

Introduction:

This exercise continues the programming portion of this experiment during which you will utilize the RSLogix 5000 software to develop a second ladder logic program that can be downloaded into the PLC in order to automate the proposed motor control system. Specifically, this program will be used to control the operation of the system’s second Induction motor which is energized by means of a PowerFlex 40 (PF40) Variable Frequency Drive (VFD) that can communicate with the PLC via the system’s Ethernet network.

Procedure:

RSLOGIX 5000 SOFTWARE

Although you could begin this part of the experiment by executing the RSLogix software and creating a “**New Project**” as you did at the beginning of part D, since you are using the exact same motor control system for part E, it is quicker simpler to open the project that you saved on a memory-stick for part D and delete that project’s ladder diagram, after which you will “**Save As**” a new project and then begin building the new ladder diagram.

Note that, by doing this, all of the **tags** that you previously created for the part D program will still exist in the new project for the part E program.


Creating the New Project

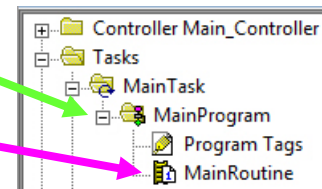
Execute the **RSLogix 5000** software by double-clicking on the **RSLogix 5000** shortcut present on the desktop of the computer or by *clicking* on the **Start** button and choosing:

All Programs→Rockwell Software→RSLogix 5000 Enterprise Series→ RSLogix 5000.

1. **Open** the project that you created and saved on a memory-stick for part D of this experiment, and then once the project is open, choose the **Save As** option under the **File** menu in the **RSLogix** window and rename your project:

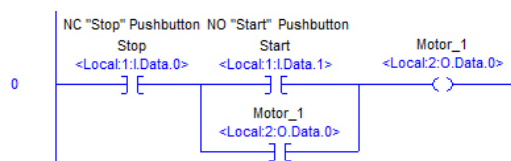
“**Lastname_Controller_P2.ACD**”

2. In the **Controller Organizer** window, expand the **MainProgram** folder by *clicking*  next to the folder.



3. **Double-click** the **MainRoutine** icon to open the **Routine Editor** window

4. The **Routine Editor** window should now display the ladder diagram that you created for part D:



5. **Click** on the rung number to the left of the rung to highlight the rung and then **press** the **Delete** key to delete the rung, after which only the “End” rung should remain in the ladder.


Notes Regarding the Operation of the Motor Control System for Program #2:

The PLC-based motor control system constructed for use during this experiment contains two induction motors, the first of which is energized by means of a contactor whose field-coil is wired directly to the PLC’s output module, and the second of which is supplied by a PowerFlex 40 VFD that will be controlled remotely by the PLC via the control system’s Ethernet network.

During part D of this experiment, you created a simple program that provided “Start-Stop” control over the operation of motor #1. Now you will create a second program that allows the PLC to control the operation of **motor #2** by means of the **PF40** VFD.

In this case, when **START** is pressed, the VFD will become energized, and it will begin supplying its motor with a set of three-phase voltages for a total of 22 seconds. Initially, the frequency of the voltages will increase from 0Hz up to 30Hz and the VFD will supply to motor at this frequency for a total of 10 seconds (including the acceleration time), after which the frequency of the voltages will increase to 60Hz and the VFD will continue operating at this new frequency for an additional 12 seconds (including the acceleration time).

Once the entire 22-second process is complete, the frequency of the voltages will automatically decrease to 0Hz and the VFD will be de-energized. But if the **STOP** button is pressed at any time during this process, the process will terminate, the frequency of the voltages will decrease to 0Hz and the VFD will be de-energized.

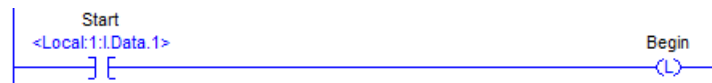
6. Add a new rung by *clicking* on the rung icon  in the *New Component* toolbar.

7. Add an *XIC* instruction to the left side of the new rung.

8. *Double-click* on the “?” above the *XIC* to enable a drop-down menu that contains all of the previously-defined *tags*. Expand that menu, *double-click* on the alias tag **Start** in that list, and then press “**Enter**” to assign it to the *XIC*.



9. Add an *OTL* to the right-side of the rung and create a New Tag named “**Begin**” for that *OTL*.



OTL Operation: When the rung condition to the left of the *OTL* becomes true, the *OTL* will set the bit associated with its assigned tag to a one (1), but when the rung condition to the left of the *OTL* becomes false, the *OTL* does nothing.

In other words, the *OTL* can set the bit but it cannot reset the bit.

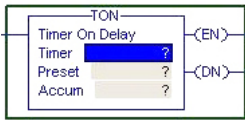
OTU Operation: When the rung condition to the left of the *OTU* becomes true, the *OTU* will reset the bit associated with its assigned tag to a zero (0), but when the rung condition to the left of the *OTU* becomes false, the *OTU* does nothing.

In other words, the *OTU* can reset the bit but it cannot set the bit.

Since **Begin** is a Boolean tag, it relates to a single bit of memory. This bit will be used within the program as a **Status Bit**, the value of which will determine the operational state of the system. In this case, when **Begin** is set to a “1”, the system will begin operating and the motor will be energized for a total of 22 seconds unless **STOP** is pressed to terminate the process early.

10. Add a new rung below the existing rung.

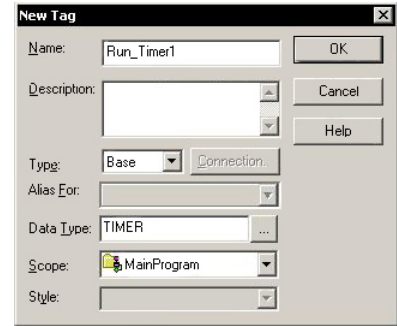
11. Add an *XIC* instruction to the left side of the new rung and assign it the tag *Begin*.



12. Select the “*Timer/Counter*” tab in the *New Component* toolbar.

13. Select a “*TON*” (*Turn-On Delay*) timer and place it to the right of the *XIC* on the new rung.

14. Right-click the “?” in the *Timer* field of the *TON* and create a New Tag named *Run_Timer1* for the timer as shown to the right. Click **OK** when done.

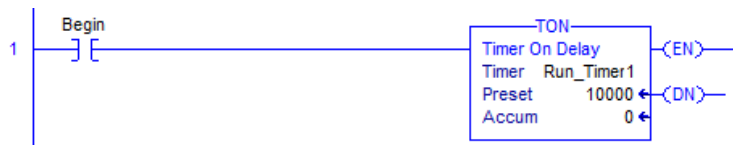


The ?’s that initially appeared in both the *Preset* and the *Accum* fields should now be replaced by 0’s (zeros).

15. Double-click the “0” in the *Preset* field of the *TON* and enter the value **10000**.

The *Preset* value is the time delay in **msec**, therefore the value **10000** relates to a time delay of **10 seconds**.

16. Rung 1 should now appear as follows:



The timer on this rung will be used to control the first 10-seconds of the motor’s operation.

TON Operation: When the rung condition to the left of the *TON* becomes true, the timer will be enabled, and its *Enable (EN)* bit will be set. The timer will then begin incrementing the value in its *Accumulator (Accum)* in real-time from the initial value stored in the *Accum* up its *Preset* value, at which time it stops counting and the *Done (DN)* bit is set.

The timer will remain in this state until the rung condition becomes false, at which time both the *EN* bit and the *DN* bit will be reset and the *Accum* will revert back to its initially defined value.

Note that, while it is enabled, the *TON’s Accum* can also be reset back to its initial value by utilizing the *RES* (reset) instruction that resides on the “*Timer/Counter*” tab in the *New Component* toolbar.

Controlling the Operation of the PowerFlex 40 Variable Frequency Drive:

When the *PowerFlex_40* was added into the initial project that you created at the beginning of part D of this experiment, the *RSLogix* software automatically created a set of base tags that refer to information that is stored in the *PF40’s* memory, similar to the base tags that the software created for the bits contained in both the *IQ-16* input module and the *OW-16* output module. But in this case, *PF40* utilizes the values stored in its memory that are referenced by these software-created base tags to define its exact operational state.

The tags that we will utilize during this part of the experiment are:

O.Start
O.Stop
O.FreqCommand

Note that when looking for the tags associated with the *PF40's* operation in the list of previously-created base tags or within a ladder diagram, the tags will all begin with either *PowerFlex_40:I* or *PowerFlex_40:O*. The *PowerFlex_40:* that appears before each of the tag names identifies the device that contains the memory in which the information identified by these tags is located (you named the drive “*PowerFlex_40*” when you added that module to the project), and the first part of the actual tag names will begin with *I* or *O* because the information referenced by these tags relates to either Input or Output functions of the drive.

Although we will not utilize them during this part of the experiment, there are a variety of other tags associated with the *PF40's* operation that you might find useful, including:

O.Forward
O.Reverse
O.ClearFaults
I.Faulted

Thus, by assigning these tags to various instructions within a ladder diagram, we can program the PLC to control the operation of the *PF40*.

PF40 Operation: The *PowerFlex_40* must be **enabled** before it can produce a set of variable frequency voltages for its Induction motor. By default, this is accomplished by pressing the green “**Start**” button on the *PF40's* front panel. Similarly, the drive can be **disabled** by pressing the red “**Stop**” button on its front panel.

But, instead of having these buttons directly energize or de-energized the drive’s circuitry, the drive’s memory contains two bits that are identified by the tags *O.Start* and *O.Stop*, and the drive references the values stored in these bit locations in order to determine its operational status.

Thus, pressing the green “**Start**” button typically causes the *O.Start* bit to be set to a **1** and the *O.Stop* bit to be reset to a **0**, in-turn triggering the drive to activate the PWM circuitry that creates the variable-frequency set of voltages that the drive supplies to connected motor (i.e. – “start the drive”).

On the other hand, pressing the red “**Stop**” button causes the *O.Start* bit to be reset to a **0** and the *O.Stop* bit to be set to a **1**, in-turn triggering the drive to (by default) lower the frequency of its output voltages to zero and then to deactivate the PWM circuitry (i.e. – “stop the drive”).

For safety reasons, pressing the “**Stop**” button will always cause the *O.Start* bit to be reset and the *O.Stop* bit to be set, in-turn deactivating the drive.

But it may not be desirable to always have the “**Start**” button activate the drive, especially if that drive is part of a larger, automated motor-control system in which a manual (unexpected) start of the drive may either pose a safety risk or cause problems with respect to the entire system’s operation. For this reason, the drive can be configured to receive a **Start** signal from a variety of sources, one of which is the “**Start**” button on the *PF40's* panel.

Previously, in part B of this experiment, you changed the value of the drive's Basic Program Group parameter **P036** from a **0** to a **5**, and it is this value that the drive utilizes in order to determine the **Start** "source". The **Start** sources referenced by those two values are:

0 – "Keypad" (default value)

5 – "Comm Port"

Since the value of the parameter was changed to a **5**, the drive will no longer recognize the "**Start**" button on its panel. Instead, the drive will now only accept **Start** commands via its communication (Ethernet) port.

But as stated above, even though the drive was configured to receive a **Start** command from its communication port, the **PF40** will accept a **Stop** command from either its communication port or by means of the red "**Stop**" button on the **PF40's** front panel.

Since the tag **PowerFlex_40:O.Start** was already created by the **RSLogix** software when the **PF40** was initially added to the project, we can remotely cause that bit to be set by assigning that tag to the appropriate output instruction in our program. Specifically, we will assign the tag to an **OTL** such that, when the rung condition for that **OTL** becomes **TRUE**, the **PLC** will send a message to the **PF40** (at the IP Address specified when adding the drive to the project) via the Ethernet network, instructing the **PF40** to **set** the "**O.Start**" bit (to **enable** the drive).

And since the **PF40** also contains an "**O.Stop**" bit that must be **reset** whenever the "**O.Start**" bit is set (the **O.Start** and **O.Stop** bits should always have opposite values), we will also assign the "**O.Stop**" bit to an **OTU** on the same rung with the "**O.Start**" **OTL**.

Similarly, the **PowerFlex_40** can be **disabled** by setting the "**O.Stop**" bit and resetting the "**O.Start**" bit (i.e. – by assigning both the "**O.Start**" bit to an **OTU** and the "**O.Start**" bit to an **OTU** on the same rung).

Once the drive has been enabled, it will begin to create and output a set of three-phase voltages, the frequency of which is determined by the value stored in the **PF40** memory location referenced by the tag "**O.FreqCommand**".

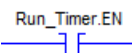
Specifically, the value stored in the tag location "**O.FreqCommand**" must be an integer number ranging from 0 → 600. This value relates to the output frequency of the drive specified in tenths of Hertz. Thus, if the value "**600**" is stored in that location, the drive will raise the frequency of its output voltages to **60.0Hz** when it is **enabled**.

We will utilize **MOV** instructions within our program change the values stored in the **PF40's** memory location referenced by **O.FreqCommand** in order to set the drive's frequency to the required values.

17. Add a rung to the bottom of your ladder diagram and place an **XIC** instruction on that rung.

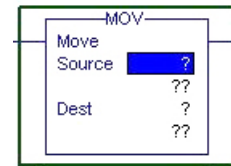
18. **Double-click** on the "?" above the **XIC**, **click** on the **down-arrow** that appears in the **Tag** field, and then **expand** the group of **Run_Timer** tags within the list of tags that appear.

Assign the **Run_Timer.EN** bit of the timer to the **XIC** by **double-clicking** on that tag, and then press "**Enter**".



19. Add a **MOV** instruction to the right of the **XIC**.

The **MOV** instruction resides under the “**Move/Logical**” tab of the **New Components** toolbar.

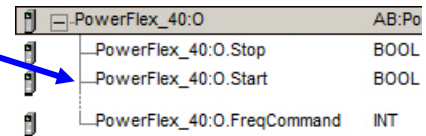


MOV Operation: When the rung condition to the left of the **MOV** becomes **TRUE**, the **MOV** instruction copies the data located in or specified by the **Source** field into the location specified by the **Destination** field (such as a frequency value into the **PF40’s “O.FreqCommand”** register).

Note that, when locations are defined in either the **Source** or **Destination** fields, the **MOV** instruction will display the values currently stored at those locations within the “??” field that appear immediately below the **Source** and **Destination** fields. Otherwise, the “??” fields will be blank.

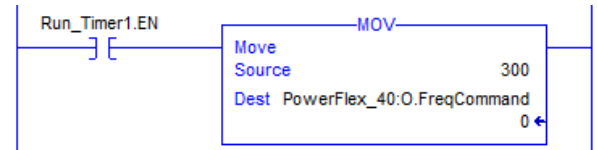
20. Since the **PF40** must first be set to supply a set of **30Hz** voltages to the motor, **double-click** the “?” in the **Source** field of the **MOV** instruction and enter the value **300**.

21. **Double-click** the “?” in the **Destination** field of the **MOV** instruction and locate “**PowerFlex_40:O.FreqCommand**” in the drop-down menu that appears.



Double-click “**PowerFlex_40:O.FreqCommand**” to assign it to the **Destination** field.

Rung 3 should now appear as shown to the right:

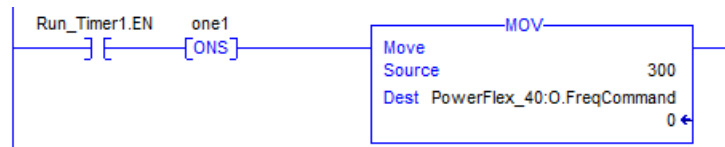


When the timer is enabled, the **Run_Timer.EN XIC** will become **TRUE** and the **MOV** instruction will copy the value **300** into the destination location defined by the tag **O.FreqCommand**.

Note that, as configured, as long as the timer remains enabled, the **MOV** instruction will continuously copy the value **300** into the **O.FreqCommand** destination. But we only need this to occur one time when the timer is first enabled. To accomplish this, we will utilize an **ONS** instruction.

22. Add a “**One Shot**” (**ONS**) to the right of the **XIC** and define a New Tag to name it “**one1**”.

Note – the **ONS** instruction is located under the “**Bit**” tab.

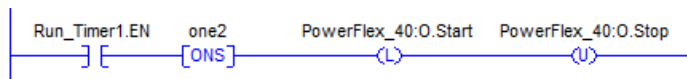


ONS Operation: When the rung condition to the left of the **ONS** becomes **TRUE**, the **ONS** is triggered, making the **ONS** **TRUE** for that one scan of that rung, and the rung will be evaluated as such for that scan of the rung.

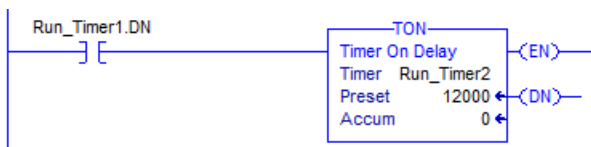
But, after the **ONS** is triggered, the **ONS** will return a **FALSE** condition during any successive scans of the rung until the **ONS** is reset.

In order to reset the **ONS** (so it can be triggered again), the rung condition to the left of the **ONS** must become **FALSE** for at least one scan of the rung, after which it can return **TRUE** to trigger the **ONS** again for an additional scan.

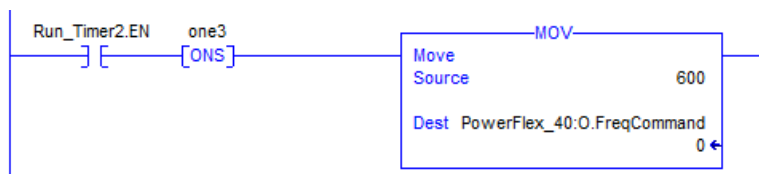
23. Add the following rung to also **enable** the drive when *Run_Timer1* is enabled:



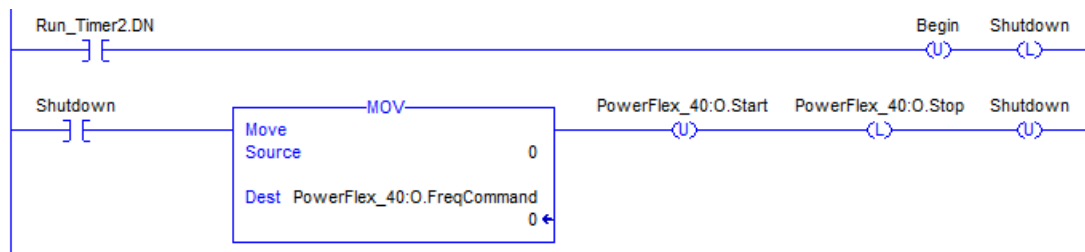
24. Add the following rung to **enable** a 12 second *Run_Timer2* when *Run_Timer1* is “done”:



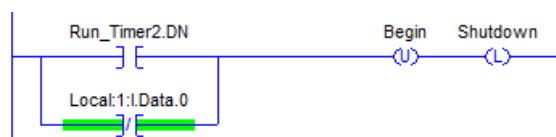
25. Add the following rung to increase the drive frequency to **60Hz** when *Run_Timer2* is enabled:



26. Add the following rungs switch (the status bits) from *Begin* to *Shutdown* mode when the **STOP** button is pressed, and to lower the frequency to **0Hz** and **disable** the drive when *Shutdown* is initiated:



27. Add a parallel branch on the following rung and place an *XIO* on the branch that will become **TRUE** and also initiate a *Shutdown* when the **STOP** button is pressed.



28. The ladder logic program should now be complete. Download the program into the PLC (as described in part D of this experiment) and test the program’s operation.

Note – a full image of the complete program appears on the next page.

29. View the last four slides of the “Ladder Logic – Conveyor Example” PowerPoint slides for a discussion on **Initializing Bits** at the beginning of a ladder logic program and add the suggested two rungs at the beginning of your program in order to initialize all of the bits that are being set/reset using OTLs/OTU’s.

