Ladder Logic

Introduction and Examples

Programmable Logic Controllers (or PLC’s) are computers with enhanced hardware that can withstand industrial environments. They are used to control manufacturing processes such as small scale assembly lines. Designed to be operated by engineers with limited knowledge of programming languages, PLC’s can be used to automate many common industrial scenarios.

To program a PLC, most users use Ladder Logic, a graphical language inspired by circuit diagrams from relay logic. In Ladder Logic, every program is represented by a set of diagrams, and every diagram is made up of symbols, representing input and output elements, such as buttons and conveyor belts; and lines, which are responsible for connecting those symbols into the diagrams.

To build a diagram in Ladder Logic we have to follow a sequence of steps. A diagram always starts with two vertical lines, called power rails, between which circuits are connected. One or more lines (horizontal and vertical) are added connecting the two power rails. Each connection between the power rails is called a rung.

Input and output symbols are placed on the rungs and identified by descriptive labels. By definition, every rung starts with one or more input symbols and ends with one output symbol.

If you are still confused about what inputs and outputs can be, here are some examples: As inputs, we usually have buttons, switches and sensors. As outputs, we usually have motors, buzzers, valves, lights, and many others.

To understand what a diagram is expressing, we have to follow some rules. A diagram in Ladder Logic is always read from the left to the right, and from the top to the bottom. Each rung on the diagram defines an operation in the control processes of an industry, such as the turning of a light, or the activation of a hoist. When a diagram is executed, the power flow cycles through the rungs, from left to right, and from to top bottom.

Notice that there are different ways of representing the inputs and outputs in a diagram. For this survey, we will focus on three different symbol representations, the normally closed and normally open contacts to represent inputs, and the coil to represent outputs.

Let’s start with an example in Ladder Logic. In this example, we will use a button to turn on a light. The possible values for the light and for the button can be represented by the table below. If we use an open contact to represent our button as follows, when the button is activated (true), the light will be on (true). On the other hand, when the button is not activated (false), the light will go off (false).

If we change the input symbol from the open contact to the closed contact representation, the logic of our diagram will also change. In such a case, when the button is activated (true), the light will be off, and when the button is deactivated (false), the light will be on, inverting the logic of our output. This diagram not only represents a different way to activate a button, but also represents the NOT operation in boolean logic.

In a more complex situation, we may have to use two buttons to activate the light. If we use two open contacts in a sequence to represent our buttons, the following control process will be executed. When both buttons are activated (true and true), the light will be on, otherwise, the light will be always off. This phenomena happens because everytime a button is not pressed (false), the power flow will be interrupted at some point of the circuit, preventing the activation of the light. This diagram also represents a boolean logic operator, called AND.

Our inputs may not be in sequence all the time. Parallel inputs are also accepted, as we can see in the following example. In such a case, when both buttons are activated (true and true), the light will also be on. The difference is that the interruption of one button, will not prevent the light from turning on. Notice that the only case where the light will be off, is the case where both buttons are not activated. This diagram also represents a boolean logic operator, called OR.