612.1

The Local Process

SANS | GIAC CERTIFICATIONS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

All reference links are operational in the browser-based delivery of the electronic workbook.

ICS612.1 | ICS Cybersecurity In-Depth

SANS

The Local Process

SANS ICS612: This course is focused on the implementation and support of a secure control system environment through a hands-on, in-depth course that is designed to change how students engineer and support ICS environments.

Jeffrey Shearer
Mr. Shearer is a member of the SANS Institute ICS team focused on developing courseware in support of the ICS curriculum. Jeffrey also acted as a Subject Matter Expert (SME) for the Global Industrial Cyber Security Professional (GICSP) certification and is a content contributor for ICS NetWars. He also participates as an advisory board member for the ICS Security Summit and Training events.

Prior to joining SANS Institute, Mr. Shearer worked at Rockwell Automation for 23 years, where his most recent role was a Senior Security Architect for Rockwell Automation's Commercial Engineering group focused on network and security designs for Industrial Automation Control Systems (IACS) and Industrial Demilitarized Zones (IDMZ). Mr. Shearer was a contributing member of the Rockwell Automation and Cisco Systems Converged Plantwide Ethernet (CPwE) team, where he participated in architecture design and validation efforts. He also co-authored publications such as Deploying Industrial Firewalls within a Converged Plantwide Ethernet Architecture, Site-to-Site VPN to a Converged Plantwide Ethernet Architecture, and Securely Traversing IACS Data across the Industrial Demilitarized Zone.

Jason Dely
Jason Dely is responsible for leading the critical infrastructure and industrial control systems (ICS) security practice for Cylance. Prior to joining Cylance, Jason held many roles at Rockwell Automation, where he assisted clients with their research, design, integration, testing, and response activities across a variety of application, security, and infrastructure initiatives. During this time, Jason gained in-depth ICS product, protocol, and operational experiences that are invaluable when it comes to evaluating and building defenses within critical infrastructure organizations. His security passion over the past 18 years of experience with ICS is founded upon balancing business requirements against people, process, and technologies unique to each organization to ensure their operations are safe, reliable, and secure.

Tim Conway

Tim Conway is currently the Technical Director – ICS and SCADA programs at SANS, and is responsible for developing, reviewing, and implementing technical components of the SANS ICS and SCADA product offerings. Additionally, he performs contract and consulting work in the areas of ICS cybersecurity with a focus on energy environments. Recognizing the need for ICS-focused cybersecurity training throughout critical infrastructure environments and an increased need for hands-on training, Tim assisted in authoring, and instructs, the ICS curriculum's newest courses and ICS NetWars challenges. Outside of SANS, Tim continues to work on projects that blend cybersecurity, operations technology, and critical infrastructure protection with a focus on the energy sector.

Chris Robinson

Chris Robinson graduated from the United States Naval Academy with a B.S. in Computer Science and served over 6 years in the United States Navy. He then began his IT security career as a consultant for Booz Allen Hamilton before he attended graduate school full time at San Diego State University, earning an M.S. in Computer Science. Following graduation, Chris worked as Computer Scientist for the Navy and was an Adjunct Professor at San Diego's Mesa Community College. Chris then transitioned into ICS security, where he is currently an ICS Principal Consultant at Cylance, applying his expertise to various ICS cybersecurity projects to ensure solutions meet the needs of a modern industrial control system. Chris has learned firsthand the unique requirements and operational constraints for securing ICS environments. Chris currently holds and maintains multiple certifications, including the CISSP, OSCP, GICSP, GISP, GISF, and CEH. Chris teaches both the SANS MGT414 and MGT415 courses and currently resides in London, UK.

Contributor

Ted Gutierrez

Ted Gutierrez, CISSP, GICSP, and GCIH, is the ICS & NERC CIP Product Manager at the SANS Institute. Mr. Gutierrez was most recently the Director of Operations Technology & NERC Compliance at Northern Indiana Public Service Company (NIPSCO), where he was responsible for compliance to NERC 693 and CIP Standards and the support of the related operations technology systems. Mr. Gutierrez has more than 25 years of experience working in the electric utility, information technology, and manufacturing industries.

ICS612 Course Outline

- Section 1: The Local Process
- Section 2: System of Systems
- Section 3: ICS Network Infrastructure
- Section 4: ICS System Management
- Section 5: Covfefe Down!

This will be a full week with a tremendous number of hands-on labs and discussions in the classroom. Please ask questions as we progress through the week. You may find from one section to the next, that you are more or less comfortable with the topics or labs covered in any given section. Please communicate with the instructors if you are having any challenges or if you want to go deeper in the content or labs; also feel free to arrange time prior to or after class to continue working on the labs or to discuss concepts covered in class with the instructor.

ICS612 Section 1 Outline (1)

- Process Environment Familiarization
- Lab 1.1: Virtual Machine(s) Setup
- Lab 1.2: Student Kit Familiarization
- Programmable Logic Controller Programming
- Lab 1.3: PLC Programming and I/O Integration
- Process Interface
- Lab 1.4: Integrating Analog Input
- Lab 1.5: Local HMI Setup and Control
- Student Pod Integration
- Lab 1.6: Configure the Shared Pod Elements
- Lab 1.7: Connect Student Kits to the Shared Pod
- Lab 1.8: Process Interrupt through Student Kit
- Lab 1.9: Local Process Environment Mapping

This page intentionally left blank.

ICS612 Section 1 Outline (2)

- Process Environment Familiarization
- Lab 1.1: Virtual Machine(s) Setup
- Lab 1.2: Student Kit Familiarization
- Programmable Logic Controller Programming
- Lab 1.3: PLC Programming and I/O Integration
- Process Interface
- Lab 1.4: Integrating Analog Input
- Lab 1.5: Local HMI Setup and Control
- Student Pod Integration
- Lab 1.6: Configure the Shared Pod Elements
- Lab 1.7: Connect Student Kits to the Shared Pod
- Lab 1.8: Process Interrupt through Student Kit
- Lab 1.9: Local Process Environment Mapping

We will begin this section by familiarizing ourselves with the virtual machines, software, individual student kit equipment, shared local process elements, and the overall process in the front of the classroom.

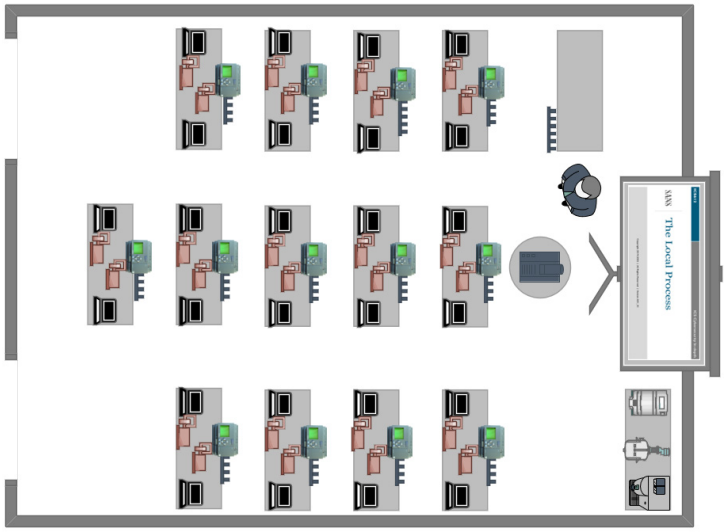
Process Environment Familiarization

Architecture
Virtual Machines
Student Kits
Shared Pods

This page intentionally left blank.

The ICS612 Classroom

- Media files – Contain multiple VMs and configurations
- Individual student kit
- Shared Pod at desk
- Common classroom process components
- Interconnected and interdependent systems
- Built, supported, attacked, and defended by you



The individual student kits have been prewired and soldered, so if you are having local issues with operational functionality, please let us know so we can troubleshoot the problem with you. The shared Pods have been prewired, staged, and verified through preclass validation steps. You will need to share the local Pod with your desk/table partner and conduct some labs jointly as there are some steps that can only be performed one at a time; otherwise, they would require considerable time to restore back to a known state after the first student completed certain steps to allow the second student to perform them again. We have highlighted these joint steps and special attention should be paid to them.

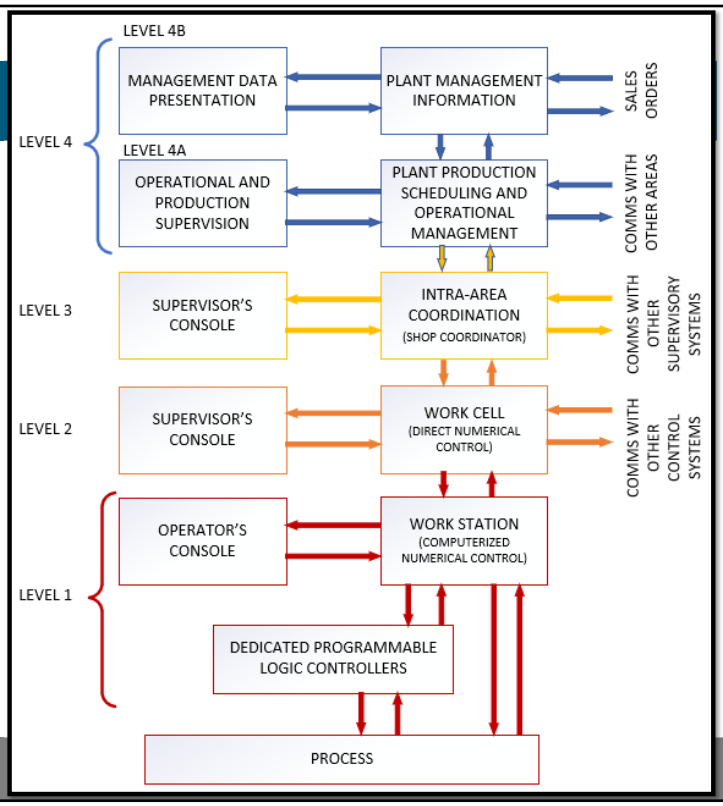
Thinking about ICS

- Know the mission
- Engineer/safe the process
- Build defensible systems
- Establish program vision
- Defend and respond
- Recover and restore

In our SANS ICS courses we typically have all students introduce themselves and state their company, background, and what they are hoping to learn in the course. As we work through this course and the hands-on labs, the course authors also want to introduce a way to think about various ICS environments that you may encounter throughout your career. It is absolutely essential to understand the overall mission that the ICS environment is there to support and enhance. It is also critical to understand the safety impacts that can occur if the control system fails or is misused in a manner that creates a human health or equipment damage risk. Based on a solid understanding of the operational mission and the safety risks present, this course will focus on architecting defensible systems, operating and maintaining those systems over time, and responding to incidents or failures.

Purdue Model – Intent

We see the start of the Purdue Model in 1989 from an ISA paper edited by Theodore J. Williams titled "A Reference Model for Computer Integrated Manufacturing (CIM): A Description from the Viewpoint of Industrial Automation"



This paper defined who needed information and how information needed to flow reliably between the various systems and users of those systems. A difference that you will see frequently when looking at reference models is the breakout of levels – there is inconsistency in the number of levels, what the level numbers are, and where devices are placed in reference to how they are being utilized.

Reference:
https://www.amazon.com/Reference-Model-Computer-Integrated-Manufacturing/dp/1556172257

Model Adaptation and Adoption

- Used in ANSI/ISA-95 for developing an automated interface between enterprise and control systems
- Picked up by ANSI/ISA-99, the standard for securing industrial control systems in the Operational Technology (OT) domain of organizations.
- We've used an interface mapping model for the backdrop of a security model
  - Sometimes this causes the model to be criticized

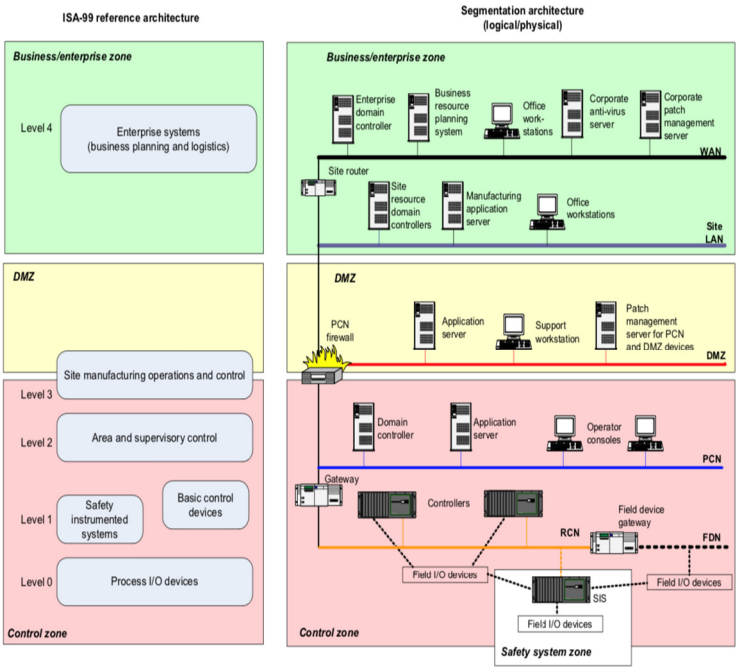


Figure A.8 – Reference architecture alignment with an example segmented architecture

The Purdue Model has been leveraged by standard bodies like ANSI/ISA-88 Batch Control, IANSI/SA-95 Enterprise – Control System Integration, and ANSI/ISA-99 Industrial Automation and Control System Security because it has identified the foundational assets and roles found within a manufacturing environment.

Because the Purdue Model has been used and modified by these standards bodies, it is widely recognized by IT and Operational Technology (OT) professionals. The popularity of the model and the common understanding of the levels has caused the model to be a starting point for all types of manufacturing environment discussions, including control system security. There are debates about the relevance of this model as it applies to manufacturing security, but one should remember that gaining a common understanding of a complex subject like ICS and manufacturing security needs to start with a common reference architecture to guide the discussion. While the model isn't perfect for all security discussions, it does provide the common backdrop for IT and OT professionals to start discussions about Plants, Area, Cells, Units, Lines and the equipment found within these logical containers. The Purdue Model and the many standards that use this model give us a documented reference model onto which we can pin our assets and talk about the data that flows between the levels, who consumes the data, and how we can secure the assets and data that move up and down the Purdue Model levels.

Model Adaptation and Adoption

- Used in ANSI/ISA-95 for developing an automated interface between enterprise and control systems
- organizations.
- We've used an interface mapping model for the backdrop of a security model
  - Sometimes this causes the model to be criticized

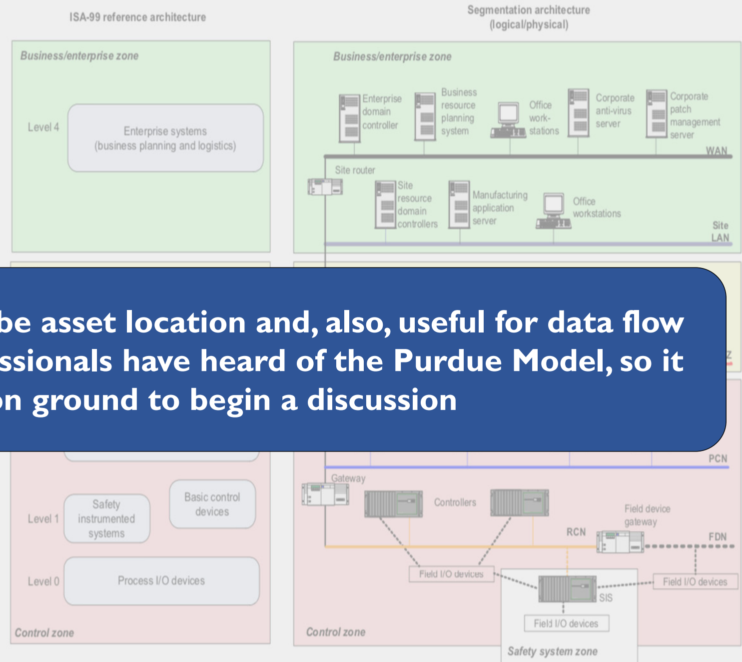
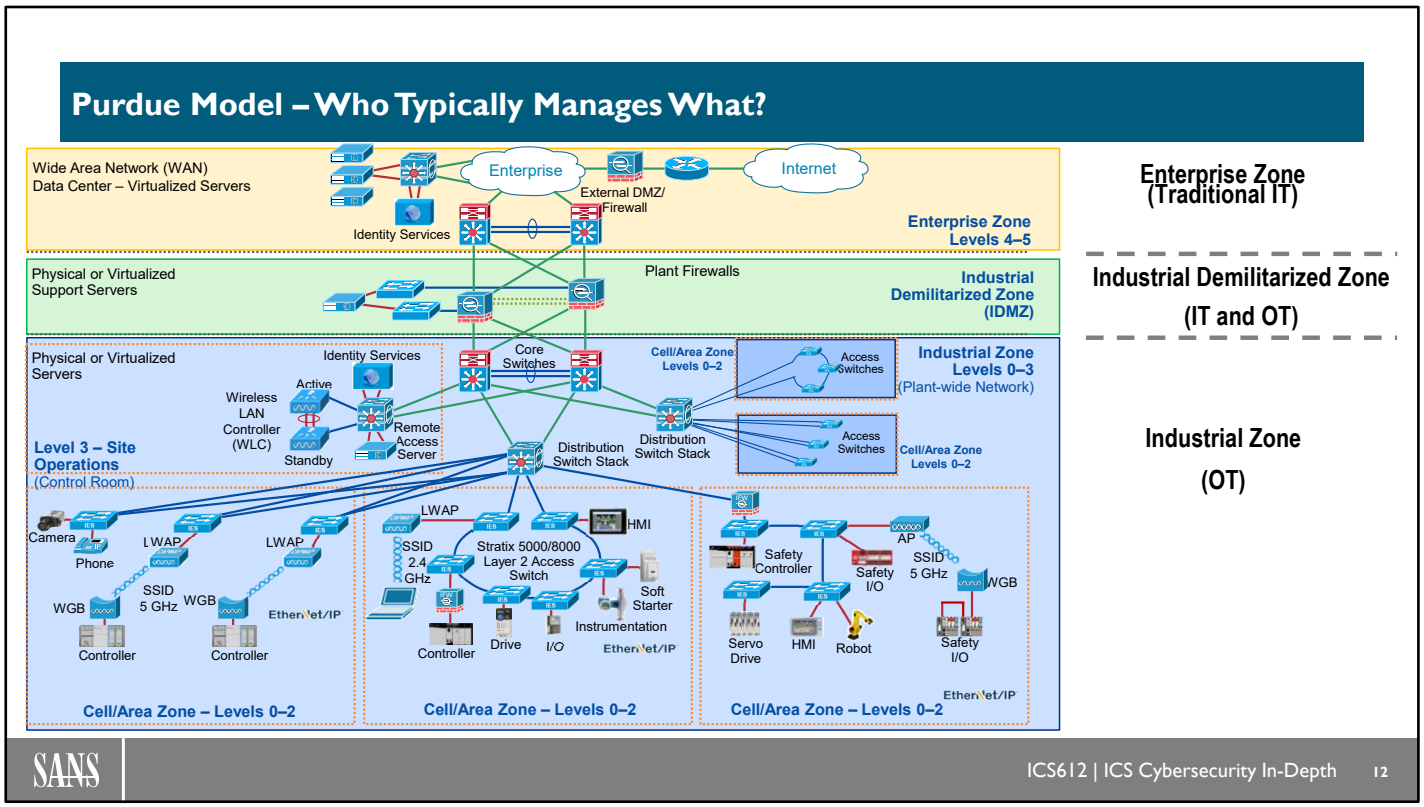


Figure A.8 – Reference architecture alignment with an example segmented architecture

The Purdue Model is useful to describe asset location and, also, useful for data flow mapping. Most OT engineering professionals have heard of the Purdue Model, so it gives us some common ground to begin a discussion

SANS

This page intentionally left blank.



Enterprise Zone – Level 5: Enterprise
Level 5 is where the centralized IT systems and functions exist. Enterprise resource management, business-to-business, and business-to-customer services typically reside at this level. Often the external partner or guest access systems exist here, although it is not uncommon to find them in lower levels (e.g., Level 3) of the framework to gain flexibility that may be difficult to achieve at the enterprise level. However, this approach may lead to significant security risks if not implemented within IT security policy and approach.

Enterprise Zone – Level 4: Site Business Planning and Logistics
Level 4 is where the functions and systems that need standard access to services provided by the enterprise network reside. This level is viewed as an extension of the enterprise network. The basic business administration tasks are performed here and rely on standard IT services. These functions and systems include wired and wireless access to enterprise network services such as the following:

Access to the Internet, access to Email (hosted in data centers)

- Non-critical plant systems such as manufacturing execution systems and overall plant reporting, such as inventory, performance, etc.
- Access to enterprise applications such as SAP and Oracle (hosted in data centers)

Manufacturing Zone – Level 3: Site Manufacturing Operations and Control
Level 3, the site level, represents the highest level of the IACS. The systems and applications that exist at this level manage plantwide IACS functions. Levels 0 through 3 are considered critical to site operations. The applications and functions that exist at this level include the following: Devices found in Level 3 are often responsible for managing control plant operations to produce the desired end product. Applications, services, and systems that are found at this level include:

- Level 3 IACS network
- Reporting (For example: Cycle times, quality index, predictive maintenance)
- Plant historian

- Detailed production scheduling
- Site-level operations management
- Asset and material management
- Control room workstations
- Patch launch server
- File server
- Other domain services, e.g., Active Directory (AD), Dynamic Host Configuration Protocol (DHCP), Dynamic Naming Services (DNS), Windows Internet Naming Service (WINS), Network Time Protocol (NTP), etc.
- Terminal server for remote access support
- Staging area
- Administration and control applications

The systems and applications in Level 3 communicate with the systems in the Enterprise zone through an Industrial DMZ. Direct communication between systems in Manufacturing and Enterprise zones is discouraged. Additionally, systems in Level 3 may communicate with systems in Levels 1 and 0.

Cell/Area Zone – Level 2: Area Supervisory Control
Level 2 represents the applications and functions associated with the Cell/Area zone runtime supervision and operation. These include the following:

- Operator interfaces or Human Machine Interfaces (HMI)s
- WGB – Work Group Bridges
- LWAP – Lightweight access points
- Alarms or alerting systems
- Control room workstations.

Depending on the size or structure of a plant, these functions may exist at the site level (Level 3).

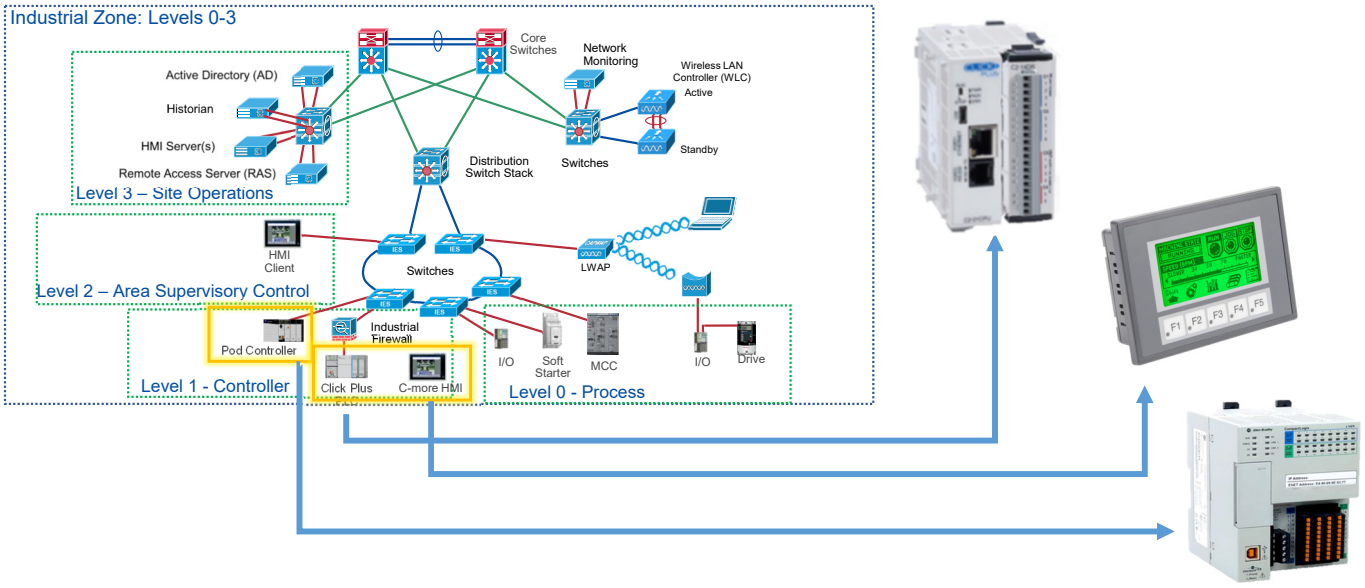
Cell/Area Zone – Level 1: Basic Control
Level 1 consists of controllers that direct and manipulate the manufacturing process, the key function of which is to interface with the Level 0 devices (e.g., I/O, sensors, and actuators). Historically in discrete manufacturing, the controller is typically a programmable logic controller (PLC). In process manufacturing, the controller is referred to as a distributed control system (DCS). The terms controller or programmable automation controller (PAC) refer to the multidiscipline controllers used across manufacturing disciplines. These include discrete, continuous process, batch, drive, motion, and safety controllers.

Cell/Area Zone – Level 0: Process
Level 0 consists of a wide variety of sensors and actuators involved in the basic manufacturing process. These devices perform the basic functions of the IACS, such as driving a motor, measuring variables, setting an output, and performing key functions such as painting, welding, bending, and so on.

Reference:

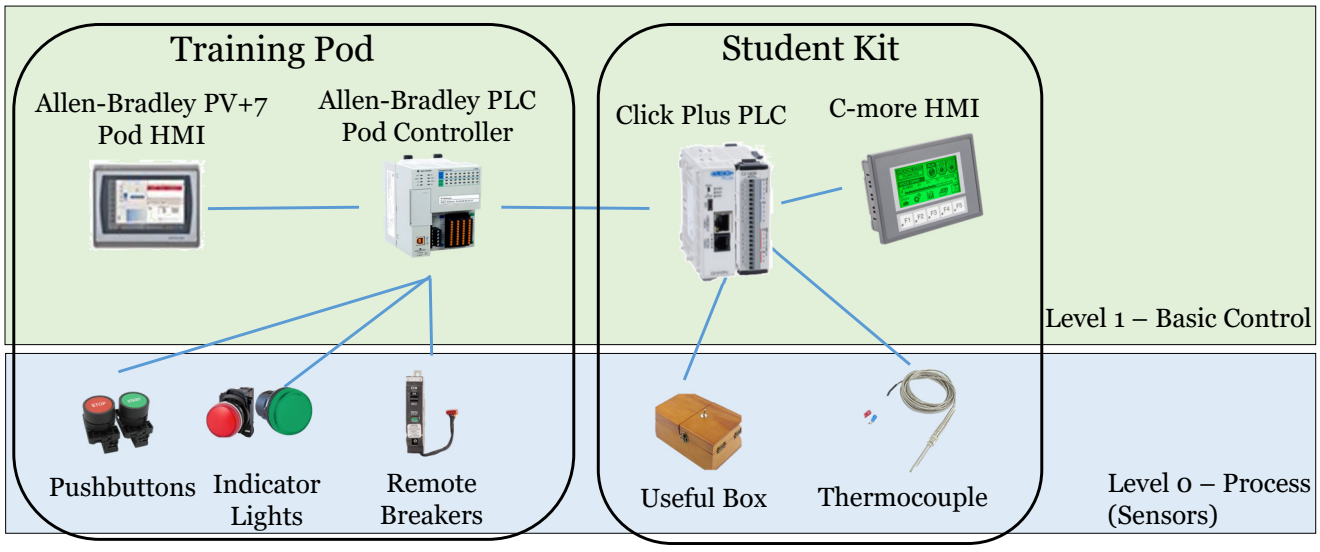
- Cisco and Rockwell Automation (2011). *Converged Plantwide Ethernet (CPwE) Design and Implementation Guide*. Cisco Systems, Inc. (n.d.). Retrieved from https://literature.rockwellautomation.com/idc/groups/literature/documents/td/enet-td001_-en-p.pdf

Reference – This Section's Labs



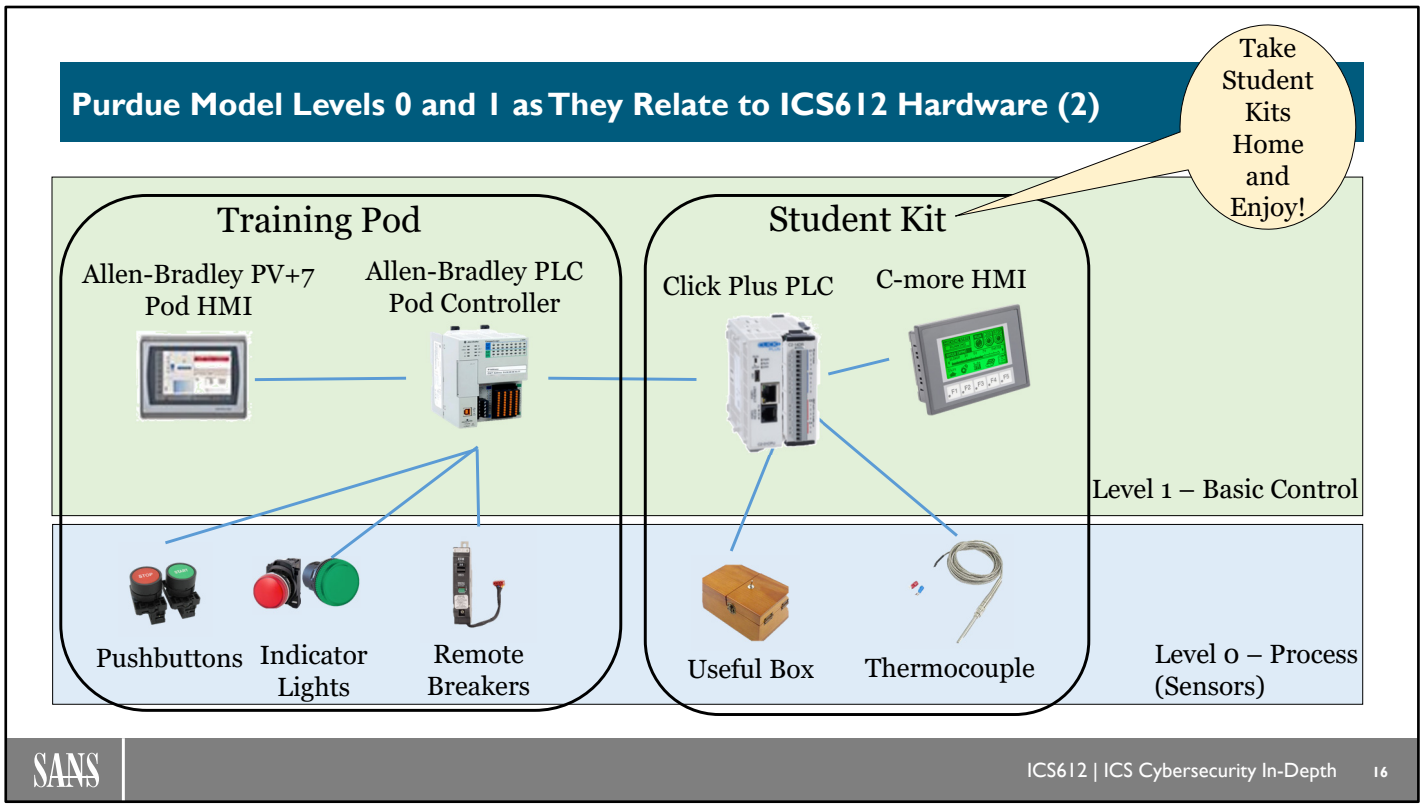
We will be starting from the bottom of the ICS architecture and working our way up to the enterprise connections throughout the week. Each student will leverage their own PLC and I/O modules to connect to external devices as well as their own local operator interface. Those individual process elements will then be initially connected to the shared student pods, which contain PLCs, operator consoles, industrial network switches, and local I/O and remote I/O components that will be leveraged later on in the course.

Purdue Model Levels 0 and 1 as They Relate to ICS612 Hardware (1)

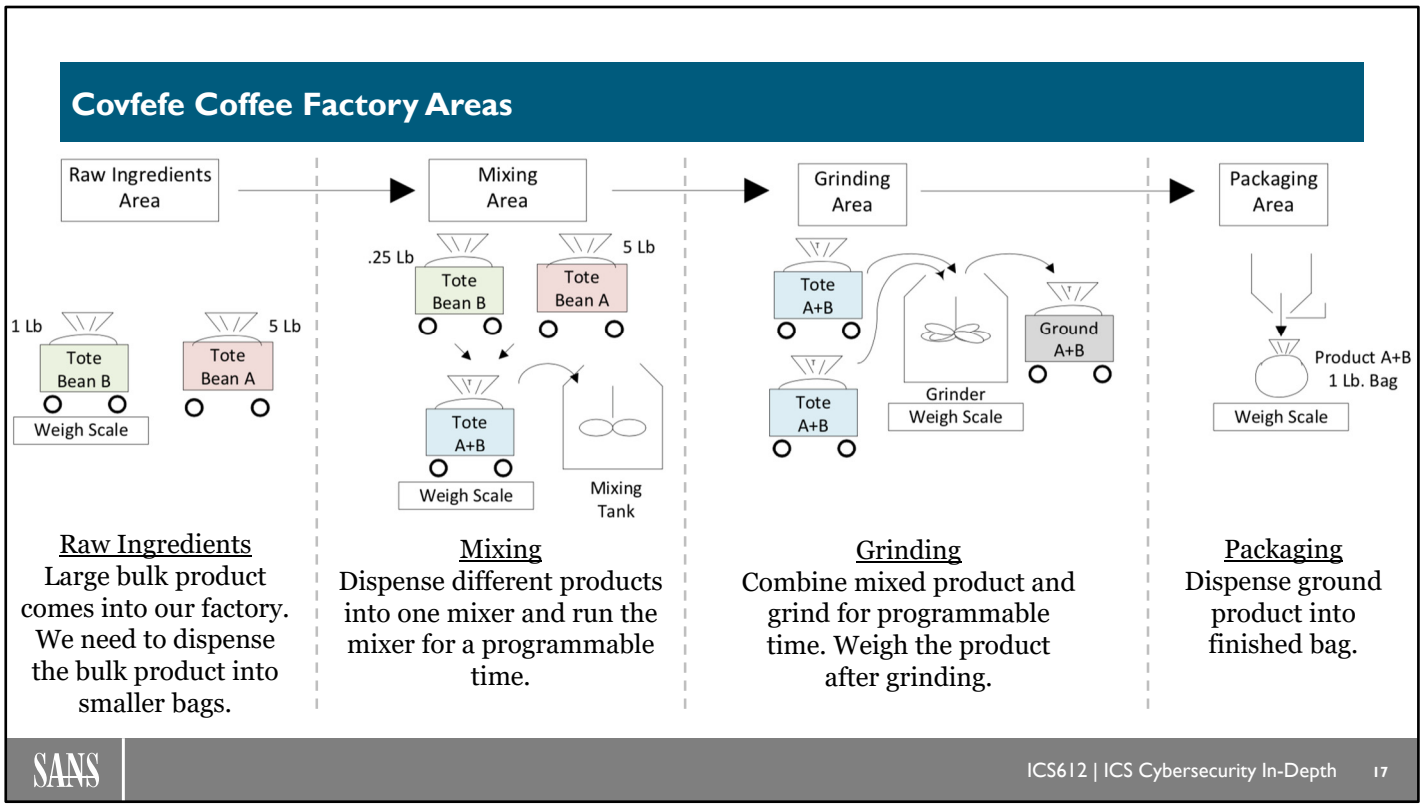


As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. On the left you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the right contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.



The items found in the student kit are yours to keep so your PLC and HMI learning can continue beyond our classroom setting. Enjoy!



The Covfefe Coffee factory is comprised of the Raw Ingredients, Roasting, Mixing, Grinding, Packaging, Warehouse Storage, Packaging, Shipping and Quality Control main areas. In our classroom setting we will be interacting with the Raw Ingredients, Mixing, Grinding, and Packaging areas.

The Raw Ingredients Area is responsible for taking the large pallets of raw coffee beans and dispensing them into smaller and more manageable bags.

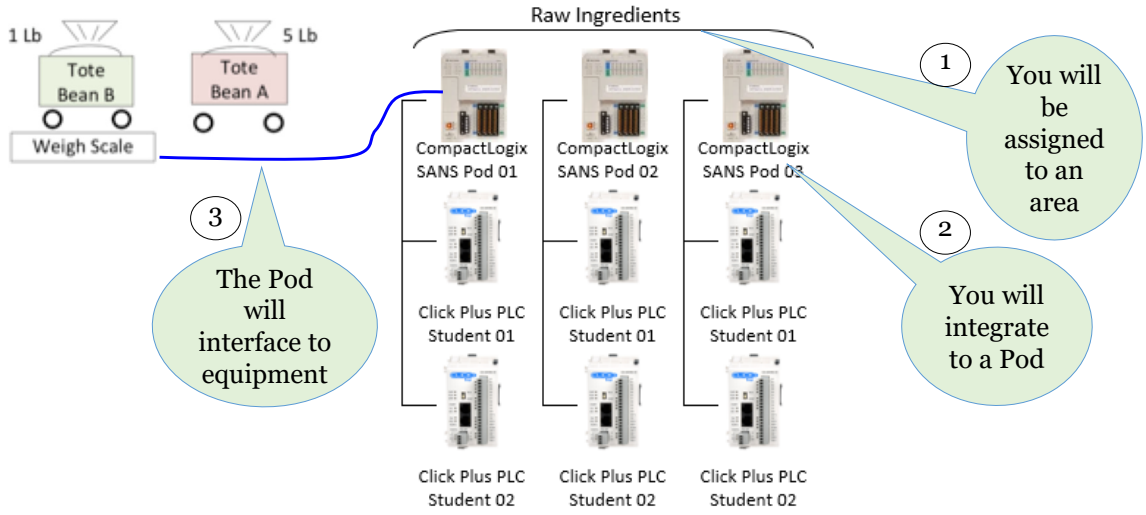
The Mixing Area is responsible for taking the smaller bags of already roasted beans and combining the different products according to a mixing recipe to create custom coffee blends. It is possible to blend custom flavor agents with the beans to create very groovy blends.

The Grinding Area is responsible for taking the mixed coffee beans and grinding the beans for a variable grinding time.

The blended grounds are then moved to the Packaging Area where the final product is dispensed into the final product bag or can.

We will also be participating in the Quality Control process by tasting the product and critiquing the blends.

Covfefe Coffee Factory Pod PLC and Student Kit Controllers



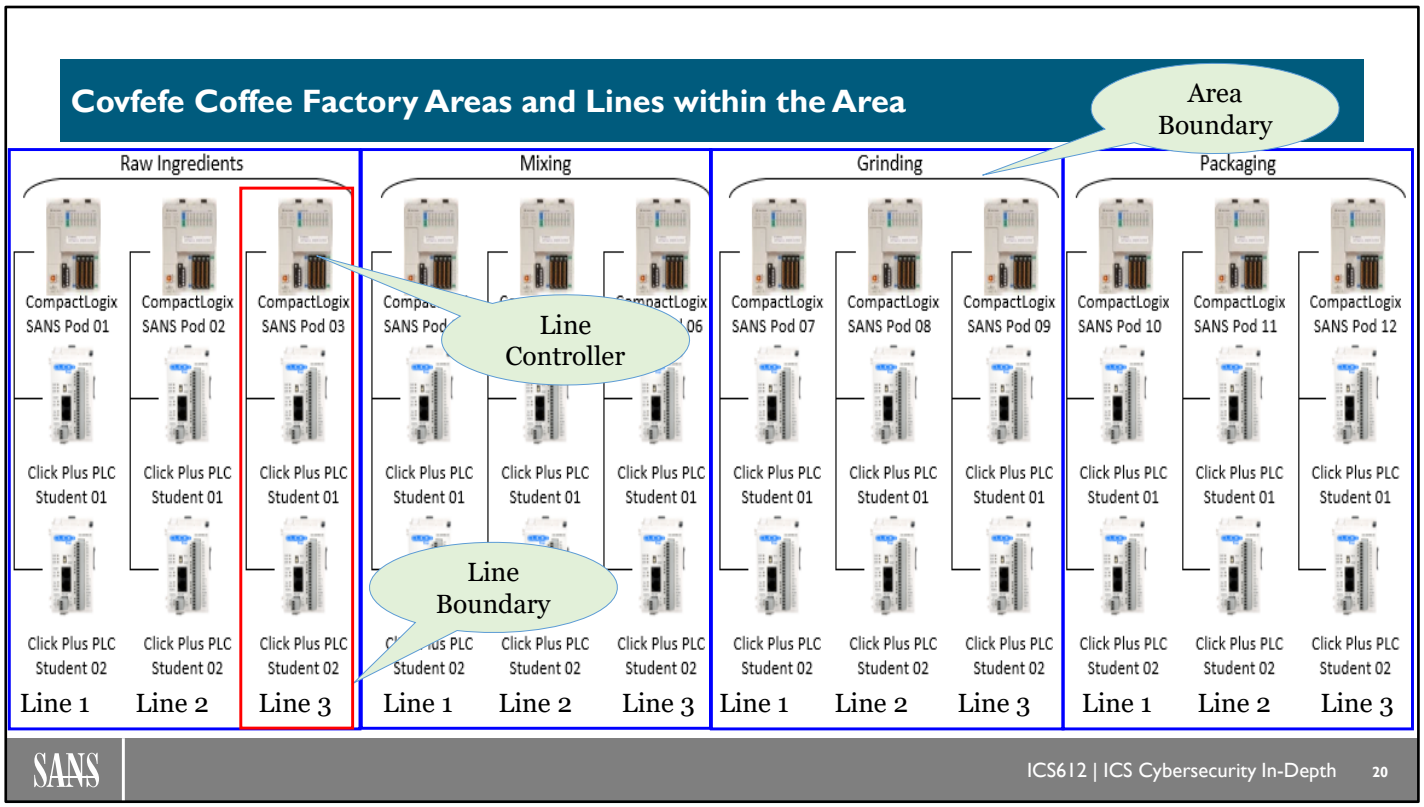
Each student will be assigned to an area of the coffee factory. The student's Click Plus PLCs will communicate to the Pod CompactLogix PLC to send product build requests. Product variables such as product types, product weights, and product mixing and grinding times will be sent from the Click Plus PLC to the CompactLogix Pod controller. The CompactLogix Pod controller will issue responses back to the student Click Plus PLCs, such as product available, product filling, product is being mixed, and packaging complete.

The Pod CompactLogix PLC will interact with the coffee factory equipment and report the status of the equipment back to the student PLC.

Covfefe Coffee Factory Overview

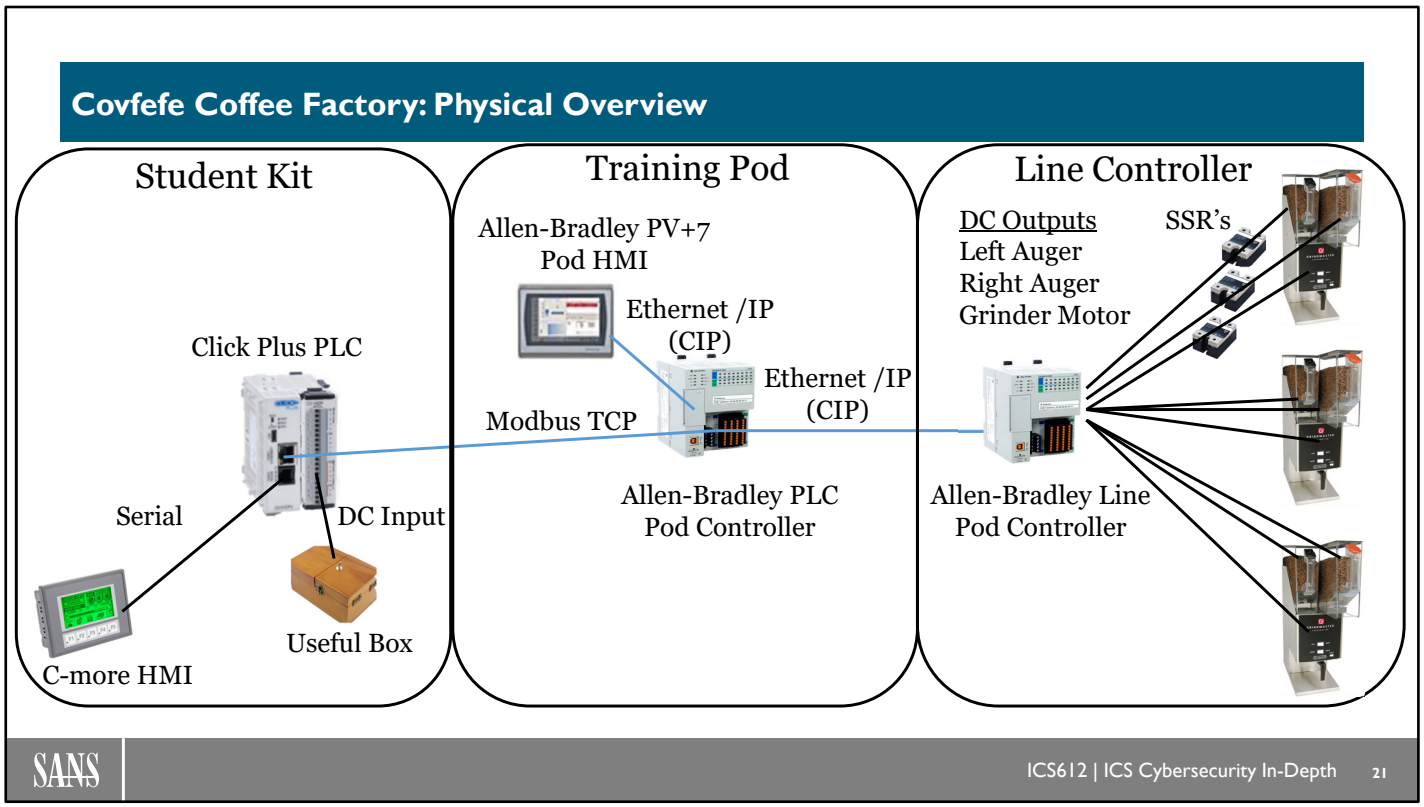
- Everyone will be assigned to an area of the coffee factory
- Everyone will use their Click Plus PLC to interact with the coffee factory
  - Lessons you learn during each lab will be used later to interact with the coffee factory
- Each Click Plus will communicate with an Allen-Bradley PLC located on each Pod to:
  - Request product types and weights
  - Request either weighing, mixing or grinding operations
  - Keep track of the bar codes as the product moves down the line
- Your C-more local HMI will be used to request and interact with the coffee factory

This page intentionally left blank.



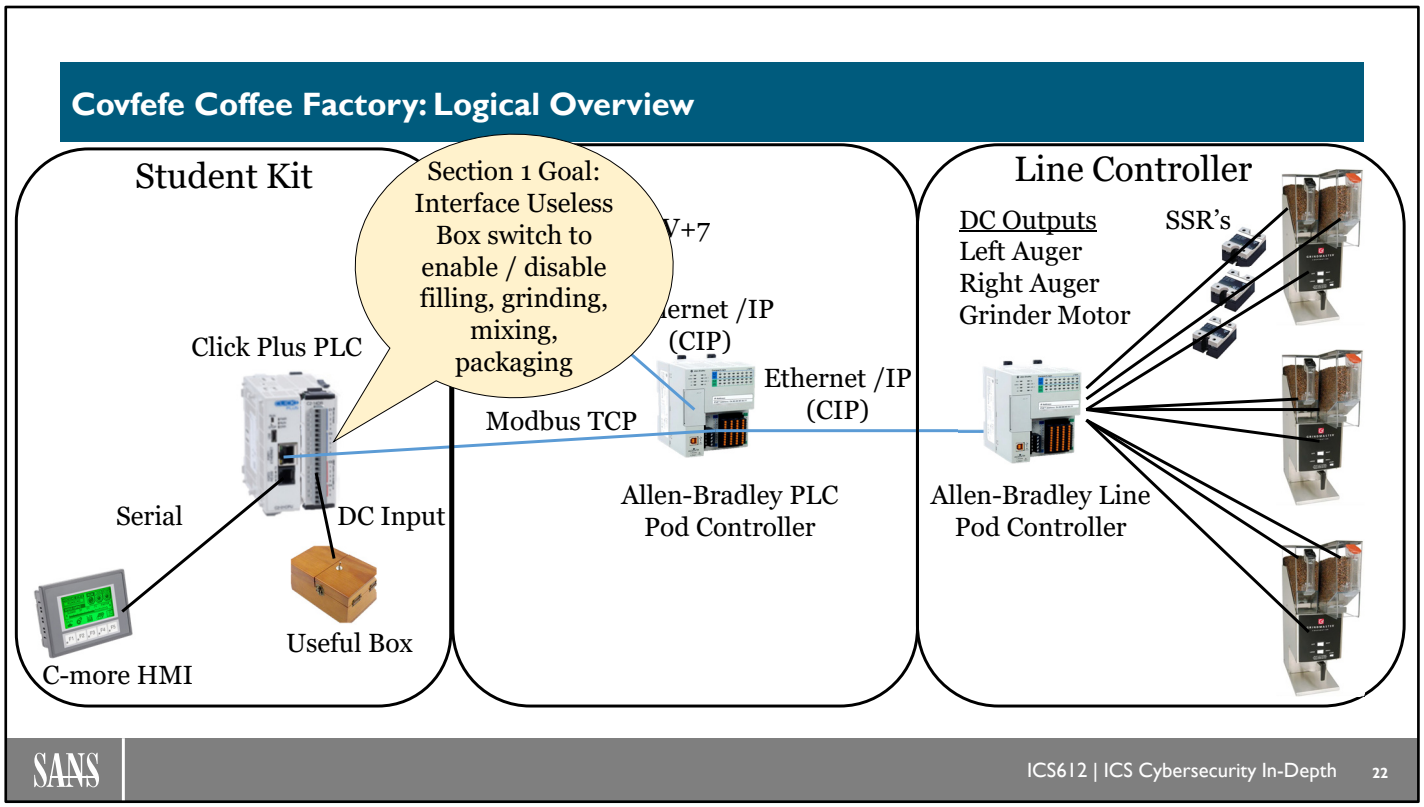
We've said the coffee factory has been divided into area boundaries called Raw Ingredients, Mixing, Grinding, and Packaging. Within the area boundary, we have an additional boundary called a "line". Each line is comprised of three student kits and one CompactLogix line controller. It is not uncommon within a manufacturing environment to have each line be able to produce a product and have a coordinating line controller. We have chosen this architecture in our classroom environment, and it is not an unusual architecture, especially as PLCs become more capable of handling more I/O and more capable of running more processes.

The downside to having a line PLC controlling more pieces of equipment without individual PLCs controlling each machine in a somewhat autonomous fashion is that if the line controller fails, it will cause more pieces of equipment to stop running and therefore have a bigger effect on production.



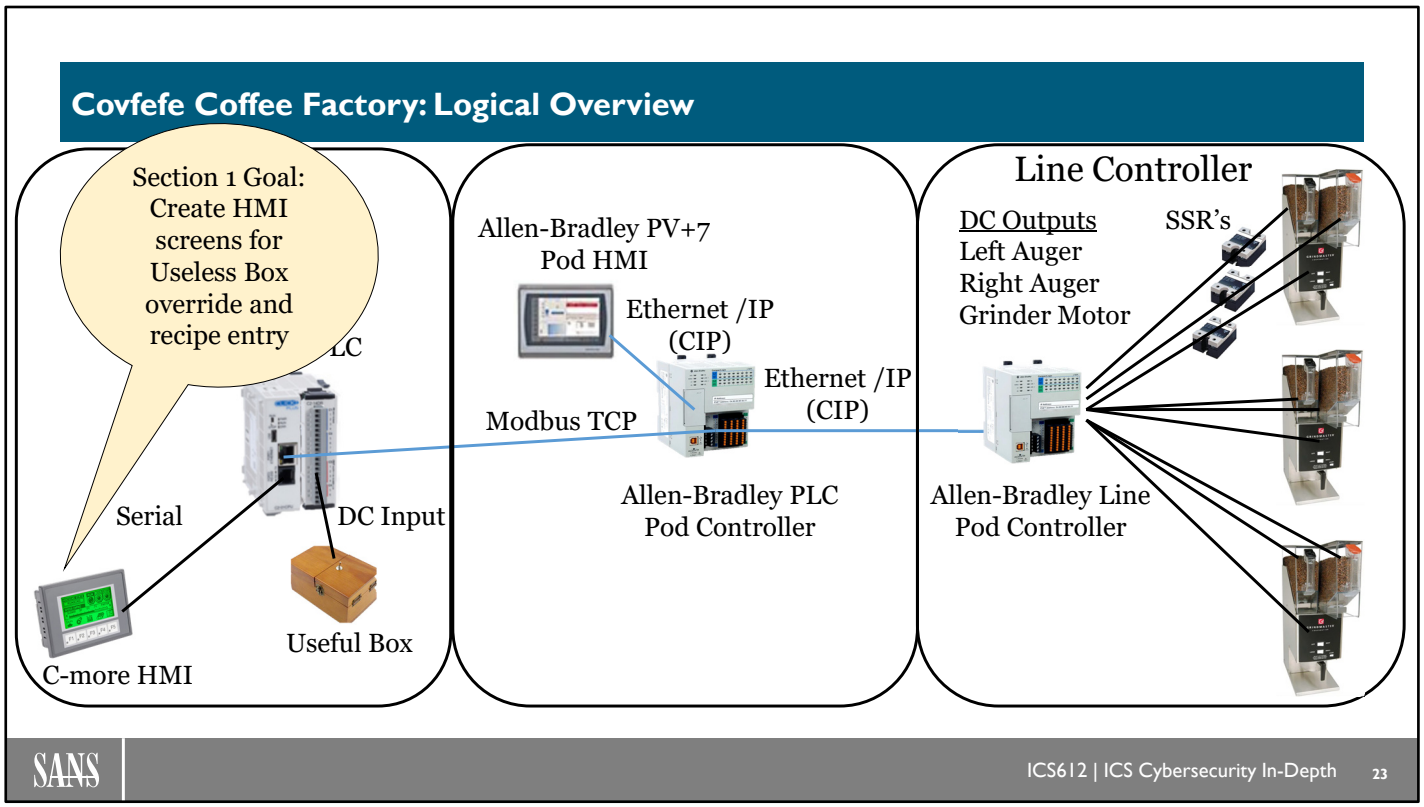
As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. On the left you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the right contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.



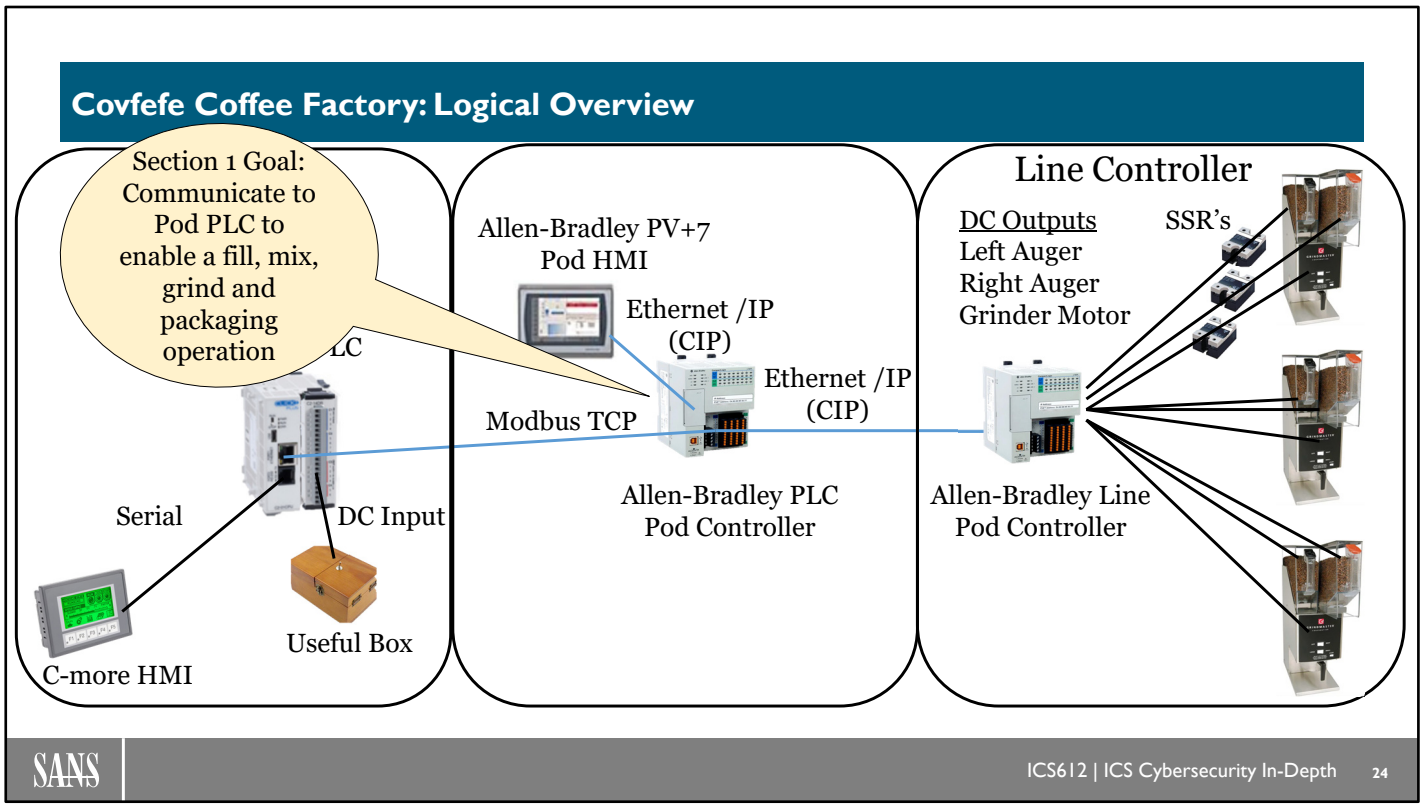
As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. On the left you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the right contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.



As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. In the middle of the slide, you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the left contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.



As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. In the middle of the graphic, you see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the left contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.

Engineering Workstation VM

- Necessary software, licenses, and configuration data
- The Engineering Workstation VM contains the following programs you will access during Section 1
  - Click Plus PLC programming software
  - C-more Micro HMI programming software
  - Rockwell Software (RS) Linx Classic
  - Rockwell Automation Studio 5000 PLC programming software
  - Rockwell Automation FactoryTalk View Machine Edition (ME) HMI programming software
  - Scanning Software

This page intentionally left blank.

Lab 1.1:Virtual Machine(s) Setup

Go to the Lab Workbook: Lab 1.1

This page intentionally left blank

Process Environment Familiarization Checkpoint 1.1

- At this point, you have copied and installed the ICS612 Windows 10 Enterprise virtual machine to your laptop.
- When working in ICS environments, having the necessary software, licenses, configs, and connectors can often be half the battle.
  - All of the necessary software to interact with and configure your student kit components is within the VM
  - All of the necessary software components and licenses are available on the VM to interact with and configure the ICS components in the classroom
  - Within many of the labs, you will be performing tasks and configuring various devices. On the VM there will typically be an "emergency break glass" configuration file for the lab, in case you cannot seem to get it.

This page intentionally left blank.

Student Kit

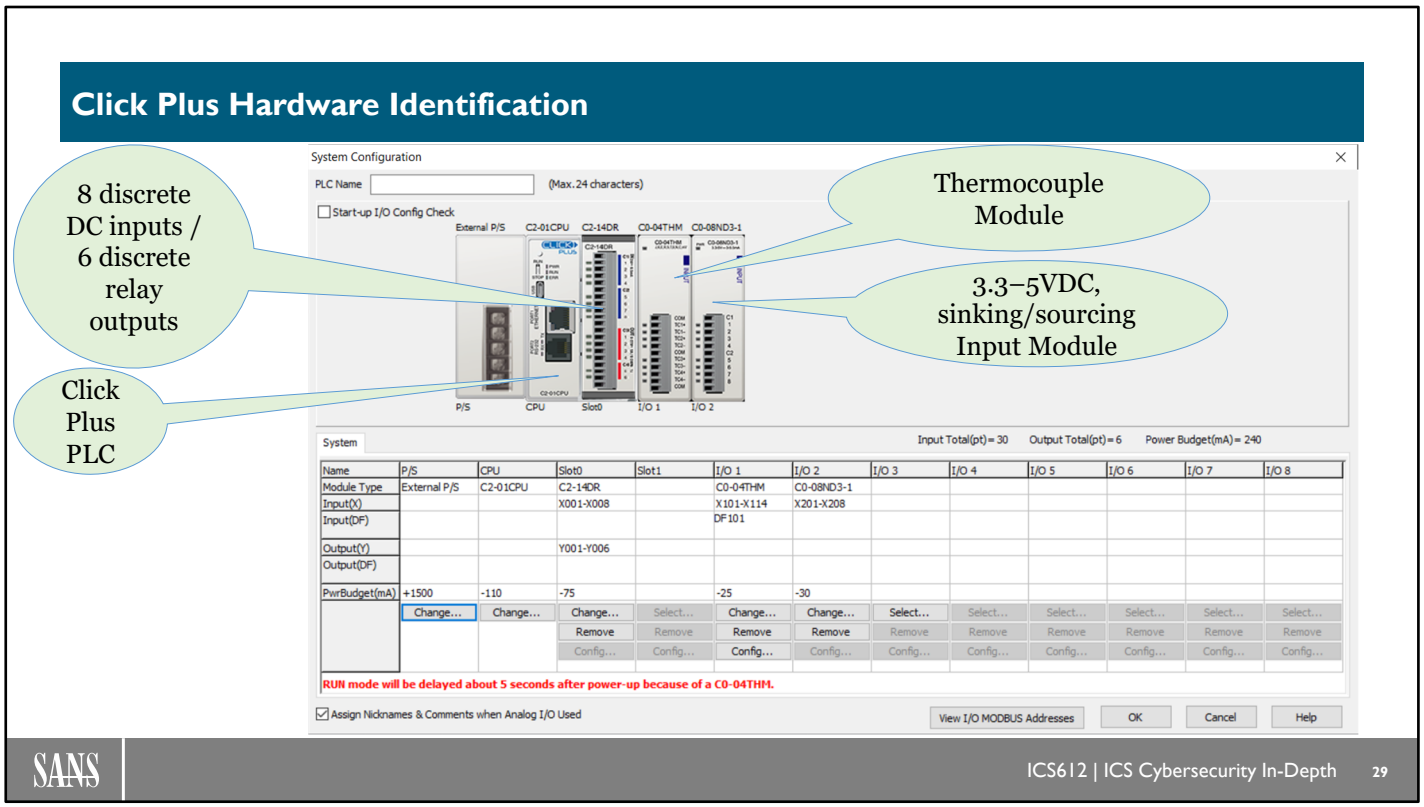
Contents:

- One (1) Click Plus PLC, (1) option slot, Ethernet, serial and microB-USB ports, no on-board I/O
- One (1) Click Plus 8-point discrete inputs and 6-point relay outputs.
- One (1) Click Plus 4-channel thermocouple input module, thermocouple
- One (1) Click Plus 8-point discrete input module, 3.3–5VDC, sinking/sourcing
- C-more Micro EA3 series touchscreen HMI
- USB/RS-232 programming cable assembly for use with C-more Micro panel
- Assembled Useful Box
- Kit Stand
- Network cables

The student kit is comprised of a Click Plus PLC made by Koyo. We have chosen this PLC because it has a very similar programming environment and similar functionality to what you will likely see within a factory environment. We chose to provide you with actual PLC hardware and software instead of a PLC facsimile so your hands-on experience would be representative of real factory experience.

We have also chosen to provide a C-more Micro Human Machine Interface (HMI) because the programming software and hardware is representative of many HMI systems found within a factory environment. The skills learned, such as tag assignments, communication configurations, function key assignment, and the general software programming environment, are quite typical of most HMI packages. While some higher-level and more costly HMI packages may have more functionality, the C-more HMI software will teach you the foundational nomenclature and techniques that will be transferrable to other HMI packages.

The Useless (Useful) box is also provided as a fun and visual learning tool to teach us about digital input and output interfacing with the Click Plus PLC. We will use the Useless Box in conjunction with a Click Plus PLC program to read the switch position and control the motor extend and retract functions.



Being able to identify the hardware of the Click Plus PLC system is the first step to understanding the PLC architecture. The PLC rack is comprised of the:

- Power Supply to provide +24VDC to the PLC processor
- Click Plus PLC Processor used to run the Ladder Logic programs(s)
- Thermocouple Module used to convert a thermocouple millivolt input signal into a digital reading
- 3.3–5VDC sinking/sourcing Input Module used to read the switch position from the Useless Box.

Power Supply

MODEL	input voltage range (Vac)	input frequency range (Hz)	output voltage nom (Vdc)	output current max (A)	output power max (W)	ripple and noise[1] max (mVp-p)	efficiency level[2] average[3] (%)	efficiency level[2] 10% (%)	no load power consumption typ (W)
SMI36-5	90 ~ 264	47 ~ 63	5	5.0	25	80	85.2	79.8	0.06
SMI36-9	90 ~ 264	47 ~ 63	9	3.34	30	90	88.2	81.4	0.07
SMI36-12	90 ~ 264	47 ~ 63	12	3.0	36	120	88.7	85.3	0.05
SMI36-15	90 ~ 264	47 ~ 63	15	2.4	36	150	89.1	84.3	0.06
SMI36-24	90 ~ 264	47 ~ 63	24	1.5	36	240	89.4	84.3	0.07
SMI36-48	90 ~ 264	47 ~ 63	48	0.75	36	480	91.2	89.1	0.07

Notes:
1. At full load, nominal input, 20 MHz bandwidth oscilloscope, each output terminated with 0.1 µF multilayer and 47 µF low ESR electrolytic capacitors.
2. CoC Tier 2 compliant
3. Average efficiency is measured at 25%, 50%, 75%, and 100% load.

INPUT

parameter	conditions/description	min	typ	max	units
voltage		90		264	Vac
frequency		47		63	Hz

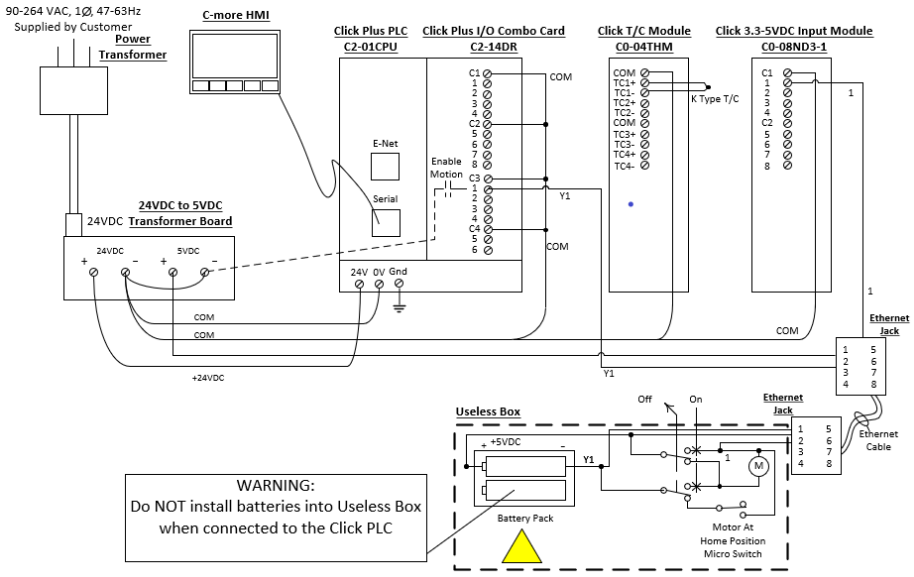
PROTECTIONS

parameter	conditions/description	min	typ	max	units
over voltage protection	output shut down			180	%
over current protection	output shut down, auto recovery			170	%
short circuit protection	output shut down, auto recovery				

The power supply has to be sized based for the Click Plus PLC based on the number of modules that will be placed in the rack.

It is always important when wiring a power supply to determine if the power supply will need protection from over current. The power supply that has been provided in your kit has over-current protection as shown above.

Logical Kit Wiring



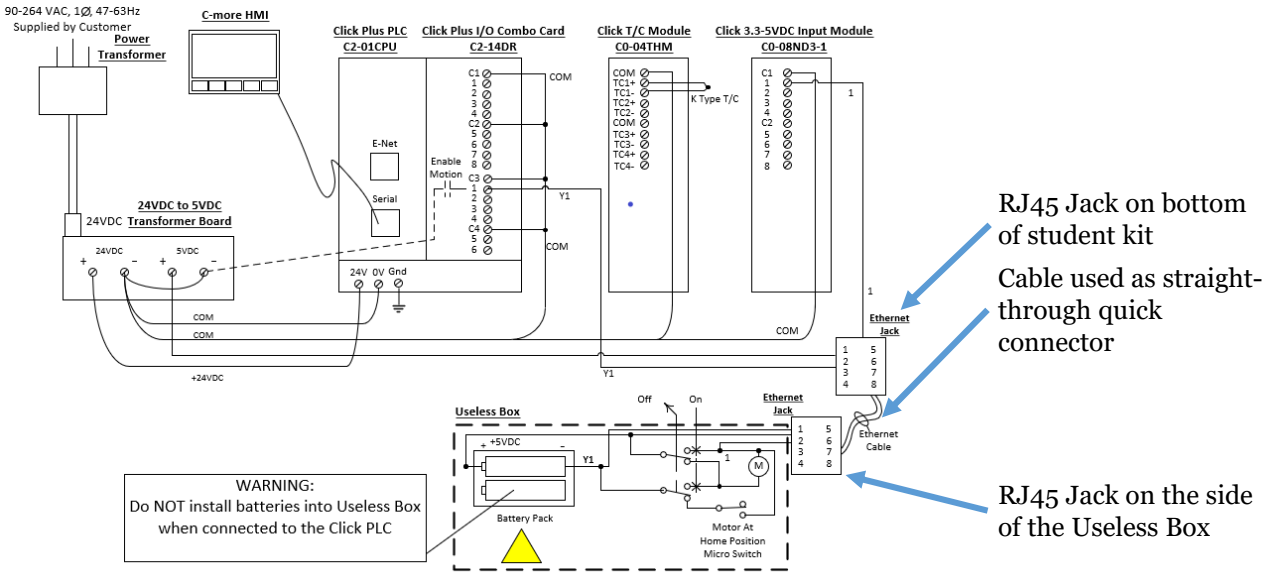
Looking at the left side of the drawing, we see a power supply that converts 90-264VAC to 24VDC and powers the Click Plus PLC processor. The 24VDC is also used to power the Useless Box through a 24VDC to 5VDC power transformer.

The Useless Box battery power pack is wired to 5VDC power supplied from the power transformer. Upon inspection you will find the negative lead of the battery pack is wired to an onboard Click Plus PLC contact output labeled Y1. When this contact is energized via Ladder Logic in a PLC program, the motor circuit will have a path to common through the Y1 contact. So, in order for the Useless Box motor to run, you must energize the Click Plus Y1 output.

When we look at the Useless Box switch input, we find it wired to the 3.3–5VDC input module. We wired the Useless Box switch to be recognized by the Click Plus input when it is in the “ON” position.

Please note, the Useless Box can be unwired from the Click Plus PLC and ran with batteries as designed if you want to run the Useless Box without being interfaced to the Click Plus PLC. WARNING: Do NOT install batteries while the Useless Box is connected to the Click PLC.

Physical Kit Wiring



The student kits and Useful Boxes have been prewired and terminated in RJ45 Jacks. This allows for students to quickly connect and disconnect the devices for the purposes of labs, shipping, or running either device independently. This also alleviates the need for wire, tools, soldering equipment, and many other necessary items within the classroom, but primarily it saves us time as a class.

WARNING: Do NOT install batteries while the Useless Box is connected to the Click PLC.

Lab 1.2: Student Kit Familiarization

Go to the Lab Workbook: Lab 1.2

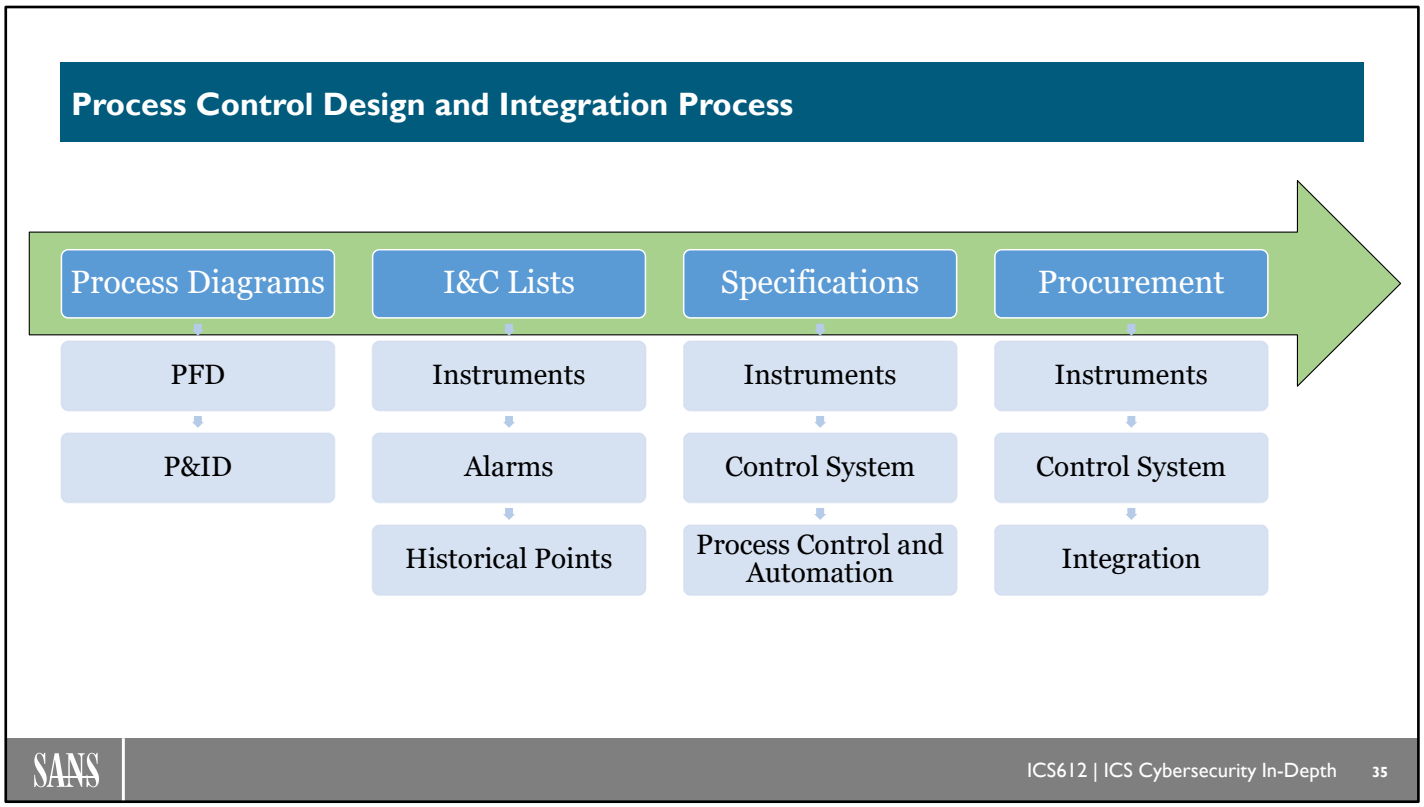
This page intentionally left blank.

Process Environment Familiarization Checkpoint 1.2

In previous courses or in your work experience you may have developed an understanding of what a PLC is and how it is used. At this point you are gaining exposure and developing capabilities in working with and setting up these devices. Working with the student kit, you have performed the following;

- Established serial and routable communications to the Click Plus PLC
- Verified correct firmware or updated to latest PLC firmware
- Configured the network settings of the device
- Downloaded an initial logic program into the Click Plus PLC
- Reviewed and understood the logic elements within the example flasher on/off program

This page intentionally left blank.



Process Control Design and Integration Process

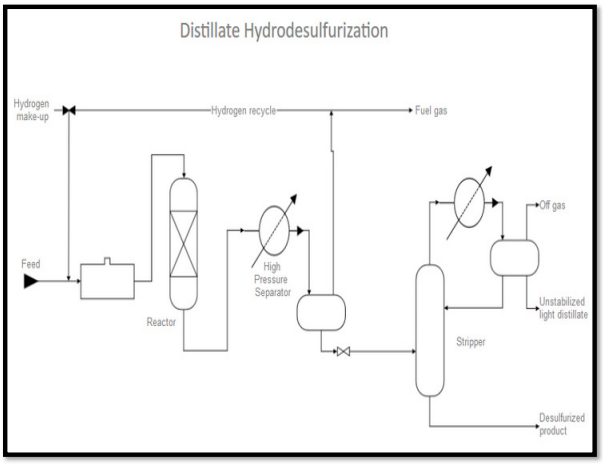
Now that we understand some basics about the devices and the wiring, we will take a look at the diagrams and lists that are generated by the engineers and designers that determine which devices will be used. We will also look at what point in the process these documents are generated.

Process Diagrams

Process Flow Diagram – PFDs are the basis for the instrument ranges

Process and Instrument Diagram – P&IDs are the basis for the human machine interface (HMI)
P&IDs depict the instruments

ISA S5.5 – Standard for graphic symbols used in process displays



https://www.smartdraw.com/process-flow-diagram/process-flow-diagram-software.htm

The basic diagrams called the PFD (Process Flow Diagram) and P&ID (Process and Instrumentation Diagram) give the basis for the scope of work. Each diagram typically covers one major piece of equipment and the ancillary equipment around that major piece. Sufficiently complex processes will have additional P&IDs and PFDs for support equipment such as steam, cooling water, or other utilities.

P&ID drawings establish machine communality, the "Tag" identification details the field devices, location, and function of instrument.

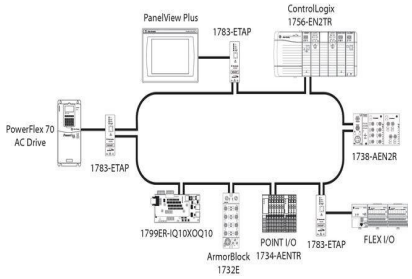
Standard Tags are identification that can be used with the drawings, programs, alarms, and documentation, instruction and navigation documents.

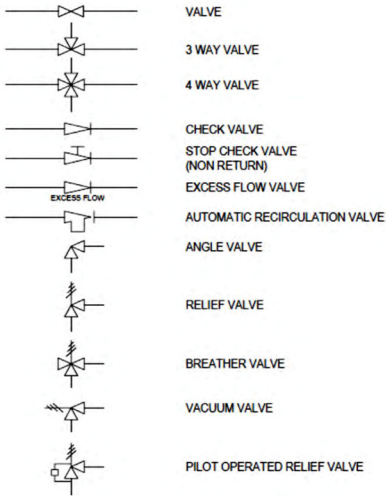
The diagrams can tell you important things about what is being monitored and controlled but not how or why it is being monitored or controlled. The diagrams, if documented properly, can indicate which instruments have interactions. For example, a flow device and a valve that are connected by dashed lines indicate a flow control loop. The dashed line may have another symbol on it indicating the signals are visible on an HMI (human machine interface) but not how the control actually occurs. The diagrams serve one other purpose: They are typically the inspiration for the operator displays. It is fairly common to find operating displays that look very similar in layout to the P&IDs.

Reference:
https://www.smartdraw.com/process-flow-diagram/process-flow-diagram-software.htm

Specifications

- Instrument
  - Make
  - Model
  - Calibration Settings
  - Standard Tagging and ID
- Control System
  - Hardware
  - Software
  - Layout / Connectivity
- Process Control and Automation
  - Controls
  - Automation





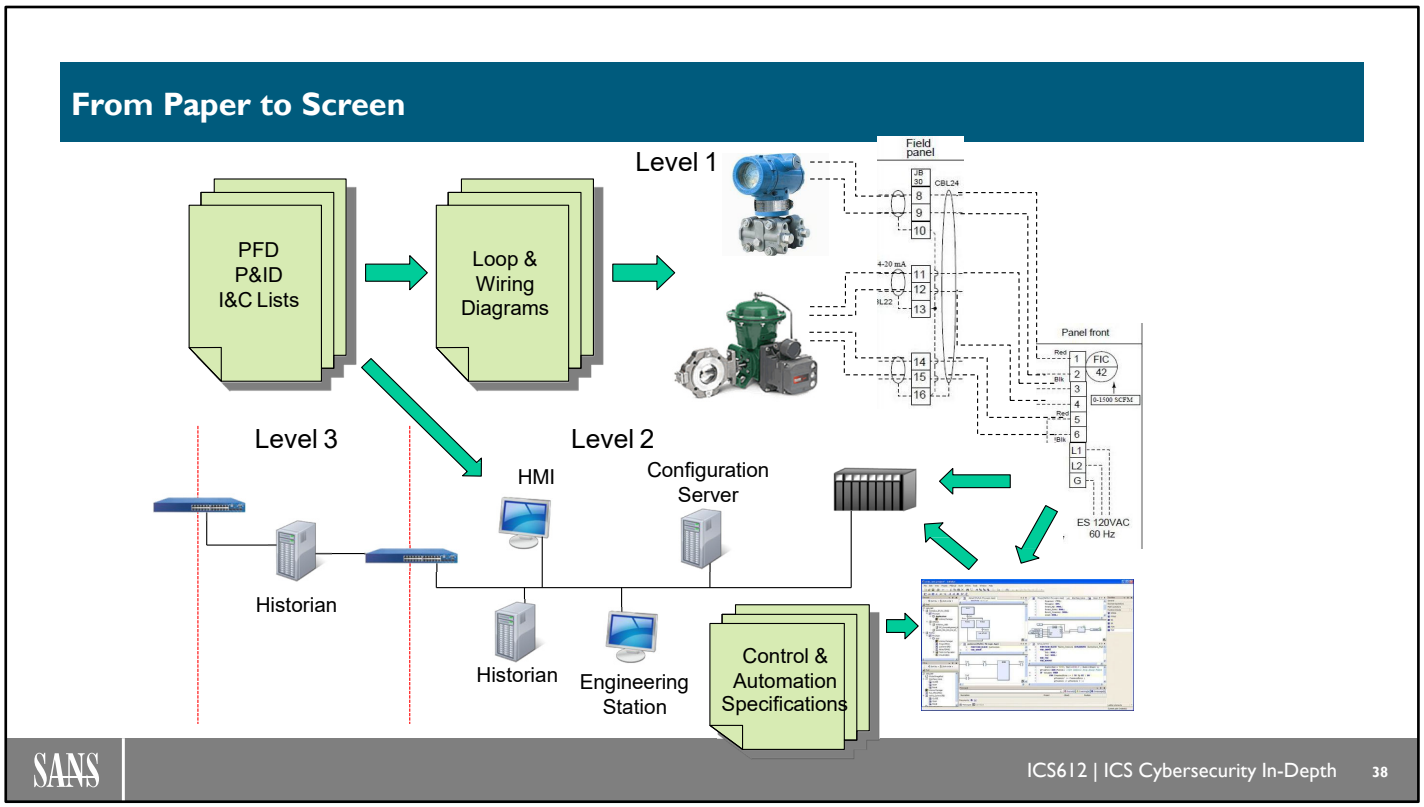
Once the instrument list has been established, the specifications can begin to be built. The specifications will include the manufacturer, model numbers, materials of construction, installation drawings, loop diagrams, wiring diagrams, and calibration routines for each instrument. This is also the time when the control logic is put to paper in an automation or control specification.

The control system specifications will indicate the make and model of the processors as well as the input and output (I/O) cards. Enclosures and cabinets that can hold the equipment will also be specified.

The automation and control specifications may be written at this point. A well written automation or control specification will not be based on a particular programming language or vendor. If no advanced controls are being performed and the only controls are simple loop controls, there may not be an automation specification.

Interlocks and permissives generally need a table or document to explain them. If a table is built, that may be the only documentation present. If a document is present, it may be incorporated into an overall control specification that indicates how loops are to be programmed, tuned, linked together, and how modes should be represented. User privileges are also outlined in both types of documents as the configuration of security controls is part of the system configuration.

If startup, runtime, and shutdown procedures are known and to be programmed this will go into an automation section or become an operations manual for the operation staff. The steps an operator would take to turn something on or put the process in a state it can produce products can be either manually performed or automatically performed by the controllers.



The drawings and diagrams are the beginning of the design that becomes the instrumentation and control component of a process. From these drawings come the lists of instruments that populate the process. Specifications are drawn up for both the control and automation of the process as well as the instrument materials of constructions, wiring diagrams, and cabinet or junction box designs.

The detailed design component is an interdisciplinary process. Engineers (such as chemical, petroleum, mechanical, civil, and electrical) work to produce design documentation that can be used for construction and operation. Each discipline looks for solutions to achieve the results the project is aimed at achieving. The process is something like this:

- Define the problem
- Gather pertinent information
- Generate multiple solutions
- Analyze and select a solution
- Test and implement the solution

The work each discipline does will impact the other disciplines. A pipe diameter changed from one inch to one-and-a-half inches will impact almost every instrument connection on that pipe as well as the pump that is moving materials through the pipe and the supports necessary to carry or hang that pipe from part of the processing structure.

ICS612 Section 1 Outline (3)

- Process Environment Familiarization
- Lab 1.1: Virtual Machine(s) Setup
- Lab 1.2: Student Kit Familiarization
- Programmable Logic Controller Programming
- Lab 1.3: PLC Programming and I/O Integration
- Process Interface
- Lab 1.4: Integrating Analog Input
- Lab 1.5: Local HMI Setup and Control
- Student Pod Integration
- Lab 1.6: Configure the Shared Pod Elements
- Lab 1.7: Connect Student Kits to the Shared Pod
- Lab 1.8: Process Interrupt through Student Kit
- Lab 1.9: Local Process Environment Mapping

This page intentionally left blank.

Programmable Logic Controller Programming

Click Plus PLC Programming
Wiring Review
Operational Goals

This page intentionally left blank.

IEC 61131-3 PLC Programming Languages

- Standard programming languages defined to drive vendors away from a proprietary language set
- Basic language types are:
  - Ladder Diagram (LD), graphical
  - Function Block Diagram (FBD), graphical
  - Structured Text (ST), textual
  - Instruction list (IL), textual (deprecated in 3rd edition of the standard)
  - Sequential Function Chart (SFC), has elements to organize programs for sequential and parallel control processing, graphical.

As with any standard, a group of interested parties come together to create an agreed-upon method for solving a problem. In the case of the IEC 61131-3 standard for Programming Languages it provides a common understanding and behavior of programming languages for a programmable logic controller (PLC). This standard is the baseline for PLC functionality but many vendors extend the standard by creating their own instructions and data types so they can bring competitive advantages to their PLC platform.

This standard does allow the user to solve their automation problems through common languages like Ladder Logic, Function Block Diagrams, Structured Text and Sequential Function Charts while being vendor independent. Said another way, if someone learns the basics of Ladder Logic, they will know basically how Ladder Logic from Vendor A, B and C will work. However, some vendors will supply specialized instructions that will not be found within the standard. Also, the behavior of the software programming environment is not described in the IEC 61131-3 standard so vendors will compete on “software ease of use” or “ability to add or remove” logic while the processor is still running other processes. Customers may select one vendor over another based on their language support or these other features.

Please note, it is not a requirement for a PLC vendor to support all of these languages. As you will see with the Click Plus PLC, only the Ladder Logic language is supported within the student kit Click Plus PLC.

Which PLC Programming Languages to Use? (1)

- Ladder Diagram (LD) is used for sequential applications
  - Used and understood by most OT engineers
  - Relay replacement language
- Function Block Diagram (FBD) is used for continuous processes
  - Used by DCS vendors like Honeywell, Emerson, etc.
  - It's hard to do sequential control so many times a PLC will be in the architecture to do the sequential logic while the DCS will supervise the operations

Ladder Logic is the traditional language for a PLC, as it is the best logical representation of the physical relay logic that is replaced and is the most widely used language inside the PLC. Ladder Logic is the best sequential control language to use for simple Boolean control. For instance, if the control requirements are to turn simple digital valves on and off during different machine phases, Ladder Logic will be the most widely deployed language to solve these types of control problems.

Function Block Diagram (FBD) is the most widely used language for continuous process control. It had its start from distributed control system (DCS) vendors. FBD programming is done by connecting blocks that represent specific functions. For instance, a Totalizer block will be used to provide an accumulated total of an analog input value. These function blocks are connected together to monitor inputs, control processes, and send output signals. Doing sequential control can be challenging so oftentimes sequential control algorithms are done in Ladder Logic.

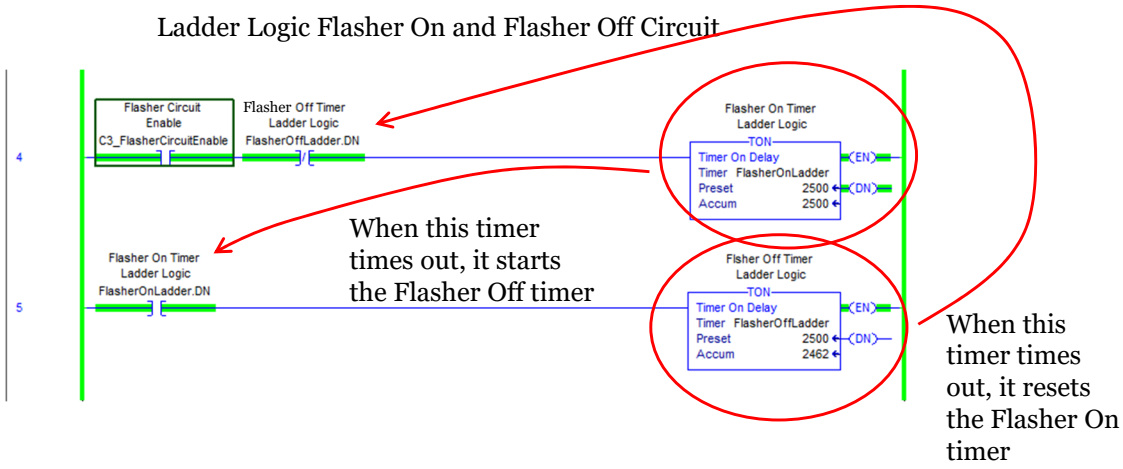
Which PLC Programming Languages to Use? (2)

- Structured Text (ST)
  - Typically adopted by high-level programmer because it's natural and fits their background
  - Can be rejected by typical Ladder Logic jockey because they don't understand the syntax
- Sequential Function Chart (SFC) is prevalent in batching software

Structured Text (ST) is a textual programming language that uses programming constructs like "If", "Then", and "Else" statements to define program execution. Persons familiar with computer programming will tend to pick this language as they find it similar to programming in other higher-level languages

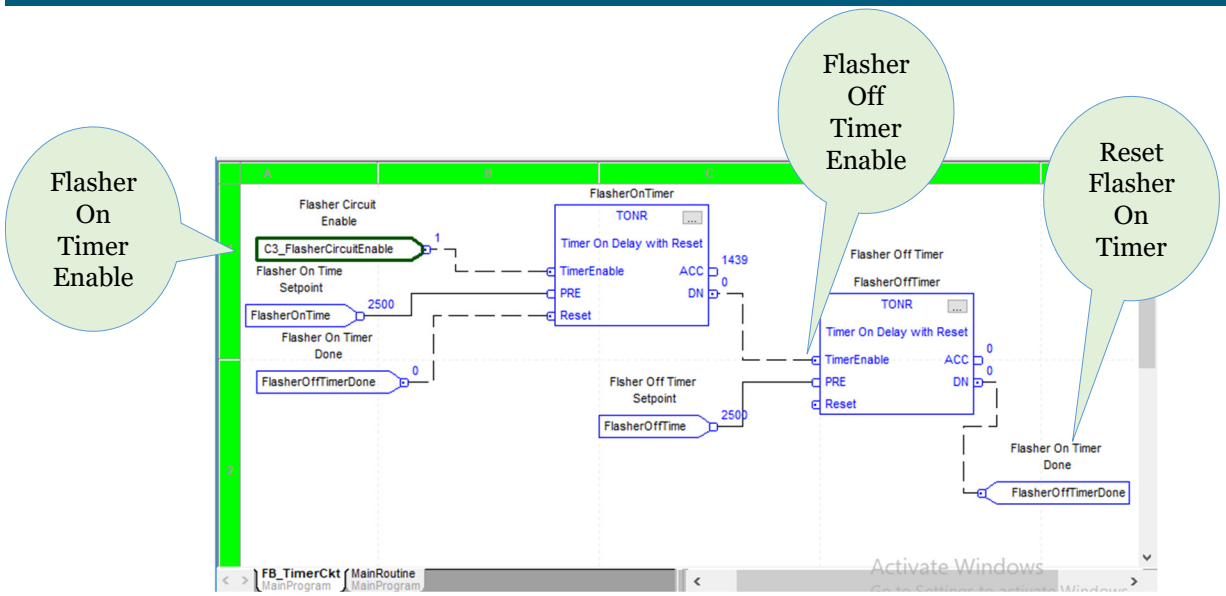
Sequential Function Chart (SFC) is popular with batching software as it uses steps and transitions in a graphical manner. While an operation is executing a step, the PLC or batch engine is monitoring a configurable set of conditions to transition to the next step. This is ideal if a process is a batch operation where a set of well-known steps must occur in a certain order for the product to be produced.

Flasher Circuit in Ladder Logic, Structured Text, and Function Block (1)



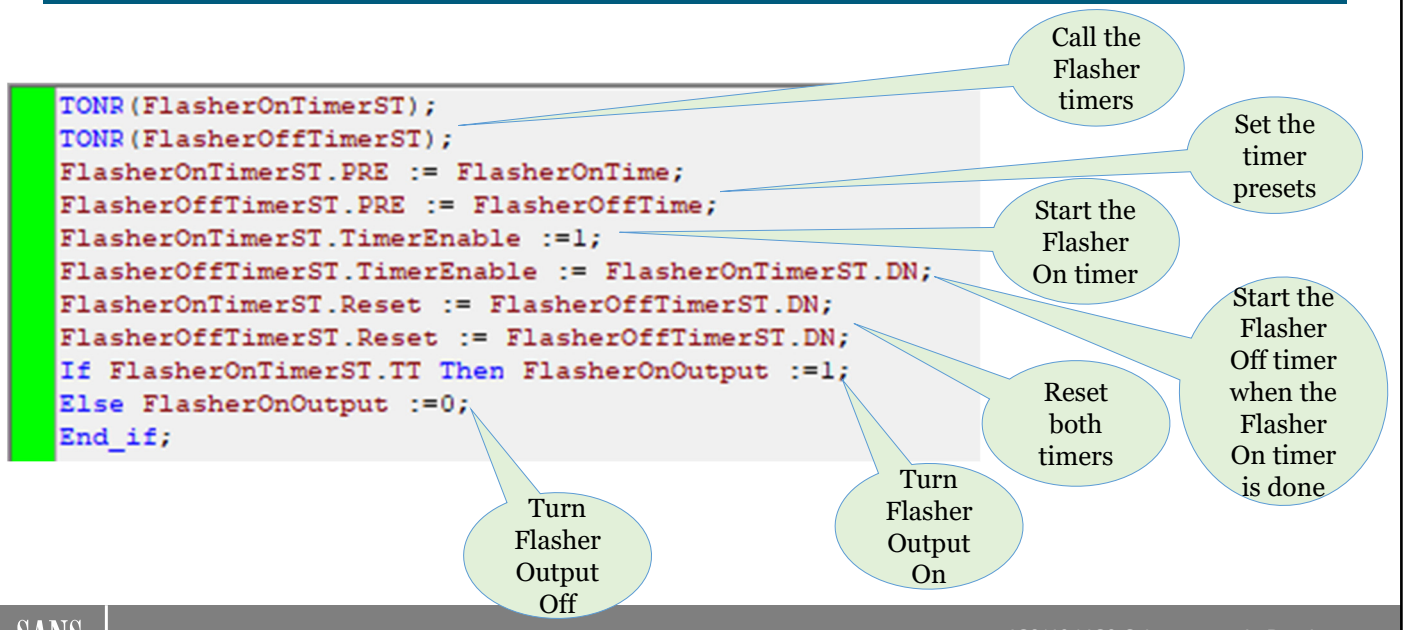
In this example, we are using Ladder Logic with two timers for the Flasher On time and the Flasher Off time. We can see that if memory address C3_FlasherCircuitEnable is True, the Flasher On timer will start to time. Once the Flasher On timer has timed out, it will enable the Flasher Off timer to run. Once the Flasher Off timer has timed out, it will reset the Flasher On timer.

Flasher Circuit in Ladder Logic, Structured Text, and Function Block (2)



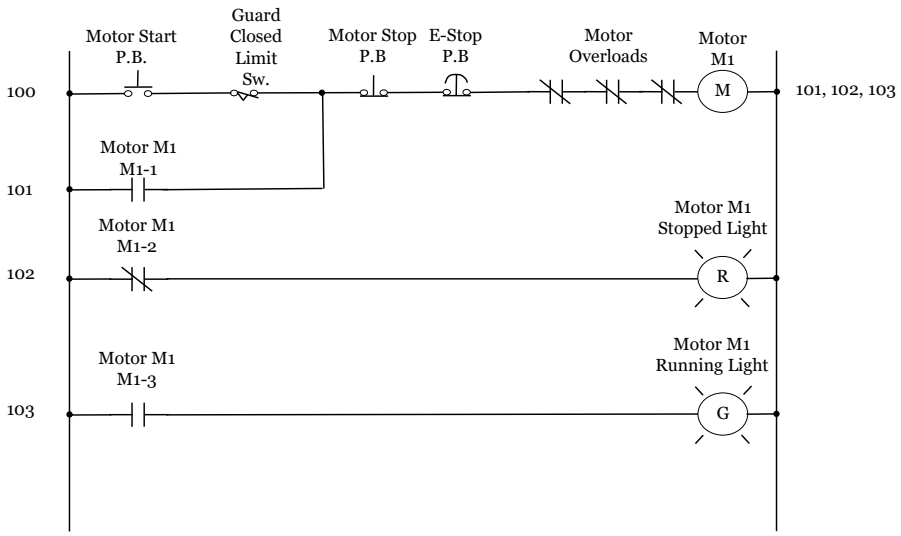
In this example, we are using the Function Block Language with two timers for the Flasher On time and the Flasher Off time. We can see that if memory address C3_FlasherCircuitEnable is True, the Flasher On timer will start to time. Once the Flasher On timer has timed out, it will enable the Flasher Off timer to run. Once the Flasher Off timer has timed out, it will reset the Flasher On timer.

Flasher Circuit in Ladder Logic, Structured Text, and Function Block (3)



In this example, we are using the Structured Text Language with two timers for the Flasher On time and the Flasher Off time. The Flasher On timer will start to time once the routine is entered. Once the Flasher On timer has timed out, it will enable the Flasher Off timer to run. Once the Flasher Off timer has timed out, it will reset the Flasher On timer and itself.

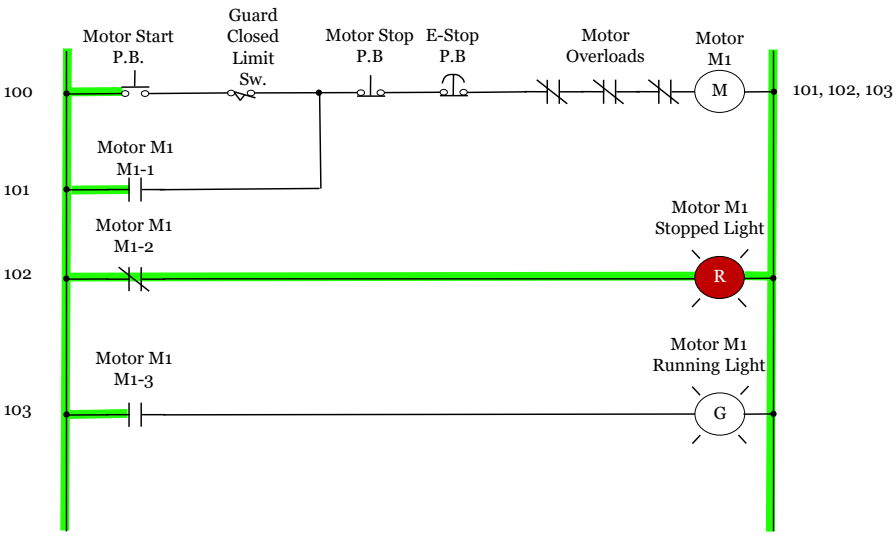
Relay Logic – The Beginnings



The Ladder Logic language was derived from relay logic diagrams where physical relays were wired in such a manner as to perform the desired control functionality.

We will examine a simple motor start circuit with two indicator pilot lights to give us a foundational understanding of relay logic and thereby a basic understanding of Ladder Logic.

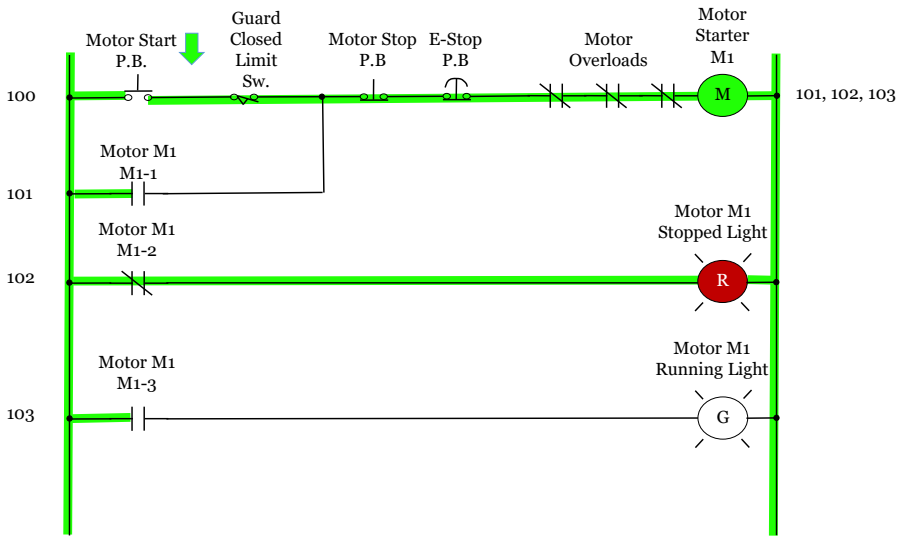
Relay Logic – Power Applied



In this example, we will state that we have electricity applied to the left and right handrails as indicated by the green vertical lines. If we were to look at how the electricity is flowing, we see that electrical potential is applied to the left vertical line which we will call a "rung", as in the rung of a ladder. We see that Motor Starter M1 on the very right-hand side of the first rung labeled 101 is not energized. Because Motor Start M1 is not energized, the Normally Closed M1 contact on rung 102 allows electricity to pass through the contact and thereby energize the Motor M1 Stopped Light (shown in Red).

Also notice to the right of Motor Starter M1 there is a cross-reference to where this motor starter's contacts are used. In this example, lines 101, 102, and 103 contain Motor Starter M1 contacts.

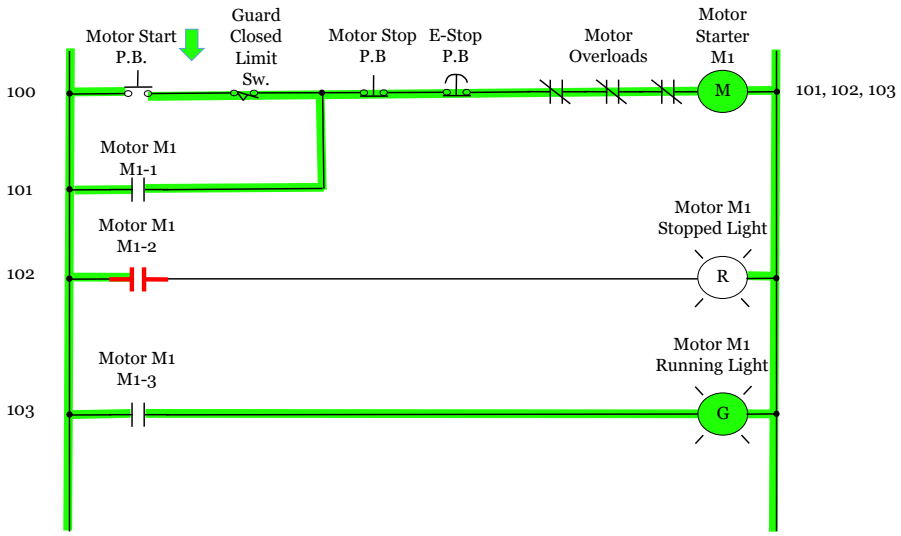
Relay Logic – Push Motor Start Pushbutton



When we push the Motor Start Pushbutton, electricity can now flow to Motor Starter M1 because the Guard Closed Limit Switch is closed, the Motor Stop Pushbutton is not being pushed, the Emergency Stop Pushbutton is not being pushed, and the Motor Overloads are not tripped.

We will see in the next slide that all of the Motor Starter M1 contacts will change state once the Motor M1 coil is energized.

Relay Logic – Push Motor Start Pushbutton and Motor Starter M1 Energizes

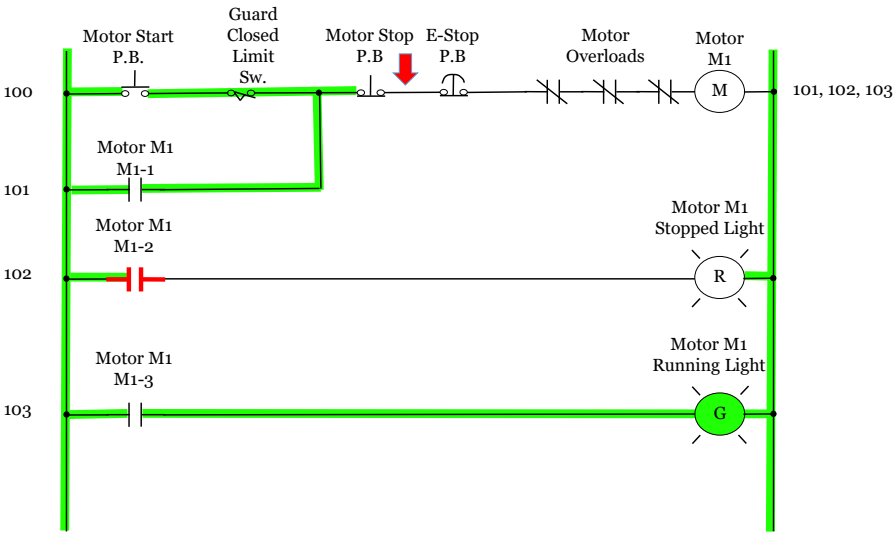


In this moment, we see Motor Starter M1 is energized and the contacts on rung 101, 102, and 103 have changed states. The Normally Open contacts on Rung 101 and 103 have closed and are now passing electricity while the Normally Closed contact on rung 102 has opened and has turned the Motor M1 Stopped Light off. We see the Motor M1 Running Light is now on. A really important concept to grasp is that the Motor M1 M1-1 contact found on rung 101 is closed and is "sealing" the circuit around the Motor Stop Pushbutton and the Guard Closed Limit Switch.

You will note that if you release the Motor Start Pushbutton or open the Guard, it will not stop the motor because of the "sealing" or "latching" circuit.

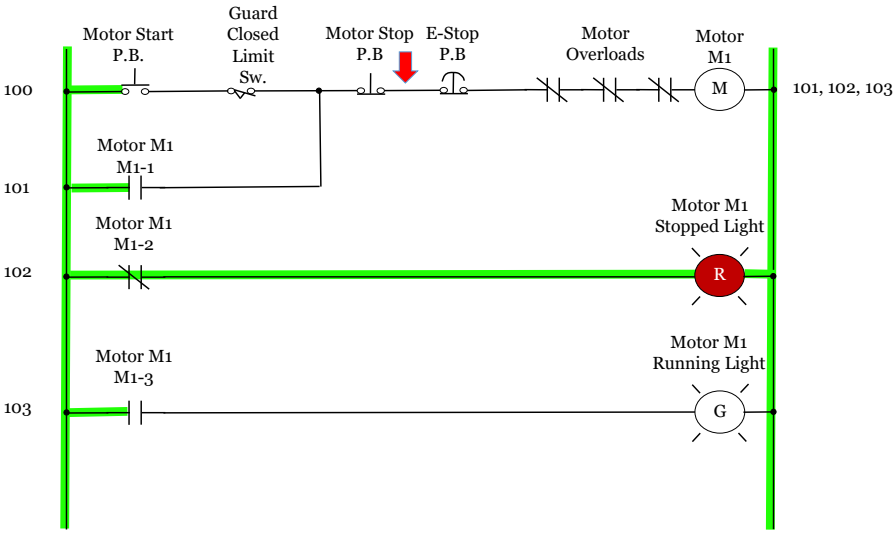
Note: This is not a good situation if you open a Guard and expect the Motor to stop! Normally a "seal in" circuit would seal around a gate but not around a guard!

Relay Logic – Push Motor Stop Pushbutton and Motor Starter M1 De-Energizes (1)



If you push the Motor Stop Pushbutton this will drop the electricity to the Motor Starter M1 and thereby cause the contacts on the Motor Starter M1 to change states.

Relay Logic – Push Motor Stop Pushbutton and Motor Starter M1 De-Energizes (2)



Once the M1 Motor Starter contacts change state, it will drop the Motor M1 Motor Running Light and the Motor M1 Stopped Light will illuminate.

PLC Ladder Logic Emulates Relay Logic

- Ladder Logic does not eliminate the need for sensors, emergency Stop Pushbuttons, motor starters, etc.
  - Any power component that cannot be virtualized, like safety devices, guards and gate switches, motor and drive components, manual control switches, and manual indicator lights that are used for backup
- The Ladder Logic language did add mathematical operations, bit shift, and compare instructions
- Sophisticated PLCs also allow you to create custom data structures and custom instructions made from native data types and instructions

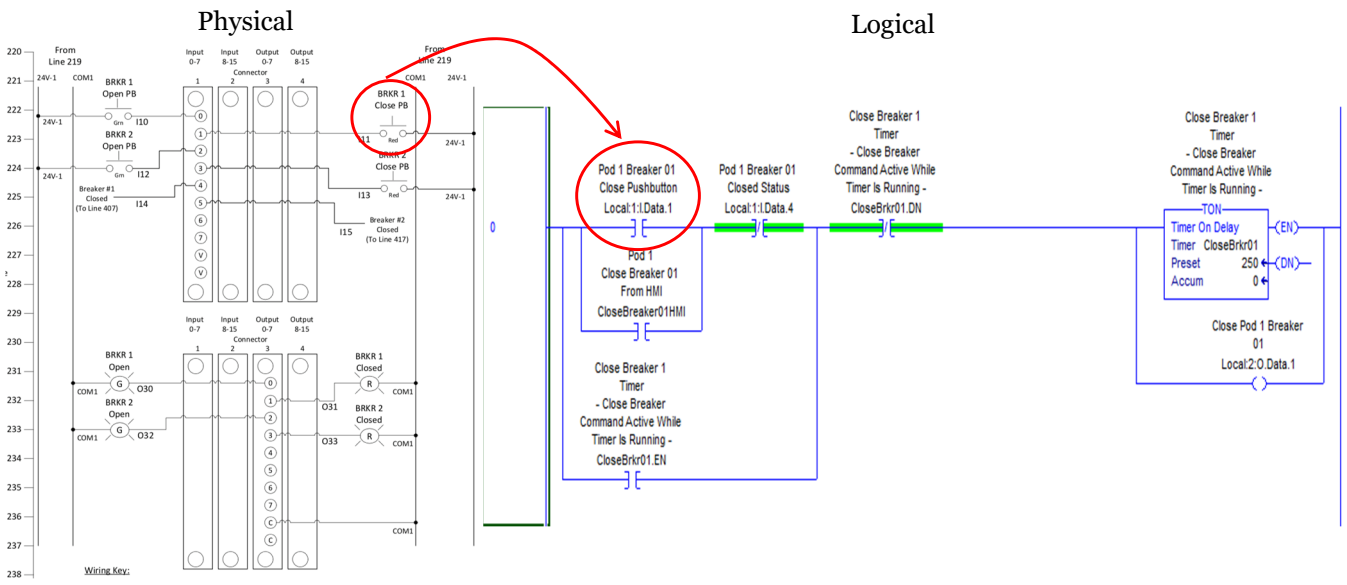
The monetary savings of replacing the relays with a PLC and being able to have flexible programming within the PLC has made the days of relay logic obsolete. Disclaimer: There are still some industries that require relay logic as a backup to PLCs or as the main function control. Using relay logic is the exception and not the rule.

Deploying a PLC has not eliminated the need for input sensors, emergency sensors, gate and guard hardwired sensors, motor starters, etc. Stating the obvious, high-power components are not substituted for PLCs or connected directly to a PLC input or output. Many high-power elements are connected to PLCs through transducers and low power sensors or controllable relays to allow PLCs to monitor and control.

It is common to find the implementation of Ladder Logic to be extended to include mathematical operations, program controls like jumping to subroutines, motion control functions, timers, and counters.

Sophisticated PLCs allow the user to create custom instructions that can be used as classes and instantiated to decrease the required PLC memory and speed up the execution of the code. These next-generation PLCs also allow the programmer to create custom data types so they can encapsulate data types that can be used with custom devices or programs.

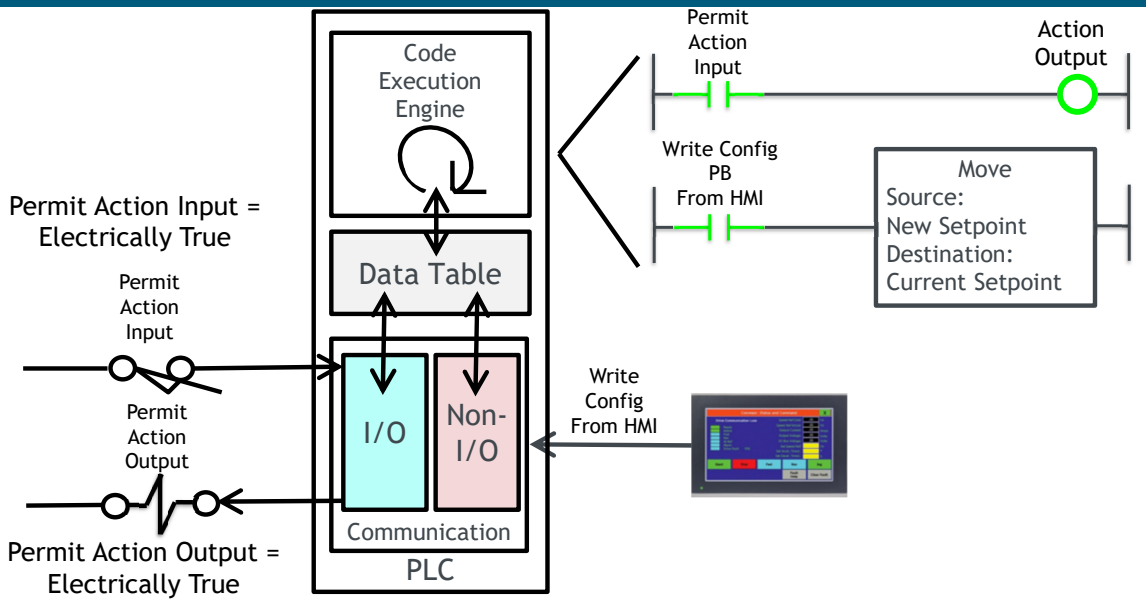
Wire Physical Devices to PLC – Solve Logic inside the PLC



The intent of a common PLC architecture is to wire the physical devices to digital and analog input and output modules and author the PLC code to monitor and control these devices. The PLC technologies have not eliminated the need for sensors and output control devices.

We actually see a great investment in making the sensors and output control devices much smarter and network-capable.

PLC / PAC Simplified Internal Architecture (1)



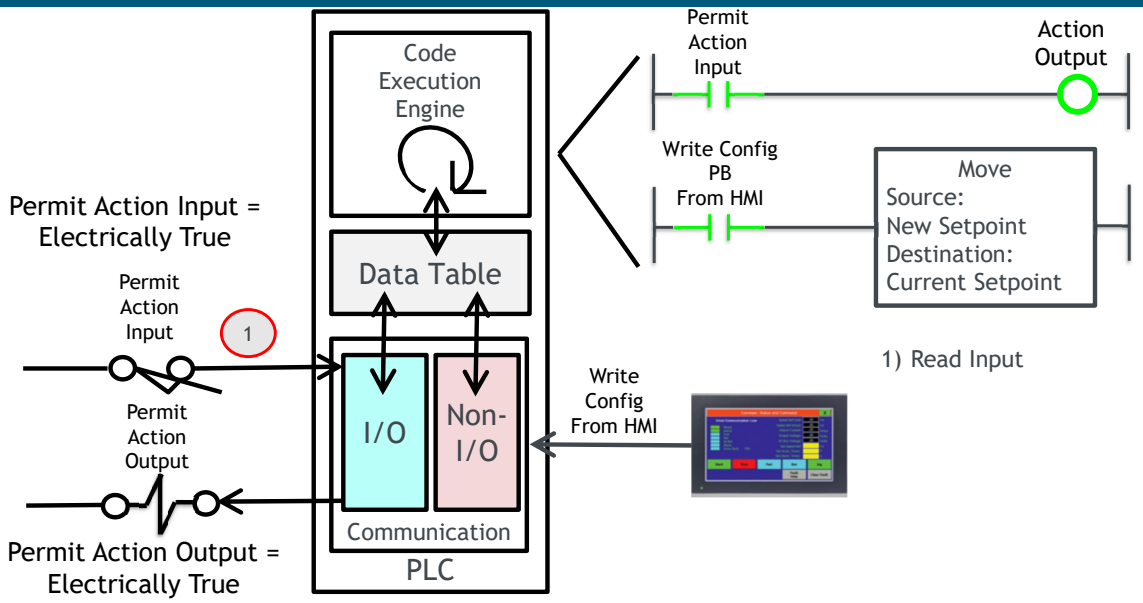
In order to understand a PLC, we must understand the internal architecture of the PLC.

In a high-level abstraction of the PLC shown above, we see that the PLC has a code execution engine responsible for running the program. We also see that the PLC has memory for storing declared variables the program uses. This memory has long been referred to as the "data table". The data table also contains the values and the status of the physical inputs and outputs.

PLCs also have the requirement to service communications from programming terminals, HMIs, other PLCs, and sometimes from queries from higher-level systems like Historians.

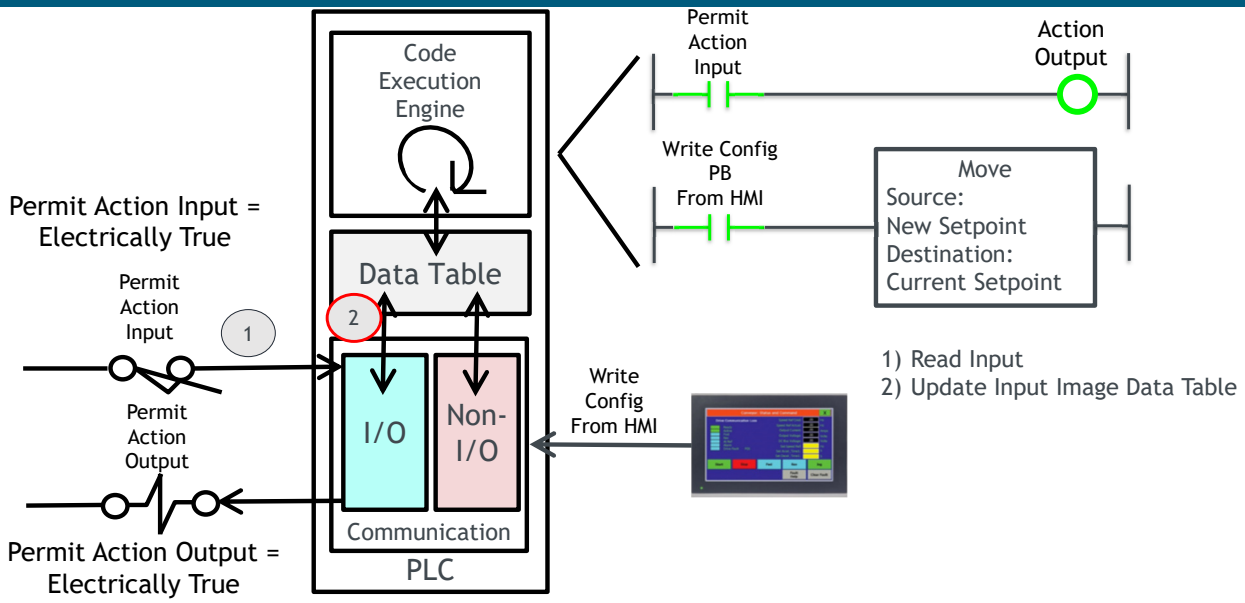
We will walk through the sequence of a typical PLC input-to-output cycle.

PLC / PAC Simplified Internal Architecture (2)

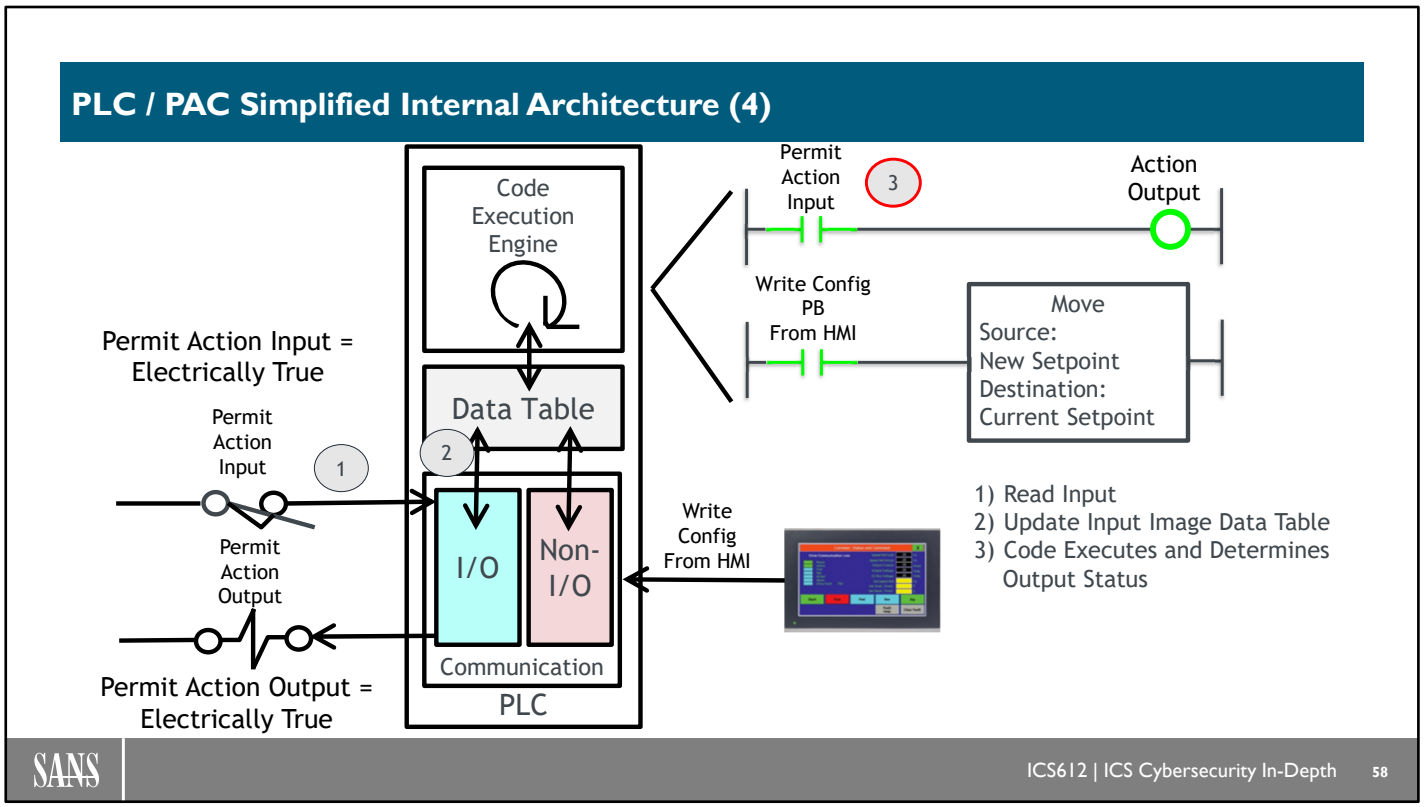


The first step in our example is to read the status of the inputs. In this particular example, we see that the permit action input limit switch is closed, and the input is read as “true” or Boolean 1.

PLC / PAC Simplified Internal Architecture (3)

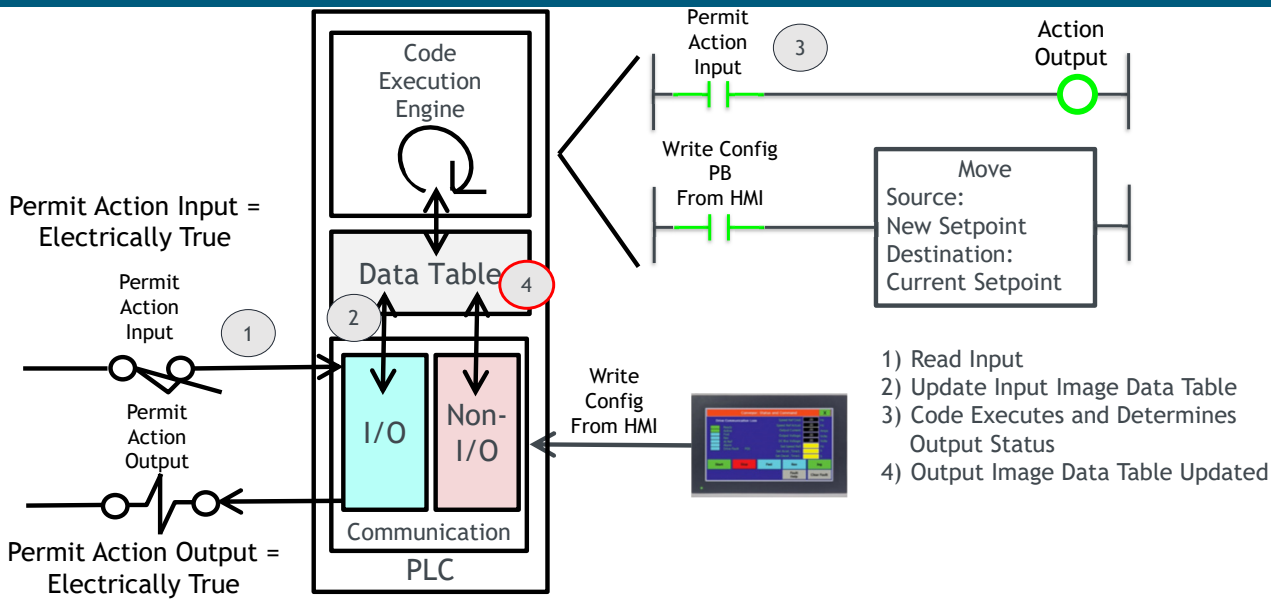


In our next step, the input statuses are copied to our data table, where the program can read these variables and solve the logic.



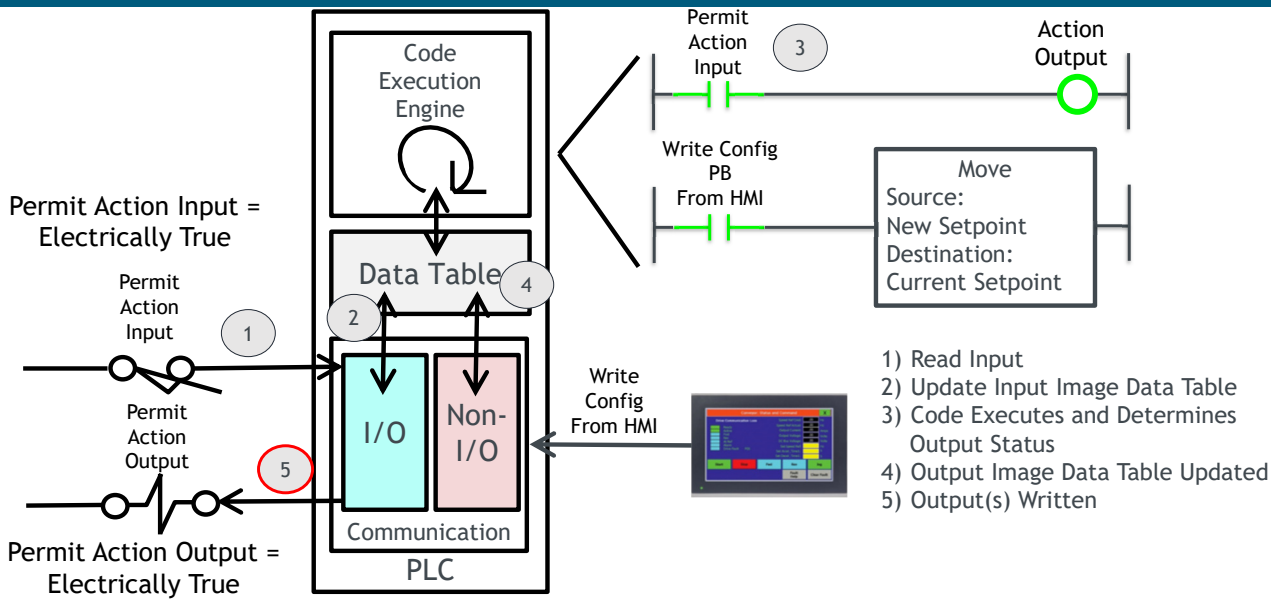
In step three, we will use the input data within the program to solve the logic. In this particular case, the "Action Output" is on because our "Permit Action Input" is true.

PLC / PAC Simplified Internal Architecture (5)

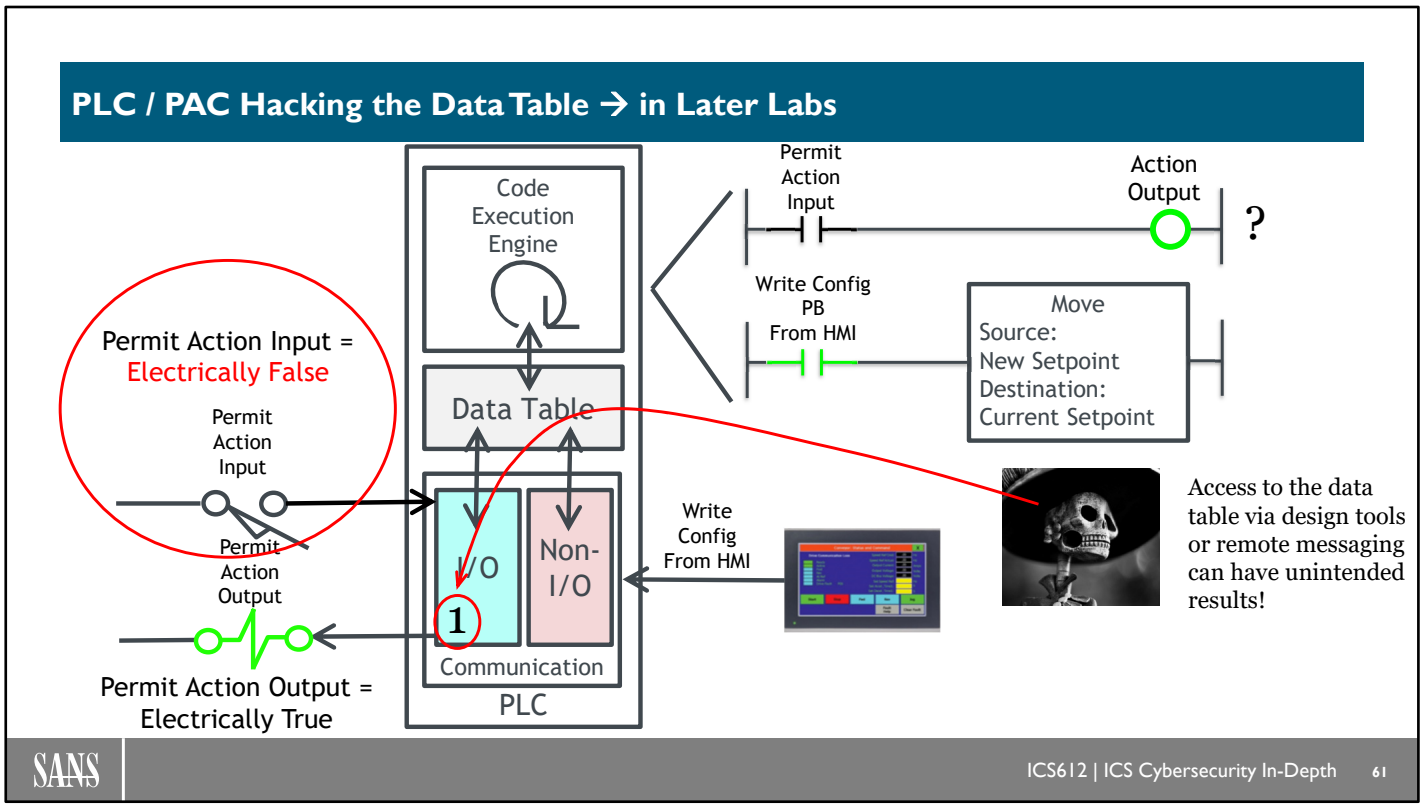


In step four we will update the outputs in the data table based on how the logic solved after the program scan. In this particular case, we will write the "Action Output" status to true or Boolean 1.

PLC / PAC Simplified Internal Architecture (6)



The output status in the data table is updated in the output image table, the Permit Action Output is written to "true", and the output is turned on.

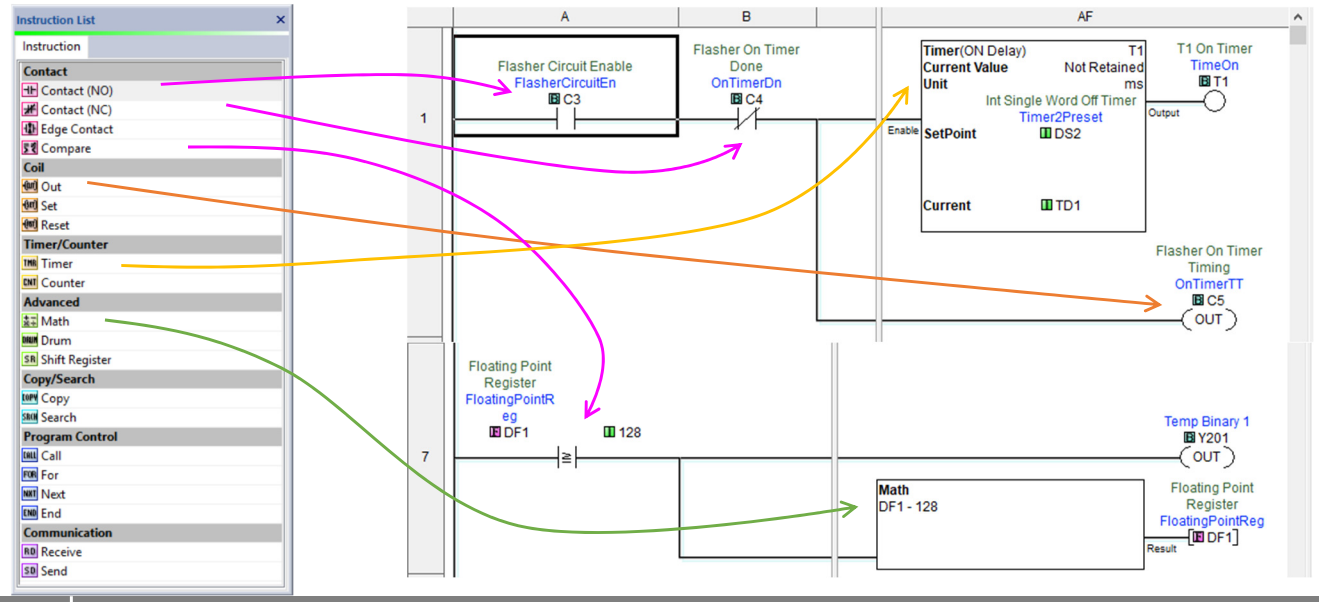


Understanding the idea of the data table and how it affects the way the program solves the output status is very important. Controlling access to this memory is paramount because it becomes possible to turn outputs off or on without respect to the input status. This can be achieved by directly modifying the value by changing the variables within the data table.

The data table is also the container for sensor scaling and calibration information as well as mathematical control calculations, which can be affected by changing these stored values. For instance, a possible attack vector could be to change the sensor scaling configuration, which will cause the PLC to work with incorrect sensor values. The PLC misbehavior is not caused by changing the code, which is easier to detect, but rather by simply making changes in the data table.

Understanding how to gain access to the PLC is like knowing where the holes are in the castle walls. It's okay to have means in and out of the PLC, but it's important to understand where the entry points are and the possible effects of each.

Click Plus PLC Instructions

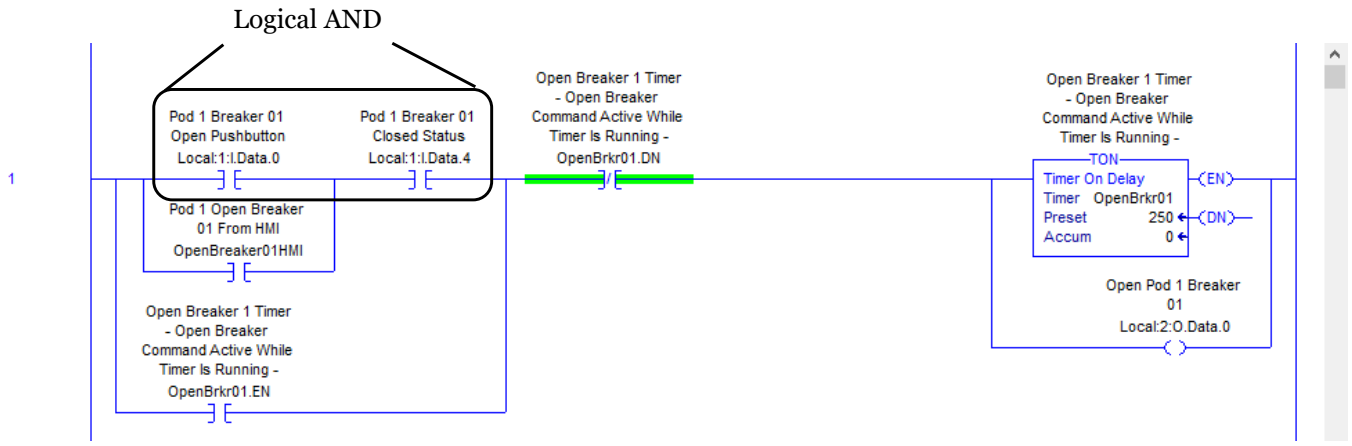


The Click Plus PLC Ladder Logic instruction set is representative of most PLCs on the market today. On the left-hand side of the slide, we see the instructions used to make a Ladder Logic Program. In this particular example we see Rung 1 with a Timer instruction while on Rung 7 we see the use of the mathematical comparison of floating-point register DF1 being less than or equal to 128 in order to turn output Y201 to an on state.

We will explain the Boolean logic found in Ladder Logic programming on the next set of slides. This slide is simply meant to inform the reader of some of the Ladder Logic instructions supported by the Click Plus PLC.

Understanding Ladder Logic – Boolean Operations: AND

Open Pushbutton = True AND Breaker 01 Closed = True



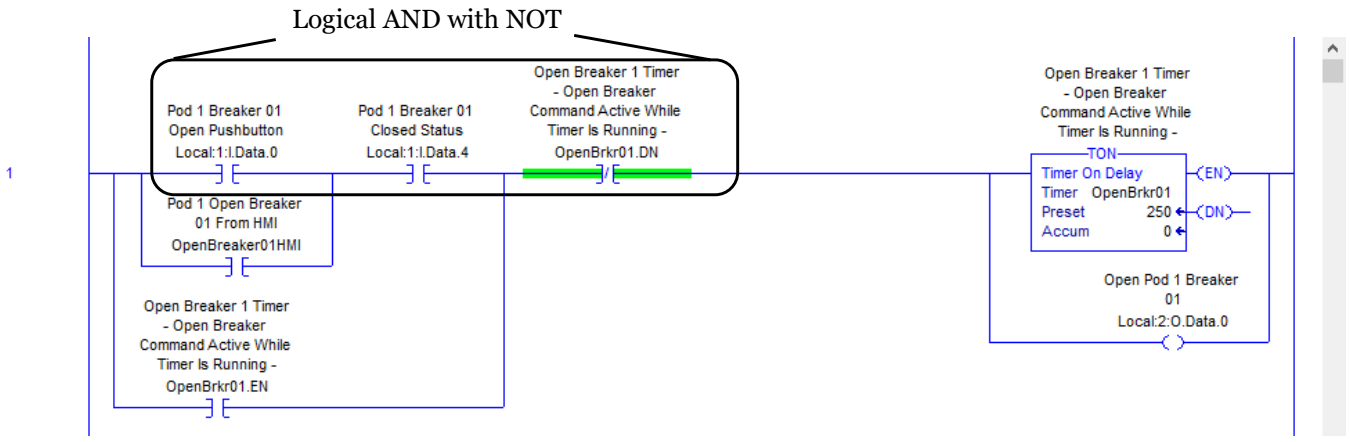
Boolean logic is the basis for most Ladder Logic programming, with the exception of program controls and mathematical computations.

In the above example, we are showing a Timer On (TON) circuit that can be dissected into a Boolean expression to see if we should start the timer timing or turn the timer off.

The first expression we will look at is a Boolean AND. If the Pod 1 Breaker 01 Open Pushbutton is True (in this case being pushed) AND the Pod 1 Breaker 01 Closed Status is True (in this case the breaker is closed) then this part of the rung will solve True.

Understanding Ladder Logic – Boolean Operations: AND NOT

Open Breaker Physical Pushbutton = True AND Breaker 01 Closed = True AND NOT Open Breaker Timer Done = False: Then Timer OpenBrkr01 would start timing



Moving on, if the Pod 1 Breaker 01 Open Pushbutton is True (in this case being pushed) AND the Pod 1 Breaker 01 Closed Status is True (in this case the breaker is closed) and the Open Breaker 1 Timer Done Bit is False (in this case the OpenBrkr01 is not timed out) then this part of the rung will solve True.

So, let's talk this through. We want to push the Pod 1 Breaker 01 Open Pushbutton and we want to check that the Pod 1 Breaker 01 is not already open. Why? For two reasons: First reason – we don't want to give it an open command if it's already open; and second because if you look at the electrical specs of the breaker you can burn out the coils if you electrically keep the coil energized for more than 250 milliseconds.

So now we also know the purpose of the timer. We want to limit the coil on time to 250 milliseconds.

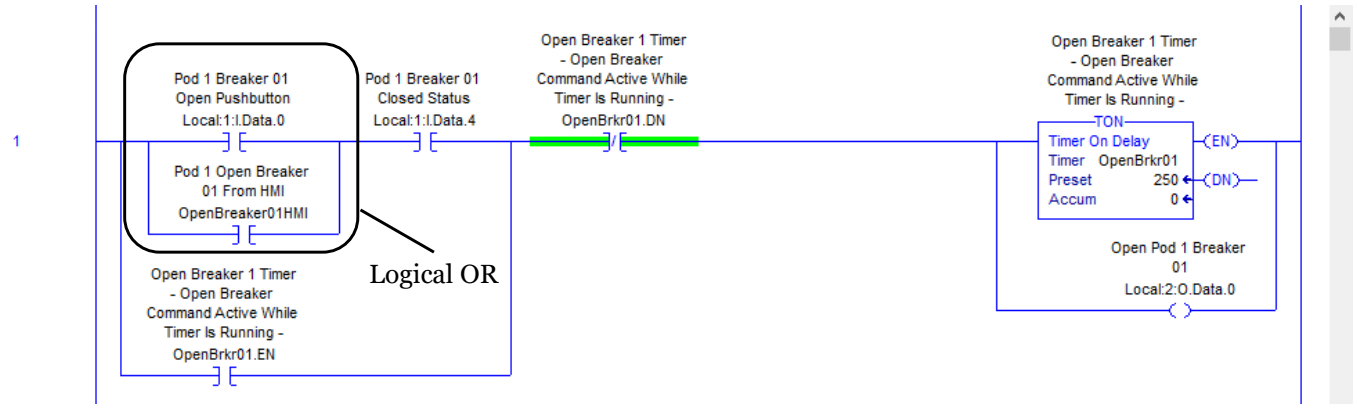
Note: If you know the electrical characteristics of devices and how to destroy them then you should protect the devices with your programming.

Question: Have you ever heard of a famous scenario where the electrical characteristics of a centrifuge were known, and it was used to destroy the equipment?

Let's continue to the next slide.

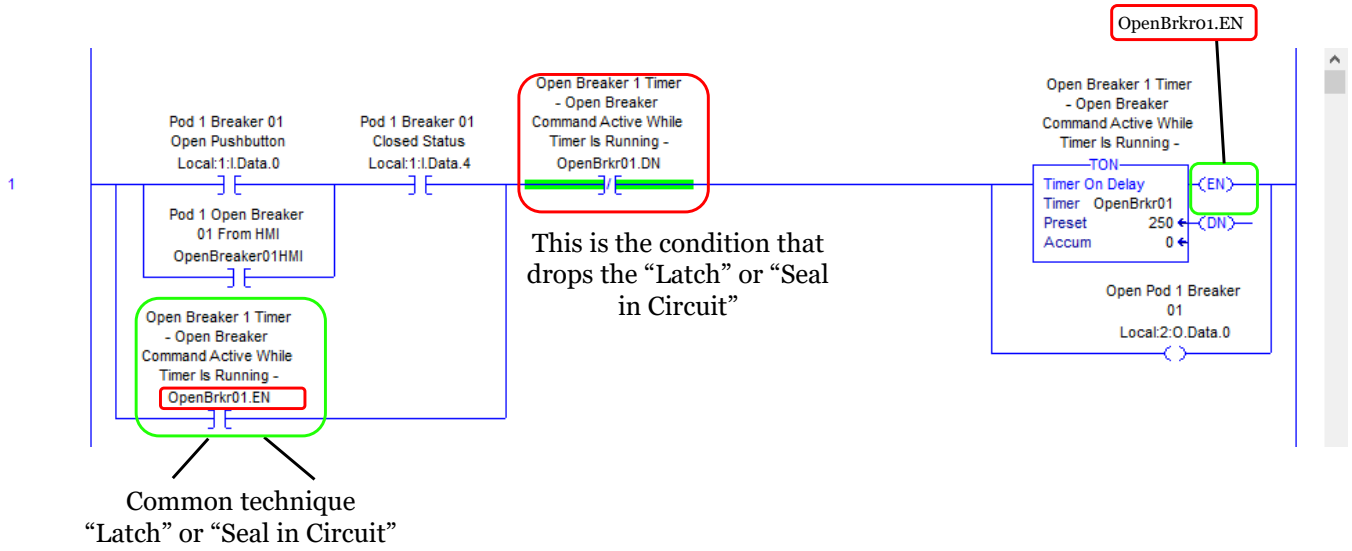
Understanding Ladder Logic – Boolean Operations: OR

Open Breaker Physical Pushbutton = True OR Open Breaker From HMI Panel = True AND Breaker 01 Closed = True AND NOT Open Breaker Timer Done = False:
Then Timer OpenBrkr01 would start timing



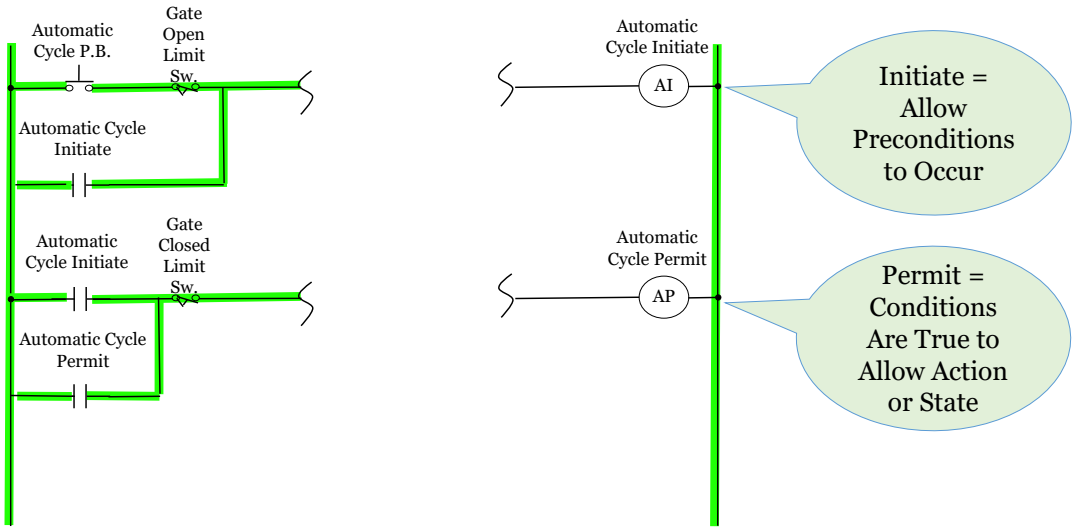
The Logical "OR" is used in this case if an operator wants to open Pod 1 Breaker 01 from an HMI pushbutton.

Understanding Ladder Logic – Common Techniques



Let's look at a common technique called "Latching" or "Seal in Circuit". If we look at this circuit, once the timer is Enabled or Running by using the "OpenBrkr01.EN" (read this as the timer is Enabled bit) then we can keep the timer running until it times even when the Open Pushbutton or HMI Pushbutton is not being pushed. The Timer being Done (OpenBrkr01.DN) is the condition that drops the "Latch" or "Seal in Circuit".

You May Run across "Initiate" and "Permit" Terminology

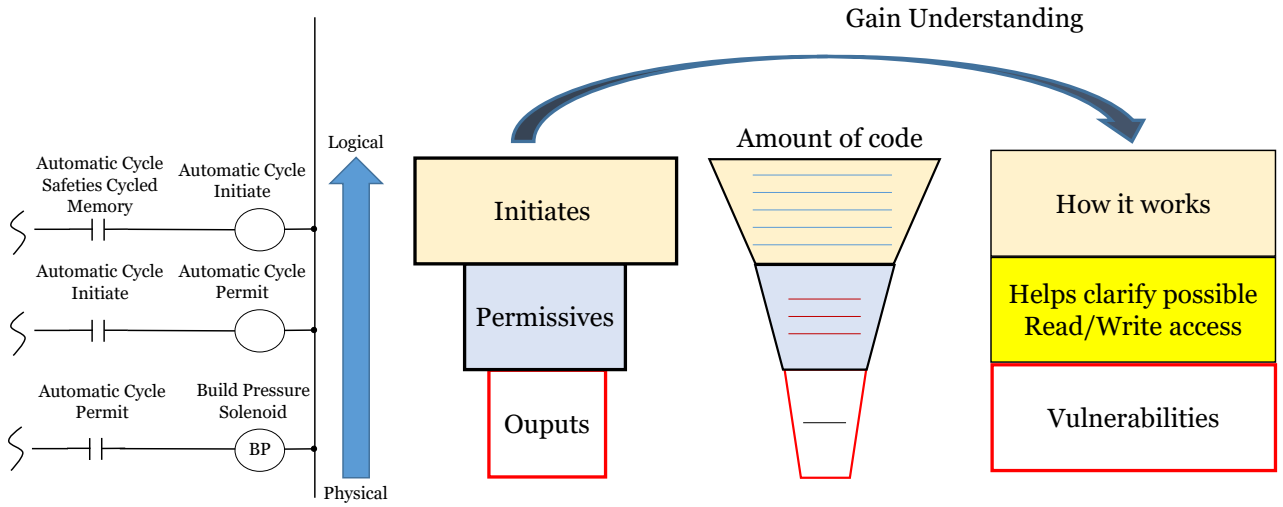


It is not uncommon to run across logic that is labeled "Initiate" and "Permissives" or "Permit".

The concept is to have a condition specified that will "Initiate" the machine to perform a function while the "Permissive" or "Permit" bit will cause the machine to actually perform the function.

An example could be: An operator pushes the Automatic Cycle Pushbutton. The Initiate condition is to close a safety gate. The Permit machine movement bit will become true once the gate is closed.

How to Dig into Ladder Logic When You Aren't Sure How? Use the Upside-Down Wedding Cake Approach



The outputs of a PLC determine what real-world actions can occur so in order to understand a ladder logic program, start with the outputs and work the logic backwards. We call this the "upside wedding cake" approach because we start with singular points like output solenoids, lights, valves, etc. and work our way backwards through the logic to understand the conditions that are required to energize or de-energize the output(s).

Working backwards through the logic starting from the outputs can also help us determine what sources other than ladder logic, function block or other logic that can read or write to our system.

Compiler (A.K.A – Design Tools)

- Vendors provide design software that compiles and downloads PLC programs to the hardware
  - If you think about it, LD, FBD, ST, and SFC languages can compile down to simple Boolean logic
  - The math functions are typically handled in some sort of math co-processor
- Most vendors will design Application-Specific Integrated Circuits (ASIC) that optimize the Boolean logic solving and a method to pause the logic solving while they are handling mathematical functions

Most Ladder Logic can be simplified to Boolean logic. Many vendors have designed Application-Specific Integrated Circuits (ASIC) that optimize the Boolean logic solving and do mathematical functions that are not solved with simple Boolean logic.

Click Plus PLC Data and Memory Types (1)

Data Type	S/W Icon	Data Ranges
Bit	B	0, 1
Integer (Single Word)	I	-32,768 to 32,767
Integer2 (Double Word)	I2	-2,147,483,648 to 2,147,483,647
Floating Point	F	-3.4028235E+38 to 3.4028235E+38
HEX (Hexadecimal)	H	0000h to FFFFh (The HEX data type requires the 'h' after the value.)
Text (Single Character)	T	Single ASCII character (ASCII code: 00h to FFh.)
ASCII Code	A	ASCII code $00 to $FF (The ASCII Code data type requires the '$' before the value.)

Memory Type	Symbol	Data Type	S/W Icon	Definition
Input Point	X	Bit	B	The Discrete Input points are represented by the "X" symbol.
Output Point	Y			The Discrete Output points are represented by the "Y" symbol.
Control Relay	C			The Control Relay bits are represented by the "C" symbol. These internal bits are typically used for ladder program control. They do not represent any real world inputs or outputs.
Timer	T			The Timers are represented by the "T" symbol. The Timer status bit is used to indicate when the Current Value of the timer equals its Preset Value.
Counter	CT			The Counters are represented by the "CT" symbol. The Counter status bit is used to indicate when the Current Value of the counter equals its Preset Value.
System Control Relay	SC			The internal System Control Relays, represented by the "SC" symbol, are pre-defined bits which represent the status of specific system functions.
Data Register	DS	Integer	I	Single word integer data registers are represented by the "DS" symbol.
	DD	Integer2	I2	Double word integer data registers are represented by the "DD" symbol.
	DH	HEX	H	Single word Hex data registers are represented by the "DH" symbol.
	DF	Floating Point	F	Data Floating Point registers are IEEE format Real number values represented by the "DF" symbol as 32 bit words.
Input Register	XD	HEX	H	The Input Registers, represented by the "XD" symbol, contain groups of Discrete Input points in a 16 bit word format.
Output Register	YD			The Output Registers, represented by the "YD" symbol, contain groups of Discrete Output points in a 16 bit word format.
Timer Register	TD	Integer	I	The Timer Registers, represented by the "TD" symbol, contain the corresponding Timer's accumulative value in a 16 bit data register.
Counter Register	CTD	Integer2	I2	The Counter Registers, represented by the "CTD" symbol, contain the corresponding Counter's accumulative value in a 32 bit data register.
System Data Register	SD	Integer	I	The internal System Data Registers, represented by the "SD" symbol, are pre-defined words which represent the status of specific system functions.
Text	TXT	Text	T	The Text data registers, represented by the "TXT" symbol, are used to store and manipulate ASCII text data.

Anytime you study language programming, you typically dive into supported data types and the ranges of each. This is also true when diving into PLC programming. You need to familiarize yourself with the Click Plus PLC data and memory types.

Click Plus PLC Data and Memory Types (2)

Data Type	S/W Icon	Data Ranges
Bit	B	0, 1
Integer (Single Word)	I	-32,768 to 32,767
Integer2 (Double Word)	I2	-2,147,483,648 to 2,147,483,647
Floating Point	F	-3.4028235E+38 to 3.4028235E+38
HEX (Hexadecimal)	H	0000h to FFFFh (The HEX data type requires the 'h' after the value.)
Text (Single Character)	T	Single ASCII character (ASCII code: 00h to FFh.)
ASCII Code	A	ASCII code $00 to $FF (The ASCII Code data type requires the '$' before the value.)

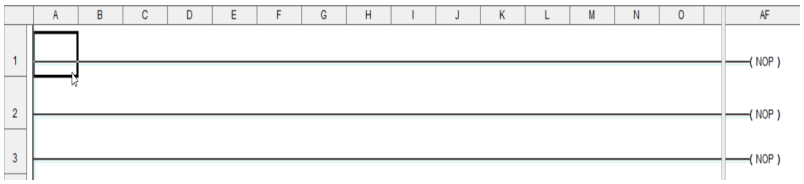
Memory Type	Symbol	Data Type	S/W Icon	Definition
Input Point	X	Bit	B	The Discrete Input points are represented by the "X" symbol.
Output Point	Y			The Discrete Output points are represented by the "Y" symbol.
Control Relay	C			The Control Relay bits are represented by the "C" symbol. These internal bits are typically used for ladder program control. They do not represent any real world inputs or outputs.
Timer	T			The Timers are represented by the "T" symbol. The Timer status bit is used to indicate when the Current Value of the timer equals its Preset Value.
Counter	CT			The Counters are represented by the "CT" symbol. The Counter status bit is used to indicate when the Current Value of the counter equals its Preset Value.
System Control Relay	SC			The internal System Control Relays, represented by the "SC" symbol, are pre-defined bits which represent the status of specific system functions.
Data Register	DS	Integer	I	Single word integer data registers are represented by the "DS" symbol.
	DD	Integer2	I2	Double word integer data registers are represented by the "DD" symbol.
	DH	HEX	H	Single word Hex data registers are represented by the "DH" symbol.
	DF	Floating Point	F	Data Floating Point registers are IEEE format Real number values represented by the "DF" symbol as 32 bit words.
Input Register	XD	HEX	H	The Input Registers, represented by the "XD" symbol, contain groups of Discrete Input points in a 16 bit word format.
Output Register	YD			The Output Registers, represented by the "YD" symbol, contain groups of Discrete Output points in a 16 bit word format.
Timer Register	TD	Integer	I	The Timer Registers, represented by the "TD" symbol, contain the corresponding Timer's accumulative value in a 16 bit data register.
Counter Register	CTD	Integer2	I2	The Counter Registers, represented by the "CTD" symbol, contain the corresponding Counter's accumulative value in a 32 bit data register.
System Data Register	SD	Integer	I	The internal System Data Registers, represented by the "SD" symbol, are pre-defined words which represent the status of specific system functions.
Text	TXT	Text	T	The Text data registers, represented by the "TXT" symbol, are used to store and manipulate ASCII text data.

You will use DS registers to communicate from your Click Plus PLC to the Allen-Bradley PLC

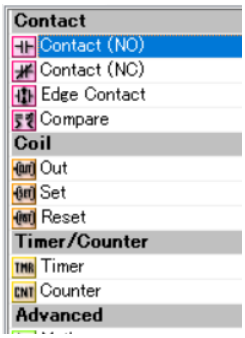
You will become familiar with the single-word Integer known as DS which stands for Data Register – Single Word. We will use this data type for Modbus communications between the Click Plus and the Allen-Bradley PLC.

Click Plus PLC Software Logistics – Drag and Drop Contacts, Coils, Timers, etc. (1)

Many PLC programming applications will open with a screen like this



With the Click Plus software you will select the rung you wish to work with and the location on the rung, and then you will drag and drop the function you wish to add from the instruction list

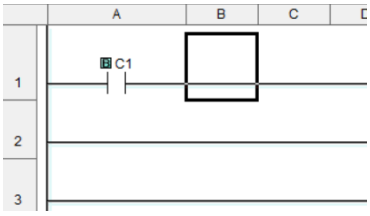


While each vendor has their own proprietary software suite that is used to interact with and configure their devices, there has been a common look and feel to many applications in this industry. You will be exposed to a number of proprietary applications throughout this course. The first application you will use for PLC programming is the Click Plus software, which will allow you to create a new project. The Lab Workbook will walk you through the step-by-step components in build-specific operational projects. Over the next set of slides, we will simply show you how to interact with the Click Plus software to program a PLC.

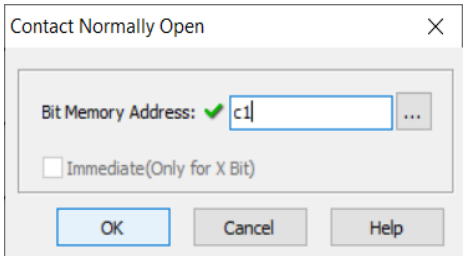
As you launch a new project you will see a series of blank lines or "rungs" with a No Operation placeholder on the far right of each rung. As you select a spot on a rung, you can drag various functions (Contacts, Coils, Timer/Counter, Math) from the instruction list.

Click Plus PLC Software Logistics – Drag and Drop Contacts, Coils, Timers, etc. (2)

When you drag and drop an object from the instruction list to a location, you will be presented with an object-specific properties window.



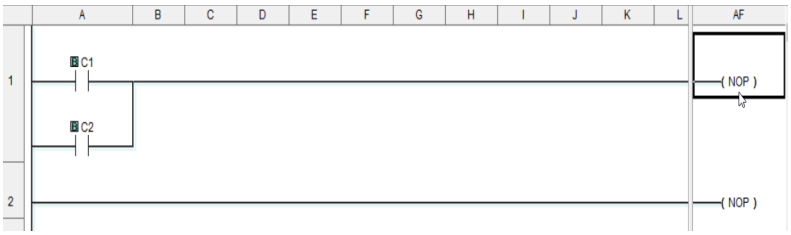
The properties window may be very simple and only require a name, as is the case with the Normally Open contact instruction object.



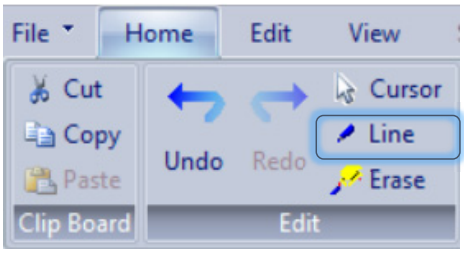
As you consider your program, it is important to understand that the PLC will process the logic in the same way that you typically read a book in English: Top down, left to right. When the end is reached, it starts reading again and will continue looking for any interrupt or state change to take action on. Dragging and dropping a Normally Open contact into the first area on the first rung will prompt you to name the object placed.

Click Plus PLC Software Logistics – Creating a Branch (1)

A common need in logic programming is the need for a parallel branch in which more than one contact can drive an effect on a rung



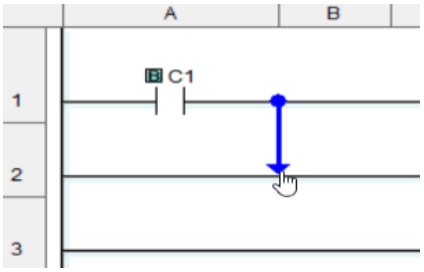
To achieve this within the Click Plus software you will need to use the line tool



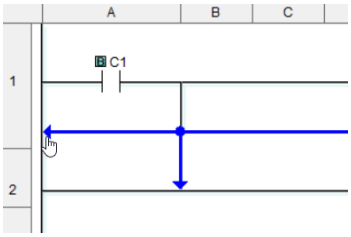
With one object on the first rung, you may want another Normally Open contact in parallel to the first, which would require a connector branch in the logic. Again, while all vendors do this somewhat differently, within the Click Plus software you will use the line tool.

Click Plus PLC Software Logistics – Creating a Branch (2)

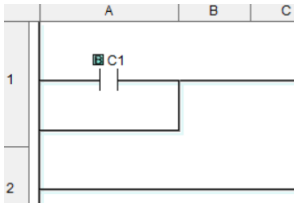
To use the Line tool, you will highlight the rung and location you wish to branch from, then select the Line tool. When the blue line appears – Click Plus the location you want to go next.



The Line tool will automatically show options for where you can branch next. Again, Click Plus where you want to go next



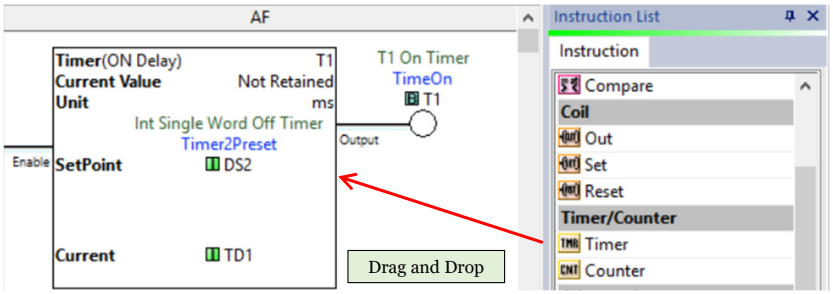
When finished hit Esc



In general, you highlight the area on the rung where you want to add an object, then select the line tool. Where you want the first branch segment to end, you Click Plus the end of the arrow with the Hand pointer. After the first segment is placed, the line tool suggests available branch paths. Again, you Click Plus the end of the arrow with the Hand pointer depending on how you want the branch logic to flow.

Click Plus PLC Software Logistics – Drag and Drop Timers

Timer or counter functions are used in many processes, and with the Click Plus software you simply highlight the rung and location that you wish to place the function and then you drag and drop the timer or counter instruction

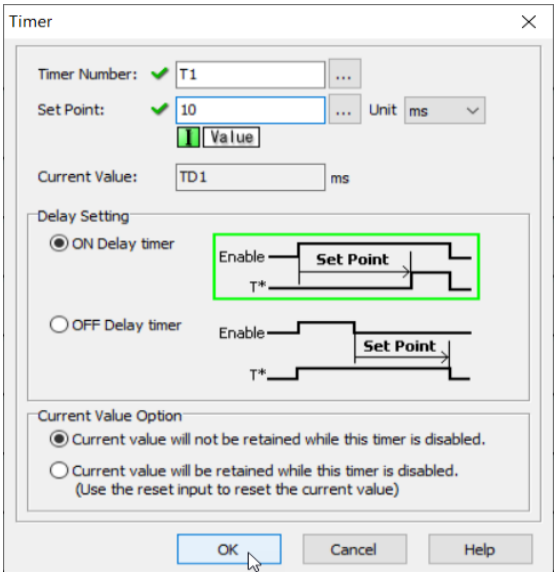


Timers and counters will be used in many ways in this course and in the real world. As mentioned before, the way they are depicted and used depends on the vendor and the application, but with the Click Plus software, you simply highlight the area on the rung where you wish to place the timer/counter function and then you drag and drop the item from the function list.

Click Plus PLC Software Logistics –Timer Properties

After you place a timer function on a rung, you are prompted with an object-specific properties window. For a timer instruction you need to:

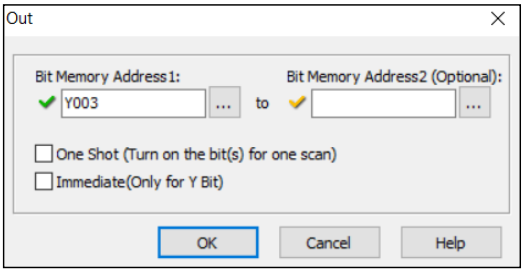
- Name the timer
- Set the timer value
- Select On or Off delay
- Retain current value, or not, when disabled



As with all functions, you will need to configure some settings in order for it to work. When the timer function is placed, the Click Plus software will pop up a window where you name the instruction and establish a time unit and scale depending on the operation. You will also need to pick the On or Off delay, as well as whether you want the timer present value to be preserved or not when the timer is disabled. These are examples of operations-specific engineering decisions that need to be configured in a manner that supports the overall safe and reliable operation.

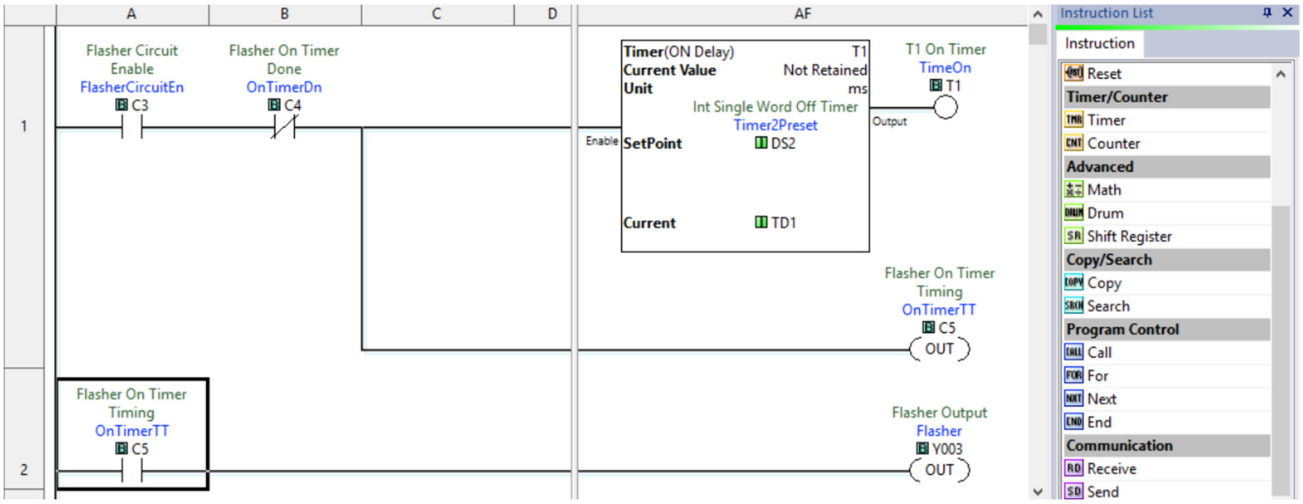
Click Plus PLC Software Logistics – Outputs

You will at some point wish to drive some process output points in your logic. With the Click Plus software, you will drag and drop the Out instruction and set the address that you wish to activate as an output



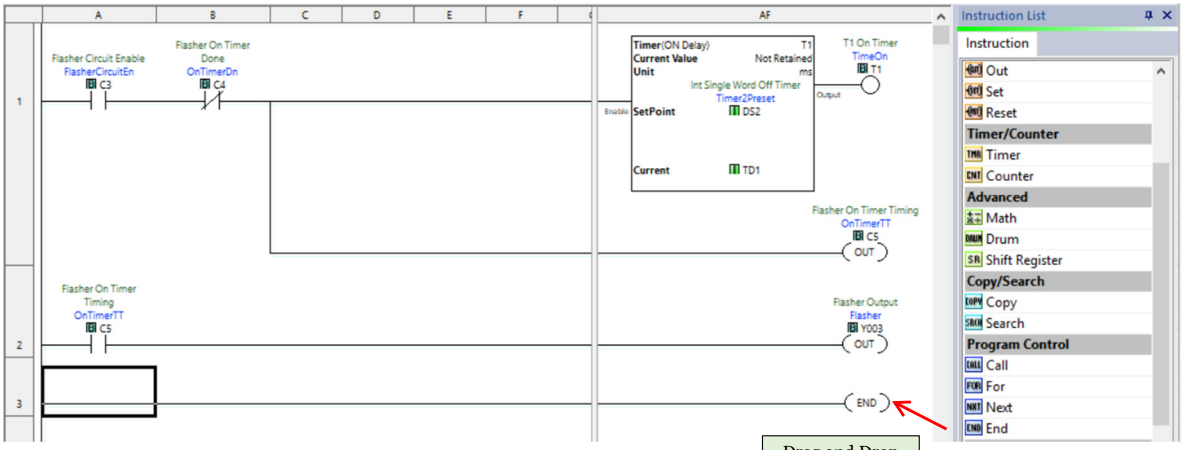
Selecting an output coil function will drive an action. This is where you begin to see the cyber-to-physical nature of industrial control systems and PLCs specifically. When you highlight an area on a rung in your logic and drag and drop an Out function, you will need to configure the address and decide how you want the output value to behave.

Click Plus PLC Software Logistics – Example



This screen shows the various example components that we just walked through, and in this example if Normally Open contact C3 is activated, then the timer will execute, and the output will be seen on Y003.

Click Plus PLC Software Logistics – Ending a Program

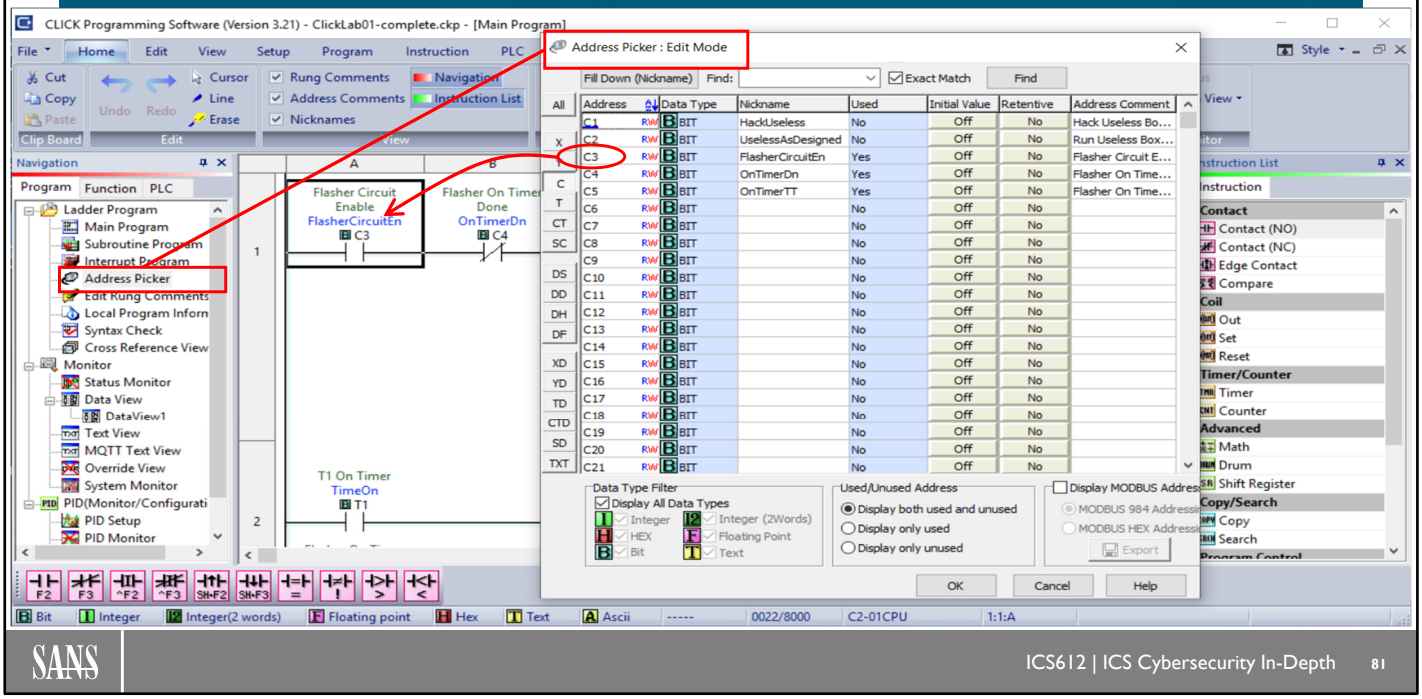


Drag and Drop

To successfully compile, add an "End" Instruction

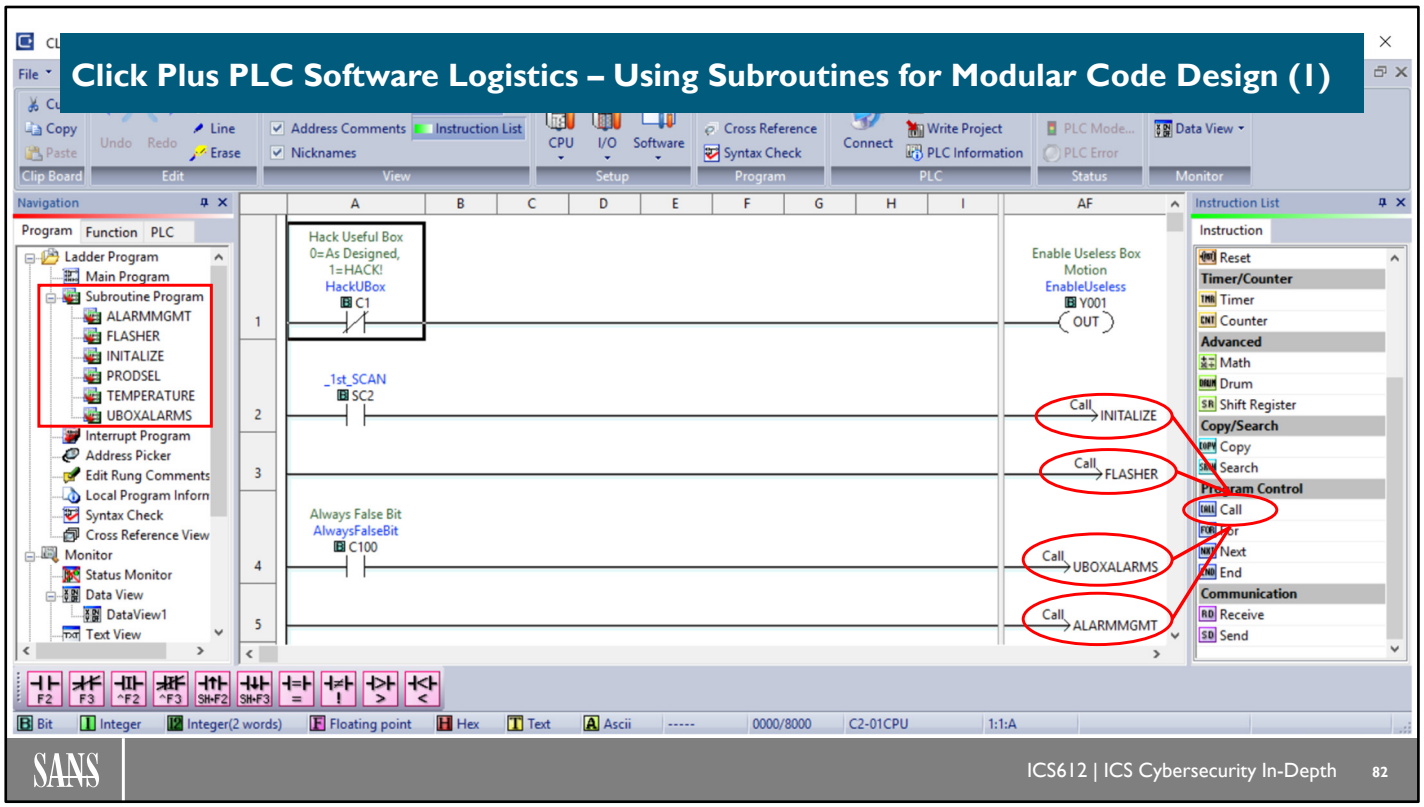
One last function needs to exist in the Click Plus software Ladder Logic program. We need to add an End function on the right side of the last rung in our program. The Click Plus Program needs to have an END Instruction as the last rung of Ladder Logic code in order to compile correctly.

Click Plus PLC Software Logistics – Entering Address and Comments



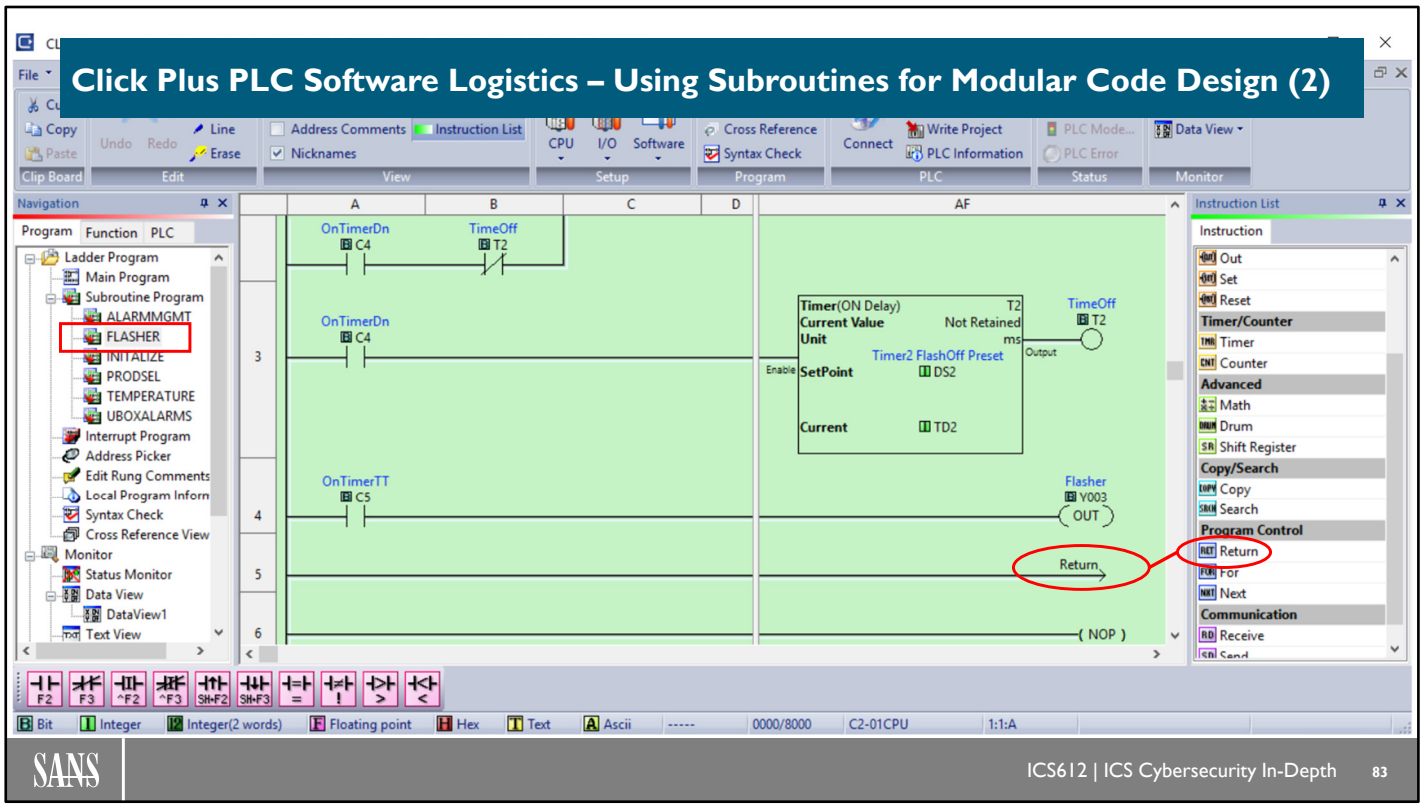
The Click Plus PLC software is used to define intuitive names and comments to the Inputs, Outputs, Timers, Counters and memory registers. By selecting “Address Picker”, the memory types are displayed. As we see in this example, the “C” or Control Relay entries are displayed along with their respective Address Comment in the right-hand column

The other item to note in this screenshot is the mode in which we are editing the program. At this particular time, we are in the “Offline” mode. Offline is defined as working on the project on our local computer whereas “Online” mode is defined as working on the program on the Click Plus PLC. Please note, the Offline project and the Online program can be different; it is important to be aware of which program you are editing so you know which is the most current.



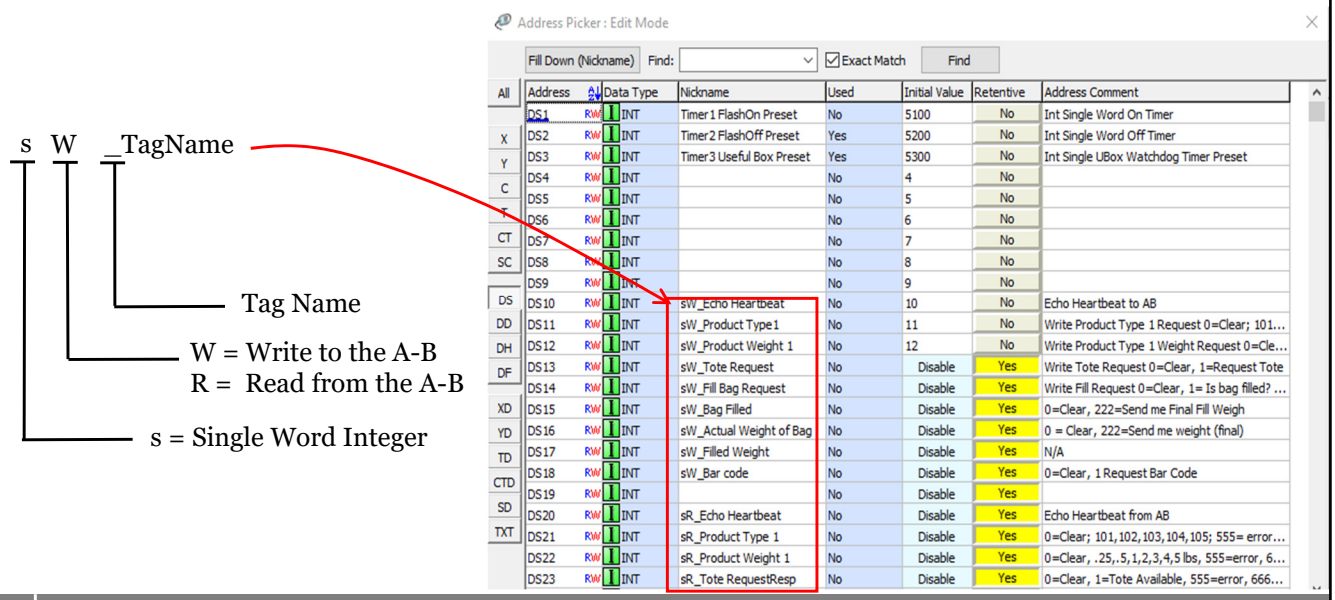
The Click Plus PLC supports the implementation of subroutines. Designing our programs by utilizing subroutines allows us to use modular coding practices and makes our code readability and troubleshooting much easier.

In order to use a subroutine, you will simply right-Click Plus “Subroutine Program” and the “Add New Subroutine Program” selection will become available. In the Main Program, you will simply drag a “Call” Program Control Element into the Main Program and select the Subroutine Program you wish to call.



When you define and program your subroutine, you must add a “Return” instruction in order for the project to compile.

Click Plus Tag Naming Convention



We have defined a tag naming convention to help us understand the type and function by inspecting the tag name. Written somewhat like Hungarian Notation

The first letter will identify the data type. In our example “s” identifies our tag as a Single Word Integer. An “f” would identify the tag as a floating point, a “t” would represent a timer.

The second letter will identify if the tag is a Write to the Allen-Bradley CompactLogix Pod Controller or a Read from the CompactLogix.

We follow the first two-letter identifier with a readable tag name.

Be aware, most advanced PLC programmers will create some type of naming convention so finding out the meaning behind the tag names can help understand the PLC code.

How Do You Know What We Want You to Program?

- Unlike other software disciplines, the PLC coding discipline has not reached the maturity level of formally gathering and documenting requirements then verifying and validating that you've "programmed the thing right and that you've programmed the right thing"
  - Even those disciplined enough to do verification and validation efforts will allow the requirements and documents to get stale
- We will use State Diagrams and State Transition Tables in order to convey what we want you to program and to verify that you have indeed achieved the programming goal

PLC programmers have earned the reputation of "coding cowboys," or "code slingers," because in many cases formal requirements have not been documented, leaving the interpretation of machine operation and understanding of the process to the PLC coder. They oftentimes "code until it works," without a full understanding of operation and, more importantly, what error conditions should be programmed.

We have chosen to introduce State Diagrams and State Transition Tables as a means of conveying what we want our programs to do. We are not trying to turn everyone into Finite State Machine experts but rather bring some formality to requirements documentation and offer a way of verifying and validating program operation.

State Diagrams and State Transition Tables References

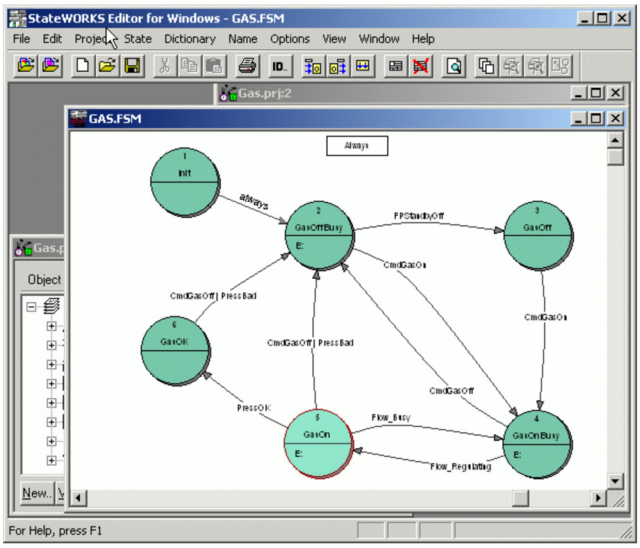
Modeling Software with Finite State Machines

A Practical Approach

Ferdinand Wagner
Ruedi Schmuki
Thomas Wagner
Peter Wolstenholme

Reference: http://www.stateworks.com/book/book/

StateWORKS Studio Editor - screen shots



State Diagrams and State Transition Tables are not new nor invented for this course but rather are being used by some software development firms to accurately describe the “States” and “State Transitions” within their software.

References to State Diagrams and State Transition Tables can be found in the book Modeling Software with Finite State Machines.

Reference:
http://www.stateworks.com/book/book/

When Do We Need State Diagrams and State Transition Tables?

- A control system determines its outputs depending on its inputs
  - Inputs could be physical or logical
- If the present input values are sufficient to determine the output, the system is a ***combinational system*** and does not need to support the concept of state therefore not needing State Diagrams
  - Example: If a switch is used to turn on a light, there isn't any state information that is required. This is like a light switch in a room.
- If the control system needs some additional information about the sequence of input changes to determine the outputs, the system is a ***sequential system***
  - Example: If you program a four-digit combination lock, you must remember the state of all four entries before allowing the unlock state

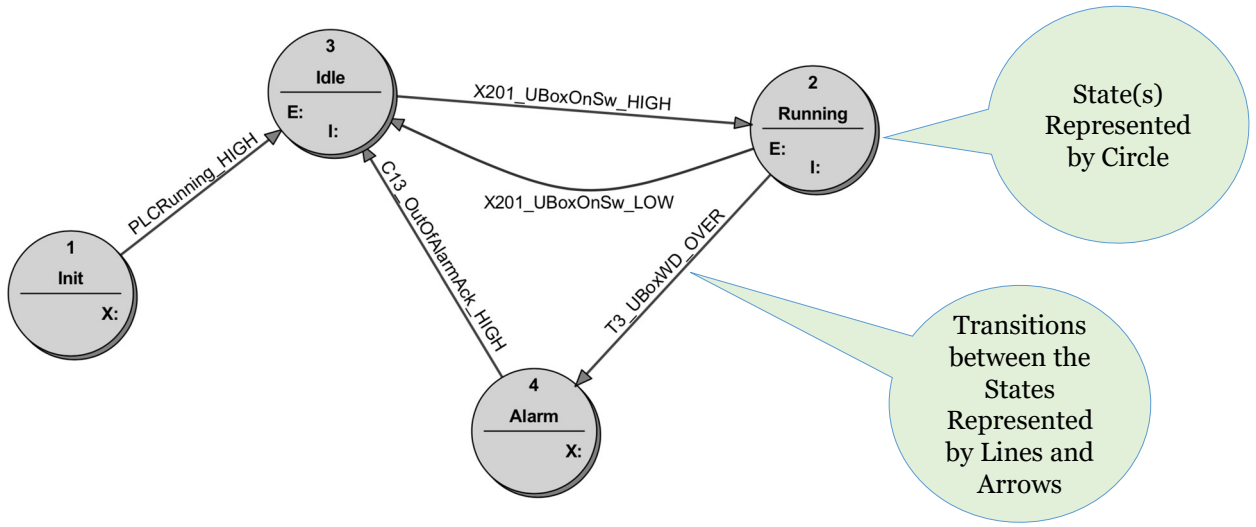
Taken from the book Modeling Software with Finite State Machines, we find that not all simple programs require State Diagrams to fully explain operation. Such systems are known as "combinational systems." However, for many machines or processes, it is necessary to understand all of the information that got the program to its current point; this is not a simple one-to-one mapping of input status to output state. These types of systems are referred to as "sequential systems."

This is an important point because most PLC programmers will start programming without full knowledge of all the possible states and conditions that should cause an alarm or how to properly transition from an alarm state back to a proper running state. In a lot of cases, the programmer simply codes and runs the machine or process until an unplanned scenario arises. Then the program is changed to meet the condition without going back to investigate other possible unplanned scenarios.

To create a State Diagram and a State Transition Table, it is necessary to have conversations and find agreement with the mechanical designer, the process engineer, the sales team members (what are the customer's expectations?), the customer, and other actors that hold some piece of the operational puzzle.

Reference:
http://www.stateworks.com/book/book/

State Diagram for Useful Box Hack



Let's start by understanding "States" and "Transitions". A formal method for defining how many states a machine should have does not exist so an engineer will identify the various states of operation. Oftentimes a mechanical engineer and/or a chemical engineer or other subject matter expert will understand the possible machine states.

In our example, we are going to inhibit the Useless Box motor from running if we energize a "C1" coil tag named C1_HackUBox.

States
In our example above we have the following Useless Box states:
Init (Initialize)
Idle
Running
Alarm

Initialize is used to give us a State Machine starting point. It should also be noted that the StateWORKS software enforces an Init state for their simulation engine.

The Idle state is used when the Useless Box is waiting for the switch to be thrown to the ON position.

The Running state is defined when the Useless Box switch is in the ON position and motor circuit has the ability to run.

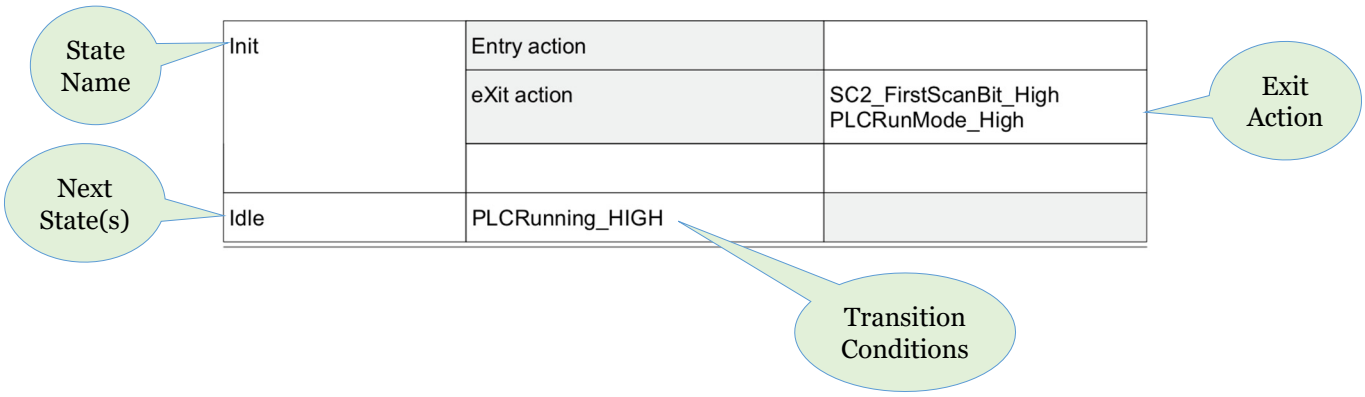
The Alarm state is active whenever an abnormal condition exists.

Transitions
To move between the states, one must define the transition conditions. A condition could be a digital value changing or an analog value reaching some key value.

It should be noted the diagram states contain the letters E, X, and I within the state bubbles. These stand for "Entry", "Exit" and "Inputs" respectively and will be discussed in detail in the next slides.

The creation of State Diagrams and State Transition Tables can be time-consuming and oftentimes can be uncomfortable to discuss because working through this layer of detail can be foreign to automation engineers. We also find that “nailing down” the exact operational behaviors can frustrate the team because no one person has the answer to the question, “How should this machine behave?” In the automation of machines that behavior is dependent upon who programmed the machine rather than reflecting the proper machine sequencing defined by Sequence of Operation documents, State Diagrams, State Transition Tables or some other document that conveys proper operation.

State Transition Tables for Useful Box Hack (1)



Inside the states is information about:

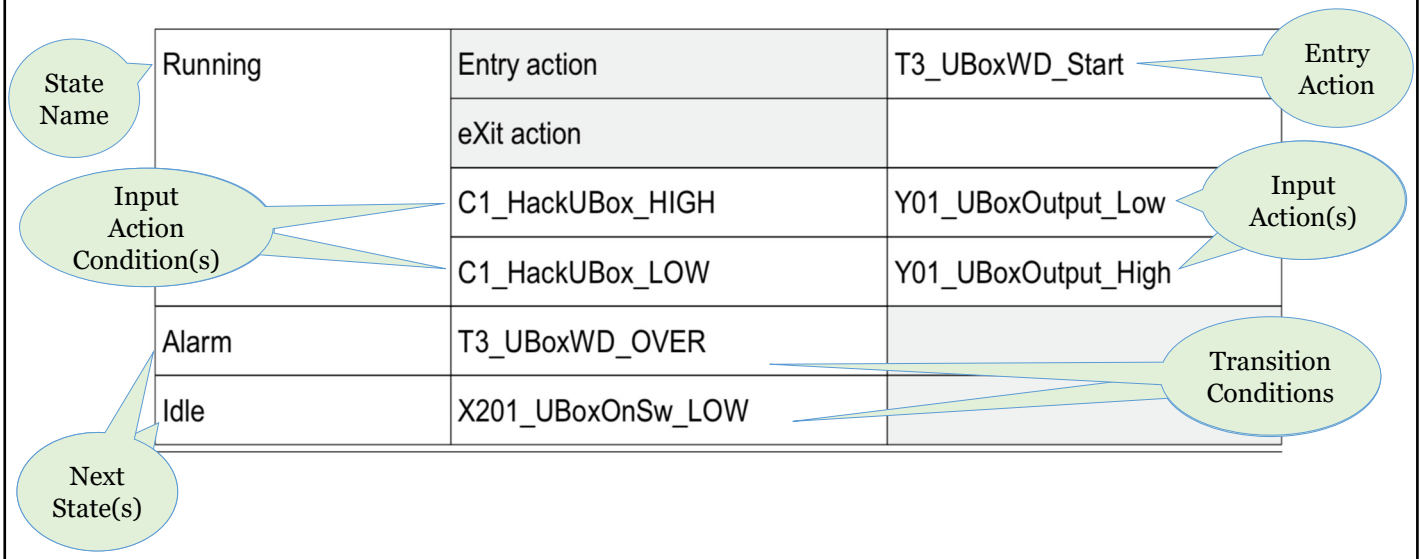
- The state name
- What is the next state we transition to?
- What condition would cause us to transition to the next state?
- Entry action? What variables do you want to change upon entering this state?
- Exit action? What variables do you want to change upon exiting this state?

This information is defined as a State Transition Table.

Moving through this type of exercise forces us to think about the conditions of the machine that we must handle in our PLC program.

In our example above, we have the Init (Initialize) state. We move to the Idle state once the PLCRunning bit is True or Boolean 1. As we exit the Init state and transition to the Idle state, we turn the SC2_FirstScanBit and the PLCRunMode bits to True or Boolean 1.

State Transition Tables for Useful Box Hack (2)

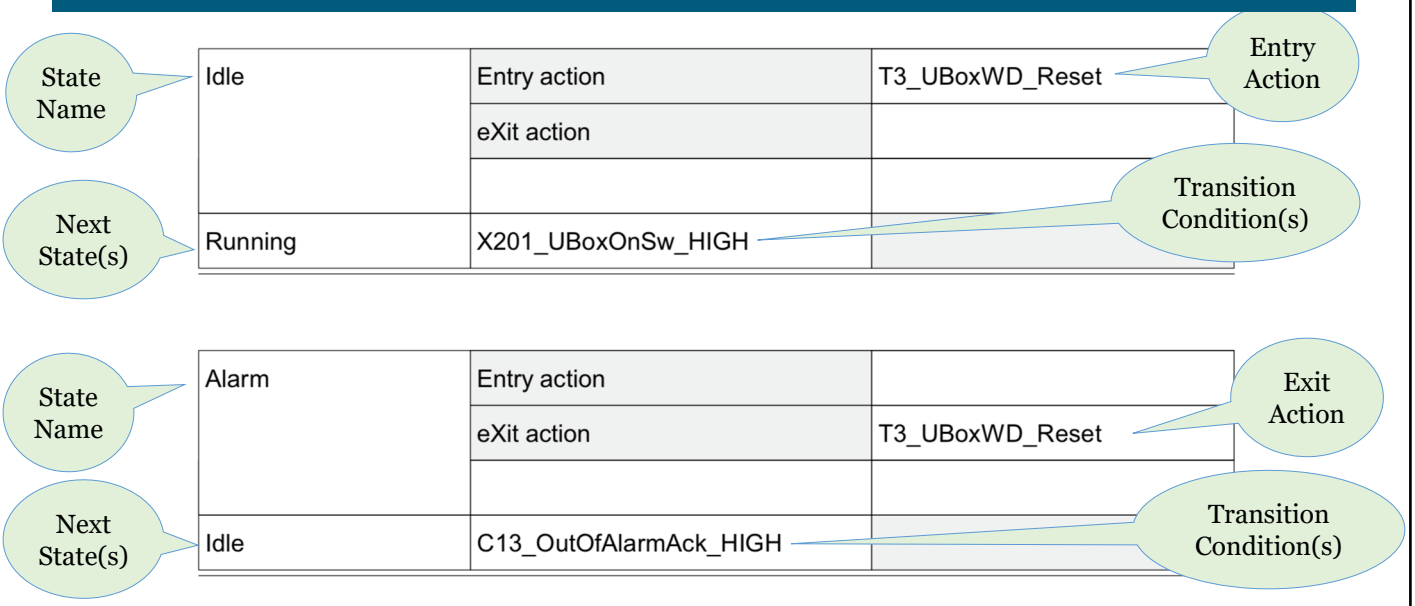


In our Running state example, we see we can move to the Alarm state if the Timer T3_UBoxWD (Watch Dog) timer is done (OVER). We would also move to the Idle state if the X201_UBoxOnSw is off or LOW. Please note, the X designation in the tag name represents a physical input to the Click Plus PLC. So, in this case, when we see the Useless Box switch move from the ON position to the OFF position, this would mean the Useless Box motor has run and flipped the switch from ON to OFF. In this case, we want to move to the Idle state and wait for the next flip of the switch.

Let's continue reviewing the State Transition Table. In the Entry action field, we see we want to start the T3_UBoxWD timer. This watchdog timer is used to transition us to the Alarm state.

Now let's examine the "Input Action Conditions" and "Input Actions" fields. While in a defined state, there will be variables and actions that will change during the state, but they are not variables that will move us to another state. For instance, we may see the C1 variable move from True or HIGH to a False or LOW. This in itself is not a condition that will cause us to transition to another state, but it is a variable that could change while we are in the current Running state.

State Transition Tables for Useful Box Hack (3)

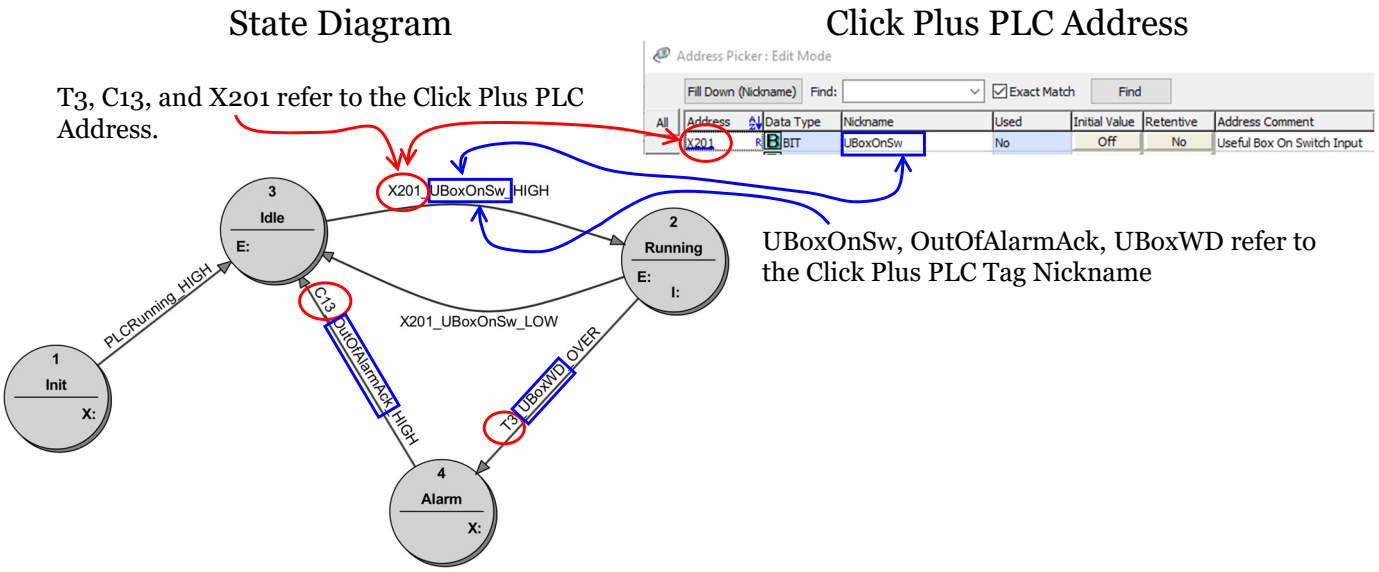


As we continue with the review of our State Transition Tables, we see the Idle state and the Alarm state definitions.

During the Idle state, we are waiting for the Useless Box on switch to transition from a Low to a High condition.

The Alarm state will move to the Idle state once the C13_OutOfAlarmAck bit is High. This can be interpreted as "take us out of Alarm condition once the Alarm(s) are Acknowledged and an Alarm condition doesn't exist".

Click Plus PLC Variable Naming Convention (1)

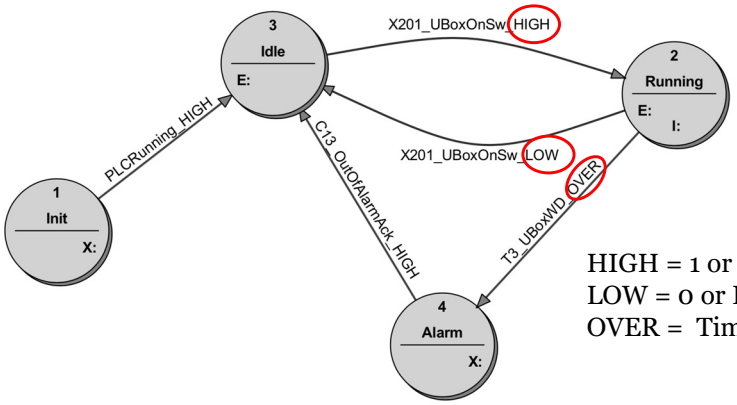


Some of the challenges of PLC programming is moving from a State Diagram and State Transition Tables document to actual PLC code. We try to lessen the confusion by mapping the PLC tag names to the State Diagram and State Transition Tables names.

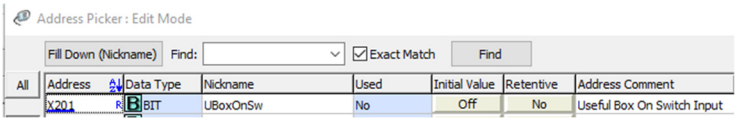
While not a completely accurate one-to-one mapping, we see the naming conventions can be represented closely enough to cross-reference both artifacts.

Click Plus PLC Variable Naming Convention (2)

State Diagram



Click Plus PLC Address



HIGH = 1 or True
LOW = 0 or False
OVER = Timer Done

These indicate state and are typically built into the PLC software and don't need additional definition

We see here the conventions for HIGH, LOW, and OVER are meant to represent a PLC 1 or True, a 0 or False, and a Timer Done respectively.

Lab 1.3: PLC Programming and I/O Integration

Go to the Lab Workbook: Lab 1.3

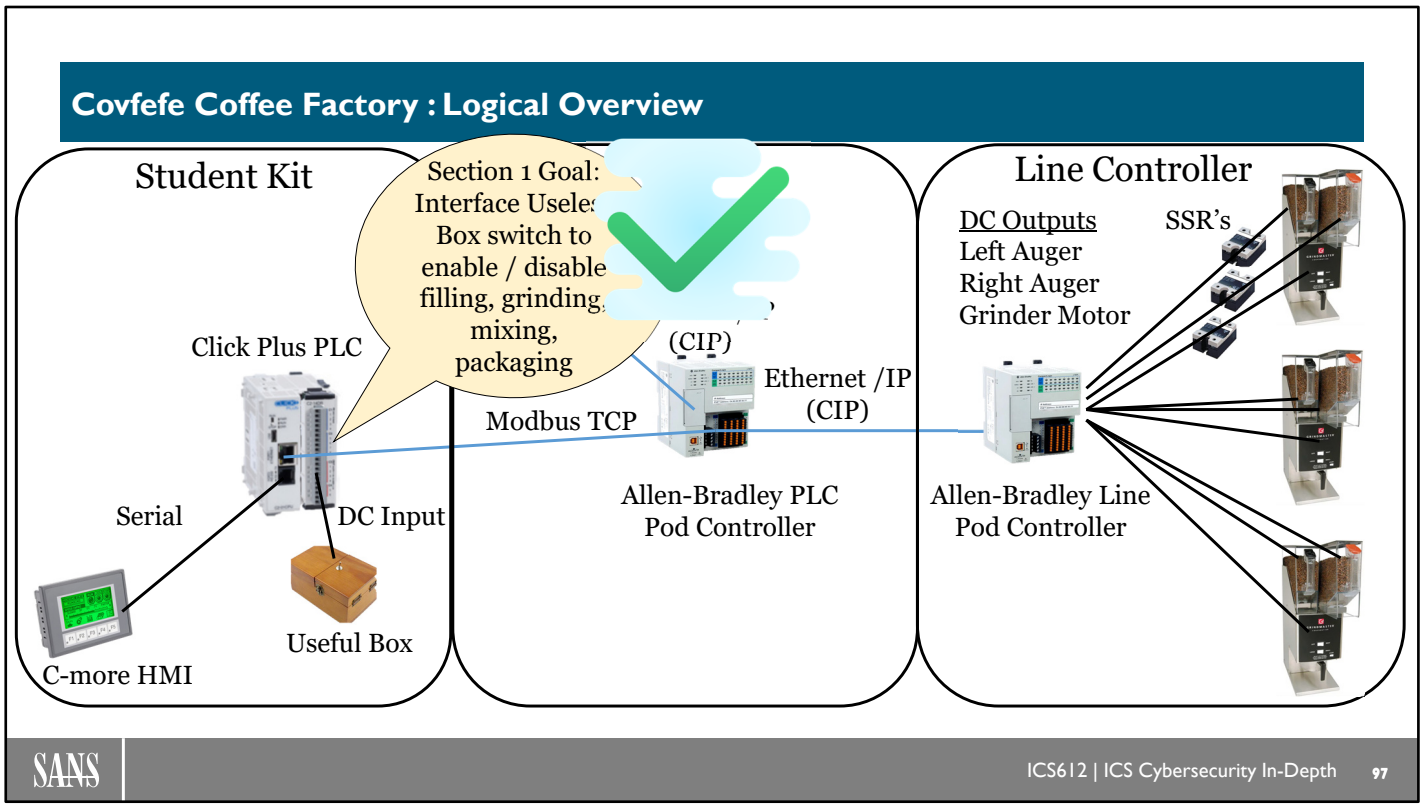
This page intentionally left blank.

Programmable Logic Controller Programming Checkpoint 1.3

Within operational environments you will encounter greenfield projects that are designed from scratch, brownfield projects which are modifications or upgrades to existing environments, and a wild variety of swamp fields, surprise fields, and the rare field of dreams projects. All start with an operational plan, translated in P&ID, PFDs, wiring diagrams, and ultimately implemented across engineered ICS components. You have:

- Verified the wiring of your Useless Box to the Click Plus PLC
- Created a PLC program to control your Useless Box with your Click Plus PLC
- Created a Useless Box local control inhibit where the Useless Box switch is disabled

This page intentionally left blank.



As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. In the middle of the slide, you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the left contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.

ICS612 Section 1 Outline (4)

- Process Environment Familiarization
- Lab 1.1: Virtual Machine(s) Setup
- Lab 1.2: Student Kit Familiarization
- Programmable Logic Controller Programming
- Lab 1.3: PLC Programming and I/O Integration
- Process Interface
- Lab 1.4: Integrating Analog Input
- Lab 1.5: Local HMI Setup and Control
- Student Pod Integration
- Lab 1.6: Configure the Shared Pod Elements
- Lab 1.7: Connect Student Kits to the Shared Pod
- Lab 1.8: Process Interrupt through Student Kit
- Lab 1.9: Local Process Environment Mapping

This page intentionally left blank.

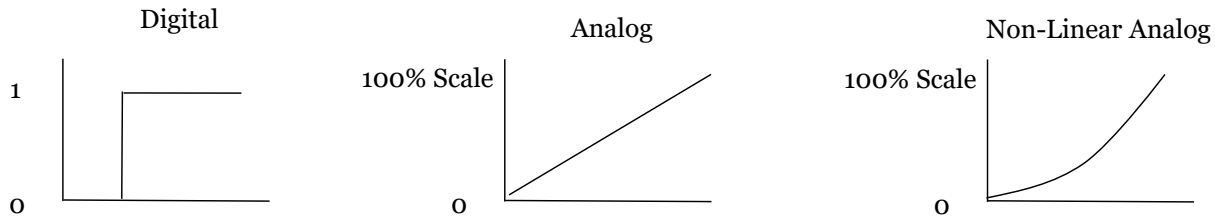
Process Interface

Local Control
Process Monitoring
Alarming

This page intentionally left blank.

Digital and Analog

- Digital inputs and outputs only have two states
  - Binary zero (0), which indicates off or false
  - Binary one (1), which indicates on or true
- Analog inputs and outputs have variable states within a scaled range
  - 0 to 100% range
- Most Analog input signals, and control responses are non-linear



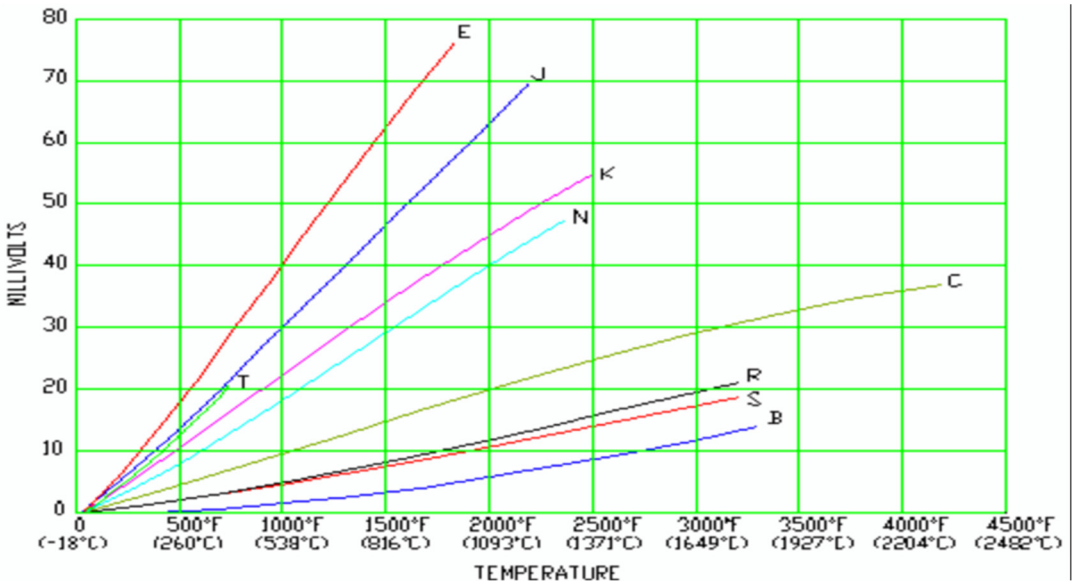
Any PLC manufacturer will support an array of input sensors and electrically drive many types of outputs. In the majority of cases, our input and output types are mostly either digital or analog.

Digital inputs or outputs (I/O) only have two states they can be in:
1. Binary zero (0), which indicates off or false
2. Binary one (1), which indicates on or true

Analog I/O can be operated within a variable range which will be defined in two manners. First, the analog operation will have voltage or current operational and secondly a specification for the digital resolution for the sensor or output. For example, an analog output may operate between 0 and 10VDC with 16 bits of resolution. This means that for the 0 to 10VDC range using a 16-bit Analog-to-Digital converter the resolution is 0.153mV per bit change.

Most processes and sensors do not have a linear input or process response to a linear output. In other words, sensor feedback in most cases will not have a linear response. Also, a process like temperature control or filling rates will not change linearly to linear outputs. This is why controlling a non-linear process is so challenging.

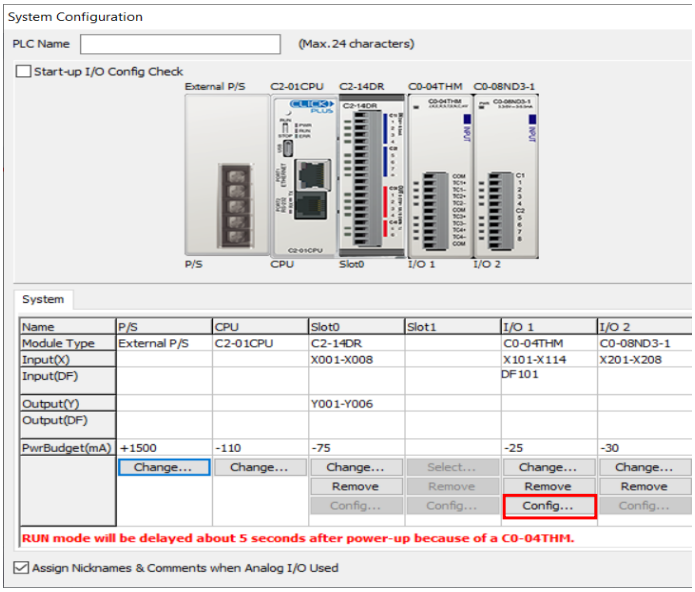
Thermocouples Are Non-Linear Analog Inputs



To provide an example, look at the thermocouple millivolt output on the Y axis as the temperature changes on the X axis. We see how the different types of thermocouples respond in a non-linear fashion.

Intro to the Lab 1.4 – Thermocouple Configuration

- Thermocouple modules have the non-linear curves built into the firmware to compensate for each type of T/C
- You will change the system configuration to accept a type K thermocouple

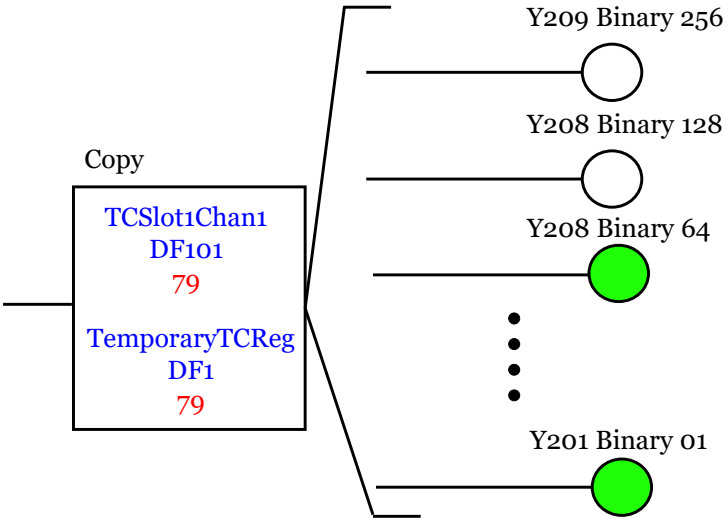


To compensate for the non-linear behavior of the thermocouple inputs, a thermocouple input module will have these non-linear profile curves programmed into the firmware to compensate for the non-linearity of each type of thermocouple.

In this next lab, you will configure the thermocouple input module to accept a K-type thermocouple, thereby allowing the module to specifically use the K-type non-linear curve to report an accurate temperature.

Intro to the Lab 1.4 – Parse out Current Value and Display as Binary

- Assign the current temperature to a floating-point register
- Write a PLC program using output bits Y209 through Y201 to represent the binary equivalent of the current temperature
- You will use these bits in a C-more HMI project later

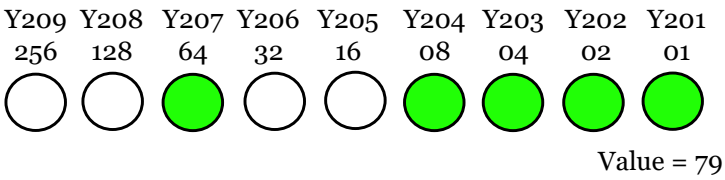


In this next lab, you will also map the current floating-point temperature as reported by the thermocouple input module. This is represented in a binary fashion. You will map the binary representation to output bits Y209 through Y201.

Note: The "Y" represents the Click Plus digital output image bits.

How to Tackle Lab 1.4? Parse out Current Value and Display as Binary

- We can start by assigning outputs to represent a binary value. In our example Y201 represents binary 01, Y202 represents binary 02, Y203 represents binary 04, etc.
- The temperature value will be broken into binary pieces to represent the whole value. For example, the value 79 is represented as 64+8+4+2+1



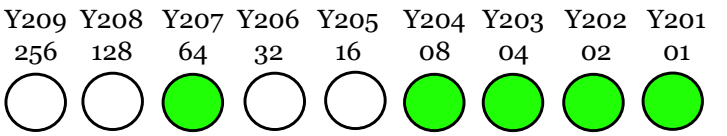
If we want to represent a whole value using a binary system and display this value on our PLC 's outputs, we must first assign the values to our outputs.

In our example, Y201 will represent binary 1, Y202 will represent binary 2, Y203 will represent binary 4 and so on.

We can take a value such as the number 79 and break it down into a binary representation such as binary 64 + binary 8 + binary 4 + binary 2 + binary 1 is equal to the number 79.

Programming Lab 1.4: Parse out Current Value and Display as Binary (1)

- Next, we need to figure out how to take the current value and break this into the binary representation
- Start by storing the current value into a temporary math register. This is a "best practice" as you don't work with the actual raw value because it may change during your math calculations



Actual Temp = 79

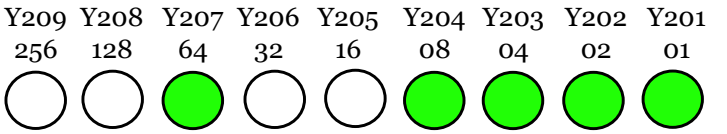
temp = Actual Temp

temp = 79

As a best practice, we should store the actual temperature value into a math register we can manipulate because the actual temperature value may be changing as we solve our logic, and we will want to subtract binary values from the actual temperature in order to turn on the appropriate outputs.

Programming Lab 1.4: Parse out Current Value and Display as Binary (2)

Now, we need to compare our "temp" variable against each binary value to determine which outputs to turn on to represent our value. We do this by subtracting the largest binary value and storing the remainder for our next comparison



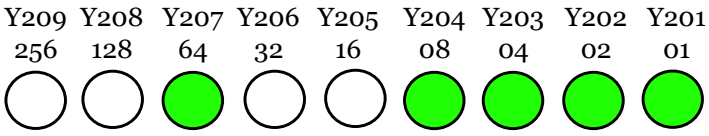
Actual Temp = 79

temp = Actual Temp		temp = 79
If temp >= 256 then Turn on Output Y209 temp– 256 = temp	FALSE	temp = 79
If temp >= 128 then Turn on Output Y208 temp – 128 = temp	FALSE	temp = 79
If temp >= 64 then Turn on Output Y207 temp – 64= temp	TRUE	temp = 15

In order to determine which binary values will represent our temperature, we will subtract the first binary value when our temperature variable is greater than or equal to the binary value. After we subtract the binary value from the temperature variable, we will store the remainder and compare the remainder to the rest of the binary values.

Programming Lab 1.4: Parse out Current Value and Display as Binary (3)

We continue to compare the remainder against the next largest binary value, subtracting this value from the remainder until we reach complete all binary values and the remainder value is zero



Actual Temp = 79

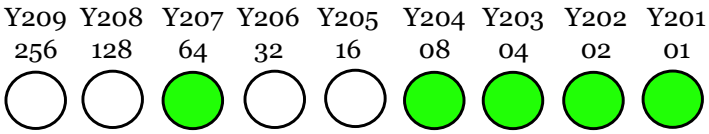
temp = 15

If temp >= 32 then Turn on Output Y206 temp– 32= temp	FALSE	temp = 15
If temp >= 16 then Turn on Output Y205 temp – 128 = temp	FALSE	temp = 15
If temp >= 08 then Turn on Output Y204 temp – 08 = temp	TRUE	temp = 7

We will continue to compare the remainder against the rest of the binary values until we find condition where the remainder is greater than or equal to the next binary value.

Programming Lab 1.4: Parse out Current Value and Display as Binary (4)

We continue to compare the remainder against the next largest binary value, subtracting this value from the remainder until we reach complete all binary values and the remainder value is zero



Actual Temp = 79

temp = 7

If temp >= 04 then Turn on Output Y203 temp– 4= temp	TRUE	temp = 3
If temp >= 02 then Turn on Output Y202 temp – 2 = temp	TRUE	temp = 1
If temp >= 01 then Turn on Output Y201 temp – 1= temp	TRUE	temp = 0

We continue subtracting the remainder each time we find the condition where the remainder value is greater than or equal to the binary value we are comparing. We do this until we complete the comparison of binary one. By doing this, we will determine the binary representation of the temperature value.

Lab 1.4: Integrating Analog Input

Go to the Lab Workbook: Lab 1.4

This page intentionally left blank.

Process Interface Checkpoint 1.4

Within control system environments, the various engineered sensors and actuators in use often have specific input/output module or controller requirements based on the signal ranges and tolerances. Understanding the various components and the necessary settings is critical to establishing a safe and reliable process. You have:

- Validated the wiring of a K-type thermocouple to your Click Plus PLC
- Modified the setup configuration of the thermocouple module to accept a K-type thermocouple
- Displayed the current value in the Click Plus PLC register DF1
- Parsed out the current value into a binary temperature format

This page intentionally left blank.

Electronic Operator Interface (EOI) vs. Human Machine Interface (HMI)

- Electronic Operator Interface (EOI) typically used "on machine" as a backup to the Human Machine Interface (HMI)
  - EOIs are used for local control and most often used in non-connected "skids" or sometimes for backup of the main HMI system
  - EOIs typically run an embedded OS and are not dependent on remote servers
- You may hear references to EOI like "local HMI", "PanelViews", "On Machine HMIs", "Local Control HMIs"
  - C-more Micro will act as an EOI throughout the course
  - Automation Direct refers to the C-more Micro as a touchscreen HMI
- The lessons for this section will use the C-more Micro HMI to communicate to the Click Plus PLC via serial cable and Modbus

We need to understand the process and operational requirements in order to give the operator the tools they require to keep the machine or process running. In some cases, an Electronic Operator Interface (EOI) is added directly on a machine so the operator can have line-of-sight control or in the rare case when higher-level HMIs are not available.

In this class, you will use your C-more Micro HMI to read status from your Click Plus PLC, to send commands to your Click Plus PLC, and ultimately through Modbus communications give and receive commands to the Pod CompactLogix PLC.

EOI and HMI Functionality to Consider

- Alarm and Event management
  - Database integration (i.e. are alarms and events in the same database?)
  - Synchronization and client awareness
- Integration of PLC tag discovery
  - Can the HMI "ask" the PLC for tags directly or is it asking a third-party driver package?
  - Does the HMI maintain a separate tag database? How does it synchronize with the PLC during "Creating new tags, Reading the existing tags, Updating tag information and Deleting tag" (C,R,U,D) operations?

When we implement HMI or EOI solutions, we not only need to consider the graphics capabilities, but we also need to consider how well our HMI choice will allow us to manage normal and abnormal conditions. For the engineering designer, we also need to consider how easily we can discover and map PLC tags to our HMI software.

In some cases, the PLC vendor will also have an HMI solution that offers better PLC tag discovery mechanisms, which can lead to an easier implementation. However, some third-party solutions offer better usability and a wider range of PLC vendor connectivity; this can be beneficial to those plants that have multiple PLC vendor solutions installed.

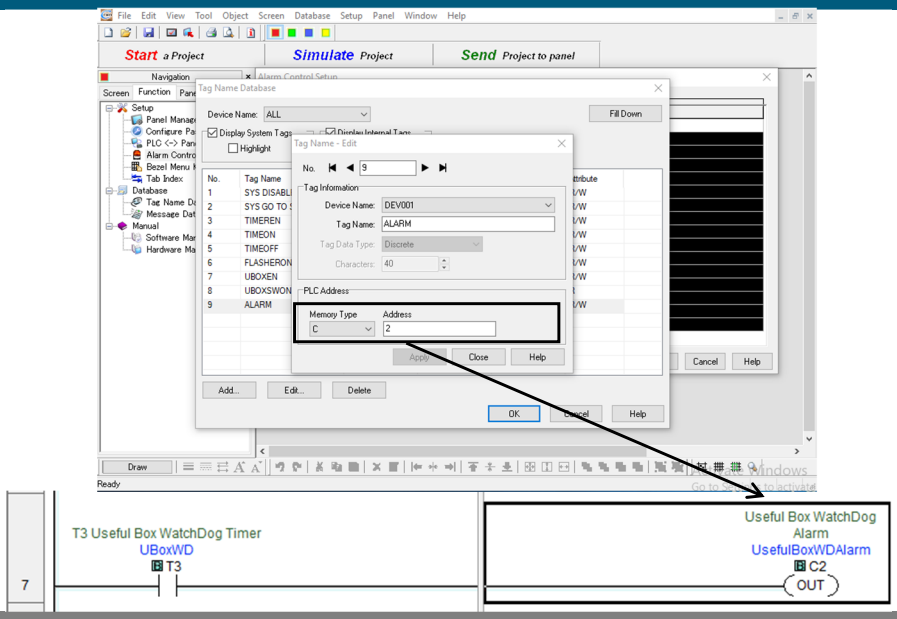
Alarms – What Are Alarms and Where Are They Defined?

- Alarms are special events
  - For forensics it would be "convenient" if events and alarms were part of the same database, but which database? The HMI database, the PLC database, or maybe a Historian database?
- Should alarms be located in the HMI vs. alarm bits in the PLC?
  - The answer to this question lies in the answer to another question –> "Does the PLC have to do something when an event or alarm occurs?" Most times the answer is "yes"; so the PLC will need the most current status of local alarms.

Alarms and alarm management are critical to situational awareness and a considerable amount of design time should be spent in the area of alarm states. In many offerings, the PLC database and the HMI database are different and, in some cases, within an HMI package, the events and alarm databases are also separate. This can present a challenge when trying to do alarm or event forensics.

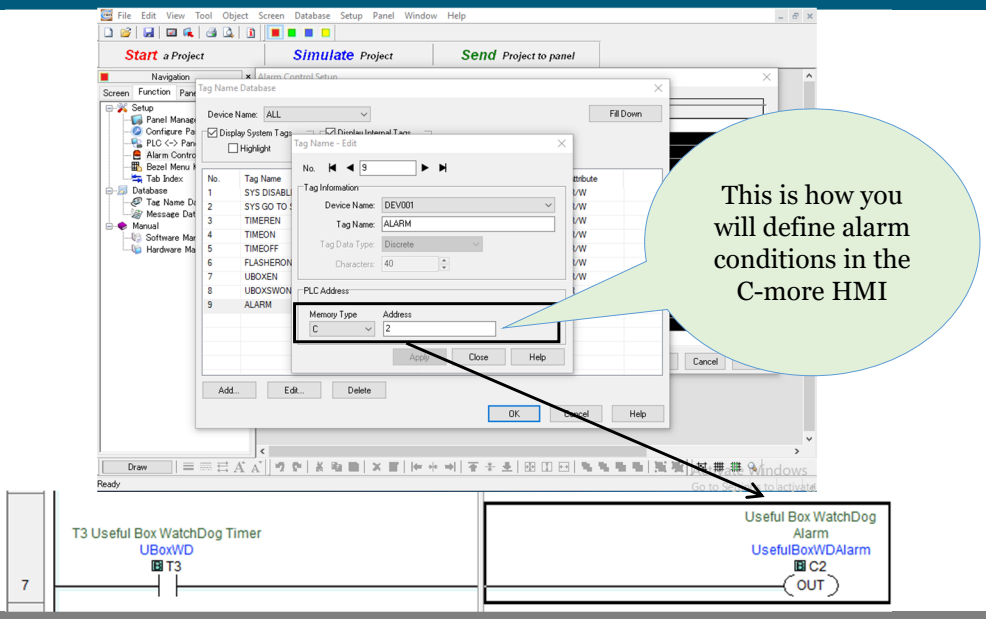
In almost every case, a process alarm must be ultimately acted upon by the PLC. Therefore, the PLC should be generating the alarm, while the HMI is a window into the PLC alarm. While alarm conditions can be written into the HMI and sent to the PLC, it is important to understand that this can be unreliable.

Most HMI Packages Scan Bit(s) in the PLC That Are Defined as Alarm Bit(s) (1)



With most HMI packages, you are allowed to define alarm tags that map to a PLC tag. In the example above, we have defined coil memory location “C2” to represent a Useful Box Watchdog alarm. We define and map that tag in the C-more HMI software so it can be related to an alarm condition in our next lab.

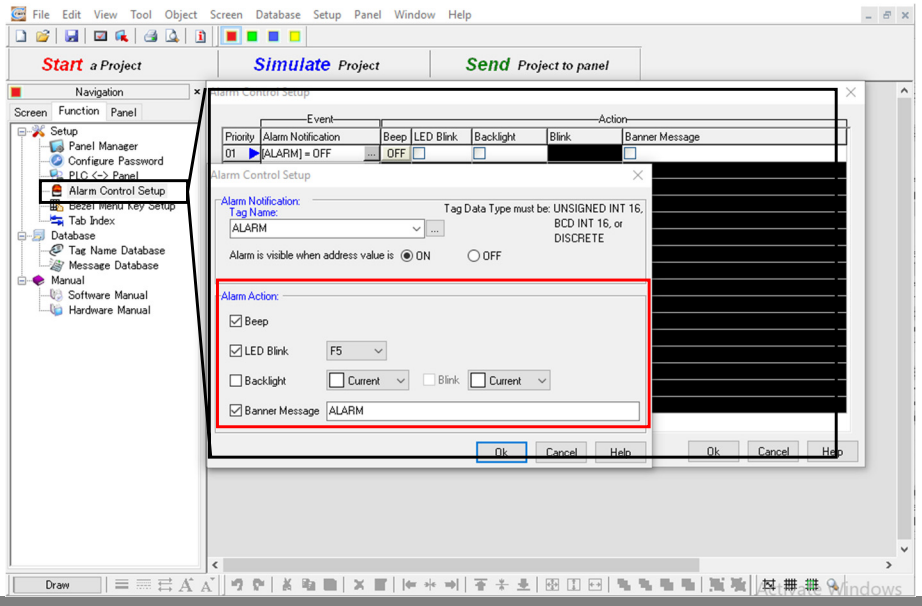
Most HMI Packages Scan Bit(s) in the PLC That Are Defined as Alarm Bit(s) (2)



While we identify and map the alarm tag to a PLC alarm bit, we can see that coordinating the PLC definitions with the HMI software package can be inaccurate. For instance, I have no visual indication in the HMI software that I have mapped the correct PLC bit in my HMI project, nor do I really know if the PLC bit is being used properly.

In big projects, it's not uncommon for one person to be doing the HMI design while another person is working on the PLC code. We can see that coordination of memory location mapping is paramount in order for the HMI to have the full list of alarms that are defined in the PLC.

Most HMI Packages Have Alarm Communication and Tag Configurations



It is very common for an HMI package to annunciate alarm conditions through a banner, flashing a button, or sounding an audible alarm. With the C-more Micro HMI programming software we have the ability to beep, cause a function key to flash, and display a banner message on top of the screen.

Alarm Management Challenges

- Challenge -> How to propagate alarm and event status to subscribers?
  - An architecture with multiple HMI clients that are subscribed to alarms from a PLC may miss the alarm if the HMI is turned off or has a communication issue
- Challenge -> How to propagate acknowledgements from multiple clients?
  - How to tell what station and which user acknowledged an alarm can be a challenge
- Challenge -> Synchronizing multiple alarm sources between many HMI clients
- Challenge -> Coordinating alarm definitions within large definitions (i.e., which bits are alarms?)
- Challenge -> Efficiency of overtaxing the PLC and efficiency of network traffic (i.e., on change notice vs. constant scanning)

We mentioned earlier that alarm management can be quite challenging, but what does this really mean? Let's look at a few use cases that can help highlight what we mean by alarm management challenges.

First, when we have a PLC alarm condition, how can we make sure all clients receive the alarm? What happens if an alarm subscriber or client is powered off and then comes back online? Should the PLC be aware of all the clients and republish the alarm to those clients that weren't online? In some cases, this isn't possible and, in some cases, if the HMI is scanning the alarm bit, the operator will know the alarm exists.

In the second scenario, if we have a PLC alarm and one of the HMI stations acknowledges the alarm while another HMI alarm client was turned off, how will you synchronize the alarm states? Most higher-level HMI packages account for alarm database synchronization while most EOI packages can't and don't.

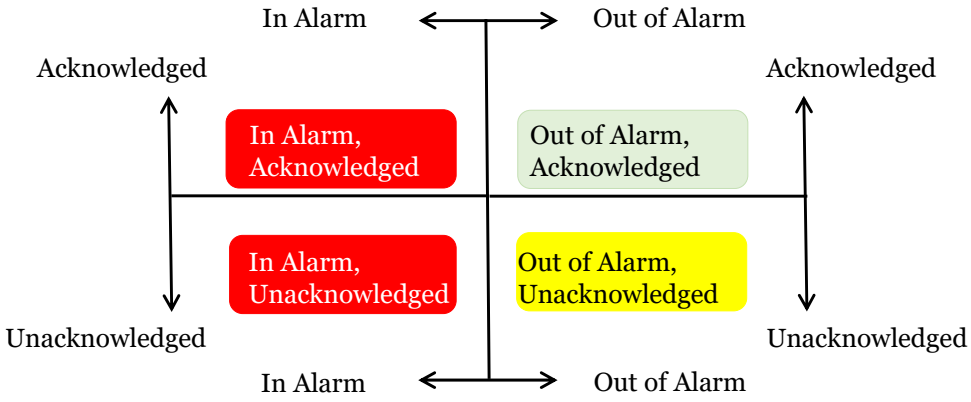
In the third interesting scenario, if we need to have alarms roll up to a central alarm-reporting station, how do we synchronize across multiple PLCs and across multiple lines in multiple areas? Again, for many higher-level HMI packages the application subscribes or becomes part of the larger HMI "universe" so it can synchronize alarm states. Another interesting conversation to dig into is timestamping of alarms and coordinating time across the above-mentioned boundaries. Alarm reporting and forensics is only as accurate as the timestamp, but we'll leave that deep conversation for another time.

It can also be quite challenging to coordinate which PLC bits will act as alarm bits for HMIs that scan the PLC. In any automation project, the tag definitions need to be well thought out and communicated between team members. As you may already know, the PLC coding process is pretty fluid and alarm definition bits will change as alarm scenarios are discovered or invented.

The PLC has limited communication capabilities that in most cases are a lesser priority than running and solving the program. An HMI architecture design should consider the PLC resource capabilities

before just connecting a bunch of HMIs to the network and polling the PLC for tags. Some of the newer-generation PLC and HMI offerings will allow the HMI to be notified when an alarm bit changes, vs. constantly polling the PLC tags. This leads to better network efficiency and reduces the possibility of overtaxing the PLC.

Alarm States (1)

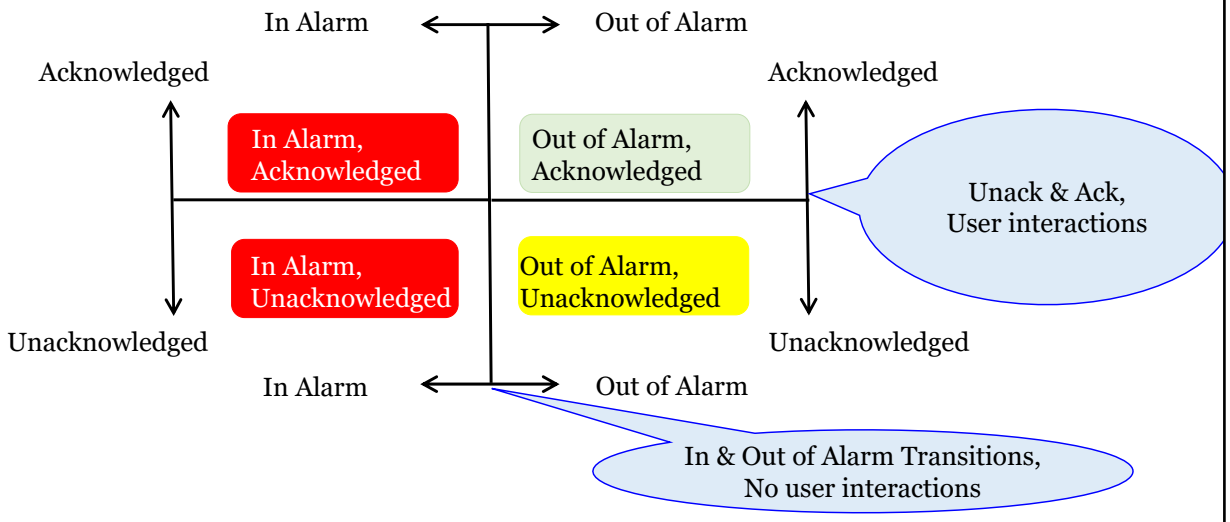


Alarms also have a well-defined "State Machine". An alarm condition can either be in or out of alarm and acknowledged or unacknowledged by the operator.

If we look at these four states, they are:

1. In Alarm and Acknowledged by the operator
2. In Alarm and Unacknowledged by the operator
3. Out of Alarm and Acknowledged by the operator
4. Out of Alarm and Unacknowledged by the operator

Alarm States (2)



We are stating the obvious, but it bears mentioning that the acknowledged and unacknowledged state of an alarm requires user interventions. This is important insomuch as you do not want to move between those two states programmatically. This can be a deep thought if we imagine it as a nefarious actor that works to promote an alarm from unacknowledged to acknowledged and let the HMI system remove the alarm without real operator intervention.

Alarm System Rules (1)

Two Golden Rules of Alarm System Design:
1) Never remove an alarm from the alarm banner or alarm screen without a user acknowledgement
2) Don't display an alarm unless you want the operator to "do" something

What is an Alarm? (ISA-18.2)

- An audible and/or visual indication to the operator that an equipment malfunction, process deviation or other abnormal condition requires a response.

	Operator Must Act	FYI to the Operator
Abnormal	Alarm	Alert
Expected	Prompt	Message

So, this brings us to a couple of simple golden rules with regards to alarm systems. There are more good guidelines for alarm management in the ISA 18.2 specification, but we'll mention two that are critical to alarm management.

1. Never remove an alarm from the alarm banner or alarm screen without a user acknowledgement
2. Don't display an alarm unless you want the operator to "do" something

Alarms are meant to spur operator action; you want operators to observe the alarm and "do" something about it. Also, you don't want an alarm system to remove the alarm announcement without the operator acknowledging they have seen the alarm.

Alarm System Rules (2)

Two Golden Rules of Alarm System Design:
1) Never remove an alarm from the alarm banner or alarm screen without a user acknowledgement
2) Don't display an alarm unless you want the operator to "do" something

Alarm systems are great targets because they are oftentimes the only window into "what's going wrong." If I can suppress an alarm, then I won't get operator intervention.

What is an Alarm? (ISA-18.2)

- An audible and/or visual indication to the operator that an equipment malfunction, process deviation or other abnormal condition requires a response.

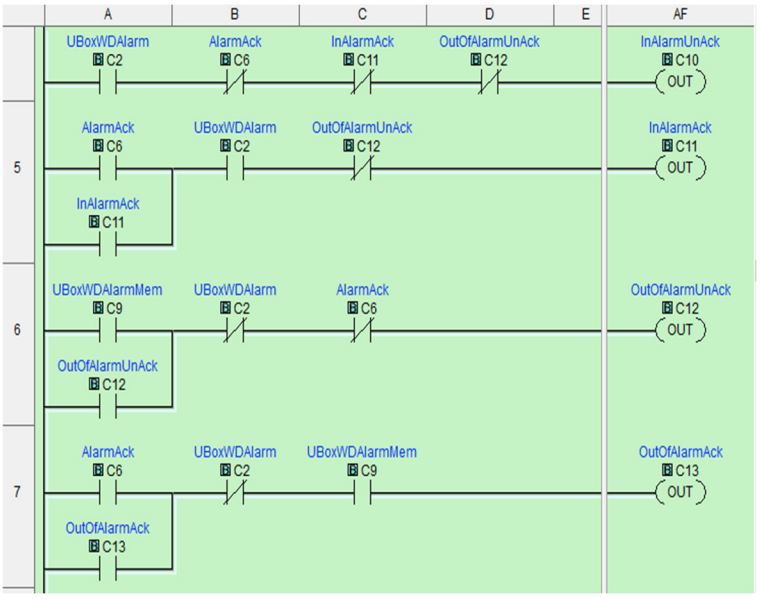
	Operator Must Act	FYI to the Operator
Abnormal	Alarm	Alert
Expected	Prompt	Message

Alarm systems are great targets because they are oftentimes the only window into "what's going wrong." If an alarm condition can exist but be hidden from the HMI station, the operator may not know of the critical situation because they have lost real-time visibility of the alarm and therefore cannot react. Also think about this: If the alarm system is compromised then the rest of the system should be considered compromised.

C-More HMI Alarm State Support

You will have the wonderful opportunity to investigate alarm state programming in the Click Plus PLC and see the visualization on the C-more HMI

- In Alarm : Unacknowledged (IU)
- In Alarm : Acknowledged (IA)
- Out of Alarm : Unacknowledged (OU)
- Out of Alarm : Acknowledged (OA)



In many cases, the EOI doesn't support the concept of the Alarm four-state machine. Remembering that an alarm can be:

1. In Alarm and Acknowledged by the operator
2. In Alarm and Unacknowledged by the operator
3. Out of Alarm and Acknowledged by the operator
4. Out of Alarm and Unacknowledged by the operator

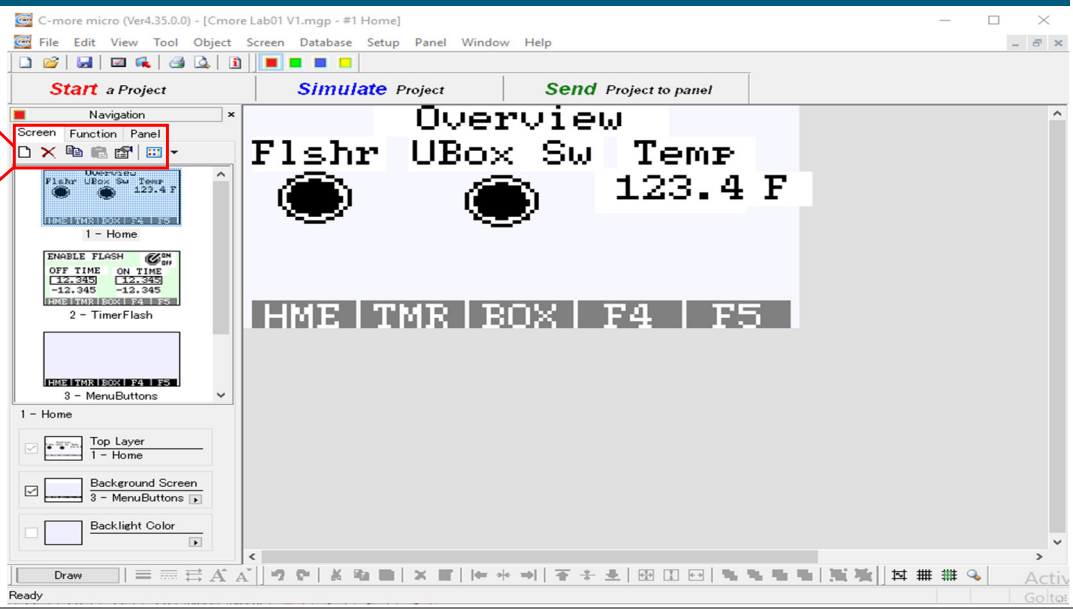
With that said, we have to build in the alarm states in the PLC and roll up that status to the EOI.

More fully featured HMI packages, like the Rockwell Automation FactoryTalk View product that you will be interacting with in future labs, handle the four states of the alarm conditions for you. For our C-more and Click Plus PLC student kits, you will need to be aware that alarm states and state management will be programmed in the Click Plus PLC.

C-More Micro Software Logistics (1)

Three Main Panels
- Screen Controls
- Function
- Panel

Screen Controls
- New Screen
- Delete Screen
- Copy Screen



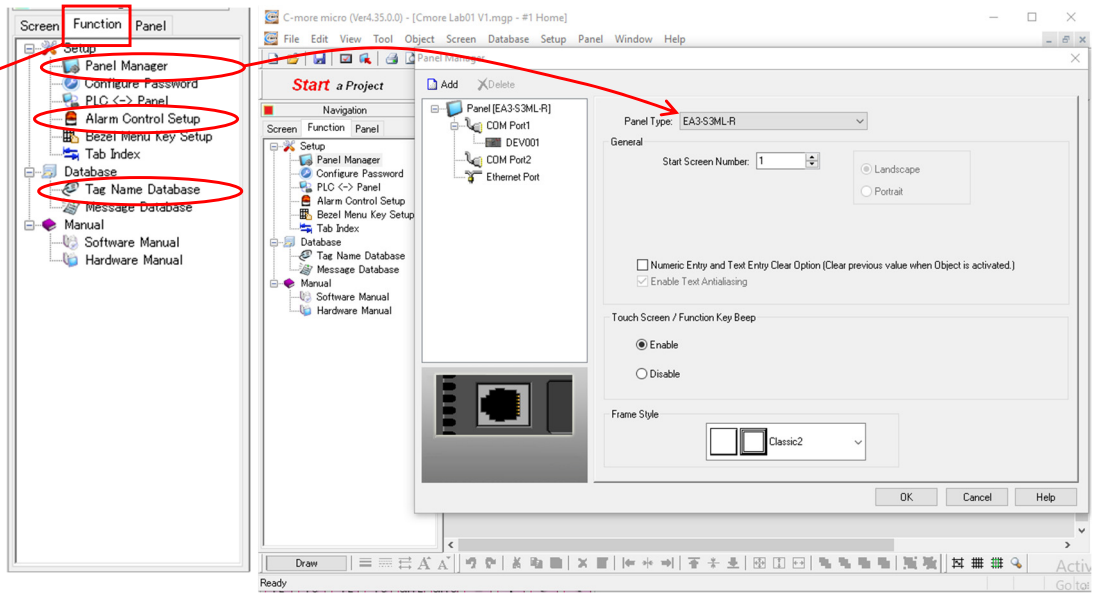
The C-more Micro is programmed and downloaded with the C-more Micro software. It has the traditional EOI/HMI features that can be found by exploring the three main navigation panels. They are:

1. Screen
2. Function
3. Panel

The Screen navigation panel allows you to create a new screen, delete an existing screen and make a copy of an existing screen.

C-More Micro Software Logistics (2)

Three Main Panels
- Screen Controls
- Function
- Panel

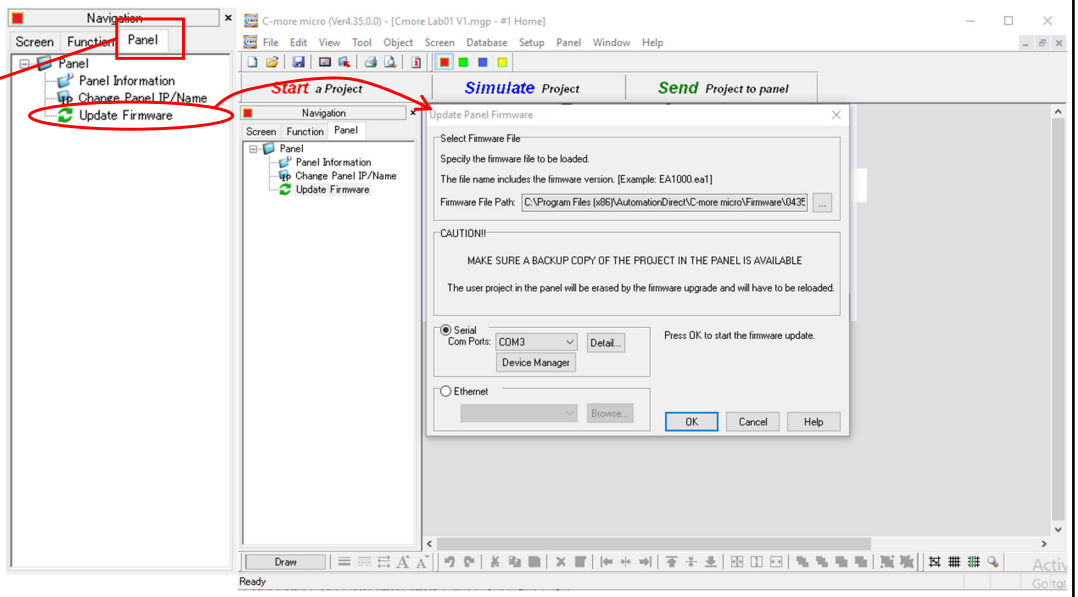


The Function navigation screen provides you with:

- Panel Manager, which allows you to define the hardware this project is going to be downloaded and running on
- Alarm Setup, which allows you to define the PLC or internal tags that will display alarm conditions
- Tag Name Database selection, which allows you to define HMI tags that will be used to monitor PLC registers and tags.

C-More Micro Software Logistics (3)

Three Main Panels
- Screen Controls
- Function
- Panel

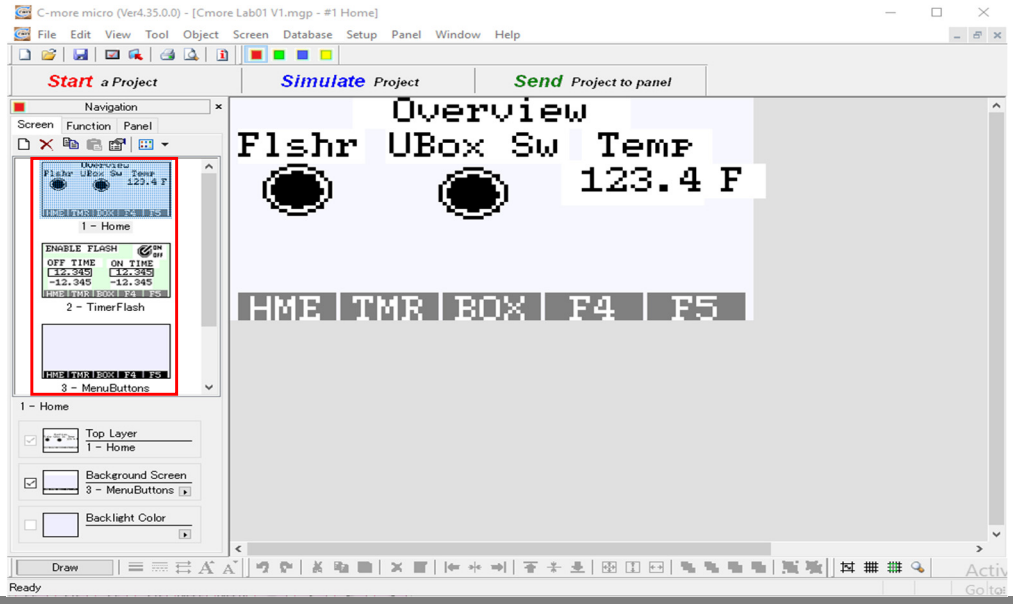


The Panel navigation screen allows you to manage the C-more Micro panel, which includes updating the firmware on the panel.

C-More Micro Software Logistics – Create Three Screens

In the next lab, you will modify three screens

- Overview
- Flasher Timer Enable
- Menu Buttons

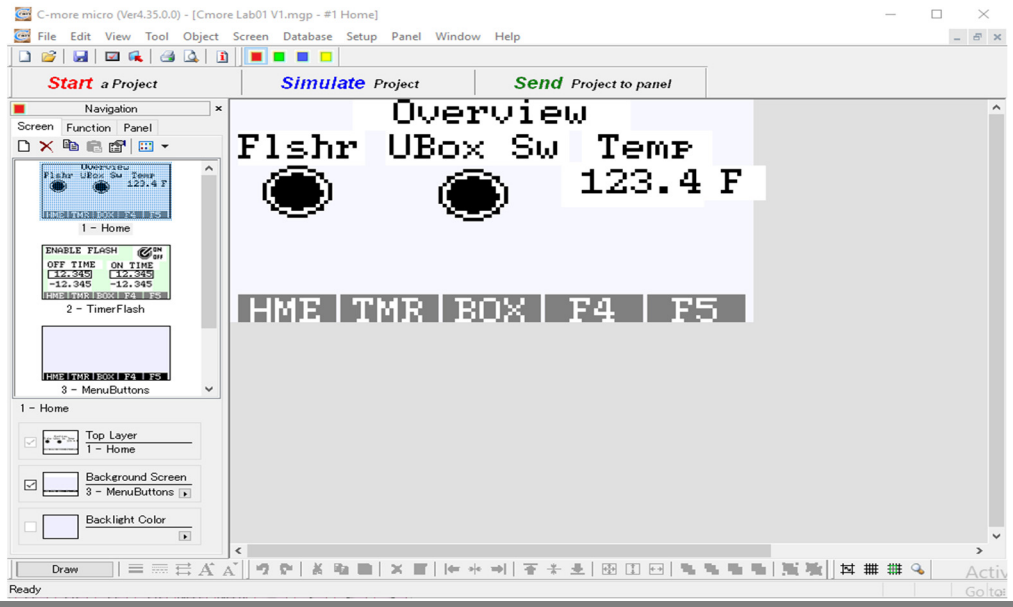


In the next lab, you will investigate the functions and native objects of the C-more Micro HMI software. You will open up a C-more project, change some of the existing objects, and also, create new objects by using the copy functionality. You will learn how to map PLC tags to animate the screen objects.

C-More Micro Software Logistics – Overview Screen

Overview Screen contains

- Flasher On Status Indicator
- Ubox On Position Indicator
- Temperature Readout
- Menu Button background screen



The Overview Screen contains several objects, such as:

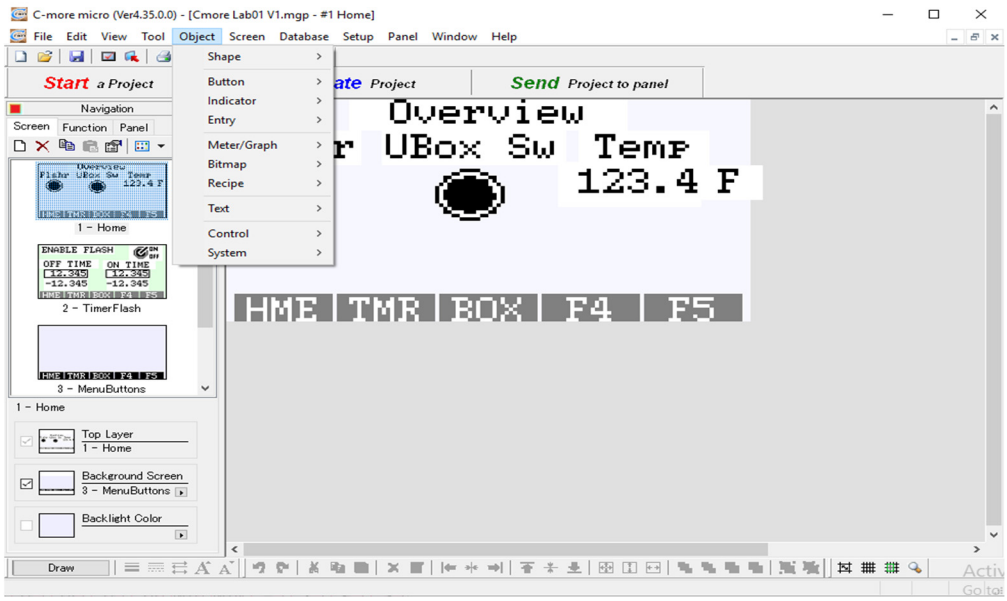
- Flasher On Status Indicator
- Ubox On Position Indicator
- Temperature Readout
- Menu Button background screen

These objects are mapped to Click Plus PLC registers; you will have an opportunity to investigate the mapping capabilities and object properties.

C-More Micro Software Logistics – Use the Object Menu

Overview Screen contains

- Flasher On Status Indicator
- Ubox On Position Indicator
- Temperature Readout (Indicator Display)
- Menu Button (Background screen)

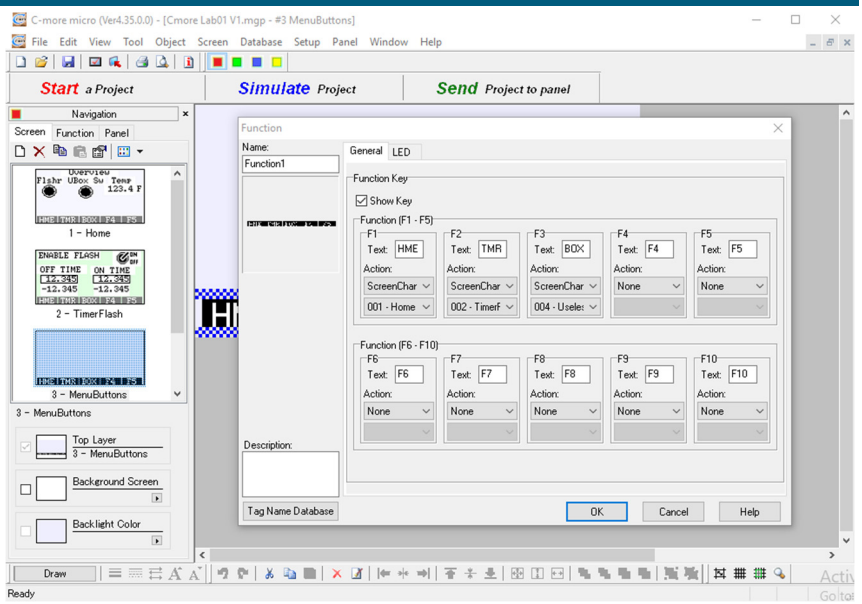


Most of the object definitions and creations start with the Object menu item found across the top of the menu bar.

C-More Micro Software Logistics – Creating the Menu Background Screen

Menu Button Screen

- Assign functionality to the HMI function keys
  - Object -> System -> Function
- This is much like "Visio," where you can use this for a background screen instead of re-creating the menu keys on each screen

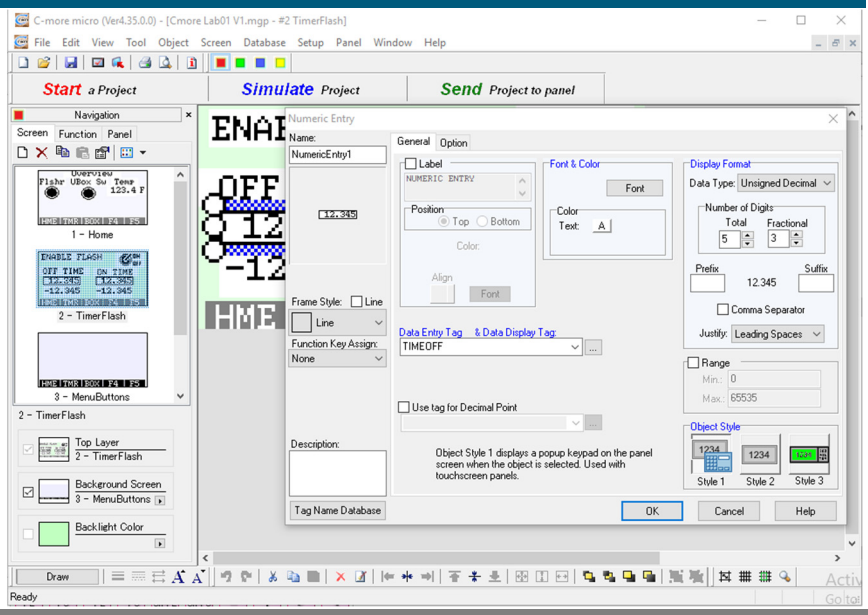


The function keys on the C-more panel can be mapped to change the screen being displayed. The functionality can be found under the Object menu, under System and then under Function. You will see we have created a Menu Button screen that will be used much like a Visio background screen. On your other screens, you will check the "Background Screen" and select "Menu Buttons". This enables you to reuse the Menu Button screen instead of re-creating the screen change functionality.

C-More Micro Software Logistics – Creating the Timer Flasher Control Screen (1)

Timer Screen

- You will create timer on entry with respective timer accumulated value readout
- Timer Setpoints will be created from Object -> Entry -> Numeric Entry
- Timer accumulated display readouts will be created from Object -> Indicator -> Numeric Display



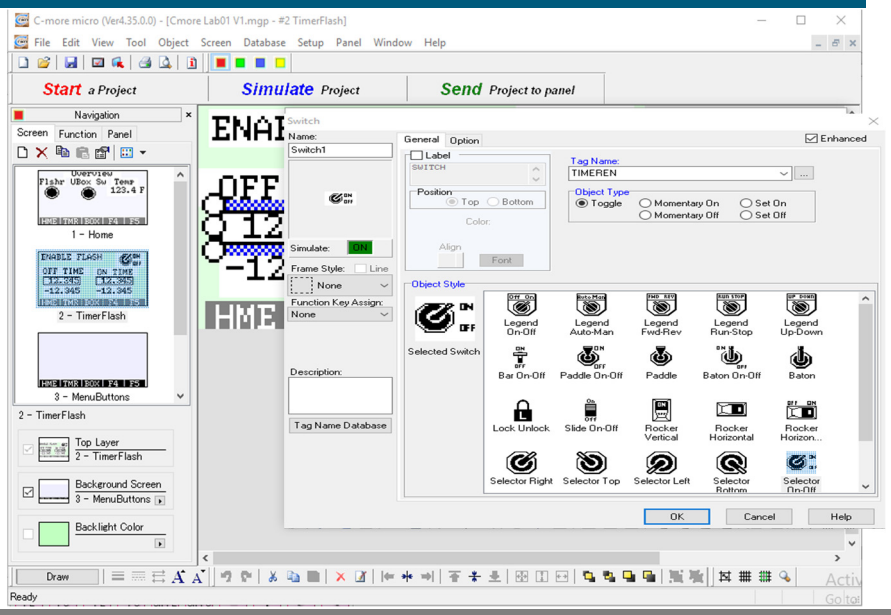
We have included a Timer Control screen where you can enter a time setpoint for the Flasher output to stay on and you can also enter a time setpoint for the flasher output to stay off.

Timer Setpoint entries are created from Object -> Entry -> Numeric Entry, while Timer accumulated display readouts will be created from Object -> Indicator -> Numeric Display.

C-More Micro Software Logistics – Creating the Timer Flasher Control Screen (2)

Timer Screen

- Timer Enable HMI switch will be created from Object -> Button -> Switch

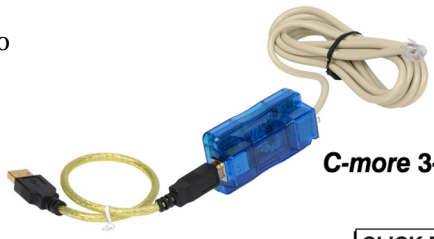


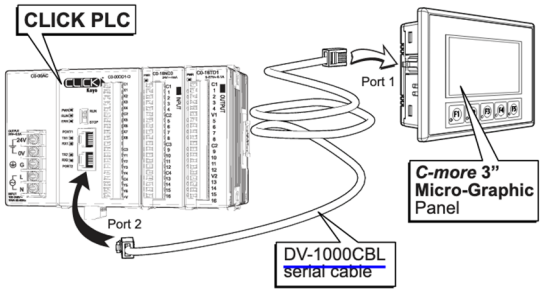
You will also see a timer enable switch to allow the timer flashing circuit to run in the Click Plus PLC. Switch objects can be created from the Object menu under Button, and then select Switch.

This switch is mapped to the TIMEREN tag, which is then mapped to the Click Plus PLC tag C3.

Connecting the PC to the C-More

- USB-to-Serial Converter
  - Used to transfer C-more projects to the C-more HMI
- Press F1 and F5 simultaneously to bring up the setup screen





C-more 3-inch Micro Port 1 to CLICK PLC Port 2

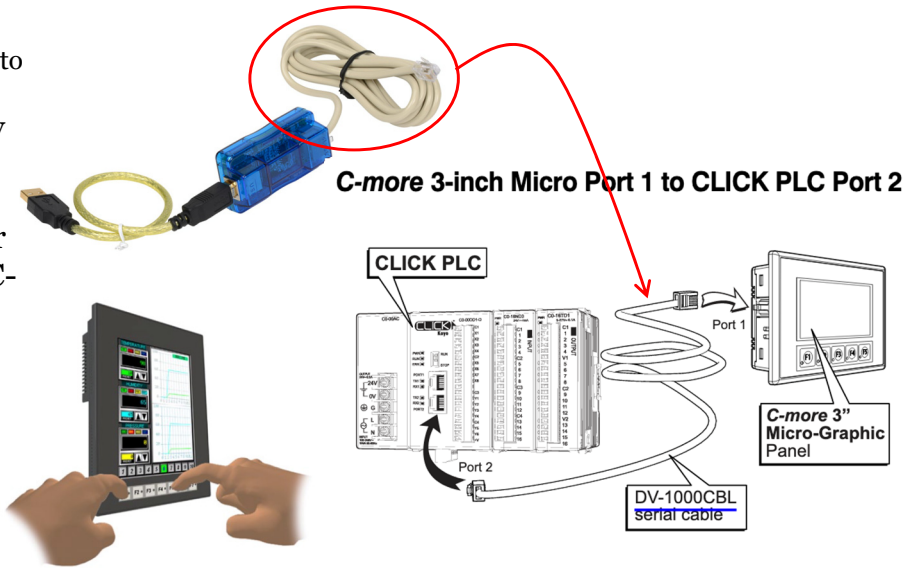
YOU MUST BE ON THE SETUP SCREEN TO DOWNLOAD A PROGRAM!!!

In order to download the C-more project to the C-more panel, you will use the USB-to-Serial converter. The C-more Micro panel must be on the setup screen; in order to display the setup screen, you will push the F1 and F5 function keys simultaneously.

Connecting the PC to the C-More and the C-More to the Click Plus

- USB-to-Serial Converter
  - Used to transfer C-more projects to the C-more HMI
- Press F1 and F5 simultaneously to bring up the setup screen
- The serial cable is then used for communications between the C-more HMI and the Click Plus PLC
  - You will configure the Click Plus PLC port 2 as serial Modbus

YOU MUST BE ON THE SETUP SCREEN TO DOWNLOAD A PROGRAM!!!



After a successful download of a C-more project to the C-more panel, you will remove the serial cable from the USB converter and plug this end into Port 2 of the Click Plus PLC as shown.

Lab 1.5 Overview

- 30 minutes to complete the HMI lab
- The Flasher Timer Control screen is partially completed, and your task will be to fully complete this screen. You will do this by copying objects and remapping the tags
- The Menu Button screen is created but you will map the functionality of the buttons
- You will download a completed binary temperature display screen and verify your binary temperature Click Plus PLC code
- You will verify the Useful Box alarm screen is working in all alarm states

The objective of this lab is to familiarize you with the C-more programming software by opening up an existing project and making modifications to existing objects. You will also create new objects and map the function key buttons in order to change the screens.

Another objective of this lab is to familiarize you with the USB-to-Serial converter hardware and how you will use the serial cable for establishing communications between the C-more Micro HMI and the Click Plus PLC.

Lab 1.5: Local HMI Setup and Control

Go to the Lab Workbook: Lab 1.5

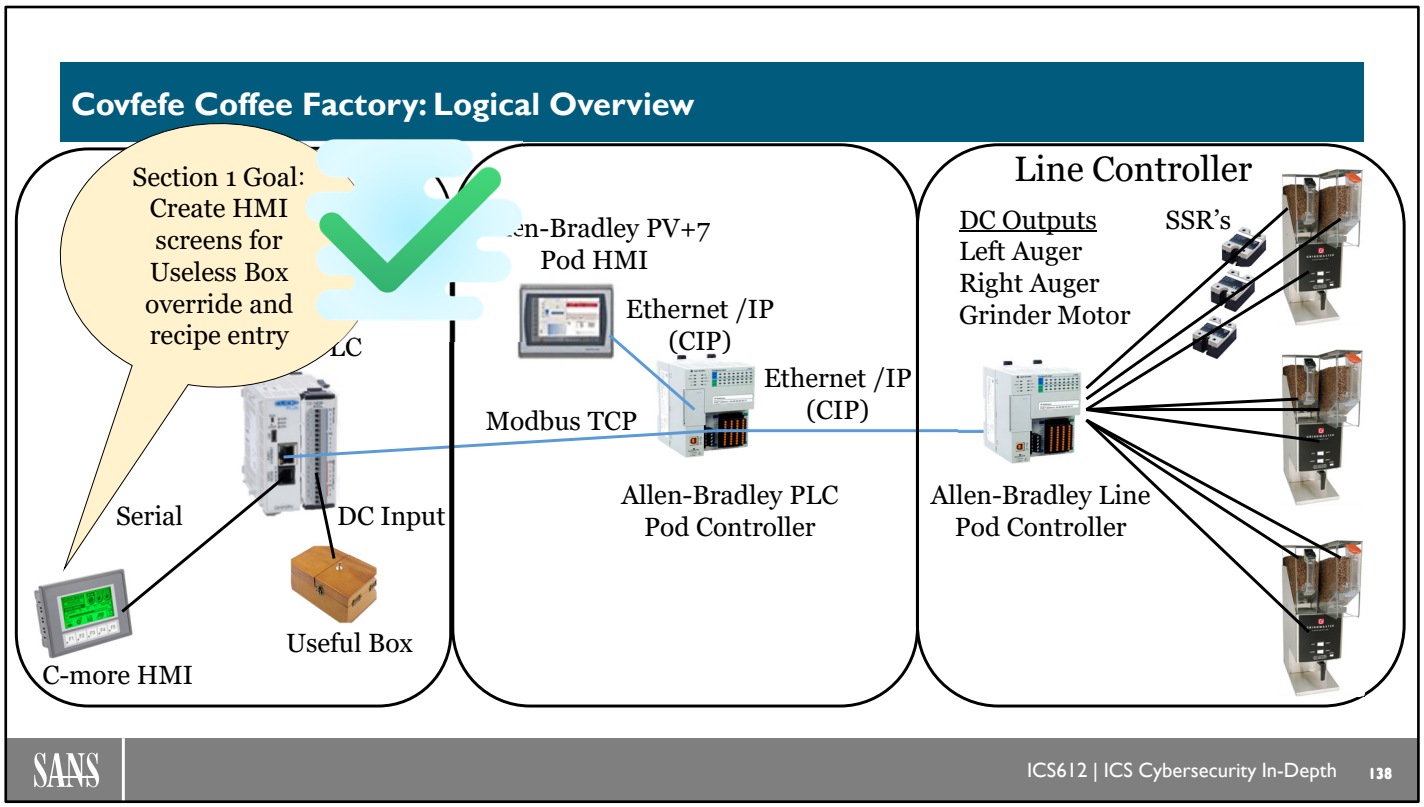
This page intentionally left blank.

Process Interface Checkpoint 1.5

Processes under control will typically need a well-designed operator display that allows a human to visually see the state of the process and easily interact with it and control it. Designing and developing Human Machine Interfaces in a manner that helps operators make decisions is a bit of an art form.

- Developed an HMI Overview screen and the Flasher Timer Control, and mapped the function keys on the Menu Button screen
- Downloaded the HMI program to the C-more HMI panel through the USB-to-Serial converter and established communications from the C-more HMI and the Click Plus PLC via Serial Modbus
- Mapped the temperature binary indicators on the HMI screen to the Click Plus PLC code and verified it worked properly.
- Verified the Useful Box local control inhibit alarm and alarm state are working properly.

This page intentionally left blank.



As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. In the middle of the slide, you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the left contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.

ICS612 Section 1 Outline (5)

- Process Environment Familiarization
- Lab 1.1: Virtual Machine(s) Setup
- Lab 1.2: Student Kit Familiarization
- Programmable Logic Controller Programming
- Lab 1.3: PLC Programming and I/O Integration
- Process Interface
- Lab 1.4: Integrating Analog Input
- Lab 1.5: Local HMI Setup and Control
- Student Pod Integration
- Lab 1.6: Configure the Shared Pod Elements
- Lab 1.7: Connect Student Kits to the Shared Pod
- Lab 1.8: Process Interrupt through Student Kit
- Lab 1.9: Local Process Environment Mapping

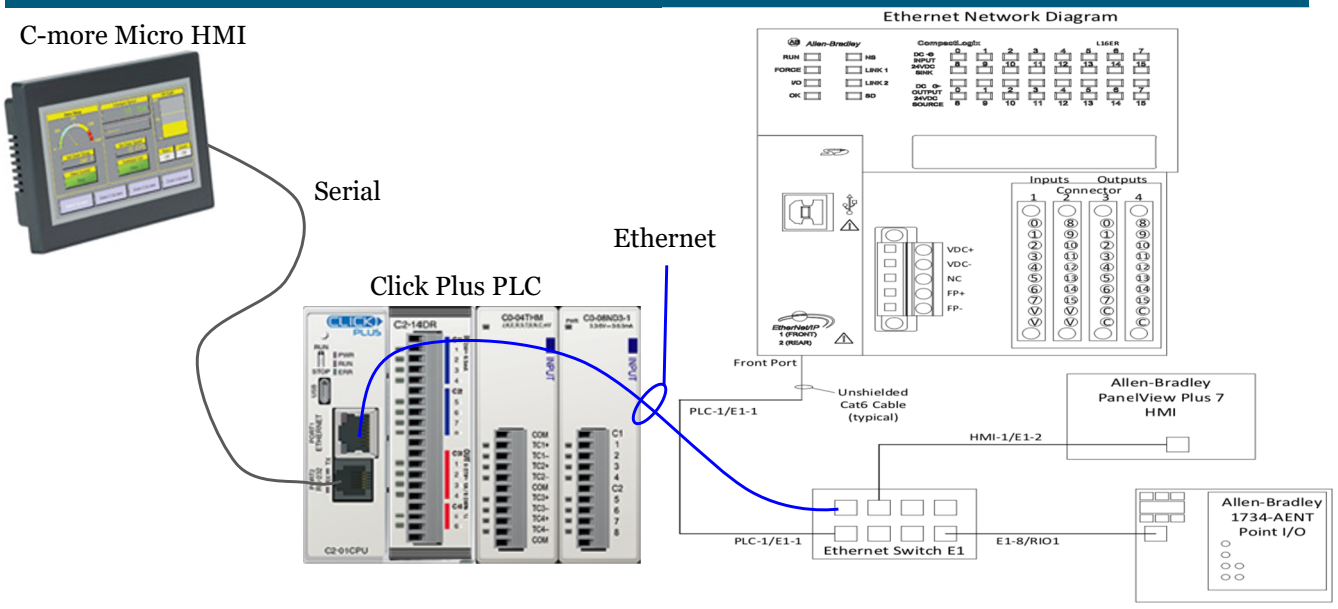
This page intentionally left blank.

Student Pod Integration

Programming the Controller
Connecting the Student Kits to the Shared Pod
Process Integration
Environment Discovery and Mapping

This page intentionally left blank.

Connectivity of Click Plus PLC to Allen-Bradley Pod PLC



We have been familiarized with the wiring and operation of the Click Plus PLC and the C-more Micro HMI panel through hands-on labs. We are now going to connect the Click Plus PLC through Ethernet to the Pod Stratix 5700 Managed Switch.

The Pod contains the following hardware items of interest:

- Allen-Bradley CompactLogix PLC
- Allen-Bradley PanelView Plus 7 HMI
- Allen-Bradley Stratix 5700 Managed Ethernet Switch
- Allen-Bradley 1734 Point I/O Adapter

RSLinx Classic – Ethernet to Common Industrial Protocol (CIP) (1)

RSLinx® Classic

RSLinx Classic is the most widely-installed Communications software in automation today. All editions of RSLinx Classic deliver the ability to browse your automation networks, configure and diagnose network devices. RSLinx Classic supports a wide range of applications through a scalable offerings that provide both DDE and Classic OPC DA data servers to permit 3rd party software to access information within your control system.

FACTORYTALK LINX GATEWAY GETTING RESULTS GUIDE

Find all the information on installing configuration and using the FactoryTalk Linx Gateway.

Read More

FACTORYTALK LINX GETTING RESULTS GUIDE

Find all the information on installing configuration and using the FactoryTalk Linx.

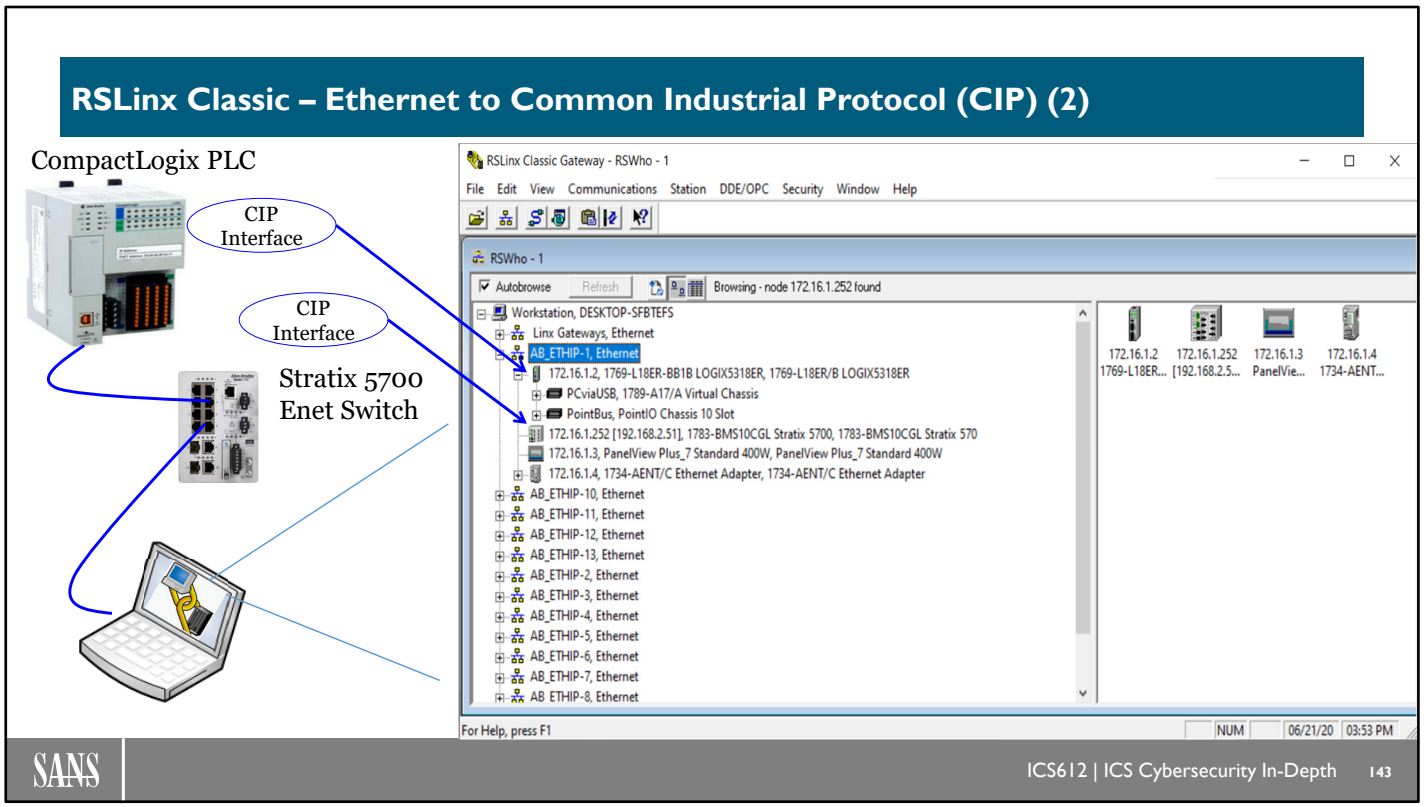
Learn More

RSLINX CLASSIC GETTING RESULTS GUIDE

Find All of the information for Installing, configuring and using RSLinx Classic

Learn More

In order to communicate from your computer to the CompactLogix PLC, you will use a communication gateway package from Rockwell Automation called RSLinx Classic.



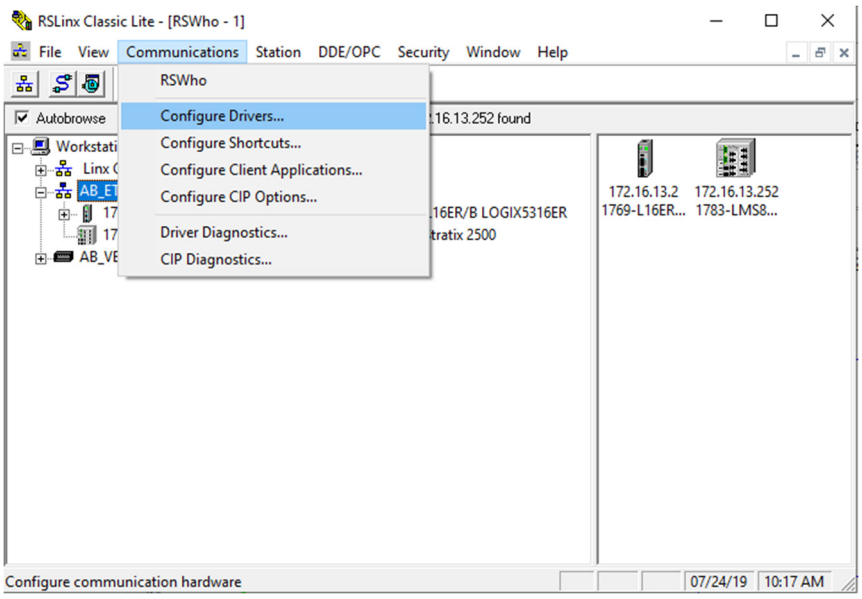
RSLinx Classic allows you to configure communications from your Ethernet adapter to interrogate Ethernet devices that support the Common Industrial Protocol (CIP), amongst other ICS protocols. In our particular use case, we will use RSLinx Classic to interface to the CompactLogix PLC for uploading, downloading, importing, and exporting PLC Logic.

If you look at the screen capture above on the right, you will see there are two devices shown under the A-B_ETHIP-1, Ethernet driver and they are:

- 1769-L18ER-BB1B LOGIX5318ER (CompactLogix PLC)
- 1783-BMS10CGL Stratix 5700 Managed Ethernet Switch

Both of these devices support CIP communications and therefore are capable of responding to the RSLinx Classic query.

RSLinx – Setting up Your IP Address and Confirming the Driver Configuration (1)

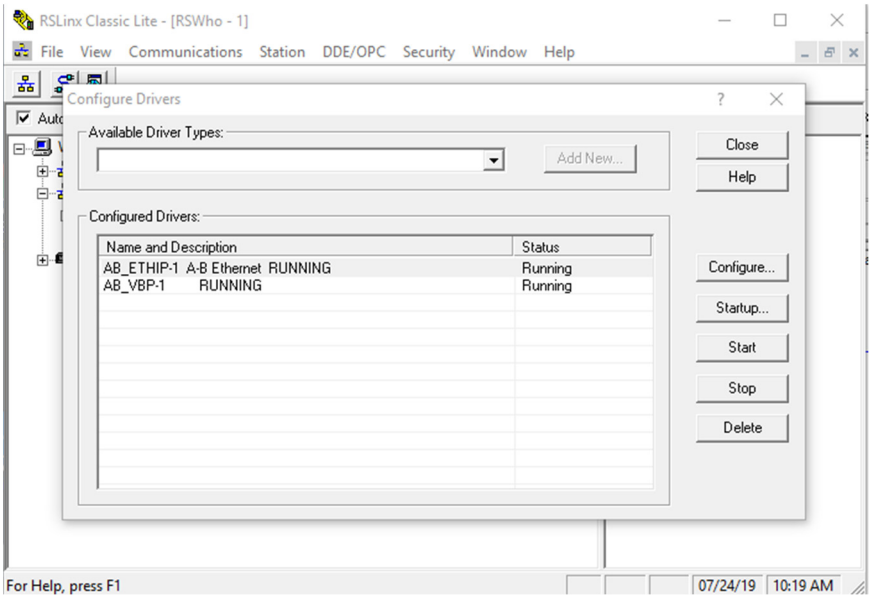


In the next lab, you will configure your RSLinx Classic communication driver with specific IP Addresses based on your Pod assignment.

If you look under the communications menu bar, the first selection you will find is RSWho. RSWho is a term used in this context for polling the network to look for automation devices that support the CIP protocol. We call this “RSWho-ing”.

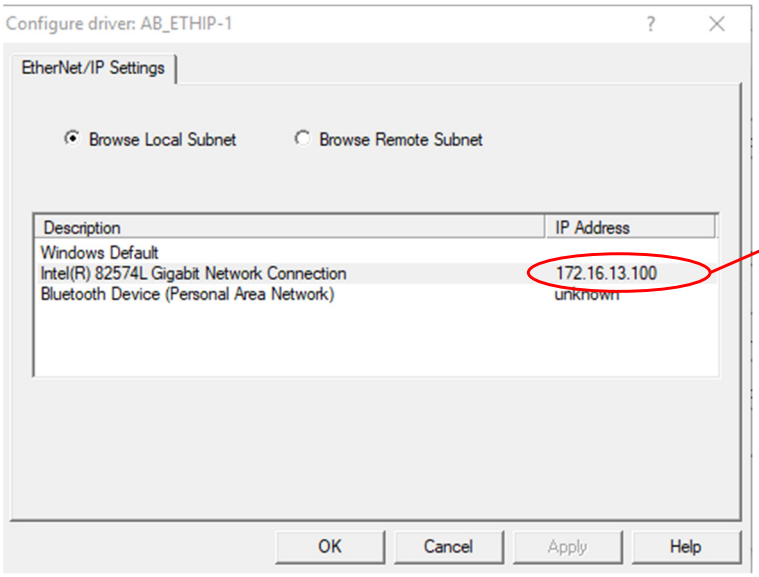
Underneath the RSWho menu item you will find “Configure Drivers...”. You will use this menu selection to configure your RSLinx Classic CIP driver.

RSLinx – Setting up Your IP Address and Confirming the Driver Configuration (2)



Under the configuration of A-B_ETHIP-1, you will have the opportunity to select your Ethernet adapter and save this configuration.

RSLinx – Setting up Your IP Address and Confirming the Driver Configuration (3)

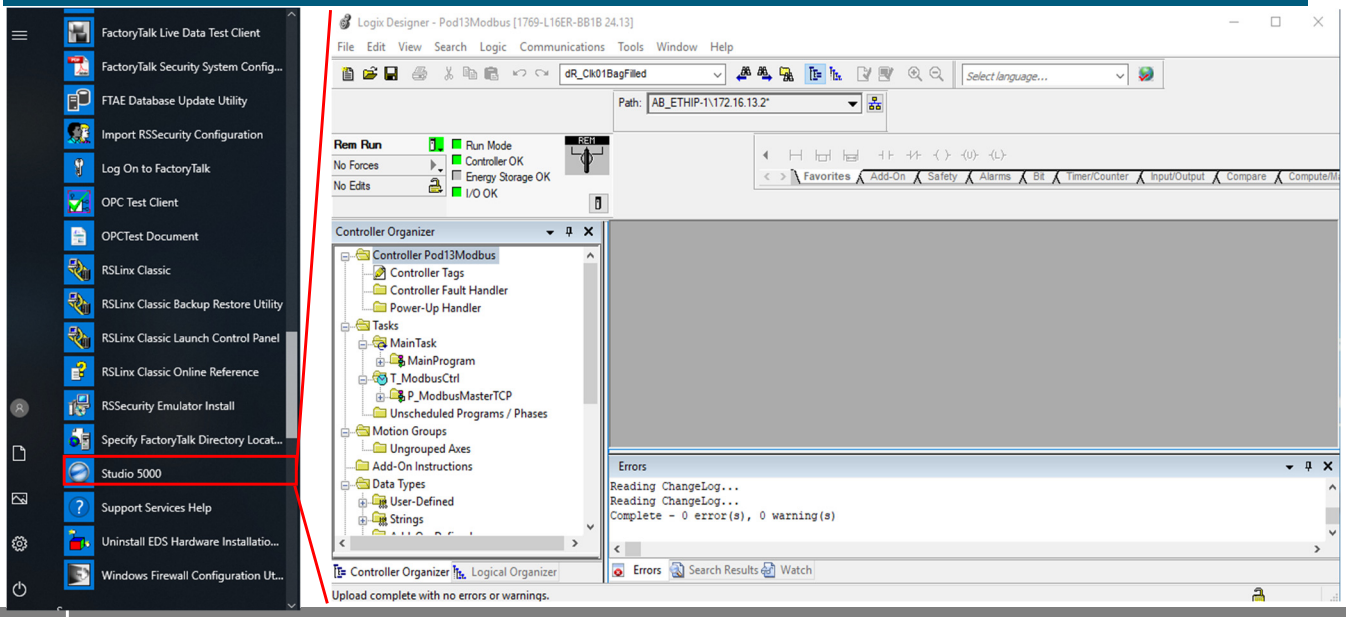


- You will plug your Ethernet cable into the Pod Stratix 5700 switch.
- You will set this driver to the Pod network address

You will be assigned an IP Address with which to set your Ethernet adapter. RSLinx Classic will recognize your local Ethernet adapter and you will configure the driver to use this Ethernet card.

RSLinx Classic is the communication foundation upon which all the software relies to communicate with the Allen-Bradley PLC within this classroom. You will do this exercise in the lab to make sure RSLinx Classic successfully identifies the CompactLogix PLC and the Stratix 5700 switch.

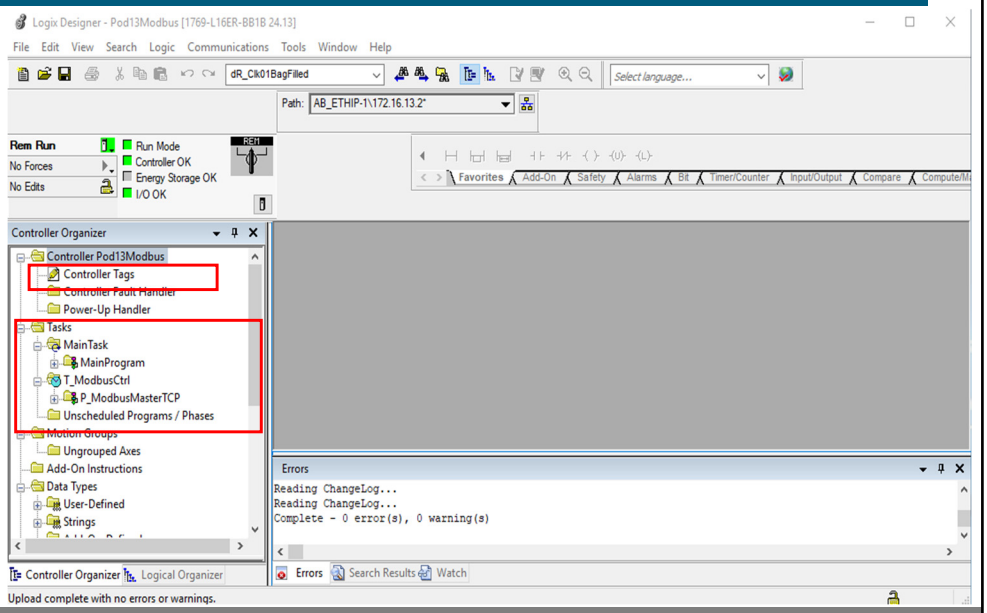
Studio 5000 – Programming and Compiler Software for Allen-Bradley CompactLogix



Studio 5000 software is used to program, download, upload, import, and export programs to the CompactLogix and other Allen-Bradley PLCs. You will launch Studio 5000 in the next lab, and become familiar with the programming environment, changing the tool from Offline to Online mode, uploading the active program, and importing a program while the PLC is still running.

Studio 5000 – Controller Organizer for Tags, Tasks, and Programs

- Tags are variable declarations and physical device definitions
- Tasks are containers for programs and tags
  - Tasks can be scheduled as periodic (recommended) or continuous



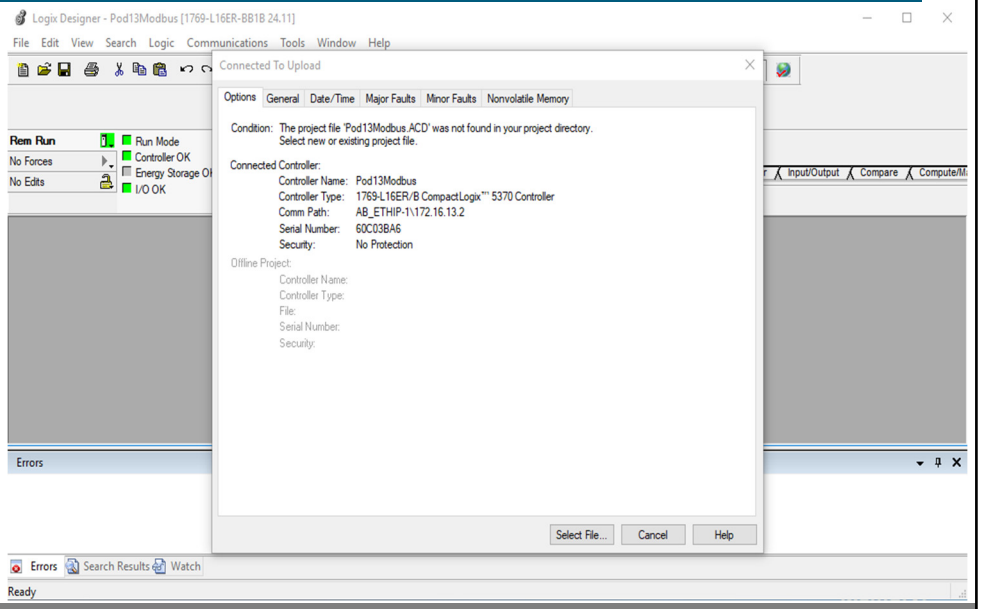
As with the Click Plus PLC, the CompactLogix PLC uses tags for variable declarations and for physical device definitions like Input Cards, Output Cards, Ethernet Cards, etc.
Studio 5000 also supports the subroutine paradigm and some other expansive features like:

- User-defined data types
- User-defined instruction capabilities
- Motion control profiling
- Online program merges through import functionality
- Adding I/O while the processor is still running
- Redundant controller architectures
- Support for all the IEC 61131-3 languages

The CompactLogix platform also allows the user to use continuous tasks or periodic task scheduling. While this can be a long dry conversation, let's just say we recommend using periodic tasks so we know in a predicative manner how often the logic will get scanned within a very precise and specific time interval. This is especially important if you are running closed loop algorithms like Proportional, Integral, and Derivative (PID) instructions.

Studio 5000 – Go Online and Upload the Program

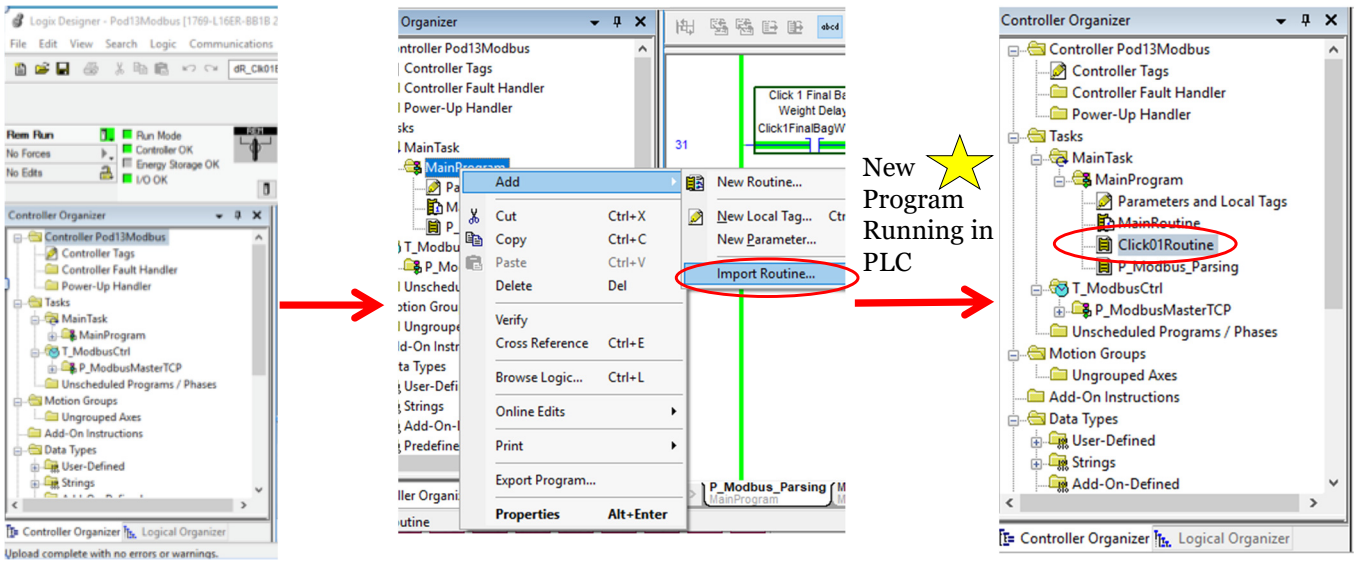
- You will "Go Online" with the PLC
- You will "Upload" the program in the PLC
  - Uploading will allow you to save the logic, tags, rung comments, user-defined data types, etc.



We will use the Pod PLCs in this class as the "Master" PLC copy because we will have multiple students changing the Pod PLC logic. Studio 5000, and more specifically the CompactLogix controller, is capable of storing all the project artifacts like logic, tags, current tag values, rung comments, tag comments, etc. This is an important note; not all PLCs will store all the artifacts within the PLC so be aware that connecting to a PLC and assuming it will have all the project documentation is not a given.

In this next lab, you will upload the project from the PLC.

Import and Merge a New Program with the Running Pod PLC



Once the project is uploaded from the CompactLogix PLC, you will save the program locally. You will then go Online with the PLC and import a program that is specific to your area and Pod assignment.

The Logix family of Allen-Bradley controllers have the ability to continue to control the process while adding new I/O and programs. This feature is important to continuous process control customers so they can continue to run their process while making modifications to the PLC without interrupting the running operations.

Pod PanelView HMI

- The Pod PanelView will be used to display Area and Line status
- You will have an opportunity to exit the running application and load a .mer (compiled HMI file) from an onboard flash drive
- You will interact with the Allen-Bradley CompactLogix and Click Plus PLCs

The Allen-Bradley PanelView Plus 7 is used for the Pod HMI and is used to communicate with the CompactLogix. The PanelView also has Kepware drivers loaded on it, which give it the capability to communicate Modbus and other protocols directly.

You will interact with the PanelView (PV) by loading HMI configurations through a removable flash drive that is inserted in the PanelView.

Lab 1.6: Configure the Shared Pod Elements

Go to the Lab Workbook: Lab 1.6

This page intentionally left blank.

Student Pod Integration Checkpoint 1.6

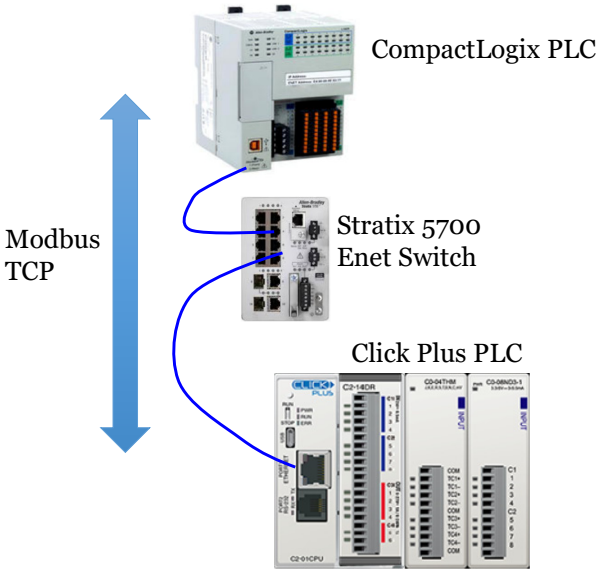
Most ICS devices require proprietary software, cables, configuration files, and practitioner knowledge to perform tasks. Gaining exposure to different devices provides exposure and experience to a whole new set of tools and experiences.

- Configured the RSLinx Classic communication drivers so you could "Go Online" with the Pod CompactLogix PLC
- Uploaded the running program from the Pod CompactLogix PLC and saved it on your computer
- Imported and merged a CompactLogix PLC program into the running Pod PLC
- Stopped the running PanelView application and loaded a new project

This page intentionally left blank.

Connectivity through Modbus

- The Pod CompactLogix PLC is communicating with the Click Plus PLC via Modbus TCP
- The Pod CompactLogix PLC is configured to Read and Write Inputs, Outputs, and Data Registers
- Both the Click Plus and the CompactLogix PLCs will move data in and out of these preconfigured registers to transmit commands, responses, and handshake data



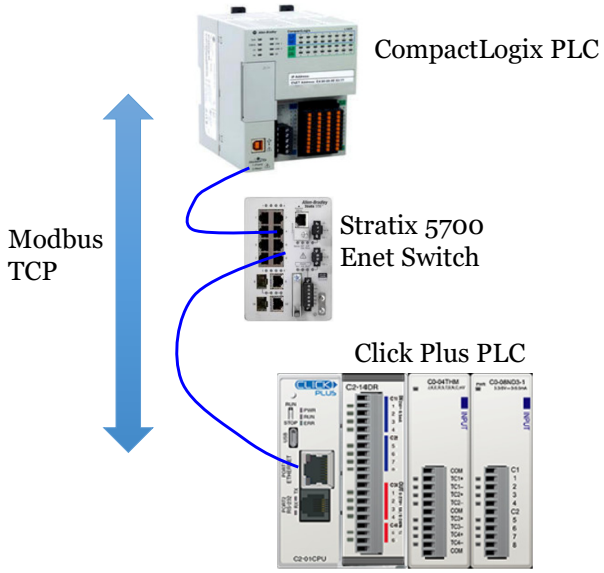
The CompactLogix PLC is capable of communicating to Modbus TCP devices through a capability called Message Instructions. The CompactLogix has been loaded with a Modbus Master Ladder Logic program that contains configurable parameters to allow it to talk with Modbus slave devices like the Click Plus PLC. This logic allows the engineer to configure data table parameters like:

- IP Address of the Modbus Slave
- Modbus Read Coils, Write Coils, Read Register, and Write Register commands
- Communicate with multiple Modbus TCP enabled devices

While you won't go through the details of the Ladder Logic code, you will interact with the data table registers that communicate to your Click Plus PLC.

PLC Modbus Registers: The Lab Will Detail Your Area's Exact Mapping

Click Plus PLC Register	Description	A-B PLC Raw Array Register	A-B PLC Buffered Register
DS10	Heartbeat Echo	MBTU_MB_4xx[0]	dR_Clk01HeartbeatEcho
DS11	Product Type x Request	MBTU_MB_4xx[1]	dR_Clk01ProdType01Req
DS12	Product Weight x Request	MBTU_MB_4xx[2]	dR_Clk01Prod01WeightReq
DS13	Tote Request	MBTU_MB_4xx[3]	dR_Clk01ToteRequest
DS14	Fill Request	MBTU_MB_4xx[4]	dR_Clk01FillBagRequest
DS15	Bag Filled	MBTU_MB_4xx[5]	dR_Clk01BagFilled
DS16	Actual Bag Weight	MBTU_MB_4xx[6]	dR_Clk01BagActlWeight
DS17	Filled Weight	MBTU_MB_4xx[7]	dR_Clik01FilledWeight
DS18	Bar Code	MBTU_MB_4xx[8]	dR_Clk01ProdBarCode01
DS19		MBTU_MB_4xx[9]	
DS20	Heartbeat From CLX	MBTU_MB_4xx[10]	dW_Clk01Heartbeat
DS21	Product Type x Response	MBTU_MB_4xx[11]	dW_Clk01ProdType01Res
DS22	Product Weight x Response	MBTU_MB_4xx[12]	dW_Clk01Prod01WeightRes
DS23	Tote Request Response	MBTU_MB_4xx[13]	dW_Clk01ToteReqRes
DS24	Fill Request Response	MBTU_MB_4xx[14]	dW_Clk01FillResponse
DS25	Bag Filled Response	MBTU_MB_4xx[15]	dW_Clk01BagFilled
DS26	Actual Bag Weight	MBTU_MB_4xx[16]	dW_Clk01BagActWeight
DS27	Filled Weight	MBTU_MB_4xx[17]	dW_Clik01FilledWeight
DS28	Bar Code	MBTU_MB_4xx[18]	dW_Clk01ProdBarCode01



The information on this slide is showing a general tag mapping schema. The left column shows the "DS" register numbers that the CompactLogix PLC is writing to and reading from. The Click Plus's DS registers DS10-DS18 are being written to by the Click Plus PLC and being read from the CompactLogix PLC.

The MBTU_MB_4xx[0] – MBTU_MB_4xx[18] represent the Modbus message queue inside the CompactLogix PLC. This area of memory represents the active message registers.

In the last column, we see the A-B CompactLogix PLC buffered registers. The concept is to move data into the buffered registers and then commit the changes to the active Modbus message registers as represented by MBTU_MB_4xx[0] – MBTU_MB_4xx[28].

In your Lab Workbook you will receive specific register mapping information. The above example may not be relevant for your Pod so consult the Lab Workbook.

Lab 1.7: Connect Student Kits to the Shared Pod

Go to the Lab Workbook: Lab 1.7

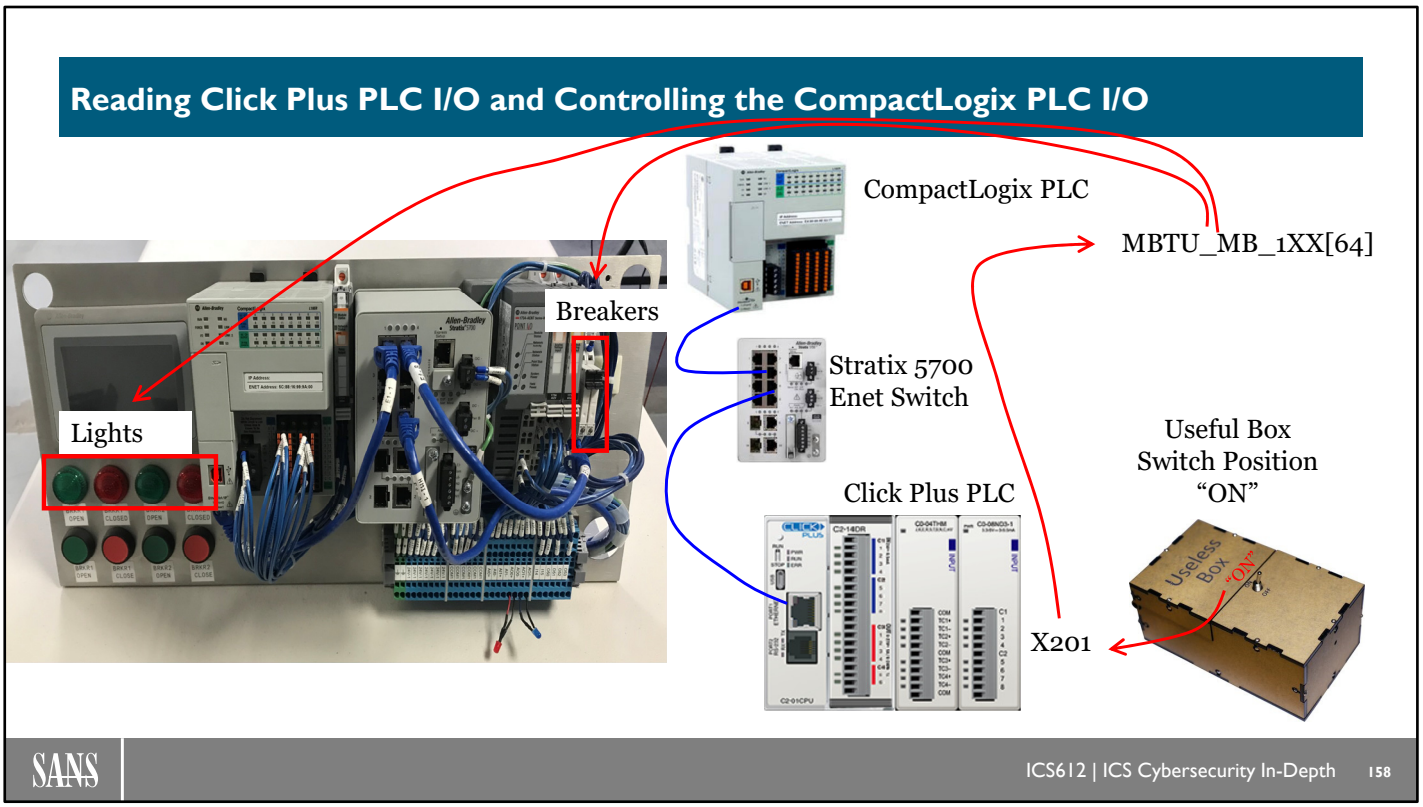
This page intentionally left blank.

Student Pod Integration Checkpoint 1.7

Interoperating across various vendor devices and interconnecting ICS devices across multiple protocols is a very common reality for operational environments; it is rare that a process is entirely one vendor implementation.

- Verified heartbeat registers in order to verify communications
- Entered coffee plant requests via the Click Plus PLC and C-more Micro HMI and verified the Pod CompactLogix line PLC is fulfilling the Click Plus PLC request to fill, mix, grind, or package the coffee order
- Developed an understanding of the register layout behind the CompactLogix Modbus Master logic. This was performed by changing the Modbus request data table and issuing a "rebuild" command.

This page intentionally left blank.



We have highlighted the Modbus communication capabilities between the Click Plus and the CompactLogix PLC. The Useful Box switch input status is contained within the Click Plus PLC data tables and therefore can be transmitted to the CompactLogix PLC. In this next lab, you will use the Useful Box switch to trigger the Pod breaker(s) to change state and perhaps map a status light to the Useful Box switch.

Standardized communications like Modbus TCP and CIP are great enablers for interconnecting and sharing data to interested consumers. With this amount of connectivity, however, it becomes challenging to ensure the environment remains secure, and by secure in this context I mean the environment operates as intended.

Lab 1.8: Process Interrupt through Student Kit

Go to the Lab Workbook: Lab 1.8

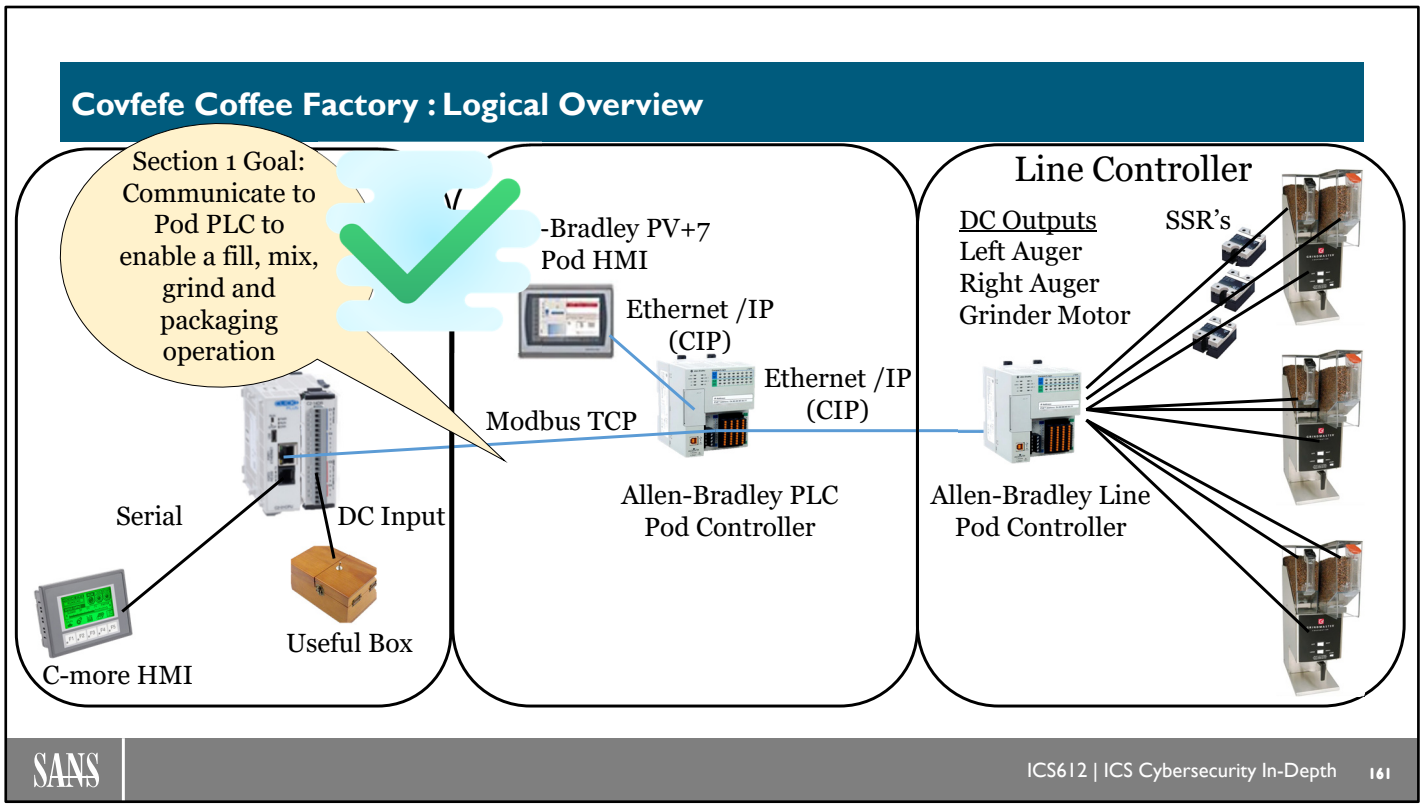
This page intentionally left blank.

Student Pod Integration Checkpoint 1.8

As any element of a process environment is added, changed, or removed, it is required to verify all associated downstream, upstream, and peer relationships in logic, tags, displays, status, or controls that exist.

- Verified that moving the Ubox switch to the "On" position interrupted an action on the shared student Pod process
- Verified the same interruption was achievable remotely through the operator display C-more panel causing a disruption of the filling, mixing, grinding, or packaging process on the Pod CompactLogix PLC.

This page intentionally left blank.



As we look at Levels 0 and 1 hands-on exercises, we will use two systems to achieve the PLC and HMI learning objectives. In the middle of the slide, you will see the training Pod hardware that consists of an Allen-Bradley PanelView HMI and the Allen-Bradley (A-B) CompactLogix PLC. The training Pod also contains push buttons, indicator lights and remote breakers that the A-B PLC will use for input and output control.

The student kit as shown on the left contains the Click Plus PLC and the C-more HMI that will be used during student labs. The Click Plus PLC will communicate with the A-B PLC via Modbus TCP sharing data register information and I/O status. The student kit also contains a Useless Box that will be transformed into a Useful Box that will be controlled by the Click Plus PLC in order to show you how "useful" a simple input switch, motor circuit, and power source can be to gain knowledge about PLC systems. The student kit also contains a K-type thermocouple to demonstrate analog input capabilities of the Click Plus PLC.

Mapping the Environment

- In this section's final exercise, we want you to Nmap the environment to see device IDs, OS fingerprint info, active listening ports, and what those ports are used for vs. registered for on IANA.
- Understanding your inventory and assets is essential to supporting and defending them.
- In section 4, we will show some differences between the Click Plus PLC proprietary device ID, the A-B proprietary ferret tool, and an asset inventory capability.

Using the Kali VM, you will use Nmap to become familiar with your environment. In this short exercise, we want you to identify all the devices, the open ports, and what services might be available on these devices.

Lab 1.9: Local Process Environment Mapping

Go to the Lab Workbook: Lab 1.9

This page intentionally left blank.

Student Pod Integration Checkpoint 1.9

Know your environment! All cybersecurity controls programs, frameworks, standards, and defense strategies start with the need to know the hardware and software inventories of the devices in use. This has traditionally been a challenge for OT environments. While we will cover this topic more throughout the week, it is important to understand how these devices look on a network and how they communicate.

- Verified the network connectivity by mapping the ICS assets
- Created an initial data flow diagram artifact

This page intentionally left blank.

Section 1 Summary (1)

- We have learned about PLC and HMI operations through hands-on labs
- Specifically, we have programmed the Click Plus PLC to control the Useful Box.
- We configured a local control inhibit capability in the Useful Box to override motor operations
- We created and verified a Useful Box alarm when the control inhibit was engaged so we can annunciate the condition
- We integrated our Click Plus PLC into the Pod CompactLogix controller

This page intentionally left blank.

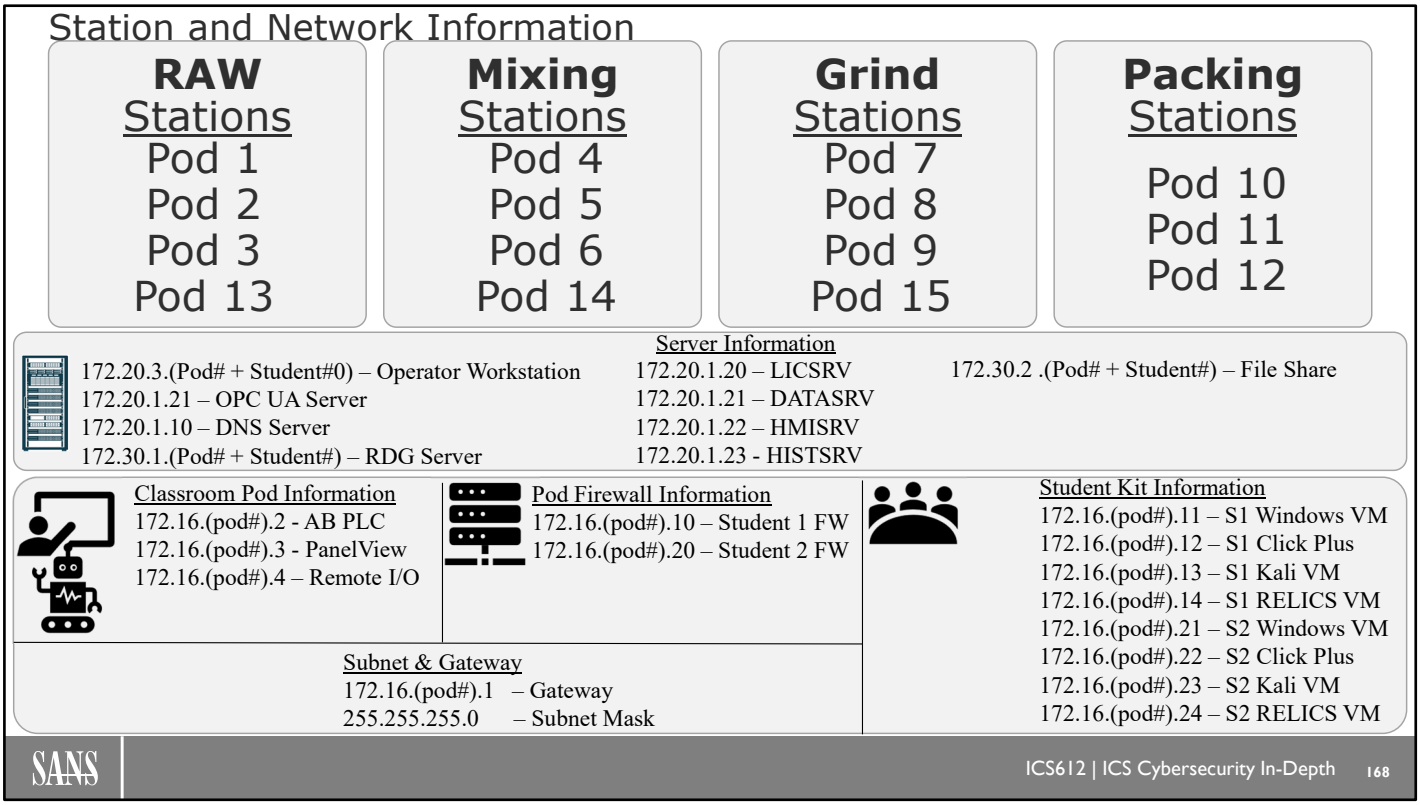
Section 1 Summary (2)

- We saw how Modbus communications can be used to communicate I/O and data register values between different vendor PLCs
- We interrupted the coffee-making process at Layer 1 in the Purdue Model by directly communicating with the data table

This page intentionally left blank.

Section 1 Eval Link

This page intentionally left blank.



This page intentionally left blank.