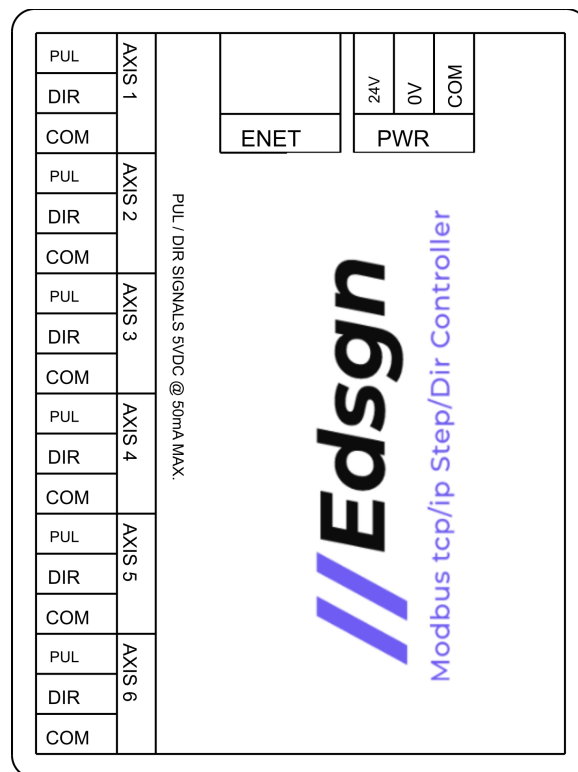


# Edsgn Multi-Axis Modbus TCP/IP Coordinated Motion Controller V1.1



# USER MANUAL

## 1. Product Overview

Thank you for choosing Edsgn Ethernet based Step and Direction Motion Controller. We are please that our products are going to let you successfully complete your motion control project. The Edsgn driver uses MODBUS /TCP IP communication protocol to allow high speed communications with almost any Ethernet equipped PLC.

It is an integration of a high performance motion controller and high speed communications controller. Built in features include up to 6 axis of fully coordinated motion axis. Any axis not connected to a physical drive can also be used as a coordinated virtual axis. Up to 15 pre-planned motions can be used to transition from one motion to the next. Fully configurable steps per user units allow for scaling all motion units to your specific machine characteristics. Rather than always converting your engineering units to some step count set your steps per user units once and see actual positions and units MM / REVOLUTION / INCH / ETC.

The controller uses a standard Ethernet interface and is compatible with 10M/100M bps network interfaces. Connect to the controller with standard Ethernet media rather than special serial cables or wires. Allows for controller to be positioned next to the drives for shorter Step/Direction signal wires. Better communications with more cost effective media.

Standard 24VDC input from a Class 2 Power Supply. Din Rail mountable, with oversized screw terminal wire connectors for a robust easy to use interface.

## 2. Controller Electrical Characteristics

- **Power Supply** 20-28VDC (Typical 24VDC) Class 2
- **Input Current** Maximum 2.0A
- **Maximum Output Frequency** 30 Khz (Coordinated) – Up to\* 100 Khz (single axis moves)
- **Communications Interface** 1 x standard RJ45 port for 10/100M Modbus TCP/IP
- **Outputs** 5VDC Sourcing (max. 30MA per channel)

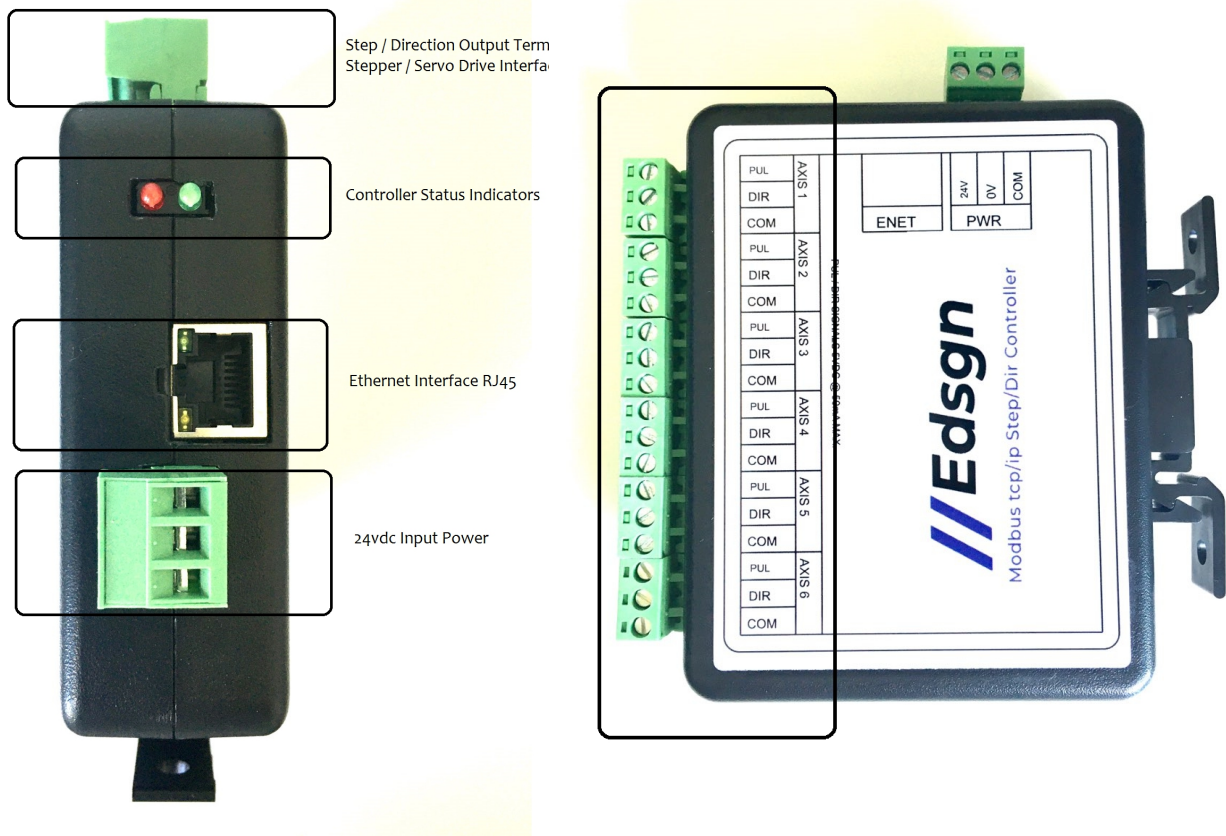
\* maximum output frequency depends on many physical attributes of your system. To maximize frequency keep lines as short as possible. Isolate from any “Noisy” AC power conductors. Use shielded cables.

### 3. Safety

The transportation, installation, use or maintenance of this product must be carried out by personnel who are professionally qualified and familiar with the above operations. In order to minimize potential safety hazards, you should comply with all local and national safety regulations when using this equipment. Different areas have different safety regulations. You should ensure that the equipment is installed and used in accordance with your region. Specification. System errors can also result in equipment damage or personal injury. We do not guarantee that this product is suitable for your specific application and we cannot be held responsible for the reliability of your system design. Be sure to read all relevant documents before installation and use. Improper use may result in equipment damage or personal injury. Please strictly observe the relevant technical requirements during installation. Be sure to confirm the grounding of each device in the system. Un-grounded systems cannot guarantee safe use of electricity. Some components inside the product may be damaged by external static electricity. Before touching the product, the operator should ensure that there is no static electricity and avoid contact with objects that are easily electro-statically charged (chemical fibre, plastic film, etc.). If your device is placed in the control cabinet, please close the control cabinet cover or the door during operation, otherwise it may cause equipment damage or personal injury. Wait at least 6 seconds after turning off the power to touch the product or remove the wiring. Capacitive devices may still store dangerous electrical energy after power failure and require some time to release. To ensure safety, you can measure it with a multimeter before touching the product. Please observe the safety instructions presented in this manual, including clear warning symbols for potential safety hazards, and read and become familiar with these instructions before installation, operation and maintenance. The purpose of this paragraph is to inform users of the necessary safety instructions and to reduce the risk of personal injury and equipment safety. Miscalculations of the importance of safety prevention can cause serious damage or render the device unusable.

## 4. Hardware Connections

The hardware connectors are as follows:



## 5. Power supply connection

Connecting the controller to a 24V Class 2 DC power supply: 24V is connected to the 24VDC power supply **positive**, 0V is connected DC power supply is **negative**. The input voltage must be between **20~28VDC**. Do not exceed this specification. If your power output does not have a fuse or some other device that limits the short-circuit current, place a properly sized fast-blow fuse (no more than 2Amps) between the power supply and the controller to protect the controller supply wiring and downstream equipment.

**!!! Do not reverse the power connections. Controller can be perminatly damaged by improper connections!!!!**

# Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

## 6. Digital Output Ports

The Edsgn Step / Dir Controller has between 2 and 12 digital outputs depending on the model. Digital output ports are configured for specific axis control. The pin connections to the controller are made via the oversized 12-22 AWG 3 Pole screw terminals. Connectors are as follows:

| CONNECTOR | PIN | NAME         | DESCRIPTION   |
|-----------|-----|--------------|---|
| 1         | 1   | STEP AXIS 1  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 1 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |
| 2         | 1   | STEP AXIS 2  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 2 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |
| 3         | 1   | STEP AXIS 3  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 3 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |
| 4         | 1   | STEP AXIS 4  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 4 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |
| 5         | 1   | STEP AXIS 5  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 5 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |
| 6         | 1   | STEP AXIS 6  | Step Output Signal 5VDC 30mA Max. Supply                                      |
|           | 2   | DIR . AXIS 6 | Direction Output Signal 5VDC 30mA Max. Supply                                 |
|           | 3   | COM          | Internally Connected to 0V / Ground Use to connect to external drive COM / 0V |

## 7. Network connection and IP address settings

Before you start, please confirm that you have the following components:

- A small slotted screwdriver for tightening the connector screws
- A computer with a Modbus Client software for testing (Modbus Tester by Schneider Electric works well)
- A 24VDC power supply
- A network cable to connect the motion controller to the computer

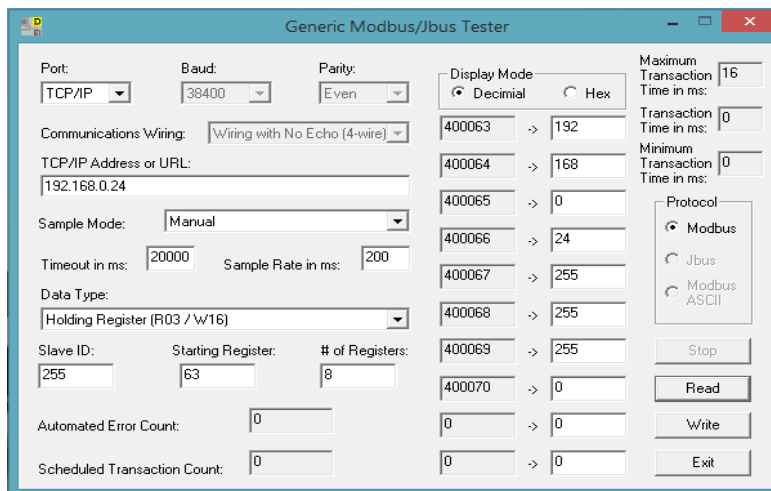
# Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

## 7.1 Connecting your controller to your computer using Ethernet

The RJ-45 connector on the EPR60 drive is a 100BASE-TX (100Mbps) compliant interface that can be connected using a standard network cable. Please use a CAT5 or CAT5e (or higher) network cable.

The process of connecting to the Edsgn Motion Controller from a computer requires these steps:

1. Set the computer IP address and Sub-net address to a valid range to match the default address of your motion controller. There are many well document procedures on how to do this. Just consult the internet.
  1. Default IP 192.168.0.26 (6 axis version), 192.168.0.24 (4 axis version), 192.168.0.23 (3 axis version)
  2. Default Sub-net is 255.255.255.0
  3. Gateway IP is by always xxx.xxx.xxx.1 this cannot be changed.
2. Download and install Schneider Electric Modbus Tester software. Open Tester software. Connect RJ-45 cable from computer Ethernet Port to Edsgn RJ-45 Port.
3. Connect 24V DC power to Edsgn Motion Controller and power ON.



### USE THE FOLLOWING SETTINGS TO CONFIRM YOU ARE CONNECTED:

Port: TCP/IP

TCP/IP Address of URL: Enter your controllers default IP address (192.168.0.26, 192.168.0.24, 192.168.0.23)

Sample Mode: Set to Manual

Data Type: Holding Register (R03 / W16)

Slave ID: 255

Starting Register: 63

# of Registers: 8

Then Press the “READ” button on the right hand side. If your connected you will see the IP address and Sub-net values populate the registers.

## 7.2 Testing a Motion Instruction Command

**\*\*\*You do not need to be connected to a drive in order to test these commands. We recommend not connecting to a physical axis unit you have confirmed a working connection\*\*\*\*\***

**USE THE FOLLOWING SETTINGS TO ISSUE A MOTION INSTRUCTION:**

Port: TCP/IP

TCP/IP Address of URL: Enter your controllers default IP address (192.168.0.26, 192.168.0.24, 192.168.0.23)

Sample Mode: Set to Manual

Data Type: Holding Register (R03 / W16)

Slave ID: 255

Starting Register: 1

# of Registers: 5

Set display Mode to “Decimal”

Enter the Following into the Modbus Register Boxes.

400001 – 1 (Move ID Number)

400002 – 1 (Enable Controller / Absolute Move / Coordinate Axis)

400003 – 12 (Move Velocity Units/Second)

400004 – 0 (Go-to Position Float High)

400005 – 17500 (Go-to Position Float Low) (Combined Data 400004 and 400005 = 880.00 Units)

Then Press the “Write” button on the right hand side. If your connected you will see the Green Status LED on your Edsgn Motion controller start flashing, indicating that a move is in process.

Generic Modbus/Jbus Tester

Port: TCP/IP Baud: 38400 Parity: Even Display Mode:  Decimal  Hex

Communications Wiring: Wiring with No Echo (4-wire)

TCP/IP Address or URL: 192.168.0.24

Sample Mode: Manual

Timeout in ms: 20000 Sample Rate in ms: 200

Data Type: Holding Register (R03 / W16)

Slave ID: 255 Starting Register: 1 # of Registers: 5

Automated Error Count: 0

Scheduled Transaction Count: 0

Maximum Transaction Time in ms: 15

Transaction Time in ms: 15

Minimum Transaction Time in ms: 0

Protocol:  Modbus  Jbus  Modbus ASCII

Stop Read Write Exit

|        |    |       |
|--------|----|-------|
| 400001 | -> | 1     |
| 400002 | -> | 1     |
| 400003 | -> | 2     |
| 400004 | -> | 0     |
| 400005 | -> | 17500 |
| 0      | -> | 0     |
| 0      | -> | 0     |
| 0      | -> | 0     |
| 0      | -> | 0     |
| 0      | -> | 0     |

## 7.2 Testing a Status Update Command

USE THE FOLLOWING SETTINGS TO RECEIVE CURRENT POSITION AND MOVE STATUS UPDATES:

Port: TCP/IP

TCP/IP Address of URL: Enter your controllers default IP address (192.168.0.26, 192.168.0.24, 192.168.0.23)

Sample Mode: Set to Manual OR Scheduled

Sample Rate in ms: 50

Data Type: Holding Register (R03 / W16)

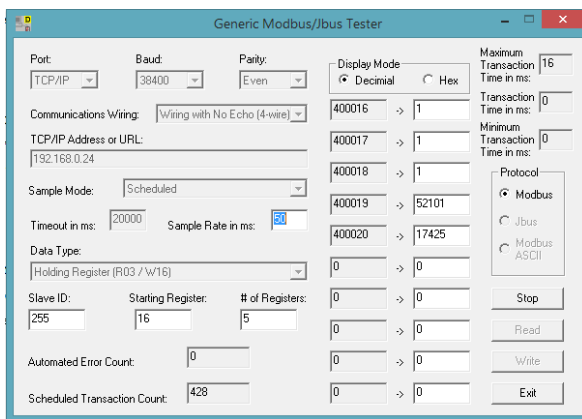
Slave ID: 255

Starting Register: 16

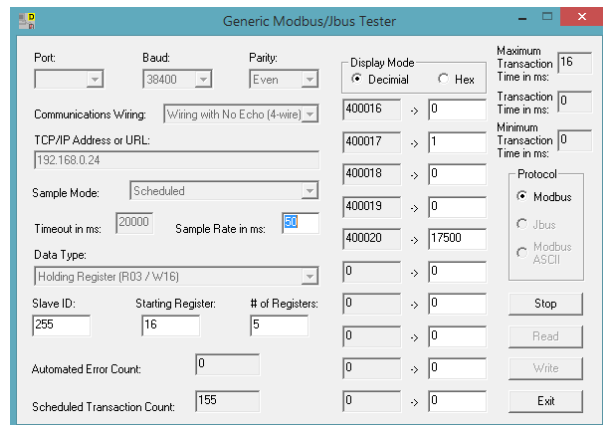
# of Registers: 5

Set display Mode to “Decimal”

Then Press the “Read” button on the right hand side. If your connected you will see the register data be populated by the controller.



(Screen Shot of Move In Process)



(Screen Shot after Move Completed – No additional Instructions)

### Register Details

400016 – Move ID of Move In Process (Move ID value 1-255 in motion, 0 if Stopped)

400017 – Move ID of last Move received by Motion Controller (Can buffer up-to 15 move ahead)

400018 – Motion Status (0 – Stopped, 1 - Running)

400019 – Current position Float Data High Bytes

400020 – Current position Float Data Low Bytes

\*\*\*\*\*See section below for full list of EDSGN Controller Registers.

## 8. Detailed Register List

All registers are 16 bits long. Some are joined together to form 32 bit floats. Example if you are sending the controller a Go-to position. You will have to copy the float data into 2 x 16 bit signed integers. Same as if you are monitoring the current position value. You will have to re-pack the 2xintegers into a float data type.

# Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

| Address | Parameter Type | Type                            | Values           | Units          | Default Values | Description  |
|---------|----------------|---------------------------------|------------------|----------------|----------------|--|
| 40001   | Write          | Move ID Number                  | 0-255            | -              | 0              | Can be used to track moves issued to the controller. Give each motion instruction a Move ID number to send along with the Move data. The controller will update Register [400016] when move is actively being executed. It will also update Register [400017] when move parameters have been added to the internal move buffer and are waiting to be executed. |
| 40002   | Write          | Move Command Word               | 0-7              | -              | 0              | Bit Addressing 16 bit word.<br>Bit[0] Controller Enable 0 = Fast Stop All Axis, 1 = Enabled<br>Bit[1] Absolute or Relative 0 = Absolute Move, 1 Relative Move<br>Bit[2] Coordinated or Independent Move 0 = Coordinated 1 = Independent<br>Bit[3] - [15] Not in use  |
| 40003   | Write          | Velocity                        | 0-32767          | User Units/sec | 0              | Velocity in Units per Second. Only used for Coordinated move instructions. If controller is using independent move configuration [40002 bit[2] = 1] moves will be done at Maximum velocity setting for the axis.   |
| 40004   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40005   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40006   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40007   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40008   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40009   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40010   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40011   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40012   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40013   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40014   | Write          | Target Position Float High Byte | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40015   | Write          | Target Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Target position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The target position used in the controller is a 32 bit float reconstructed from the High Byte and Low Byte address for the Axis.  |
| 40016   | Read           | Active Move ID Number           | 0-255            | -              | 0              | Used to feedback move in process from the controller. Each Motion instruction requires a separate Move ID number to be sent along with the Move data. Upto 15 motion instructions can be sent to the controller to preplan motion paths. The controller will update Register [400016] when a move is currently being executed by the controller.               |
| 40017   | Read           | Last Buffered Move ID           | 0-255            | -              | 0              | Reports back the Motion Instruction ID number that was last received and put into the internal execution buffer.   |
| 40018   | Read           | Motion Status Word              | 0-2              | -              | 0              | Bit Addressing 16 bit word.<br>Bit[0] Motion Status 0 = Stopped, 1 = Move in Progress<br>Bit[1] Enable Status 0 = Disabled, 1 = Controller Enabled   |
| 40019   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40020   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40021   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40022   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40023   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40024   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40025   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40026   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40027   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40028   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40029   | Read           | Actual Position Float High Byte | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |
| 40030   | Read           | Actual Position Float Low Byte  | -32768 to +32767 | User Units     | 0              | Current position of the Axis in real time. Actual position of the Axis is transmitted through modbus registers. Each register is a signed 16 bit integer. The actual position sent from the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis.   |





## 9.0 Additional Details of Registers

Notes on Registers : There are 2 types of registers. Process Data and Parameter Data.

1. **Process Data (Read / Write) – Use to update target positions and read actual position updates**

**Write Data - The first block of registers is designed to be sent as a group. Depending on your axis count you should send data to registers [400001] to [400009] for a 3 axis controller, or [400001] to [400015] for a 6 axis in a single write instruction. These registers contain all data in order to perform a move command. The registers were specially organized for this purpose.**

**Read Data – The second block of registers is designed to be read frequently. These provide motion state as well as live axis position data. You should set up your read instruction to poll [400016] to [400030] for a 6 axis, [400016] to [400024] for a 3 axis as a single group read. The registers were specially organized for this purpose.**

2. **Parameter Data – Can be written to or read in single or multiple messages. These are only for setup and troubleshooting purposes.**

**Should only send to registers when no move is in process. Registers can be read at any time.**

\*\*\*\*\*Start of Process Write Data\*\*\*\*\*

### Register [400001] – Move ID Number – Write Data

This only needs to be sent when adding motion instructions to the controllers motion planner / buffer or to execute a move. Parameter instructions do not require an ID. The Move ID Number is used to track the motions that have been added to the motion planner, also what move is currently being performed. The move ID number can be any integer value between 1 and 255. Typically you would issue each separate motion instruction with a specific ID number. Then monitor register[400017] and the motion controller will put the last received move ID number into that register. Once you have received confirmation that the controller has added your motion instruction to the internal buffer, you can send the next motion instruction and repeat the process. You can have up to 15 motion instructions loaded into the planner at any given time. The move ID number currently being executed will be loaded into register[400016]. By monitoring this register you can see what move is currently active. When all moves have been completed the active move ID number in register[400016] will become 0.

### Register [400002] – Move Command Word - Write Data

Bit[0] Enable motion controller. If bit[0] = 0 controller will decelerate all axis to a fast stop and no motion instructions will be processed. Set bit[0] = 1 to internally enable the motion controller. Typically this would be mapped to the safety control within your program. Use an instantaneous feedback contact from the safety relay to control this bit. Use delay contact from the safety relay to control the power to your drives. That way the controller will automatically decelerate the drives without loss of steps during a safety stop condition.

Bit[1] Leave this bit as 0 and controller will perform absolute positioning moves. Set this bit to a 1 and the move instruction will perform a relative move.

Bit[2] Leave this bit as 0 and controller will coordinate all axis moves at target velocity set in register[400003]. Set this bit as a 1 and controller will move each axis independently at the max velocity rate for the axis set for the individual axis.

Bit[4]-[15] Spares

Bit Mapping is MSBF. Example (Enable Drive and perform a Relative Move, Coordinated with all axis INT = 2, 0000000000000011)

### Register [400003] – Coordinated Move Velocity (Units/Sec) – Write Data

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

Only used if “Coordinated / Independent” address[400002] bit [2] is set to 0. This will be the velocity of the axis with the most amount of travel. All other axis will motions will be slower in order to perform a coordinated move all axis will start and the same time and end at the same time with the axis with the most travel moving the fastest and axis with the least amount of travel moving the slowest. Actual velocity units will be derived from the steps/user unit values.

### **Register [400004 – 400005] Axis # 1 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400004] is the high / most significant portion of the 32 bit float, register [40005] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2 registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **Register [400006 – 400007] Axis # 2 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400006] is the high / most significant portion of the 32 bit float, register [40007] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2 registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **Register [400008 – 400009] Axis # 3 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400008] is the high / most significant portion of the 32 bit float, register [40009] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2 registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **Register [400010 – 400011] Axis # 4 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400010] is the high / most significant portion of the 32 bit float, register [400011] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **Register [400012 – 400013] Axis # 5 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400012] is the high / most significant portion of the 32 bit float, register [400013] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2 registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **Register [400014 – 400015] Axis # 6 Target Position – Write Data**

Combine registers to create a float position. Each register is a 16 bit value. Target Positions for the axis are 32 bit floats. In order to send the 32 bit float values using the modbus protocol they must be copied into 2 -16 bit integers. The first register [400014] is the high / most significant portion of the 32 bit float, register [400015] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will separate a float into 2 x integer values. In the PLC use a “Copy / Transfer” instruction to convert the 32 bit float target position to 2x16 bit integers. Add those 2 registers to the modbus send instruction to update the target position in the controller. Even though it is a float value due to the engineering units conversions and internal step / direction pulse counter limitations the largest single move is 32766.99 units. If you attempt to send a long move the controller will fault and require a reset.

### **\*\*\*\*\*Start of Process Read Data Registers\*\*\*\*\***

### **Register [400016] Active Move ID Number – Read Data**

Used to feedback move in process from the controller. Each Motion instruction requires a separate Move ID number to be send along with the Move data. Up-to 15 motion instructions can be sent to the controller to pre-plan motion paths. The controller will update Register [400017] when a move is currently being executed by the controller.

### **Register [400017] Last Buffered Move ID – Read Data**

Up to 15 Motion instructions can be buffered to the controller motion planner. Monitor this register to know when your motion instruction has been received and loaded into the controllers move buffer. If you disable the controller, all moves will be removed from the buffer. Register [400016] will show the last move that was being executed when enable was removed.

### **Register [400018] Motion Status Word – Read Data**

Feedback from the controller. If bit[0] value is = 0 all axis are stopped. If bit[0] value = 1 one or several axis are in motion.

Feedback from the controller. If bit[1] value is = 0 controller is disabled. If bit[1] value = 1 controller is enabled and ready to perform a motion instruction.

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

### **Register [400019 – 400020] Axis # 1 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value. In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400019] is the high / most significant portion of the 32 bit float, register [400020] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

### **Register [400021 – 400022] Axis # 2 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value. In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400021] is the high / most significant portion of the 32 bit float, register [400022] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

### **Register [400023 – 400024] Axis # 3 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value. In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400023] is the high / most significant portion of the 32 bit float, register [400024] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

### **Register [400025 – 400026] Axis # 4 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value. In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400025] is the high / most significant portion of the 32 bit float, register [400026] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

### **Register [400027 – 400028] Axis # 5 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value. In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400027] is the high / most significant portion of the 32 bit float, register [400028] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

### **Register [400029 – 400030] Axis # 6 Current Position – Read Data**

Real time position feedback. The current position is a 32bit float using the IEEE-754 32 bit single precision standard. Each modbus register is a 16 bit value.

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

In order to get the 32 bit float values using the modbus protocol they must be copied from 2 -16 bit integers. The first register [400029] is the high / most significant portion of the 32 bit float, register [400030] is the low / less significant portion. Most if not all Programmable controllers will have a copy or transfer function that will combine a 2 x integer values into a float IEEE-754 Single precision 32 bit (real). In the PLC use a “Copy / Transfer” instruction to convert the 2x16 bit integers into a 32 bit float of the current position.

\*\*\*\*\***End of Read Continuous Process Block**\*\*\*\*\*

### **Register [400031 – 400032] Axis # 1 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

### **Register [400033 – 400034] Axis # 2 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

### **Register [400035 – 400036] Axis #3 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

### **Register [400037 – 400038] Axis # 4 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

### **Register [400039 – 400040] Axis # 5 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

### **Register [400041 – 400042] Axis # 6 Steps per User Unit – Read / Write Data**

Steps per unit. The actual steps per unit in the controller is a 32 bit float which will have to be reconstructed from the High Byte and Low Byte address for the Axis. Sets the number of pulses required to move the axis a single User Unit. Units are any user defined value. MM / deg / inch. Default Combined LOW and High Bytes = 400 Steps per Revolution of Motor. All other position derived values (velocity / acceleration / deceleration) are based on the units setup with this variable. In the PLC use a “Copy” instruction to convert 2x16 bit signed integers into a 32 bit float. Make sure to take into consideration the maximum output frequency of the motion controller when setting steps per unit. We recommend not using any more than 1000 Pulse per revolution setting on your drives. At 1000 pulse per revs and a max. Coordinated output frequency of 30KHz each motor should be able to reach a velocity of 1800 RPM. Which is well above what most stepper motors will require. If you are using AC Servo drives consider a lower setting 400 pulse per revolution, which should allow for 4500 RPM. Maximum pulse rates depend on many factors.

### **Register [400043] Axis # 1 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

### **Register [400044] Axis # 2 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400045] Axis # 3 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400046] Axis # 4 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400047] Axis # 5 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400048] Axis # 6 Maximum Velocity User Units per Second – Read / Write Data**

A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. A non-coordinated move will move the axis at this velocity. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Best practice, even if only doing single axis moves. Set move type as Coordinated Linear and only update the single axis target position. That way you can modify the velocity continuously (for each move) using Register [400003]. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400049] Axis # 1 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values.

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400050] Axis # 2 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400051] Axis # 3 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400052] Axis # 4 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400053] Axis # 5 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400054] Axis # 6 Maximum Acceleration User Units per Second<sup>2</sup> – Read / Write Data**

A non-coordinated move will use this value. A Coordinated linear move will use this value in computational algorithms to calculate axis specific values. This value is stored in EEPROM memory. Do not continuously write this value with each move. It is best to set this once during a machine start-up sequence. Even if only doing single axis moves. Due to the inherent resonant frequencies created by stepper motors. This value may require tuning to set the ideal axis performance.

### **Register [400055] Axis Direction Word – Read / Write Data**

Invert the direction output bit. Set specific bits in the word to invert specific axis. This is a 16 bit integer value. Bit[0] = Axis 1, Bit[1] = Axis 2, Bit[2] =

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

Axis 3, Bit[3] = Axis 4, Bit[4] = Axis 5, Bit[5] = Axis 6. (MSBF ex. 000000000001 invert X axis direction signal). Not all drives have the ability to use this function.

### **Register [400056] Axis Set Reference Position Word – Write Data**

Set specific bits in the word to reference specific or all axis. This is a 16 bit integer value. Bit[0] = Axis 1, Bit[1] = Axis 2, Bit[2] = Axis 3, Bit[3] = Axis 4, Bit[4] = Axis 5, Bit[5] = Axis 6. Use a integer value to set individual bit in the word ON or OFF to reference a single axis or multiple axis at one time. Set all bit high to reference all axis. Or send integer values to reference single or multiple axis. (MSBF ex. 00010001 ref axis 1 and 5). When the controller receives a value in this register other then zero. It will set the positions of the axis to the value from the individual reference position setting registers [400057] to [400062]. You can send Registers [400056] to [400062] as a block of registers to simplify programming.

### **Register [400057] Axis 1 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it will use the data in this register to set the position of the axis. Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400058] Axis 2 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it will use the data in this register to set the position of the axis . Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400059] Axis 3 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it will use the data in this register to set the position of the axis. Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400060] Axis 4 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it will use the data in this register to set the position of the axis. Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400061] Axis 5 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it will use the data in this register to set the position of the axis. Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400062] Axis 6 Reference Position – Read / Write Data**

Value is a signed 16 bit Integer. Only whole numbers can be used as reference positions. Depending on the bit in the word when Register[400056] is sent it

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

will use the data in this register to set the position of the axis. Used to “Home” the axis. Can be any 16 bit signed integer value. This value is not stored in Eeprom. If power has been cycled it must be re-sent to the controller.

### **Register [400063] Set the first byte of the IP address - Read / Write Data**

Default Address is: 192.168.0.23 IP address is stored in EEPROM memory. To update the IP address send new value to this address then cycle power to controller. After power cycle controller will use new values. To reset back to default values write a 0 to address [400080]

### **Register [400064] Set the second byte of the IP address - Read / Write Data**

Default Address is: 192.168.0.23 IP address is stored in EEPROM memory. To update the IP address send new value to this address then cycle power to controller. After power cycle controller will use new values. To reset back to default values write a 0 to address [400080]

### **Register [400065] Set the Third byte of the IP address - Read / Write Data**

Default Address is: 192.168.0.23 IP address is stored in EEPROM memory. To update the IP address send new value to this address then cycle power to controller. After power cycle controller will use new values. To reset back to default values write a 0 to address [400080]

### **Register [400066] Set the Forth byte of the IP address - Read / Write Data**

Default Address is: 192.168.0.23 IP address is stored in EEPROM memory. To update the IP address send new value to this address then cycle power to controller. After power cycle controller will use new values. To reset back to default values write a 0 to address [400080]

### **Register [400067] Set the First byte of the Subnet address - Read / Write Data**

Default SUBNET is: 255.255.255.0 SUBNET Address is stored in EEPROM memory. To update the SUBNET send new values to appropriate addresses then cycle power to controller. After power cycle controller will use new values.

### **Register [400068] Set the Second byte of the Subnet address - Read / Write Data**

Default SUBNET is: 255.255.255.0 SUBNET Address is stored in EEPROM memory. To update the SUBNET send new values to appropriate addresses then cycle power to controller. After power cycle controller will use new values.

### **Register [400069] Set the Third byte of the Subnet address - Read / Write Data**

Default SUBNET is: 255.255.255.0 SUBNET Address is stored in EEPROM memory. To update the SUBNET send new values to appropriate addresses then cycle power to controller. After power cycle controller will use new values.

### **Register [400070] Set the Forth byte of the Subnet address - Read / Write Data**

Default SUBNET is: 255.255.255.0 SUBNET Address is stored in EEPROM memory. To update the SUBNET send new values to appropriate addresses then cycle power to controller. After power cycle controller will use new values.

### **Register [400071] First byte of the MAC address - Read / Write Data**

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400072] Second byte of the MAC address - Read / Write Data**

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400073] Third byte of the MAC address - Read / Write Data**

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400074] Forth byte of the MAC address - Read / Write Data**

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400075] Fifth byte of the MAC address - Read / Write Data**

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400076] Sixth byte of the MAC address - Read / Write Data**

This is a fixed value from factory. This product ships from factory with a unique identifier. This value can be changed by the end user if required. Send as a signed 16 bit integer. Value from 0-255 will be converted to HEX and used as address. This value can freely be changed and should be controlled by local network administrator. Be sure no 2 devices on your system have the same MAC address as this will cause erratic commutations.

### **Register [400077] Spare**

No function

### **Register [400078] Spare**

No function

### **Register [400079] Software Reset - Write**

## Edsgn Modbus TCP/IP Ethernet Motion Controller V1.1

Basically like resetting the power just through software commands. Setting this register value to a 1 will software reset the controller.

### **Register [400080] Fixed Register Value (31820) - Read**

This register will always have this value assigned to it. If you are trying to find out if you Modbus addressing is Zero or One based you can poll this address. Depending on the address that returns this value it will determine what your register shift is. If you are trying to confirm bit structure with your controller when you read from this register you should only ever see the value 31820. If you do not see 31820 you are either looking at the wrong register. Or you do not have the correct data type.

### **Register [400081] Factory Reset – Write**

Send a value of 1 to this register to reset your controller to factory default settings. Be advised this will reset every adjustable parameter back to the original settings. Steps per unit, MAC Address, IP Address, etc. You may lose communications with your device if network addresses have changed. If you are connected using a separate IP address you will have to re-connect using the factory default address.

### **Register [400082] Heartbeat Timer – Read**

Timer value internally updated every millisecond. Value being sent is a signed 16 bit integer. This value will rollover. Used for communications debugging or connection status logic.