ECET 4530

Industrial Motor Control

Introduction to
Ladder Logic Programming II
(in the RSLogix environment)

1

Tags

When placing a **new instruction** within a ladder logic program, one (or more*) instruction parameters may need to be defined in order for the instruction to function properly.

For example, the icon for a newly-placed **XIC** (**Logic Instruction**) will appear as:

where the "**?**" displayed above the **XIC's** icon indicates a parameter, in this case the instruction's **Tag**, that the programmer must define.

*** - Although the only parameter required for a Boolean instruction is a Tag, more complex instructions may require additional parameters, such as the Preset and Accum values for an On-Delay Timer.**

2

Tags

Tags contain information that identifies data stored in memory, allowing that data to be linked to the operation of one (or more) specific instruction(s).

Thus, given the following **XIC**:

the "**A**" displayed above the **XIC** icon is the **NAME** of a **Tag** that identifies a specific bit in memory, the value of which determines the state of the **XIC**.

3

Tags

Tags also contain additional information that characterizes the data, such as the amount of memory required to store the data and whether any access restrictions have been placed on that data.

But before we discuss some of the more detailed information that is contained in a tag, let's first look at the two primary **TYPES** of tags that you will utilize or see in your ladder logic programs:

- **Base Tags**
- **Alias Tags**

4

Base Tags vs. Alias Tags

A **Base Tag** is a tag that, when initially created, results in the allocation of memory and provides a reference to the data stored at that memory location, thus allowing the data to be directly linked to the operation of an instruction.

An **Alias Tag** is a tag that is used to provide a **different name** (i.e. – an alias) to a previously defined Base Tag.

Note – Alias tags allow the assignment of an arbitrary name to a previously defined Base Tag. This can be especially useful when an existing Base Tag has a complex or potentially confusing name, such as those automatically created by the RSLogix software* when the type and configuration of the PLC is initially defined.

*** - We will discuss some of these automatically-created Base Tags in just a few minutes!**

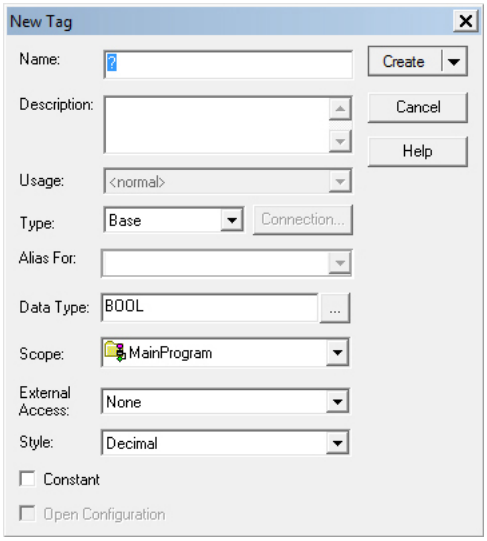
5

Creating a New Tag

When a **New Tag** is created, a variety of information must be provided, including:

- **NAME**
- **TYPE**
- **DATA TYPE**
- **SCOPE**
- **EXTERNAL ACCESS**
- **STYLE**

An optional "text" **DESCRIPTION** may also be provided if desired.

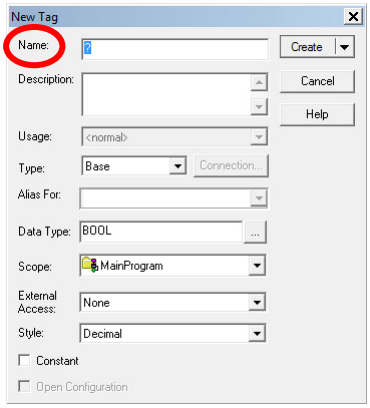


6

New Tag Information Fields

NAME: an alpha/numeric identifier (including underscores) that is assigned to the tag and thus to the data stored in memory.

Names must begin with an alphabetic character, cannot be greater than 40 characters in length, and are not case sensitive.



A strategically-chosen and representative tag NAMES can greatly increase the readability of a ladder logic program, especially for someone other than the original programmer.

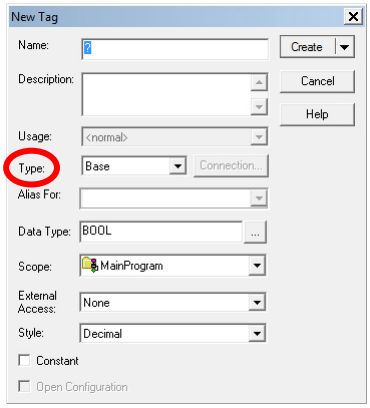
7

New Tag Information Fields

TYPE: Base – this results in an allocation of memory, the amount of which is determined by the DATA TYPE field.

Alias – the allows for the creation of an alias (nickname) for a previously defined tag.

When Alias is selected, the **ALIAS FOR** field becomes active in order to select the existing tag for which the alias applies.

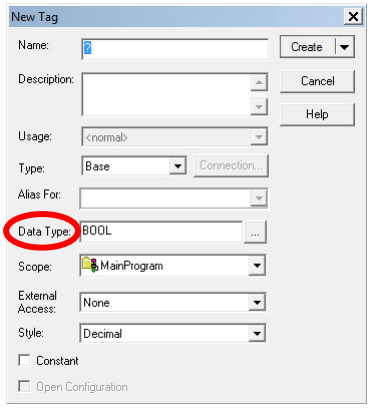


8

New Tag Information Fields

Although the remaining fields must be defined when creating a New Tag, if the tag was created by right-clicking a "**?**" in the tag location of a newly-placed instruction, the RSLogix software will place default values within the remaining fields that will typically suffice for most applications of that instruction.

Be careful not to change the default values that the software provides unless you fully understand the implications of those changes because that can cause a critical error resulting in a non-functioning program.



9

New Tag Information Fields

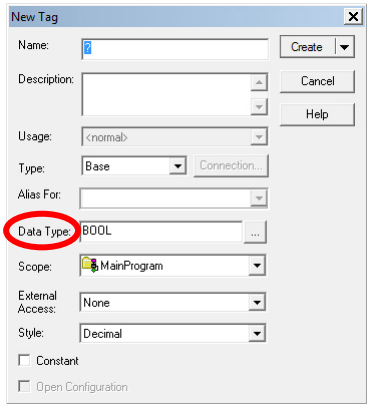
DATA TYPE: two of the data types are:

BOOL (Boolean) – Boolean tags are associated with **one bit** of memory, the value of which can be either:

0 or **1**

INT (Integer) – Integer tags require **sixteen bits** of memory, in which integer values (defined as binary #s) may be stored, ranging from:

–32768 to **+32767**



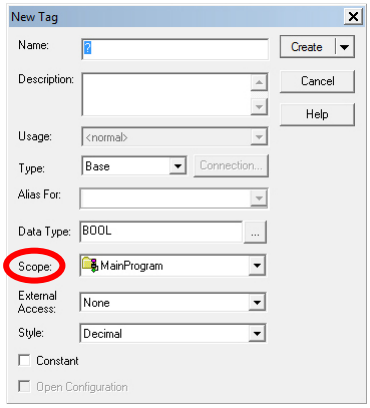
10

New Tag Information Fields

SCOPE: this field defines whether a tag is **global** or **local**.

The RSLogix 5000 software allows a program to be broken down into multiple smaller programs or subroutines.

A **global** tag is available across all of the subroutines, while a **local** tag is associated with a specific subroutine, such as Main Program.



11

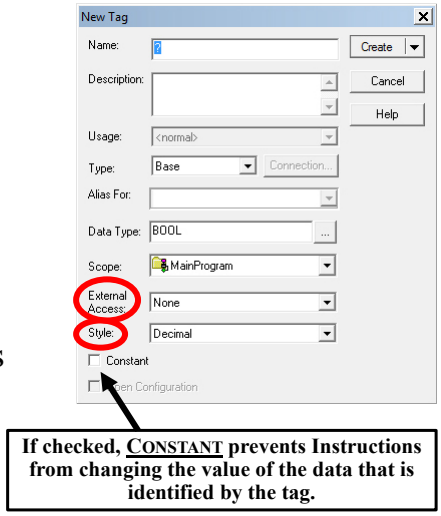
New Tag Information Fields

EXTERNAL ACCESS:

This field defines whether a tag is allowed **Read/Write**, **Read Only**, or no access (**None**) from programs running on external devices such as an HMI (Human Machine Interface) panel.

STYLE: this field only defines the format is which data is displayed (it does not affect the DATA TYPE).

Binary – base 2 or **Decimal – base 10**



If checked, CONSTANT prevents Instructions from changing the value of the data that is identified by the tag.

12

Initial Software Configuration

When initially creating a "**New Project**" within the RSLogix programming environment, the type and configuration of the system devices (PLC, etc.) must first be specified.

For Example: The **PLCs** contained in the Q-215 Machines Lab are:
- Allen-Bradley, **1769-L32E, Compact Logix 5332E PLCs**

Located in Bus Position #1 (**Local 1**) of each PLC is a:
- **1769-IQ16/A, 16-Port, 24V_{DC}, Input Module**

Located in Bus Position #2 (**Local 2**) of each PLC is a:
- **1769-OW16/A, 16-Port, Relay-type, Output Module**

Connected to the PLC via its **Ethernet (Comm) port** is a:
- **PowerFlex 40-E Variable Frequency Drive (VFD)**

(The method for entering the above information is not covered within this presentation)

13

Initial Software Configuration

The RSLogix software then uses this system information to pre-configure any of the device-specific features of the various components, making those features readily accessible to the user (programmer) without requiring the user to configure those components manually.

For Example: The **PowerFlex 40-E VFD** utilizes an integer value that is stored within its memory to determine the **frequency** of the output voltages that it is supplying to its motor-load.

When the VFD is entered into the system configuration, the RSLogix software automatically creates a **Base Tag** that identifies the specific memory location within the VFD, thus allowing the user to manipulate the value, in-turn directly controlling the **output frequency** of the drive.

14

Control of the System Devices

A solid understanding of operation of each of the system devices and the manner in which the RSLogix software provides for control (or monitoring) of those devices is required in order to be able to integrate all of those devices into a functioning system, the overall operation of which will be dictated by a user-created program that will be downloaded into the PLC.

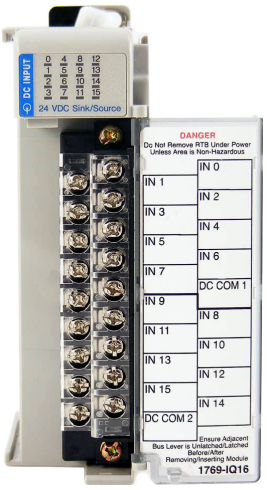
Thus, let us begin by discussing the operation of the PLC's input module and how the RSLogix software provides the means for linking the state of a logic instruction (within a ladder-logic program) to the status of an input port, in-turn allowing the PLC to react to changes in the state (open or closed) of a device (pushbutton, etc.) that is wired to one of those ports.

15

1769-IQ16 Input Module

The **1769-IQ16/A** is a **16-port**, **24V$_{DC}$**, **discrete input module** that is configured to detect either the presence or absence of a **24V$_{DC}$** signal at each of its input ports (terminals).

To facilitate this task, the input module contains **16 bits** of memory that are reserved to denote the status of the module's 16 input ports.



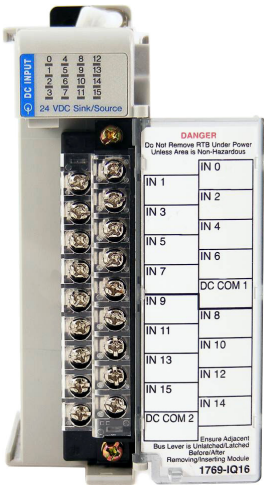
16

1769-IQ16 Input Module Operation

When the 1769-IQ16 **detects** that **$24V_{DC}$** is **present** at a specific input port, the module **sets the bit** associated with that port to a **1**.

But when the 1769-IQ16 **detects** that **$24V_{DC}$** is **not present** a specific input port, the module **resets the bit** associated with that port to a **0**.

It is the values of these bits that can be used within a ladder-logic program to link the status of the input ports to the operational logic of the program.



17

1769-IQ16 Input Module Base Tags

When the 1769-IQ16 module is added into a new project, the RSLogix software creates **16** Boolean-type **Base Tags**, one for each bit that is associated with the status of an Input Port.

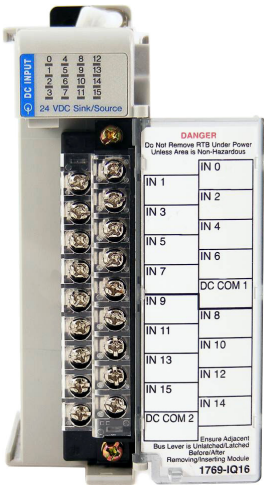
The format of the Base Tag names is:

Local:1:I.Data.X

where **X** ranges from **0 → 15** for each of the module's 16 Input Ports.

Note that the actual base tag name is "I.Data.X" where X ranges from 0 → 15.

The "Local:1" in front of the tag name instructs the PLC that the tag refers to a memory location that resides within the module that is (directly) connected "locally" to the PLC in the 1st slot position.



18

1769-IQ16 Input Module Operation

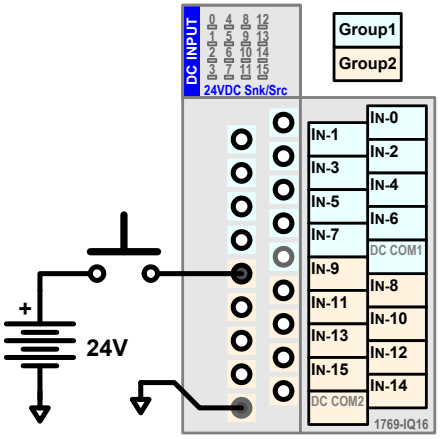
Thus, if a **NO pushbutton** is connected between a **24V$_{DC}$** source and terminal **IN-9**:

When the **button is pressed**,

- **24V$_{DC}$** is **present** at **IN-9**
- The module **sets the bit** **I.Data.9 ➡ 1**

When the **button is released**,

- **24V$_{DC}$** is **removed** from **IN-9**
- The module **resets the bit** **I.Data.9 ➡ 0**

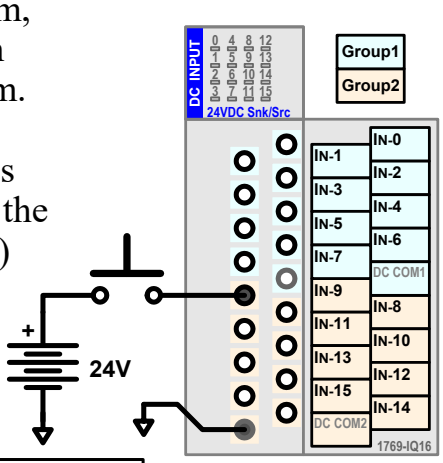


19

Linking the IQ16 to a Logic Instruction

By assigning one of these base tags to a logic instruction within a ladder-logic program, the state of the associated input port can affect the overall operation of the system.

For example, if the tag **Local:1:I.Data.9** is assigned to the **XIC** (shown below), then the **state** of the **XIC** (and the **Rung Condition**) will be determined by the value of bit **I.Data.9**:



Note that Base Tags created by the RS Logix software for the Input Module are displayed between angle-brackets. <Base Tag>

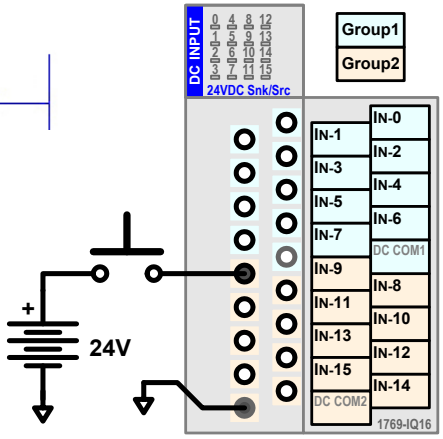
20

Linking the IQ16 to a Logic Instruction

Given the following ladder rung in which the **XIC** is assigned the tag **Local:1:I.Data.9**:

While the **button** is **NOT pressed**,

- **24V$_{DC}$** is **not present** at **IN-9**
- The module **resets the bit** **I.Data.9 ➡ 0**
- **XIC-Local:1:I.Data.9 ➡ FALSE** (making the **Rung Condition FALSE**)
- **OTE-Y resets bit Y ➡ 0**



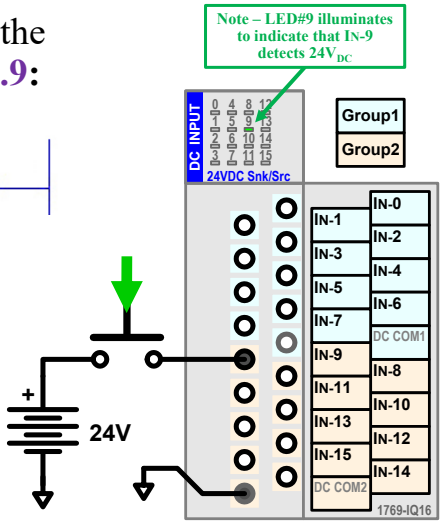
21

Linking the IQ16 to a Logic Instruction

Given the following ladder rung in which the **XIC** is assigned the tag **Local:1:I.Data.9**:

When the **button** is **pressed**,

- **24V$_{DC}$** is **present** at **IN-9**
- The module **sets the bit** **I.Data.9 ➡ 1**
- **XIC-Local:1:I.Data.9 ➡ TRUE** (making the **Rung Condition TRUE**)
- **OTE-Y sets bit Y ➡ 1**

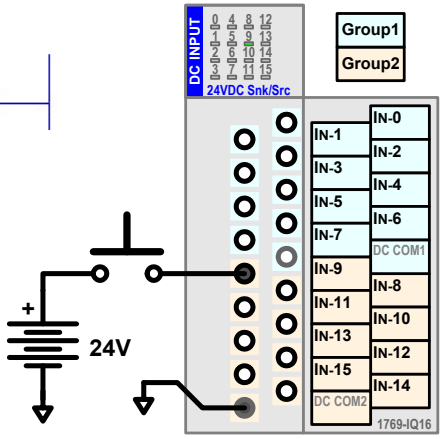


22

Linking the IQ16 to a Logic Instruction

Although the **state** of **Input-9** is **linked** to the **state** of the following **XIC**:

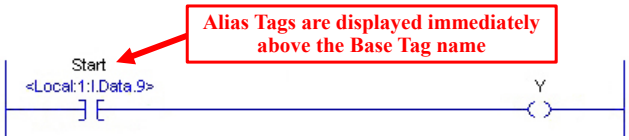
the **function** of the **pushbutton** wired to Input Port **9** is not clearly indicated, which may be confusing when trying to create, modify, or debug the ladder diagram, especially to anyone other than the original programmer.



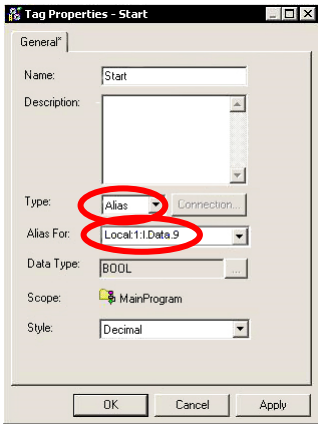
23

Assigning as Alias to an IQ16 Base Tag

Alias Tags may be created for the 1769-IQ16's Base Tags to help clarify the ladder diagram:



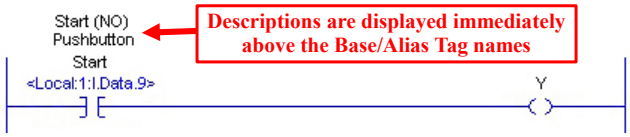
For example, if the intended function of the button wired to Input Port **9** is to "start a process", then an **Alias Tag** named **Start** may be created for the base tag **Local:1:I.Data.9**.



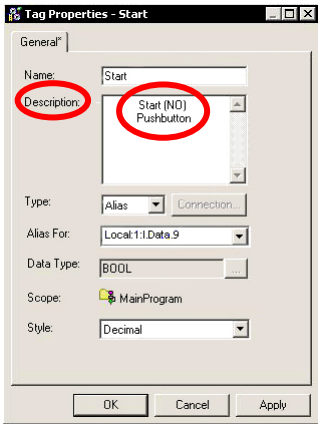
24

Assigning as Alias to an IQ16 Base Tag

Alias Tags may be created for the 1769-IQ16's Base Tags to help clarify the ladder diagram:



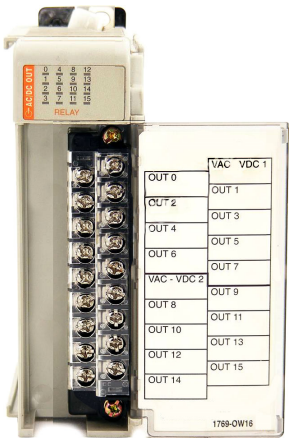
Additionally, if a **description** is added for the Alias Tag named **Start**, then the description will also appear in the ladder diagram immediately above the Alias Tag name.



25

1769-OW16 Output Module

The **1769-OW16/A**, is a **16-port**, **relay-type**, **discrete output module** that has a NO contact wired between each of its output ports (terminals) and a common terminal.



By closing or opening these contacts, the OW16 can either make or break conductive paths between the common terminal and the various output ports.

If a voltage source is connected to the common terminal and loads are connected to the output ports, the OW16 can then be used to energize or de-energize those loads.

26

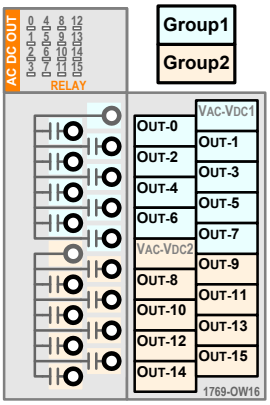
1769-OW16 Output Module

Note that the 1769-IQ16's **output ports** are separated into two groups:

- **Group 1: OUT-0 → OUT-7**
- **Group 2: OUT-8 → OUT-15**

The NO contacts for **Group 1** are connected between its output ports and the common terminal **VAC-VDC1**.

The NO contacts for **Group 2** are connected between its output ports and the common terminal **VAC-VDC2.**



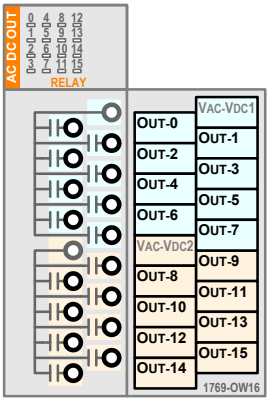
27

1769-OW16 Output Module Operation

The 1769-OW16 output module also contains **16 bits** of memory, the values stored in which used to set the status of each of the module's 16 output ports.

When the module detects that a specific bit is **0**, the module **opens** the NO contact between the port associated with that bit and the port's common terminal.

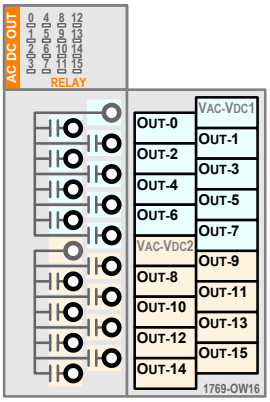
But when the module detects that a specific bit is **1**, the module **closes** the NO contact between the port associated with that bit and the port's common terminal.



28

1769-OW16 Output Module Base Tags

When the 1769-OW16 module is added into an RSLogix project, the software creates **16** Boolean-type **Base Tags**, one for each bit that is used to determine the status of an Output Port.



The format of the Base Tag names is:

Local:2:O.Data.X

where **X** ranges from **0 → 15** for each of the 16 Output Ports.

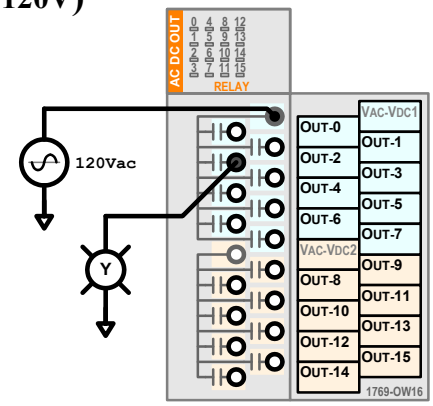
Note that the actual base tag name is "O.Data.X" where X ranges from 0 → 15.

The "Local:2" in front of the tag name instructs the PLC that the tag refers to a memory location that resides within the module that is (directly) connected "locally" to the PLC in the 2nd slot position.

29

1769-OW16 Output Module Operation

Thus, if a **120V$_{ac}$** source is connected to terminal **VAC-VDC1** and a **yellow lamp (rated at 120V)** is connected to port **OUTPUT-2**:



When **bit O.Data.2 = 0**,

- The module will **OPEN** the the NO **contact** for **OUT-2**
- The **Yellow Lamp** will turn **OFF**

When **bit O.Data.2 = 1**,

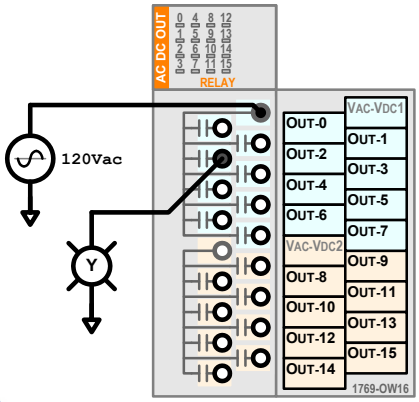
- The module will **CLOSE** the the NO **contact** for **OUT-2**
- The **Yellow Lamp** will turn **ON**

30

Linking the OW16 to an Output Instruction

By assigning one of these base tags to an output instruction within a ladder-logic program, the state of an output port can be affected by that output instruction.

For example, if the tag **Local:2:O.Data.2** is assigned to the **OTE** shown below, the output instruction can **set** or **reset** the **bit O.Data.2**, causing **output 2** to turn **ON** or **OFF**.

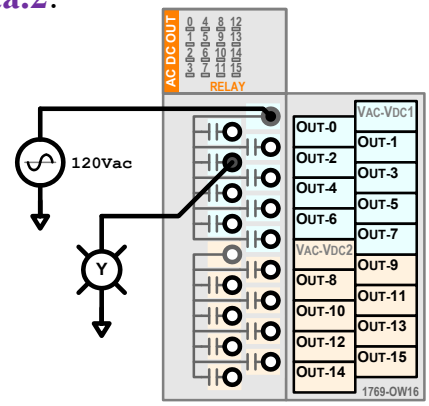


Note that, as shown, an Alias Tag called "Yellow_Light" also been created for the Base Tag <Local:2:O.Data.2>

31

Linking the OW16 to an Output Instruction

Given the following ladder rung in which the **OTE** is assigned the tag **Local:2:O.Data.2**:



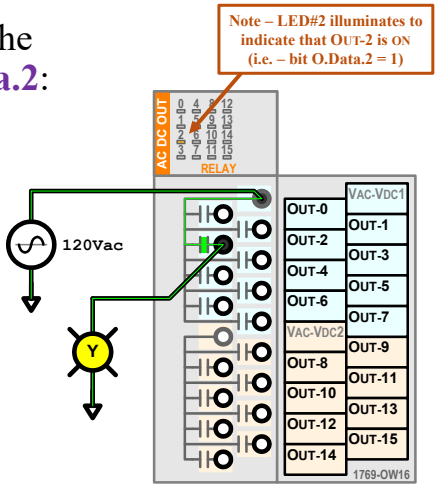
When the **Rung Condition** is **FALSE**,

- The **OTE resets** the **bit** **O.Data.2 ➡ 0**
- The module **OPENS** the NO **contact** for **OUT-2**
- The **Yellow light** turns **OFF**

32

Linking the OW16 to an Output Instruction

Given the following ladder rung in which the **OTE** is assigned the tag **Local:2:O.Data.2**:



When the **Rung Condition** is **TRUE,**

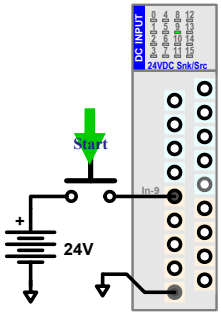
- The **OTE sets** the **bit** **O.Data.2 ➡ 1**
- The module **CLOSES** the NO **contact** for **OUT-2**
- The **Yellow light** turns **ON**

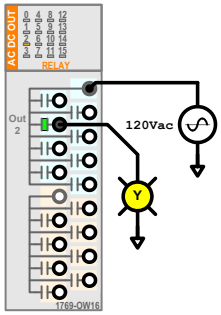
33

From Input to Output

Given the system shown below, when the **button** is **pressed:**

- **24V$_{DC}$** is **applied** to terminal **IN-9**
- The IQ-16 **sets bit I.Data.9 ➡ 1**
- **XIC-Start ➡ TRUE**
- **Rung Condition ➡ TRUE**



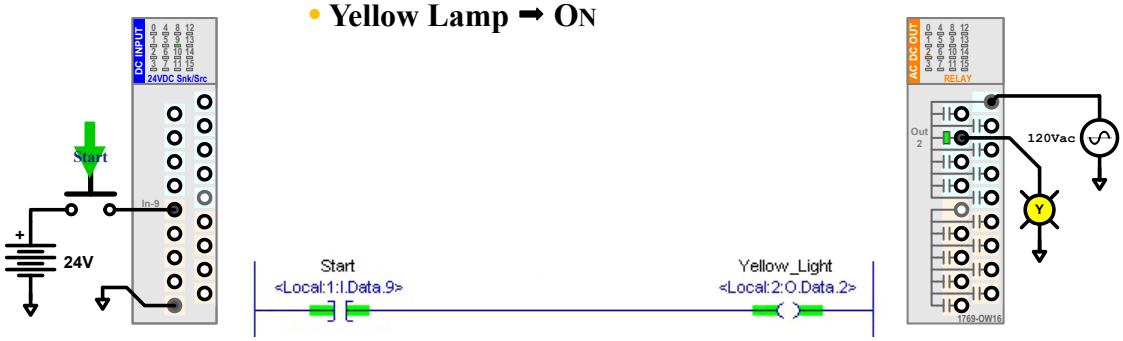


34

From Input to Output

Given the system shown below, when the **button** is **pressed:**

- While the **Rung Condition ➡ TRUE**
- **OTE-Yellow_Light sets bit O.Data.2 ➡ 1**
- The **OW-16 CLOSES contact** for **OUT-2**
- **Yellow Lamp ➡ ON**

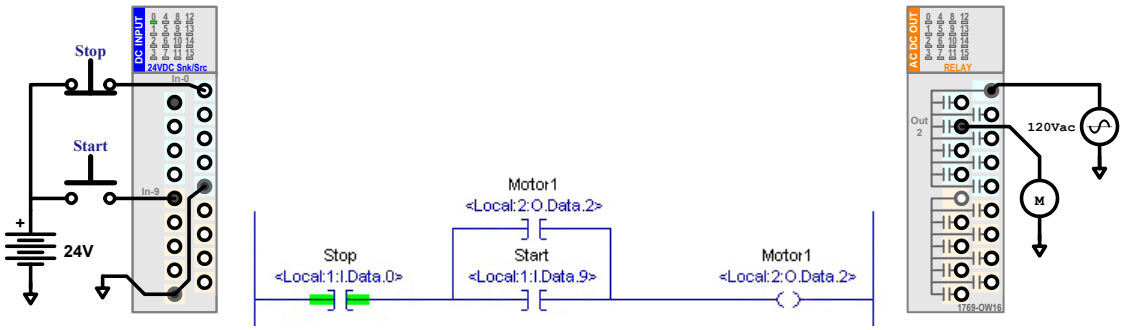


35

Stop/Start Controller using an OTE

Let's look at the following control system that will be utilized as part of a **PLC-based**, **Stop/Start motor controller**.

Note that the figure does not depict the power portion of the system that contains the controlled motor and the contactor's main contacts.

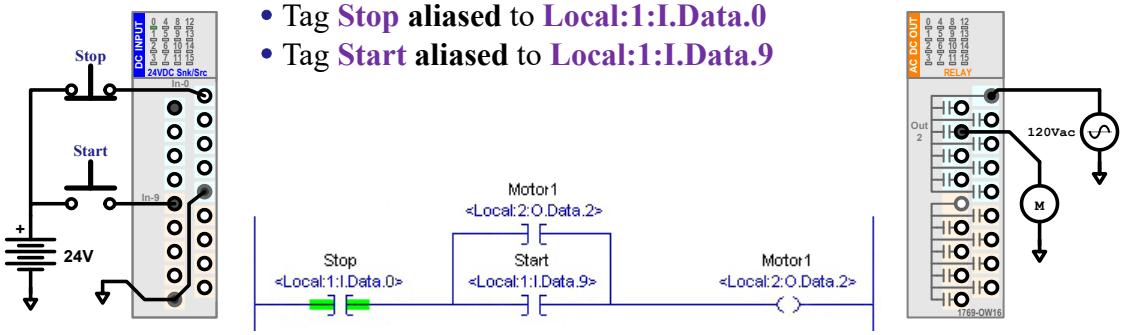


36

Stop/Start Controller using an OTE

Given a system with the following **input configuration**:

- **24V$_{DC}$ is connected** to the **NC** and **NO pushbuttons**
- **NC** button (**STOP**) **connected** to **IN-0**
- **NO** button (**START**) **connected** to **IN-9**
- **DC-COM1** and **DC-COM2 connected** to **DC ground**
- Tag **Stop aliased** to **Local:1:I.Data.0**
- Tag **Start aliased** to **Local:1:I.Data.9**

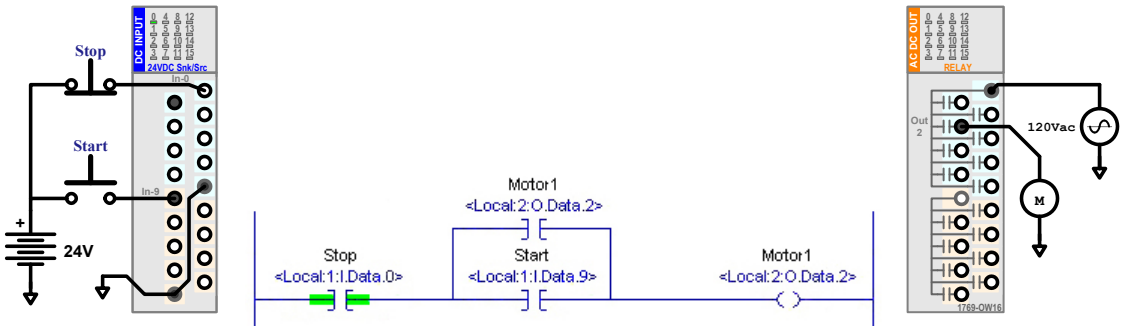


37

Stop/Start Controller using an OTE

Given a system with the following **output configuration**:

- Tag **Motor1 aliased** to **Local:2:O.Data.2**
- **120V$_{ac}$ connected** to **VAC-VDC1**
- **Field coil** of contactor **M connected** to **OUT-2**

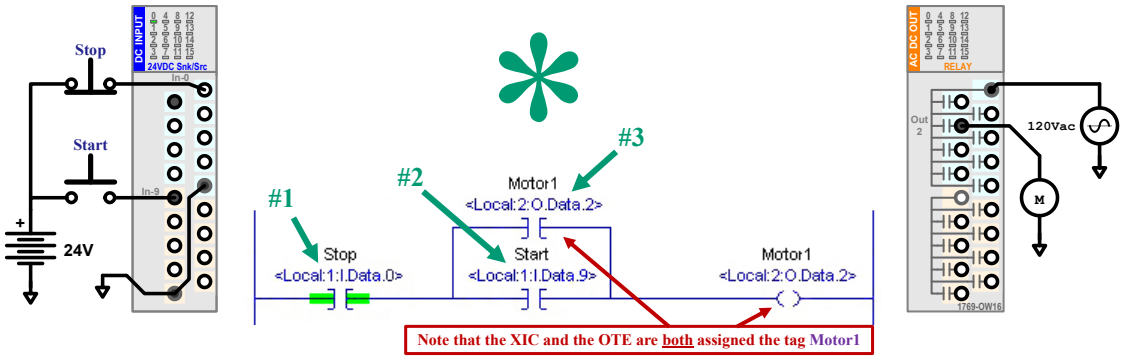


38

Stop/Start Controller using an OTE

Given a system with the following **ladder-logic rung**:

- **Stop** alias tag **assigned** to **XIC #1**
- **Start** alias tag **assigned** to **XIC #2**
- **Motor1** alias tag **assigned** to **OTE** and **XIC #3**



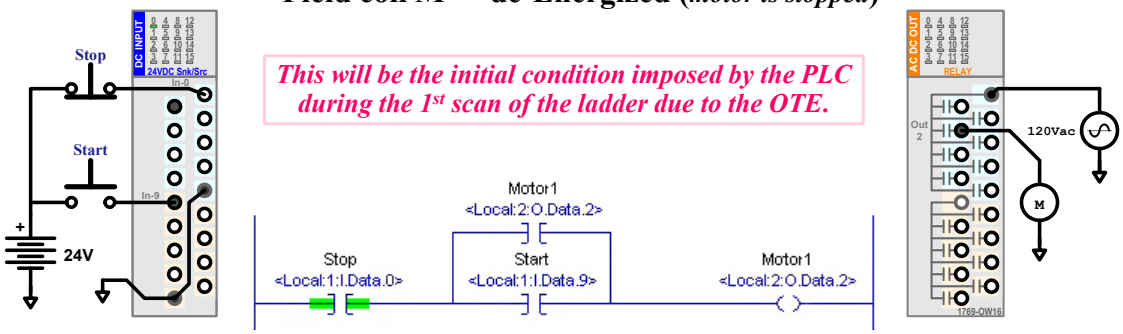
39

Stop/Start Controller using an OTE

Pre-Scan of the ladder when the PLC is initially set to **RUN** mode:

- During the **Pre-Scan**, all **Rung Conditions** ➡ **FALSE**
- **OTE-Motor1 resets bit O.Data.2 ➡ 0**
- **Contact** for **OUT-2 ➡ OPEN**
- **Field coil M ➡ de-Energized** (***motor is stopped***)

This will be the initial condition imposed by the PLC during the 1st scan of the ladder due to the OTE.

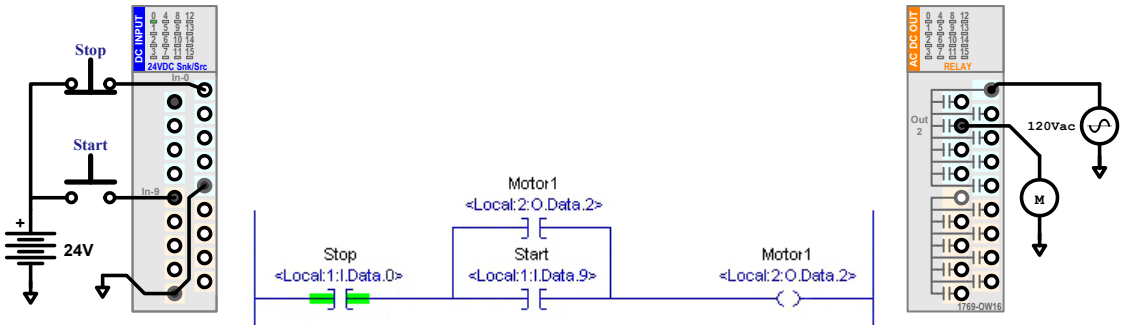


40

Stop/Start Controller using an OTE

While **no buttons** are **pressed** (assuming bit **O.Data.2** is initially **0**):

- **24V$_{DC}$ is applied** to **IN-0** **24V$_{DC}$ not applied** to **IN-9**
- **Bit I.Data.0 ➡ 1** **Bit I.Data.9 ➡ 0**
- **XIC-Stop ➡ TRUE** **XIC-Start ➡ FALSE**
- **Rung Condition ➡ FALSE**

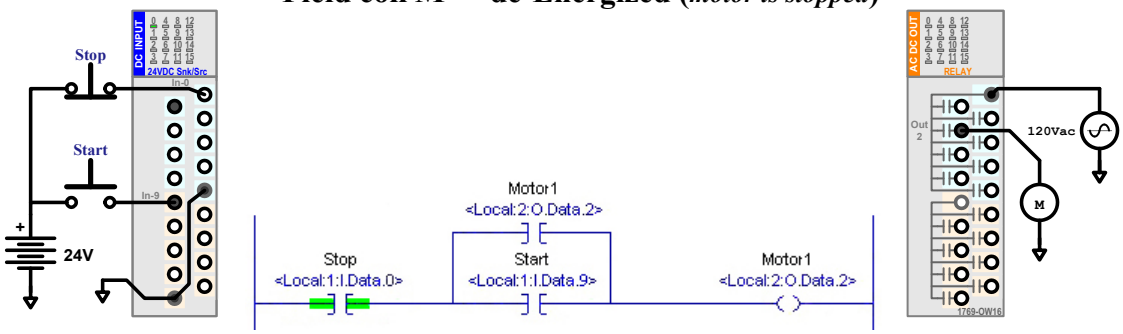


41

Stop/Start Controller using an OTE

While **no buttons** are **pressed** (assuming bit **O.Data.2** is initially **0**):

- While the **Rung Condition ➡ FALSE**
- **OTE-Motor1 resets bit O.Data.2 ➡ 0**
- **Contact for OUT-2 ➡ OPEN**
- **Field coil M ➡ de-Energized (*motor is stopped*)**



42

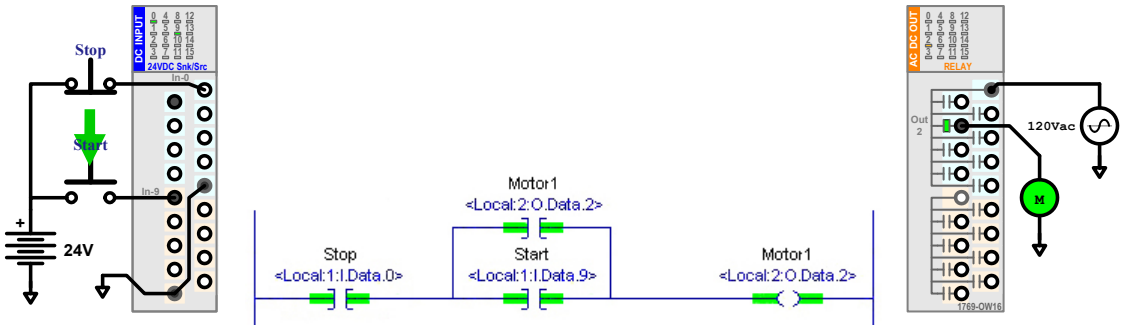
Stop/Start Controller using an OTE

When the "**Start**" button is **pressed**:

- **24V$_{DC}$ is applied to IN-0**
- **Bit I.Data.0 ➡ 1**
- **XIC-Stop ➡ TRUE**

24V$_{DC}$ is applied to IN-9
Bit I.Data.9 ➡ 1
XIC-Start ➡ TRUE

- **Rung Condition ➡ TRUE**



43

Stop/Start Controller using an OTE

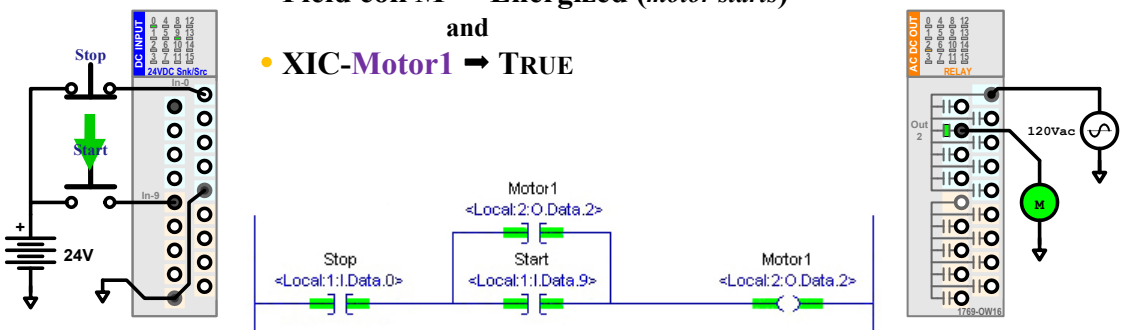
When the "**Start**" button is **pressed**:

- When the **Rung Condition ➡ TRUE**

- **OTE-Motor1 sets bit O.Data.2 ➡ 1**
- **Contact OUT-2 ➡ CLOSED**
- **Field coil M ➡ Energized (*motor starts*)**

and

- **XIC-Motor1 ➡ TRUE**



44

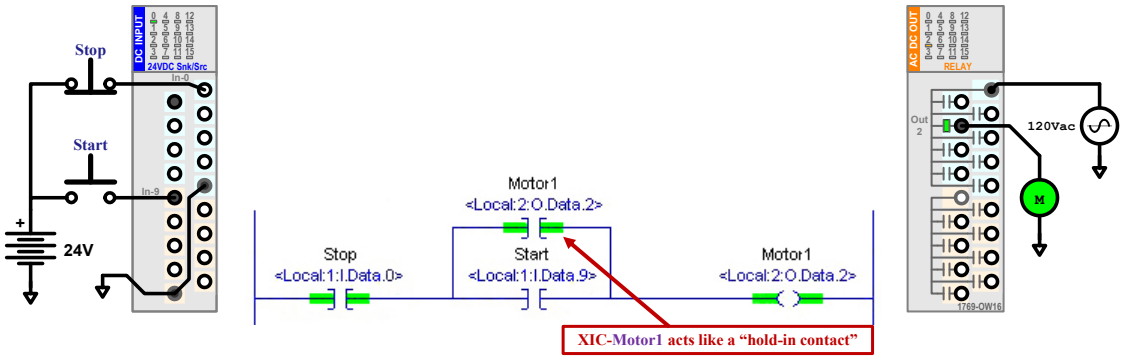
Stop/Start Controller using an OTE

When the "**Start**" button is **released**:

- **24V$_{DC}$ is applied** to **IN-0**
- **Bit I.Data.0 ➡ 1**
- **XIC-Stop ➡ TRUE**

24V$_{DC}$ is removed from **IN-9**
Bit I.Data.9 ➡ 0
XIC-Start ➡ FALSE

- **Rung Condition** remains **TRUE** (due to **XIC-Motor1**)

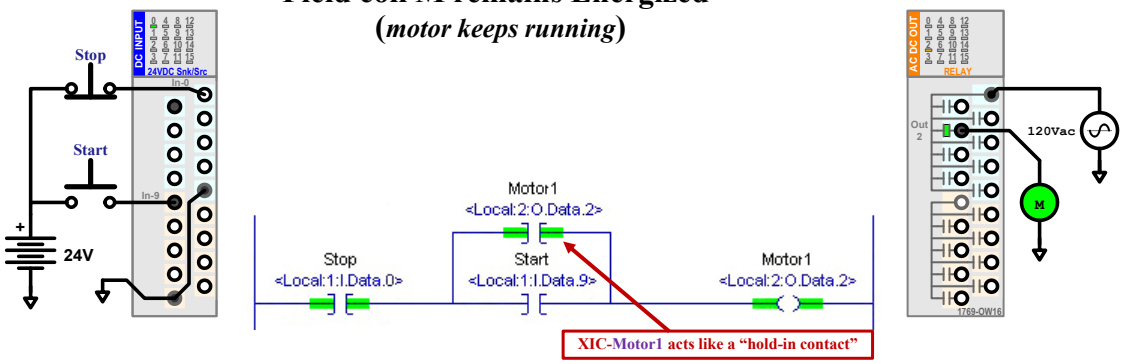


45

Stop/Start Controller using an OTE

When the "**Start**" button is **released**:

- While the **Rung Condition** remains **TRUE** (due to **XIC-Motor1**)
- **OTE-Motor1 holds bit O.Data.2 ➡ 1**
- **Contact OUT-2 remains CLOSED**
- **Field coil M remains Energized**
  (***motor keeps running***)

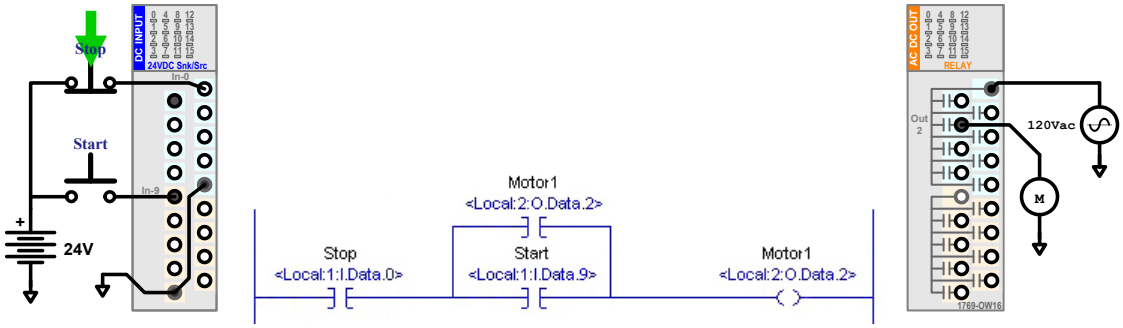


46

Stop/Start Controller using an OTE

When the "**Stop**" button is **pressed**:

- **24V$_{DC}$ is removed** from **IN-0**
- **Bit I.Data.0 ➡ 0**
- **XIC-Stop ➡ FALSE**
- **Rung Condition ➡ FALSE**



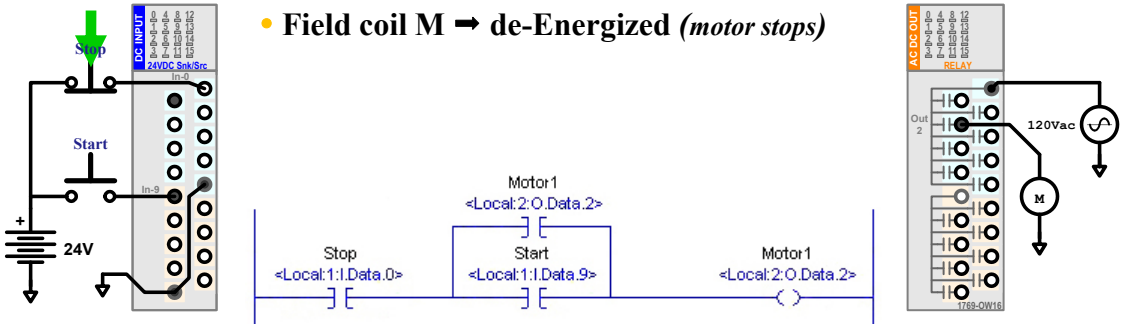
47

Stop/Start Controller using an OTE

When the "**Stop**" button is **pressed**:

- When the **Rung Condition ➡ FALSE**

- **OTE-Motor1 resets bit O.Data.2 ➡ 0**
- **XIC-Motor1 ➡ FALSE**
- **Contact OUT-2 ➡ OPENS**
- **Field coil M ➡ de-Energized** ***(motor stops)***

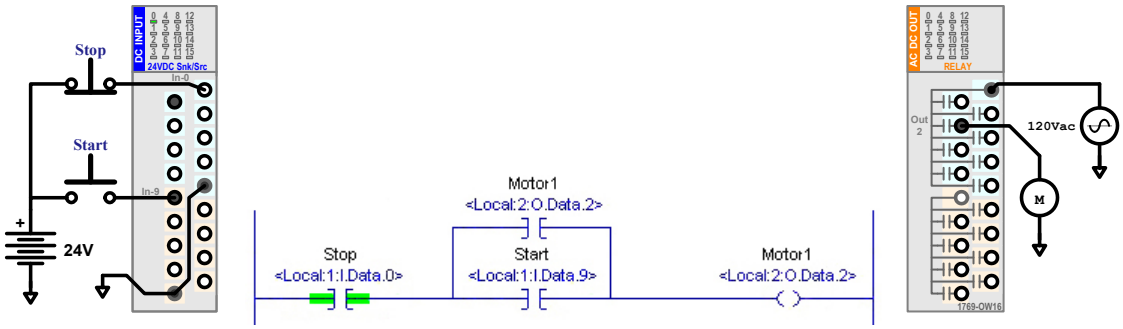


48

Stop/Start Controller using an OTE

When the "**Stop**" button is **released**:

- **24V$_{DC}$ is re-applied** to **IN-0**
- **Bit I.Data.0 ➡ 1**
- **XIC-Stop ➡ TRUE**
- **Rung Condition remains FALSE**



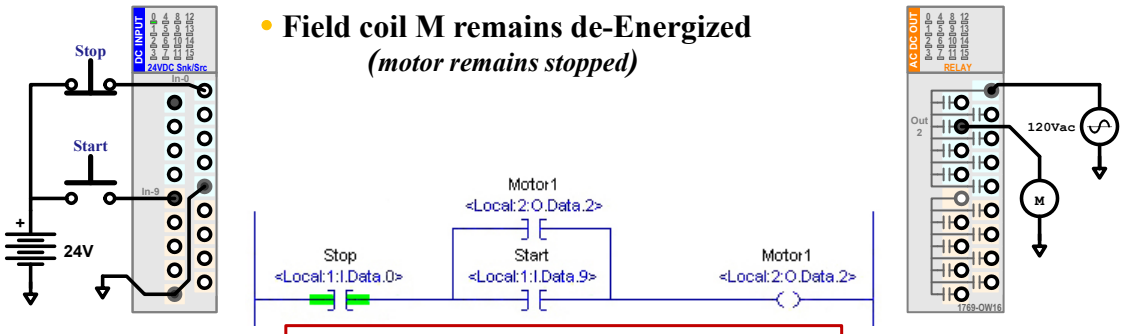
49

Stop/Start Controller using an OTE

When the "**Stop**" button is **released**:

- While the **Rung Condition remains FALSE**

- **OTE-Motor1 holds bit O.Data.2 ➡ 0**
- **XIC-Motor1 remains FALSE**
- **Contact OUT-2 remains OPENED**
- **Field coil M remains de-Energized**
  (motor remains stopped)



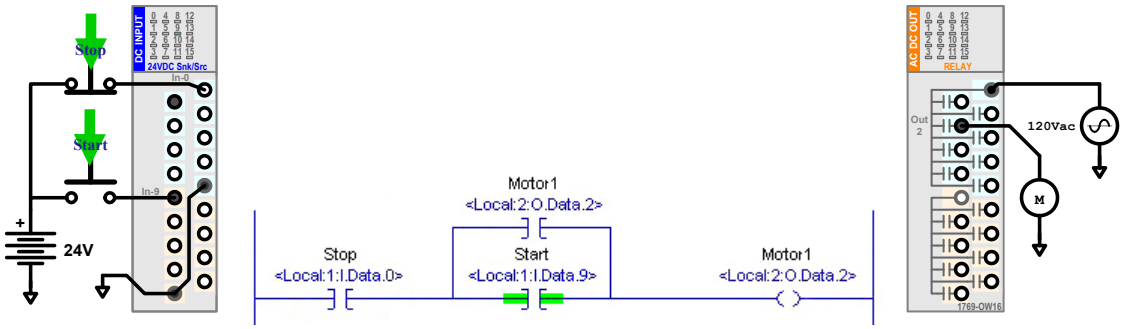
The system has returned to the initial "stopped" condition.

50

Stop/Start Controller using an OTE

What if "**Stop**" and "**Start**" are **pressed** at the **same time?**

- **24V$_{DC}$ removed** from **IN-0** **24V$_{DC}$ is applied** to **IN-9**
- **Bit I.Data.0 ➡ 0** **Bit I.Data.9 ➡ 1**
- **XIC-Stop ➡ FALSE** **XIC-Start ➡ TRUE**
- **Rung Condition ➡ FALSE**



51

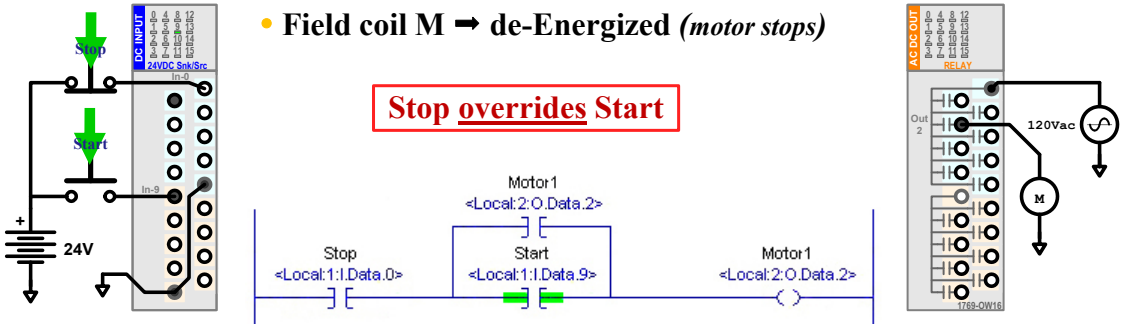
Stop/Start Controller using an OTE

What if "**Stop**" and "**Start**" are **pressed** at the **same time?**

- When the **Rung Condition ➡ FALSE**

- **OTE-Motor1 resets bit O.Data.2 ➡ 0**
- **XIC-Motor1 ➡ FALSE**
- **Contact OUT-2 ➡ OPENS**
- **Field coil M ➡ de-Energized** ***(motor stops)***

Stop overrides Start

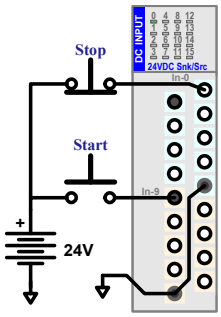


52

Stop/Start Controller using an OTL & OTU

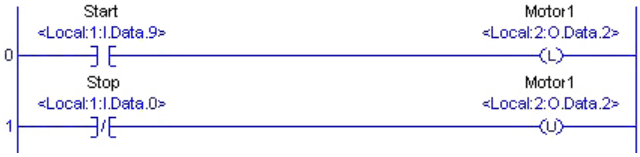
Under "normal" conditions when no buttons are pressed:

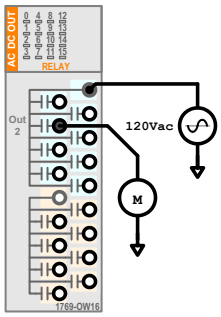
- $24V_{DC}$ is applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE — Input Port 9 ➡ FALSE
- Bit I.Data.0 ➡ 1 — Bit I.Data.9 ➡ 0
- XIC-Stop ➡ FALSE — XIC-Start ➡ FALSE
- Rung Condition 1 ➡ FALSE — Rung Condition 0 ➡ FALSE



Assuming Bit O.Data.2 = 0 ⇦ *(Initial Condition)*

- OTL-Motor1 and OTU-Motor1 do nothing
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor is stopped)*





53

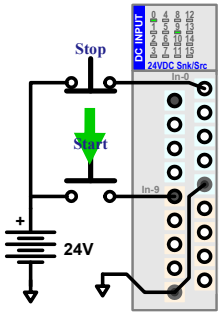
Stop/Start Controller using an OTL & OTU

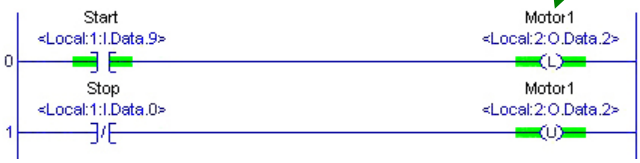
When the "Start" button is pressed:

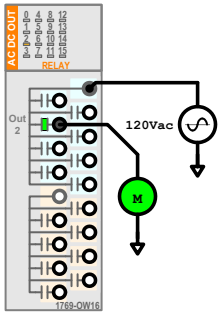
- $24V_{DC}$ is applied to IN-9 (by the NO pushbutton)
- Input Port 9 ➡ TRUE
- Bit I.Data.9 ➡ 1
- XIC-Start ➡ TRUE
- Rung Condition 0 ➡ TRUE

Green bar behind OTL/OTL denotes that the bit referenced by their tag is a 1
Local:2:O.Data.2 = 1

- OTL-Motor1 Sets Bit O.Data.2 ➡ 1
- Output Port 2 ➡ TRUE
- Contact OUT-2 ➡ CLOSED
- Field coil M ➡ Energized *(motor starts)*







54

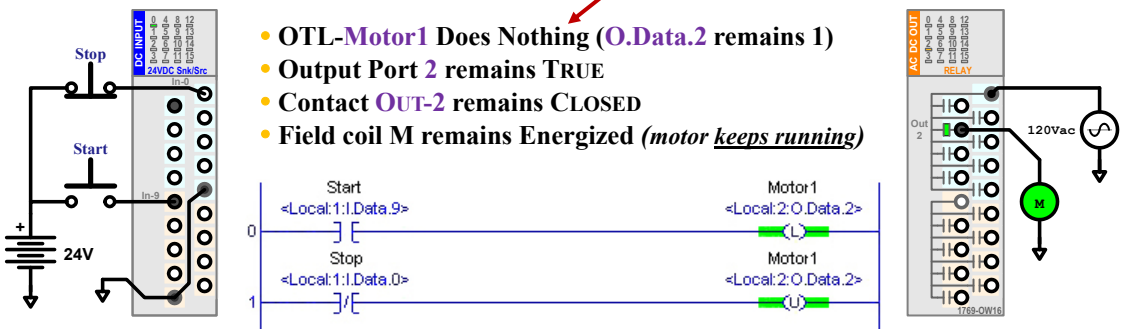
Stop/Start Controller using an OTL & OTU

When the "Start" button is released:

- 24V_{DC} is removed from IN-9 (by the NO pushbutton)
- Input Port 9 ➡ FALSE
- Bit I.Data.9 ➡ 0
- XIC-Start ➡ FALSE
- Rung Condition 0 ➡ FALSE

OTL-Motor1 cannot reset the bit O.Data.2, therefore when OTL-Motor1 ➡ FALSE, the bit O.Data.2 remains 1

- OTL-Motor1 Does Nothing (O.Data.2 remains 1)
- Output Port 2 remains TRUE
- Contact OUT-2 remains CLOSED
- Field coil M remains Energized *(motor keeps running)*



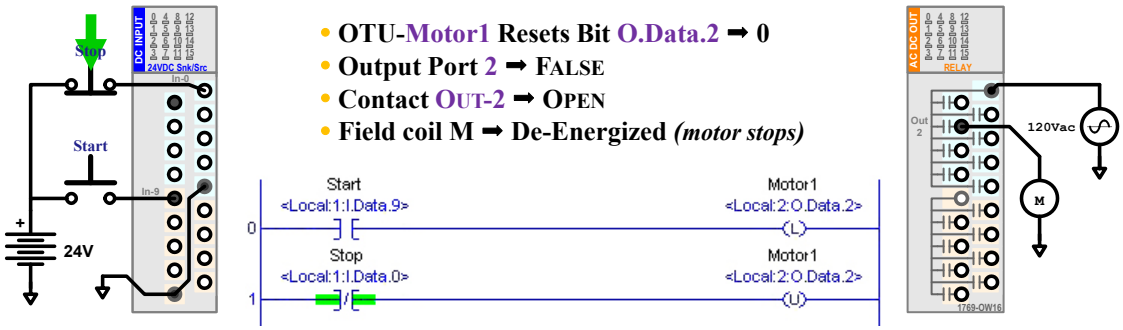
55

Stop/Start Controller using an OTL & OTU

When the "Stop" button is pressed:

- 24V_{DC} is removed from IN-0 (by the NC pushbutton)
- Input Port 0 ➡ FALSE
- Bit I.Data.0 ➡ 0
- XIO-Stop ➡ TRUE
- Rung Condition 1 ➡ TRUE

- OTU-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor stops)*



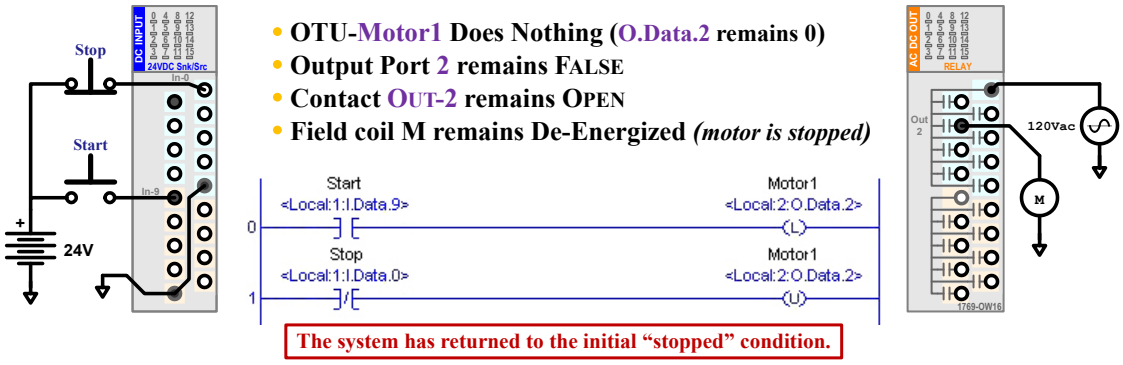
56

Stop/Start Controller using an OTL & OTU

When the "Stop" button is released:

- $24V_{DC}$ is re-applied to IN-0 (by the NC pushbutton)
- Input Port 0 ➡ TRUE
- Bit I.Data.0 ➡ 1
- XIO-Stop ➡ FALSE
- Rung Condition 1 ➡ FALSE

- OTU-Motor1 Does Nothing (O.Data.2 remains 0)
- Output Port 2 remains FALSE
- Contact OUT-2 remains OPEN
- Field coil M remains De-Energized *(motor is stopped)*



The system has returned to the initial "stopped" condition.

57

Stop/Start Controller using an OTL & OTU

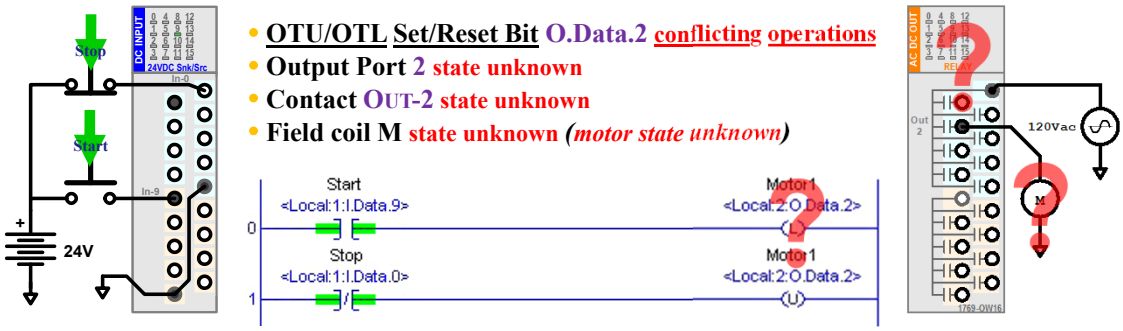
What if "Stop" and "Start" are both pressed at the same time:

- $24V_{DC}$ is removed from IN-0
- Input Port 0 ➡ FALSE
- Bit I.Data.0 ➡ 0
- XIO-Stop ➡ TRUE
- Rung Condition 1 ➡ TRUE

$24V_{DC}$ is applied to IN-9
Input Port 9 ➡ TRUE
Bit I.Data.9 ➡ 1
XIC-Start ➡ TRUE
Rung Condition 1 ➡ FALSE

Actual operation will depend on the placement of the OTL & OTU within the overall ladder diagram

- OTU/OTL Set/Reset Bit O.Data.2 conflicting operations
- Output Port 2 state unknown
- Contact OUT-2 state unknown
- Field coil M state unknown *(motor state unknown)*



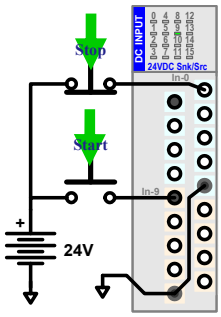
58

Stop/Start Controller using an OTL & OTU

What if "Stop" and "Start" are both pressed at the same time:

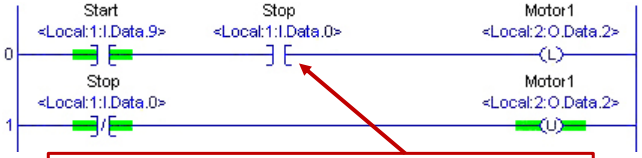
- $24V_{DC}$ is removed from IN-0
- Input Port 0 ➡ FALSE
- Bit I.Data.0 ➡ 0
- XIO-Stop ➡ TRUE
- Rung Condition 1 ➡ TRUE

$24V_{DC}$ is applied to IN-9
Input Port 9 ➡ TRUE
Bit I.Data.9 ➡ 1
XIC-Start ➡ TRUE XIC-Stop ➡ FALSE
Rung Condition 1 ➡ FALSE

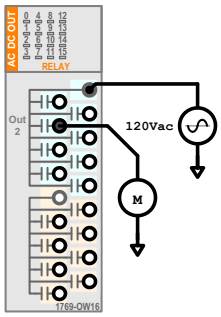


- OTU-Motor1 Resets Bit O.Data.2 ➡ 0
- Output Port 2 ➡ FALSE
- Contact OUT-2 ➡ OPEN
- Field coil M ➡ De-Energized *(motor stops)*

Stop overrides Start



XIC-Stop is FALSE while "Stop" is pressed, preventing OTL from latching bit



59

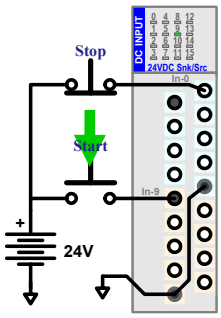
Destructive Bit Reference Error using OTEs

When the "Start" button is pressed:

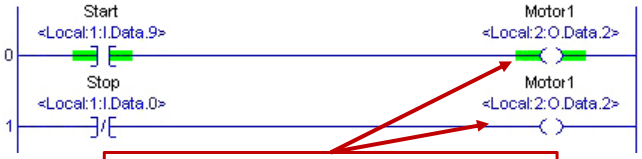
- $24V_{DC}$ was already at IN-0
- Input Port 0 ➡ TRUE
- Bit I.Data.0 ➡ 1
- XIO-Stop ➡ FALSE
- Rung Condition 1 ➡ FALSE

$24V_{DC}$ is applied to IN-9
Input Port 9 ➡ TRUE
Bit I.Data.9 ➡ 1
XIC-Start ➡ TRUE
Rung Condition 1 ➡ TRUE

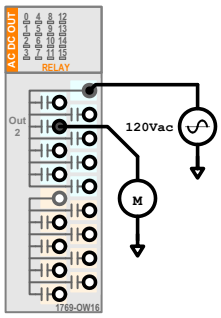
There should never be two or more OTEs with the same tag in a ladder diagram!!



- OTE/OTE Set/Reset O.Data.2 conflicting operations
- Output Port 2 state unknown
- Contact OUT-2 state unknown
- Field coil M state unknown *(motor state unknown)*



OTE-Motor1 on Rung 0 conflicts with OTE-Motor1 on Rung 1



60

PowerFlex 40 Variable Frequency Drive

When properly configured* and connected to the PLC via a communications network, the operation of a PowerFlex 40 Variable Frequency Drive (PF40 VFD) can also be controlled and monitored by the PLC.

This is accomplished by either manipulating or monitoring the values stored in specific locations within the PF40's memory.

*** - detailed information regarding the configuration of the PF40 VFD is not provided in this presentation.**

Refer to parts B and C of the CompactLogix-based Motor Control System tutorial lab exercise for wiring and configuration information of the PF40.

61

PowerFlex 40 – Tags

When a PF40 is initially added into an RSLogix project, a set of Tags are automatically created by the RS5000 software, similar to those created for the IQ16 and OW16 input and output modules:

Input Tags

VFD:I	AB:PowerFlex40_Drive_8Bytes:I:0
VFD:I.DriveStatus	INT
VFD:I.Ready	BOOL
VFD:I.Active	BOOL
VFD:I.CommandDir	BOOL
VFD:I.ActualDir	BOOL
VFD:I.Accelerating	BOOL
VFD:I.Decelerating	BOOL
VFD:I.Alarm	BOOL
VFD:I.Faulted	BOOL
VFD:I.AtReference	BOOL
VFD:I.CommFreqCnt	BOOL
VFD:I.CommLogicCnt	BOOL
VFD:I.ParmsLocked	BOOL
VFD:I.DigIn1Active	BOOL
VFD:I.DigIn2Active	BOOL
VFD:I.DigIn3Active	BOOL
VFD:I.DigIn4Active	BOOL
VFD:I.OutputFreq	INT

Output Tags

VFD:O	AB:PowerFlex40_Drive_4Bytes:O:1
VFD:O.LogicCommand	INT
VFD:O.Stop	BOOL
VFD:O.Start	BOOL
VFD:O.Jog	BOOL
VFD:O.ClearFaults	BOOL
VFD:O.Forward	BOOL
VFD:O.Reverse	BOOL
VFD:O.OptoOutput1	BOOL
VFD:O.OptoOutput2	BOOL
VFD:O.AccelRate1	BOOL
VFD:O.AccelRate2	BOOL
VFD:O.DecelRate1	BOOL
VFD:O.DecelRate2	BOOL
VFD:O.FreqSel01	BOOL
VFD:O.FreqSel02	BOOL
VFD:O.FreqSel03	BOOL
VFD:O.RelayOutput	BOOL
VFD:O.FreqCommand	INT

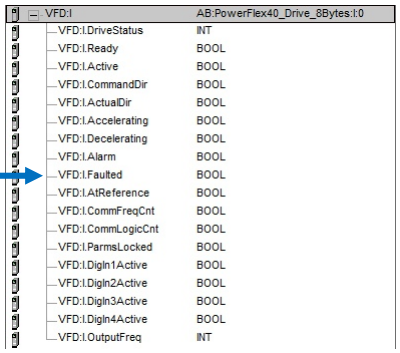
62

VFD – Input Tags

The <u>Input Tags</u> refer to memory locations within the PF40's memory that contain information relating to the VFD's operational state:

Input Tags

FOR EXAMPLE
When the VFD becomes faulted, the bit referred to by the tag:
I.Faulted
is set to a 1.
Thus, the value of this bit can be monitored by the PLC, and if a fault occurs, the PLC can be programmed to take the appropriate action.



Not that the "VFD" that appears in front of the tag names is the name assigned to the PF40 when added into the RSLogix project.

Note that the values stored in the Input Tag locations are set by the VFD based upon its operational state and **<u>cannot</u>** be changed by the PLC (user).

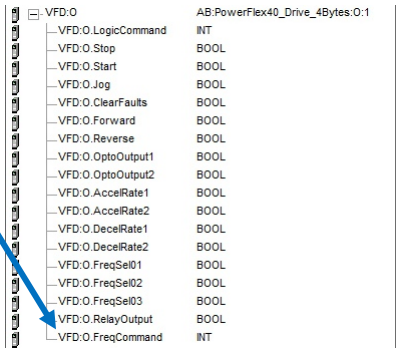
63

VFD – Output Tags

The <u>Output Tags</u> refer to memory locations within the PF40's memory that contain information that determines the operational state of the VFD:

Output Tags

FOR EXAMPLE
The integer value stored in the location defined by the tag:
O.FreqCommand
determines the output frequency of the drive.
Thus, a MOV command can be utilized to write a new value to this location in order to change the drive's output frequency.



Not that the "VFD" that appears in front of the tag names is the name assigned to the PF40 when added into the RSLogix project.

Note that the values stored in the Output Tag locations **<u>can</u>** be changed by the PLC (user) in order to change the operational state of the drive.

64

Starting & Stopping the VFD

Two tags define whether the PF40 is enabled or disabled:

O.Start

O.Stop

The O.Start and O.Stop bits should always contain opposing values.

To <u>enable</u> (start) the drive, the O.START bit should be set ➡ 1 and the O.STOP bit should be reset ➡ 0.

To <u>disable</u> (stop) the drive, the O.STOP bit should be set ➡ 1 and the O.START bit should be reset ➡ 0.

65

Setting the Direction of the VFD

Two tags also define direction of rotation for the PF40's motor:

O.Forward

O.Reverse

The O.Forward and O.Reverse bits should also always contain opposing values.

To set the drive for <u>forward</u> operation, the O.FORWARD bit should be set ➡ 1 and the O.REVERSE bit should be reset ➡ 0.

To set the drive for <u>reverse</u> operation, the O.REVERSE bit should be set ➡ 1 and the O.FORWARD bit should be reset ➡ 0.

66

Setting the Output Frequency of the VFD

A single tag defines the output frequency of the PF40:

O.FreqCommand

To change the frequency of the waveforms being output by the drive, a new (integer) value should be written into the location defined by the O.FREQCOMMAND tag.

Although the PF40 sets its output frequency based on the value stored in the O.FreqCommand memory location, the PF40 interprets the last digit of value as tenths of a Hertz.

Thus, in the example shown above, if a value of "200" is moved (stored) in the O.FreqCommand memory location, the PF40 will set its output frequency to 20.0 Hertz.

67

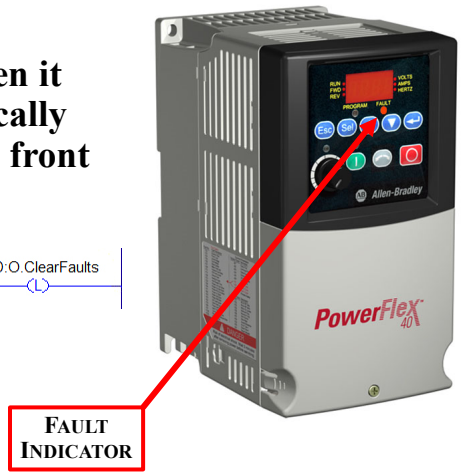
ClearFaults – VFD

The tag:

O.ClearFaults

can be used to clear the drive when it becomes faulted, similar to physically pushing the "Stop" button on the front panel of the PF40.

Warning – The rung shown above will automatically clear any fault that occurs regardless of the fault type. Although this will be useful in the lab environment while programming the PLC, this could lead to hazardous situations in a live (industrial) setting.



68