the Add-On tab will show the new modules. The new FBD modules can now be selected from the toolbar.



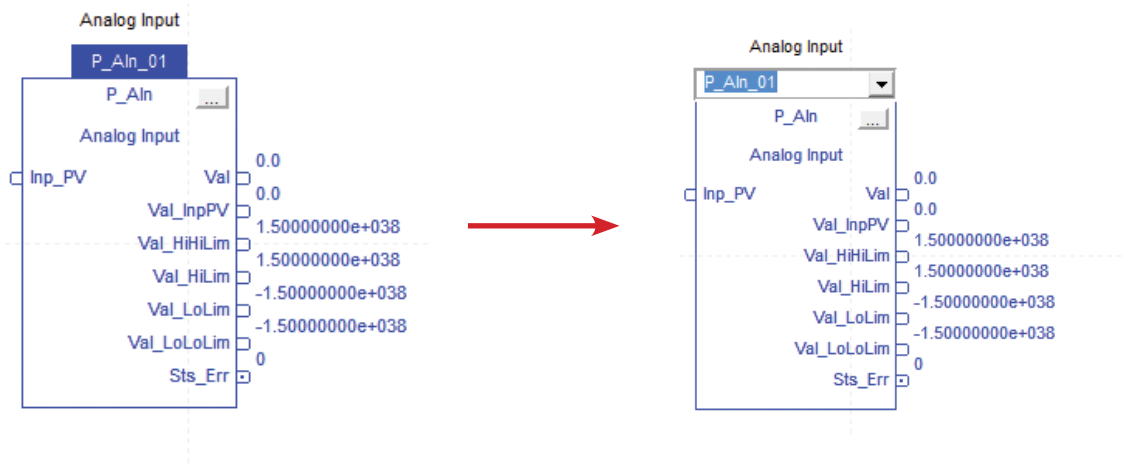
Select the P_AIn and the AOI will be placed in the SubRoutine sheet 1.



Rename the sheet as Analog Input.

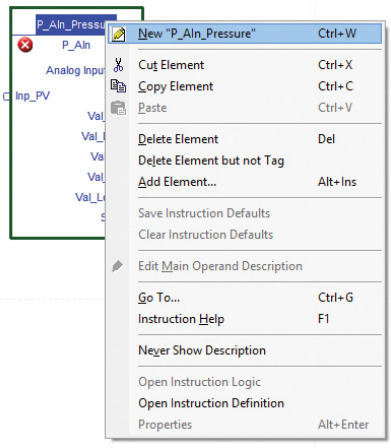


Double-click on the title for the AOI. This will allow you to edit the tag name of the AOI.

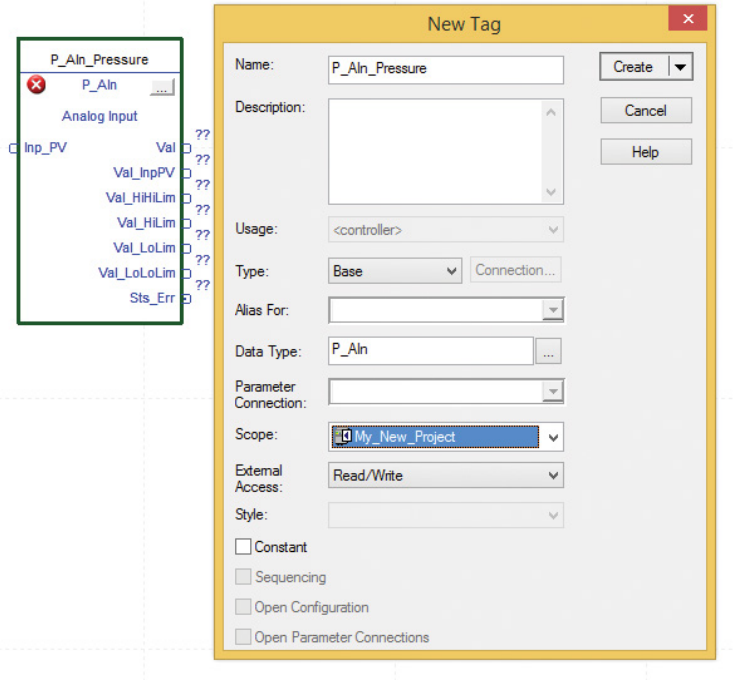


Change the instruction tag name to P_AIn_Pressure and the tag name will change. There will be an X denoting that there is a problem with the tag name. The tag is still stored in the local tags for the MainRoutine.

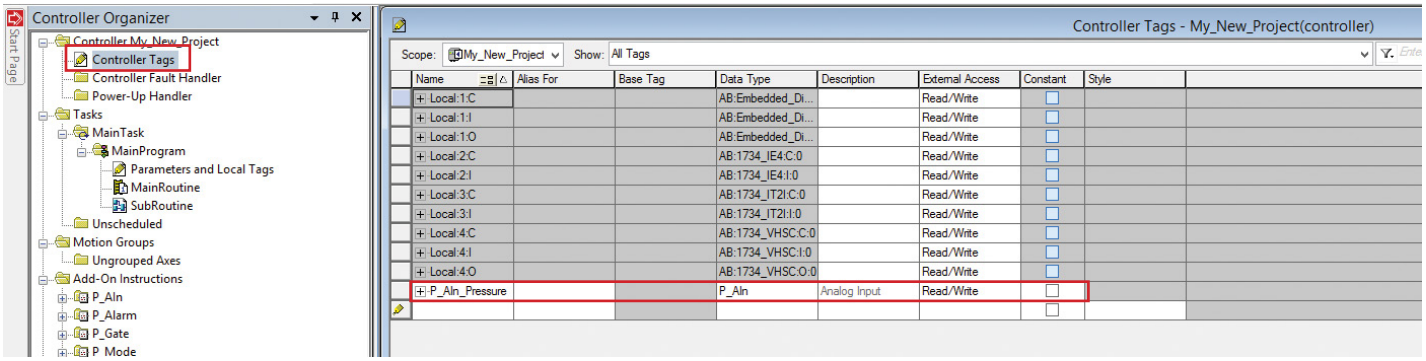
Next, we want to make sure that the tag is stored in the controller. Right-click on the tag and select **New "P_AIn_Pressure"**



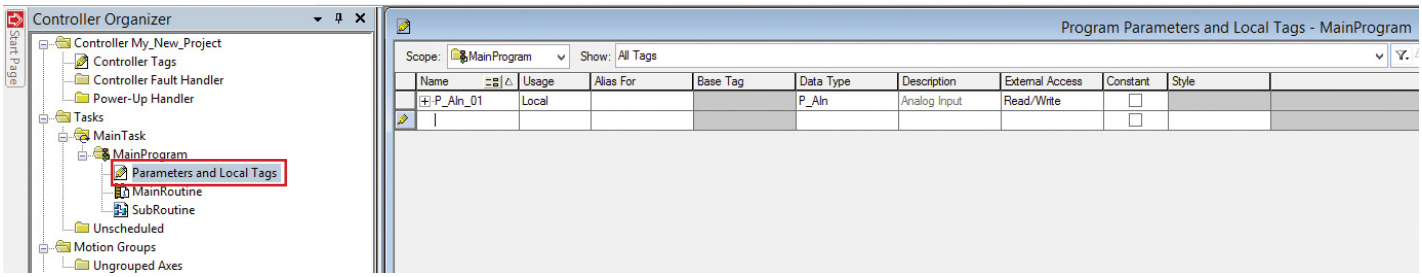
The New Tag window will open. Click on **Create** to make a new tag under the Scope of My_New_ Project. The new tag will be created as a Controller Tag, not a local MainRoutine tag. Storing tags in this manner will help the communication process between Studio 5000 programming and FactoryTalk View ME.



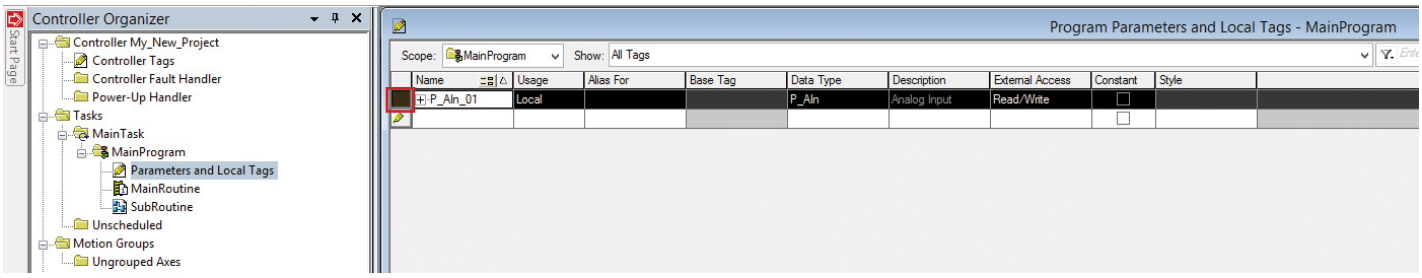
Double-Click on Controller Tags. You will see the new tag P_AIn_Pressure in the list.



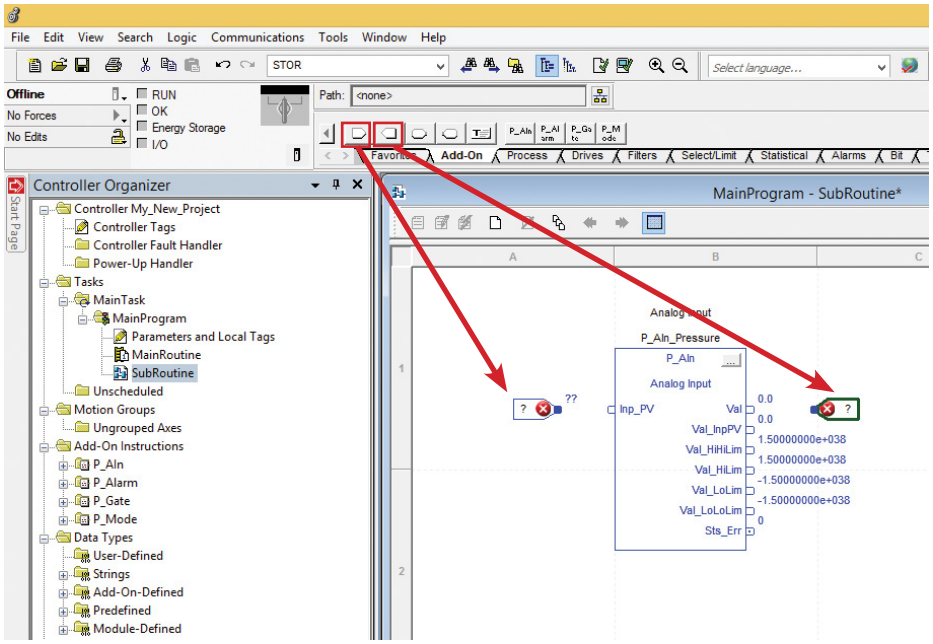
Next delete the local tag stored in the **MainProgram** > **Parameters and Local Tags.**



Click on the left square next to the tag that needs to be deleted. The whole tag will be highlighted in black. Right-click on the square and select Delete.

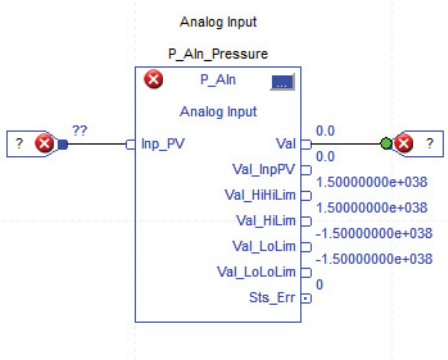


Go back to the SubRoutine window. Drag an input and output connection to the Analog Input page.

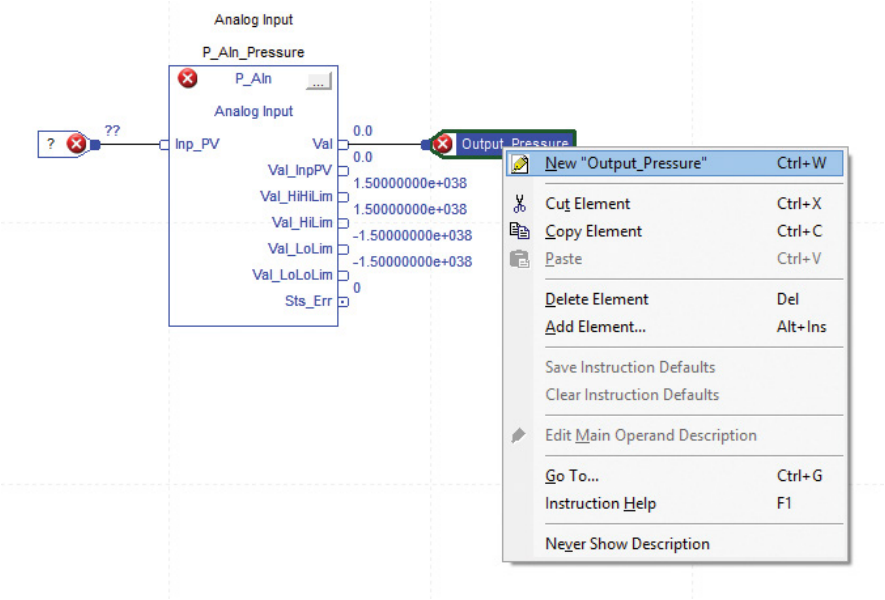


Section 1: Studio 5000 Logix Designer Programming Environment

Wire the input and output of the P_AIn_Pressure FBD. Click on the nodes and select the desired location to wire the tag to.

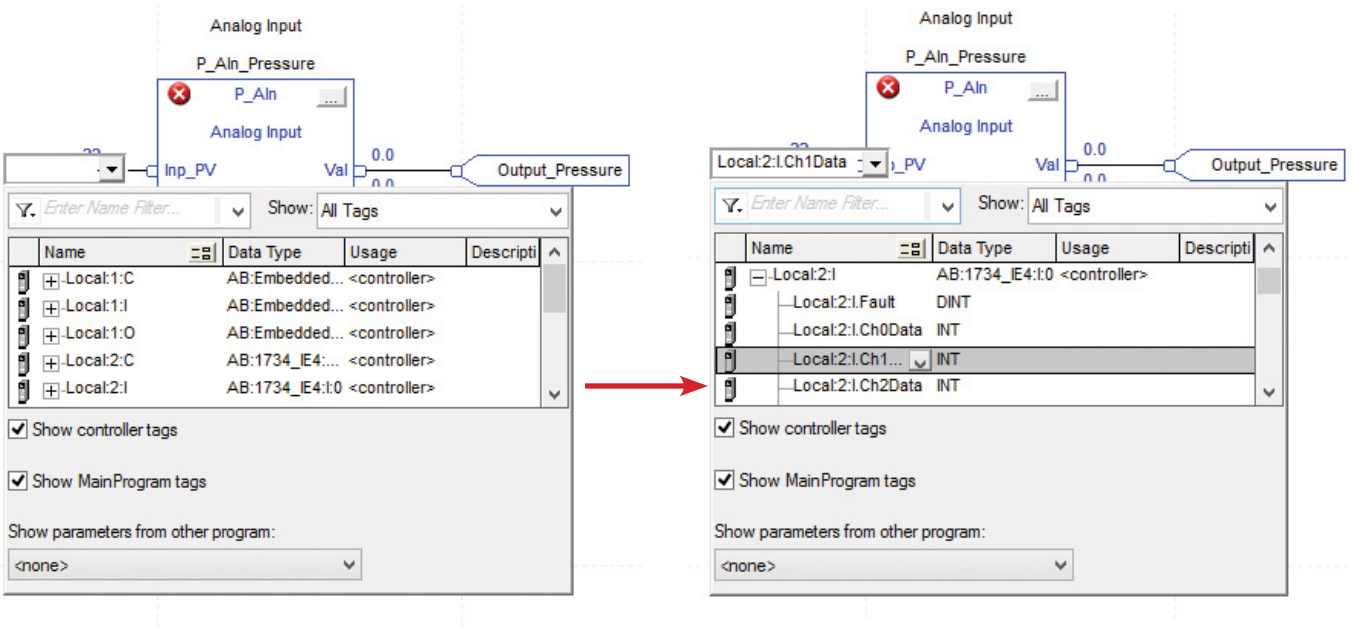


Rename the output tag to Output_Pressure. Create a new tag in the scope of My_New_Project and select **Create**. The red X next to Output_Pressure will go away.

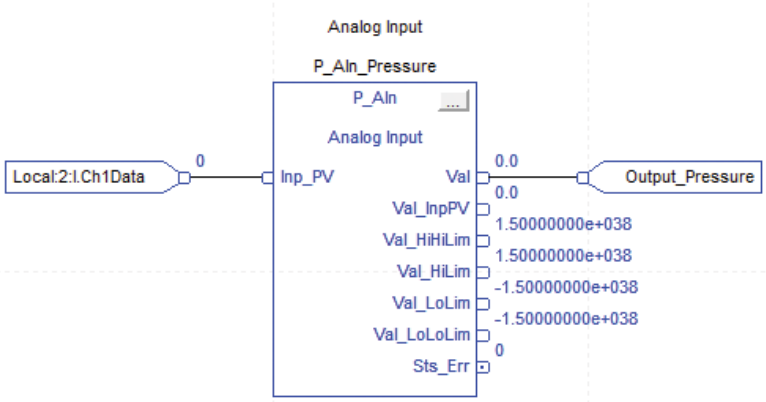


Next we need to wire the input to the proper channel on the 1734-IE4C. Recall that the pressure tranducer was connected to.

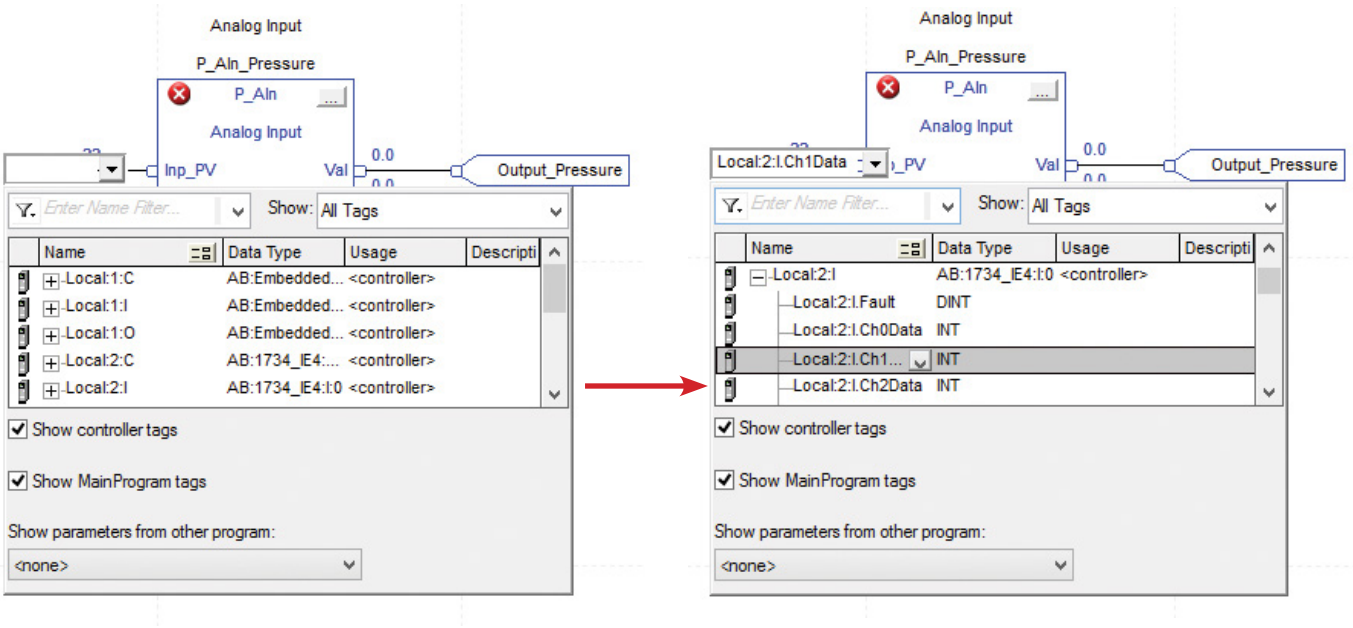
Wire the input and output of the P_AIn_Pressure FBD. Click on the nodes and select the desired location to wire the tag to. Locate the **Local:2:I.Ch1Data** tag and select it to connect the tag to the Inp_PV.



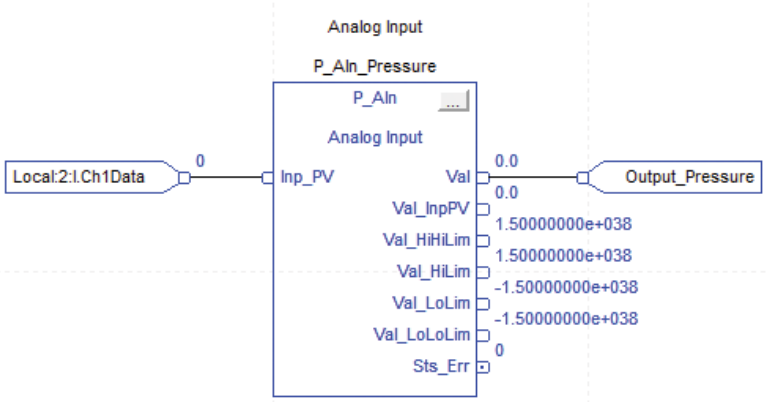
The red X error will go away and the P_AIn_Pressure FBD is wired correctly.



Wire the input and output of the P_AIn_Pressure FBD. Click on the nodes and select the desired location to wire the tag to. Locate the **Local:2:I.Ch1Data** tag and select it to connect the tag to the Inp_PV.

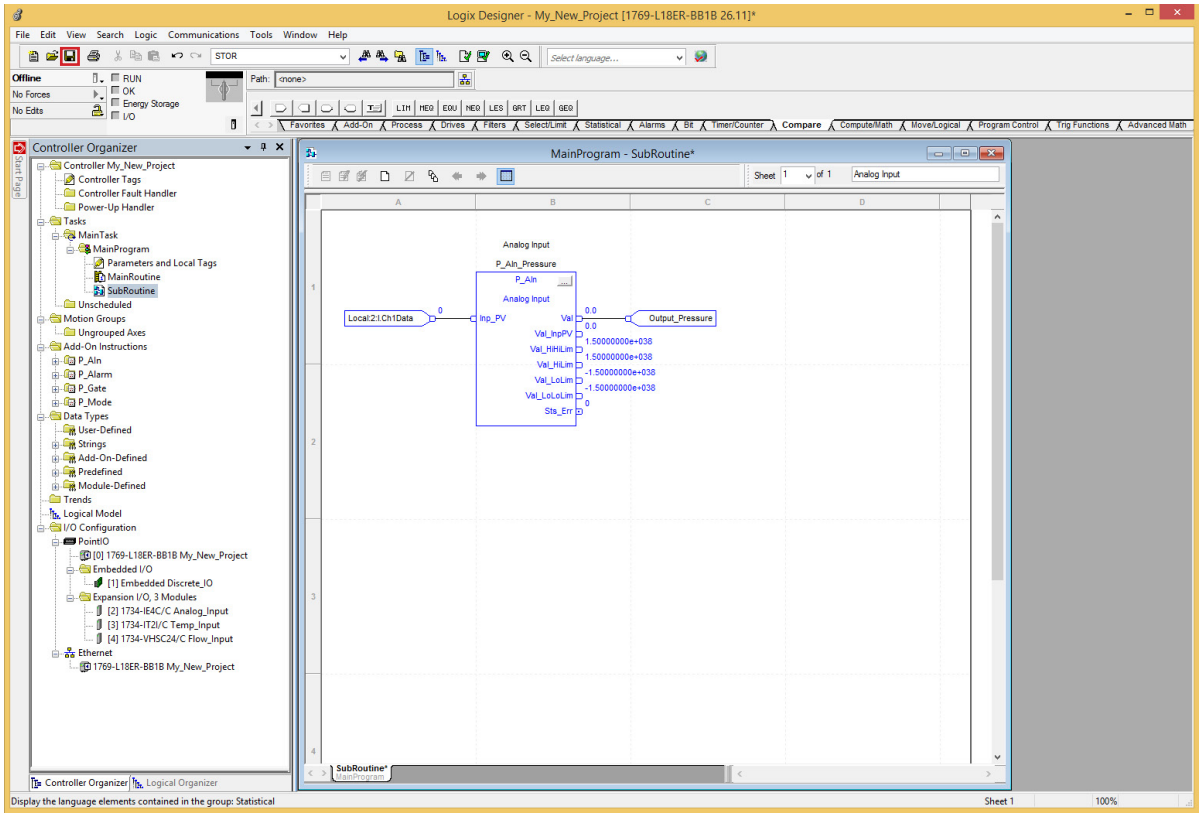


The red X error will go away and the P_AIn_Pressure FBD is wired correctly.



Section 1: Studio 5000 Logix Designer Programming Environment

Save My_New_Project by clicking the save button. Your new project should look like the diagram below:



Skill Builder Section 1, Focus 4

1. How does a function block diagram simplify programming?

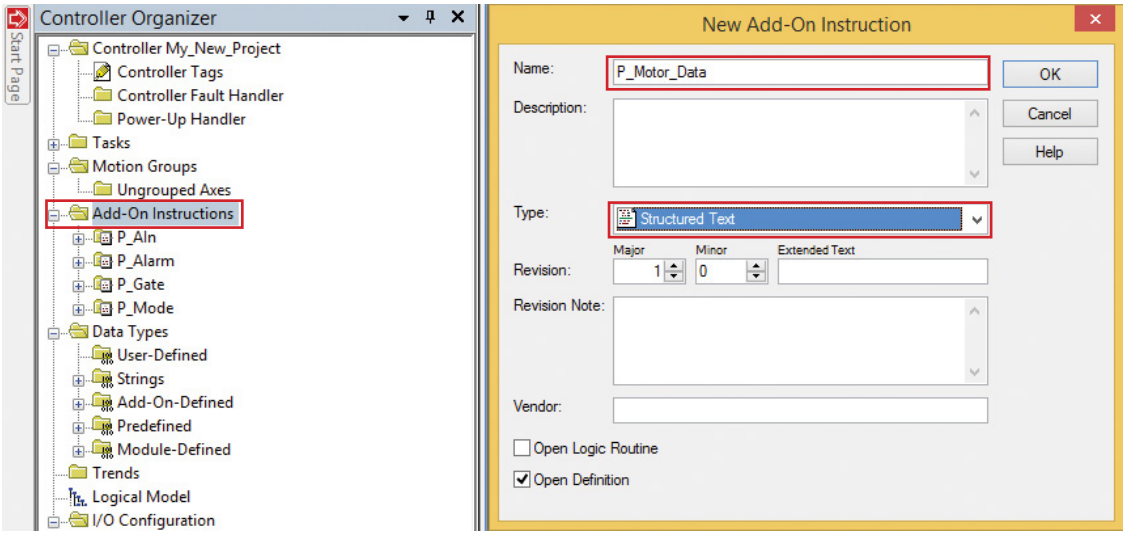
2. Describe how a function block diagram communicates tag data with a PAC.

3. Does the programmer need to know ladder logic to use the function block diagrams?

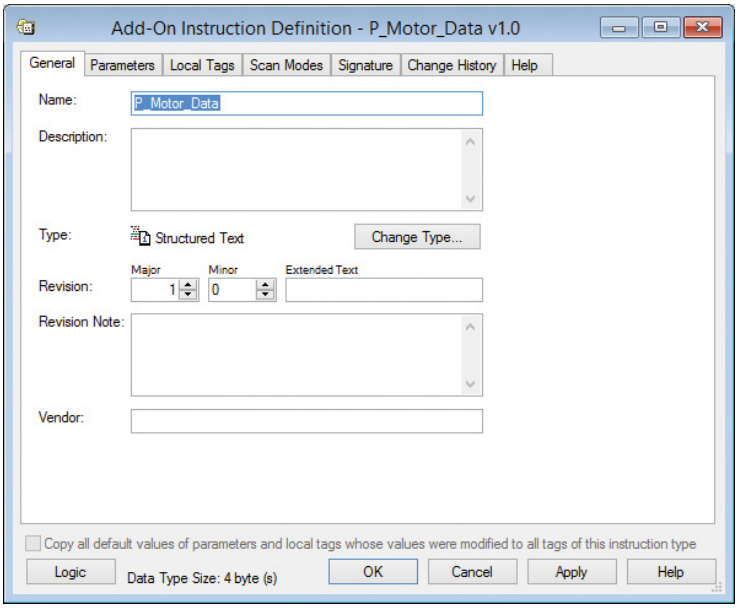
Focus 5: Structured Text

Structured Text programming is another feature of the Logix Designer programming environment. It allows the programmer addional ways to acomplish a process task. Structured text can also be used to build custom AOIs that can be used on the FBD.

Next we can create a new AOI for the motor data. Right-click on Add-On Instructions to add a new instruction. Name the instruction **P_Motor_Data**. Be sure to select the **Structured Text** for the Type of instruction.

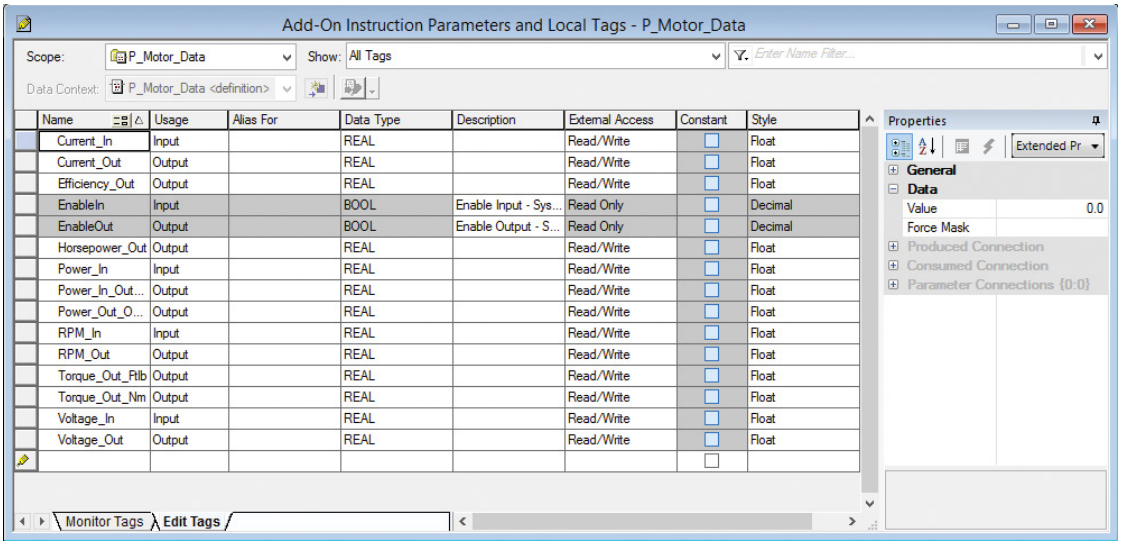


The new AOI definition window will pop up. Here we can add a description and enter the local tags that will be needed for the instructions.



Next, create the new tags for the AOI. Be sure to select the proper usage for the tag (Input vs Output) It is sometimes useful to label inputs and outputs directly in the tag. It can avoid confusion when writing the programming for the local tags.

- Current_In
- Current_Out
- Efficiency_Out
- Horsepower_Out
- Power_In
- Power_In_Output
- Power_Out_Output
- RPM_In
- RPM_Out
- Torque_Out_Ftlb
- Torque_Out_Nm
- Voltage_In
- Voltage_Out



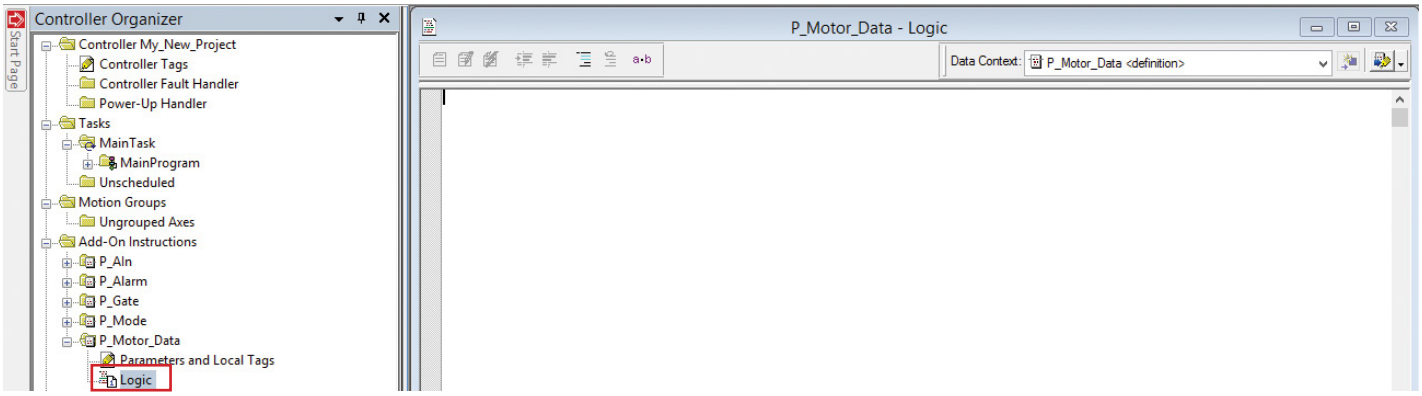
In order to write the code for the P_Motor_Data structured text, we must first investigate some basic programming techniques. This form of programming can account for many different scenarios. This is universal (to some extent) to most programming languages. The main structured text statements in FLUID*Mechatronix*™ are:

If
Then
Else
Elsif
End_If

The main goal of the P_Motor_Data AOI is to create outputs for the types of motor data that the FLUID*Mechatronix*™ system requires. This same task can be accomplished by many of the built in FBD instructions such as less than, greater than, and logic. However, structured text can put all of these variables into one FBD. This consolidates programming and saves the PAC memory space.

In programming, data is read from left to right, top to bottom (just like reading a book).
The first structured text programming that needs to be written is the RPM Output code.

Open the structured text Logic window by double-clicking **Logic** in the **P_Motor_Data AOI**.



Write the following text in that window:

```
(*RPM Output*)
```

If the code is written correctly it will turn green. This code is strictly a means of labeling code within the structured text. It is not necessary for the program to operate, but allows the programmer to quickly navigate the code if any changes or problems occur. After placing the label, we can write the specific code below for the RPM data:

```
If RPM_In > 0 Then
    RPM_Out := RPM_In;
```

Local tags in the structured text will automatically turn red. This lets the programmer know that an eligible tag has been selected for the specific task. Keep in mind, structured text programming also has a built in error checker similar to the other types of programming.

This code is simply checking to see IF RPM data is available (any number greater than 0) on RPM_In. Then, RPM_Out is defined to be equal to (:=) RPM_In. The semi-colon basically says that the specific part of the instruction is finished. This may seem redundant, but this is needed to show proper motor output.

Next we need to define when the output of RPM_Out is zero using the Else programming statement. In this case Else represents all other cases from above. Our next line of code is the following:

```
Else
    RPM_Out := 0;
End_If;
```

Else RPM_Out is defined to be equal to zero. This means that all other cases for RPM_In will provide an RPM output of zero. With this code, the HMI can display a value of 0 when the motor is not running. If we didn't have this code, the RPM_Out value would be empty or blank on the HMI control screen. The End_If code is there to mark the end of the If statement. The end must be marked so the processor knows when that code has been executed. It is like the period at the end of the sentence. It must be defined in order to move on to the next sentence.

The final RPM programming statement should be the following:

```
(*RPM Output*)
If RPM_In > 0 Then
     RPM_Out := RPM_In;
Else
    RPM_Out := 0;
End_If;
```

Next we can use similar arguments to define the motor data for voltage and current. Write the code as followes:

```
(*Voltage Current Output*)
If Voltage_In > 0 Then
     Voltage_Out := Voltage_In / 10;
     Current_Out := Current_In / 100;
Else
     Voltage_Out := 0;
     Current_Out := 0;
End_If;
```

The data from the PowerFlex 525 is scaled (multiplied) at the output. The code above will be used to take that data in, and scale it back to its proper output units. When voltage is present, current is also present so we don't need two different If statements. Structured text programming allows computation of basic mathematics on local tags. Voltage_In / 10 takes the input voltage and divides the number by 10. This gets the motor voltage data to display the voltage correctly in terms of Volts. Current_In / 100 takes the input current and divides the number by 100. This gets the motor current data to display the current in terms of Amps. Note: In this case the If and Else statement have more than 1 condition. Each condition needs to be ended with a semi-colon or the processor will get confused.

The next section of code is a little more complicated. It uses the same basic features, but more math is needed to calculate power, torque, and efficiency. Write the code below into My_New_Project.

```
(*Torque Power Output*)
If Power_In > 0 Then
    Torque_Out_Nm := (9.5488 * Power_In * 10 / RPM_In);
    Power_In_Output := (Current_In / 100) * (Voltage_In / 10);
    Power_Out_Output := Power_In * 10;
    Efficiency_Out := (Power_Out_Output / Power_In_Output)*100;
Else
    Torque_Out_Nm := 0;
    Power_In_Output := 0;
    Power_Out_Output := 0;
    Efficiency_Out := 0;
End_IF;
```

In order to understand the mathematics, we must first investigate torque. Basically, torque measures the amount of force required to make an object rotate. The force causes the shaft to move a specific distance, so it is very similar to work. The torque equation is as follows:

$$\tau = \frac{9.5488 \times P_{OUTPUT}}{RPM}$$

In this case, the power output from the VFD would need to be in kW. The VFD outputs its power value in hecto watts hW. The value is scaled to output data with the best resolution. In order to get our values kW, the power needs to be multiplied by 10. That is why the equation has Power_In * 10. Therefore,

Torque_Out_Nm := (9.5488*Power_In * 10 / RPM_In)

We know from Unit 1 that Power = Current X Voltage. We can calculate the power that the motor uses by taking the current and voltage values that run the motor. The next line of code is using the power equation. Like before, we need to divide current by 100 and voltage by 10 because the VFD is actually scaling the output values. Therefore,

Power_In_Output := (Current_In / 100) * (Voltage_In / 10)

The other power values output from the VFD (Power_In) are the amount of power that the motor is creating. Since energy is lost in all mechanical processes, this value is less than the power required to drive the motor. Like before, we needed to multiply this by 10 because of the output scaling that the VFD requires.

Using the two power values, we can calculate the efficiency of the moter by using the following equation. Power_Out_Output is the useful power that we get from the motor and is considered power out in the equation. Power_In_Output is the amount of power required to run the motor. Efficiency can be calculated with the following equation:

$$E = \frac{P_{OUT}}{P_{IN}} \times 100\%$$

In all other cases, we want these values to be zero, so the rest of the code will be the following:

```
Else
    Torque_Out_Nm := 0;
    Power_In_Output := 0;
    Power_Out_Output := 0;
    Efficiency_Out := 0;
End_IF;
```

The last piece of structured text from this section is the horsepower of the system. In automation plants, many of the motors and components are rated in horsepower. It is sometimes beneficial to show the horsepower that you get from the motor. This is power that the motor is producing (Power_In_Output), and is different than the power required to run the motor (Power_Out_Output). We can use the horsepower equation to set up the code. Power must be in kW.

$$HP = 1.341 \times P$$

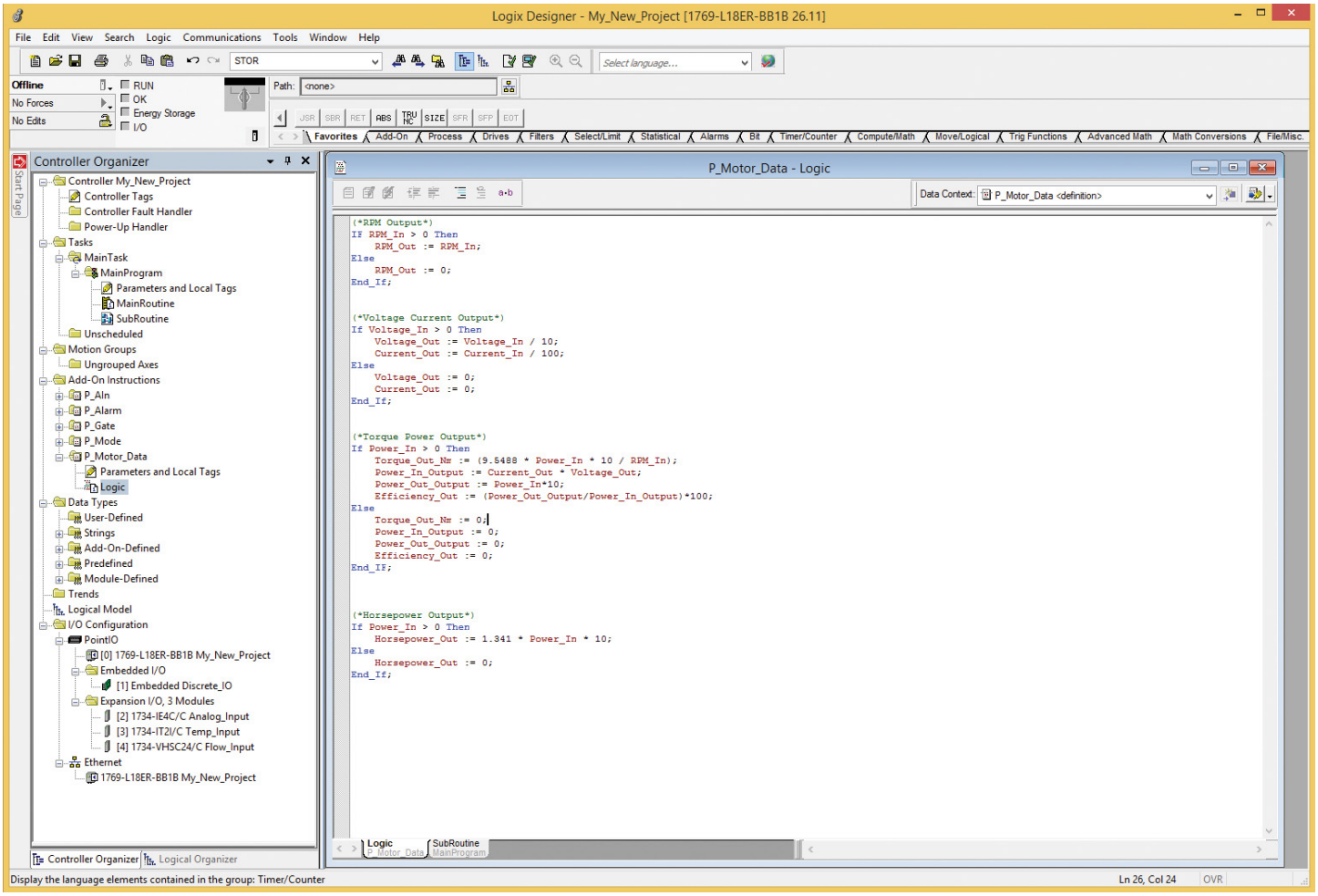
The structured text code would be the following:

```
(*Horsepower Output*)
If Power_In > 0 Then
    Horsepower_Out := 1.341 * Power_In * 10;
Else
    Horsepower_Out := 0;
End_If;
```

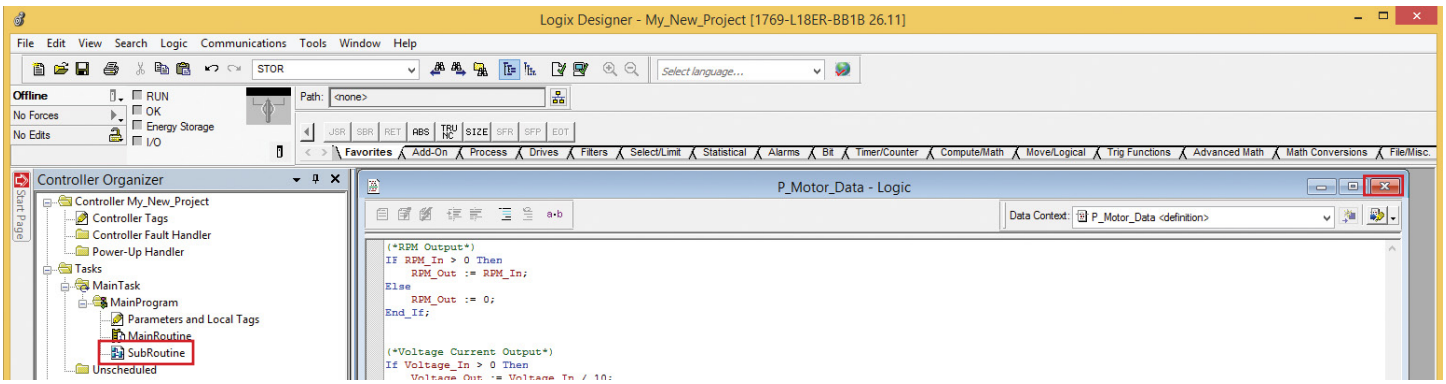
The result is the amount of useful work that the system can use over time. Note: this is different from the 3 HP rating on the motor.

The final structured text for the P_Motor_Data AOI is below:

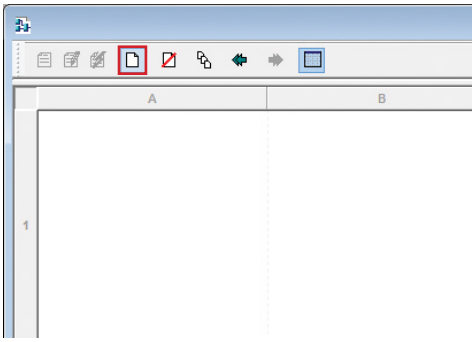
Keep in mind, this is not the only way to accomplish the task of monitoring motor data. This is setup with structured text to show how mathematics and equations can easily be handled with this form of programming.



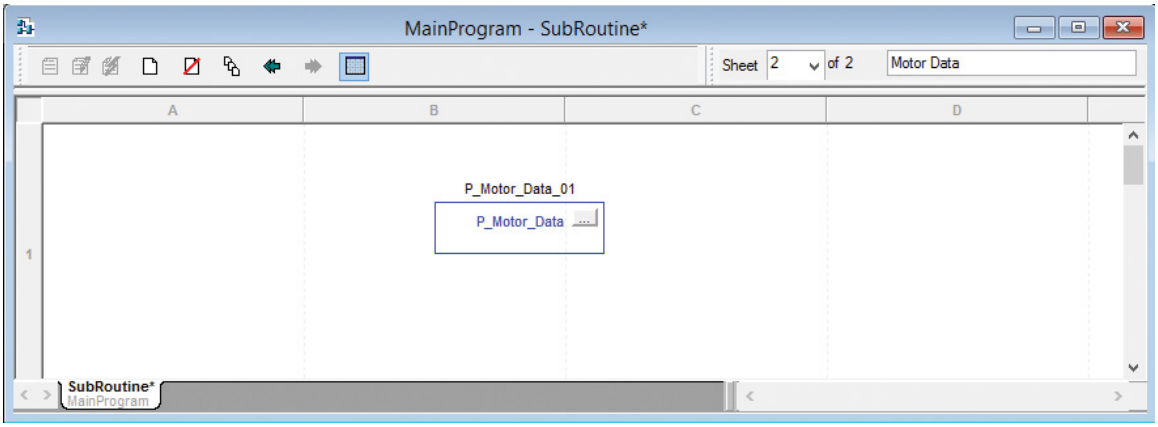
The next step in programming is to wire the P_Motor_Data AOI in the function block. Close the logic window for the AOI and open the function block diagram by clicking on the SubRoutine.



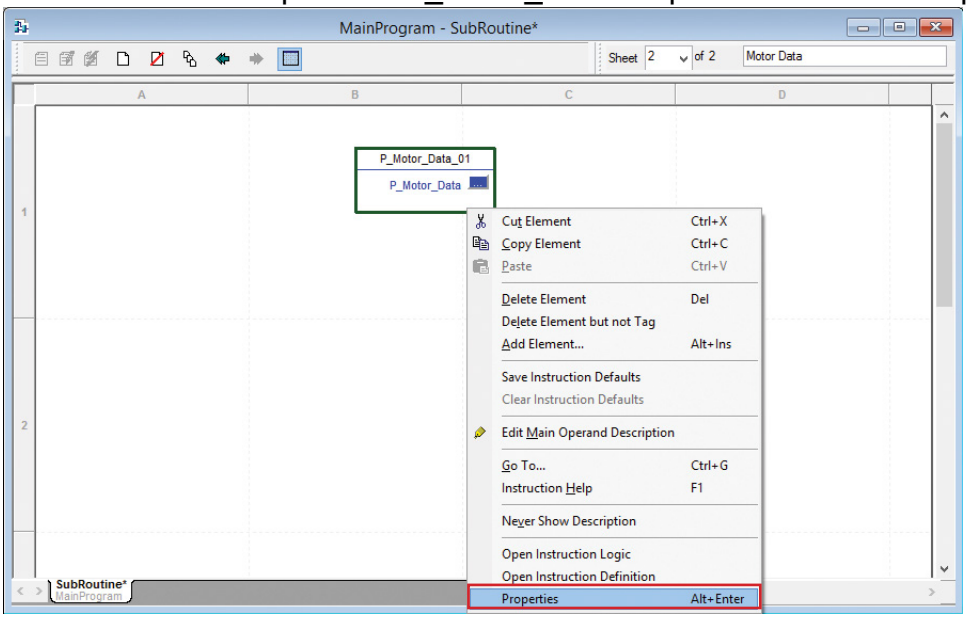
Open a new sheet in the function block diagram by clicking on the sheet image. Name the new sheet Motor Data.



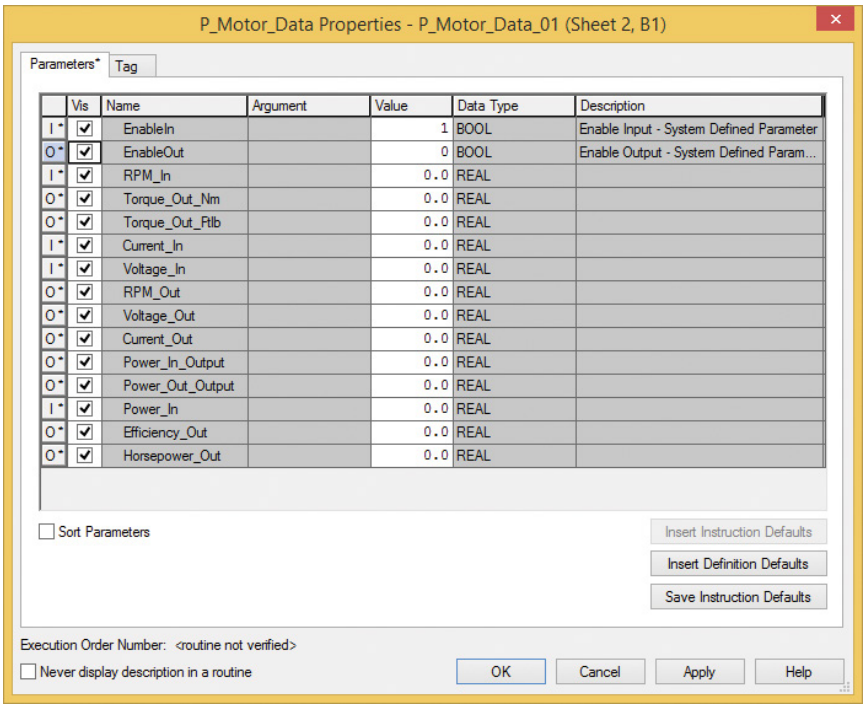
Navigate to the add-on toolbar above the sheet. P_Motor_Data should now show up as a button. Click on the button and P_Motor_Data AOI will be added to the new function block diagram. The AOI will look like it is empty. We need to enable all of the inputs and outputs that were customized with structured text.



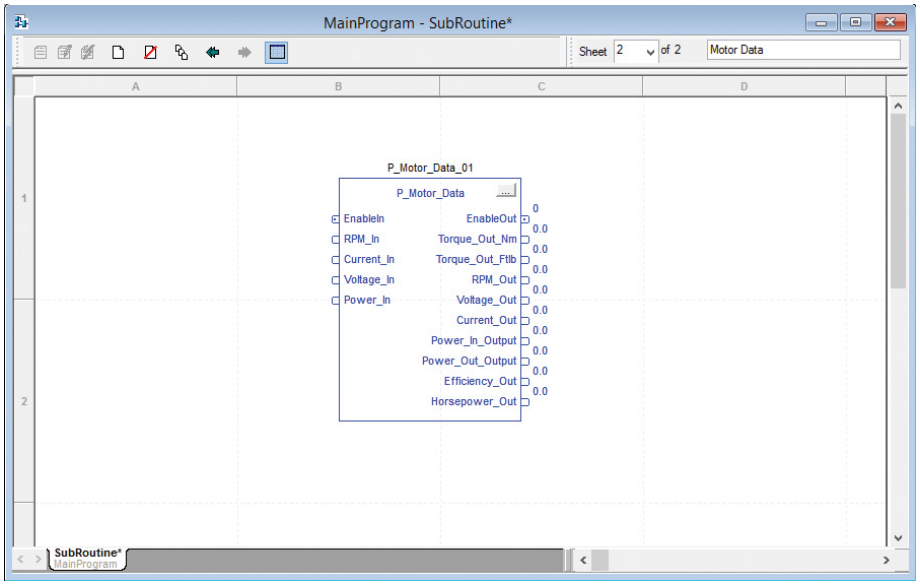
Right-click on the AOI and select Properties. P_Motor_Data Properties window will open.



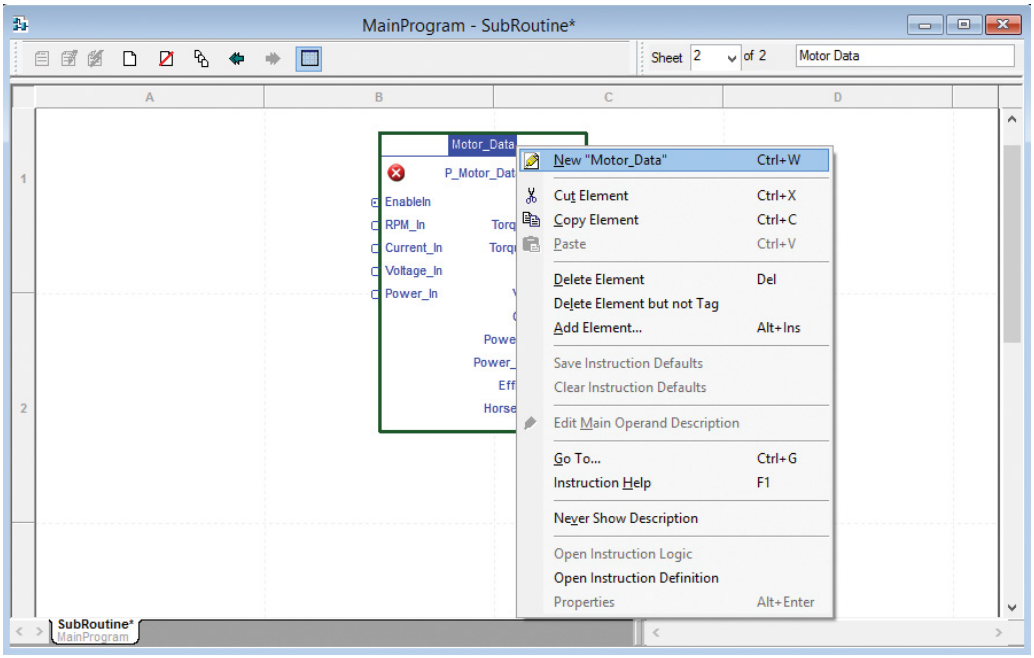
Enable the desired items by clicking on the checkboxes in the Vis column. Click OK to enable the items. Note: there is a column to the left showing I/O data. This may be useful for finding inputs or outputs quickly.



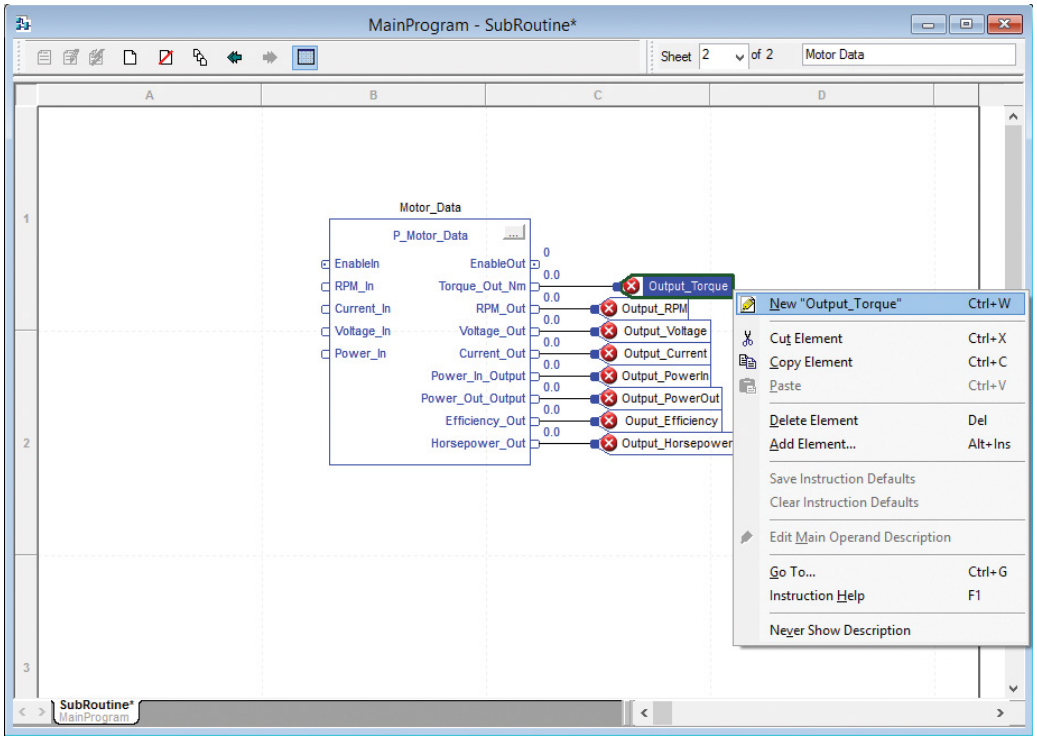
The AOI should now show the desired I/O data.



Rename the AOI Motor_Data and create a new tag Motor Data in the scope of My_New_Project. Delete P_Motor_Data_01 from MainProgram > Parameters and Local Tags.

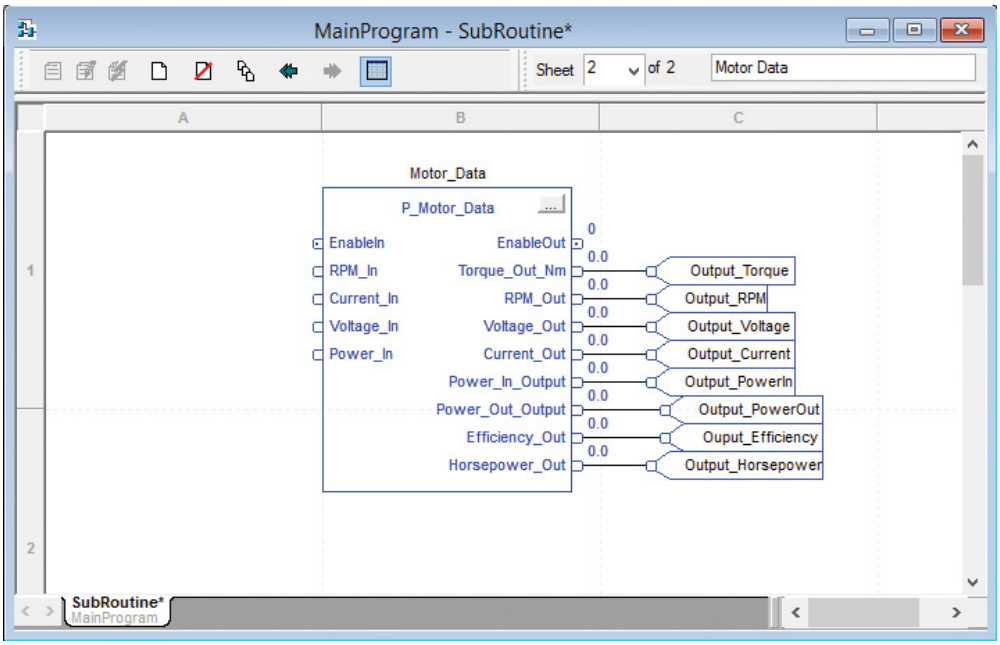


Drag 8 output tags from the toolbar to the fuction block diagram. Name all of the output tags as shown below. Right-click on each tag and add a new output tag to the scope My_New_Project. This allows the PLC access to the data calculated from the AOI (since AOI data is local).



Section 1: Studio 5000 Logix Designer Programming Environment

The finished AOI should look like the diagram below. It should also be free of errors. We are not going to add inputs at this time. Inputs will be added in Unit 6: VFD.



Skill Builder Section 1, Focus 4

1. Explain why structured text can simplify and replace function block diagram mathematics.

2. Describe how structured text can be used in conjuction with a function block diagram.

3. Why is structured text is similar to other programming languages?