ORCA™ Motors with LabVIEW

User Guide 230713

Version 1.1, February 2026

CONTENTS

REVISION HISTORY

Version	Date	Author	Reason
1.0	July, 2023	rm	Initial Release
1.1	Feb, 2026	rm	ORCA-USB compatibility

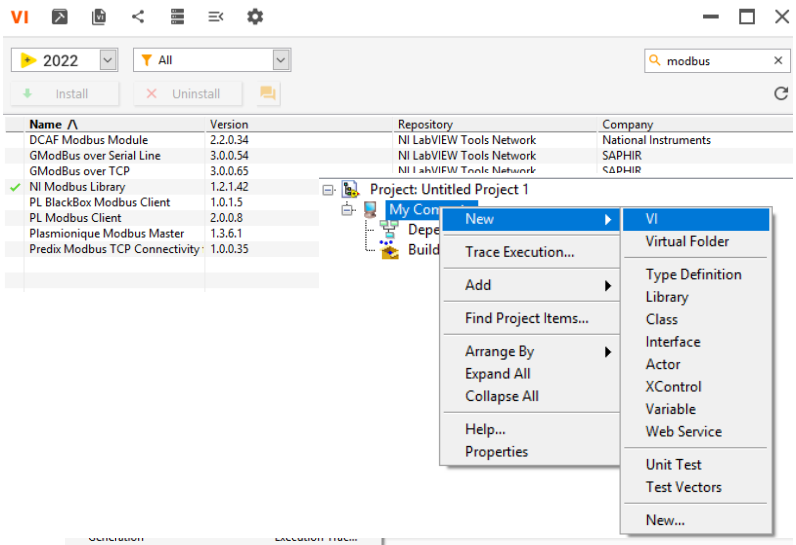
Overview

This is a quick guide to control and / or receive data from ORCA motors using the popular LabVIEW software. While following this example, it is helpful to also have the IrisControls (the ORCA motor's GUI) running for debugging purposes but is not necessary for regular use.

Getting the NI Modbus Library

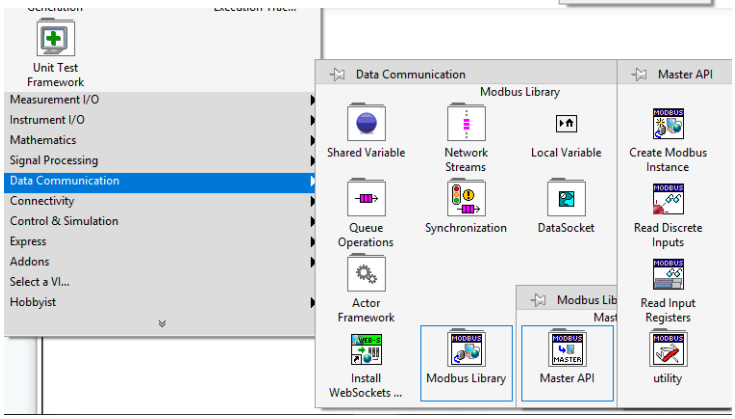
1. Download National Instrument's VI Package Manager: https://www.vipm.io/download/.

2. Search for the NI Modbus Library and install.

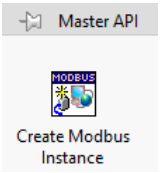


3. Create a new Blank Project and add a new VI.

4. The function blocks used in this guide can be accessed by right clicking in the Block Diagram window and following the path: Data Communication-> Modbus Library -> Master API.



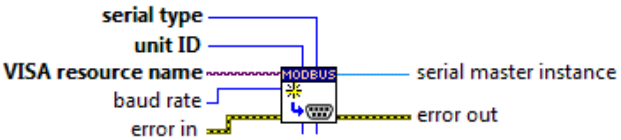
Establish Serial Port

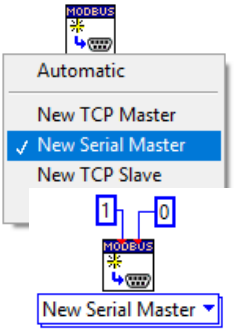


[Block Diagram Window]

1. Right click and find the ‘Create Modbus Instance’ function block in the Master API section.

2. From the dropdown on the block select ‘New Serial Master’.

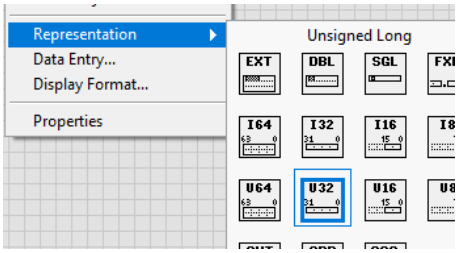




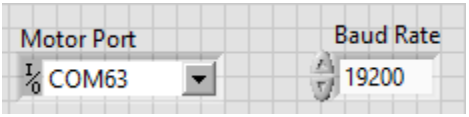
3. Use a ‘Numeric Constant’ for ‘serial type’ and ‘unit ID’ as these will remain the same throughout the example. Setting ‘serial type’ to 0 will use RTU rather than ASCII. The unit ID will be the motor’s server address which defaults to 1 but can be configured.

[Front Panel Window]

4. Add VISA resource control (I/O -> VISA Resource) and Numeric Control (Numeric -> Numeric Control) elements. Right click on the Numeric Control element and change the Representation to U32.

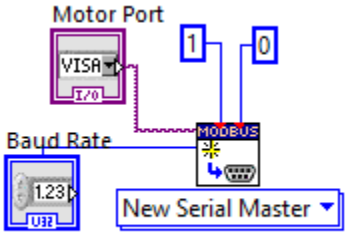


5. The motor uses 19200 as the default baud rate. The serial COM port associated with the motor’s RS422 port should be selected as the VISA resource name.



[Block Diagram Window]

6. Wire the VISA resource name and Baud Rate entries to their respective input on the Serial Master.

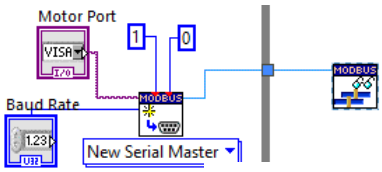


Read and Display Motor Errors

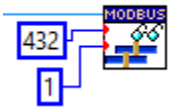
[Block Diagram Window]

1. Add a while loop structure. The loop will run continuously so a False bool can be wired to the conditional terminal.

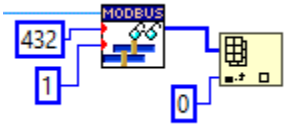
2. Add a 'Read Holding Registers' function block from Modbus Library. The 'Modbus master in' input is wired to the 'Create Modbus Instance' block.



3. To read the motor's active errors, use address 432, ERROR_0 register.
(A full list of the registers can be found in the ORCA's memory map appendix of the ORCA Motor Reference Manual.)

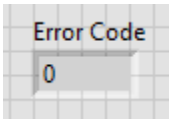


4. The 'register values' output from the read block are in an array. Use the 'Index Array' function block to get the first element in the array.



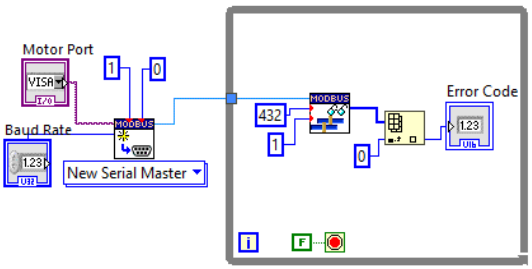
[Front Panel Window]

5. Add a 'Numeric Indicator' to display the ORCA motor's active Error Code. Change the Representation to U16.



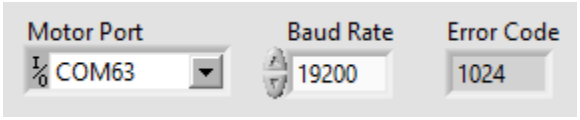
[Block Diagram Window]

6. Wire the 'element' output of the indexed array to the Error Code indicator.



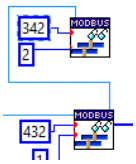
[Front Panel Window]

7. Press the Run button, Modbus messages will begin to be sent to the motor.



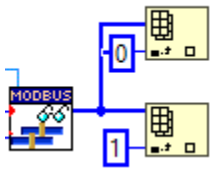
Error Code 1024 is active indicating invalid voltage.

Read and Plot Motor Position

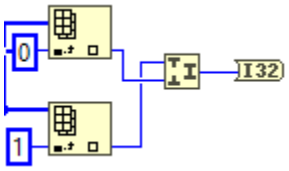


[Block Diagram Window]

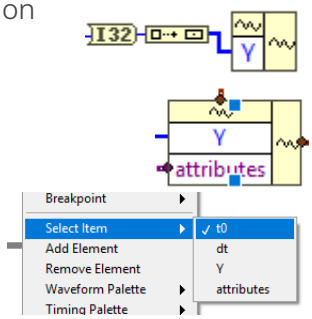
1. Add a second ‘Read Holding Registers’ function block. This one will read the motor’s position data which is spread across 2 registers starting at SHAFT_POS_UM (342). Connect the ‘Modbus master in’ to the error read register block’s ‘Modbus master out’.



2. The output will be a two element array that will need to be joined into a single U32. Add two more ‘Index Array’ function blocks with one indexed to 0 (this is the low U16) and one indexed to 1 (this is the high U16). Wire the register values output array to both of these blocks.

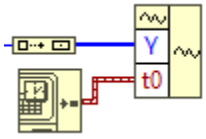


3. Add a ‘Join Numbers’ function block to join two U16 to a single U32 value.The position data is signed, add a ‘Numeric Conversion’ to I32.



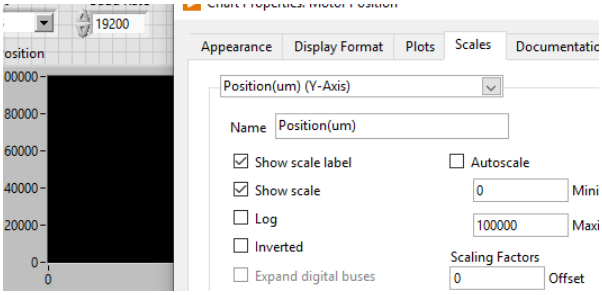
4. To graphically display this data, add the ‘Build Waveform’ function block. The Y input takes a 1D array, add a ‘Build Array’ block to convert the I32 output to a 1D array.

5. Select the ‘Build Waveform’ function block and stretch it down to display one attribute’s input. Right click the attributes input, go to Select Item -> t0.



6. Add a ‘Get Data/Time in Seconds’ function block and wire its output to the input ‘t0’.

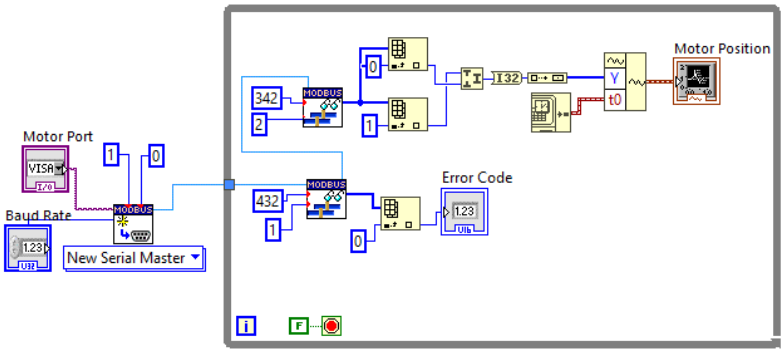
[Front Panel Window]



7. Add a ‘Waveform Chart’. Right click on the chart and select Properties. Go to the Scales tab, select the Y-Axis from the drop down menu and set a minimum of 0 and maximum of 100000. The Y-Axis will represent the position of the motor in micrometers.

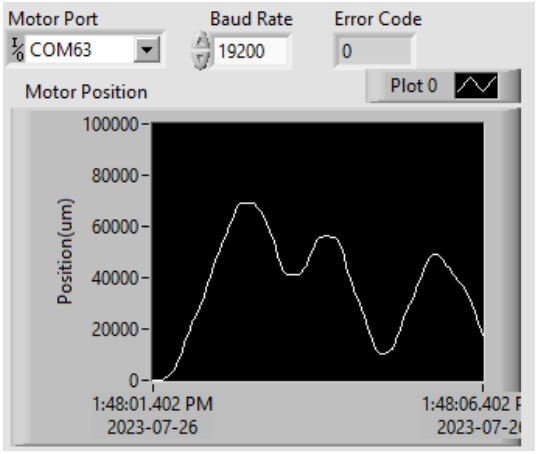
[Block Diagram Window]

8. Connect the ‘Build Waveform’ output to the input of the Motor Position ‘Waveform Chart’.

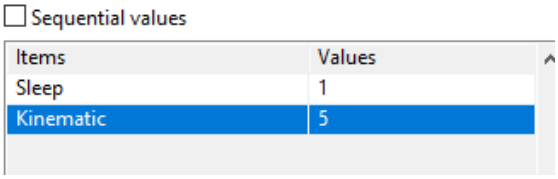


[Front Panel Window]

9. Press the Run button. While moving the motor’s shaft, the position will display on the plot. Note, the motor is now being powered with 24 V and no longer has an Error Code present.

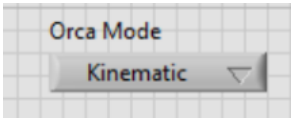


Change Motor Mode of Operation



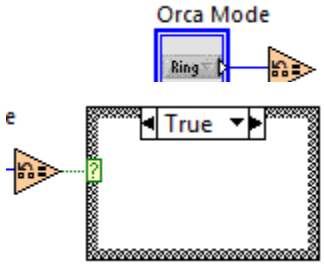
[Front Panel Window]

1. Add a 'Menu Ring' element to select the motor's mode of operation. Right click on it and select Edit Items. Deselect 'Sequential values' and add a Sleep option with a value of 1 and a Kinematic option with the value of 5 (this can be extended to add all Orca Series motor modes).



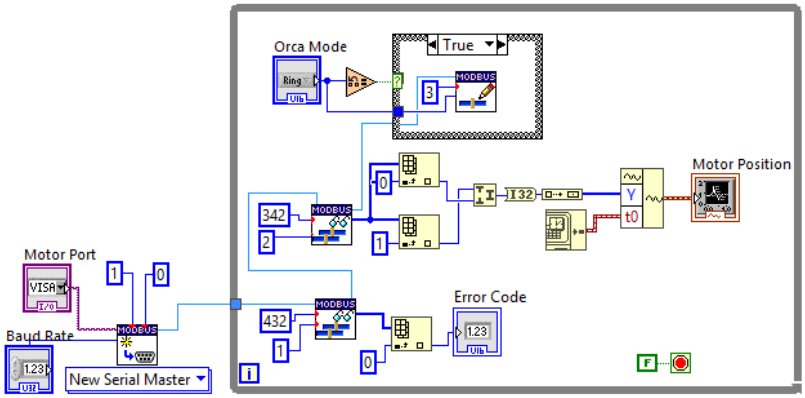
[Block Diagram Window]

2. To check for a change to selected mode, add an 'Is Value Changed' operator and wire the input to the 'ORCA Mode' output.



3. Add a 'Case Structure' and wire the output of the 'Is Value Changed' operator to the case selector.

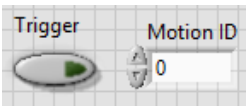
4. Changing the selection should trigger a write to the ORCA motor's Control Register 3 to change the mode of operation. In the True case, add a 'Write Single Holding Register' function block. The address input will be CTRL_REG_3(3), and the output of the ORCAMode 'Menu Ring' will be the value written. Wire the 'Modbus master in' to the previous Modbus read's 'Modbus master out'. In the false case no action is needed.



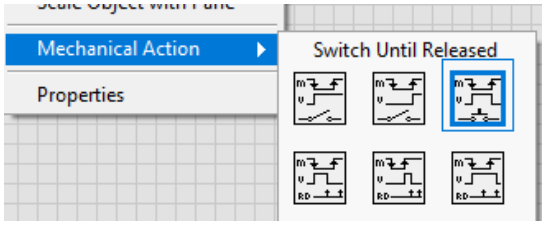
Trigger Motions

[Front Panel Window]

1. Now that Kinematic Mode can be selected. A button will be added to trigger a specified motion ID. Add a 'Push Button' and a 'Numerical Control' element.



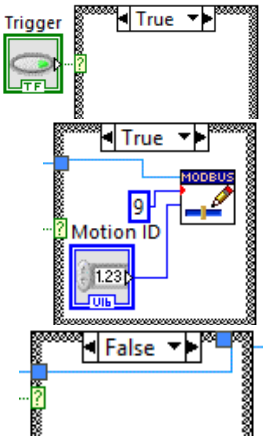
2. Right click the button and change the Mechanical Action to 'Switch Until Released' to create a momentary button.



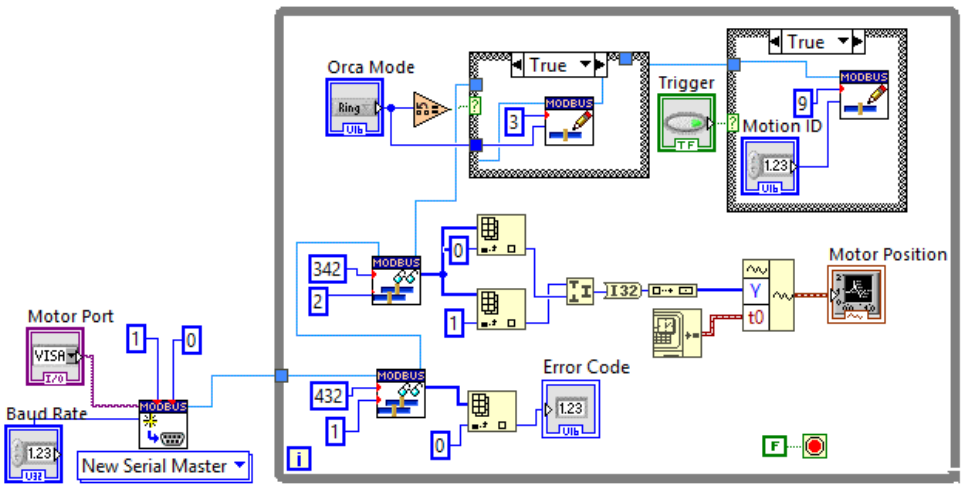
[Block Diagram Window]

3. Add another 'Case Structure' with the 'Trigger' button as the input to the selector.

4. In the True case, add a 'Write Single Holding Register' block. The address input will be KIN_SW_TRIGGER(9). The Motion ID will wire to the register to write input.

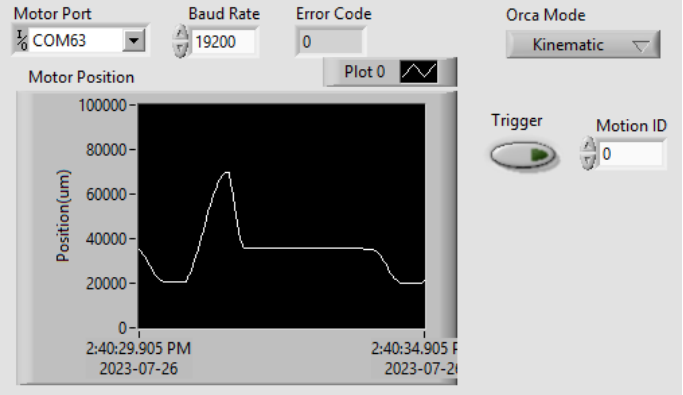


5. Wire the 'Modbus master in' to the previous Modbus read's 'Modbus master out'. Ensure the wire is also connected through the False case of the previous structure.



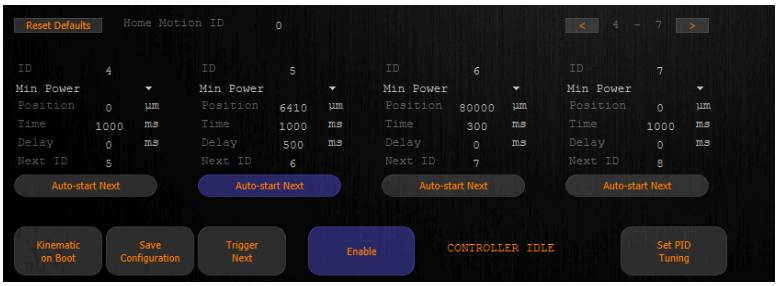
[Front Panel Window]

6. Press the Run button. Set the ‘ORCA Mode’ to ‘Kinematic’.
7. With the motor’s default Kinematic Mode motions configured, pressing the ‘Trigger’ button with ‘Motion ID’ 0 will move the motor through a set of positions then stop and hold until triggered again.



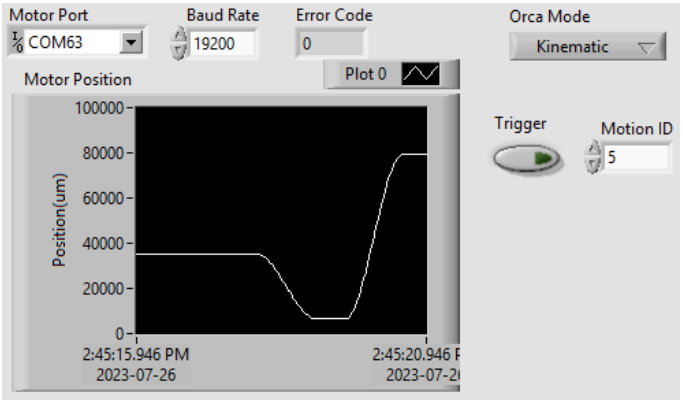
[IrisControls]

8. Additional motions can be set up and saved through the ORCA GUI in IrisControls.



[Front Panel Window]

9. Trigger the newly configured motion through LabVIEW.



Increase Frame Rate

If you are interested in streaming commands to the motor at a faster rate, the Baud Rate can be increased to 1250000 which will change the frequency from ~60 Hz to ~160 Hz. This can be done by changing the Default Baud Rate on the motor and saving the value and then changing the baud rate on the LabVIEW side to match.

Limited Performance

ORCA motors feature custom Modbus function codes which are not compatible with the LabVIEW Modbus library. Without the ORCA motor's function codes, communications are limited to ~160 Hz.

Experiencing Modbus Errors

In order to continue running with occasional missed messages due to broken usb packets etc, the Modbus error code can be checked and cleared if it is a simple missed response message to prevent the program from stopping on a single failed message.