



## **Blū Logic** User Manual and Reference

Cyclotronics manufactures electronic controllers for our customers' products. One thing that makes them unique is that they are programmed in Ladder Logic which is the most common paradigm for programming industrial controls. Blū Logic is an integrated environment for developing ladder logic diagrams which can then be downloaded to our custom controllers.

Ladder Logic produces very robust and readable programs capable of executing multiple control processes, synchronous or not, concurrently. Changing one section of or process in a Ladder Logic program is much less likely to affect another section than is the case with more computer-oriented languages.

This manual introduces Ladder Logic in general, explains how to use the Blū Logic program, and includes a reference to all of the Ladder Logic elements available to build ladder diagrams for your controller.

Because all of our controllers are custom designed, their features vary widely. For example some have digital and analog I/O while others have digital I/O only. Some have displays or screens and some have adjustable controls. But the Blū Logic program and this manual apply to the entire line. Therefore, some of the features described in this manual may not be available on your particular model. The Blū Logic program will not give you access to these features, if you properly select the matching controller model when you start the program.



<b>Introduction to Ladder Logic</b>	<b>5</b>
<b>Installation</b>	<b>11</b>
<b>Selecting the Controller Model</b>	<b>11</b>
Assets and the Asset Palette	13
Building the Diagram	13
Wiring Connections	14
Selection and Multi-Cell Operations	15
Copy, Cut, and Paste	15
Insert and Delete Rungs	15
Reconfiguring Assets	16
Clip Leads and Lockout Tags	17
<b>Downloading to a Blū Logic Controller</b>	<b>18</b>
<b>Reports</b>	<b>19</b>
Print Diagram	19
Index	19
Asset List	19
<b>Troubleshooting</b>	<b>19</b>
<b>Language Details</b>	<b>21</b>
<b>Contacts and Coils</b>	<b>21</b>
<b>Rung Execution Order</b>	<b>23</b>
<b>Timers and Counters</b>	<b>24</b>
<b>Shift Registers</b>	<b>28</b>
<b>Sequencers</b>	<b>29</b>

<b>IF and REG=</b>	<b>30</b>
<b>Mathematical Expressions (Formulae)</b>	<b>31</b>
Operators in Precedence Order	31
<b>Analog Inputs and Outputs</b>	<b>32</b>
<b>Modbus Communications</b>	<b>33</b>
<b>Controller Startup</b>	<b>35</b>
<b>Integer Math</b>	<b>37</b>
<b>Blū Logic Elements</b>	<b>41</b>
Contacts	41
Coils	41
Timers	42
Counters	42
<b>Special and Shortcut Keys</b>	<b>43</b>
<b>Execution Time of Operations and Elements</b>	<b>45</b>

## ***Introduction to Ladder Logic***

You may want to skip this section if you have used ladder logic before. This introduction is intended for the new user. If so, you'll find the next section on page 11.

A ladder diagram is a kind of electrical circuit diagram. It looks like a ladder. There are two long vertical lines, one on either side, representing the hot and neutral power lines. The rungs of the ladder are combinations of switch contacts and loads (like lamps or relay coils) connected together with lines representing wiring. When a combination of switches is on, the current can flow from the hot rail through the switches and loads to the neutral rail.

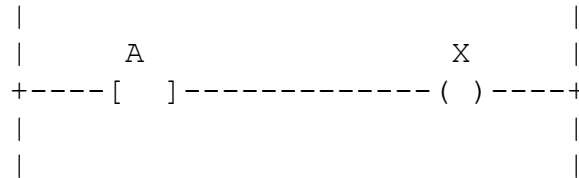
Ladder Logic is a computer programming language in which the programs are ladder diagrams. It doesn't look like a program, but it is. Computer software, or sometimes a patient engineer, can translate the ladder diagram to a series of instructions which are executed by a controller. The diagram delivers the same behavior that a physical circuit of actual relays, timers, etcetera would deliver if wired according to the diagram.

Ladder Diagrams have been around for over a century. Ladder Logic was invented in the late 1960's by a group of engineers trying to simplify the complex task of wiring relay logic to build control systems for automation in the auto manufacturing industry. Dick Morley was part of the team and is widely recognized as the father of the Programmable Controller and Ladder Logic. (His many other accomplishments include anti-lock brakes.)

The original idea was that a ladder diagram was much easier to understand than a conventional computer program and could be used by engineers who were not trained in computer programming, then mostly written in arcane "assembly language". These days, most engineers who work with automation actually do have experience with conventional text-based computer languages but Ladder Logic has other advantages and is still widely used in automation.

At the time, virtually all interaction with computers was textual, and with a fixed-width font at that. Graphics were rare, low-resolution, and slow. A lot of modern ladder logic programs, including Blū Logic, use graphics to display the diagrams but the language was originally designed so that the diagrams could be drawn with standard text symbols.

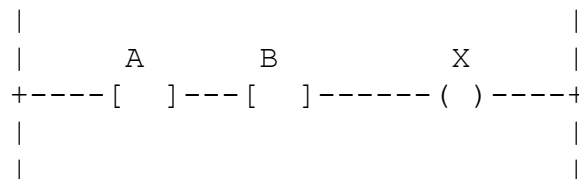
For example, here is a single “rung” of ladder logic.



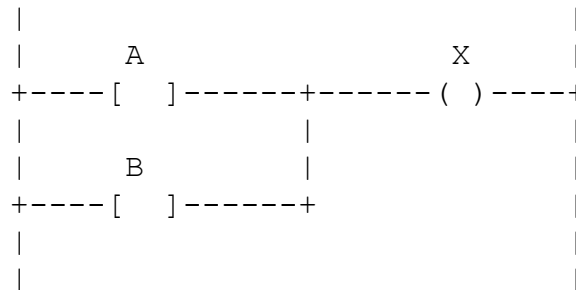
The vertical line on the left side is the hot rail. The vertical line on the right is the neutral rail. An imaginary voltage is present between these rails. The element  $-[ ]-$ , labeled A, on the left is a “contact”. In this case, it is a Normally-Open (NO) contact. The connection is broken until a corresponding coil or physical input turns on. Then the contact “closes” and current can flow through it, energizing elements wired to its output, on the right side. The element  $-( )-$ , labeled X, on the right side of the diagram is a “coil”. In this simple diagram, when the contact is closed, the coil will be energized. The coil can either operate other contacts in the diagram (a so-called “internal relay”) or can operate an external circuit in the real world.

There are other forms of both contacts and coils. The Normally Closed (NC) contact looks like  $--[/]--$ . It passes current when the corresponding physical input or coil is *not* energized. With just these three elements, logic of arbitrary complexity can be implemented. There’s actually a mathematical proof of that statement but I won’t bore you with it here.

Here is another example. This one has two contacts A and B. Because both contacts must be closed to complete the circuit and let current flow to the coil X, this circuit implements a logical “AND” function. Both A *AND* B must be closed to operate the coil X.



If there's an AND, then there has to be an OR, right? In the following, closing either A *OR* B will complete the circuit and operate the coil:



Suppose you need to implement a surge tank. You have a constant in-flow of product from some upstream process and you need to deliver it to a downstream step on demand. Your surge tank will be equipped with a pump to send the product downstream and a pair of level sensors for the maximum and minimum levels of the tank.

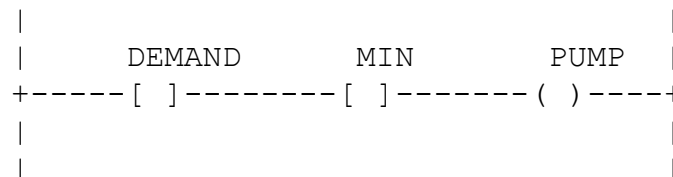
Your controller would have inputs from the level sensors and the downstream demand signal and these will appear in the diagram as contacts. Let's name them

MAX	This contact closes when the product reaches the upper level switch.
MIN	Likewise, this is the lower level switch. It is closed when the product level is above some minimum and open when the level is below it.
DEMAND	Indicates that the downstream process wants product.

You'll also need some outputs which will appear as coils in the ladder diagram

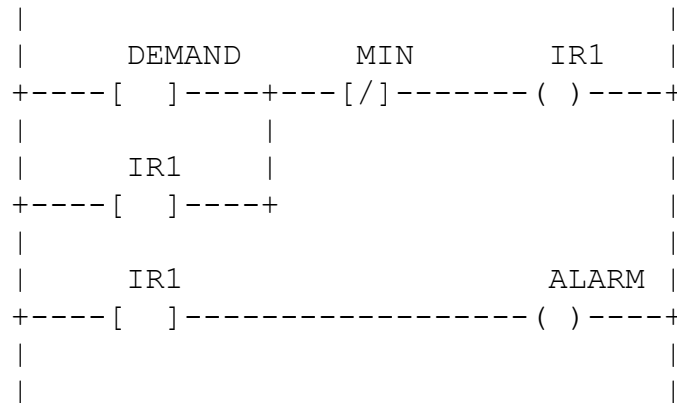
PUMP	Energize this coil to pump product down stream.
HOLD	Tell the upstream process to pause .
ALARM	Warn someone that we have a problem.

So the first thing you need is to turn on the pump when the downstream process requests product. You'll want to stop the pump if the tank is pulled down below the minimum level. Your rung will look like this:

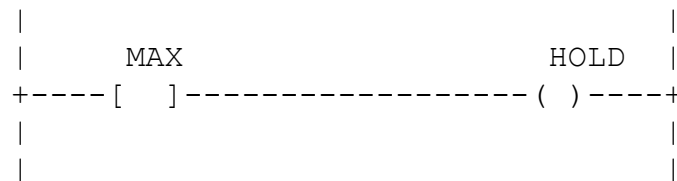


And, if the tank level does fall below the minimum switch, we need to alert an operator. We're only going to sound the alarm to alert the operator if the downstream demand signal is on, but once the alarm is activated, we want to keep it on until the low level sensor is activated again. We'll use an internal relay IR1, for this. Note that the second

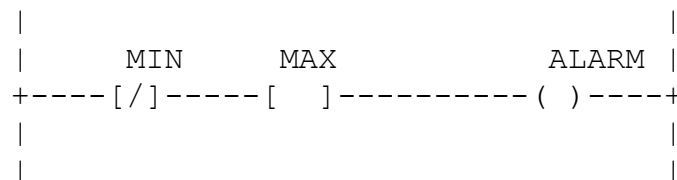
rung below latches the internal relay until the level rises above the MIN switch, operates the switch, and opens the corresponding NC contact, even if the demand signal goes away in the meantime.



If the tank level reaches the upper sensor, we need to stop the upstream process



And finally, if we ever see MIN de-energized, implying the level is below the MIN switch, and MAX energized, implying that the level is above the MAX switch, at the same time, we know something is wrong.



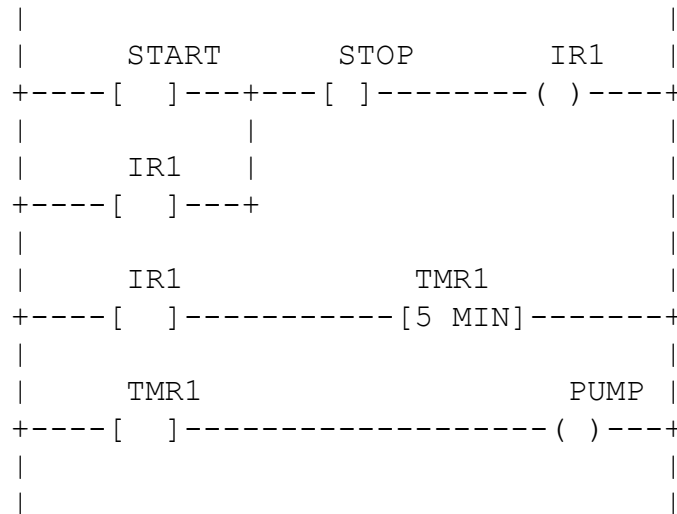
Besides contacts and coils, Ladder Logic includes Timers, Counters, and Registers for making computations. Blū Logic and some other versions also have Shift Registers and Sequencers.

Got it? Here's another example.

Suppose you want to control a submersible pump. You will want a start button with a sealing contact to keep the pump running after the operator takes his finger off of the button. You will want a stop button. Stop buttons in automation are normally closed so that the system will stop if a wire to the stop button breaks.



A submersible pump application can suffer from “backspin”. When the pump stops for some reason, gravity pulls the effluent back down the pipe and spins the pump like a turbine, in the opposite of the normal direction. If you try to start the motor while it is backspinning, you can break the shaft. So a control needs to prevent a restart while the pump could still be back-spinning. This can be accomplished with a timer.

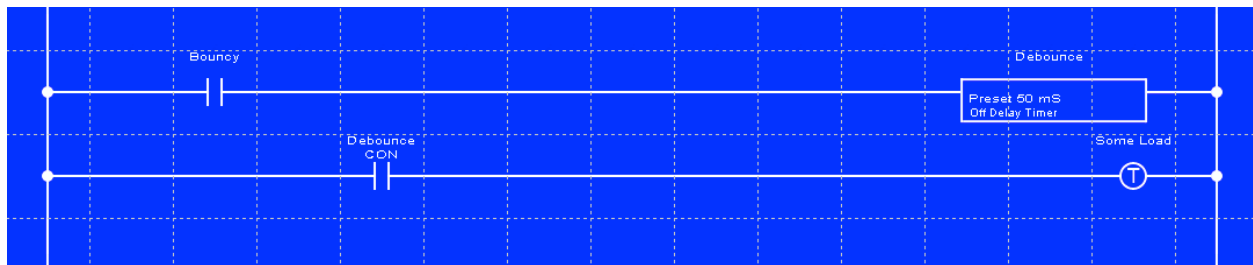


By now, you should be able to follow the sealing logic. IR1 is an internal relay. When the start button is pushed, and the stop button is NOT pushed, IR1 will be energized. IR1’s contact then bypasses the start button and keeps IR1 energized until the stop button is pressed. Notice that you do NOT use a Normally Closed contact to represent the stop button. The button itself is a normally closed switch. A Normally Closed contact reverses the sense of any kind of switch. The NO contact in the example OPENS when the stop button is pushed because that is when the button’s electrical switch opens. It might help to think of the stop button as an “enable” switch which is closed until someone pushes on it.

TMR1 is an “On Delay Timer”. When IR1 closes and energizes the input to the timer, it begins to run. When it reaches its preset of 5 minutes, the timer’s contacts will close and energize the pump. An On Delay Timer does not delay turning off. If the stop button is pressed and IR1 is de-energized, the timer’s contacts open immediately and shut off the pump. The physical output associated with the “Pump” coil is normally not the pump motor itself but a contactor or motor starter which handles the larger load of the motor.

This introduction has presented the basics of Ladder Logic. There’s more to it, although less than you might think. We haven’t touched on analog I/O nor explained counters and other flavors of timer. Refer to the *Reference* section of this manual for a complete list of the elements available in Blū Logic.

In theory, a ladder diagram is a set of rules that are applied concurrently, just as physical relays would work. In practice, since the CPU inside your Blū controller is a sequential machine, ladder diagrams are executed sequentially and the practical ladder programmer is aware of this fact and depends on it. One pass through the diagram, top rung to bottom, is called a “scan”. One of the important specifications of a controller is its “scan time” because this determines how quickly the diagram can respond to events. Scan times of modern controllers are typically a few tens of milliseconds. Controllers based on Blū Logic are extremely fast. Scan times for small or medium sized programs are rarely more than one millisecond. In fact, the scan time is so fast, that your diagram may react to contact bounce of mechanical switch contacts as if the switch was operated more than once. It is a good idea to design logic so that additional activations of a given contact do not matter, but this is not always possible. If contact bounce is a problem in your application, use an off-delay timer to de-bounce the contact.



It is generally considered safe to assume that the rungs of the diagram are processed in order from the top to the bottom of the diagram. However, it is risky to make any particular assumption about the order of execution of elements *within a single rung*.

Real-world ladder diagrams comprise hundreds, even thousands, of rungs. As such, they can implement remarkably sophisticated behavior and control large and complex machines, production lines, and even entire factories.

There are many good resources available for learning more about Ladder Logic. From Wikipedia to paper books such as “Programmable Logic Controllers - an introduction” by W. Bolton. The web site [www.plcs.net](http://www.plcs.net) offers a popular book about Ladder Logic and includes an extensive online tutorial and an interactive forum for discussing PLCs in general and Ladder Logic in particular.

## Using Blū Logic

### Installation

Blū Logic requires a computer with the Microsoft Windows operating system. It is compatible with Windows XP, 2000, Vista, and Windows 7. Compared to most Windows software, installation of Blū Logic is very simple. You just download the program to your desktop. When you are ready to use it, double-click to launch the program. There are no additional drivers, DLLs, or other special files to install. To un-install Blū Logic, just delete the file from your desktop.

Blū Logic is not very resource intensive and should run well on even the slowest PC you can buy these days. It does require a USB port to connect to your controller.

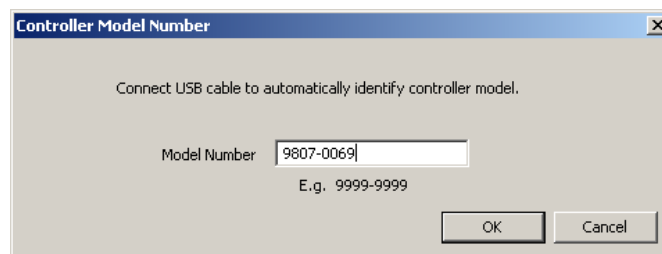
Each Blū Logic ladder diagram you create is saved in a single file with the extension “.BLU”. You choose where you wish to store these files on you hard drive. Some users will want to keep them with other documents relating to the same project. Others will want to create a single common folder and store all of their Blū Logic ladder diagrams in one place. The complete computer novice (who probably isn't reading this) may just keep them on the desktop.

Remember that there is no way to retrieve a downloaded program from the controller. This is by design to deter reverse engineering of our customers' products. However, it also means that you must be sure to keep track of the .BLU file for each of your projects so that you can refer to it in the future. To be doubly safe, print out the diagram (Under the Reports menu) and archive the printout with other project documentation.

### Selecting the Controller Model

Blū Logic must know which model of Blū controller you need to develop a diagram for. Whether it is a standard model or a custom design, each Blū controller has a model number. This is an 8 digit number in the form 9999-9999. You will find this number printed on the controller's printed circuit board. Please take care to select the correct model so the program will present you with the matching feature set.

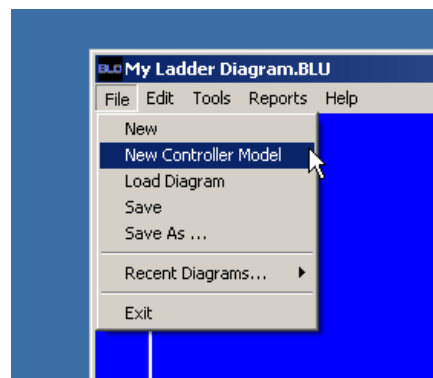
When you start Blū Logic for the first time, it will display a dialog requesting the model number. You can either type it in, or you can connect the controller's USB port and the



software will interrogate the controller and fill in the model number for you. Click “OK”.

When you select a controller model for the first time, Blū Logic will contact our servers and download a list of that model’s assets and other information it needs to develop diagrams for a specific controller model. In order to download, Blū Logic must be able to access the Internet. After a given controller model is downloaded one time, Blū Logic will keep a copy of the information on your hard drive so it is not necessary to always have an Internet connection. But it is necessary for your PC to be on-line each time you ask Blū Logic to prepare a diagram for a controller model you haven’t used before.

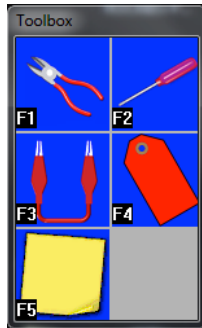
Blū Logic will remember your controller model between sessions. If you have occasion to work with more than one model, then you can specify the model when you create a new diagram. Select “New Controller Model” from the File menu.



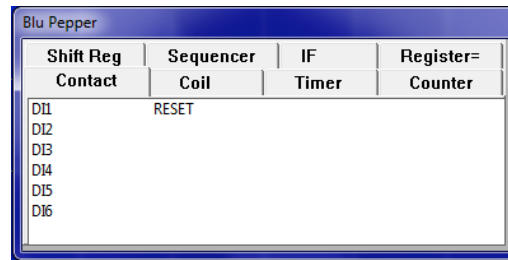
## **Using the Editor**

The Blū Logic ladder editor allows you to enter and modify logic in the form of a diagram. Operation of the editor is very intuitive but, even if you are an experienced ladder programmer, you will want to review this section because there are some special tricks and keystrokes that are unique to this design. If you don’t want to read all of this, well OK, but at least glance at the “Special Keys” section in the reference.

When you start the Blū Logic program, it will present you with an empty ladder diagram in the main window. Two additional windows will also open and can be positioned separately on your screen. These are the Toolbox window and the Asset Palette. The Toolbox includes the screwdriver for making connections, the wire cutter for breaking them. It also provides the alligator clip lead for temporarily shorting a contact and the lockout tag for temporarily forcing a contact open. At the bottom of the toolbox is the sticky note tool which you can use to add comments to your diagram.



Toolbox



Asset Palette

Using the editor in general consists of selecting a “tool” or an asset, applying it to the diagram one or more times, selecting another tool, and so-on. The tools in the Tool Box can also be selected by using the function keys F1 through F5. Press the ESC key to deselect the current tool.

### Assets and the Asset Palette

The Asset Palette shows you a list of the coils, contacts, and registers for your Blū controller. In a new diagram, only the built-in coils and contacts corresponding to the I/O points of your particular Blū Logic controller will appear. You can rename the built-in assets to correspond to your application. And you can create internal relays, timers, counters, shift registers, sequencers, and additional registers as needed. The Asset Palette has a tab for each type of resource: Contact, Coil, Timer, Counter, Shift Register, Sequencer, REG=, and IF.

### Building the Diagram

There are a couple of ways to lay down circuit elements to build your diagram. The most obvious is to click on an asset in the Asset Palette and then click on the diagram to place an instance of it. If you click on [NEW RELAY], [NEW TIMER], [NEW COUNTER], [NEW REGISTER], [NEW SHIFT REG], or [NEW SEQUENCER], a dialog will pop up for you to name and configure the new asset.

For each timer or counter you create, a contact and two associated registers are also created. The contact is named with the name of the timer and the suffix .CON such as MyTimer.CON. A timer will have registers with suffixes .TIM for the timer and .PRE for the preset. A counter will have .CNT and .PRE registers.

Timers, Counters, Shift Registers, Sequencers, and the REG= element must be at the far right side of the diagram, up against the neutral rail. These blocks are like the “motor” part of the timer. The contacts for each of these assets appear in the Asset Palette and are added to the diagram separately. It is fairly common to have one of the contacts of a coil or counter wired to energize its own reset input. Another common feedback circuit is to connect a shift register contact to its own data input so that the bit pattern constantly circulates.

When you select an asset, the cursor changes to the graphical representation of that element which we call the “tool”. This is also true of the wire cutter, sticky note, and screwdriver. To de-select any tool, hit the ESC or ESCAPE key on the keyboard.

If the currently selected tool is a contact, you can hit the space bar on your keyboard to rotate between NO, NC, RISE, and FALL variants. For a coil, you can rotate between COIL, SET, CLEAR, and TOGGLE.

If you select a coil from the Asset Palette, it will appear as a coil only over the right-most column of the ladder diagram. Anywhere to the left, it will appear as a contact. If you use it as a contact, it reflects the current state of the coil. This is naturally how internal relays are employed but you can do the same thing with any output point.

When you click on the diagram with a tool, any circuit element previously occupying that cell is replaced.

Blū Logic includes an alternate way for the experienced user to enter logic elements. To use this method, you must first make a change in the preferences. Click “Edit” and choose “Preferences.” Make sure “Clear selection when an asset or tool is clicked.”

With this option checked, as it is initially, any time you click on an asset in the asset pallet or a tool in the tool box, any selected cells are de-selected. With the option unchecked, you can select a cell and then click an asset to enter it in that cell. Try this:

Click on an empty cell in the diagram. It will turn dark to indicate that it is selected. You can drive this selection around the diagram with the keyboard arrows. If you click an item from the Asset Palette when you have a single, empty cell selected, an instance of that asset will be placed in the empty cell and the selection cursor will move to the right, ready for the next element. You can also configure the program to put a single-cell-wide wire between each element placed in this way, which facilitates making vertical connections later. If you need to change the type of a contact or coil, for example from NO to NC, use the left arrow to move back to it, hit the space bar to change it, and then hit the right arrow to go back to the next empty space.

You can think of the two methods as either clicking in the palette first and then on the diagram or clicking on the diagram first and then the palette. The former seems to be more natural but the latter method requires less movement of the mouse overall.

### Wiring Connections

The screwdriver is used to make connections. Click the screwdriver in the Toolbox and your cursor will change to a screwdriver. Click on either free end of a coil, contact, or other element. Holding the left mouse button down, drag the cursor to the other end of the desired connection. When you release the button, the program will attempt to wire

your two points together. You can exercise more control over the wiring by doing it in segments. Click and drag in either a purely horizontal or vertical direction and release. Click, drag, and release to create the next segment and so-on until you reach your destination. Contacts in the left-most column of the diagram and coils in the right-most column will connect themselves to the vertical power supply rails.

Typically, you should let the automatic connection router try first. Most of the time, it does a good job. If you don't like the result, type Ctrl-Z to undo the connection. Then manually wire the connection step by step. With time, you'll get a feel for when you can trust the automatic connection router and when it needs a little help.

The wire cutter tool is used to delete circuitry. If you delete a wire segment or a circuit element, all wiring connected to it will automatically be ripped out. Holding down the SHIFT key as you click will temporarily disable auto rip-up. You can turn this feature off persistently in the Preferences dialog. Look for it under the Edit menu.

### Selection and Multi-Cell Operations

You can use the mouse to select a cell or a collection of cells of the diagram. Left-click a cell to select it. Left-click and drag to select a rectangular region. Each time you click or click and drag, the previous selection is lost. To retain previous selections, hold down the CTRL key as you click or click and drag. In this way, you can select any combination of cells. The DEL or DELETE key will clear all selected cells in a single operation. There are other operations in the Edit menu which you can then invoke and the chosen operation will be performed on all selected cells.

### Copy, Cut, and Paste

When one or more cells of the diagram are selected, you can copy them and then paste the copy elsewhere. Cut is the same as copy but Cut will also delete the original instances. Copy, Cut, and Paste appear in the "Edit" menu and can also be activated with the shortcut keystrokes Ctrl-C, Ctrl-X, and Ctrl-V respectively.

Copying is complicated by certain syntactic rules such as, *you can only use one instance of a given timer or counter, and all timers, counters, sequencers, etc must be in the rightmost column*. In order to comply with these rules, the Paste operation may refuse to paste an element which already appears elsewhere in the diagram and may relocate some elements to the right-most column if you Paste a selection farther to the left than when you Copied it.

### Insert and Delete Rungs

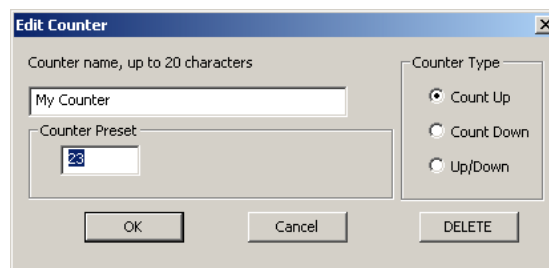
The Edit menu also has "Insert Rung" and "Delete Rung" functions. To insert a rung to open up space for more logic, click anywhere in the rung and then choose "Insert Rung" from the Edit menu. A new, blank, rung will appear *above* the rung you had selected. If

you select more than one row of the diagram, by clicking and dragging, then that same number of rungs will be inserted.

If you insert a rung between two or more rungs which are connected vertically, these vertical connections are automatically extended through the new rungs. You can easily delete these connections if they are not required. Click and drag to select your rung or rungs, hold down SHIFT, and click the DEL key on the keyboard.

### Reconfiguring Assets

When you create a new asset, a small dialog window pops up and lets you name and configure the new asset. To change this information later, right-click on the element in the diagram, or on the master copy of it in the Asset Palette. This will pop up the dialog again and let you make changes. If you change the name of a register that is referred to in formulas elsewhere, in REG= and IF rungs, the program will offer to automatically change all of these references to the new name. Normally, you will want to take advantage of this assistance.



Each REG= and IF element includes a formula, the result of which is either compared to the register or assigned to it. If you right-click on one of these elements in the diagram, you can edit the formula. Only that one instance will be affected by the change.

When editing a formula, an auxiliary window pops up with the names of all registers. At any time, you can click a register in this list as a fast way of typing it. The auxiliary window disappears when you close the formula editor.

### Sticky Notes

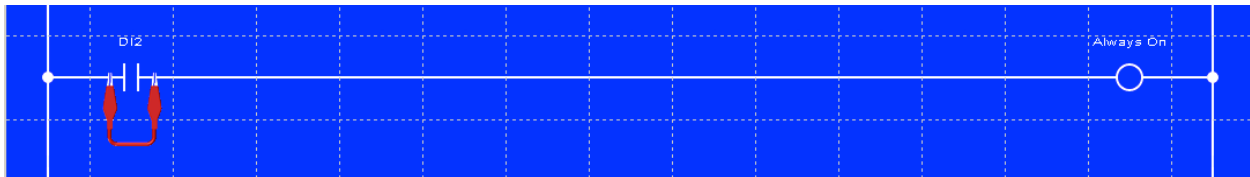
The sticky note tool lets you add notes to your diagram. Click on the yellow sticky note in the toolbox. Then move the cursor over the diagram to where you want the note and click. A dialog will pop up and allow you to enter the text for your note. Click "OK" when done.

You can move and edit sticky notes later. To reposition a note, left-click on it and drag it to the new position. To edit or delete the note, right-click on the note. An editing dialog will appear.

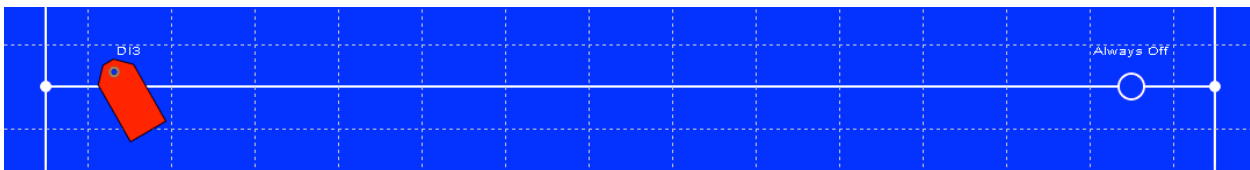


## Clip Leads and Lockout Tags

The Clip Lead and the Lockout Tag are used to temporarily force the state of a contact without irreversibly changing your diagram. If a clip lead is applied to a contact, it will always pass current regardless of the state of the coil or external input. That is, the contact will always act as if it was closed. Note that, in the case of a normally-closed contact, this is equivalent to *not* operating the corresponding physical input or internal relay coil. Either an NO or NC contact will pass current when bypassed by a clip lead.



The lockout tag does the opposite. It forces a contact to be open. A tagged contact will not pass current.



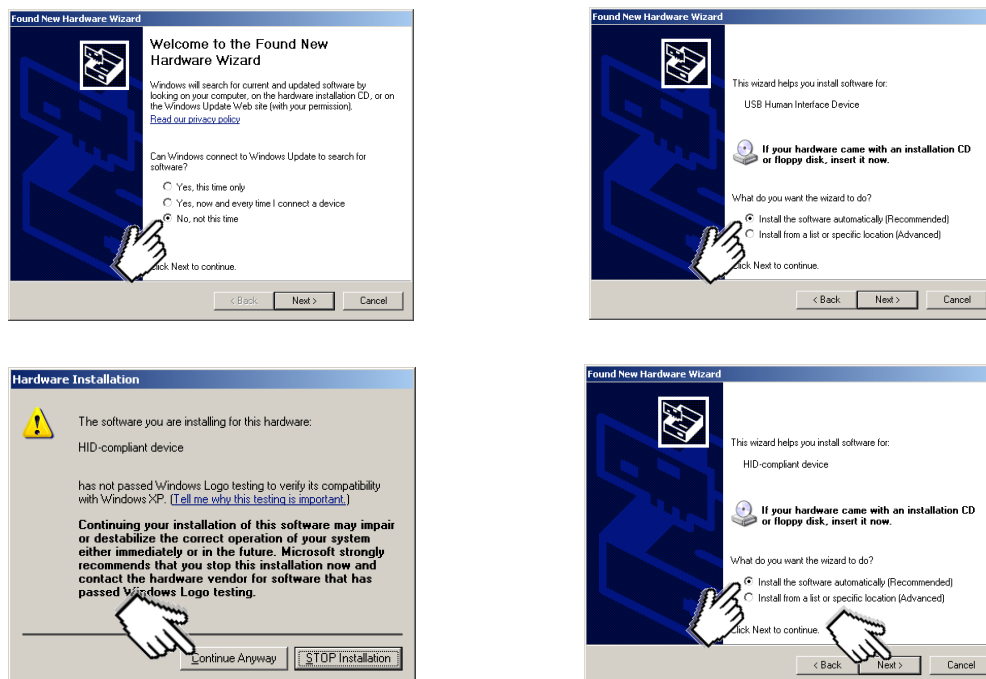
Clip leads and Lockout tags are applied by clicking on the corresponding tool in the Toolbox and then clicking on a contact or an IF element. A clip lead or lockout tag only affects the single contact instance it is applied to. Other contacts in the diagram with the same name are not affected unless each one has a clip lead or tag applied. To force all contacts with a given name, hold down the Shift key when clicking the tool on a contact.

Clip leads and Lockout tags can be removed by clicking the same tool on the same contact a second time. In addition, all clip leads and lockout tags can be removed from the diagram in a single step with the “Clear clips and tags” command in the Edit menu.

## Downloading to a Blū Logic Controller

To actually do anything with your ladder diagram, it must be loaded into your controller. Connect it to your PC with a USB cable. If the controller is scanning at the time the USB cable is connected, it will stop and turn off all outputs.

The controller's USB programming port is designed to emulate a Windows Human Interface Device (HID). This allows it to connect via the built-in Windows driver, hidusb.sys, rather than requiring a special driver. The first time you connect a Blū Logic controller to your PC, Windows may, depending on the version, display one or more dialogs to associate it with the HID driver. Respond in the affirmative. Some examples are shown below with the appropriate responses indicated.



You will probably have to repeat this process if you later use a different USB port or if you otherwise re-arrange your USB network.

Once your controller is connected, and the driver is associated if required, you are ready to download your diagram. Under the Tools menu, choose "Download." Blū Logic will do some error checking of your program and, if all is well, will send it to your controller. If already connected to power, the controller will start scanning the diagram as soon as the USB cable is disconnected.

If Blū Logic finds problems with your diagram, a new window will pop up with a list of error messages. Click on an error message to quickly find that part of your diagram. You can close the error message window by clicking the X at the right side of the title bar.

You can run the download error check by itself to make sure that the download will succeed. You do not need a controller connected to do this. This function is called “Error Check” and it’s also in the Tools menu.

To avoid confusing the program, do not connect more than one controller at a time to your PC. If you do, the program will only download to one of the connected controllers, but which one is unpredictable.

## **Reports**

Under Reports in the main menu, you will find the following choices:

### **Print Diagram**

This does exactly what it sounds like. The ladder diagram is printed out on paper. This is useful for documentation, when you are trying to trouble-shoot your logic, or as a way to communicate the machine’s logic to other personnel.

### **Index**

The printout of a very large diagram may span many pages. To help you find your way around in it, print an Index report. The Index lists all of the assets (Contacts, Coils, Timers, Counters, etc) in alphabetical order. It describes what type of asset each is. Then it shows the line numbers, found on the diagram printout, for each rung in the ladder where an instance of that element appears.

### **Asset List**

The printed Asset List report shows a list of all assets and shows some of the internal details associated with the elements. If you rename some of the built-in assets, the Asset List may be helpful in deciphering which physical terminal of the controller is associated with which logical asset. Otherwise, you will not normally have any use for this information but factory support personnel may ask for it.

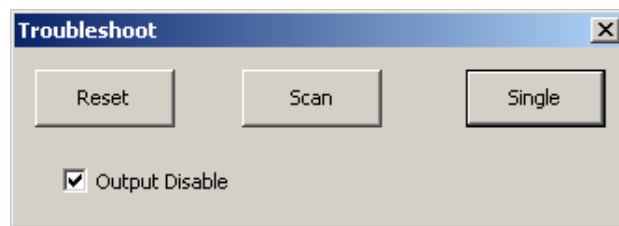
## **Troubleshooting**

Blū Logic features a Troubleshooting function in which the controller scans your diagram while still connected to the PC via the USB cable. This allows Blū Logic to display the states of contacts and coils and the values of timers, counters, and other registers as they change.

Diagram scan time is greatly reduced in troubleshooting mode. How much so is a complex function of computer speed, USB network speed and topology, and diagram complexity but it will definitely scan more slowly than the stand-alone controller with the

USB cable disconnected. This change in timing may cause a dramatic change in the behavior of your diagram. Depending on what you are controlling, it could even be dangerous so consider the consequences before using troubleshooting mode and troubleshoot with the outputs disabled if at all possible.

When you select Troubleshooting from the Tools menu, a dialog appears with three buttons.



The Reset button performs the power-up initialization of the diagram as described on page 33, including setting the RESET contact. The Scan button will start and stop scanning. The Single button will cause the controller to execute a single scan through the diagram and then pause until you press Single (or Scan) again. Single-scanning can be used to study your logic in super slow motion detail.

While running in Troubleshooting mode, an LED is displayed just to the lower, right corner of each contact or coil, indicating its logic state. The value in a timer or counter register is displayed in the upper, right corner of the timer or counter. The value of other registers are displayed next to IF and REG= elements.



The Output Disable checkbox forces the MCR (Master Control Relay) to be open at the beginning of each scan. *Your diagram can still turn the MCR back on!* So the checkbox is not a complete guarantee that an actuator will not operate unexpectedly.

When you're done with troubleshooting mode, close the troubleshooting dialog by clicking on the X at the right side of the title bar.

Your controller must be connected to the PC to use Troubleshooting mode, but the I/O and power terminals do not have to be connected to anything.

## Language Details

### Contacts and Coils

Blū Logic supports four kinds of contacts. NO, NC, RISE, and FALL. Normally open contacts look like this



A Normally Open (NO) contact is closed (passes current) when the corresponding physical input is energized. It is open otherwise. For an internal relay, an NO contact is closed when the corresponding coil is energized and open otherwise. Each physical output can also be referred to by contacts that reflect its state. For a given physical input, physical output, or internal relay, any number of contacts may be used in the diagram.

A Normally Closed (NC) contact



is just the opposite. It is closed when the corresponding physical input, physical output, or internal relay is *de*-energized.

A RISE contact



is closed for the single scan during which its associated physical input or internal coil changes from off to energized. You can think of the output as a pulse which fires on a rising edge of the input.

A FALL contact



is the opposite of a RISE contact. A FALL contact is closed for the single scan during which its associated physical input or internal coil is first de-energized following a period of being energized.

Every Blū Logic controller has a contact named RESET. This contact is closed for a single scan of the ladder at startup. You can use it to set up initial conditions, such as latching on the MCR, for example.

Blū Logic features two kinds of relays, normal and latching.

A normal coil is represented by



in the ladder diagram. It must appear in the right-most column of the diagram, touching the neutral rail. It doesn't really have an "output" per se, its connection to the neutral rail is part of the physical wiring analogy represented by ladder logic. When the left side of a coil is energized, the corresponding physical output will be turned on. When the input is de-energized, the physical output turns off.

An internal relay may have any number of contacts and any number of coils. When an input to a coil of an Internal Relay is energized, then all of that Internal Relay's contacts will operate.

A latching relay can have three different types of coils (and any number of each type)



Set



Reset



Toggle

When the input to a SET coil is energized, the associated physical output or internal relay turns on. It stays on even when the input to the SET coil is turned off. An energized input to the RESET coil of the same relay will un-latch the relay and turn the output back off. If both SET and RESET coils are on in the same rung, the result is indeterminate and may be unpredictable.

A TOGGLE coil will cause the state of a latching coil to be reversed. If the output was on, an energized TOGGLE will turn it off and vice-versa. A TOGGLE is only sensitive to a rising edge on its input. If the input is maintained, the relay won't constantly toggle on and off.

If you use a SET, RESET, or TOGGLE coil with any relay, then it is defined as a latching relay. If you try to combine a normal coil with latching relay coils in the same relay, you will get an error when you try to download the diagram to your controller.

Every Blū Logic diagram has one Master Control Relay called “MCR”. If the MCR is off, then all physical outputs will be turned off regardless of the state of associated coils in the diagram. Contacts associated with an internal relay will still operate even if the MCR is off. The state of latching relays is also not affected by the MCR. The MCR interrupts the signal between the coil and the physical circuit. It does not affect the logic of the diagram. However, you can put MCR contacts in your diagram and these *will* be opened if the MCR is turned off.

If there is no MCR coil in a ladder diagram, then the MCR is always on. If there is an MCR coil anywhere in the diagram, then the MCR will initially be off when the diagram starts scanning at power-up. None of the outputs will function until you turn on the MCR. (And you’ll call tech support and they’ll eventually figure it out and you’ll feel like a fool! It can happen to anyone.)

### **Rung Execution Order**

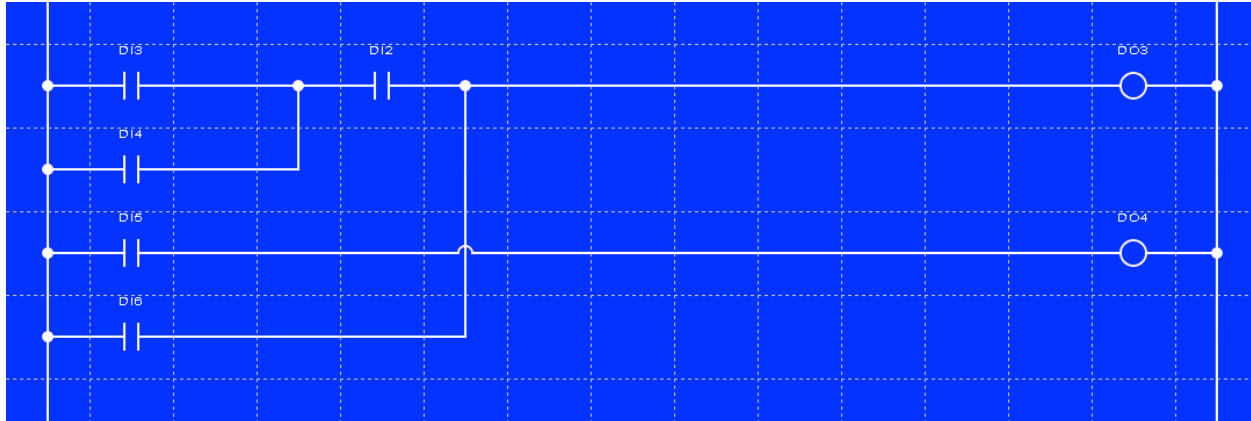
Blū Logic guarantees that rungs are executed in the order that they appear in the diagram. They are executed from the top of the diagram to the bottom. The values of input contacts and output coils tied to physical I/O points are frozen during the scan of the ladder. That is, the execution sequence is like this.

1. Read all physical inputs and set the value of contact assets (which are internal copies).
2. Scan or “solve” the ladder diagram, from top to bottom.
3. Update all physical outputs to the value of coil assets. Unless the MCR is off, in which case all physical outputs will be turned off regardless of the state of their coils.
4. Repeat. (Don’t you always wonder when to stop shampooing? Ladder diagrams never stop repeating.)

Blū Logic allows more complex interconnections in a diagram than most ladder logic systems so we need to be careful with the definition of a rung. A Blū Logic rung can have any number of connections to the power rail and the neutral rail and can have vertical and horizontal wiring that crosses without connecting. Wiring can even cross other rungs!

For purposes of the execution sequence guarantee, a rung is a collection of elements with wired connections between them. If there are no wired connections, other than through the power and neutral rails, between two groups of elements, then they are not in the same rung.

Note that an isolated section of elements meeting this definition could actually appear in the middle of another rung because all connections could cross the isolated rung without actually connecting to it. In this case, the rung with the upper-most connection to the power rail executes first, including elements above and below the isolated rung, and then the isolated rung executes. But it's best not to create such a confusing situation in the first place.



Although Blū Logic lets you build very complex rungs, this doesn't mean you always have to. It is generally good practice to keep individual rungs simple. It makes the diagram easier to understand and troubleshoot at some later date. Remember, it is common for ladder based automation systems to run for many years without any changes or program maintenance. It is entirely possible that nobody will need to work on your program until after you have retired.

With the above in mind, you only need to know two simple rules about execution order.

1. Rungs are evaluated in order from the top of the diagram to the bottom.
2. The order of evaluation within an interconnected rung is unpredictable and may even change when seemingly unrelated changes are made to the diagram.

## **Timers and Counters**

Timers and Counters are similar elements and have some common features. The big difference is that timers count by themselves and counters only count when an energized input tells them to. Each includes a register that contains the count or elapsed time. Each contains a preset register which is compared to the counter/timer register. Each includes a contact which closes based on a comparison between the register and the preset.

There are three kinds of timers in Blū Logic: "On Delay", "Off Delay" and "Stopwatch". The On Delay timer starts timing when its input is energized and closes its contacts when the timer reaches its preset. At this point, the timer stops running and the



contacts will remain closed as long as the input is still continuously energized. When the input is de-energized, the contacts instantly open and the timer resets to zero. If the input is de-energized before the preset time is reached, then the timer resets immediately and the contacts will never close at all.

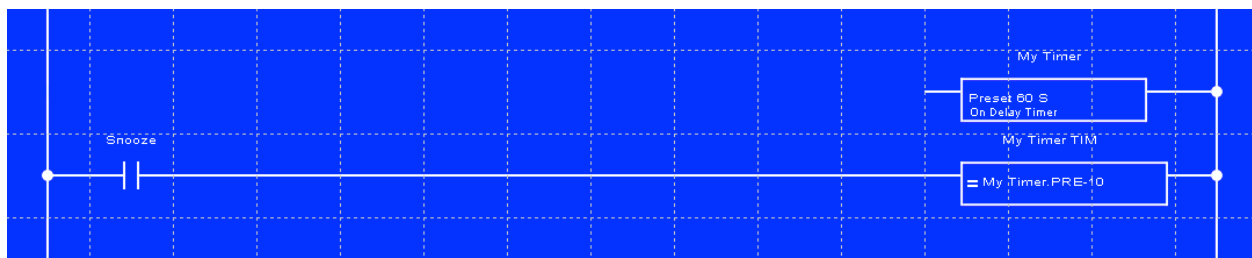
An Off Delay works the same way but the timer runs when the input is de-energized and the contacts open when the preset is reached. Any time the input is energized, the contacts are closed.

A Stopwatch timer runs when the input is energized and stops when the input is de-energized. Unlike On and Off Delay timers, the value in the stopwatch timer's register is retained between changes in the input state. The timer keeps track of the accumulated time that the input is energized, no matter how much it switches off and on in the meantime. When the preset is reached, the timer's contacts close and the timer freezes. Because the register does not automatically reset, a Stopwatch timer also has a Reset input. When this input is energized, the timer resets to zero. If one of the timer's own contacts is fed back to the reset input, the timer will reset to zero when it reaches the preset, but all contacts will close for a single scan of the ladder. That is, they will pulse.

Any timer or counter can be reset to zero using a REG= rung in the form



A REG= rung can be used to preset a timer or counter to any value at any time. A REG= rung can also modify the preset, MyTimer.PRE, of a timer or counter. Complex behaviors can be implemented using this technique. For example, a ten-second “snooze” input could be added to a timer with a rung like this



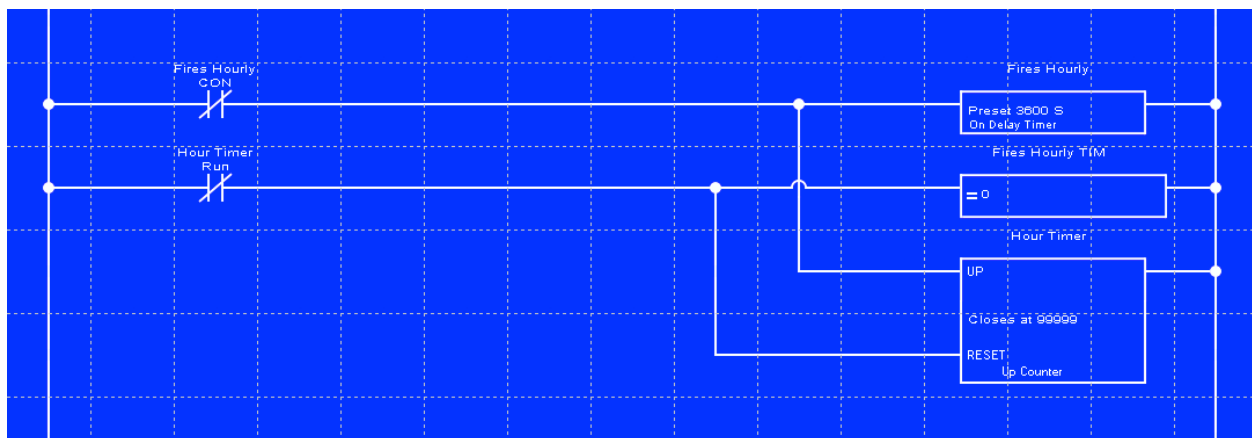
When a timer runs, it “ticks” at an one of four rates. Milliseconds, Seconds, Minutes, or Hours. The rate is chosen when the timer is created and cannot be changed by the

ladder diagram. The value in the timer's registers represents elapsed time in the unit of the timebase. A Millisecond timer running for 10 seconds will reach a value of 10,000. Timers are guaranteed not to "tick" between scans. The value of a timer changes only when the rung containing the timer is scanned (unless another rung modifies it directly with a REG= element). If the scan time is more than a millisecond, millisecond timers can advance by more than one count at a time to keep up. For this reason, be careful about comparing the .TIM register of a fast timer for equality to single value. It may skip that value. Use greater-than or less-than comparisons to know when a timer passes a certain point.

The four timebases are global to the system and are not synchronized to the input of a particular timer element. That is, an Hour timebase timer with a preset of 10 will expire in 10 hours only if it is started at an exact hour anniversary of power-up of the controller. If started at an asynchronous time, it could expire any time between 9 and 10 real-time hours of the input turning on.

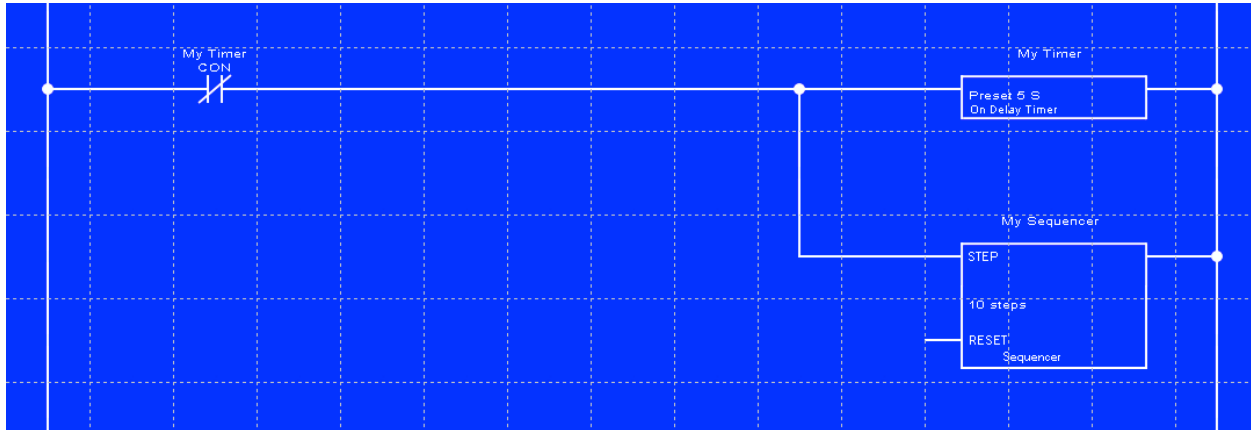
Note that, because the registers are 32 bits, the available range of values lets you preset a millisecond timer to run for days or a second timer to run for decades. So, if you need a minute or hour timer with tighter synchronization, you can probably use a second timer instead.

If you absolutely have to have something like an hour timer synchronized to the start of the timer, use a second or millisecond timer to create your own one-hour timebase and feed it to a counter.



As in the above example, one common application of a timer is to generate a periodic pulse. This can be implemented with either an On Delay or an Off Delay timer but the On Delay version is perhaps more clearly self-documenting. Connect the timer's input to the power rail through a Normally Closed contact of the timer itself. When the system starts, that contact will be closed. It will stay closed for the interval preset in the timer. Once the timer reaches the preset, it opens the NC contact, this stops and resets the timer, which closes the contact again, and the process repeats. The timer's contacts will be closed for a single scan, every time the timer expires. Any number of copies of

the timer's contact can then be positioned elsewhere in the diagram to trigger something periodically. A typical application for this trick is to advance a Sequencer by pulsing its Step input.



There are three kinds of counters. An Up Counter increments its register on each rising edge of its input. A Down Counter decrements on each rising edge. Each counter is assigned a preset value. An Up/Down counter has both Up Count and Down Count inputs and can count in either direction. All three timers have a Reset input.

Counters count on the rising edge of an input. That is, they do not count on each scan that the input is energized, only on the first scan that it becomes energized, following one or more scans of being de-energized.

An Up Counter starts at zero and counts up. When the preset value is reached, the counter contacts close. The counter freezes at the preset value and ignores additional Up count pulses. Often, a counter contact is wired to the counter's Reset input to zero the counter at this point. If this is the case, then the counter contact closes for a single-scan pulse.

A Down Counter starts at its preset value and counts down. When the count reaches zero, the contacts close. As with the Up Counter, the counter's contact can be wired to the Preset input to make it emit a single scan pulse and automatically recycle. Otherwise, the counter will freeze at zero with the contacts remaining continuously closed. The Preset input to a Down Counter causes the preset to be loaded into the counter. To un-freeze the counter, energize its Preset input or load some other non-zero value into the register with a REG= element.

Although the Up and Down count inputs of counters are only sensitive to rising edges, the RESET input of an Up Counter and the PRESET input of a Down Counter constantly respond to the state of the input. If you hold RESET active, the counter register will stay at zero even if there are also pulses on the count input.

An Up/Down Counter has the same behavior as an Up counter but has an additional input that makes it count down. The behavior of the contacts is the same as the Up

Counter. The contacts are closed when the count register reaches the preset. The Reset input to an Up/Down counter sets it to zero. Sometimes you need a contact which closes when the Up/Down counter is at zero, as well as a contact for the preset. You can use an IF element to create an auxiliary contact for any timer or counter. In this case, you would use IF MyUpDownCounter.CNT=0.

Another way to directly manipulate a counter's CNT register is with a REG= element. You can use REG= to preset a counter to any value at any time. If that value is greater than or equal to the counter's preset, then the counter's contacts will close.

Modifying a counter's CNT register with a REG= does not instantly change the contacts but the contacts will update to be consistent with the new register value the next time the rung containing the actual counter element is scanned.

## **Shift Registers**

A shift register holds a train of 1 and 0 bits. You set the length of the train when you click on [New Shift Reg] in the Asset Palette. The maximum length is 32 bits.

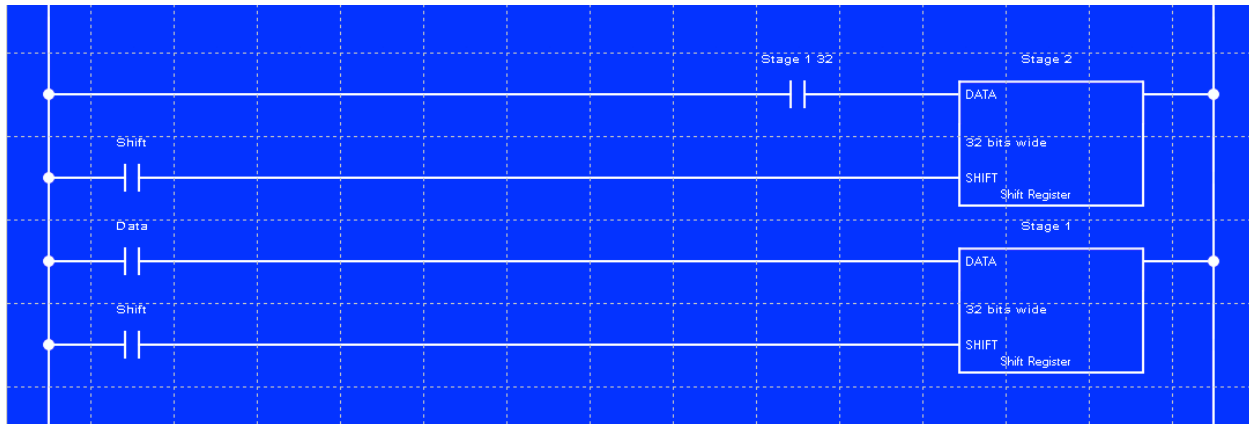
For each bit in the shift register, there is an associated contact. These are identified by the name of the shift register, and a number, such as MyShifter.01, MyShifter.02, and so on up to the number of bits you specified when you created the shift register.

Each shift register has a Data and a Shift input. Each rising edge of the Shift input will cause all of the bits to move over one space. Bit 1 will replace bit 2, bit 2 will replace bit 3, and so on. The state of the Data input at that time determines the new value of bit number 1. If the data input is energized, a 1 will be inserted. Otherwise, it will be a 0. The bit that shifts out the other end is lost. You can cause the shift register to recirculate by wiring a contact for the last bit to the data input. Now the bits can be thought of as going around in a circle.

Shift registers are commonly used to track the state of a workpiece in a pipelined series of operations. If an inspection operation determines that a bottle is broken, for example, a later process step should not fill it with product and a still later step should push it off the line into a reject chute. To implement this, the Data input of a shift register is controlled by the output from the inspection step. The Shift input is pulsed once for each movement of the line. Downstream steps, such as the fill station and the scrap diverter, can refer to the appropriate contact, corresponding to how many steps they are down the line from the inspection step.

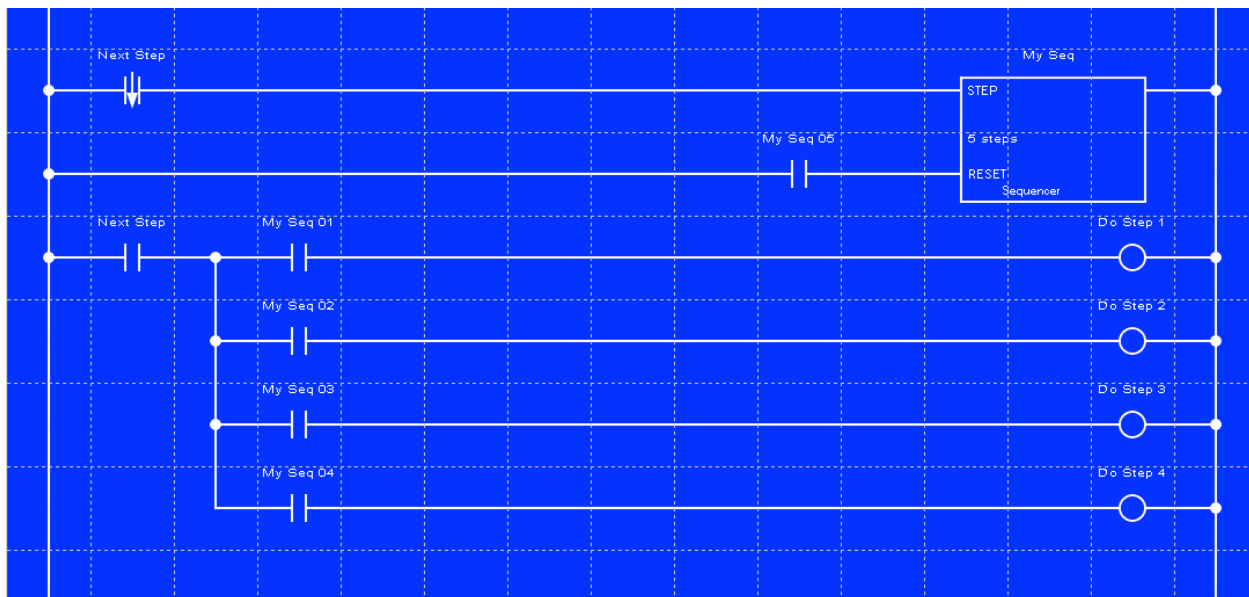
The maximum size of a shift register is 32 bits. If you need a longer register, you can connect two or more together. All Shift inputs should be pulsed together and the .32 contact of each shift register should be wired to the Data input of the next. The shift registers must be shifted from the far end to the front. In other words, the stages should

appear in the ladder diagram from the first stage at the bottom up to the final stage above all others.



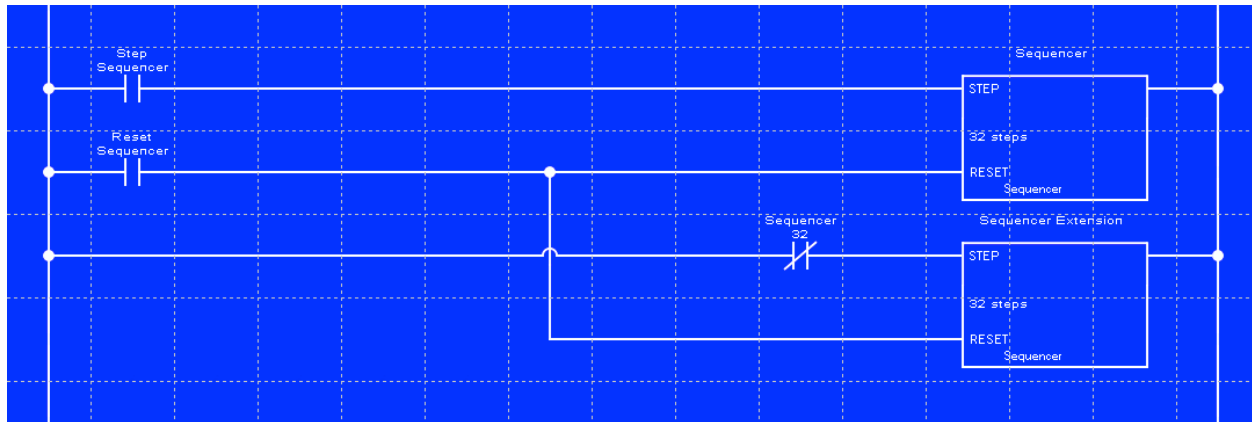
## Sequencers

A sequencer can have up to 32 contacts and it closes them one at a time, in turn. At power-up or when the Reset input is energized, contact MySequencer.01 is closed and all others are open. Each rising edge of the Step input opens the current contact and closes the next one. A sequencer also has a Reset input that sets it back to the initial condition, with contact .01 closed. If you wire the highest-numbered contact to the Reset input, then the sequencer will repeat endlessly. A sequencer can be used to control a step-by-step process. Each of the sequencer's contacts activates one step of the process when it closes.



Like a shift register, a sequencer can be extended beyond 32 contacts by daisy-chaining two or more together. If wired as shown below, output 2 of the Sequencer

Extension will activate on the step after the last output of the upper sequencer, Sequencer.32



### IF and REG=

To load a value into any register, whether it is a stand-alone holding register, part of a timer, counter, shift register, or sequencer, or associated with an analog output, use a REG= ladder element. REG= evaluates a formula, explained below, and sets the specified register to the result. The assignment only happens if the logic input to the REG= element is turned on. The assignment occurs on every scan in which the input is energized. That is, it is *not* edge triggered.

Besides performing math, a REG= element can be used to add asynchronous resets, additional count inputs, and other advanced functions to timers and counters as mentioned in the Timers and Counters section.

To test a register value, use an IF element. Like REG=, an IF element includes a formula. In this case, the formula is evaluated when the element's logic input is on. The result is compared to the value in the specified register. If the comparison is logically true (and the input to the element is on), then the element's output will turn on. Turning off the input will turn off the output regardless of the value in the register.

IF elements referenced to the .TIM register of a timer can be used to create signals at intermediate times.

Capiche?

## **Mathematical Expressions (Formulae)**

The REG= and IF ladder logic elements require what computer scientists call an expression, and the rest of us call a formula, to be entered. In the REG= case, when the expression is evaluated, the register is set to the result of the calculation. In an IF element, the result is compared to the contents of the specified register.

Expressions consist of terms combined with operators. Terms may be numerical literal constants, like 42, or register names. Operators are + - \* / and % for integer math or & | ^ and ~ for bitwise boolean operations.

Register names are strings of characters, words. A register name has to start with a letter A through Z but may also contain digits 0 through 9, the underscore \_, #, and spaces. The maximum length of a register name is 20 characters. The names are not case-sensitive and any space characters are there only for readability and are not significant otherwise. That is, the register name “My Register” refers to the same register as “myregister” or even “myREGls t e r” Just to be clear, all other characters including the underscore are significant. “My\_Register” is not the same as “MyRegister” or “My Register”.

When you create a timer or counter, the system will make up some register names. These names always contain a period “.” so you won’t accidentally create your own register name which matches a system-generated name. For example, each timer has two registers, MyTimer.TIM, which is the actual timer, and MyTimer.PRE, which is the preset register.

Constants can be either decimal or hexadecimal. A decimal constant is simply a string of the digits 0 through 9. A hex constant is introduced by a \$ and can include 0 through 9 and A through F digits. For example \$2A has the same value as 42.

Operators follow a pretty standard precedence order. Multiplication \* is performed before addition + so that  $2+3*4$  results in 14 and not in 20. The precedence can be explicitly controlled by using parentheses. A sub-expression in parentheses will be evaluated first so  $(2+3)*4$  is equal to 20. Operators with equal precedence are evaluated from left to right.

### Operators in Precedence Order

( )	Subexpression in parentheses
- ~	Unary negation, Boolean NOT
* / %	Multiplication, Division, Modulus
+ -	Addition and Subtraction

<< >>	Bitwise shift left and shift right
&	Bitwise logical AND
^	Bitwise logical exclusive OR
	Bitwise logical inclusive OR

Expressions are limited to 100 characters. If this is not enough, you'll have to use shorter register names or break your expression up into multiple REG= rungs. If you have spaces in your register names or between operators, you can take these out to make the most of the 100 character limit.

The values in registers are signed 32 bit integers. So their range is limited to about plus and minus two billion. All math is also performed in 32 bits.

Division by zero does not cause an error but results in either the maximum possible value 2,147,483,647 or the minimum possible value -2,147,483,648 depending on the signs of the operands.

Division is the slowest math operation. Division takes 25 times as long as multiplication so a lot of division operations can quickly increase the scan time. Modulo uses division, so it is also slow. An exception to this rule is division by a constant power of two, such as 16, 256, 512, etc. Blū Logic recognizes this special situation and implements the division as a faster shift operation. Division by a variable value, such as a register or a subexpression involving a register, will always be slow regardless of the value because Blū Logic cannot predict the value of the register in advance.

### **Analog Inputs and Outputs**

If your controller has analog input and/or output points, they will appear as registers in the IF tab of the Asset Palette and can be used in expressions. Analog output points appear in the IF and REG= tabs. You can't set the value of an analog input, so they do not appear under REG=.

Analog I/O points are commonly 10 or 12 bits. This means the range of possible values is 0 to 1023 for a 10 bit point and 0 to 4095 for a 12 bit point. A twelve-bit zero to five volt input, for example, will have a value of 0 at 0 volts, of 4095 at 5 volts, and 2048 at 2.5 volts.

All Blū Logic registers are 32 bits but the unused bits of the register corresponding to an analog I/O point will always be zero.

See the specifications for a specific controller model for the designations and ranges its analog I/O points.



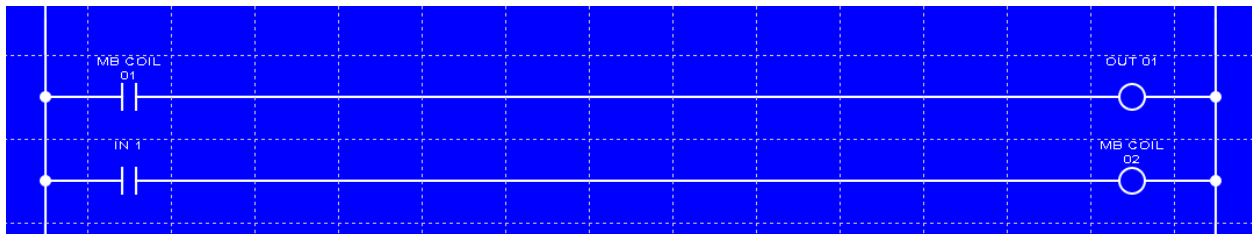
## Modbus Communications

If your custom Blū controller includes a Modbus serial port, then it will respond to the Modbus RTU protocol to read and write relays and registers. This facility is often used to allow other PLCs, computer systems, or Man-Machine interfaces to interact with your controller.

The Modbus architecture supports two kinds of data: bits (called “coils” in Modbus terminology and “relays” elsewhere in this document) for on/off values and registers for numeric values.

In most small PLCs with Modbus support, Modbus messages from the Modbus master device directly access the I/O points and registers of the ladder diagram. Blū Logic takes a different approach for several reasons. Blū Logic provides dedicated Modbus bits and registers for the Modbus master device to access. The ladder diagram can write values to registers or bits for the Modbus master to read or it can refer to Modbus registers and bits which are written by the master.

The Modbus master therefore can only access information that the ladder diagram is specifically intended to expose to the outside world. Physical I/O, internal relays, and registers cannot be affected by Modbus messages unless the ladder diagram explicitly provides a path from one to the other.



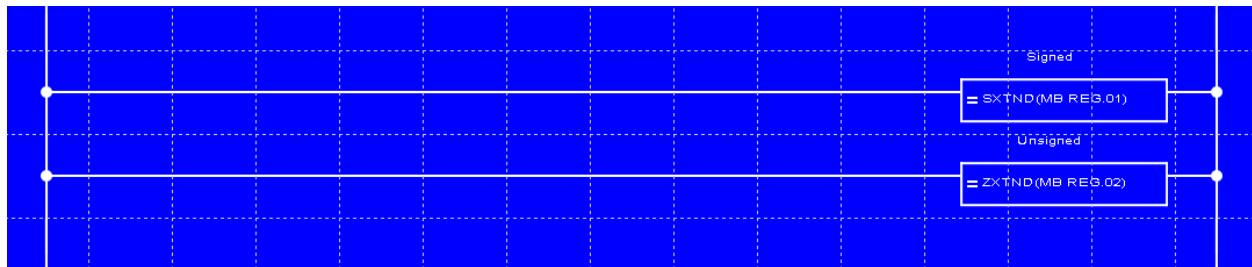
Blū Logic provides Modbus 0x coils and Modbus 4x registers. Modbus commands which operate on 0x or 1x values will access the coils. Modbus commands which operate on 3x or 4x registers will access the registers.

Modbus register values are only 16 bits long. They can be treated as signed values from -32,768 to +32,767 or as unsigned values from 0 to 65,535. Blū Logic registers and computations support 32 bit values. This means some care must be taken when passing numbers back and forth between a Blū Logic ladder diagram and a Modbus device.

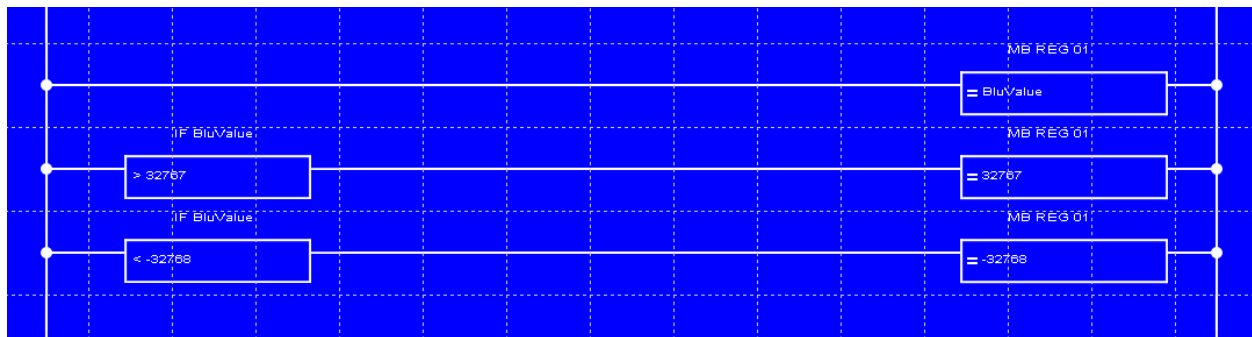
Note that, as long as you only deal with values within the unambiguous range of 0 to 32,767, you don't have to worry about the following.

If a Modbus register is referenced in a Blū Logic expression, such as in an IF or REG= element, then the Modbus value is assumed to be unsigned. If the Modbus value is

actually signed, this will result in unexpected behavior. For instance, -23 written to a Modbus register and then copied to a Blū Logic register would come out as 65,513. Blū Logic provides a math function to solve this problem. The function SXTND for “Sign eXTeND” converts a signed Modbus number to a 32 bit, signed Blū Logic value. There is also a “Zero eXTeND” function, ZXTND, which does the same thing for unsigned Modbus numbers. If you don’t use either function, the behavior is as if you used ZXTND.



Going the other way from a 32 bit Blū Logic register to Modbus can also cause surprises. Because the 32 bit number must be truncated, values outside the range which can be represented by 16 bits will change. If a Blū Logic register with the value of 100,000 is written to a Modbus register, the value seen by the Modbus device will be either 34,464 if the Modbus device treats it as unsigned or -31,072 if the Modbus device treats it as signed. As a precaution, you may want to limit the range of values before writing them to Modbus registers.



## **Controller Startup**

When your controller is powered up, or when the USB cable is unplugged after downloading a diagram, an initialization is performed before the ladder begins to scan.

- All latching relays are turned off.
- All Timers are set to 0.
- All Up and Up/Down Counters are set to 0.
- All Down Counters are set to their Preset value.
- All Shift Registers are filled with zeroes.
- All Sequencers are reset and thus their number 01 contact closes.
- If there are any MCR coils in the diagram, the MCR is turned off. Otherwise, it is turned on.
- All other registers are set to 0.
- The special contact RESET is closed. It will open at the end of the first scan and never close again until the power is cycled.



## ***Integer Math***

Blū Logic and the controllers it is compatible with support only integer math. To the uninitiated, this may seem like a major deficiency and to preclude working with fractional values.

In fact, there are advantages to integer math and there are simple techniques for handling fractional values in an integer system. Floating-point math may seem to be more precise, but this is not necessarily true and the fact that it seems true can lead to dangerous assumptions. The technique of performing math on fractional values using only integers is known as “fixed point.”

Suppose you want to represent a value in the range of 0 to 100%. You wish to be able to represent values which are a fraction of a percent. For clarity, let's say that you want to represent values between 0 and 1, since this is equivalent, and that you want the maximum possible precision.

To represent this range of values in floating point is easy. 0 is 0, a tenth is 0.1, half is 0.5, 90% is 0.9 and 1 is 1. If you want to, for example, set the value of a 10 bit analog output from this value, you multiply it by 1023.

Let's represent this same range in integers with 8 decimal places. So we map the range of 0 to 1 onto the range of integers from 0 to 100,000,000. 0 is 0, a tenth is 10,000,000, half is 50,000,000 and 90% is 90,000,000. If you want to set the value of a 10 bit analog output, you divide by 97,751 (i.e.  $100,000,000 / 97751 = 1023$ ).

The integer representation is superior for several reasons. First, computations can be much faster, resulting in a faster overall scan time. Second, you may be surprised to know that certain “easy” values, such as 0.1 and 0.9, cannot be precisely represented as a binary fraction. Just as a third cannot be precisely represented by a precise decimal fraction because it is an infinite repeating decimal of 0.333333..., the binary representation of a tenth is a repeating binary fraction and thus cannot be precisely represented no matter how many bits you have. Third, the integer representation is *more* precise. IEEE 32 bit floating point format only uses 24 bits to represent the mantissa (the significant digits). Other bits of the 32 bits of storage must be reserved for the sign and the exponent. A 32 bit integer uses all 32 bits for the mantissa. In short, the precision of representing 0 to 1 in floating point is 1 part in 16 million (approximately 7 decimal places) whereas the precision of representing 0 to 1 in the integer form described above is 1 part in 100 million (8 decimal places). The available precision of a 32 bit integer is a little more than 9 decimal places.

Floating point is valuable if you need to represent very large numbers and very small numbers or when the range of possible values is not known in advance. But in a control system, we are more likely to know the possible range of values and we are more

interested in precision, speed, and predictability. Relying on floating point math can hide anomalies that the control system designer really should be aware of.

Blū Logic provides 32 bit integer math. Most small PLCs only supply 16 bits and this is adequate for the majority of control problems. PLCs that do provide floating point normally use a 32 bit format, commonly IEEE-754 single-precision format.

All-integer representation of process values is common. In fact, it's so common that many off-the-shelf man-machine interfaces have built-in capabilities to scale operator inputs and outputs to integer values.

The easy way to think about it is this: figure out how many decimal places you need, make sure you always have that many places even if some on the right are zeros. Got that? OK, now just take away the decimal point. Easy.

Suppose you want 3 decimal places. Just multiply every number in the system by 1,000. You have to be careful when you multiply two such scaled values in an expression, because the product of two numbers each scaled by 1,000 will be scaled by  $1,000 * 1,000$ . So you have to divide the product by 1,000 once for each multiplication to keep the scale factor at 1,000.

For example, suppose you choose a scaling factor of 1,000 as suggested above. The the quantity "three and a quarter" would be represented as 3,250 and the quantity "one half" would be 500. If you multiply these two numbers together, you want the result to be "one and five-eighths" which should be represented as 1,625. But the product of two quantities scaled by 1,000 will be scaled by 1,000,000 (1,000 times 1,000). That is, the result of the example will be 1,625,000. You can fix this by dividing by 1,000 to remove the extra scaling factor.

You have to be careful that the intermediate value, the product before the division, does not exceed the range of 32 bit integers, -2,147,483,648 to +2,147,483,647. You may think that you do not have to worry about this in floating point calculation, but the truth is that if your intermediate product exceeds 24 bits of significance, the binary fraction will be truncated resulting in a loss of precision *which is hidden from you!*

Don't just blindly convert every multiplication into a multiplication and a division. Think about the scale factor on a system level. You may find that you only have to perform one division, or even none, depending on the ranges of inputs, outputs, and constants. Try to combine all of the constant scaling factors, such as the conversion of an analog input to engineering units, the fixed point scale factor, and the conversion from engineering units to a value for an analog output into a single operation. For example, you might multiply the analog input reading by a constant, perform all calculations in the scaled units, divide the final result by a second constant, and end up with a suitable value for the analog output.

The above discussion is very decimal-centric. But the hardware processor in your controller works in binary. The scale factor does not have to be a power of 10. If you choose a scaling factor that is an integer power of two, then the math operations can be performed more quickly. Dividing by 1024 is about 10 times faster than division by 1000. Some of the fastest numbers to divide by are 65536, 256, 131072, 2, and 4.

For more information on this subject, check out the Wikipedia articles on “Fixed-point Arithmetic” and “Binary Scaling” or Chapter 9 of “Embedded Systems Building Blocks, Second Edition” by Jean J. Labrosse.





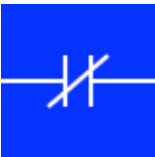
## Reference

### Blū Logic Elements

#### Contacts



Normally open contact. Can represent a hardware digital input, the current state of a digital output, an internal relay, or a discrete output from one of the built-in peripherals.



Normally closed contact. This contact is closed when the corresponding input or coil is de-energized and open when the input is energized.



Rising edge detector. The output of this element is true for one scan of the ladder after a false-to-true transition of the associated coil or physical input. The effect is a pulse for each rising edge on the input.



Falling edge detector. The output of this element is true for one scan of the ladder after a true-to-false transition of the input.

#### Coils



Standard coil which can represent a physical discrete output, an internal relay, or a discrete input to a built-in peripheral. If this coil is present on a relay which also has SET, RESET, or TOGGLE coils, unexpected behavior may result.



SET coil. A true signal to this coil latches the relay on. It will stay on until a RESET or TOGGLE coil (with the same name) is activated.



RESET coil. A true signal to this coil latches the relay off.



TOGGLE coil. On the rising edge of a true signal to this coil, the relay will change state.

### Timers

- On Delay** An On Delay timer starts running when the input becomes true. When the time reaches the preset, the output comes on. The output stays on until the input goes false again. If the input goes false before the timer reaches the preset, it stops and is reset back to zero immediately. The timer register and the preset value can be manipulated by REG=rungs.
- Off Delay** An Off Delay turns on its output when its input becomes true. When the input goes false again, the output stays on. The timer starts running. When the preset is reached, the timer output turns off.
- Stopwatch Timer** A Stopwatch Timer, also called a Retentive Timer, runs when its input is on and freezes when the input is false. Unlike the other timers, it does not reset when the input turns off. The Stopwatch Timer output comes on when the preset is reached and stays on until the timer is reset.

### Counters

- Up Counter** Each rising edge of the input causes the counter register to increment. The counter output turns on when the count is equal to or exceeds the preset. The counter can be wired to reset itself at the preset value.
- Down Counter** Each rising edge of the input causes the counter register to decrement. The counter output is on when the count is zero. The counter can be wired to load itself with the preset when it reaches zero.
- Up/Down Counter** An Up/Down Counter is just like an Up Counter but with an additional input to make it count down. The behavior of the output contact and the reset input are the same as an Up Counter.
- Shift Register** A shift register is a train of up to 32 bits that move over one space on each rising edge of the SHIFT input. The RESET input sets all bits to zero. The shift register can be wired so that words shifting out the end are recirculated back to the beginning.
- Sequencer** A sequencer has up to 32 contacts that energize one at a time. Contact 01 is on initially or when the Sequencer is reset. Each rising edge on the STEP input cause the current contact to open and the next to close. A sequencer can be made to repeat by wiring the highest-numbered output

to the reset input.

- REG=** This element sets the value of a register to the result of a specified formula. The formula can combine constants and register values including the registers in timers, counters, and embedded peripherals. The available operators are addition, subtraction, multiplication, division, and modulus plus boolean operators for AND OR and XOR. The register to which the result is transferred can be an internal register or can be the timer or preset of a timer or the counter or preset of a counter. REG= is a very powerful tool, A register can be assigned with the result of a formula involving itself. A counter could be implemented by a REG= rung like “MyCounter = MyCounter+1”. A REG= element has an input and the assignment operation occurs on each scan in which this input is on.
- IF** An IF element compares a register, either a pure working register, one of the registers of a timer or counter, or an analog input. The number in the register is compared to the result of a formula. Any of the usual compare operators can be used, = != < > <= or >=. The formula can combine constants and other register values including the registers in timers, counters, and analog inputs and outputs. The available operators are addition, subtraction, multiplication, division, and modulus. The output of an IF element is energized if the input is energized and if the result of the comparison is true. The IF formula does not support logical operators such as AND and OR but these are easily implemented by combining multiple IF rungs in the ladder diagram.

### **Special and Shortcut Keys**

- ESC** The Escape key clears the current tool. If no tool is active, the key de-selects any selected cells of the ladder. Typically, you will hit the ESC key after inserting several identical elements or at the end of wiring connections between elements with the screwdriver tool.
- ARROWS** If you select one cell of the diagram by left-clicking, you can then move that selection around with the keyboard arrows. This functionality is most often used in conjunction with clicking the Asset Palette to add elements to the diagram.
- BACKSPACE** Moves to the left one space and erases the contents of that cell. The Backspace key is designed to be used when you are entering logic by clicking assets in the Asset Palette. It is equivalent to hitting the left arrow followed by the DELETE key.
- SPACE BAR** When a cell is selected, or a tool that represents a contact or a coil is active, you can hit the SPACE BAR to switch between the variants such as

NO, NC, RISE, and FALL contacts. You can change more than one at a time by selecting multiple elements and hitting the SPACE BAR.

- CTRL** If you hold down the control key while you click to select, your previous selection is retained. In this way, you can build up a disjoint selection.
- CTRL-C** Copy selected cells.
- CTRL-V** Paste a copy of cells from the preceding Copy or Cut operation.
- CTRL-X** Cut (copy and then delete) selected cells.
- CTRL-Z** Hold down Control and type Z to undo the last editing operation.
- TAB** Used to move from one field to the next in a dialog (this is standard Windows behavior). Hold down SHIFT while you TAB to move backwards.
- DEL** The DELEte key erases the contents of selected cells. It does the same thing as the wire cutter tool. But the wire cutter can only clear one cell at a time. You can click and drag the mouse to select a range of cells and then DELEte them all with a single key stroke.
- SHIFT** If you hold down SHIFT while you DELEte, the state of your automatic rip-up preference is reversed. That is, if you have auto-rip enabled and wires are normally ripped up when you delete, holding down shift will NOT rip up wires outside of your selection. If you have auto-rip disabled in the preferences dialog, holding down SHIFT while deleting will temporarily enable auto-rip up.
- RIGHT-CLICK** (All right, so it's not really a key). If you right-click on a timer, counter, shift register, REG=, or IF element in the diagram, you can modify it. You can also right-click on any of these assets in the Asset Palette to modify or rename them.

## Execution Time of Operations and Elements

The following table will give you an idea of the amount of time each element adds to the scan time of a diagram. This reference is intended to help you choose between alternate implementation strategies, and is not completely precise in all cases. For very small diagrams, this table will under-estimate the scan time. For a large diagram with a lot of math, it will over-estimate scan time. Scan time can also vary slightly from one scan to another, especially if serial communication is in use.

Because not all controller designs run at the same speed, these times may vary between models. However, the relative speed of these operations is the same across the entire family, at the time of this writing. No matter how fast a regular coil executes, it is always twice as fast as a toggle coil, for example.

Connection to Power	65 nS	Any Contact	125 nS
Coil	190 nS	Set Coil	125 nS
Reset Coil	125 nS	Toggle Coil	430 nS
Wiring tee	125 nS	On Delay Timer	3 uS
Off Delay Timer	3 uS	Stopwatch Timer	3 uS
Up Counter	3.1 uS	Down Counter	2.6 uS
Up/Down Counter	4 uS	Shift Register	1.2 uS
Sequencer	1.5 uS	IF	375 nS
REG =	250 nS	Register name in expression	125 nS
Constant	125 nS	Parenthesis	0 nS
+	125 nS	- (Negation or subtraction)	125 nS
*	1.2 uS	/	30 uS
%	30 uS	<<	4.2 uS
>>	4.2 uS	&	125 nS
	125 nS	^	125 nS
Unary ~	125 nS		

nS = nanosecond = 1/1,000,000,000 seconds.

uS = microsecond = 1/1,000,000 seconds.



# **BLÜ**

## Appendix

This appendix provides supplemental information about your particular controller model: Blü Logic Demonstrator





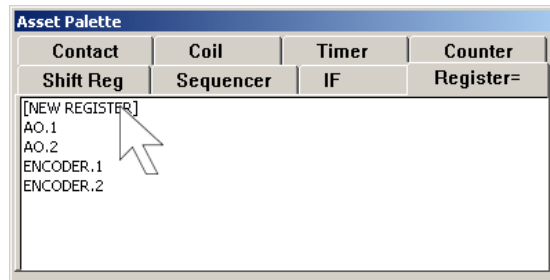
## Blū Logic Demo PLC User Interface Addendum

This document is a supplement to the Blū Logic user reference manual. You should have read that manual and will need to be familiar with the Blū Logic program in general before this material will make sense.

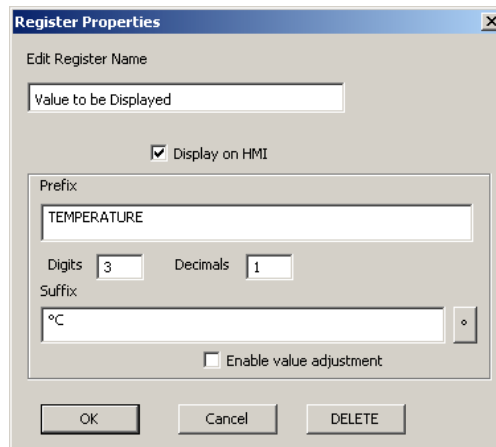
Your Blū Logic demonstrator includes a user interface comprising an LCD display and a pair of knobs. The display can show up to four values at a time. The left knob selects a value to be adjusted and the right knob adjusts that value up or down. If more than four values are configured for display, the left knob will also scroll the display to reveal the other values.

Values on the display come from Blū Logic registers. Registers can be displayed only or can be made adjustable by the user. It is good practice to avoid modifying, in the ladder diagram, values that the user can adjust as the resulting interaction can be confusing.

Ensure you have a register for each value to be displayed or adjusted. As explained in the Blū Logic instruction manual, you can create a register by clicking on the [NEW REGISTER] entry in the **Register=** tab of the Asset Palette window.



To configure a value to be displayed, right-click on the register name in the **Register=** tab of the Asset Palette to bring up the register properties dialog.




Check the **Display** on HMI box. Fill in the Prefix, Digits, Decimals and optionally the Suffix field. They work as follows:

**Prefix** is text that appears to the left of the value on the display. Normally, this is a label for the value like “TEMPERATURE”.

**Digits** sets the total number of spaces that the number will occupy on the display. This number includes all digits, both to the left and the right of the decimal point, one for the decimal point itself, if Decimals is non-zero, and one space for the sign of the number if it becomes negative. For example, 4 Digits could represent any of the following

9999  
-999  
9.99  
-9.9

**Decimals** sets the number of decimal places. All Blū Logic values are integers. So when you set a number of decimals, the decimal point is inserted into the integer value. For example, if you set 3 decimals and the value in the register is one-thousand, then you would see 1.000 on the display. As explained above, the number of decimals and one for the decimal point are included in the number of Digits.

**Suffix** specifies text that appears to the right of the displayed value. It is typically used for units such as “mm”, “V”, etc. The  button on the dialog allows you to enter a degrees symbol for a temperature unit such as °C.

The width of each line of the display is 20 characters. Thus, the total of the length of the prefix, Digits, and the length of the suffix must not exceed 20.

To allow the user to adjust a displayed value, check the **Enable value adjustment** box in the register properties dialog. When the diagram is scanning, rotating the Select knob will move the underscore cursor through the digits of each value. Rotating the Adjust knob will increase or decrease the value at the selected digit.

## DEMO PLC SPECIFICATIONS

<b>CPU Specifications</b>	
Program Memory <sup>1</sup>	16K Instructions Approx 2,500 minimal rungs
Register Memory <sup>2</sup>	Up to 1,000 registers, 32 bits each
Internal Coils <sup>2</sup>	Up to 10,000 coils
Scan Time <sup>3</sup>	< 1 millisecond typical
Math	32 bit signed integer
Timer Resolution	1 millisecond, 1 second, 1 minute, 1 hour
<b>Supply Voltage</b>	10 to 32 VDC
<b>Supply Current</b>	25 mA max all outputs and relays off
<b>I/O</b>	
Digital Outputs	4 Sourcing outputs. Supply-1/2V at 2 amps
Output Current <sup>4</sup>	2 amps per output up to 4 amps total
Digital Inputs	4 Sinking inputs
Input Voltage	5 to 30 VDC. Load < 10mA at 24VDC
Analog Inputs	2 each 0-5 VDC, 10 bit. Load > 40KΩ
Analog Outputs	2 each 0-5 VDC, 100 mA, 12 bit
Relay Outputs	2 each Form C, 10A AC or 8A DC
<b>User Interface</b>	4 line by 20 character monochrome LCD and two encoder knobs
<b>Programming</b>	Blū Logic Ladder Language IDE
System Requirements	Windows PC with USB port
Language Elements	Logic, Timers, Counters, Shift Registers, Sequencers, Integer Math
<b>Mating Connectors</b>	Phoenix Contact MSTB 2,5/XX-S-5,08 Power: 2-pole MSTB 2,5/2-S-5,08 I/O: Any combination of 24 total poles.
<b>Environmental</b>	-20 to +70 °C 5 to 95% RH, Non-condensing
<b>Dimensions</b>	135 x 80 x 30 mm, not including mating connectors

<sup>1</sup> “Minimal rung” defined as one NO contact and one normal coil.

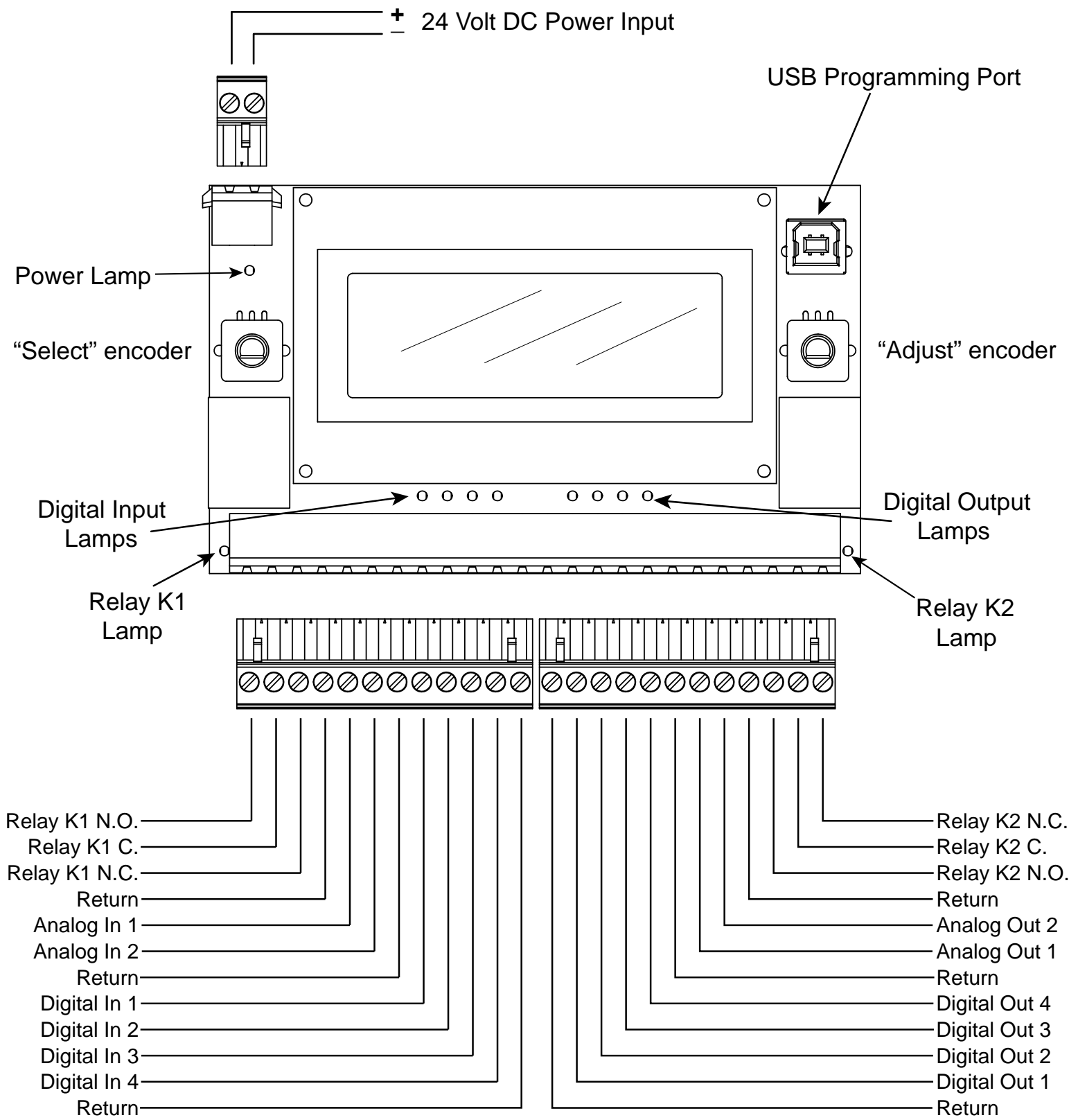
<sup>2</sup> Registers and internal coils share memory so limits interact.

<sup>3</sup> Scan time guaranteed < 1 mS for any mix of logic but no register operations.

<sup>4</sup> Derate to 1 amp when supply < 16V and temperature > 50°C

Math functions add, subtract, multiply, divide, modulo, negate  
and bit-wise NOT, OR, AND, XOR

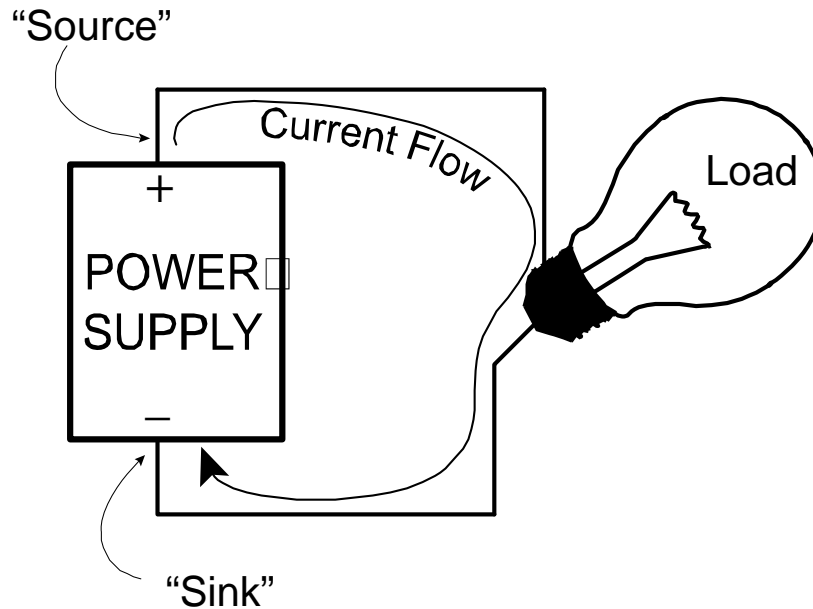
Blū Logic programming software manual available on request.  
Specifications subject to change without notice.



Relay contacts are isolated.  
 All other I/O returns are connected to negative pole of 24V power input.  
 USB port is also referenced to 24V power return.

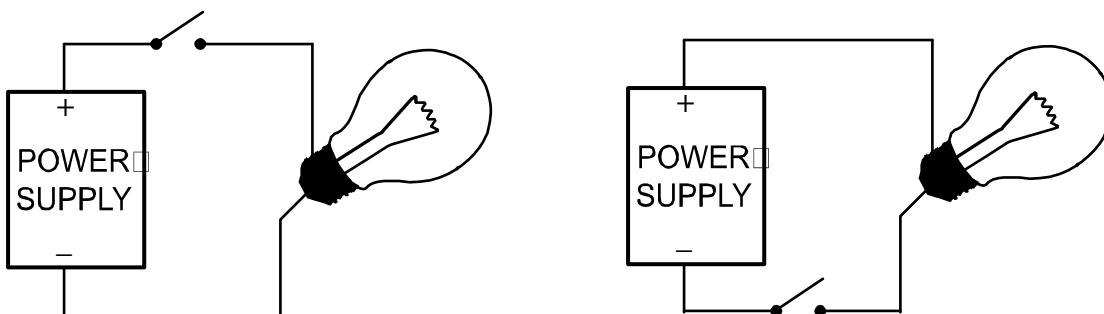
## Sourcing and Sinking

For an electrical current to flow, there must be a complete, closed path from the source, such as the positive terminal of a power supply or a battery, to the sink. In the case of the power supply or battery mentioned above, the sink would be the negative terminal. The negative terminal of the power supply in a control system is often called the “return” “common” or “ground” (ground is the least correct of these terms but probably the most used.)

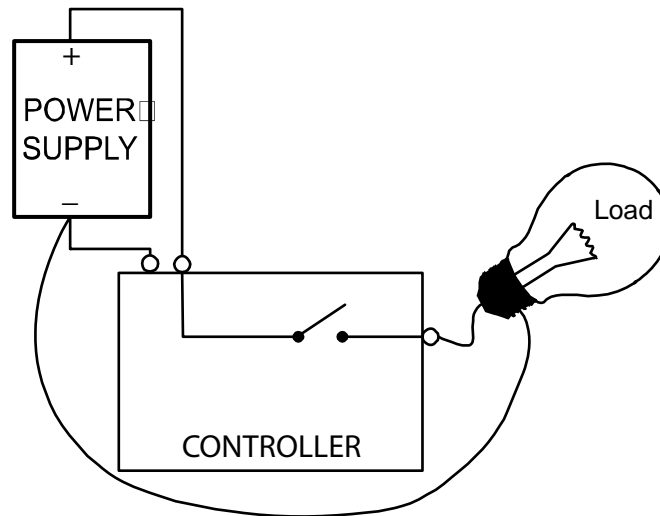


Consider the case of a controller output driving a load such as a lamp, relay coil, solenoid, motor, or maybe the input of another electronic device. Inside the controller output is some kind of a switch, usually a transistor, sometimes an electromechanical relay. The switch is opened to break the circuit when the load is to be turned off and the switch is closed to complete the circuit when the load is to be turned on.

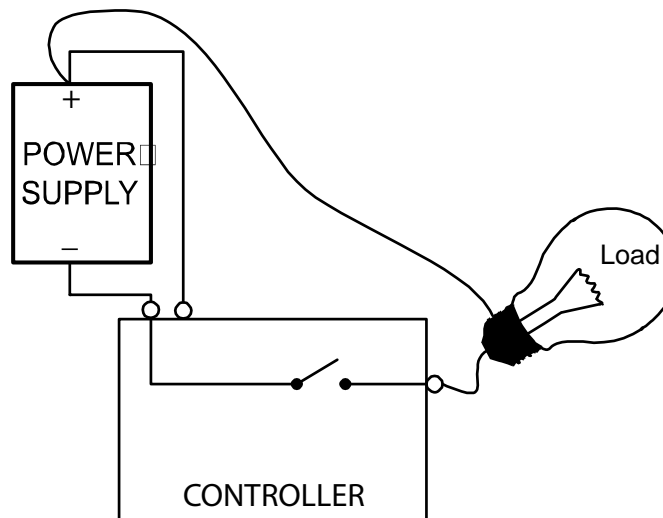
The switch could either be located between the power source, i.e. the positive terminal of the power supply, or between the load and the power sink. The former case is referred to a “sourcing” output and the latter is called a “sinking” output.



In the case of most Blu Logic controllers, a sourcing output has the positive side of the switch connected to the positive terminal of the power supply inside of the controller. Thus, a load is connected between the output terminal and the return (aka ground) terminal of the power supply. For convenience in wiring, extra return terminals are often provided in the same connector or terminal block as the output terminals.

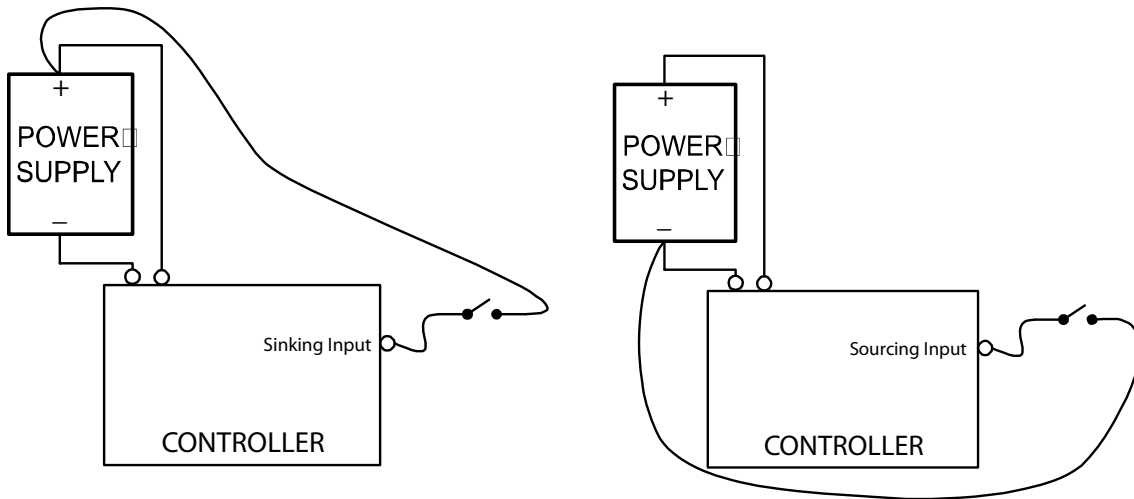


A sinking output requires a switch between the negative side of the load and the sink. In Blu Logic controllers, the negative side of the switch in a sinking output is connected to the negative power supply terminal within the controller.



Inputs can also be sourcing or sinking. A sourcing input has an internal path to the positive power supply terminal. Thus, to complete the circuit and turn on the input, the external wiring needs to connect the input terminal of the controller to return.

A sinking input completes the path to the return, or negative supply, inside the controller. To activate a sinking input, the external wiring needs to connect the input terminal to the positive power supply.



*Don't read this paragraph if you're already confused:* The source may be negative and the sink positive and the terms would still be valid. This alternate “positive ground” convention is rarely encountered in industrial automation or indeed most modern electronics. It's rare enough that few explanations of the concepts of sourcing and sinking even mention it. In this document, we have assumed that the source is always the positive terminal of the power supply. This nearly universal convention is called “negative ground.”

Many industrial sensors, such as photoelectric sensors, proximity sensors, temperature switches, etc are referred to as either “NPN” or “PNP”. This is because the outputs are often constructed with transistors and an NPN transistor is suitable for a sinking output where a PNP transistor is more suitable for a sourcing output. All you really need to know is that PNP is another name for sourcing and NPN is another name for sinking.

One final note. When you connect an output to an input, they need to be of opposite types. A sinking input is activated by a path to the positive supply. Thus, a sinking input can be driven by a sourcing output. A “PNP” sensor has a sourcing output so it would be connected to a sinking input on a controller.

If you connect a sinking output to a sinking input, you will form a complete circuit from the return back to the return again. There is no power source in the circuit, so no current will flow and the input will not be activated.

Analogously, a sourcing input would need to be driven by a sinking output.