## Modular Electronics Learning (ModEL) PROJECT



## INTRODUCTION TO PLCs

© 2019-2025 by Tony R. Kuphaldt – under the terms and conditions of the Creative Commons Attribution 4.0 International Public License

Last update = 29 May 2025

This is a copyrighted work, but licensed under the Creative Commons Attribution 4.0 International Public License. A copy of this license is found in the last Appendix of this document. Alternatively, you may visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA. The terms and conditions of this license allow for free copying, distribution, and/or modification of all licensed works by the general public.

ii

# Contents

1	Intr	roduction					
	1.1	Recommendations for students					
	1.2	Challenging concepts related to programmable logic controllers (PLCs)					
	1.3	Recommendations for instructors					
2	Cas	Case Tutorial					
	2.1	Example: NAND function in a PLC 1					
	2.2	Example: simple PLC comparisons					
	2.3	Example: high-pressure PLC-based alarm 1					
	2.4	Example: PLC output statuses from process switch stimuli					
	2.5	Example: PLC process switch statuses from contact status coloring					
	2.6	Example: Arduino versus PLC dual-LED control					
	2.7	Example: Arduino versus PLC logic functions					
	2.8	Example: Arduino versus PLC on-delay timer					
3	Tut	Tutorial 3					
	3.1	The purpose of PLCs					
	3.2	Simple relay control of a cooling system					
	3.3	2003 relay control of a cooling system					
	3.4	2003 PLC control of a cooling system					
	3.5	PLC flexibility					
	3.6	Human-Machine PLC interfaces 4					
4	Der	vivations and Technical References 4					
	4.1	Feature comparisons between PLC models					
		4.1.1 Viewing live values					
		4.1.2 Forcing live values					
		4.1.3 Special "system" values 4					
		4.1.4 Free-running clock pulses					
		4.1.5 Standard counter instructions					
		4.1.6 High-speed counter instructions					
		4.1.7 Timer instructions					
		4.1.8 ASCII text message instructions					
		4.1.9 Analog signal scaling					

#### CONTENTS

	4.2	Legacy	Allen-Bradley memory maps and I/O addressing
	4.3	Koyo	"drum" sequencer instructions
	4.4	Allen-	Bradley sequencer instructions
	4.5	SELog	ic control equations
		4.5.1	Basic logical functions
		4.5.2	Set-reset latch instructions
		4.5.3	One-shot instructions
		4.5.4	Counter instructions
		4.5.5	Timer instructions 71
5	Que	estions	73
	5.1	Conce	ptual reasoning
		5.1.1	Reading outline and reflections
		5.1.2	Foundational concepts
		5.1.3	Relay ladder logic analogy for a PLC
		5.1.4	Sourcing versus sinking PLC I/O
		5.1.5	Sketching wires to PLC discrete I/O
		5.1.6	Two different motor control programs
		5.1.7	Redundant coils in a program
		5.1.8	Determining bit statuses from switch conditions
		5.1.9	Determining necessary switch conditions for bit statuses
		5.1.10	Determining color highlighting from bit statuses
		5.1.11	Determining color highlighting from switch conditions
		5.1.12	Determining bit statuses from color highlighting
		5.1.13	Determining process switch stimuli from color highlighting
		5.1.14	Determining necessary switch contact types
		5.1.15	Motor start-up limit counter
		5.1.16	Allen-Bradley counter program
		5.1.17	Room occupancy counter
		5.1.18	PLC timing diagrams
		5.1.19	Sequenced-start conveyor belts
		5.1.20	Switch contact types for a timed conveyor control
		5.1.21	Air compressor control program
	5.2	Quant	itative reasoning
		5.2.1	Miscellaneous physical constants
		5.2.2	Introduction to spreadsheets
		5.2.3	Frequency divider program
		5.2.4	Integer format error between PLC and HMI
	5.3	Diagno	ostic reasoning
	0.0	5.3.1	Incorrect PLC output wiring 126
		5.3.2	Troubleshooting motor control program
		5.3.3	Troubleshooting motor control PLC from I/O indicators
		5.3.4	Turbine low-oil trip
		5.3.5	Motor starter diagnosis from color highlighting
		5.3.6	Cannery counter diagnosis
		5.3.7	Parking garage counter faults
		0.0.1	r annuel Parale connect mane

iv

### CONTENTS

	5.3.8 Possible faults in a PLC/HMI pump control system	139				
	5.3.9 Possible faults in a PLC/HMI package-counting system	141				
	5.3.10 Diagnostic tests on a failed PLC/HMI pump control system	142				
	5.3.11 Correcting PLC program errors	143				
6	Projects and Experiments	151				
	6.1 Recommended practices	151				
	6.1.1 Safety first!	152				
	6.1.2 Other helpful tips	154				
	6.1.3 Terminal blocks for circuit construction	155				
	6.1.4 Conducting experiments	158				
	6.1.5 Constructing projects	162				
	6.2 Experiment: contact and coil demonstration program	163				
	6.3 Experiment: PLC implementation of basic logic functions	165				
	6.4 Experiment: PLC implementation of an arbitrary truth table	166				
	6.5 Experiment: PLC-controlled motor starter	168				
	6.6 Experiment: counter demonstration program	170				
	6.7 Experiment: timer demonstration program	172				
	6.8 Experiment: PLC-controlled inrush-limiting motor starter	174				
	6.9 Experiment: HMI display of PLC bits and words	176				
	6.10 Experiment: arithmetic demonstration program	177				
	6.11 Experiment: data transfer demonstration program	179				
	6.12 Project: PLC-controlled system	181				
A	Problem-Solving Strategies	183				
в	Instructional philosophy	185				
С	Tools used					
D	D Creative Commons License					
$\mathbf{E}$	E Version history					
In	ndex					

1

CONTENTS

## Chapter 1

# Introduction

### **1.1** Recommendations for students

A programmable logic controller or PLC is a specialized form of industrial computer, designed to be programmed by the end user for many different types of discrete and continuous process control applications. The word "programmable" in its name reveals just why PLCs are so useful: the enduser is able to program, or instruct, the PLC to execute virtually any control function imaginable.

PLCs were introduced to industry as electronic replacements for electromechanical relay controls. In applications where relays typically control the starting and stopping of electric motors and other discrete output devices, the reliability of an electronic PLC meant fewer system failures and longer operating life. The re-programmability of a PLC also meant changes could be implemented to the control system strategy must easier than with relay circuits, where re-wiring was the only way to alter the system's function. Additionally, the computer-based nature of a PLC meant that process control data could now be communicated by the PLC over *networks*, allowing process conditions to be monitored in distant locations, and by multiple operator stations.

The legacy of PLCs as relay-replacements is probably most evident in their traditional programming language: a graphical convention known as a *Ladder Diagram*. Ladder Diagram PLC programs resemble ladder-style electrical schematics, where vertical power "rails" convey control power to a set of parallel "rung" circuits containing switch contacts and relay coils. A human being programming a PLC literally draws the diagram on the screen, using relay-contact symbols to represent instructions to read data bits in the PLC's memory, and relay-coil symbols to represent instructions writing data bits to the PLC's memory. When executing, these graphical elements become colored when "conductive" to virtual electricity, thereby indicating their status to any human observer of the program. This style of programming was developed to make it easier for industrial electricians to adapt to the new technology of PLCs. While Ladder Diagram programming definitely has limitations compared to other computer programming languages, it is relatively easy to learn and diagnose, which is why it remains popular as a PLC programming language today.

While ladder diagram programming was designed to be simple by virtue of its resemblance to relay ladder-logic schematic diagrams, this very same resemblance often creates problems for students encountering it for the very first time. A very common misconception is to think that the contact and coil symbols shown on the editing screen of the PLC programming software are somehow *identical* or at least *directly representative* of real-world contacts and coils wired to the PLC. This is not true. Contacts and coils shown on the screen of PLC programming software applications are *instructions* for the PLC to follow, and their logical states depend both on how they are drawn in the program *and* upon their related bit states in the PLC's memory.

The relationship between a discrete sensor (e.g. switch) and the colored state of a ladder diagram element inside of a PLC follows a step-by-step chain of causation:

- 1. Physical closure of the discrete switch causes electricity to flow through the switch's contact.
- 2. This electrical current flows through the PLC input terminal wired to that switch.
- 3. This energization causes a corresponding bit in the PLC's memory to become "high" (1).
- 4. Any Ladder Diagram "contact" instruction associated with that bit will become "actuated". If the contact instruction is normally-open, the "1" bit state will "close" the contact instruction and cause it to be colored. If the contact instruction is normally-closed, the "1" bit state will "open" is and cause it to be un-colored.
- 5. If all elements in a rung of the Ladder Diagram program are colored, the final instruction (at the far right end of the rung) will become activated and will cause it to fulfill its function.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to test whether a switch is normally-open or normally-closed?
- How might an experiment be designed and conducted to test the "scan time" of a PLC program?
- What are some practical applications of PLCs?
- What is the "normal" status of a switch?
- How does a "two-out-of-three" alarm or shutdown system function?
- What is a "nuisance trip"?
- Are there applications where a hard-wired relay control system might actually be better than a system using a PLC?
- What purpose is served by the color highlighting feature of PLC program editing software?
- What is an HMI panel, and where would one be useful?
- How might you alter one of the example analyses shown in the text, and then determine the behavior of that altered circuit?
- Devise your own question based on the text, suitable for posing to someone encountering this subject for the first time

# 1.2 Challenging concepts related to programmable logic controllers (PLCs)

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- Normal status of a switch to say that a switch is "normally open" or "normally closed" refers to its electrical state *when at rest*, and not necessarily the state you will typically find the switch in. The root of the confusion here is the word *normal*, which most people take to mean "typical" or "ordinary". In the case of switches, though, it refers to the zero-stimulation status of the switch as it has been manufactured. In the case of PLCs it refers to the status of a "virtual switch" when its controlling bit is 0.
- Seal-in contacts the simplest and most common way to make momentary-contact "Start" and "Stop" switches latch a motor's state on and off is to wire an auxiliary contact from the contactor in parallel with the Start switch to "seal in" that circuit once the contactor energizes. The same logical structure may be implemented in PLC ladder-diagram programming by placement of a virtual contact in parallel with the initiating contact, that additional contact being controlled by the coil of that same rung. This, however, complicates analysis of the circuit by granting it *states*. State-based logic is more complex than combinational logic because the status of state-based logic depends not only on input conditions but also on past history. This means a person must analyze the PLC program before and after input condition changes to determine how it will respond.
- Sourcing versus Sinking output currents a common misconception is that since PLC outputs are called "outputs" it must mean that current only ever *exits* those terminals. This is untrue. All that "output" actually signifies is the fact that the PLC is outputting *information* consisting of voltage values measured between that terminal and ground. Sometimes the assertion of an "on" state requires that the PLC output channel actually draws current *in* through its "output" terminal!

### **1.3** Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

• **Outcome** – Demonstrate effective technical reading and writing

<u>Assessment</u> – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

<u>Assessment</u> – Interpret elements of a real PLC ladder-logic program such as those shown in the Simplified and Full Tutorial chapters, including identification of virtual coils, virtual contacts (normally-open and normally-closed), color-highlighting, and associated statuses of those elements based on color highlighting.

• **Outcome** – Apply the concept of "normal" contact status to real and virtual contacts in PLC systems

<u>Assessment</u> – Predict the status of every PLC program virtual contact and PLC program coil in a ladder-style diagram for various input switch state combinations. Good starting points include the Case Tutorial section "Example: NAND function in a PLC" as well as the two-out-of-three water flow alarm system shown in the Simplified Tutorial chapter.

• **Outcome** – Properly associate physical switch states with their associated PLC bit states, PLC virtual contact states, PLC virtual coil states, and PLC output states in a PLC-controlled system.

<u>Assessment</u> – Determine PLC bit states based on given physical switch stimuli in a schematic diagram; e.g. pose problems in the form of the "Determining bit statuses from switch conditions" Conceptual Reasoning question.

<u>Assessment</u> – Determine physical switch stimuli based on given PLC bit states in a schematic diagram; e.g. pose problems in the form of the "Determining necessary switch conditions for bit statuses" Conceptual Reasoning question.

<u>Assessment</u> – Determine PLC ladder-diagram program element coloring based on given physical switch stimuli in a schematic diagram; e.g. pose problems in the form of the "Determining color highlighting from switch conditions" Conceptual Reasoning question.

<u>Assessment</u> – Determine PLC bit states based on color-highlighting shown in a live view of a PLC's ladder-diagram program; e.g. pose problems in the form of the "Determining bit statuses from color highlighting".

<u>Assessment</u> – Determine physical switch stimuli based on color-highlighting shown in a live view of a PLC's ladder-diagram program; e.g. pose problems in the form of the "Determining process switch stimuli from color highlighting" Conceptual Reasoning question.

#### 1.3. RECOMMENDATIONS FOR INSTRUCTORS

• Outcome – Sketch wire connections necessary to interface a PLC to a controlled process

<u>Assessment</u> – Sketch wire placement in a pictorial diagram showing how switches would connect to the input channels of a PLC and how loads would connect to the output channels of the same PLC; e.g. pose problems in the form of the "Sketching wires to PLC discrete I/O" Conceptual Reasoning question.

#### • Outcome – Diagnose a fault within a PLC-controlled system

<u>Assessment</u> – Identify possible faults to account for a system's improper function based on an examination of the color highlighting in a live view of the PLC's ladder-diagram program; e.g. pose problems in the form of the "Troubleshooting motor control program" and "Motor starter diagnosis from color highlighting" Diagnostic Reasoning questions.

<u>Assessment</u> – Identify possible faults to account for a system's improper function based on an examination of the I/O status indicators on the front of the PLC; e.g. pose problems in the form of the "Troubleshooting motor control PLC from I/O indicators" Diagnostic Reasoning question.

#### • Outcome – Independent research

<u>Assessment</u> – Locate PLC I/O module datasheets and properly interpret some of the information contained in those documents including number of I/O channels, voltage and current limitations, sourcing versus sinking capability, etc.

# Chapter 2

# **Case Tutorial**

The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module - can you explain *why* the circuits behave as they do?

## 2.1 Example: NAND function in a PLC









## 2.2 Example: simple PLC comparisons

The following illustration shows wiring and a sample relay ladder logic (RLL) program for an Allen-Bradley MicroLogix 1000 PLC:



Note how Allen-Bradley I/O is labeled in the program: input bits designated by the letter I and output bits designated by the letter O.

In order to energize the LED, the switch connected to input terminal 0 must be off (open) and the switch connected to input terminal 1 must be on (closed).

#### 2.2. EXAMPLE: SIMPLE PLC COMPARISONS

The following illustration shows wiring and a sample relay ladder logic (RLL) program for a Siemens Simatic S7-200 PLC:



Note how Siemens I/O is labeled in the program: input bits designated by the letter I and output bits designated by the letter Q.

In order to energize the LED, either the switch connected to input terminal 0.0 must be on (closed) or the switch connected to input terminal 0.1 must be off (open).

The following illustration shows wiring and a sample relay ladder logic (RLL) program for a Koyo CLICK PLC:



Note how Koyo I/O is labeled in the program: input bits designated by the letter X and output bits designated by the letter Y.

In order to energize the LED, at least one of the following conditions must be met:

- X1 switch turned on (closed) and X2 switch turned off (open)
- X2 switch turned on (closed) and X1 switch turned off (open)

Either the switch connected to input terminal 0.0 must be on (closed) or the switch connected to input terminal 0.1 must be off (open).

## 2.3 Example: high-pressure PLC-based alarm

Step #1 – fluid pressure inside the process vessel is low, and the alarm lamp is de-energized:



**Step #2** – the pressure switch detects a high fluid pressure condition, triggering the PLC to energize the alarm lamp:



**Step #3** – fluid pressure decreases below the pressure switch's trip point, but the PLC maintains power to the alarm lamp to signify that a high-pressure event happened:



120 VAC "line" power





**Step #5** – the lamp remains de-energized, awaiting another high-pressure event before energizing again:



120 VAC "line" power

# 2.4 Example: PLC output statuses from process switch stimuli

Suppose we have a Siemens S7-200 PLC connected to a pair of process switches and lamps as shown in this illustration:



We happen to know that the fluid pressure is 30 PSI and the fluid level is 4 feet. The PLC's program is as follows:



With pressure being *below* the switch's trip point, we know that pressure switch will be in its "normal" (resting) state which happens to be *closed*. This closed status energizes input I1.2, setting that bit inside the PLC's memory to a value of one (1). This 1 bit status actuates the normally-open I:1.2 contact instruction, causing it to close (i.e. become colored).

With level being *above* the switch's trip point, we know that level switch will be in its "actuated" (non-resting) state which happens to be *closed*. This closed status energizes input I0.7, setting that bit inside the PLC's memory to a value of one (1). This 1 bit status actuates the normally-open I:0.7 contact instruction, causing it to close (i.e. become colored).

With both of those PLC contact instructions in the top rung being colored, color makes it to

coil Q0.3, setting that bit to a 1 state. Discrete output Q0.3 becomes energized, causing the Green lamp to energize as well. The same 1 bit status of Q0.3 prompts the normally-closed Q0.3 contact instruction to open (i.e. become un-colored), thereby preventing color from reaching coil Q0.1. This de-energizes output Q0.1 and makes the Red lamp de-energized.

# 2.5 Example: PLC process switch statuses from contact status coloring

The following illustration shows a Koyo "CLICK" PLC connected to three process switches and one lamp, with the live status of each PLC instruction shown as well:



Based on the displayed color highlighting of the PLC program's virtual contacts, we may conclude the following:

- The X1 contact instruction is colored which means it is closed. It is also drawn as normallyclosed, and for a normally-closed PLC contact instruction to be closed the bit controlling it must be a zero (0). This means the X1 input must be de-energized, which in turn means the pressure switch contact must be open. Since the pressure switch is normally-closed, in order to be in the open state we must have a high pressure condition.
- The X2 contact instruction is colored which means it is closed. It is drawn as normally-open, and for a normally-open PLC contact instruction to be closed the bit controlling it must be a one (1). This means the X2 input must be energized, which in turn means the flow switch contact must be closed. Since the flow switch is normally-open, in order to be in the closed state we must have a high flow condition.
- The X3 contact instruction is un-colored which means it is open. It is drawn as normally-closed, and for a normally-closed PLC contact instruction to be open the bit controlling it must be a one (1). This means the X3 input must be energized, which in turn means the temperature switch contact must be closed. Since the temperature switch is normally-open, in order to be in the closed state we must have a **high temperature** condition.
- No color reaches the Y1 coil instruction which means its bit state will be a zero (0). This makes the Y1 output de-energized, ensuring the lamp will be de-energized as well.

#### 2.5. EXAMPLE: PLC PROCESS SWITCH STATUSES FROM CONTACT STATUS COLORING23

The following illustration shows a Siemens S7-200 PLC connected to two process switches and two lamps, with the live status of each PLC instruction shown as well:



Based on the displayed color highlighting of the PLC program's virtual contacts, we may conclude the following:

- The I0.2 contact instruction is un-colored which means it is open. It is also drawn as normallyopen, and for a normally-open PLC contact instruction to be open the bit controlling it must be a zero (0). This means the I0.2 input must be de-energized, which in turn means the flow switch contact must be open. Since the flow switch is normally-closed, in order to be in the open state we must have a **high flow** condition.
- The I1.0 contact instruction is colored which means it is closed. It is also drawn as normallyclosed, and for a normally-closed PLC contact instruction to be closed the bit controlling it must be a zero (0). This means the I1.0 input must be de-energized, which in turn means the temperature switch contact must be open. Since the temperature switch is normally-open, in order to be in the open state we must have a **low temperature** condition.
- No color reaches the Q0.3 coil instruction which means its bit state will be a zero (0). This makes the Q0.3 output de-energized, ensuring the Green lamp will be de-energized as well.
- Color does reach the Q0.1 coil instruction which means its bit state will be a one (1). This makes the Q0.1 output energized, ensuring the Red lamp will be energized as well.

Q 000000 Powe 🗖 Run Fault ŗ I:0 I:0 I:0 0:0 VAC VDC VDC Ø Ø  $\oslash$ Ò  $\oslash$ Ø Ø Ø  $\oslash \oslash \oslash$ 0

The following illustration shows an Allen-Bradley MicroLogix PLC connected to three process switches and a lamp, with the live status of each PLC instruction shown as well:

Based on the displayed color highlighting of the PLC program's virtual contacts, we may conclude the following:

- The I:0/0 contact instruction is un-colored which means it is open. It is drawn as normallyclosed, and for a normally-closed PLC contact instruction to be open the bit controlling it must be a one (1). This means the I:0/0 input must be energized, which in turn means the level switch contact must be closed. Since the level switch is normally-closed, in order to be in the closed state we must have a **low level** condition.
- The I:0/2 contact instruction is colored which means it is closed. It is drawn as normally-open, and for a normally-open PLC contact instruction to be closed the bit controlling it must be a one (1). This means the I:0/2 input must be energized, which in turn means the temperature switch contact must be closed. Since the temperature switch is normally-open, in order to be in the closed state we must have a **high temperature** condition.
- The I:0/3 contact instruction is colored which means it is closed. It is also drawn as normallyclosed, and for a normally-closed PLC contact instruction to be closed the bit controlling it must be a zero (0). This means the I:0/3 input must be de-energized, which in turn means the flow switch contact must be open. Since the flow switch is normally-open, in order to be in the open state we must have a **low level** condition.
- No color reaches the 0:0/1 coil instruction which means its bit state will be a zero (0). This makes the 0:0/1 output de-energized, ensuring the lamp will be de-energized as well.

### 2.6 Example: Arduino versus PLC dual-LED control

An *Arduino* is a popular model of microcontroller designed for student and hobbyist use, where text-based code stored in the microcontroller's memory dictates how it will respond to input signals to control devices connected to its outputs.

Here we see a partial wiring diagram and Sketch-language code causing an Arduino Nano microcontroller to control the states of two light-emitting diodes (LEDs) when a pushbutton switch is pressed and released:



Pressing the pushbutton switch energizes pin D5, and that electrically-energized state is read as a "1" value and stored in the *button* variable. This same "1" value gets written to pin D13 to energize that LED, while pin D12 gets the opposite value ("0") written to it to de-energize its LED.

Releasing the pushbutton switch de-energizes pin D5, causing a "0" value to be stored in the *button* variable. This same "0" value gets written to pin D13 to de-energize that LED, while pin D12 gets the opposite value ("1") written to it to energize its LED.

A *Programmable Logic Controller* or *PLC* is a special-purpose industrial computer where code stored in the PLC's memory dictates how it will respond to input signals to control devices connected to its outputs. The most common programming language for PLCs is *ladder diagram* which resembles a certain type of electrical wiring diagram, designed that way to make it familiar to electricians.

Here we see a partial wiring diagram and ladder-diagram code causing a PLC to control the states of two light-emitting diodes (LEDs) when a pushbutton switch is pressed and released:



Pressing the pushbutton switch energizes input X1, and that electrically-energized state is read as a "1" value and stored in the PLC's X1 variable. This "1" value causes the normally-open virtual switch contact to close and activate output Y1 to turn on that LED, and also causes the normally-closed virtual switch contact to open and de-activate output Y4 to turn off that LED.

Releasing the pushbutton switch de-energizes input X1 and clears the X1 variable to a "0" value, causing both of the virtual switch contacts to return to their resting states, de-activating output Y1 and re-activating output Y4.

## 2.7 Example: Arduino versus PLC logic functions

An AND function is where multiple conditions all must be "true" in order to generate a response. An OR function is where just one of those multiple conditions need be "true" to make the same response. The following Arduino program implements both of these logical functions, with one LED designated for each and with three pushbutton switches providing the multiple conditions (inputs):



In this program the LED connected to Arduino pin 13 represents the result of an AND function for the three inputs, while the LED connected to pin 12 represents the result of an OR function. Pressing any switch causes pin 12 LED to energize, but all three must be pressed simultaneously to energize pin 13. PLCs also implement AND and OR functions, but do so by connecting their virtual switches either in "series" (all in a row) or in "parallel", respectively:



In this program the LED connected to PLC output Y1 represents the output of an AND function for the three inputs, while the LED connected to output Y4 represents the output of an OR function. Pressing any switch causes the Y4 LED to energize, but all three must be pressed simultaneously to energize Y1.

### 2.8 Example: Arduino versus PLC on-delay timer

An "on-delay" timer is one where you electrically activate the timer, but instead of turning on an LED or other device immediately, it delays for a set time before turning it on. Here we see a partial wiring diagram and Sketch-language code causing an Arduino Nano microcontroller to delay the turn-on of a light-emitting diode (LED) when a pushbutton switch is pressed:



Here we see a partial wiring diagram and ladder-diagram code causing a PLC to implement an on-delay timing function:



## Chapter 3

# Tutorial

### 3.1 The purpose of PLCs

To understand what a *Programmable Logic Controller* or *PLC* does, it is helpful to study industrial control technology pre-dating PLCs. Like many technologies, PLCs were invented as a solution to a problem, that problem being *how to easily configure electrical control systems for industrial machines and processes.* An example of a PLC being used to control a portion of a municipal wastewater treatment facility is shown below:



The three grey-colored PLCs shown above each contain a power supply module, a processor module, and multiple input/output ("I/O") modules where a multitude of signal wires connect. Each one is a rugged computer designed to run all day for decades on end, faithfully controlling the starting and stopping of pumps, conveyors, mixers, electrically-operated water valves, and alarm units. At its essence, a PLC implements fundamental logic functions such as AND, OR, and NOT, and are configured for any particular application by a *program* containing instructions on how to behave.

Prior to the advent of PLCs, most industrial control circuits used hard-wired switches and electromechanical relays to implement the necessary AND, OR, and NOT logical functions. The PLC was invented as a means to implement the same logical functionality but in such a way that the circuitry did not have to be re-wired whenever a change was necessary. Being a programmable digital device like any digital computer, a PLC could be re-configured for a different control logic scheme simply by altering bit-states in its digital memory.

### 3.2 Simple relay control of a cooling system

Let's begin with a practical example. Suppose we have a water cooling system for a large and expensive machine such as an industrial engine, and need an alarm system to monitor the flow of cooling water because a loss of cooling water would mean destruction for this machine. Coolant is so critically important that three different pumps provide cooling water through separate pipes to this same machine, the redundancy of pumps and pipes ensuring a greater level of cooling reliability. However, if water flow to this machine ceases for any reason, we want our electrical alarm system to activate a solenoid valve that will shut down the machine, as well as an indicator light to alert people what happened.

If we install three flow-sensing switches with normally-closed<sup>1</sup> contacts, one switch per water pipe, and connect them in series with each other to feed electrical power to the solenoid valve and to the alarm light, we will have a practical alarm-and-shutdown system for this expensive machine. The necessary connections between these components are shown in the following "ladder diagram" which is a common form of documentation for electrical control systems:



All three switches will need to close in order to energize the shutdown solenoid and the alarm lamp. This means all three water pipes' flows would have to cease in order to have all three of these flow switches return to their resting ("normal") states. Thus, this system will not take action unless and until cooling water flow stops through all three pipes. If we happen to lose cooling water flow through any one or any two pipes at a time, the machine will still be allowed to operate and the alarm lamp will not energize.

Such a system seems perfectly appropriate if the flow of water through just one pipe will be enough to sufficiently cool the machine, since the only real emergency would be a loss of cooling water flow for *all* three pipes.

<sup>&</sup>lt;sup>1</sup>Recall that the "normal" status of an electrical switch is its electrical status in a condition of no physical stimulus. That is, a "normally-closed" (NC) switch will exhibit closed contacts when it is at rest. For a water flow switch, "rest" means a condition of no water flow. Since we want our flow switches to conduct electricity to the shutdown solenoid when water flow stops, we need switches that close with no water flow – i.e. normally-closed flow switch contacts. These NC contacts will be held in their "open" states by the presence of adequate water flow, which means under regular operating conditions they will actually be open. This is what is so confusing about the "normal" status of switches: a switch's "normal" status may or may not happen to match its *typical* status when everything is operating as it should!
However, suppose one of these three flow switches happens to *fail* in an open state, remaining open even when water flow stops. If water flow through all three pipes happens to cease while the switch is faulted like this, the system will neither shut down the machine nor turn on the alarm lamp as intended! By designing the circuit such that all three switches must indicate low flow before initiating a shutdown of the machine, we have left that machine vulnerable to any of the water flow switches failing open.

One solution to this problem is to re-configure the circuit so that the three switch contacts are wired in parallel with each other rather than in series. This way, the machine will shut down and the alarm lamp will energize if *any one or more* of the switches indicate low cooling water flow:



No longer is the machine vulnerable to a single water flow switch failing open – now, *all three switches* would have to fail open to leave the machine unprotected in the event of a total cooling water outage. However, the trade-off with the parallel-wired design is that now the circuit will shut down the machine and activate the alarm lamp even if just a single water pipe's flow is too low, even if there is still sufficient water flow through the other two to cool it. In other words, this newly-wired system errs on the side of paranoia, and therefore is prone to *nuisance trip events* where the machine shuts down unnecessarily.

A compromise solution to this new problem is to design the circuit in such a way that it takes *two* out of three water flow switches agreeing to either let the machine run or to shut it down. This way, a single failed-open flow switch will not jeopardize the machine, and neither will a loss of cooling water through just one pipe result in a "nuisance trip". However, it is electrically impossible to wire just three switch contacts to fulfill this two-out-of-three logic function.

# 3.3 2003 relay control of a cooling system

We can implement a two-out-of-three shutdown/alarm system by using three electromechanical relays, letting each flow switch activate one relay coil and then using multiple switches in each relay wired in a series-parallel configuration to form the two-out-of-three logic:



Each of these control relays (CR1, CR2, and CR3) has one coil (symbolized by a circle) and two normally-open contacts (symbolized by parallel vertical line segments), the association between coil and contact(s) indicated by label as is standard in "ladder logic" diagrams. In the configuration shown, the shutdown solenoid and alarm lamp energize if any two or more flow switches close, and conversely the machine will still run if any two or more flow switches remain open. This is the desired two-out-of-three shutdown logic.

Although this is a practical solution, it involves a fair amount of wiring between relay coils and switch contacts, and the addition of those relays constitute more points of failure because electromechanical relays have moving parts and therefore wear out over time. Furthermore, if we wanted to add more features such as making the alarm lamp energize on a one-out-of-three basis while keeping the shutdown solenoid on two-out-of-three logic, we would likely have to install relays containing more switch contacts and of course spend significant time re-wiring everything. The circuit as shown does indeed work, but it offers no flexibility for future changes.

# 3.4 2003 PLC control of a cooling system

A *Programmable Logic Controller* is a solid-state digital computer designed to replicate the functionality of those control relays without the inherent limitations. Each of the three water flow switches energizes its own *input terminal* on the PLC (labeled with X designators). The solenoid coil and alarm lamp connect to individual *output terminals* on the PLC (labeled with Y designators). Inside the PLC is a microprocessor executing a program written in a graphical programming language resembling relay contacts and coils, instructing it as to how and when it should activate each Y output based on the states of the X inputs:



Each "relay contact" inside the PLC program is actually a *read instruction* examining the electrical state of its respective X input (as controlled by each flow switch), and based on that state will either "open" or "close" its portion of the virtual circuit. Each "coil" in the PLC program is a *write instruction* commanding its respective Y output to either turn on or turn off real electrical

power to the external load. In essence, the PLC program acts like a virtual circuit passing or blocking virtual electricity through virtual contacts to virtual coils.

A helpful feature of modern PLCs is *color highlighting* to show the status of these virtual switches and virtual coils as they block or conduct imaginary electricity. For example, in a case where water flow switches A and C are closed but switch B is open, the PLC would respond as shown below:



In this example only the bottom "rung" of this "ladder" program has virtual continuity all the way through to provide virtual electricity to virtual coils Y1 and Y5. On the other two rungs we see both X1 and X3 "contacts" in closed states, but because of the non-colored X2 "contact" in each of those two paths no virtual electricity may "flow" there.

Color highlighting is extremely useful when troubleshooting PLC-controlled systems to determine the internal state of the PLC's program at any time<sup>2</sup>.

 $<sup>^{2}</sup>$ Readers familiar with text-based computer programming languages know this as the *debugging* feature of their

# 3.5 PLC flexibility

Adding capabilities to a PLC-controlled system is as simple as editing its programming:



In this modification we maintained the existing two-out-of-three logic for the shutdown function but changed the alarm function to one-out-of-three. Had this system been controlled by physical relays rather than a PLC, these same changes would have required new wiring, new or different

development environment, where the software will show you bit states, register values, program step, and many other important parameters to show you what state the program is in at any given step of its execution. This is a standard feature for PLC programming due to its diagnostic value.

relays<sup>3</sup>, and of course time to perform all the physical work. Modifying a PLC's program, by contrast, requires only a few minutes to complete.

Now consider another feature change in this system, this time turning the "Alarm" lamp into an "All-Good" lamp – illuminating when all three flow switches detect adequate water flow. As before, all we need to change is the PLC's programming:



Note the use of *normally-closed* virtual contacts in the rung driving coil Y5. Being normallyclosed, these contacts will become colored (i.e. virtually conductive) whenever their associated input is de-energized, and will un-color (i.e. virtually "open") whenever their associated input is energized. Thus, the only way coil Y5 can receive virtual electricity necessary to turn on the "All-Good" lamp is if all three inputs (X1, X2, and X3) are de-energized, which in turn means all three water flow switches must be held in their open states by healthy amounts of water flow.

<sup>&</sup>lt;sup>3</sup>If the original three relays CR1, CR2, and CR2 only offered two normally-open contacts each, we would either have to replace them with triple-pole-contact relays or added three more relays with coils paralleled to provide the new contacts necessary for the new 1003 alarm function.

#### 3.5. PLC FLEXIBILITY

To illustrate, here is the same diagram annotated to show electrical currents, PLC indicator LEDs, and color highlighting of the PLC "contact" and "coil" instructions when all three flow switches are held in their open states due to healthy water flow through all three pipes:





If water flow ceases through pipe A, that flow switch will return to its "normal" closed state, resulting in the following response from the PLC:

#### 3.5. PLC FLEXIBILITY

If water flow ceases through both pipes A and B, those flow switches will return to their "normal" closed states, resulting in the following response from the PLC:



It is highly recommended as an *active reading* exercise to analyze all of these annotated diagrams to fully understand the relationship between real-world switch status (open or closed), PLC input status (energized or not), PLC virtual contact status (colored or not), PLC virtual coil status (colored or not), PLC output status (energized or not), and finally real-world load status (energized or not). A proper understanding of this cause-and-effect chain is the first and most important principle of PLCs, as mastering it is key to understanding all other ladder-diagram PLC programming.

This exploration of how a PLC may be used to replace electromechanical relays in a shutdown/alarm control circuit describes just a small portion of a modern PLC's capabilities. In addition to virtual contacts and virtual coils, PLCs offer virtual timers, virtual counters, digital communication capability, and a host of other features.

A modern trend in PLC technology is the ability to program in languages other than "ladder diagram". The ladder diagram language was invented for the sake of making PLC programming easy to understand for technicians familiar with electrical wiring, but compared to many other programming languages ladder diagram is extremely limited. So, some PLCs may be programmed in text-based languages and/or in function blocks for increased versatility.

# 3.6 Human-Machine PLC interfaces

One of the more impressive features of a PLC is its ability to communicate data over digital networks to other PLCs, other computers, and/or to display screens so that operations personnel can visually perceive the states of digital bits and words within the PLC. The common term to describe visual display screens intended for use with PLCs is *Human-Machine Interface* or *HMI*. HMIs are really nothing more than "hardened" personal computers built ruggedly and in a compact format to facilitate their use in industrial environments. Most industrial HMI panels come equipped with touch-sensitive screens, allowing operators to press their fingertips on displayed objects to change screens, view details on portions of the process, etc. An example illustration of an HMI working in conjunction with a PLC is shown here:



A photograph of a real HMI appears next:



# Chapter 4

# Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

# 4.1 Feature comparisons between PLC models

In most cases, similarities are far greater for different models of PLC than differences. However, differences do exist, and it is worth exploring the differences in basic features offered by an array of PLC models.

#### 4.1.1 Viewing live values

- <u>Allen-Bradley Logix 5000</u>: the *Controller Tags* folder (typically on the left-hand pane of the programming window set) lists all the tag names defined for the PLC project, allowing you to view the real-time status of them all. Discrete inputs do not have specific input channel tag names, as tag names are user-defined in the Logix5000 PLC series.
- <u>Allen-Bradley PLC-5, SLC 500, and MicroLogix</u>: the *Data Files* listing (typically on the lefthand pane of the programming window set) lists all the data files within that PLC's memory. Opening a data file window allows you to view the real-time status of these data points. Discrete inputs are the I file addresses (e.g. I:0/2, I:3/5, etc.). The letter "I" represents "input," the first number represents the slot in which the input card is plugged, and the last number represents the bit within that data element (a 16-bit word) corresponding to the input card.
- <u>Siemens S7-200</u>: the *Status Chart* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory. Discrete inputs are the I memory addresses (e.g. I0.1, I1.5, etc.).
- <u>Koyo (Automation Direct) DirectLogic and CLICK</u>: the *Data View* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory. Discrete inputs are the X memory addresses (e.g. X1, X2, etc.).

#### 4.1.2 Forcing live values

- <u>Allen-Bradley Logix 5000</u>: forces may be applied to specific tag names by right-clicking on the tag (in the program listing) and selecting the "Monitor" option. Discrete outputs do not have specific output channel tag names, as tag names are user-defined in the Logix5000 PLC series.
- <u>Allen-Bradley PLC-5, SLC 500, and MicroLogix</u>: the *Force Files* listing (typically on the lefthand pane of the programming window set) lists those data files within the PLC's memory liable to forcing by the user. Opening a force file window allows you to view and set the real-time status of these bits. Discrete outputs are the 0 file addresses (e.g. 0:0/7, 0:6/2, etc.). The letter "0" represents "output," the first number represents the slot in which the output card is plugged, and the last number represents the bit within that data element (a 16-bit word) corresponding to the output card.
- <u>Siemens S7-200</u>: the *Status Chart* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory, and enabling the user to force the values of those addresses. Discrete outputs are the Q memory addresses (e.g. Q0.4, Q1.2, etc.).
- <u>Koyo (Automation Direct) DirectLogic and CLICK</u>: the *Override View* window allows the user to force variables within the PLC's memory. Discrete outputs are the Y memory addresses (e.g. Y1, Y2, etc.).

#### 4.1.3 Special "system" values

Every PLC has special registers holding data relevant to its operation, such as error flags, processor scan time, etc.

- <u>Allen-Bradley Logix 5000</u>: various "system" values are accessed via GSV (Get System Value) and SSV (Save System Value) instructions.
- <u>Allen-Bradley PLC-5</u>, <u>SLC 500</u>, and <u>MicroLogix</u>: the Data Files listing (typically on the lefthand pane of the programming window set) shows file number 2 as the "Status" file, in which you will find various system-related bits and registers.
- <u>Siemens S7-200</u>: the *Special Memory* registers contain various system-related bits and registers. These are the SM memory addresses (e.g. SM0.1, SMB8, SMW22, etc.).
- <u>Koyo (Automation Direct) DirectLogic and CLICK</u>: the *Special* registers contain various system-related bits and registers. These are the SP memory addresses (e.g. SP1, SP2, SP3, etc.) in the DirectLogic PLC series, and the SC and SD memory addresses in the CLICK PLC series.

#### 4.1.4 Free-running clock pulses

- <u>Allen-Bradley SLC 500</u>: status bit S:4/0 is a free-running clock pulse with a period of 20 milliseconds, which may be used to clock a counter instruction up to 50 to make a 1-second pulse (because 50 times 20 ms = 1000 ms = 1 second).
- <u>Siemens S7-200</u>: Special Memory bit SM0.5 is a free-running clock pulse with a period of 1 second.
- <u>Koyo (Automation Direct) DirectLogic</u>: Special relay SP4 is a free-running clock pulse with a period of 1 second.

#### 4.1.5 Standard counter instructions

- <u>Allen-Bradley Logix 5000</u>: CTU count-up, CTD count-down, and CTUD count-up/down instructions.
- <u>Allen-Bradley SLC 500</u>: CTU and CTD instructions.
- <u>Siemens S7-200</u>: CTU count-up, CTD count-down, and CTUD count-up/down instructions.
- Koyo (Automation Direct) DirectLogic: UDC counter instruction.

#### 4.1.6 High-speed counter instructions

- <u>Allen-Bradley SLC 500</u>: HSU high-speed count-up instruction.
- <u>Siemens S7-200</u>: HSC high-speed counter instruction, used in conjunction with the HDEF high-speed counter definition instruction.

#### 4.1.7 Timer instructions

- <u>Allen-Bradley Logix 5000</u>: TOF off-delay timer, TON on-delay timer, RTO retentive on-delay timer, TOFR off-delay timer with reset, TONR on-delay timer with reset, and RTOR retentive on-delay timer with reset instructions.
- <u>Allen-Bradley SLC 500</u>: TOF off-delay timer, TON on-delay timer, and RTO retentive on-delay timer instructions.
- <u>Siemens S7-200</u>: TOF off-delay timer, TON on-delay timer, and TONR retentive on-delay timer instructions.

#### 48

#### 4.1.8 ASCII text message instructions

- <u>Allen-Bradley Logix 5000</u>: the "ASCII Write" instructions AWT and AWA may be used to do this. The "ASCII Write Append" instruction (AWA) is convenient to use because it may be programmed to automatically insert linefeed and carriage-return commands at the end of a message string.
- <u>Allen-Bradley SLC 500</u>: the "ASCII Write" instructions AWT and AWA may be used to do this. The "ASCII Write Append" instruction (AWA) is convenient to use because it may be programmed to automatically insert linefeed and carriage-return commands at the end of a message string.
- <u>Siemens S7-200</u>: the "Transmit" instruction (XMT) is useful for this task when used in Freeport mode.
- <u>Koyo (Automation Direct) DirectLogic</u>: the "Print Message" instruction (PRINT) is useful for this task.

#### 4.1.9 Analog signal scaling

- <u>Allen-Bradley Logix 5000</u>: the I/O configuration menu (specifically, the *Module Properties* window) allows you to directly and easily scale analog input signal ranges into any arbitrary numerical range desired. Floating-point ("REAL") format is standard, but integer format may be chosen for faster processing of the analog signal.
- <u>Allen-Bradley PLC-5, SLC 500, and MicroLogix</u>: raw analog input values are 16-bit signed integers. The SCL and SCP instructions are custom-made for scaling these raw integer ADC count values into ranges of your choosing.
- <u>Siemens S7-200</u>: raw analog input values are 16-bit signed integers. Interestingly, the S7-200 PLC provides built-in potentiometers assigned to special word registers (SMB28 and SMB29) with an 8-bit (0-255 count) range. These values may be used for any suitable purpose, including combination with the raw analog input register values in order to provide mechanical calibration adjustments for the analog input(s).
- <u>Koyo (Automation Direct) DirectLogic</u>: you must use standard math instructions (e.g. ADD, MUL) to implement a y = mx + b linear equation for scaling purposes.
- <u>Koyo (Automation Direct) CLICK</u>: the I/O configuration menu allows you to directly and easily scale analog input signal ranges into any arbitrary numerical range desired.

# 4.2 Legacy Allen-Bradley memory maps and I/O addressing

A wise PLC programmer once told me that the first thing any aspiring programmer should learn about the PLC they intend to program is how the digital memory of that PLC is organized. This is sage advice for any programmer, especially on systems where memory is limited, and/or where I/O has a fixed association with certain locations in the system's memory. Virtually every microprocessor-based control system comes with a published *memory map* showing the organization of its limited memory: how much is available for certain functions, which addresses are linked to which I/O points, how different locations in memory are to be referenced by the programmer.

Discrete input and output channels on a PLC correspond to individual *bits* in the PLC's memory array. Similarly, analog input and output channels on a PLC correspond to multi-bit *words* (contiguous blocks of bits) in the PLC's memory. The association between I/O points and memory locations is by no means standardized between different PLC manufacturers, or even between different PLC models designed by the same manufacturer. This makes it difficult to write a general tutorial on PLC addressing, and so my ultimate advice is to consult the engineering references for the PLC system you intend to program.

The most common brand of PLC in use in the United States at the time of this writing (2019) is Allen-Bradley (Rockwell), and a great many of these Allen-Bradley PLCs still in service happen to use a unique form of I/O addressing<sup>1</sup> students tend to find confusing.

 $<sup>^{1}</sup>$ The most modern Allen-Bradley PLCs have all but done away with fixed-location I/O addressing, opting instead for *tag name* based I/O addressing. However, enough legacy Allen-Bradley PLC systems still exist in industry to warrant coverage of these addressing conventions.

File number	File type	Logical address range
0	Output image	0:0 to 0:30
1	Input image	I:0 to I:30
2	Status	S:0 to $S:n$
3	Binary	B3:0 to B3:255
4	Timers	T4:0 to T4:255
5	Counters	C5:0 to C5:255
6	Control	R6:0 to R6:255
7	Integer	N7:0 to N7:255
8	Floating-point	F8:0 to F8:255
9	Network	x9:0 to x9:255
10  through  255	User defined	x10:0 to x255:255

The following table shows a partial memory map for an Allen-Bradley SLC 500 PLC<sup>2</sup>:

Note that Allen-Bradley's use of the word "file" differs from personal computer parlance. In the SLC 500 controller, a "file" is a block of random-access memory used to store a particular type of data. By contrast, a "file" in a personal computer is a contiguous collection of data bits with collective meaning (e.g. a word processing file or a spreadsheet file), usually stored on the computer's hard disk drive. Within each of the Allen-Bradley PLC's "files" are multiple "elements," each element consisting of a set of bits (8, 16, 24, or 32) representing data. Elements are addressed by number following the colon after the file designator, and individual bits within each element addressed by a number following a slash mark. For example, the first bit (bit 0) of the second element in file 3 (Binary) would be addressed as B3:2/0.

In Allen-Bradley PLCs such as the SLC 500 and PLC-5 models, files 0, 1, and 2 are exclusively reserved for discrete outputs, discrete inputs, and status bits, respectively. Thus, the letter designators O, I, and S (file types) are redundant to the numbers 0, 1, and 2 (file numbers). Other file types such as B (binary), T (timers), C (counters), and others have their own default file numbers (3, 4, and 5, respectively), but may also be used in some of the user-defined file numbers (10 and above). For example, file 7 in an Allen-Bradley controller is reserved for data of the "integer" type (N), but integer data may also be stored in any file numbered 10 or greater at the user's discretion. Thus, file numbers and file type letters for data types other than output (O), input (I), and status (S) always appear together. You would not typically see an integer word addressed as N:30 (integer word 30 in the PLC's memory) for example, but rather as N7:30 (integer word 30 *in file* 7 of the PLC's memory) to distinguish it from other integer word 30's that may exist in other files of the PLC's memory.

 $<sup>^{2}</sup>$ Also called the *data table*, this map shows the addressing of memory areas reserved for programs entered by the user. Other areas of memory exist within the SLC 500 processor, but these other areas are inaccessible to the technician writing PLC programs.

This file-based addressing notation bears further explanation. When an address appears in a PLC program, special characters are used to separate (or "delimit") different fields from each other. The general scheme for Allen-Bradley SLC 500 PLCs is shown here:



Not all file types need to distinguish individual words and bits. Integer files (N), for example, consist of one 16-bit word for each element. For instance, N7:5 would be the 16-bit integer word number five held in file seven. A discrete input file type (I), though, needs to be addressed as individual bits because each separate I/O point refers to a single bit. Thus, I:3/7 would be bit number seven residing in input element three. The "slash" symbol is necessary when addressing discrete I/O bits because we do not wish to refer to all sixteen bits in a word when we just mean a single input or output point on the PLC. Integer numbers, by contrast, are collections of 16 bits each in the SLC 500 memory map, and so are usually addressed as entire words rather than bit-by-bit<sup>3</sup>.

Certain file types such as timers are more complex. Each timer "element<sup>4</sup>" consists of *two* different 16-bit words (one for the timer's accumulated value, the other for the timer's target value) in addition to no less than *three* bits declaring the status of the timer (an "Enabled" bit, a "Timing" bit, and a "Done" bit). Thus, we must make use of both the decimal-point and slash separator symbols when referring to data within a timer. Suppose we declared a timer in our PLC program with the address T4:2, which would be timer number two contained in timer file four. If we wished to address that timer's current value, we would do so as T4:2.ACC (the "Accumulator" word of timer number two in file four). The "Done" bit of that same timer would be addressed as T4:2/DN (the "Done" bit of timer number two in file four)<sup>5</sup>.

<sup>&</sup>lt;sup>3</sup>This is not to say one *cannot* specify a particular bit in an otherwise whole word. In fact, this is one of the powerful advantages of Allen-Bradley's addressing scheme: it gives you the ability to precisely specify portions of data, even if that data is not generally intended to be portioned into smaller pieces!

<sup>&</sup>lt;sup>4</sup>Programmers familiar with languages such as C and C++ might refer to an Allen-Bradley "element" as a *data* structure, each type with a set configuration of words and/or bits.

 $<sup>{}^{5}</sup>$ Referencing the Allen-Bradley engineering literature, we see that the accumulator word may alternatively be addressed by number rather than by mnemonic, T4:2.2 (word 2 being the accumulator word in the timer data structure), and that the "done" bit may be alternatively addressed as T4:2.0/13 (bit number 13 in word 0 of the timer's data structure). The mnemonics provided by Allen-Bradley are certainly less confusing than referencing word and bit numbers for particular aspects of a timer's function!

A hallmark of the SLC 500's addressing scheme common to many legacy PLC systems is that the address labels for input and output bits explicitly reference the physical locations of the I/O channels. For instance, if an 8-channel discrete input card were plugged into slot 4 of an Allen-Bradley SLC 500 PLC, and you wished to specify the second bit (bit 1 out of a 0 to 7 range), you would address it with the following label: I:4/1. Addressing the seventh bit (bit number 6) on a discrete output card plugged into slot 3 would require the label 0:3/6. In either case, the numerical structure of that label tells you exactly where the real-world input signal connects to the PLC.

To illustrate the relationship between physical I/O and bits in the PLC's memory, consider this example of an Allen-Bradley SLC 500 PLC, showing one of its discrete input channels energized (the switch being used as a "Start" switch for an electric motor):



#### SLC 500 4-slot chassis

If an input or output card possesses more than 16 bits – as in the case of the 32-bit discrete output card shown in slot 3 of the example SLC 500 rack – the addressing scheme further subdivides each element into *words* and bits (each "word" being 16 bits in length). Thus, the address for bit number 27 of a 32-bit input module plugged into slot 3 would be 1:3.1/11 (since bit 27 is equivalent to bit 11 of word 1 – word 0 addressing bits 0 through 15 and word 1 addressing bits 16 through 31):

54



A close-up photograph of a 32-bit DC input card for an Allen-Bradley SLC 500 PLC system shows this multi-word addressing:



The first sixteen input points on this card (the left-hand LED group numbered 0 through 15) are addressed I:X.0/0 through I:X.0/15, with "X" referring to the slot number the card is plugged into. The next sixteen input points (the right-hand LED group numbered 16 through 31) are addressed I:X.1/0 through I:X.1/15.

Legacy PLC systems typically reference each one of the I/O channels by labels such as "I:1/3" (or equivalent<sup>6</sup>) indicating the actual location of the input channel terminal on the PLC unit. The IEC 61131-3 programming standard refers to this channel-based addressing of I/O data points as *direct addressing*. A synonym for direct addressing is *absolute addressing*.

Addressing I/O bits directly by their card, slot, and/or terminal labels may seem simple and elegant, but it becomes very cumbersome for large PLC systems and complex programs. Every time a technician or programmer views the program, they must "translate" each of these I/O labels to some real-world device (e.g. "Input I:1/3 is actually the *Start* pushbutton for the middle tank mixer motor") in order to understand the function of that bit. A later effort to enhance the clarity of PLC programming was the concept of addressing variables in a PLC's memory by arbitrary names rather than fixed codes. The IEC 61131-3 programming standard refers to this as *symbolic addressing* in contrast to "direct" (channel-based) addressing, allowing programmers arbitrarily

<sup>&</sup>lt;sup>6</sup>Some systems such as the Texas Instruments 505 series used "X" labels to indicate discrete input channels and "Y" labels to indicate discrete output channels (e.g. input X9 and output Y14). This same labeling convention is still used by Koyo in its DirectLogic and "CLICK" PLC models. Siemens continues a similar tradition of I/O addressing by using the letter "I" to indicate discrete inputs and the letter "Q" to indicate discrete outputs (e.g. input channel 10.5 and output Q4.1).

name I/O channels in ways that are meaningful to the system as a whole. To use our simple motor "Start" switch example, it is now possible for the programmer to designate input I:1/3 (an example of a *direct address*) as "Motor\_start\_switch" (an example of a *symbolic address*) within the program, thus greatly enhancing the readability of the PLC program. Initial implementations of this concept maintained direct addresses for I/O data points, with symbolic names appearing as supplements to the absolute addresses.

The modern trend in PLC addressing is to avoid the use of direct addresses such as I:1/3 altogether, so they do not appear anywhere in the programming code. The Allen-Bradley "Logix" series of programmable logic controllers is the most prominent example of this new convention at the time of this writing. Each I/O point, regardless of type or physical location, is assigned a *tag name* which is meaningful in a real-world sense, and these tag names (or *symbols* as they are alternatively called) are referenced to absolute I/O channel locations by a database file. An important requirement of tag names is that they contain no space characters between words (e.g. instead of "Motor start switch", a tag name should use hyphens or underscore marks as spacing characters: "Motor\_start\_switch"), since spaces are generally assumed by computer programming languages to be delimiters (separators between different variables).

# 4.3 Koyo "drum" sequencer instructions

The *drum* instruction offered in Koyo PLCs is a model of simplicity itself. This instruction is practically self-explanatory, as shown in the following example:





The three-by-three grid of squares represent steps in the sequence and bit states for each step. Rows represent steps, while columns represent output bits written by the drum instruction. In this particular example, a three-step sequence proceeds at the command of a single input (X001), and the drum instruction's advance from one step to the next proceeds strictly on the basis of elapsed time (a *time base* orientation). When the input is active, the drum proceeds through its timed sequence. When the input is inactive, the drum halts wherever it left off, and resumes timing as soon as the input becomes active again.

Being based on time, each step in the drum instruction has a set time duration for completion. The first step in this particular example has a duration of 10 seconds, the second step 15 seconds, and the third step 18 seconds. At the first step, only output bit Y001 is set. In the second step, only output bit Y002 is set. In the third step, output bits Y002 and Y003 are set (1), while bit Y001 is reset (0). The colored versus uncolored boxes reveal which output bits are set and reset with each step. The current step number is held in memory register DS1, while the elapsed time (in seconds) is stored in timer register TD1. A "complete" bit is set at the conclusion of the three-step sequence.

Koyo drum instructions may be expanded to include more than three steps and more than three output bits, with each of those step times independently adjustable and each of the output bits arbitrarily assigned to any writable bit addresses in the PLC's memory.

This next example of a Koyo drum instruction shows how it may be set up to trigger on *events* rather than on elapsed times. This orientation is called an *event base*:



Koyo CLICK PLC program

Here, a three-step sequence proceeds when enabled by a single input (X001), with the drum instruction's advance from one step to the next proceeding only as the different event condition bits become set. When the input is active, the drum proceeds through its sequence when each event condition is met. When the input is inactive, the drum halts wherever it left off regardless of the event bit states.

For example, during the first step (when only output bit Y001 is set), the drum instruction waits for the first condition input bit X002 to become set (1) before proceeding to step 2, with time being irrelevant. When this happens, the drum immediately advances to step 2 and waits for input bit X003 to be set, and so forth. If all three event conditions were met simultaneously (X002, X003, and X004 all set to 1), the drum would skip through all steps as fast as it could (one step per PLC program scan) with no appreciable time elapsed for each step. Conversely, the drum instruction will wait as long as it must for the right condition to be met before advancing, whether that event takes place in milliseconds or in days.

# 4.4 Allen-Bradley sequencer instructions

Rockwell (Allen-Bradley) PLCs use a more sophisticated set of instructions to implement sequences. The closest equivalent to Koyo's drum instruction is the Allen-Bradley SQO (Sequencer Output) instruction, shown here:



Rockwell SLC 500 PLC program

You will notice there are no colored squares inside the SQO instruction box to specify when certain bits are set or reset throughout the sequence, in contrast to the simplicity of the Koyo PLC's drum instruction. Instead, the Allen-Bradley SQO instruction is told to read a set of 16-bit words beginning at a location in the PLC's memory arbitrarily specified by the programmer, one word at a time. It steps to the next word in that set of words with each new position (step) value. This means Allen-Bradley sequencer instructions rely on the programmer already having pre-loaded an area of the PLC's memory with the necessary 1's and 0's defining the sequence. This makes the Allen-Bradley sequencer instruction more challenging for a human programmer to interpret because the bit states are not explicitly shown inside the SQO instruction box, but it also makes the sequencer far more flexible in that these bits are not fixed parameters of the SQO instruction and therefore may be dynamically altered as the PLC runs. With the Koyo drum instruction, the assigned output states are part of the instruction itself, and are therefore fixed once the program is downloaded to the PLC (i.e. they cannot be altered without editing and re-loading the PLC's program). With the Allen-Bradley, the on-or-off bit states for the sequence may be freely altered<sup>7</sup> during run-time. This is a very useful feature in recipe-control applications, where the recipe is subject to change at the whim of production personnel, and they would rather not have to rely on a technician or an engineer to re-program the PLC for each new recipe.

<sup>&</sup>lt;sup>7</sup>Perhaps the most practical way to give production personnel access to these bits without having them learn and use PLC programming software is to program an HMI panel to write to those memory areas of the PLC. This way, the operators may edit the sequence at any time simply by pressing "buttons" on the screen of the HMI panel, and the PLC need not have its program altered in any "hard" way by a technician or engineer.

The "Length" parameter tells the SQO instruction how many words will be read (i.e. how many steps are in the entire sequence). The sequencer advances to each new position when its enabling input transitions from inactive to active (from "false" to "true"), just like a count-up (CTU) instruction increments its accumulator value with each new false-to-true transition of the input. Here we see another important difference between the Allen-Bradley SQO instruction and the Koyo drum instruction: the Allen-Bradley instruction is fundamentally *event-driven*, and does not proceed on its own like the Koyo drum instruction is able to when configured for a *time* base.

Sequencer instructions in Allen-Bradley PLCs use a notation called *indexed addressing* to specify the locations in memory for the set of 16-bit words it will read. In the example shown above, we see the "File" parameter specified as **#B3:0**. The "#" symbol tells the instruction that this is a *starting* location in memory for the first 16-bit word, when the instruction's position value is zero. As the position value increments, the SQO instruction reads 16-bit words from successive addresses in the PLC's memory. If B3:0 is the word referenced at position 0, then B3:1 will be the memory address read at position 1, B3:2 will be the memory address read at position 2, etc. Thus, the "position" value causes the SQO instruction to "point" or "index" to successive memory locations.

The bits read from each indexed word in the sequence are compared against a static mask<sup>8</sup> specifying which bits in the indexed word are relevant. At each position, only these bits are written to the destination address.

As with most other Allen-Bradley instructions, the sequencer requires the human programmer to declare a special area in memory reserved for the instruction's internal use. The "R6" file exists just for this purpose, each element in that file holding bit and integer values associated with a sequencer instruction (e.g. the "enable" and "done" bits, the array length, the current position, etc.).

 $<sup>^{8}</sup>$ In this particular example, the mask value is FFFF hexadecimal, which means all 1's in a 16-bit field. This mask value tells the sequencer instruction to regard *all* bits of each B3 word that is read. To contrast, if the mask were set to a value of 000F hexadecimal instead, the sequencer would only pay attention to the four least-significant bits of each B3 word that is read, while ignoring the 12 more-significant bits of each 16-bit word. The mask allows the SQO instruction to only write to selected bits of the destination word, rather than always writing all 16 bits of the indexed word to the destination word.

Data File B3 (bin) -- BINARY

To illustrate, let us examine a set of bits held in the B3 file of an Allen-Bradley SLC 500 PLC, showing how each row (element) of this data file would be read by an SQO instruction as it stepped through its positions:

			`		'												
B3:0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	If $FIIe = #B3:0$ , then
B3:1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	Read at position = 1
B3:2	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	Read at position = 2
B3:3	0	0	0	1	0	0	0	0	0	1	0	1	0	1	1	0	Read at position = 3
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

The sequencer's position number is added to the file reference address as an *offset*. Thus, if the data file is specified in the SQO instruction box as **#B3:0**, then **B3:1** will be the row of bits read when the sequencer's position value is 1, **B3:2** will be the row of bits read when the position value is 2, and so on.

The *mask* value specified in the SQO instruction tells the instruction which bits out of each row will be copied to the destination address. A mask value of FFFFh (FFFF in *hexadecimal* format) means all 16 bits of each B3 word will be read and written to the destination. A mask value of 0001h means only the first (least-significant) bit will be read and written, with the rest being ignored.

Let's see what would happen with an SQO instruction having a mask value of 000Fh, starting from file index **#B3:0**, and writing to a destination that is output register **0:0.0**, given the bit array values in file **B3** shown above:



When this SQO instruction is at position 2, it reads the bit values 0010 from B3:2 and writes only those four bits to 0:0.0. The "X" symbols shown in the illustration mean that all the other bits in that output register are untouched – the SQO instruction does not write to those bits because they are "masked off" from being written. You may think of the mask's zero bits inhibiting source bits from being written to the destination word in the same sense that *masking tape* prevents paint from being applied to a surface.

The following Allen-Bradley SLC 500 PLC program shows how a pair of SQO instructions plus an on-delay timer instruction may be used to duplicate the exact same functionality as the "time base" Koyo drum instruction presented earlier:



The first SQO instruction reads bits in the B3 file array, sending only the three least-significant of them to the output register 0:0.0 (as specified by the 0007h mask value). The second SQO instruction reads integer number values from elements of the N7 integer file and places them into the "preset" register of timer T4:0, so as to dynamically update the timer's preset value with each step of the sequence. The timer, in turn, counts off each of the time delays and then enables both sequencers to advance to the next position when the specified time has elapsed. Here we see a tremendous benefit of the SQO instruction's indexed memory addressing: the fact that the SQO instruction reads its bits from arbitrarily-specified memory addresses means we may use SQO instructions to sequence any type of data existing in the PLC's memory! We are not limited to turning on and off individual bits as we are with the Koyo drum instruction, but rather are free to index whole integer numbers, ASCII characters, or any other forms of binary data resident in the PLC's memory.

Data file windows appear on the computer screen showing the bit array held in the B3 file as well as the timer values held in the N7 file. In this live screenshot, we see both sequencer instructions at position 2, with the second SQO instruction having loaded a value of 15 seconds from register N7:2 to the timer's preset register T4:0.PRE.

Note how the enabling contact address for the second SQO instruction is the "enable" bit of the first instruction, ensuring both instructions are enabled simultaneously. This keeps the two separate sequencers synchronized (on the same step).

#### 4.4. ALLEN-BRADLEY SEQUENCER INSTRUCTIONS

Event-based transitions may be implemented in Allen-Bradley PLCs using a complementary sequencing instruction called SQC (Sequencer Compare). The SQC instruction is set up very similar to the SQO instruction, with an indexed file reference address to read from, a reserved memory structure for internal use, a set length, and a position value. The purpose of the SQC instruction is to read a data register and compare it against another data register, setting a "found" (FD) bit if the two match. Thus, the SQC instruction is ideally suited for detecting when certain conditions have been met, and thus may be used to enable an SQO instruction to proceed to the next step in its sequence.

The following program example shows an Allen-Bradley MicroLogix 1100 PLC programmed with both an SQO and an SQC instruction:



The three-position SQO (Sequencer Output) instruction reads data from B3:1, B3:2, and B3:3, writing the four least-significant of those bits to output register 0:0.0. The three-position SQC (Sequencer Compare) instruction reads data from B3:6, B3:7, and B3:8, comparing the four least-significant of those bits against input bits in register I:0.0. When the four input bit conditions match the selected bits in the B3 file, the SQC instruction's FD bit is set, causing both the SQO instruction and the SQC instruction to advance to the next step.

Lastly, Allen-Bradley PLCs offer a third sequencing instruction called *Sequencer Load* (SQL), which performs the opposite function as the Sequencer Output (SQO). An SQL instruction takes data from a designated source and writes it into an indexed register according to a position count value, rather than reading data from an indexed register and sending it to a designated destination as does the SQO instruction. SQL instructions are useful for reading data from a live process and storing it in different registers within the PLC's memory at different times, such as when a PLC is used for *datalogging* (recording process data).

# 4.5 SELogic control equations

Some programmable logic controllers use Boolean-style equations to describe logical functions. A company called Schweitzer Engineering Laboratories (SEL) manufactures control equipment for the electric power industry, including a wide range of *protective relays* and *programmable automation* controllers, which use Boolean equations to describe logic. Their brand name for this is *SELogic*.

#### 4.5.1 Basic logical functions

For example, suppose we wished to implement a three-input AND function as well as a three-input OR function in a SEL controller with the three inputs being IN201, IN202, and IN203. The symbolic functions and their equivalent SELogic equations are shown in the following illustration:



SEL programming software<sup>9</sup> provides text-entry fields for typing these control equations. A more primitive interface makes use of the built-in serial terminal server capability of SEL controllers, allowing programming edits to be made with nothing more than a serial terminal (e.g. personal computer running terminal emulator software such as **Termite** or **PuTTY** or **Hyperterminal**) on a command-line interface. The following screenshot shows these two SELogic control equations being edited in a window:

Outputs (100)	
SELogic Control Equations	
OUT101 OUT101 (SELogic)	20.044
IN201 AND IN202 AND IN203	
OUT102 OUT102 (SELogic)	
IN201 OR IN202 OR IN203	

In this particular controller (an SEL-2440 "DPAC" Discrete Programmable Automation Controller) all inputs are numbered beginning with 201 and all outputs beginning with 101.

 $<sup>^9\</sup>mathrm{SEL}\xspace{-5030}$  AcSEL erator QuickSet software was used for all of these examples.

A simple combinational logic function is shown in this next illustration:



All SELogic equations support text *comments*, which is an important detail for any non-trivial coding. Comments are ignored by the controller, but serve as notes for any future programmers examining the code. In SELogic, comments are preceded by the hashtag symbol (#). For example, here is a comment as it would appear following an SELogic equation for a simple AND function:

OUT102 = IN205 AND IN208 # THIS IS AN AND FUNCTION

Characters to the left of the **#** are regarded as executable code by the controller, while text to the right of the **#** are merely comments.

#### 4.5.2 Set-reset latch instructions

Set-Reset (or S-R) latch instructions are useful for applications such as motor starter controls, where the controller must "latch" the motor on after momentary closure of the "Start" pushbutton switch, and latch the motor off following momentary actuation of the "Stop" pushbutton. PLCs programmed in Ladder Diagram code typically use either *retentive coils* ("Set" and "Reset" coils) or seal-in contacts "wired" in parallel with the Start contact instruction. SELogic provides a dedicated function for latching called a *Latch Bit*.

Latch instructions in SELogic require a number of different parameters to be specified. First and foremost is the allocation of usable latch instructions in the controller's memory. Like legacy-style industrial PLCs with limited memory resources, the programmer first needs to declare to the PLC how many counters, timers, etc. will be used in the program so that the PLC may allocate limited memory to those functions. For SELogic latches, this is done through the ELAT parameter. For example the equation ELAT = 3 allocates three latches which are called LT01, LT02, and LT03. After that, each latch requires assignment of its *Set* (SET) and Reset (RST) inputs. A screenshot showing the Set and Reset inputs of a single latch instruction appears here:

Latch E	Sit Set/Reset	Equations
1	Range = 1 to 32, N	
Latch Bit 1		
SET01 SET0	1 (SELogic)	
IN201		
RST01 RST0	1 (SELogic)	
IN202		

Listed as plain ASCII text, the SELogic equations would appear as follows to declare a single latch instruction and then configure its two inputs as shown in the previous screenshot, where IN201 causes the latch to set and IN202 causes the latch to reset:

```
ELAT = 1
SET01 = IN201
RST01 = IN202
```

In SELogic, the name of the latch (e.g. LT01) is its output bit. Using the example just shown, we could direct output OUT108 to be controlled by LT01's output with an additional line of SELogic code:

OUT108 = LT01

#### 4.5.3 One-shot instructions

One-shot functionality also is provided within SELogic by the R\_TRIG and F\_TRIG instructions, referring to *rising-edge* and *falling-edge*, respectively. The purpose of a "one-shot" instruction is to activate the output for a single scan of the controller's program upon a false-to-true (rising edge) or true-to-false (falling edge) transition of the input signal. This next illustration shows examples of each:


#### 4.5.4 Counter instructions

Counter instructions in SELogic require a number of different parameters to be specified. First and foremost is the allocation of usable counter instructions in the controller's memory. Like legacy-style industrial PLCs with limited memory resources, the programmer first needs to declare to the PLC how many counters, timers, etc. will be used in the program so that the PLC may allocate limited memory to those functions. For SELogic counters, this is done through the ESC parameter. For example the equation ESC = 2 allocates two counters which are called SC01 and SC02. After that, each counter requires assignment of input variables such as *Preset Value* (PV), Reset (R), Load PV (LD), *Count up input* (CU) and Count down input (CD). A screenshot showing all of these variables (enabling two counters, followed by the parameters for Counter 1) appears here:

ori 1. o. ...

2	Range = 1 to 32, N	
SELogic Co	unter 1	
SCO1PV Co	unter Preset Value SC01PV	
50	Range = 1 to 65000	
SC01R Cou IN204	nter Reset Input SC01R (SELogic)	
SC01LD Co	unter Load PV Input SC01LD (SELog	gic)
IN203		
IN203 SC01CU Co	unt Up Input SC01CU (SELogic)	
IN203 SC01CU Co IN201	unt Up Input SC01CU (SELogic)	
IN203 SC01CU Co IN201 SC01CD Co	unt Up Input SC01CU (SELogic) unt Down Input SC01CD (SELogic)	

Listed as plain ASCII text, the SELogic equations would appear as follows to declare two counter instructions and then configure the first counter's inputs: a preset value of 50, IN204 causing it to reset, IN203 causing it to load the preset value, IN201 causing it to increment, and IN202 causing it to decrement:

ESC = 2 SC01PV = 50 SC01R = IN204 SC01LD = IN203 SC01CU = IN201SC02CD = IN202 Once configured, the accumulated value of the first counter instruction is addressed simply as SC01. Discrete outputs based on the attainment of certain count values may be driven by comparison statements such as =, >, and <. For example, to activate output OUT107 whenever the first counter's accumulated value exceeds 12, you would need to write the following SELogic equation:

OUT107 = SC01 > 12

The comparison statement SC01 > 12 is either true or false – that is to say, it is a *Boolean* quantity – and so its truth-value fits well with the discrete status of output OUT107. When the counter's value exceeds 12, OUT107 activates; if equal to or less than 12, OUT107 de-activates.

#### 4.5.5 Timer instructions

Like counter instructions, *timer* instructions in SELogic require a number of different parameters to be specified. First and foremost is the allocation of usable timer instructions in the controller's memory. Like legacy-style industrial PLCs with limited memory resources, the programmer first needs to declare to the PLC how many counters, timers, etc. will be used in the program so that the PLC may allocate limited memory to those functions. For SELogic timers, this is done through the ESV parameter. For example the equation ESV = 1 allocates memory space for a single timer called SV01. After that, the timer requires assignment of input variables such as *Timer Pickup* (PU), *Timer Dropout* (DO), and *Input*. A screenshot showing all of these variables for a single timer appears here:

SELogic Variables/Timers

	Range = 1 to 64, N
ELogic Va	riable 1
SV01 Inpu	(SELogic)
IN201	
SV01PU Tir	ner Pickup (seconds)
	Range = 0.000 to 16000.000

The SELogic timer instruction is capable of both on-delay and off-delay. This is the meaning of the "pickup" and "dropout" time delays: the *pickup* time refers to a time delay on activation of the timer (i.e. on-delay) while the *dropout* time refers to a time delay on de-activation (i.e. off-delay). For example, if we only wished to have an on-delay SV01 timer with a delay time of 5 seconds we would set SV01PU = 5 and SV01DO = 0. If we merely wished for an off-delay timer with a delay time of 8 seconds, we would set SV01PU = 0 and SV01DO = 8. If we wanted a timer to exhibit *both* an on-delay of 2 seconds *and* an off-delay of 3 seconds, we would set the pickup and dropout parameters exactly as shown in the above screenshot image. Note the resolution of these time settings: down to 0.001 seconds, or 1 millisecond.

In SELogic, the name of the timer (e.g. SV01) is its input, or controlling, bit. The time-delayed output bit of the timer is addressed by adding the suffix "T" to the timer name. Using our example of timer SV01, its time-delayed output bit would be SV01T. Below is an example of this timer's output being set to control output OUT105:

OUT105 = SV01T

## Chapter 5

# Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read<sup>1</sup> the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture<sup>2</sup>, the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

<sup>&</sup>lt;sup>1</sup>Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading provess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

 $<sup>^{2}</sup>$ Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

GENERAL CHALLENGES FOLLOWING TUTORIAL READING

- <u>Summarize</u> as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an <u>intelligent child</u>: as simple as you can without compromising too much accuracy.
- <u>Simplify</u> a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text <u>make the most sense</u> to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to <u>misunderstand the text</u>, and explain why you think it could be confusing.
- Identify any <u>new concept(s)</u> presented in the text, and explain in your own words.
- Identify any <u>familiar concept(s)</u> such as physical laws or principles applied or referenced in the text.
- Devise a <u>proof of concept</u> experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to <u>disprove</u> a plausible misconception.
- Did the text reveal any <u>misconceptions</u> you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- <u>Devise a question</u> of your own to challenge a reader's comprehension of the text.

GENERAL FOLLOW-UP CHALLENGES FOR ASSIGNED PROBLEMS

- Identify where any <u>fundamental laws or principles</u> apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a <u>thought experiment</u> to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own <u>strategy</u> for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the <u>most challenging part</u> of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any <u>extraneous</u> information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- <u>Simplify</u> the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a <u>limiting case</u> (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the <u>real-world meaning</u> of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it <u>qualitatively</u> instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it <u>quantitatively</u> instead, proposing simple numerical values for the variables.
- Were there any <u>assumptions</u> you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

GENERAL FOLLOW-UP CHALLENGES FOR EXPERIMENTS OR PROJECTS

- In what way(s) was this experiment or project <u>easy to complete</u>?
- Identify some of the <u>challenges you faced</u> in completing this experiment or project.

- Show how <u>thorough documentation</u> assisted in the completion of this experiment or project.
- Which <u>fundamental laws or principles</u> are key to this system's function?
- Identify any way(s) in which one might obtain <u>false or otherwise misleading measurements</u> from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system <u>unsafe</u>?

## 5.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking<sup>3</sup>. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

 $<sup>^{3}</sup>Analytical$  thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. Synthetic thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

## 5.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" - Francis Bacon

Francis Bacon's advice is a blueprint for effective education: <u>reading</u> provides the learner with knowledge, <u>writing</u> focuses the learner's thoughts, and <u>critical dialogue</u> equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do <u>all</u> of the following after reading any instructional text:

 $\checkmark$  Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

 $\checkmark$  Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problemsolving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

 $\checkmark$  Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

 $\checkmark$  Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

 $\checkmark$  Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

 $\checkmark$  Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

78

#### 5.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.



## 5.1.3 Relay ladder logic analogy for a PLC

Analyze the status of all relay contacts and lamps in this hard-wired relay "ladder logic" control circuit, annotating the diagram to show electrical conductivity and non-conductivity of all contacts and energization statuses of all relay coils and loads:



Assume the following input conditions:

- Pushbutton switch *unpressed*
- Pressure *above* trip threshold
- Selector switch in its *right-hand* position

Now, analyze the status of this PLC-controlled system assuming the same input conditions. Note the distinction between the 120 VAC circuitry and the "virtual circuit" in the blue-shaded area representing the program executed by the PLC's microprocessor. As with the relay-based system, annotate all real-world contact conductivities and coil energization statuses, as well as show which of the PLC program's elements should be "colored" to reflect virtual conductivity and virtual energization:



Assume the same input conditions:

- Pushbutton switch *unpressed*
- Pressure *above* trip threshold
- Selector switch in its *right-hand* position

How is the PLC-controlled system similar to the hard-wired relay control system? How is it different?

Challenges

- What advantages does the PLC enjoy over the hard-wired relay control system?
- What advantages does the electromechanical relay design enjoy over the PLC control system?

#### 5.1.4 Sourcing versus sinking PLC I/O

Discrete (on/off) I/O for PLCs often works on AC (alternating current) power. AC input circuitry usually consists of an optocoupler (LED) with rectification and a large dropping resistor to allow 120 Volt AC operation. AC output circuitry usually consists of TRIACs. Explain how both of these technologies work.

DC I/O for a PLC generally consists of optocoupled LEDs for inputs and bipolar transistors for outputs. Some examples are shown in the following schematics. Note carefully the different variations:



Determine for each of these input and output module types, whether they would be properly designated *sourcing* or *sinking*.

#### Challenges

• Determine how real input and output devices (e.g. switches, solenoid coils) would need to be connected to the I/O terminals of these modules.

#### 5.1.5 Sketching wires to PLC discrete I/O

Sketch the wires necessary to connect two pressure switches and two relay coils to the following Allen-Bradley MicroLogix 1000 PLC (model 1761-L10BWA, with 6 discrete DC inputs either sourcing or sinking, and 4 discrete relay contact outputs). Be sure to wire the two switches so they *source* current to the PLC's inputs (the low-pressure switch to I/2 and the high-pressure switch to I/5, normally-open contacts on both) and wire the relay coils so the PLC *sources* current to them (0/0 and 0/1):







Challenges

• Define "sinking" and "sourcing" as these terms apply to PLC I/O terminals.

## 5.1.6 Two different motor control programs

Two technicians, Jill and Bob, work on programming Siemens S7-200 PLCs to control the starting and stopping of electric motors. Both PLCs are wired identically, as shown:



However, despite being wired identically, the two technicians' PLC programs are quite different. Jill's program uses *retentive coil* instructions ("Set" and "Reset" coils) while Bob's uses a "seal-in" contact instruction to perform the function of latching the motor on and off:



Explain how both of these PLC programs function properly to control the starting and stopping of the electric motor.

Challenges

- It is ordinarily a bad thing to assign identical bit addresses to multiple coil instructions in a PLC program. With Jill's retentive coil program, however, this is not only permissible but in fact necessary for its proper operation. Explain why this is.
- A common misconception of students first learning PLC programming is to think that the type of contact instruction used in the PLC program must match the type of switch contact connected to that input (e.g. "A N.O. PLC instruction must go with a N.O. switch"). Explain why this is incorrect.
- Explain how both PLC programs will react if both the "start" and "stop" pushbuttons are simultaneously pressed.
- Alter both PLC programs to be "fail-safe" (i.e. shut the motor off) if ever the stop pushbutton switch fails circuit open.

## 5.1.7 Redundant coils in a program

In relay ladder logic (RLL) programming, it is considered bad practice to have multiple instances of an identical (standard) "relay" coil in a program:



Explain why this is considered poor practice in PLC programming. Next, determine the status of the Pump\_run output channel given the following bit states:

- Timer\_01 = 1
- Level\_low = 1
- Switch\_hand = 0
- $OL_contact = 0$
- $Sump_wet = 0$

#### Challenges

• Explain why it *is* acceptable to have redundant retentive coils in a PLC Ladder Diagram program.

## 5.1.8 Determining bit statuses from switch conditions

#### PLC #1

Suppose we have an Allen-Bradley MicroLogix 1000 PLC connected to three momentary-contact pushbutton switches as shown in this illustration:



Determine the bit statuses of I:0/0, I:0/1, and I:0/2 when switch A is unpressed (released), switch B is unpressed (released), and switch C is pressed.

#### PLC #2

Suppose we have a Siemens S7-200 PLC connected to two process switches as shown in this illustration:



Determine the bit statuses of 10.2 and 11.1 when the temperature switch senses 194  $^o{\rm F}$  and the flow switches senses 19 GPM.

#### **PLC #3**

Suppose we have an Allen-Bradley SLC 500 PLC connected to two process switches as shown in this illustration:



Determine the bit statuses of I:1/3 and I:1/5 when the level switch senses 3 feet and the pressure switch senses 14 PSI.

#### PLC #4

Suppose we have a Siemens S7-200 PLC connected to two process switches as shown in this illustration:



Determine the bit statuses of 10.2 and 11.1 when the temperature switch senses 122  $^o{\rm F}$  and the flow switches senses 15 GPM.

#### **PLC #5**

Suppose we have an Allen-Bradley MicroLogix 1000 PLC connected to three momentary-contact pushbutton switches as shown in this illustration:



Determine the bit statuses of I:0/0, I:0/1, and I:0/3 when switch A is pressed, switch B is unpressed (released), and switch C is pressed.

#### Challenges

• Explain the meaning of the word "normal" as it applies to a switch contact.

#### 5.1.9 Determining necessary switch conditions for bit statuses

Suppose we have an Allen-Bradley SLC 500 PLC connected to two process switches as shown in this illustration:



Determine the process conditions necessary to generate the following input bit statuses in the PLC's memory:

- I:1/3 = 1
- I:1/5 = 0

Challenges

- What do the numbers "1" and "3" mean for input bit I:1/3 in this Allen-Bradley PLC?
- What do the numbers "1" and "5" mean for input bit I:1/5 in this Allen-Bradley PLC?
- How would you specify the bit address for the fourth channel on the other input card of this PLC?

## 5.1.10 Determining color highlighting from bit statuses

#### PLC #1

Suppose a Siemens 545 PLC has the following input bit states:

- X1 = 0
- X2 = 1
- X3 = 0

Sketch color highlighting for the contacts and coils in the PLC's program given these bit statuses, also determining the status of output bit Y1:



#### PLC #2

Suppose we have a Siemens S7-200 PLC connected to a pair of momentary-contact pushbutton switches and light bulbs as shown in this illustration:



Examine the following relay ladder logic (RLL) program for this Siemens PLC, determining the statuses of the two lamps provided both switches are simultaneously pressed by a human operator:



Finally, draw color highlighting showing how these "contact" instructions will appear in an online editor program given the stated input conditions.

#### Challenges

- Sketch a logic gate diagram implementing the same function as the first PLC program.
- Identify the significance of the labels "X" and "Y" for this PLC's bits. What do you suppose "X" signifies? What do you suppose "Y" signifies?

## 5.1.11 Determining color highlighting from switch conditions

Suppose we have an Allen-Bradley model "SLC 500" PLC connected to a pair of process switches and lamps as shown in this illustration:



Examine the following "offline-view" (i.e. non-color-highlighted) relay ladder logic (RLL) program for this Allen-Bradley PLC, determining the statuses of the two lamps given a temperature of 172  $^{o}$ F and a flow of 5.1 GPM:



Finally, draw color highlighting showing how these "contact" instructions will appear in an online editor program given the stated input conditions.

Challenges

- Identify the significance of the labels "I" and "O" for this PLC's bits.
- Identify the significance of the first and second numbers in each bit label (e.g. the numbers "1" and "2" in the bit address I:1/2, for example). What pattern do you see as you compare the I/O connections with the respective contact instructions in the PLC program?

## 5.1.12 Determining bit statuses from color highlighting

#### **PLC #1**

Examine this "live" display of a Siemens S7-300 PLC's program, and from this determine all bit statuses represented by the color highlighting in this ladder logic program:



- I0.2 =
- I0.5 =
- I1.1 =
- Q0.1 =
- Q0.6 =

#### PLC #2

Examine this "live" display of a Siemens S7-300 PLC's program, and from this determine all bit statuses represented by the color highlighting in this ladder logic program:



- I0.7 =
- I1.1 =
- Q0.1 =
- Q0.3 =

#### **PLC #3**

Examine this "live" display of an Allen-Bradley PLC's program, and from this determine all bit statuses represented by the color highlighting in this ladder logic program:



- I:0/1 =
- I:0/2 =
- I:0/7 =
- 0:4/2 =
- 0:4/5 =

#### PLC #4

Examine this "live" display of an Allen-Bradley PLC's program, and from this determine all bit statuses represented by the color highlighting in this ladder logic program:



- I:0/0 =
- I:0/1 =
- I:0/2 =
- I:0/3 =
- I:0/4 =
- 0:2/1 =
- 0:2/2 =

#### Challenges

- PLC training expert Ron Beaufort teaches students to think of a "normally-open" PLC program contact instruction as a command to the PLC's processor to "Go look for a 1". Conversely, he teaches students to think of a "normally-closed" instruction as a command to "Go look for a 0". Explain what Mr. Beaufort means by these phrases, and how this wisdom relates to this particular problem. Incidentally, Mr. Beaufort's excellent instructional videos (available freely on YouTube) are quite valuable to watch!
- Identify the significance of the labels "I" and "Q" for this PLC's bits. What do you suppose "I" signifies? What do you suppose "Q" signifies?

## 5.1.13 Determining process switch stimuli from color highlighting

Suppose we have a PLC connected to three process switches as shown in this illustration:



Based on the highlighting you see in the "live" PLC program display, determine as best you can the pressure and temperature stimulating each switch.

Also, determine the following bit states within the PLC's memory corresponding to the same color-highlighting:

- X1 =
- X3 =
- X5 =
- Y1 =

• Y2 =

Challenges

• Identify which LED indicators on the PLC's face would be lit in this condition.

#### 5.1.14 Determining necessary switch contact types

The following PLC program was written to control the operation of a large electric motor-driven pump. A variety of "permissive" inputs protect the pump from damage under abnormal conditions, and one permissive in particular ("valve open") will only allow the pump to start up if one of the valves in the piping system is in the full-open position:



Identify the type of contact (either NO or NC) necessary for each of these electrical switch contacts, based on the trip condition (either high or low) and how each input is applied in the PLC program:

- Start pushbutton = NO or NC?
- Throttling valve open limit = NO or NC?
- Stop pushbutton = NO or NC?
- High bearing temperature = NO or NC?
## 5.1. CONCEPTUAL REASONING

- High vibration = NO or NC?
- High motor temperature = NO or NC?
- Low oil pressure = NO or NC?

# Challenges

• Explain why it is a helpful problem-solving strategy to first identify the necessary virtual contact *coloring* which will allow the motor to start up and keep running (i.e. the condition of all permissives during correct operating conditions).

## 5.1.15 Motor start-up limit counter

This Koyo "CLICK" PLC has been programmed to control the starting and stopping of an electric motor, including a *counter* instruction to prevent the motor from being started up more than a specified number of times:



Identify the counter instruction in the program shown, its input "connections", and also how the result of the counter reaching its pre-set limit forces the motor to stop. Also, determine the maximum number of times the motor may be started up, assuming the counter's current value goes to zero when the Reset button is pressed.

Finally, determine how to modify this PLC program so that the counter may be manually reset by the operator without requiring a separate pushbutton labeled "Reset".

#### Challenges

- If an operator presses the "Start" button multiple times while the motor is already running, do these button-presses get counted by the counter instruction, or do only the real motor start-up events get counted?
- What do you suppose the label "CTD1" represents inside the counter instruction?
- Note the number of times the bit Y1 is referenced inside this PLC program: once in a coil instruction and twice in contact instructions. Is there any limit to how many times a bit address may be used in a PLC program?
- Describe the purpose of the first contact instruction labeled Y1 in this program, explaining why it is often referred to as a *seal-in* contact.

# 5.1.16 Allen-Bradley counter program

Analyze this Allen-Bradley PLC program and explain what it is supposed to do:



## 5.1.17 Room occupancy counter

This Siemens S7-200 PLC has been programmed to count the number of people in a room, by incrementing a counter every time a person enters through the doorway, and decrementing that same counter whenever someone exits through the same doorway. The two optical switches activate whenever their respective light beams are broken by someone passing through. Their horizontal separation is just a couple of inches – much less than the girth of a person's torso. The operating status of each switch is that it energizes the PLC input when the light beam is broken:



Examine the program in this PLC for counting people, and determine how it is able to differentiate between a person entering the room and a person leaving the room:



108

#### 5.1. CONCEPTUAL REASONING

Challenges

- Explain how a *timing diagram* of the switch states would be helpful in analyzing the operation of this PLC program.
- *Transition* (edge-detecting) functions are implemented in Allen-Bradley PLCs using the *one-shot rising* (OSR) instruction. Research how the OSR instruction is used, and how it differs from the "P" and "N" contacts shown in this Siemens PLC program.
- Will this system still function properly if the optical sensors are spaced farther apart than the width of a human body? Explain why or why not.

# 5.1.18 PLC timing diagrams

Identify the type of PLC timer instruction (i.e. *on-delay*, *off-delay*, or *retentive*) capable of producing the following timing diagrams, as well as each timer instruction's preset value in *seconds*. Assume each horizontal-axis division of this timing diagram represents one second:



Challenges

• Modify the amount of time delay for any of these timer functions, and re-sketch the output.

## 5.1.19 Sequenced-start conveyor belts

A gravel-crushing operation uses three long conveyor belts to move rock from the quarry to the crusher. The belts must be started up in a particular sequence to avoid overloading the electric motors driving them:



First, determine a start-up sequence that makes sense: which conveyor belt should start first, next, and last? What might happen if the sequence were reversed? Why not simply start all conveyor motors simultaneously?

#### 5.1. CONCEPTUAL REASONING

This operation uses a Siemens S7 series PLC to control the three conveyor belts. Analyze this program and explain how it accomplishes the task of starting up the three conveyors in sequence:



Lastly, determine where you might add a contact instruction for an *emergency shutoff* safety switch, so that all three conveyors stop simultaneously if ever the safety switch is actuated.

#### Challenges

• How long is the time delay between conveyor start-ups? How might this time delay be altered

if needed?

- Suppose a warning siren were added to the system, sounding for a full 15 seconds before the first conveyor belt starts. How would you modify the PLC program to include this additional functionality?
- Suppose a technician uses the PLC's *force* utility to force bit T2 to a "0" state. How will this affect the operation of the system? Could the consequences of this force be dangerous in any way?
- Suppose a technician uses the PLC's *force* utility to force bit Q0.1 to a "0" state. How will this affect the operation of the system? Could the consequences of this force be dangerous in any way?

## 5.1.20 Switch contact types for a timed conveyor control

Suppose an Allen-Bradley PLC controls the starting and stopping of a conveyor belt, using a timer to sound an audible warning siren for 5 seconds before the conveyor belt starts up (to warn people before the belt begins to move):



Determine the necessary contact connections (form-A or form-B) on the real-life Start, Stop, and emergency Pull-Cable switches to complement the virtual contact types in the PLC program.

Start switch = form-A or form-B?

Stop switch = form-A or form-B?

Pull-Cable switch = form-A or form-B?

#### Challenges

• How could you modify this program so that the operator has to hold the "Start" pushbutton switch actuated for the duration of the warning siren before the motor would start (i.e.

everything would simply stop if the operator only *momentarily* pressed the "Start" button)?

- Suppose a technician decides to use the *force* utility in the PLC to force bit B3:0/0 to a "0" state in order to test the warning siren's operation without actually starting up the conveyor belt. Explain what is flawed with this testing strategy, and identify a better approach.
- How will this system behave if the pull-cable switch fails open?
- How will this system behave if the stop switch fails shorted?

## 5.1. CONCEPTUAL REASONING

## 5.1.21 Air compressor control program

An Allen-Bradley Logix5000 PLC is used to control the starting and stopping of an air compressor based on momentary-contact pushbutton switch inputs as well as high and low pressure switches (PSH and PSL, respectively). Analyze this program and explain how it is supposed to work:







(continued from previous page)

In particular, answer these following questions:

- Determine the "normal" electrical statuses of all switches (e.g. NO or NC) connected to the inputs of this PLC, based on an examination of the respective contact instructions within the PLC program.
- Why is is important that a *retentive* timer instruction be used for the calculation of total run-time?
- What is the significance of the maintenance warning light controlled by this PLC?

### Challenges

- Note how all instructions in this Logix5000 PLC program are addressed by *tagname* rather than by hardware addresses (e.g. 1:2/6, 0:3/1). How do you suppose the PLC "knows" which real I/O points to associate with which instructions in the program?
- How will this system behave if the reset switch fails shorted?

# 5.1. CONCEPTUAL REASONING

- How will this system behave if the high-pressure switch fails open?
- How will this system behave if the high-pressure switch fails shorted?
- How will this system behave if the low-pressure switch fails open?
- How will this system behave if the low-pressure switch fails shorted?

# 5.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problemsolving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases<sup>4</sup>" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely<sup>5</sup> on an answer key!

<sup>&</sup>lt;sup>4</sup>In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

<sup>&</sup>lt;sup>5</sup>This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students* to be self-sufficient thinkers. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

#### 5.2. QUANTITATIVE REASONING

## 5.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation ( $\sigma$ ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as  $1.25663706212(19) \times 10^{-6}$  H/m represents a center value (i.e. the location parameter) of  $1.25663706212 \times 10^{-6}$  Henrys per meter with one standard deviation of uncertainty equal to  $0.0000000000019 \times 10^{-6}$  Henrys per meter.

Avogadro's number  $(N_A) = 6.02214076 \times 10^{23} \text{ per mole } (\text{mol}^{-1})$ 

Boltzmann's constant  $(k) = 1.380649 \times 10^{-23}$  Joules per Kelvin (J/K)

Electronic charge  $(e) = 1.602176634 \times 10^{-19}$  Coulomb (C)

Faraday constant  $(F) = 96,485.33212... \times 10^4$  Coulombs per mole (C/mol)

Magnetic permeability of free space  $(\mu_0) = 1.25663706212(19) \times 10^{-6}$  Henrys per meter (H/m)

Electric permittivity of free space  $(\epsilon_0) = 8.8541878128(13) \times 10^{-12}$  Farads per meter (F/m)

Characteristic impedance of free space  $(Z_0) = 376.730313668(57)$  Ohms  $(\Omega)$ 

Gravitational constant (G) = 6.67430(15)  $\times$   $10^{-11}$  cubic meters per kilogram-seconds squared (m^3/kg-s^2)

Molar gas constant (R) = 8.314462618... Joules per mole-Kelvin (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (*h*) = **6.62607015** ×  $10^{-34}$  joule-seconds (J-s)

Stefan-Boltzmann constant ( $\sigma$ ) = 5.670374419... × 10<sup>-8</sup> Watts per square meter-Kelvin<sup>4</sup> (W/m<sup>2</sup>·K<sup>4</sup>)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from http://physics.nist.gov/constants, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

#### 5.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell B3, cell A8). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	Α	В	С	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells A1 through A3 and cells C1 through C3 exist solely for readability and are not involved in any calculations. Cell B1 contains a sample distance value while cell B2 contains a sample time value. The formula for computing speed is contained in cell B3. Note how this formula begins with an "equals" symbol (=), references the values for distance and speed by lettered column and numbered row coordinates (B1 and B2), and uses a forward slash symbol for division (/). The coordinates B1 and B2 function as *variables*<sup>6</sup> would in an algebraic formula.

When this spreadsheet is executed, the numerical value 39.74576 will appear in cell B3 rather than the formula = B1 / B2, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell B1 or a different value for time is entered into cell B2, cell B3's value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell B3 may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in B1 and B2. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

<sup>&</sup>lt;sup>6</sup>Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

#### 5.2. QUANTITATIVE REASONING

 $Common^7$  arithmetic operations available for your use in a spreadsheet include the following:

- Addition (+)
- Subtraction (-)
- Multiplication (\*)
- Division (/)
- Powers (^)
- Square roots (sqrt())
- Logarithms (ln(), log10())

Parentheses may be used to ensure<sup>8</sup> proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of  $ax^2 + bx + c$ :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	В		
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)		
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)		
3	a =	9		
4	b =	5		
5	C =	-2		

This example is configured to compute roots<sup>9</sup> of the polynomial  $9x^2 + 5x - 2$  because the values of 9, 5, and -2 have been inserted into cells B3, B4, and B5, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new *a*, *b*, and *c* coefficients into cells B3 through B5. The numerical values appearing in cells B1 and B2 will be automatically updated by the computer immediately following any changes made to the coefficients.

<sup>&</sup>lt;sup>7</sup>Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

<sup>&</sup>lt;sup>8</sup>Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

<sup>&</sup>lt;sup>9</sup>Reviewing some algebra here, a root is a value for x that yields an overall value of zero for the polynomial. For this polynomial  $(9x^2 + 5x - 2)$  the two roots happen to be x = 0.269381 and x = -0.82494, with these values displayed in cells B1 and B2, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	В	С
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	C =	-2	

Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary<sup>10</sup> – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

 $<sup>^{10}</sup>$ My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out how I constructed a solution. This is a general principle I believe all computer programmers should follow: document and arrange your code to make it easy for other people to learn from it.

#### 5.2. QUANTITATIVE REASONING

## 5.2.3 Frequency divider program

An important pump in a chemical process is turned by an electric motor, and operators want to have visual indication in the control room that the pump is indeed turning. There is no way to attach a speed switch to the pump shaft (that would be too easy!). Instead, someone has installed a proximity switch near the pump shaft, situated to pick up the passing of a keyway in the shaft with each rotation. Thus, the proximity switch will output a "pulse" signal when the pump shaft is spinning:



Operations personnel wanted the indicator light in the control room to blink when the pump is running, for an indication of shaft motion. The problem is, the shaft turns much too fast (approximately 1750 RPM) to directly drive the indicator with the proximity switch signal, and so an Allen-Bradley PLC was programmed to produce a slower blink using this program:



Explain how this program works to fulfill the function of a *frequency divider*, converting the high-speed pulse signal of the proximity switch into a low-speed blink for the operator light.

## Challenges

- Explain how a *frequency divider* circuit built out of J-K flip-flop integrated circuits functions, and then describe how this PLC program is similar in principle.
- Explain how to speed up the blinking rate of the light for any given motor shaft speed.

## 5.2.4 Integer format error between PLC and HMI

Suppose a PLC program contains a counter instruction that counts in unsigned 16-bit integer format. An HMI connected to this PLC, however, is configured to read and interpret this counter's accumulated value as a *BCD* number instead of unsigned binary.

Determine how the HMI will interpret and display a PLC counter accumulated value of **38199** (decimal).

#### Challenges

- If the PLC's counter increments by one, what will the HMI display read?
- If the PLC's counter increments by two, what will the HMI display read?
- If the PLC's counter increments by three, what will the HMI display read?

# 5.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

## 5.3.1 Incorrect PLC output wiring

The PLC shown below is supposed to control the energization of a lamp and a small motor. Someone wired both loads to different PLC output channels as shown in the illustration below, but was surprised to find neither load energized when its respective PLC output channel activated:



Explain what is wrong in this circuit, and then re-sketch the wiring so that these loads will function properly. Incidentally, this same error happens to be a *very* common misconception among students new to PLCs. Identify what the fundamental misconception is, and how it may be remedied.

## Challenges

- What information should we ideally have regarding the PLC and the two loads shown in order to design a complete circuit that will function safely and reliably?
- From the information available in the illustration, is it possible to formulate an educated guess about the type of discrete output channels offered by this PLC?

#### 5.3. DIAGNOSTIC REASONING

## 5.3.2 Troubleshooting motor control program

Suppose we have an Allen-Bradley MicroLogix 1000 controller connected to a pair of momentarycontact pushbutton switches and contactor controlling power to an electric motor as shown in this illustration:



This motor control system has a problem, though: the motor refuses to start when the "Start" pushbutton is pressed. Examine the "live" display of the ladder logic program inside this Allen-Bradley PLC to determine what the problem is, assuming an operator is continuously pressing the "Start" pushbutton as you examine the program:



Identify at least two causes that could account for all you see here.

Challenges

- Identify the symptoms resulting from the "Start" switch being failed open.
- Identify the symptoms resulting from the "Stop" switch being failed open.
- Identify the symptoms resulting from the contactor coil being failed open.

#### 5.3. DIAGNOSTIC REASONING

## 5.3.3 Troubleshooting motor control PLC from I/O indicators

Suppose we have an Allen-Bradley SLC 500 controller connected to a pair of momentary-contact pushbutton switches and contactor controlling power to an electric motor as shown in this illustration:



This motor control system has a problem, though: the motor refuses to start when the "Start" pushbutton is pressed. Closely examine the pictorial diagram (including the status LEDs on the PLC's I/O cards), then identify at least two faults that could account for the motor's refusal to start.

## Challenges

• Explain why knowledge of the PLC program is not necessary to diagnose this fault.

# 5.3.4 Turbine low-oil trip

A PLC is being used to monitor the oil pressure for a steam turbine driving an electrical generator, shutting steam off to the turbine if ever the oil pressure drops below a 10 PSI limit. The turbine's lubrication oil pump is driven by the turbine shaft itself, supplying itself with pressurized lubricating oil to keep all the turbine bearings properly lubricated and cooled:



## 5.3. DIAGNOSTIC REASONING

Another technician programmed the PLC for the start/stop function, but this program has a problem:



## Real-world I/O wiring

# PLC program



Identify what this problem is, and fix it! Hint: the oil pump is driven by the turbine, and as such cannot generate any oil pressure until the turbine begins to spin.

## Challenges

• Explain how this problem could be fixed by the addition of a *timer* instruction to the PLC program.

# 5.3.5 Motor starter diagnosis from color highlighting

# PLC #1

Identify one plausible fault to explain why this PLC-controlled motor refuses to start up when the "Start" button is pressed, given the following wiring diagram and online PLC program display shown below. Note: no one is pushing any buttons at this time.







## 5.3. DIAGNOSTIC REASONING

## PLC #2

Identify one plausible fault to explain why this PLC-controlled motor refuses to start up when the "Start" button is pressed, given the following wiring diagram and online PLC program display shown below. Note: no one is pushing any buttons at this time.



# PLC program



## PLC #3

Identify one plausible fault to explain why this PLC-controlled motor refuses to start up when the "Start" button is pressed, given the following wiring diagram and online PLC program display shown below. Note: no one is pushing any buttons at this time.



# PLC program



## Challenges

• If you did not have access to a computer to view the PLC's live status, how could you diagnose these faults?

## 5.3.6 Cannery counter diagnosis

A PLC is used to count the number of cans traveling by on a conveyor belt in a fish canning factory. An optical proximity switch detects the passage of each can, sending a discrete (on/off) signal to one of the PLC's input channels. The PLC then counts the number of pulses to determine the number of cans that have passed by:



One day the canning line operator tells you the PLC has stopped counting even though cans continue to run past the proximity switch as the conveyor belt moves. Identify what you would do to begin diagnosing this problem, justifying each step you would take.

Challenges

- Identify different areas or components within this system that could possibly be at fault, as a prelude to identifying specific diagnostic steps.
- Are there any ways you could diagnose this problem without the use of test equipment (e.g. multimeter)?
- Explain the significance of the "sourcing" and "sinking" labels on the I/O cards as well as the proximity switch.

#### 5.3. DIAGNOSTIC REASONING

## 5.3.7 Parking garage counter faults

The following Koyo CLICK PLCs are supposed to count the number of cars entering a parking garage, using a pressure-sensitive switch that the cars drive over when entering the garage. The car-count value is sent to a computer in the main office via a network cable plugged into the PLC. The parking attendant is able to reset the count to 0 at the end of his shift, using a key-switch:



## PLC #1

Suppose the counter's current value as displayed on the main office computer is stuck at 574 no matter how many more cars drive over the pressure switch and enter the garage. Explain how you would go about diagnosing the problem in this system, justifying each step you would take.

### **PLC #2**

Suppose the counter's current value as displayed on the main office computer is stuck at 1357 no matter how many more cars drive over the pressure switch and enter the garage. A voltmeter connected to the terminals of the reset switch registers 23.9 Volts DC all the time. Does this voltmeter test provide any diagnostically useful information to us?

#### PLC #3

Suppose the counter's current value as displayed on the main office computer is stuck at 822 no matter how many more cars drive over the pressure switch and enter the garage. A voltmeter connected between terminals X2 and C1 registers 0.0 Volts DC all the time. Does this voltmeter test provide any diagnostically useful information to us?

### **PLC #4**

Suppose the counter's current value as displayed on the main office computer is stuck at 0 no matter how many more cars drive over the pressure switch and enter the garage. A voltmeter connected between terminals X2 and C1 registers 25.1 Volts DC all the time. Does this voltmeter test provide any diagnostically useful information to us?

## Challenges

- For any diagnostic tests deemed useless, identify a better diagnostic test.
- For any diagnostic tests deemed useful, identify the *next* diagnostic test you would take.

### 138

#### 5.3. DIAGNOSTIC REASONING

## 5.3.8 Possible faults in a PLC/HMI pump control system

A large water pump at a wastewater treatment facility is speed-controlled by a VFD, receiving commands from a PLC/HMI control system:



This system is newly constructed, and when the operators try starting up the pump by pressing the "Pump start" icon on the touch-screen, nothing happens. A technician temporarily connects a jumper wire across the two terminals at the VFD where the control cable lands. At this, the motor starts up and runs.

Identify the likelihood of each specified fault for this circuit. Consider each fault one at a time (i.e. no coincidental faults), determining whether or not each fault is compatible with *all* measurements and symptoms in this circuit.

- Circuit breaker off
- Touch-screen panel malfunctioning
- Programming error in PLC
- Faulted power cable between VFD and motor
- Faulted power cable between breaker and VFD
- PLC output card malfunctioning
- Open control cable
- Shorted control cable
- Open data cable

Challenges

• Identify the *next* diagnostic test or measurement you would make on this system. Explain how the result(s) of this next test or measurement help further identify the location and/or nature of the fault.

140
## 5.3.9 Possible faults in a PLC/HMI package-counting system

A PLC counts packages coming by on a conveyor belt in a manufacturing facility. An optical sensor detects these packages as they travel by on the conveyor belt:



Unfortunately, something is not working correctly in this system. The HMI display continues to read a count value of zero no matter how many packages pass by the sensor switch. This very same system worked just fine three days ago, and had been working fine for one whole year before that.

Brainstorm at least five different faults that could account for this problem, and then devise a "next test" you would conduct to narrow the field of potential faults. The simpler this test (i.e. the least amount of time to conduct and the less complicated test equipment required), the better!

Challenges

• Explain how your proposed diagnostic test would either confirm or eliminate certain fault possibilities.

## 5.3.10 Diagnostic tests on a failed PLC/HMI pump control system

A large water pump at a wastewater treatment facility is speed-controlled by a VFD, receiving commands from a PLC/HMI control system:



This system is newly constructed, and when the operators try starting up the pump by pressing the "Pump start" icon on the touch-screen, nothing happens. You happen to notice a glowing "Power" indicator LED on the VFD, but none of the warning LEDs on the VFD are lit.

Assess whether each of the following diagnostic tests would be useful on this failed system:

- Measure AC Volts at VFD input
- Measure AC Volts at VFD output
- Jumper control cable terminals at VFD
- Jumper control cable terminals at PLC output card
- Force PLC output bit on
- Measure DC Volts between PLC output terminals
- Check PLC mode (Run/Terminal/Stop)

Challenges

• What, exactly, makes a diagnostic test useful?

### 5.3.11 Correcting PLC program errors

#### PLC program #1

A PLC has been programming to control the starting and stopping of a three-phase electric motor. Shown here is a partial wiring diagram and offline PLC program display for the system:



Real-world I/O wiring and tagnames



Identify the problem(s) in this PLC program, and modify it so that it will work as it should.

#### PLC program #2

A PLC has been programming to control the starting and stopping of a three-phase electric motor. Shown here is a partial wiring diagram and offline PLC program display for the system:



Identify the problem(s) in this PLC program, and modify it so that it will work as it should.

#### 5.3. DIAGNOSTIC REASONING

#### PLC program #3

A PLC has been programming to control the starting and stopping of a three-phase electric motor. The program is supposed to require that the operator press and hold the "Start" pushbutton for at least three seconds before the motor starts and runs. Shown here is a partial wiring diagram and offline PLC program display for the system:



Identify the problem(s) in this PLC program, and modify it so that it will work as it should.

#### PLC program #4

A technician needs to write a PLC program to control a water pump driven by an electric motor. This water pump will be manually started and stopped by pushbutton switches, and shut down automatically by any one of several "permissive" switches. The operating statuses of these switches are listed here:

- Start pushbutton (normally-open): open when unpressed, closed when pressed
- Stop pushbutton (normally-closed): closed when unpressed, open when pressed
- Low water level (normally-closed): closed when level is low, open when level is adequate
- Low oil pressure (normally-open): open when pressure is low, closed when pressure is adequate
- High vibration (normally-closed): closed when still, open when vibrating
- Water leak detector (normally-open): open when dry, closed when wet (leak detected)

The technician's first attempt is shown here, but it contains a serious error. Identify and correct this error:



146

#### 5.3. DIAGNOSTIC REASONING

#### PLC program #5

A technician needs to write a PLC program to control a water pump driven by an electric motor. This water pump will be manually started and stopped by pushbutton switches, and shut down automatically by any one of several "permissive" switches. The operating statuses of these switches are listed here:

- Start pushbutton (normally-open): open when unpressed, closed when pressed
- Stop pushbutton (normally-closed): closed when unpressed, open when pressed
- Low water level (normally-closed): closed when level is low, open when level is adequate
- Low oil pressure (normally-open): open when pressure is low, closed when pressure is adequate
- High vibration (normally-closed): closed when still, open when vibrating
- Water leak detector (normally-open): open when dry, closed when wet (leak detected)

The technician's first attempt is shown here, but it contains a serious error. Identify and correct this error:



#### PLC program #6

A technician needs to write a PLC program to control a water pump driven by an electric motor. This water pump will be manually started and stopped by pushbutton switches, and shut down automatically by any one of several "permissive" switches. The operating statuses of these switches are listed here:

- Start pushbutton (normally-open): open when unpressed, closed when pressed
- Stop pushbutton (normally-closed): open when unpressed, closed when pressed
- Low water level (normally-closed): closed when level is low, open when level is adequate
- Low oil pressure (normally-open): open when pressure is low, closed when pressure is adequate
- High vibration (normally-closed): closed when still, open when vibrating
- Water leak detector (normally-open): open when dry, closed when wet (leak detected)

The technician's first attempt is shown here, but it contains a serious error. Identify and correct this error:



148

#### 5.3. DIAGNOSTIC REASONING

#### PLC program #7

A technician needs to write a program for an Allen-Bradley CompactLogix PLC to control a pressure-relieving solenoid valve in a gas processing system. A pair of high-pressure control switches signals the PLC when to open the solenoid valve: one telling the PLC to open the valve after a 3-second time delay and the other (called the "high-high" switch, with a higher trip setting) telling the PLC to open the valve immediately. A pushbutton switch serves as a manual override to open the solenoid valve immediately when pressed. In all cases, the solenoid vent valve will remain open (energized) until pressure falls below the setting of a low-pressure gas switch. The operating statuses of these switches are listed here:

- Override pushbutton (normally-open): open when unpressed, closed when pressed
- Low gas pressure (normally-closed): closed when pressure is less than 10 PSI, open when pressure exceeds 10 PSI
- High gas pressure (normally-closed): closed when pressure is less than 30 PSI, open when pressure exceeds 30 PSI
- **High-high gas pressure** (normally-open): open when pressure is less than 40 PSI, closed when pressure exceeds 40 PSI

The technician's first attempt is shown here, but it contains a serious error. Identify and correct this error:



Challenges

• ???

# Chapter 6

# **Projects and Experiments**

The following project and experiment descriptions outline things you can build to help you understand circuits. With any real-world project or experiment there exists the potential for physical harm. *Electricity can be very dangerous in certain circumstances, and you should follow proper safety precautions at all times!* 

# 6.1 Recommended practices

This section outlines some recommended practices for all circuits you design and construct.

#### 6.1.1 Safety first!

Electricity, when passed through the human body, causes uncomfortable sensations and in large enough measures<sup>1</sup> will cause muscles to involuntarily contract. The overriding of your nervous system by the passage of electrical current through your body is particularly dangerous in regard to your heart, which is a vital muscle. Very large amounts of current can produce serious internal burns in addition to all the other effects.

Cardio-pulmonary resuscitation (CPR) is the standard first-aid for any victim of electrical shock. This is a very good skill to acquire if you intend to work with others on dangerous electrical circuits. You should never perform tests or work on such circuits unless someone else is present who is proficient in CPR.

As a general rule, any voltage in excess of 30 Volts poses a definitive electric shock hazard, because beyond this level human skin does not have enough resistance to safely limit current through the body. "Live" work of any kind with circuits over 30 volts should be avoided, and if unavoidable should only be done using electrically insulated tools and other protective equipment (e.g. insulating shoes and gloves). If you are unsure of the hazards, or feel unsafe at any time, stop all work and distance yourself from the circuit!

A policy I strongly recommend for students learning about electricity is to never come into electrical contact<sup>2</sup> with an energized conductor, no matter what the circuit's voltage<sup>3</sup> level! Enforcing this policy may seem ridiculous when the circuit in question is powered by a single battery smaller than the palm of your hand, but it is precisely this instilled habit which will save a person from bodily harm when working with more dangerous circuits. Experience has taught me that students who learn early on to be careless with safe circuits have a tendency to be careless later with dangerous circuits!

In addition to the electrical hazards of shock and burns, the construction of projects and running of experiments often poses other hazards such as working with hand and power tools, potential

<sup>&</sup>lt;sup>1</sup>Professor Charles Dalziel published a research paper in 1961 called "The Deleterious Effects of Electric Shock" detailing the results of electric shock experiments with both human and animal subjects. The threshold of perception for human subjects holding a conductor in their hand was in the range of 1 milliampere of current (less than this for alternating current, and generally less for female subjects than for male). Loss of muscular control was exhibited by half of Dalziel's subjects at less than 10 milliamperes alternating current. Extreme pain, difficulty breathing, and loss of all muscular control occurred for over 99% of his subjects at direct currents less than 100 milliamperes. In summary, it doesn't require much electric current to induce painful and even life-threatening effects in the human body! Your first and best protection against electric shock is maintaining an insulating barrier between your body and the circuit in question, such that current from that circuit will be unable to flow through your body.

 $<sup>^{2}</sup>$ By "electrical contact" I mean either directly touching an energized conductor with any part of your body, or indirectly touching it through a conductive tool. The only physical contact you should ever make with an energized conductor is via an electrically insulated tool, for example a screwdriver with an electrically insulated handle, or an insulated test probe for some instrument.

<sup>&</sup>lt;sup>3</sup>Another reason for consistently enforcing this policy, even on low-voltage circuits, is due to the dangers that even some low-voltage circuits harbor. A single 12 Volt automobile battery, for example, can cause a surprising amount of damage if short-circuited simply due to the high current levels (i.e. very low internal resistance) it is capable of, even though the voltage level is too low to cause a shock through the skin. Mechanics wearing metal rings, for example, are at risk from severe burns if their rings happen to short-circuit such a battery! Furthermore, even when working on circuits that are simply too low-power (low voltage and low current) to cause any bodily harm, touching them while energized can pose a threat to the circuit components themselves. In summary, it generally wise (and *always* a good habit to build) to "power down" *any* circuit before making contact between it and your body.

#### 6.1. RECOMMENDED PRACTICES

contact with high temperatures, potential chemical exposure, etc. You should never proceed with a project or experiment if you are unaware of proper tool use or lack basic protective measures (e.g. personal protective equipment such as safety glasses) against such hazards.

Some other safety-related practices should be followed as well:

- All power conductors extending outward from the project must be *firmly* strain-relieved (e.g. "cord grips" used on line power cords), so that an accidental tug or drop will not compromise circuit integrity.
- All electrical connections must be sound and appropriately made (e.g. soldered wire joints rather than twisted-and-taped; terminal blocks rather than solderless breadboards for high-current or high-voltage circuits). Use "touch-safe" terminal connections with recessed metal parts to minimize risk of accidental contact.
- Always provide overcurrent protection in any circuit you build. *Always*. This may be in the form of a fuse, a circuit breaker, and/or an electronically current-limited power supply.
- Always ensure circuit conductors are rated for more current than the overcurrent protection limit. *Always*. A fuse does no good if the wire or printed circuit board trace will "blow" before it does!
- Always bond metal enclosures to Earth ground for any line-powered circuit. *Always*. Ensuring an equipotential state between the enclosure and Earth by making the enclosure electrically common with Earth ground ensures no electric shock can occur simply by one's body bridging between the Earth and the enclosure.
- Avoid building a high-energy circuit when a low-energy circuit will suffice. For example, I always recommend beginning students power their first DC resistor circuits using small batteries rather than with line-powered DC power supplies. The intrinsic energy limitations of a dry-cell battery make accidents highly unlikely.
- Use line power receptacles that are GFCI (Ground Fault Current Interrupting) to help avoid electric shock from making accidental contact with a "hot" line conductor.
- Always wear eye protection when working with tools or live systems having the potential to eject material into the air. Examples of such activities include soldering, drilling, grinding, cutting, wire stripping, working on or near energized circuits, etc.
- Always use a step-stool or stepladder to reach high places. Never stand on something not designed to support a human load.
- When in doubt, *ask an expert*. If anything even seems remotely unsafe to you, do not proceed without consulting a trusted person fully knowledgeable in electrical safety.

#### 6.1.2 Other helpful tips

Experience has shown the following practices to be very helpful, especially when students make their own component selections, to ensure the circuits will be well-behaved:

- Avoid resistor values less than 1 k $\Omega$  or greater than 100 k $\Omega$ , unless such values are definitely necessary<sup>4</sup>. Resistances below 1 k $\Omega$  may draw excessive current if directly connected to a voltage source of significant magnitude, and may also complicate the task of accurately measuring current since any ammeter's non-zero resistance inserted in series with a low-value circuit resistor will significantly alter the total resistance and thereby skew the measurement. Resistances above 100 k $\Omega$  may complicate the task of measuring voltage since any voltmeter's finite resistance connected in parallel with a high-value circuit resistor will significantly alter the total resistance and thereby skew the measurement. Similarly, AC circuit impedance values should be between 1 k $\Omega$  and 100 k $\Omega$ , and for all the same reasons.
- Ensure all electrical connections are low-resistance and physically rugged. For this reason, one should avoid *compression splices* (e.g. "butt" connectors), solderless breadboards<sup>5</sup>, and wires that are simply twisted together.
- Build your circuit with **testing** in mind. For example, provide convenient connection points for test equipment (e.g. multimeters, oscilloscopes, signal generators, logic probes).
- Design permanent projects with **maintenance** in mind. The more convenient you make maintenance tasks, the more likely they will get done.
- Always document and save your work. Circuits lacking schematic diagrams are more difficult to troubleshoot than documented circuits. Similarly, circuit construction is simpler when a schematic diagram precedes construction. Experimental results are easier to interpret when comprehensively recorded. Consider modern videorecording technology for this purpose where appropriate.
- **Record your steps** when troubleshooting. **Talk to yourself** when solving problems. These simple steps clarify thought and simplify identification of errors.

<sup>&</sup>lt;sup>4</sup>An example of a necessary resistor value much less than 1 k $\Omega$  is a *shunt resistor* used to produce a small voltage drop for the purpose of sensing current in a circuit. Such shunt resistors must be low-value in order not to impose an undue load on the rest of the circuit. An example of a necessary resistor value much greater than 100 k $\Omega$  is an electrostatic *drain resistor* used to dissipate stored electric charges from body capacitance for the sake of preventing damage to sensitive semiconductor components, while also preventing a path for current that could be dangerous to the person (i.e. shock).

 $<sup>^{5}</sup>$ Admittedly, solderless breadboards are very useful for constructing complex electronic circuits with many components, especially DIP-style integrated circuits (ICs), but they tend to give trouble with connection integrity after frequent use. An alternative for projects using low counts of ICs is to solder IC sockets into prototype printed circuit boards (PCBs) and run wires from the soldered pins of the IC sockets to terminal blocks where reliable temporary connections may be made.

#### 6.1.3 Terminal blocks for circuit construction

Terminal blocks are the standard means for making electric circuit connections in industrial systems. They are also quite useful as a learning tool, and so I highly recommend their use in lieu of solderless breadboards<sup>6</sup>. Terminal blocks provide highly reliable connections capable of withstanding significant voltage and current magnitudes, and they force the builder to think very carefully about component layout which is an important mental practice. Terminal blocks that mount on standard 35 mm DIN rail<sup>7</sup> are made in a wide range of types and sizes, some with built-in disconnecting switches, some with built-in components such as rectifying diodes and fuseholders, all of which facilitate practical circuit construction.

I recommend every student of electricity build their own terminal block array for use in constructing experimental circuits, consisting of several terminal blocks where each block has at least 4 connection points all electrically common to each other<sup>8</sup> and at least one terminal block that is a fuse holder for overcurrent protection. A pair of anchoring blocks hold all terminal blocks securely on the DIN rail, preventing them from sliding off the rail. Each of the terminals should bear a number, starting from 0. An example is shown in the following photograph and illustration:



Screwless terminal blocks (using internal spring clips to clamp wire and component lead ends) are preferred over screw-based terminal blocks, as they reduce assembly and disassembly time, and also minimize repetitive wrist stress from twisting screwdrivers. Some screwless terminal blocks require the use of a special tool to release the spring clip, while others provide buttons<sup>9</sup> for this task which may be pressed using the tip of any suitable tool.

<sup>&</sup>lt;sup>6</sup>Solderless breadboard are preferable for complicated electronic circuits with multiple integrated "chip" components, but for simpler circuits I find terminal blocks much more practical. An alternative to solderless breadboards for "chip" circuits is to solder chip sockets onto a PCB and then use wires to connect the socket pins to terminal blocks. This also accommodates *surface-mount* components, which solderless breadboards do not.

<sup>&</sup>lt;sup>7</sup>DIN rail is a metal rail designed to serve as a mounting point for a wide range of electrical and electronic devices such as terminal blocks, fuses, circuit breakers, relay sockets, power supplies, data acquisition hardware, etc.

<sup>&</sup>lt;sup>8</sup>Sometimes referred to as *equipotential, same-potential,* or *potential distribution* terminal blocks.

 $<sup>^{9}</sup>$ The small orange-colored squares seen in the above photograph are buttons for this purpose, and may be actuated by pressing with any tool of suitable size.

The following example shows how such a terminal block array might be used to construct a series-parallel resistor circuit consisting of four resistors and a battery:



Numbering on the terminal blocks provides a very natural translation to SPICE<sup>10</sup> netlists, where component connections are identified by terminal number:

```
* Series-parallel resistor circuit
v1 1 0 dc 6
r1 2 5 7100
r2 5 8 2200
r3 2 8 3300
r4 8 11 4700
rjmp1 1 2 0.01
rjmp2 0 11 0.01
```

#### .op .end

Note the use of "jumper" resistances rjmp1 and rjmp2 to describe the wire connections between terminals 1 and 2 and between terminals 0 and 11, respectively. Being resistances, SPICE requires a resistance value for each, and here we see they have both been set to an arbitrarily low value of 0.01 Ohm realistic for short pieces of wire.

Listing all components and wires along with their numbered terminals happens to be a useful documentation method for any circuit built on terminal blocks, independent of SPICE. Such a "wiring sequence" may be thought of as a *non-graphical description* of an electric circuit, and is exceptionally easy to follow.

 $<sup>^{10}</sup>$ SPICE is computer software designed to analyze electrical and electronic circuits. Circuits are described for the computer in the form of *netlists* which are text files listing each component type, connection node numbers, and component values.

#### 6.1. RECOMMENDED PRACTICES

An example of a more elaborate terminal block array is shown in the following photograph, with terminal blocks and "ice-cube" style electromechanical relays mounted to DIN rail, which is turn mounted to a perforated subpanel<sup>11</sup>. This "terminal block board" hosts an array of thirty five undedicated terminal block sections, four SPDT toggle switches, four DPDT "ice-cube" relays, a step-down control power transformer, bridge rectifier and filtering capacitor, and several fuses for overcurrent protection:



Four plastic-bottomed "feet" support the subpanel above the benchtop surface, and an unused section of DIN rail stands ready to accept other components. Safety features include electrical bonding of the AC line power cord's ground to the metal subpanel (and all metal DIN rails), mechanical strain relief for the power cord to isolate any cord tension from wire connections, clear plastic finger guards covering the transformer's screw terminals, as well as fused overcurrent protection for the 120 Volt AC line power and the transformer's 12 Volt AC output. The perforated holes happen to be on  $\frac{1}{4}$  inch centers with a diameter suitable for tapping with 6-32 machine screw threads, their presence making it very easy to attach other sections of DIN rail, printed circuit boards, or specialized electrical components directly to the grounded metal subpanel. Such a "terminal block board" is an inexpensive<sup>12</sup> yet highly flexible means to construct physically robust circuits using industrial wiring practices.

<sup>&</sup>lt;sup>11</sup>An electrical *subpanel* is a thin metal plate intended for mounting inside an electrical enclosure. Components are attached to the subpanel, and the subpanel in turn bolts inside the enclosure. Subpanels allow circuit construction outside the confines of the enclosure, which speeds assembly. In this particular usage there is no enclosure, as the subpanel is intended to be used as an open platform for the convenient construction of circuits on a benchtop by students. In essence, this is a modern version of the traditional *breadboard* which was literally a wooden board such as might be used for cutting loaves of bread, but which early electrical and electronic hobbyists used as platforms for the construction of circuits.

 $<sup>^{12}</sup>$ At the time of this writing (2019) the cost to build this board is approximately \$250 US dollars.

#### 6.1.4 Conducting experiments

An *experiment* is an exploratory act, a test performed for the purpose of assessing some proposition or principle. Experiments are the foundation of the *scientific method*, a process by which careful observation helps guard against errors of speculation. All good experiments begin with an *hypothesis*, defined by the American Heritage Dictionary of the English Language as:

An assertion subject to verification or proof, as (a) A proposition stated as a basis for argument or reasoning. (b) A premise from which a conclusion is drawn. (c) A conjecture that accounts, within a theory or ideational framework, for a set of facts and that can be used as a basis for further investigation.

Stated plainly, an hypothesis is an *educated guess* about cause and effect. The correctness of this initial guess matters little, because any well-designed experiment will reveal the truth of the matter. In fact, *incorrect* hypotheses are often the most valuable because the experiments they engender lead us to surprising discoveries. One of the beautiful aspects of science is that it is more focused on the process of *learning* than about the status of *being correct*<sup>13</sup>. In order for an hypothesis to be valid, it must be testable<sup>14</sup>, which means it must be a claim possible to refute given the right data. Hypotheses impossible to critique are useless.

Once an hypothesis has been formulated, an experiment must be designed to test that hypothesis. A well-designed experiment requires careful regulation of all relevant variables, both for personal safety and for prompting the hypothesized results. If the effects of one particular variable are to be tested, the experiment must be run multiple times with different values of (only) that particular variable. The experiment set up with the "baseline" variable set is called the *control*, while the experiment set up with different value(s) is called the *test* or *experimental*.

For some hypotheses a viable alternative to a physical experiment is a *computer-simulated* experiment or even a *thought experiment*. Simulations performed on a computer test the hypothesis against the physical laws encoded within the computer simulation software, and are particularly useful for students learning new principles for which simulation software is readily available<sup>15</sup>.

 $<sup>^{13}</sup>$ Science is more about clarifying our view of the universe through a systematic process of error detection than it is about proving oneself to be right. Some scient *ists* may happen to have large egos – and this may have more to do with the ways in which large-scale scientific research is *funded* than anything else – but scientific *method* itself is devoid of ego, and if embraced as a practical philosophy is quite an effective stimulant for humility. Within the education system, scientific method is particularly valuable for helping students break free of the crippling fear of *being wrong*. So much emphasis is placed in formal education on assessing correct retention of facts that many students are fearful of saying or doing anything that might be perceived as a mistake, and of course making mistakes (i.e. having one's hypotheses disproven by experiment) is an indispensable tool for learning. Introducing science in the classroom – *real* science characterized by individuals forming actual hypotheses and testing those hypotheses by experiment – helps students become self-directed learners.

 $<sup>^{14}</sup>$ This is the principle of *falsifiability*: that a scientific statement has value only insofar as it is liable to disproof given the requisite experimental evidence. Any claim that is unfalsifiable – that is, a claim which can *never* be disproven by any evidence whatsoever – could be completely wrong and we could never know it.

<sup>&</sup>lt;sup>15</sup>A very pertinent example of this is learning how to analyze electric circuits using simulation software such as SPICE. A typical experimental cycle would proceed as follows: (1) Find or invent a circuit to analyze; (2) Apply your analytical knowledge to that circuit, predicting all voltages, currents, powers, etc. relevant to the concepts you are striving to master; (3) Run a simulation on that circuit, collecting "data" from the computer when complete; (4) Evaluate whether or not your hypotheses (i.e. predicted voltages, currents, etc.) agree with the computer-generated results; (5) If so, your analyses are (provisionally) correct – if not, examine your analyses and the computer simulation again to determine the source of error; (6) Repeat this process as many times as necessary until you achieve mastery.

Thought experiments are useful for detecting inconsistencies within your own understanding of some subject, rather than testing your understanding against physical reality.

Here are some general guidelines for conducting experiments:

- The clearer and more specific the hypothesis, the better. Vague or unfalsifiable hypotheses are useless because they will fit *any* experimental results, and therefore the experiment cannot teach you anything about the hypothesis.
- Collect as much data (i.e. information, measurements, sensory experiences) generated by an experiment as is practical. This includes the time and date of the experiment, too!
- *Never* discard or modify data gathered from an experiment. If you have reason to believe the data is unreliable, write notes to that effect, but never throw away data just because you think it is untrustworthy. It is quite possible that even "bad" data holds useful information, and that someone else may be able to uncover its value even if you do not.
- Prioritize *quantitative* data over *qualitative* data wherever practical. Quantitative data is more specific than qualitative, less prone to subjective interpretation on the part of the experimenter, and amenable to an arsenal of analytical methods (e.g. statistics).
- Guard against your own bias(es) by making your experimental results available to others. This allows other people to scrutinize your experimental design and collected data, for the purpose of detecting and correcting errors you may have missed. Document your experiment such that others may independently replicate it.
- Always be looking for sources of error. No physical measurement is perfect, and so it is impossible to achieve *exact* values for any variable. Quantify the amount of uncertainty (i.e. the "tolerance" of errors) whenever possible, and be sure your hypothesis does not depend on precision better than this!
- Always remember that scientific confirmation is provisional no number of "successful" experiments will prove an hypothesis true for all time, but a single experiment can disprove it. Put into simpler terms, *truth is elusive but error is within reach*.
- Remember that scientific method is about *learning*, first and foremost. An unfortunate consequence of scientific triumph in modern society is that science is often viewed by non-practitioners as an unerring source of truth, when in fact science is an ongoing process of challenging existing ideas to probe for errors and oversights. This is why it is perfectly acceptable to have a failed hypothesis, and why the only truly failed experiment is one where nothing was learned.

The following is an example of a well-planned and executed experiment, in this case a physical experiment demonstrating Ohm's Law.

Planning Time/Date = 09:30 on 12 February 2019

HYPOTHESIS: the current through any resistor should be exactly proportional to the voltage impressed across it.

PROCEDURE: connect a resistor rated 1 k Ohm and 1/4 Watt to a variable-voltage DC power supply. Use an ammeter in series to measure resistor current and a voltmeter in parallel to measure resistor voltage.

RISKS AND MITIGATION: excessive power dissipation may harm the resistor and/ or pose a burn hazard, while excessive voltage poses an electric shock hazard. 30 Volts is a safe maximum voltage for laboratory practices, and according to Joule's Law a 1000 Ohm resistor will dissipate 0.25 Watts at 15.81 Volts  $(P = V^2 / R)$ , so I will remain below 15 Volts just to be safe.

Experiment Time/Date = 10:15 on 12 February 2019

DATA COLLECTED:

(Voltage)		(Current)	(Voltage)		(Current)
0.000 V	=	0.000 mA	8.100	=	7.812 mA
2.700 V	=	2.603 mA	10.00 V	=	9.643 mA
5.400 V	=	5.206 mA	14.00 V	=	13.49 mA

Analysis Time/Date = 10:57 on 12 February 2019

ANALYSIS: current definitely increases with voltage, and although I expected exactly one milliAmpere per Volt the actual current was usually less than that. The voltage/current ratios ranged from a low of 1036.87 (at 8.1 Volts) to a high of 1037.81 (at 14 Volts), but this represents a variance of only -0.0365% to +0.0541% from the average, indicating a very consistent proportionality -- results consistent with Ohm's Law.

ERROR SOURCES: one major source of error is the resistor's value itself. I did not measure it, but simply assumed color bands of brown-black-red meant exactly 1000 Ohms. Based on the data I think the true resistance is closer to 1037 Ohms. Another possible explanation is multimeter calibration error. However, neither explains the small positive and negative variances from the average. This might be due to electrical noise, a good test being to repeat the same experiment to see if the variances are the same or different. Noise should generate slightly different results every time.

#### 6.1. RECOMMENDED PRACTICES

The following is an example of a well-planned and executed *virtual* experiment, in this case demonstrating Ohm's Law using a computer (SPICE) simulation.

```
Planning Time/Date = 12:32 on 14 February 2019
HYPOTHESIS: for any given resistor, the current through that resistor should be
exactly proportional to the voltage impressed across it.
PROCEDURE: write a SPICE netlist with a single DC voltage source and single
1000 Ohm resistor, then use NGSPICE version 26 to perform a "sweep" analysis
from O Volts to 25 Volts in 5 Volt increments.
   * SPICE circuit
   v1 1 0 dc
   r1 1 0 1000
   .dc v1 0 25 5
   .print dc v(1) i(v1)
   .end
RISKS AND MITIGATION: none.
DATA COLLECTED:
      DC transfer characteristic Thu Feb 14 13:05:08 2019
   _____
   Index v-sweep v(1)
                                      v1#branch
    _____
       0.000000e+00 0.00000e+00 0.00000e+00
5.000000e+00 5.00000e+00 5.00000e+00
   0
          5.000000e+00 5.000000e+00 -5.00000e-03
   1
   2
         1.000000e+01 1.000000e+01 -1.00000e-02
   3
         1.500000e+01 1.500000e+01 -1.50000e-02
          2.000000e+01 2.000000e+01 -2.00000e-02
   4
   5
          2.500000e+01 2.500000e+01
                                       -2.50000e-02
Analysis Time/Date = 13:06 on 14 February 2019
ANALYSIS: perfect agreement between data and hypothesis -- current is precisely
1/1000 of the applied voltage for all values. Anything other than perfect
agreement would have probably meant my netlist was incorrect. The negative
current values surprised me, but it seems this is just how SPICE interprets
normal current through a DC voltage source.
ERROR SOURCES: none.
```

As gratuitous as it may seem to perform experiments on a physical law as well-established as Ohm's Law, even the examples listed previously demonstrate opportunity for real learning. In the physical experiment example, the student should identify and explain why their data does not perfectly agree with the hypothesis, and this leads them naturally to consider sources of error. In the computer-simulated experiment, the student is struck by SPICE's convention of denoting regular current through a DC voltage source as being *negative* in sign, and this is also useful knowledge for future simulations. Scientific experiments are most interesting when things *do not* go as planned!

Aside from verifying well-established physical laws, simple experiments are extremely useful as educational tools for a wide range of purposes, including:

- Component familiarization (e.g. Which terminals of this switch connect to the NO versus NC contacts?)
- System testing (e.g. How heavy of a load can my AC-DC power supply source before the semiconductor components reach their thermal limits?)
- Learning programming languages (e.g. Let's try to set up an "up" counter function in this PLC!)

Above all, the priority here is to inculcate the habit of hypothesizing, running experiments, and analyzing the results. This experimental cycle not only serves as an excellent method for self-directed learning, but it also works exceptionally well for troubleshooting faults in complex systems, and for these reasons should be a part of every technician's and every engineer's education.

#### 6.1.5 Constructing projects

Designing, constructing, and testing projects is a very effective means of practical education. Within a formal educational setting, projects are generally chosen (or at least vetted) by an instructor to ensure they may be reasonably completed within the allotted time of a course or program of study, and that they sufficiently challenge the student to learn certain important principles. In a self-directed environment, projects are just as useful as a learning tool but there is some risk of unwittingly choosing a project beyond one's abilities, which can lead to frustration.

Here are some general guidelines for managing projects:

- Define your goal(s) before beginning a project: what do you wish to achieve in building it? What, exactly, should the completed project *do*?
- Analyze your project prior to construction. Document it in appropriate forms (e.g. schematic diagrams), predict its functionality, anticipate all associated risks. In other words, *plan ahead*.
- Set a reasonable budget for your project, and stay within it.
- Identify any deadlines, and set reasonable goals to meet those deadlines.
- Beware of *scope creep*: the tendency to modify the project's goals before it is complete.
- Document your progress! An easy way to do this is to use photography or videography: take photos and/or videos of your project as it progresses. Document failures as well as successes, because both are equally valuable from the perspective of learning.

## 6.2 Experiment: contact and coil demonstration program

Conduct an experiment to explore the behavior of *contact* and *coil* Ladder Diagram instructions in a functioning PLC. Your experiment should test the behavior of all basic contact and coil instructions offered by the particular model of PLC available to you, and do so as simply as possible. You are strongly recommended *not* to write a PLC program that might serve some practical purpose, but rather to write the simplest possible version<sup>16</sup> of a proof-of-concept program merely showcasing how each instruction works.

An acceptable PLC program for this experiment must meet these three criteria:

- **Simple** nothing "extra" included in the program to detract from the fundamental behavior of the instruction(s) being explored.
- **Complete** nothing missing from the program relevant to the fundamental behavior of the instruction(s) being explored.
- Clearly documented every rung clearly commented in your own words.

The practical purpose of this experiment is to serve as a reference for your future self modeling how each of your PLC's instructions work. This is why the experiment should not be a practical program, because programs written to perform some practical purpose become complicated by the details of that purpose.

Your analysis of each instruction's function should be written as *comments* in the Ladder Diagram program.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

<sup>&</sup>lt;sup>16</sup>As few rungs of code as possible, with as few instructions on each rung as possible.

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\boxed{\checkmark}$  Identify any uncontrolled sources of error in the experiment.

• <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\square$  Write an analysis of experimental results and lessons learned.

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- Where in the PLC's memory are the single-bit registers (e.g. input registers, output registers, and internal bit registers) located? What symbol(s) are used to address each one?
- What happens when two contact instructions are linked to the same bit address in the PLC's memory? Do these contact instructions operated differently, or identically?
- Does your PLC offer a special type of contact or other bit-level instruction to detect the *transition* of a bit from one state to another? If so, how is this instruction used?
- What happens when two coil instructions are linked to the same bit address in the PLC's memory, but driven to different states (e.g. one "energized" and the other "de-energized")?
- Experiment with using the *force* utility in your PLC to force certain bits to fixed values regardless of program operation. How will the operation of your program be affected if a particular input bit is forced? How will the operation of your program be affected if a particular output bit is forced? How can you tell from the live program display that bits have been forced to fixed values?

# 6.3 Experiment: PLC implementation of basic logic functions

Conduct an experiment demonstrating how a circuit using toggle switches (inputs) and a PLC may be used to implement multiple basic logic functions (e.g. AND, OR, NAND, NOR, NOT, XOR, XNOR) all operating off of the same inputs. Each logic function's output should be represented by its own discrete output on the PLC.

### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\boxed{\checkmark}$  Identify any uncontrolled sources of error in the experiment.

• <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

- How would you program logic functions having more than two inputs?
- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- Write a different PLC program to implement the exact same logic functions.

# 6.4 Experiment: PLC implementation of an arbitrary truth table

Conduct an experiment demonstrating how a circuit using toggle switches (inputs) and a PLC may be used to implement an arbitrary truth table for a four-input combinational logic function. A truth table template is given here for your use, to arbitrarily write "1" and "0" states in the output column:

Α	В	С	D	Output
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\boxed{\checkmark}$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

• <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

#### • <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

#### • <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

- Identify a truth table function possible to implement with no PLC at all.
- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- It is possible to implement your chosen logic function using *only* toggle switches and no PLC?
- Write a different PLC program to implement the exact same logic function.

# 6.5 Experiment: PLC-controlled motor starter

Devise and execute an experiment to control the starting and stopping of either an AC or a DC motor using a PLC to perform the latching function and two momentary-contact switches to signal "Start" and "Stop". The "Start" switch should be wired normally-open, and the "Stop" switch normally-closed, so that any open switch wiring faults favor a stopped motor rather than a running motor.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\square$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

#### • <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

- <u>After all experimental re-runs:</u>
  - $\checkmark$  Save all data for future reference.
  - $\checkmark$  Write an analysis of experimental results and lessons learned.

#### Challenges

• Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!

#### 6.5. EXPERIMENT: PLC-CONTROLLED MOTOR STARTER

- Identify some alternative means for interposing between the PLC and the electric motor, since it is unlikely you will find a PLC with a discrete output channel rated to directly switch motor current.
- Identify some alternative programming strategies for implementing the necessary latching function.

# 6.6 Experiment: counter demonstration program

Conduct an experiment to explore the behavior of *counter* Ladder Diagram instructions in a functioning PLC. Your experiment should test the behavior of all basic counter instructions offered by the particular model of PLC available to you, and do so as simply as possible. You are strongly recommended *not* to write a PLC program that might serve some practical purpose, but rather to write the simplest possible version<sup>17</sup> of a proof-of-concept program merely showcasing how each instruction works.

An acceptable PLC program for this experiment must meet these three criteria:

- **Simple** nothing "extra" included in the program to detract from the fundamental behavior of the instruction(s) being explored.
- **Complete** nothing missing from the program relevant to the fundamental behavior of the instruction(s) being explored.
- Clearly documented every rung clearly commented in your own words.

The practical purpose of this experiment is to serve as a reference for your future self modeling how each of your PLC's instructions work. This is why the experiment should not be a practical program, because programs written to perform some practical purpose become complicated by the details of that purpose.

Your analysis of each instruction's function should be written as *comments* in the Ladder Diagram program.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

 $<sup>^{17}\</sup>mathrm{As}$  few rungs of code as possible, with as few instructions on each rung as possible.

#### 6.6. EXPERIMENT: COUNTER DEMONSTRATION PROGRAM

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

#### • <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- How can you make a single counter both *increment* (count up) and *decrement* (count down)?
- How many bits are used by your PLC in each counter instruction? How can you tell?
- Can you "force" a counter to some accumulator value in the same way you can force a discrete bit to a certain value?
- Is it possible to "pre-load" a counter to some non-zero value at the command of a single bit, such as a pushbutton switch being pressed?

# 6.7 Experiment: timer demonstration program

Conduct an experiment to explore the behavior of *timer* Ladder Diagram instructions in a functioning PLC. Your experiment should test the behavior of all basic timer instructions offered by the particular model of PLC available to you, and do so as simply as possible. You are strongly recommended *not* to write a PLC program that might serve some practical purpose, but rather to write the simplest possible version<sup>18</sup> of a proof-of-concept program merely showcasing how each instruction works.

An acceptable PLC program for this experiment must meet these three criteria:

- **Simple** nothing "extra" included in the program to detract from the fundamental behavior of the instruction(s) being explored.
- **Complete** nothing missing from the program relevant to the fundamental behavior of the instruction(s) being explored.
- Clearly documented every rung clearly commented in your own words.

The practical purpose of this experiment is to serve as a reference for your future self modeling how each of your PLC's instructions work. This is why the experiment should not be a practical program, because programs written to perform some practical purpose become complicated by the details of that purpose.

Your analysis of each instruction's function should be written as *comments* in the Ladder Diagram program.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

 $<sup>^{18}\</sup>mathrm{As}$  few rungs of code as possible, with as few instructions on each rung as possible.

#### 6.7. EXPERIMENT: TIMER DEMONSTRATION PROGRAM

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

#### • <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- Can a timer instruction be made to count backwards?
- How many bits are used by your PLC in each timer instruction? How can you tell?
- Can you "force" a timer to some accumulator value in the same way you can force a discrete bit to a certain value?
- Is it possible to "pre-load" a timer to some non-zero value at the command of a single bit, such as a pushbutton switch being pressed?

# 6.8 Experiment: PLC-controlled inrush-limiting motor starter

Devise and execute an experiment to control the starting and stopping of either an AC or a DC motor using a PLC to perform the latching function and sequencing of multiple contactors to limit the motor's inrush current. Two momentary-contact switches will signal "Start" and "Stop".

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\lfloor \checkmark \rfloor$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

• <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- Identify some alternative means for interposing between the PLC and the electric motor, since it is unlikely you will find a PLC with a discrete output channel rated to directly switch motor current.

# 6.8. EXPERIMENT: PLC-CONTROLLED INRUSH-LIMITING MOTOR STARTER

• Identify some alternative programming strategies for implementing the necessary latching function.

# 6.9 Experiment: HMI display of PLC bits and words

Devise and execute an experiment configuring a Human-Machine Interface (HMI) unit to read data bits and data words inside of a PLC. Recommended bits and words to read include those associated with counter and timer instructions, so that the HMI display will show both accumulated values and I/O bits for these PLC instructions while the PLC is running.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\boxed{\checkmark}$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\boxed{\checkmark}$  Identify any uncontrolled sources of error in the experiment.

#### • <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\square$  Write an analysis of experimental results and lessons learned.

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- Identify some of the different digital data types (e.g. signed integer, unsigned integer, floatingpoint, Boolean) readable by the HMI from the PLC, and the word size used by each PLC instruction.
### 6.10 Experiment: arithmetic demonstration program

Conduct an experiment to explore the behavior of *arithmetic* Ladder Diagram instructions in a functioning PLC. Your experiment should test the behavior of all basic arithmetic operations offered by the particular model of PLC available to you, and do so as simply as possible. You are strongly recommended *not* to write a PLC program that might serve some practical purpose, but rather to write the simplest possible version<sup>19</sup> of a proof-of-concept program merely showcasing how each instruction works.

An acceptable PLC program for this experiment must meet these three criteria:

- **Simple** nothing "extra" included in the program to detract from the fundamental behavior of the instruction(s) being explored.
- **Complete** nothing missing from the program relevant to the fundamental behavior of the instruction(s) being explored.
- Clearly documented every rung clearly commented in your own words.

The practical purpose of this experiment is to serve as a reference for your future self modeling how each of your PLC's instructions work. This is why the experiment should not be a practical program, because programs written to perform some practical purpose become complicated by the details of that purpose.

Your analysis of each instruction's function should be written as *comments* in the Ladder Diagram program.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

<sup>&</sup>lt;sup>19</sup>As few rungs of code as possible, with as few instructions on each rung as possible.

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

• <u>After all experimental re-runs:</u>

 $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

### Challenges

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- What happens when you instruct an arithmetic operation in the PLC to do something mathematically undefined, such as dividing by zero?
- Do the PLC's arithmetic instructions operate on integer values, floating-point values, or both?

### 6.11 Experiment: data transfer demonstration program

Conduct an experiment to explore the behavior of *data transfer* (i.e. communication/networking) Ladder Diagram instructions in a functioning PLC. Your experiment should test the behavior of all basic data transfer instructions offered by the particular model of PLC available to you, and do so as simply as possible. You are strongly recommended *not* to write a PLC program that might serve some practical purpose, but rather to write the simplest possible version<sup>20</sup> of a proof-of-concept program merely showcasing how each instruction works.

An acceptable PLC program for this experiment must meet these three criteria:

- **Simple** nothing "extra" included in the program to detract from the fundamental behavior of the instruction(s) being explored.
- **Complete** nothing missing from the program relevant to the fundamental behavior of the instruction(s) being explored.
- Clearly documented every rung clearly commented in your own words.

The practical purpose of this experiment is to serve as a reference for your future self modeling how each of your PLC's instructions work. This is why the experiment should not be a practical program, because programs written to perform some practical purpose become complicated by the details of that purpose.

Your analysis of each instruction's function should be written as *comments* in the Ladder Diagram program.

#### **EXPERIMENT CHECKLIST:**

• <u>Prior to experimentation:</u>

 $\checkmark$  Write an hypothesis (i.e. a detailed description of what you expect will happen) unambiguous enough that it could be disproven given the right data.

 $\checkmark$  Write a procedure to test the hypothesis, complete with adequate controls and documentation (e.g. schematic diagrams, programming code).

 $\checkmark$  Identify any risks (e.g. shock hazard, component damage) and write a mitigation plan based on best practices and component ratings.

#### • <u>During experimentation:</u>

 $\checkmark$  Safe practices followed at all times (e.g. no contact with energized circuit).

 $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

 $\checkmark$  All data collected, ideally quantitative with full precision (i.e. no rounding).

 $<sup>^{20}</sup>$ As few rungs of code as possible, with as few instructions on each rung as possible.

• <u>After each experimental run:</u>

 $\checkmark$  If the results fail to match the hypothesis, identify the error(s), correct the hypothesis and/or revise the procedure, and re-run the experiment.

 $\checkmark$  Identify any uncontrolled sources of error in the experiment.

- <u>After all experimental re-runs:</u>
  - $\checkmark$  Save all data for future reference.

 $\checkmark$  Write an analysis of experimental results and lessons learned.

#### Challenges

- Science is an *iterative* process, and for this reason is never complete. Following the results of your experiment, what would you propose for your *next* hypothesis and *next* experimental procedure? Hint: if your experiment produced any unexpected results, exploring those unexpected results is often a very good basis for the next experiment!
- How are the instructions configured or selected to either send or receive data?
- How the communicating PLCs addressed on the interconnecting network, if at all.
- How is the data communication triggered? In other words, what signals the starting time of each message?
- How are communication errors signaled within the PLC program?
- What happens when you unplug the communication cable(s) during data communication?

### 6.12 Project: PLC-controlled system

Wire and program a PLC to automatically monitor and/or control some small system. Ideas include, but are not limited to:

- Control the starting and stopping of an electric motor, using hand switches for Start and Stop initiation, with the PLC performing such functions as latching, counting the number of start/stop cycles, calculating total motor run time, flagging maintenance alerts after a certain amount of run time is reached, etc.
- Automatic starting and stopping of an air compressor to maintain relatively constant air pressure in a vessel.
- Retrofit the controls on a consumer-grade appliance, removing the factory-original control system and replacing it with a PLC. Appliance suggestions include a bread-making machine, a clothes-washing machine, a coffee making machine, a food processor, a convection oven, a pressure cooker, etc.
- Build a security alarm system for a home, sensing door and window statuses, and providing a means for only authorized people to enter the room.
- Create an alarm clock complete with audio and visual alert capabilities. Provide a means for any user to calibrate the clock against a known time standard and also be able to set wake-up times without having to access the PLC coding.
- Build a weather station monitoring temperature, wind speed, wind direction, etc. Provide alarms for cold-weather and hot-weather conditions, average daily temperature, etc.
- Build a solar tracker, automatically aiming a solar collector (photovoltaic panel, or concentrating mirror) directly at the sun using servo motors.
- Build a miniature traffic light system for a set of intersections, sequencing green/amber/red traffic lights, detecting cars on the road (using proximity switches), etc.
- Build a robot capable of navigating around a room, sensing and avoiding obstacles.

#### **PROJECT CHECKLIST:**

- <u>Prior to construction:</u>
  - $\boxed{\checkmark}$  Prototype diagram(s) and description of project scope.
  - $\checkmark$  Risk assessment/mitigation plan.
  - $\checkmark$  Timeline and action plan.
- <u>During construction:</u>
  - $\boxed{\checkmark}$  Safe work habits (e.g. no contact made with energized circuit at any time).
  - $\checkmark$  Correct equipment usage according to manufacturer's recommendations.

- $\checkmark$  Timeline and action plan amended as necessary.
- Maintain the originally-planned project scope (i.e. avoid adding features!).
- <u>After completion:</u>
  - $\checkmark$  All functions tested against original plan.
  - $\checkmark$  Full, accurate, and appropriate documentation of all project details.
  - $\checkmark$  Complete bill of materials.
  - $\checkmark$  Written summary of lessons learned.

### Challenges

- ???.
- ???.
- ???.

# Appendix A

# **Problem-Solving Strategies**

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- <u>Study principles, not procedures.</u> Don't be satisfied with merely knowing how to compute solutions learn *why* those solutions work.
- <u>Identify</u> what it is you need to solve, <u>identify</u> all relevant data, <u>identify</u> all units of measurement, <u>identify</u> any general principles or formulae linking the given information to the solution, and then <u>identify</u> any "missing pieces" to a solution. <u>Annotate</u> all diagrams with this data.
- <u>Sketch a diagram</u> to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- <u>Perform "thought experiments"</u> to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- <u>Simplify the problem</u> until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- <u>Check for exceptions</u> to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- <u>Work "backward"</u> from a hypothetical solution to a new set of given conditions.
- <u>Add quantities</u> to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- <u>Sketch graphs</u> illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- <u>Treat quantitative problems as qualitative</u> in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- <u>Consider limiting cases.</u> This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- <u>Check your work.</u> This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

# Appendix B

# Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment<sup>1</sup> where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic<sup>2</sup> dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity<sup>3</sup> through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

<sup>&</sup>lt;sup>1</sup>In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

<sup>&</sup>lt;sup>2</sup>Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

<sup>&</sup>lt;sup>3</sup>This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, the fundamental goal of education is for each student to learn to think clearly and independently. This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.
- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.
- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers the goal is to practice the articulation and defense of one's own reasoning.
- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.
- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.
- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied<sup>4</sup> effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge<sup>5</sup> one another.

To high standards of education,

Tony R. Kuphaldt

 $<sup>^{4}</sup>$ As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

 $<sup>^{5}</sup>$ Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

# Appendix C Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The GNU/Linux computer operating system

There is so much to be said about Linus Torvalds' Linux and Richard Stallman's GNU project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of Linux back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient Unix applications and scripting languages (e.g. shell scripts, Makefiles, sed, awk) developed over many decades. Linux not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's Vim text editor

Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer Vim because it operates very similarly to vi which is ubiquitous on Unix/Linux operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

#### Donald Knuth's $T_{EX}$ typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus The Art of Computer Programming, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TFX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TFX is a programmer's approach to word processing. Since T<sub>F</sub>X is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of T<sub>F</sub>X makes it relatively easy to learn how other people have created their own T<sub>F</sub>X documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft Word suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is All You Get).

#### Leslie Lamport's LATEX extensions to TEX

Like all true programming languages,  $T_EX$  is inherently extensible. So, years after the release of  $T_EX$  to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that  $T_EX$  is to LATEX as C is to C++. This means it is permissible to use any and all  $T_EX$  commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in  $T_EX$  include automatic index and table-of-content creation.

#### Tim Edwards' Xcircuit drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for Xcircuit, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

192

Gimp graphic image manipulation program

Essentially an open-source clone of Adobe's PhotoShop, I use Gimp to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although Gimp does offer its own scripting language (called Script-Fu), I have never had occasion to use it. Thus, my utilization of Gimp to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

#### SPICE circuit simulation program

SPICE is to circuit analysis as  $T_{EX}$  is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer SPICE for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of SPICE, version 2g6 being my "go to" application when I only require text-based output. NGSPICE (version 26), which is based on Berkeley SPICE version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all SPICE example netlists I strive to use coding conventions compatible with all SPICE versions.

Andrew D. Hwang's ePiX mathematical visualization programming library

This amazing project is a C++ library you may link to any C/C++ code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as Mathematica or Maple to do. It should be said that ePiX is not a Computer Algebra System like Mathematica or Maple, but merely a mathematical visualization tool. In other words, it won't determine integrals for you (you'll have to implement that in your own C/C++ code!), but it can graph the results, and it does so beautifully. What I really admire about ePiX is that it is a C++ programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a C++ library to do the same thing he accomplished something much greater. gnuplot mathematical visualization software

Another open-source tool for mathematical visualization is gnuplot. Interestingly, this tool is not part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" not be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my gnuplot output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of Gimp to do rudimentary image editing, my use of gnuplot only scratches the surface of its capabilities, but the important points are that it's free and that it works well.

Python programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type from math import \* you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (from cmath import \*). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

### Appendix D

# **Creative Commons License**

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

#### Section 1 – Definitions.

a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

i. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

#### Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

#### Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

#### Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

#### Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

#### Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

#### Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

#### Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority. Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at creativecommons.org.

### APPENDIX D. CREATIVE COMMONS LICENSE

### Appendix E

# Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

**29** May 2025 – commented out all content within the Full Tutorial chapter, and made the Simplified Tutorial be the only Tutorial. This was done because the old Full Tutorial chapter's contents have now been placed into separate PLC topical learning modules.

27 March 2025 – minor edits to the Simplified Tutorial.

7 March 2025 – reorganized the Full Tutorial chapter's structure on ladder diagram (LD) programming to be sections rather than subsections within one section.

4 March 2025 – minor typographical error correction, where the first letter in a sentence was not capitalized.

4-19 February 2025 – added more problems from the Socratic Instrumentation project.

9 November 2024 – added an Introduction section on challenging concepts.

24 July 2024 – divided the Introduction chapter into two sections, one for students and one for instructors, and added content to the instructor section recommending learning outcomes and measures. Also divided the Simplified Tutorial into sections.

**31 May 2024** – corrected errors in the "Example: PLC process switch statuses from contact status coloring" Case Tutorial section, courtesy of James Connelly, where I mixed up two of the inputs on the PLC (describing the wrong physical switches controlling inputs I:0/0 and I:0/3.

**29** April 2024 – added requirement to annotate schematic and color-highlight the PLC program in the Conceptual Reasoning question "Relay ladder logic analogy for a PLC", as well as modified the Challenge questions for the same. Also added more instructor notes to other questions.

18 May 2023 – minor wording improvement courtesy of Ron Felix.

**3-5 May 2023** – corrected multiple instances of "volts" that should have been capitalized "Volts" as well as an error in one of the Conceptual Reasoning questions where I wrongly referred to process switches as pushbuttons. Also added more questions challenging students to identify PLC virtual contact status in relation to real-world switch status, as well as new Case Tutorial examples.

**6 December 2022** – corrected an error in the Simplified Tutorial where I wrote "parallel" instead of "series" (!), and also added questions to the Introduction chapter. Also edited image\_6221 to add a symbol showing normally-closed switch B to be in the open state, and added some content to the Simplified Tutorial discussing HMIs.

**28** November **2022** – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

6 May 2022 – re-wrote Simplified Tutorial chapter.

5 May 2022 – added a Case Tutorial chapter.

10 May 2021 – commented out or deleted empty chapters.

18 March 2021 – corrected multiple instances of "volts" that should have been capitalized "Volts".

**29** June 2020 – clarified an answer in the "Allen-Bradley counter program" question regarding the number of times the motor will be allowed to start up.

20 May 2019 – added questions related to Human-Machine Interfaces (HMIs).

12 May 2019 – added more SELogic examples (S-R latch) to Technical References section, and made minor edits to other instruction examples in that section. Also added several new questions in Conceptual, Quantitative, and Diagnostic Reasoning sections. Also, re-structured sections and subsections related to Ladder Diagram (LD) programming. Also, added a section on Human-Machine Interfaces, or HMIs. Also, added an Experiment on using an HMI to read PLC bits and words.

9 May 2019 - added some PLC counter-based questions.

9 May 2019 – added more questions as well as a Project for a PLC-controlled system.

**1 May 2019** – added more questions in all categories (Conceptual, Quantitative, and Diagnostic), and also included "dry" versus "wet" output contacts for PLC discrete outputs.

**23** April 2019 – added more questions in all categories (Conceptual, Quantitative, and Diagnostic), and also included "dry" versus "wet" output contacts for PLC discrete outputs.

20 April 2019 – added some more Experiments.

19 April 2019 – minor edits to the demonstration program experiments.

16 April 2019 – continued adding content to the Technical References section on SELogic control equations for SEL's protective relays and programmable controllers.

**15** April **2019** – added a Technical References section on SELogic control equations for SEL's protective relays and programmable controllers.

14 April 2019 – added a lot of content to the Full Tutorial, as well as to the Technical References chapter.

9 April 2019 – continued adding more content to the Simplified Tutorial.

8 April 2019 – added a Technical Reference section comparing basic Ladder Diagram instructions of some common PLC models.

7 April 2019 – added content to Introduction and Tutorials.

 $30\ March\ 2019$  – document first created.

# Index

Absolute addressing, 55 Active reading, 41 Adding quantities to a qualitative problem, 184 Annotating diagrams, 183

Beaufort, Ron, 101 Breadboard, solderless, 154, 155 Breadboard, traditional, 157

Cardio-Pulmonary Resuscitation, 152 Checking for exceptions, 184 Checking your work, 184 Code, computer, 191 CPR, 152

Dalziel, Charles, 152 Dimensional analysis, 183 DIN rail, 155 DIP, 154 Direct addressing, 55 Drum instruction, Koyo PLC programming, 57

Edwards, Tim, 192 Electric shock, 152 Electrically common points, 153 Enclosure, electrical, 157 Equipotential points, 153, 155 Experiment, 158 Experimental guidelines, 159

Graph values to solve a problem, 184 Greenleaf, Cynthia, 73

HMI panel, 43 How to teach with these modules, 186 Human-Machine Interface panel, 43 Hwang, Andrew D., 193 Hyperterminal software, 65 IC, 154 Identify given data, 183 Identify relevant principles, 183 Instructions for projects and experiments, 187 Intermediate results, 183 Inverted instruction, 186

Knuth, Donald, 192

Ladder Diagram, 3 Lamport, Leslie, 192 Limiting cases, 184

Mask, Allen-Bradley SQO sequencer output instruction, 61
Memory map, 51
Metacognition, 78
Moolenaar, Bram, 191
Murphy, Lynn, 73

One-shot, 68 Open-source, 191

### PLC, 3

Potential distribution, 155 Problem-solving: annotate diagrams, 183 Problem-solving: check for exceptions, 184 Problem-solving: checking work, 184 Problem-solving: dimensional analysis, 183 Problem-solving: graph values, 184 Problem-solving: identify given data, 183 Problem-solving: identify relevant principles, 183 Problem-solving: interpret intermediate results, 183 Problem-solving: limiting cases, 184 Problem-solving: qualitative to quantitative, 184 Problem-solving: quantitative to qualitative, 184 Problem-solving: reductio ad absurdum, 184

#### INDEX

Problem-solving: simplify the system, 183 Problem-solving: thought experiment, 159, 183 Problem-solving: track units of measurement, 183 Problem-solving: visually represent the system, 183 Problem-solving: work in reverse, 184 Programmable Logic Controller, 3 Project management guidelines, 162 PuTTY software, 65 Qualitatively approaching quantitative  $\mathbf{a}$ problem, 184 Reading Apprenticeship, 73 Reading, active, 41 Reductio ad absurdum, 184–186 Safety, electrical, 152 Schoenbach, Ruth, 73 Schweitzer Engineering Laboratories, 64 Scientific method, 78, 158 Scope creep, 162 SEL, 64 SEL-2440 DPAC, 65 SEL-5030 AcSELerator QuickSet software, 65 SELogic, 64 Sequencer Compare instruction, Allen-Bradley PLC programming, 63 Sequencer Load instruction, Allen-Bradley PLC programming, 63 Sequencer Output instruction, Allen-Bradley PLC programming, 59 Set-Reset latch, 67 Shunt resistor, 154 Simplifying a system, 183 Socrates, 185 Socratic dialogue, 186 Solderless breadboard, 154, 155 SPICE, 73, 159 SPICE netlist, 156 Allen-Bradley SQC instruction, PLC programming, 63 SQL instruction, Allen-Bradley PLC programming, 63 SQO Allen-Bradley PLC instruction, programming, 59

Stallman, Richard, 191 Subpanel, 157 Surface mount, 155 Symbol, 56 Symbolic addressing, 56

Tag name, 56 Terminal block, 153–157 Terminal emulator software, 65 Termite software, 65 Thought experiment, 159, 183 Torvalds, Linus, 191

Units of measurement, 183

Visualizing a system, 183

Wiring sequence, 156 Work in reverse to solve a problem, 184 WYSIWYG, 191, 192