ELECTROMATE

Robotic and Mechatronic Solutions

Meca500 EtherCAT Master Controller Kit Example

Contents

Step 1: Hardware Configuration 2

Step 1.1: Connecting the EtherCAT System 2

Step 2: Network Configuration 3

Step 2.1: CODESYS Setup 3

Step 2.2: Configuring the Meca500 for EtherCAT 4

Step 3: CODESYS Variable Mapping 5

Step 4: Sending Motion Commands to the Meca500 from CODESYS 8

Step 4.1: MECA Function Block 8

Step 4.2: Square Movement Sequence in CODESYS 8

Revision Table

Revision #	Description	Date	Initials
1.0	Initial Release	2022-8-18	NO

Meca500 EtherCAT Master Controller Kit Example

Step 1: Hardware Configuration

Table 1 below describes the versions and part numbers for each component used in the EtherCAT Master Controller Kit project.

Table 1: Component Versions and Part Numbers

Component	Version/Model
EXOR HMI	eSMART 107
CODESYS Internal PLC Software	3.5.16.30
JMobile Software	4.5.0
Meca500 Robotic Arm	R3 FW 9
Galil Remote I/O	RIO-57420 FW 1.0e
SCHUNK Electric Gripper	MEGP 25E

Additionally, a standard 24V power supply is used to for the EXOR HMI and the Galil RIO, and two standard Ethernet cables are used to connect the EXOR HMI to the Galil RIO and to the PC. All components required to wire and connect the Meca500 to the Galil RIO are included in the Meca500-r3 demo kit.

Step 1.1: Connecting the EtherCAT System

Connect the components described in Table 1 as shown in Figure 1 below. In the network shown below, the Eth1 port of the HMI, also called Eth8 on the device, is configured as the EtherCAT master and the Eth0 port, also called Eth7 on the device, is used to connect to the HMI and load programs from a PC. The Galil RIO and Meca500 slave devices are daisy chained together through their Ethernet input and output ports.

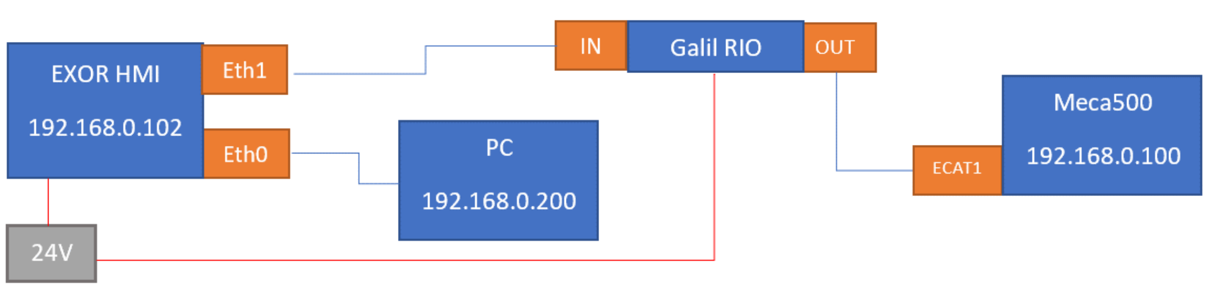

Figure 1: Suggested Configuration for EtherCAT Master Kit Connections

As shown above, ensure that each device has the same IP address network ID while the last three digits comprising the Host ID are unique to each component. Figure 2 below shows the physical layout of the connected devices.

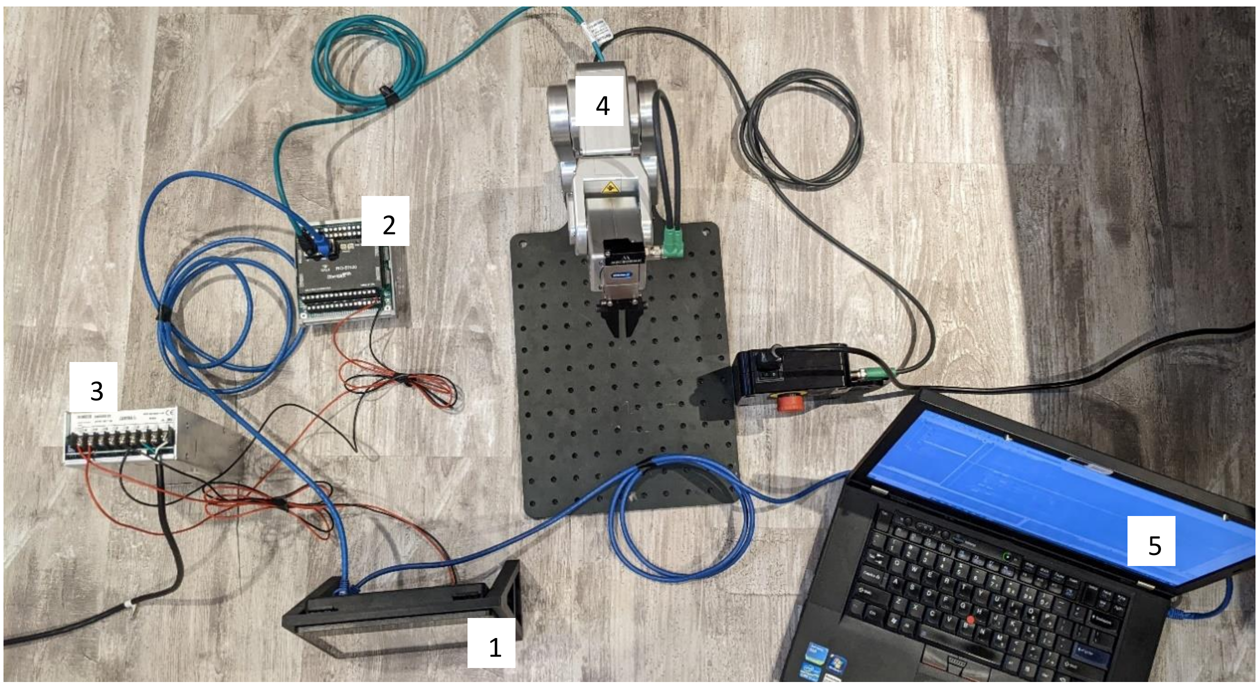

Figure 2: Photo of the Connected System for EtherCAT

Step 2: Network Configuration

Step 2.1: CODESYS Setup

The CODESYS device setup for the EtherCAT system is shown in Figure 3 below.

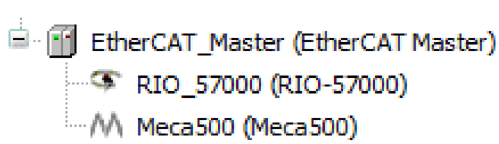

Figure 3: CODESYS device setup for EtherCAT

The Eth1 HMI port was configured as the source address of the CODESYS EtherCAT Master in the device's general settings by browsing and selecting its MAC (Figure 4).

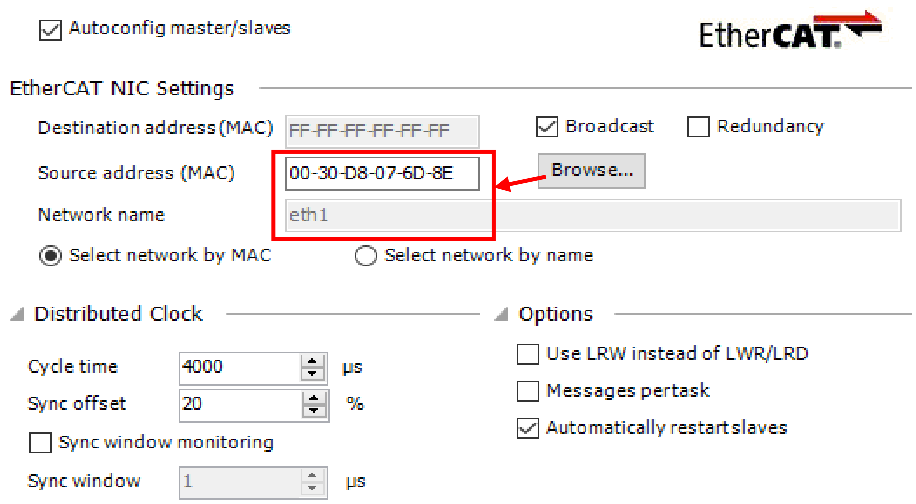

Figure 4: Configuring the EtherCAT Master Source Address

Step 2.2: Configuring the Meca500 for EtherCAT

The default communication protocol of the Meca500 is TCP/IP. To switch to the EtherCAT protocol, start by connecting to the Meca500 from a PC with the provided EtherNet cable and typing the robot's default IP address, 192.168.0.100, into a browser. In the web interface that comes up, send the SwitchToEtherCAT command after connecting to the robot (Figure 5).

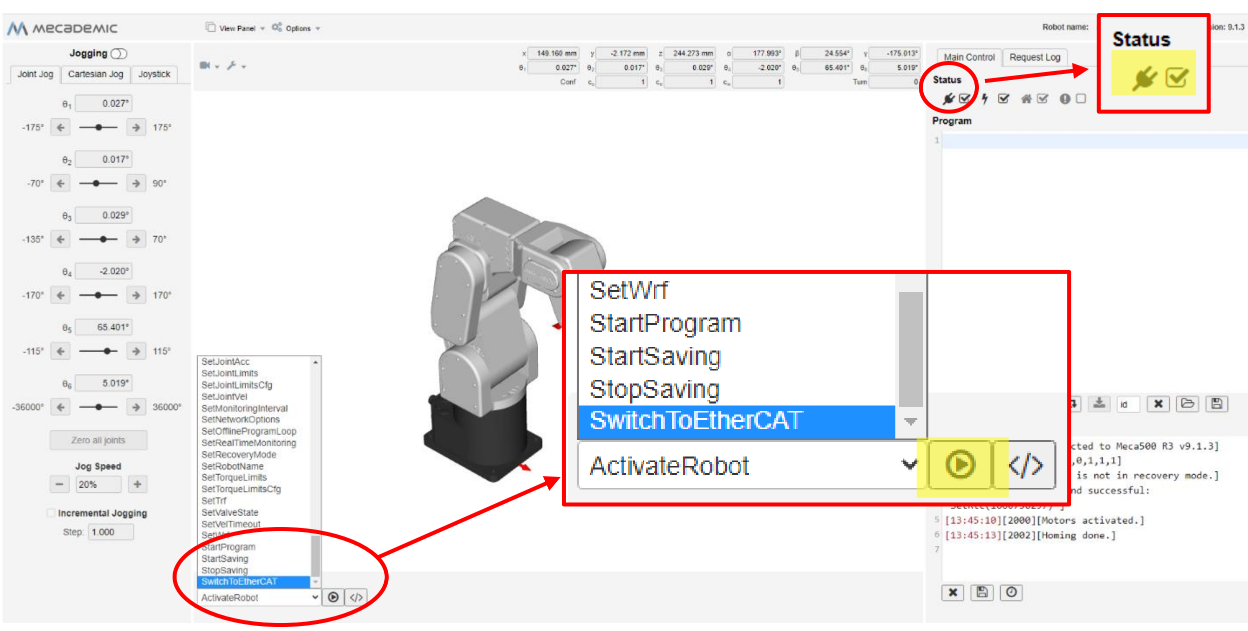

Figure 5: Switching the Meca500 to EtherCAT

Note that after switching to EtherCAT, it is no longer possible to connect to the Meca500 through the web interface. To switch back to TCP/IP mode, you will need to perform a factory reset of the Meca500 by unplugging the power supply from the AC side, and then replugging it while holding the Power button on the robot's base for about 20 seconds.

Step 3: CODESYS Variable Mapping

With the EtherCAT protocol, the MEca500 is controlled using cyclic data exchanges by detecting changes in the input and output fields addressed in the EtherCAT mapping. The IO mapping of the meca500 is obtained from the ESI file provided by Mecademic as part of the Firmware Update folder. Add the Meca500 and Galil RIO ESI files to the project from the Device repository as described in Figure 6.

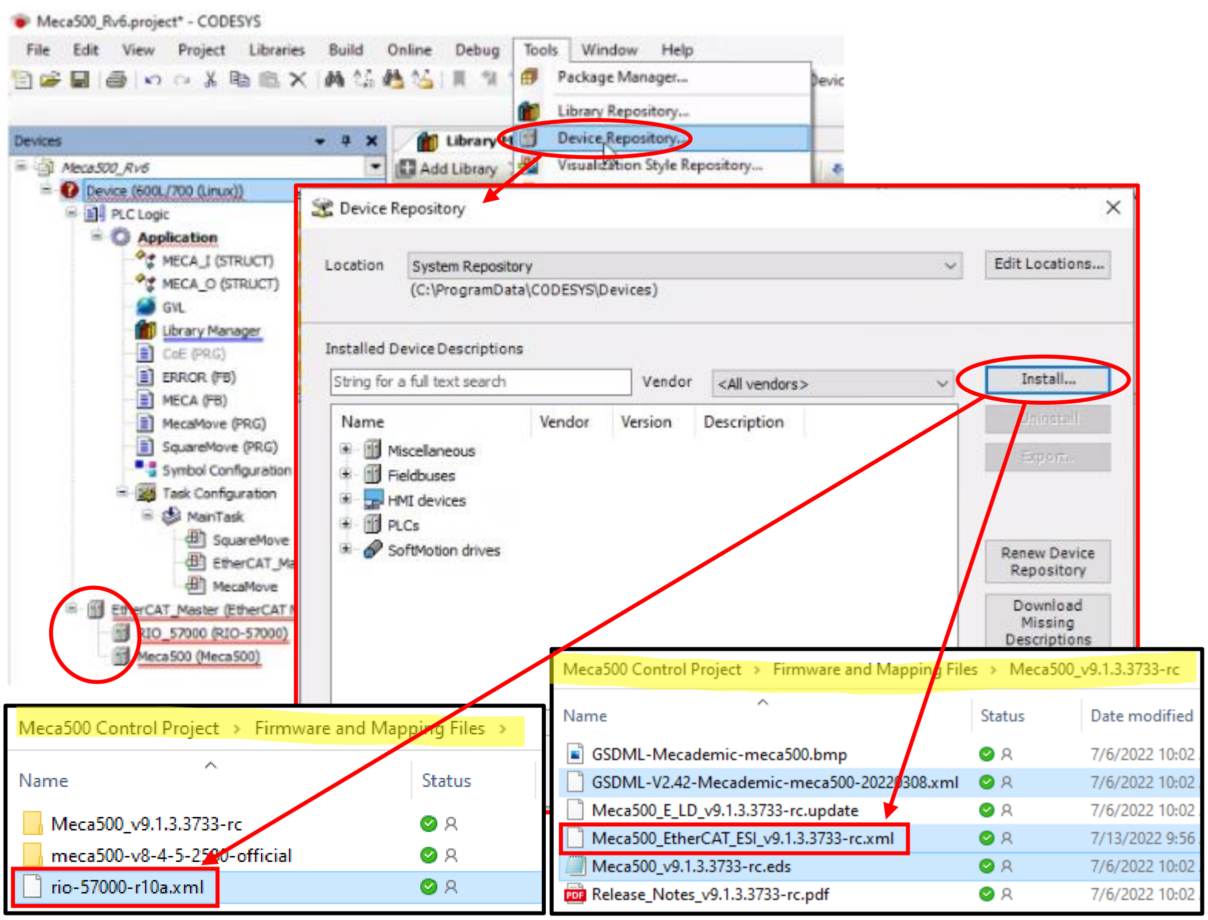

Figure 6: Adding Device Descriptions to the Device Repository

The Meca500 IO fields are described in Figure 7 below. In this example project, variables have been mapped to custom made data structures (Robot_O and Robot_I) by specifying the corresponding structure in the Variable field. Since the IO fields are automatically mapped to available addresses, the PLC code references them through structures to avoid having to edit the code every time the IO mapping is changed when adding a new device or mapping file.

Variable	Map...	Channel	Address	Type	Description
Application.GVL.Robot_O.Deactivate		Deactivate	~~%QX120.0~~	BIT	Deactivates robot when true
Application.GVL.Robot_O.Activate		Activate	~~%QX120.1~~	BIT	Activates robot when true
Application.GVL.Robot_O.Home		Home	~~%QX120.2~~	BIT	Sends command to home robot when true
Application.GVL.Robot_O.ResetError		Reset Error	~~%QX120.3~~	BIT	Clears the error code
Application.GVL.Robot_O.SimMode		Sim Mode	~~%QX120.4~~	BIT	Sim Mode
		Recovery Mode	%QX120.5	BIT	Recovery Mode
Application.GVL.Robot_O.MoveID		Move ID	~~%QW62~~	UINT	Must change for each cyclic motion command
Application.GVL.Robot_O.SetPoint		SetPoint	~~%QX126.0~~	BIT	Rising edge triggers next command in motion queue
Application.GVL.Robot_O.Pause		Pause	~~%QX126.1~~	BIT	Robot is paused when true, activated when false
Application.GVL.Robot_O.ClearMove		Clear Move	~~%QX126.2~~	BIT	Cleasrs pending commands in the motion queue
Application.GVL.Robot_O.ResetPStop		Reset PStop	~~%QX126.3~~	BIT	Resets emergency stop when true
Application.GVL.Robot_O.MoveCommand		Move Command	~~%QD32~~	UDINT	ID of the motion command being sent
Application.GVL.Robot_O.SubIndex1		SubIndex 001	~~%QD33~~	REAL	First argument of the motion command, 'x'
Application.GVL.Robot_O.SubIndex2		SubIndex 002	~~%QD34~~	REAL	Second argument of the motion command, 'y'
Application.GVL.Robot_O.SubIndex3		SubIndex 003	~~%QD35~~	REAL	Third argument of the motion command, 'z'
Application.GVL.Robot_O.SubIndex4		SubIndex 004	~~%QD36~~	REAL	Fourth argument of the motion command, 'rx'
Application.GVL.Robot_O.SubIndex5		SubIndex 005	~~%QD37~~	REAL	Fifth argument of the motion command, 'ry'
Application.GVL.Robot_O.SubIndex6		SubIndex 006	~~%QD38~~	REAL	Sixth argument of the motion command, 'rz'
		Host Time	%QD39	UDINT	Time elapsed
		BrakesControlAll...	%QX160.0	BIT	Allows brake control with BrakesEngaged when true
		BrakesEngaged	%QX160.1	BIT	Brakes Engaged when true
Application.GVL.Robot_O.StatusGripper		Dynamic Type	~~%QD41~~	UDINT	Return type of the first dynamic input register
Application.GVL.Robot_O.Dynamic1		Dynamic Type	~~%QD42~~	UDINT	Return type of the second dynamic input register
		Dynamic Type	%QD43	UDINT	Return type of the third dynamic input register
		Dynamic Type	%QD44	UDINT	Return type of the fourth dynamic input register
Application.GVL.Robot_I.Busy		Busy	~~%IX120.0~~	BIT	True when robot is activating or homing
Application.GVL.Robot_I.Activated		Activated	~~%IX120.1~~	BIT	True when Activated
Application.GVL.Robot_I.Homed		Homed	~~%IX120.2~~	BIT	True when Homed
		SimActivated	%IX120.3	BIT	true when simulation is activated
		BrakesEngaged	%IX120.4	BIT	BrakesEngaged
		RecoveryMode	%IX120.5	BIT	RecoveryMode
Application.GVL.Robot_I.Error		Error	~~%IW61~~	UINT	Current error code
		CheckPoint	%ID31	UDINT	CheckPoint
Application.GVL.Robot_I.MoveID		Move ID	~~%IW64~~	UINT	ID of the current motion command
		FIFO Space	%IW65	UINT	Num of commands that can be added to motion q...
Application.GVL.Robot_I.Paused		Paused	~~%IX132.0~~	BIT	True when Paused
Application.GVL.Robot_I.EOB		EOB	~~%IX132.1~~	BIT	Robot stopped moving and motion queue is empty

Application.GVL.Robot_I.EOM		EOM	~~%IX132.2~~	BIT	True if Robot has stopped moving
		FIFO Cleared	%IX132.3	BIT	True if motion queue is cleared
Application.GVL.Robot_I.PStop		PStop	~~%IX132.4~~	BIT	True if robot is in protective stop
		Excessive torque	%IX132.5	BIT	Excessive torque
		Offline Program ID	%IW67	UINT	ID of offline program currently running
Application.GVL.Robot_I.Joint1		SubIndex 001	~~%ID34~~	REAL	Joint 1 Angle
Application.GVL.Robot_I.Joint2		SubIndex 002	~~%ID35~~	REAL	Joint 2 Angle
Application.GVL.Robot_I.Joint3		SubIndex 003	~~%ID36~~	REAL	Joint 3 Angle
Application.GVL.Robot_I.Joint4		SubIndex 004	~~%ID37~~	REAL	Joint 4 Angle
Application.GVL.Robot_I.Joint5		SubIndex 005	~~%ID38~~	REAL	Joint 5 Angle
Application.GVL.Robot_I.Joint6		SubIndex 006	~~%ID39~~	REAL	Joint 6 Angle
Application.GVL.Robot_I.xPose		SubIndex 001	~~%ID40~~	REAL	End effector pose x
Application.GVL.Robot_I.yPose		SubIndex 002	~~%ID41~~	REAL	End effector pose y
Application.GVL.Robot_I.zPose		SubIndex 003	~~%ID42~~	REAL	End effector pose z
Application.GVL.Robot_I.rxPose		SubIndex 004	~~%ID43~~	REAL	End effector pose rx
Application.GVL.Robot_I.ryPose		SubIndex 005	~~%ID44~~	REAL	End effector pose ry
Application.GVL.Robot_I.rzPose		SubIndex 006	~~%ID45~~	REAL	End effector pose rz
Application.GVL.Robot_I.Config_Shoul...		Shoulder	~~%IB184~~	SINT	Shoulder Configuration (1 or -1)
Application.GVL.Robot_I.Config_Elbow		Elbow	~~%IB185~~	SINT	Elbow Configuration (1 or -1)
Application.GVL.Robot_I.Config_Wrist		Wrist	~~%IB186~~	SINT	Wrist Configuration (1 or -1)
		Turn	%IB187	SINT	Turn Configuration (1 or -1)
		SubIndex 001	%ID47	REAL	SubIndex 001
		SubIndex 002	%ID48	REAL	SubIndex 002
		SubIndex 003	%ID49	REAL	SubIndex 003
		SubIndex 004	%ID50	REAL	SubIndex 004
		SubIndex 005	%ID51	REAL	SubIndex 005
		SubIndex 006	%ID52	REAL	SubIndex 006
		Dynamic Type	%ID53	UDINT	Dynamic register with ID 53 for Gripper Status
Application.GVL.Robot_I.GripperHolding		Value 0	~~%ID54~~	REAL	True if Gripper is holding an object
Application.GVL.Robot_I.GripperLimit		Value 1	~~%ID55~~	REAL	TRue if gripper opening or closing limit is reached
Application.GVL.Robot_I.GripperClosed		Value 2	~~%ID56~~	REAL	True if gripper is closed
Application.GVL.Robot_I.GripperOpen		Value 3	~~%ID57~~	REAL	True if Gripper is Open
Application.GVL.Robot_I.GripperForce		Value 4	~~%ID58~~	REAL	Current force in the gripper
Application.GVL.Robot_I.fingersOpen		Value 5	~~%ID59~~	REAL	True while fingers are opening
		Dynamic Type	%ID60	UDINT	Current dynamic type ID for register 2 (default 0)
Application.GVL.Robot_I.Dynamic1		Value 0	~~%ID61~~	REAL	Value 0 of Dynamic Data object
Application.GVL.Robot_I.Dynamic2		Value 1	~~%ID62~~	REAL	Value 1 of Dynamic Data object
Application.GVL.Robot_I.Dynamic3		Value 2	~~%ID63~~	REAL	Value 2 of Dynamic Data object
Application.GVL.Robot_I.Dynamic4		Value 3	~~%ID64~~	REAL	Value 3 of Dynamic Data object
Application.GVL.Robot_I.Dynamic5		Value 4	~~%ID65~~	REAL	Value 4 of Dynamic Data object
Application.GVL.Robot_I.Dynamic6		Value 5	~~%ID66~~	REAL	Value 5 of Dynamic Data object

Reset Mapping | Always updatevariables | Enabled 1 (use bus c

Figure 7: Meca500 EtherCAT IO Fields

Step 4: Sending Motion Commands to the Meca500 from CODESYS

Step 4.1: MECA Function Block

To successfully send motion commands to the Meca500, a specific process of changing IO fields should be followed. Since this process is the same for all kinds of motion commands, a function block was created to handle all types of motion commands. A state machine is used to allow the IO fields to be sent and received at the right time. The state diagram is described in Figure 8 below.

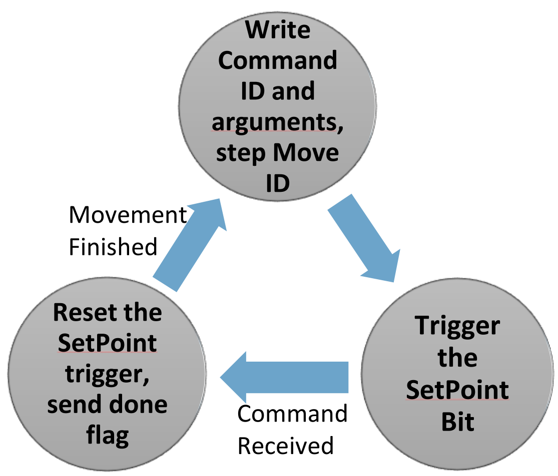

Figure 8: State Diagram for sending Motion Commands to the Meca500

The function block described above takes in 7 arguments: The ID of the motion command and its 6 arguments. After writing the desired command and arguments to the cyclic data fields and increasing the Move ID, a rising edge trigger is used to flash the SetPoint bit. For physical moves, the EOB changes to false when a move is in progress. For gripper commands, the GripperLimit field switches to false, and for configurations, several milliseconds passes. When the function block gets indication that the command has been successfully started by the robot, it outputs a done flag. This flag is used to cycle though different function block calls when programming movement sequences.

Step 4.2: Square Movement Sequence in CODESYS

This example is a simple movement sequence where the Meca500 moves in a square pattern and picks up an object at a pick point if an input signal is received from the RIO, triggering an output while the robot is holding the object. The CODESYS state machine for this movement sequence is described in Figure 9 below.

```
CASE nStep OF

    1:  //Bottom right corner
        MecaCMD(command:=MOVE_POSE, x:=249.484939575, y:=63.039520264, z:=76.484741211, rx:=10.364617348, ry:=87.368103027, rz:=-10.525827408);
        IF MecaCMD.DN=nStep THEN
            nStep:=nStep+1;
        END_IF

    2:  //Top Right Corner
        MecaCMD (command:=MOVELIN_WRF, x:=0,y:=0, z:=125, rx:=0,ry:=0, rz:=0);
        IF MecaCMD.DN=nStep THEN
            nStep:=nStep+1;
        END_IF

    3: //Top left corner
        MecaCMD(command:=MOVELIN_WRF, x:=0, y:=-125, z:=0, rx:=0,ry:=0, rz:=0);
        IF mecaCMD.DN=nStep THEN
            nStep:= nStep+1;
        END_IF

    4: //Bottom left corner
        MecaCMD(command:=MOVELIN_WRF, x:=0, y:=0, z:=-125, rx:=0, ry:=0, rz:=0);
        IF mecaCMD.DN =nStep THEN
            (*PICKUP/DROPOFF STEPS 5-7 CAN BE SET TO ACTIVATE ON RIO INPUT BY ACTIVATING THE FOLLOWING CODE:*)
            IF gvl.RIO_in THEN
                nStep:= nStep+1;
            ELSE
                //loop
                nStep:=1;
            END_IF

        END_IF
```

Figure 9: Simple CODESYS Square Move sequence using the Function Block

In the sample code above, MecaCMD is an instance of the MECA function block described in Figure 8. To ensure that the program is executed in the right order and to simplify troubleshooting, each new motion command is in a separate state. The done flag from the function block (DN) is used to move through the states as each motion command is completed.