TechNote

RS-485 Modbus Using the Model 5300 with a Mitsubishi MR-JE-_A Drive

45

Control Technology Corporation, Hopkinton, MA • 800.282.5008 • www.ctc-control.com

TechNote 45 describes how to connect a CTC Model 5300 controller to a Mitsubishi MR-JE-_A Drive using RS-485. The information in this document applies to the Model 5300 BC5311-01B.

Note that the BC5311-01A only offers RS-232 communications and would require an RS-232 to RS-485 convertor.

Overview

This document shows you how to set up communications between a CTC 5300 Controller and a Mitsubishi MR-JE-_A drive over RS-485. The BC5311-01B has an RS-485 port that can be accessed via communications Port 3.

If you do not have a basic understanding of QuickBuilder. please refer to the following manuals before attempting to connect the Model 5300 and the Mitsubishi MR-JE-_A drive.

Quick Builder QuickStart Guide
http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530030.pdf

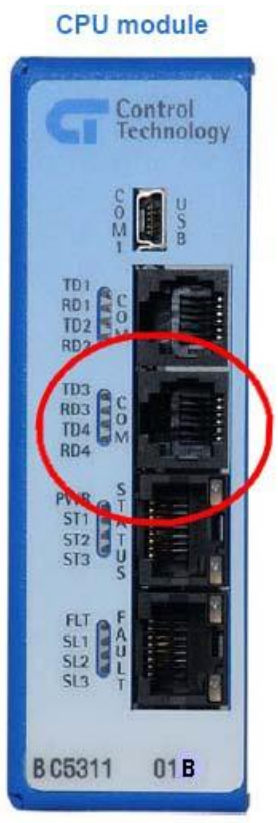
5300 Quick Register Guide
http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530006.pdf

5300 Enhancements Guide
http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530001.pdf

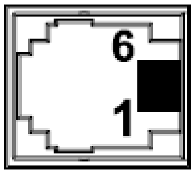
Sample QuickBuilder Code for a program that sets up communications between a Model 5300 Controller and a Mitsubishi MR-JE-_A drive over RS-485 can be found on CTC's Sample Code page.

Wiring the RS-485 Ports

The RS-485 port on the BC5311-01B is accessed via communication Port 3 as shown below:

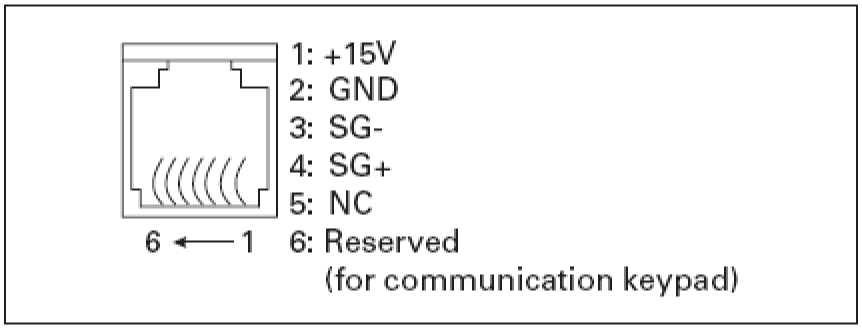


COM3 and COM4 RS232/RS485 pinouts



Pin #	Signal
1	TxD COM4
2	TXD COM3/A (+RS485)
3	Common
4	Common
5	RXD COM3/B(-RS485)
6	RXD COM4

The RS-485 port on the Mitsubishi MR- is accessed with a RJ-12 jack at the top of the drive near the Input power. The pin configuration is shown below:



1: +15V
2: GND
3: SG-
4: SG+
5: NC
6: Reserved
(for communication keypad)

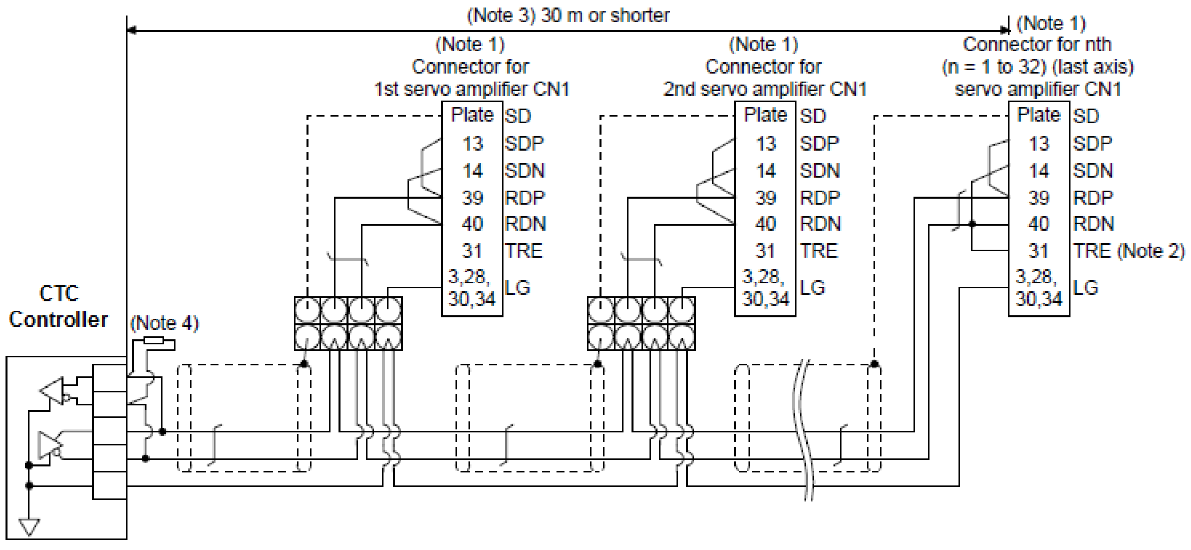
Since Pins 1 and 6 are not used on either side of the cable, a 4-pin RJ-11 connector can be used.

The cable pin out between the CTC BC5311-01B and the Mitsubishi MR-JE-_A is shown below, using 4-pin RJ-11 connectors (male connector configuration pictured below).

1.4.2 Cable connection diagram

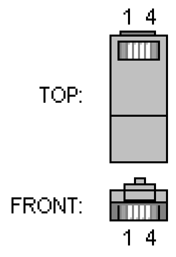
(1) Half duplex wiring

Wire the system as shown below.



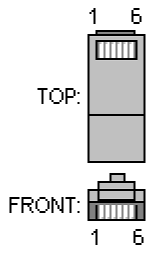
(Note 3) 30 m or shorter
(Note 1) Connector for 1st servo amplifier CN1
(Note 1) Connector for 2nd servo amplifier CN1
(Note 1) Connector for nth (n = 1 to 32) (last axis) servo amplifier CN1
Plate SD
13 SDP
14 SDN
39 RDP
40 RDN
31 TRE
3,28, 30,34 LG
TRE (Note 2)
CTC Controller
(Note 4)

Note
1. Connector set MR-J3CN1 (3M or equivalent)
   Connector: 10150-3000PE
   Shell kit: 10350-52F0-008
2. Connect TRE and RDN on the last axis.
3. The total extension length should be 30 m or shorter in a low-noise environment.
4. When a Modbus-compatible controller does not have a termination resistor, terminate the wire ends with a register of 150 Ω.



	CTC	Mitsubishi
Pin	1- B/485-	40- RDN
	4- A/485+	39- RDP
	2- Common	3, 28, 30, 34- LG

Using 6-pin RJ-12 connectors (male connector configuration shown below):



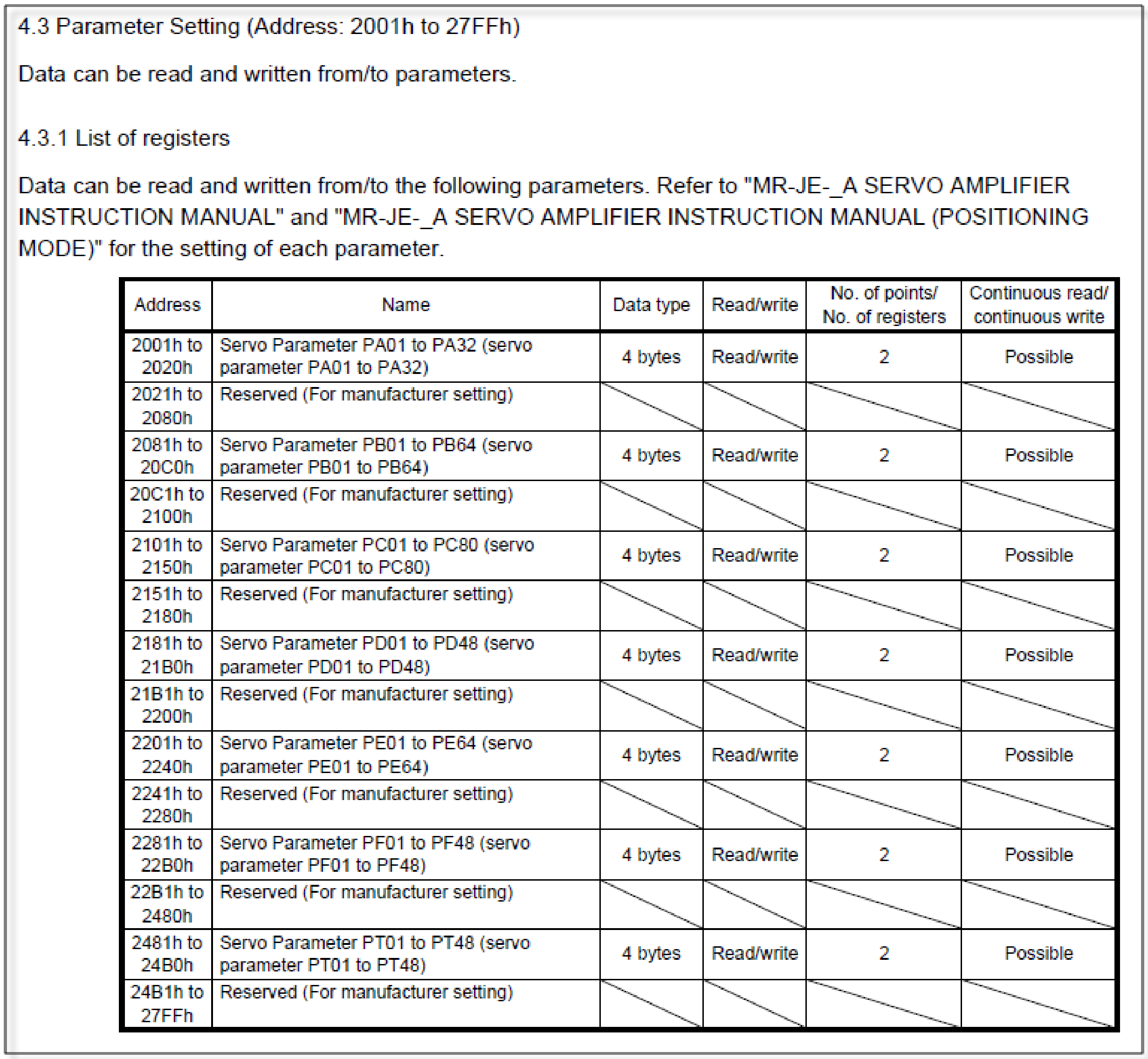
	CTC	Mitsubishi
Pin	2- B/485-	40- RDN
	5- A/485+	39- RDP
	3- Common	3, 28, 30, 34- LG

Note that you may find it necessary to add termination resister(s) depending on your environment and the length of your cable. Refer to RS-485 network specifications for more information on this.

Addressing

Select the start address of the Parameters you want to get from the Mitsubishi Drive and then select how many consecutive Parameters you would like.

For example, you may want to access all seven (7) parameters from the Basic Grouping:



4.3 Parameter Setting (Address: 2001h to 27FFh)

Data can be read and written from/to parameters.

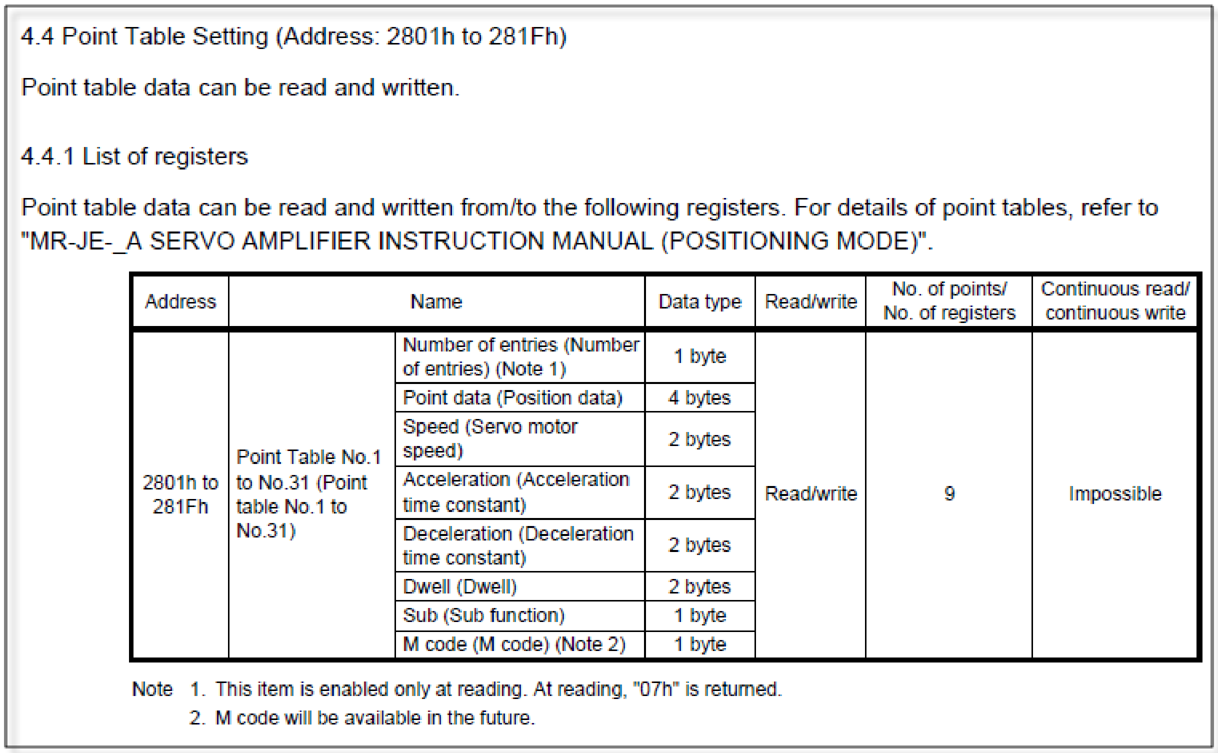
4.3.1 List of registers

Data can be read and written from/to the following parameters. Refer to "MR-JE-_A SERVO AMPLIFIER INSTRUCTION MANUAL" and "MR-JE-_A SERVO AMPLIFIER INSTRUCTION MANUAL (POSITIONING MODE)" for the setting of each parameter.

Address	Name	Data type	Read/write	No. of points/ No. of registers	Continuous read/ continuous write
2001h to 2020h	Servo Parameter PA01 to PA32 (servo parameter PA01 to PA32)	4 bytes	Read/write	2	Possible
2021h to 2080h	Reserved (For manufacturer setting)				
2081h to 20C0h	Servo Parameter PB01 to PB64 (servo parameter PB01 to PB64)	4 bytes	Read/write	2	Possible
20C1h to 2100h	Reserved (For manufacturer setting)				
2101h to 2150h	Servo Parameter PC01 to PC80 (servo parameter PC01 to PC80)	4 bytes	Read/write	2	Possible
2151h to 2180h	Reserved (For manufacturer setting)				
2181h to 21B0h	Servo Parameter PD01 to PD48 (servo parameter PD01 to PD48)	4 bytes	Read/write	2	Possible
21B1h to 2200h	Reserved (For manufacturer setting)				
2201h to 2240h	Servo Parameter PE01 to PE64 (servo parameter PE01 to PE64)	4 bytes	Read/write	2	Possible
2241h to 2280h	Reserved (For manufacturer setting)				
2281h to 22B0h	Servo Parameter PF01 to PF48 (servo parameter PF01 to PF48)	4 bytes	Read/write	2	Possible
22B1h to 2480h	Reserved (For manufacturer setting)				
2481h to 24B0h	Servo Parameter PT01 to PT48 (servo parameter PT01 to PT48)	4 bytes	Read/write	2	Possible
24B1h to 27FFh	Reserved (For manufacturer setting)				

In this case the start address of 2001H from the drive would be considered one (1) to CTC since CTC's addressing always starts at one (1). Also take into account the number of bytes that make up the data type. Four (4) bytes will require two (2) CTC registers. In this case, the number of parameters or sequential registers to get would be 14.

If you wanted to get nine (9) of the Drive Control Parameters (shown below), you would use a start address of 10242. 2801H is a hexadecimal number. This needs to be converted to decimal, which is address 10241. The final value of 10242 is due to the offset of 1 because the drive addressing starts at zero and the Model 5300 addressing starts at 1. When determining how many CTC registers are required, be sure to take into account the number of bytes that make up the various data types. In this example, nine (9) sequential registers are required.



4.4 Point Table Setting (Address: 2801h to 281Fh)

Point table data can be read and written.

4.4.1 List of registers

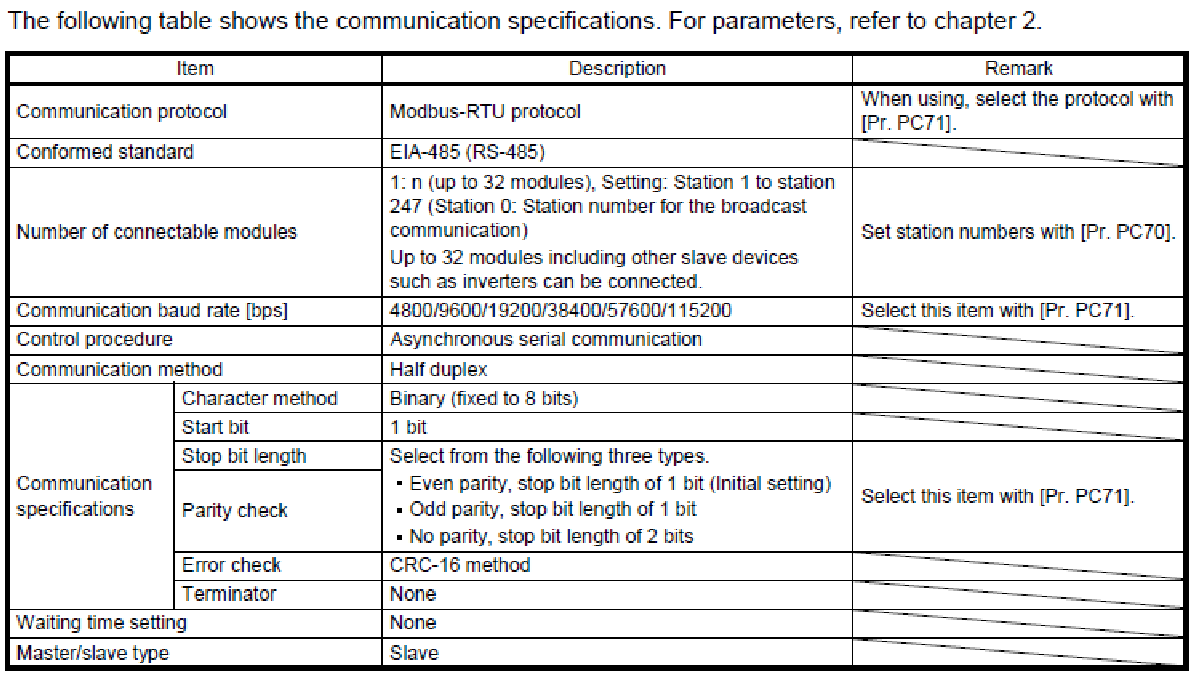
Point table data can be read and written from/to the following registers. For details of point tables, refer to "MR-JE-_A SERVO AMPLIFIER INSTRUCTION MANUAL (POSITIONING MODE)".

Address	Name		Data type	Read/write	No. of points/ No. of registers	Continuous read/ continuous write
2801h to 281Fh	Point Table No.1 to No.31 (Point table No.1 to No.31)	Number of entries (Number of entries) (Note 1)	1 byte	Read/write	9	Impossible
		Point data (Position data)	4 bytes			
		Speed (Servo motor speed)	2 bytes			
		Acceleration (Acceleration time constant)	2 bytes			
		Deceleration (Deceleration time constant)	2 bytes			
		Dwell (Dwell)	2 bytes			
		Sub (Sub function)	1 byte			
		M code (M code) (Note 2)	1 byte			

Note 1. This item is enabled only at reading. At reading, "07h" is returned.
2. M code will be available in the future.

Mitsubishi MR-JE-_A Drive Settings

When you are setting the communication parameters, make sure the RS-485 settings are compatible between the two devices.



The following table shows the communication specifications. For parameters, refer to chapter 2.

Item		Description	Remark
Communication protocol		Modbus-RTU protocol	When using, select the protocol with [Pr. PC71].
Conformed standard		EIA-485 (RS-485)	
Number of connectable modules		1: n (up to 32 modules), Setting: Station 1 to station 247 (Station 0: Station number for the broadcast communication) Up to 32 modules including other slave devices such as inverters can be connected.	Set station numbers with [Pr. PC70].
Communication baud rate [bps]		4800/9600/19200/38400/57600/115200	Select this item with [Pr. PC71].
Control procedure		Asynchronous serial communication	
Communication method		Half duplex	
Communication specifications	Character method	Binary (fixed to 8 bits)	
	Start bit	1 bit	
	Stop bit length	Select from the following three types. ▪ Even parity, stop bit length of 1 bit (Initial setting) ▪ Odd parity, stop bit length of 1 bit ▪ No parity, stop bit length of 2 bits	Select this item with [Pr. PC71].
	Parity check		
	Error check	CRC-16 method	
	Terminator	None	
Waiting time setting		None	
Master/slave type		Slave	

The basic communication parameters for the MR-JE-_A are listed above.

In this example we set them as follows:

Parameter	Description	Setting
PR. PC71	Comm. Protocol	Modbus RTU mode 8 data bits, no parity, 2 stop bits
PR. PC71	Modbus Address	(each device in a multi-drop system must be unique)
PR. PC71	Transmission Speed	19200 Baud

We used default for the remaining settings. Refer to the MR-JE-_A manual for more information on these settings.

CTC 5300 Settings

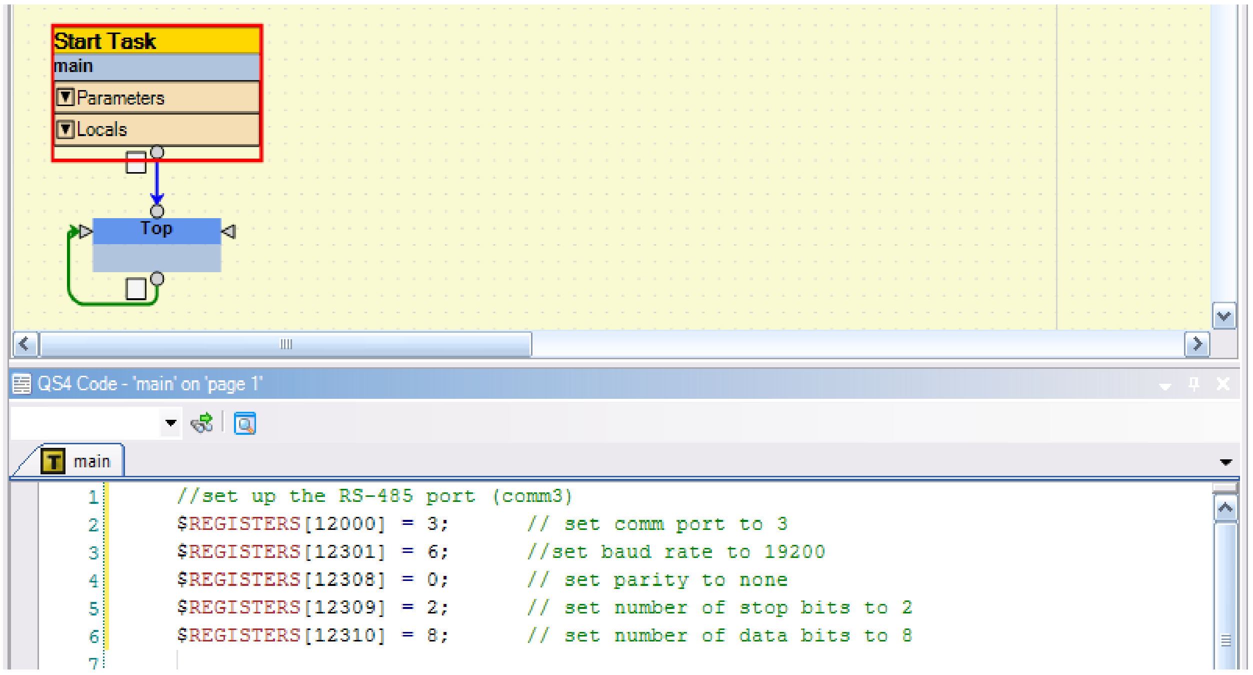
RS-485 Communications Settings

On the CTC 5300, first set up the RS-485 parameters for Port 3. The Registers used to set up communication for the serial ports are as follows:

Communications Registers for Serial Settings

REG 12000: Com port selection, 1=COM1, 2=COM2, 3 thru 7 = TCP raw socket
REG 12000: Selected port status 0=not busy, 1=busy
REG 12301: Baud Rate (2=1.2K, 3=2.4K, 4=4.8K, 5=9.6K, 6=19.2K, 7=38.4k)
REG 12308: Serial Port Parity, 0=none (default), 1=odd, 2=even
REG 12309: Serial Port Stop Bits, 1 (default), 2
REG 12310: Serial Data Bits, 7 or 8 (default)

Set up the registers in the beginning of the start task of your program, as shown below:



```
//set up the RS-485 port (comm3)
$REGISTERS[12000] = 3;      // set comm port to 3
$REGISTERS[12301] = 6;      //set baud rate to 19200
$REGISTERS[12308] = 0;      // set parity to none
$REGISTERS[12309] = 2;      // set number of stop bits to 2
$REGISTERS[12310] = 8;      // set number of data bits to 8
```

5300 Modbus Master Settings

This section discusses how to set up the 5300 as a Modbus Master. More in-depth information on setting up a Modbus master can be found here: http://www.ctc-control.com/customer/techinfo/docs/5200_951/951-520002.pdf

The 5300 controller can run numerous Modbus TCP Master connections and a single RTU/ASCII Serial connection at the same time, to differing devices, limited only by the performance desired.

Modbus Register Overview

REG 21000-21299: Modbus parameters are organized in groups of 10:

21xx0-21xx3: IP address of target controller
21xx4: Start register in target (first register to read in target controller)
21xx5: Number of sequential registers to read, 1 to 100
21xx6: Poll time (time between reads) in ms. 50mS is min, 0 reads once
21xx7: Status, 0=offline, 1=ok, -1=fail, -2=busy connecting, -3=busy reading
-4=busy writing, -5=timed out, -10=aborted
21xx8: Index offset register, pointer to stack parameters, see address below
1000 – Peer Request Time-Out
1001 – Peer Request Failed
1002 – Peer Request Retry Counter
1003 – Protocol Index Register
1004 – TCO Client Support Register
1005 – Modbus Master Unit ID
1006 – Modbus Master Exception
1007 – Register Remapping Start (23000 – 24999)
1008 – Modbus Master MAX Retries
1009 – Modbus Master Retry Counter
1010 – Modbus Master Timeout
1011 – Modbus Master Block Size
1999 – Peer Request Initiate
2000 – 2099 – Peer Request Write Block
21xx9: Data for the 21xx8 index pointer. Point w/ index → store data here

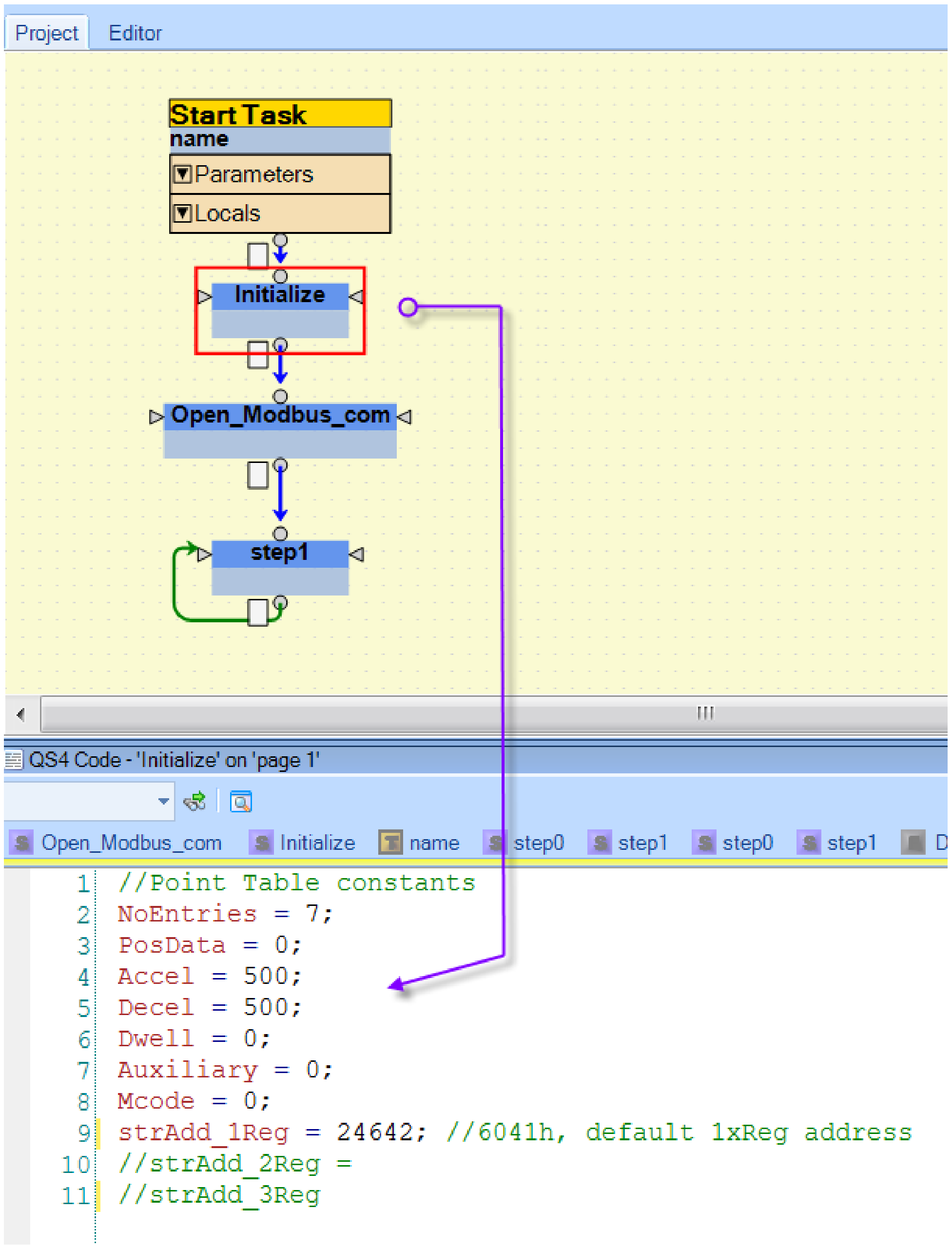
Since we are looking using a serial RS-485 connection, we can set the 21xx0 registers to any value.

The Modbus Master Unit ID Offset Register tells the Modbus Master which device ID to connect to and communicate with.

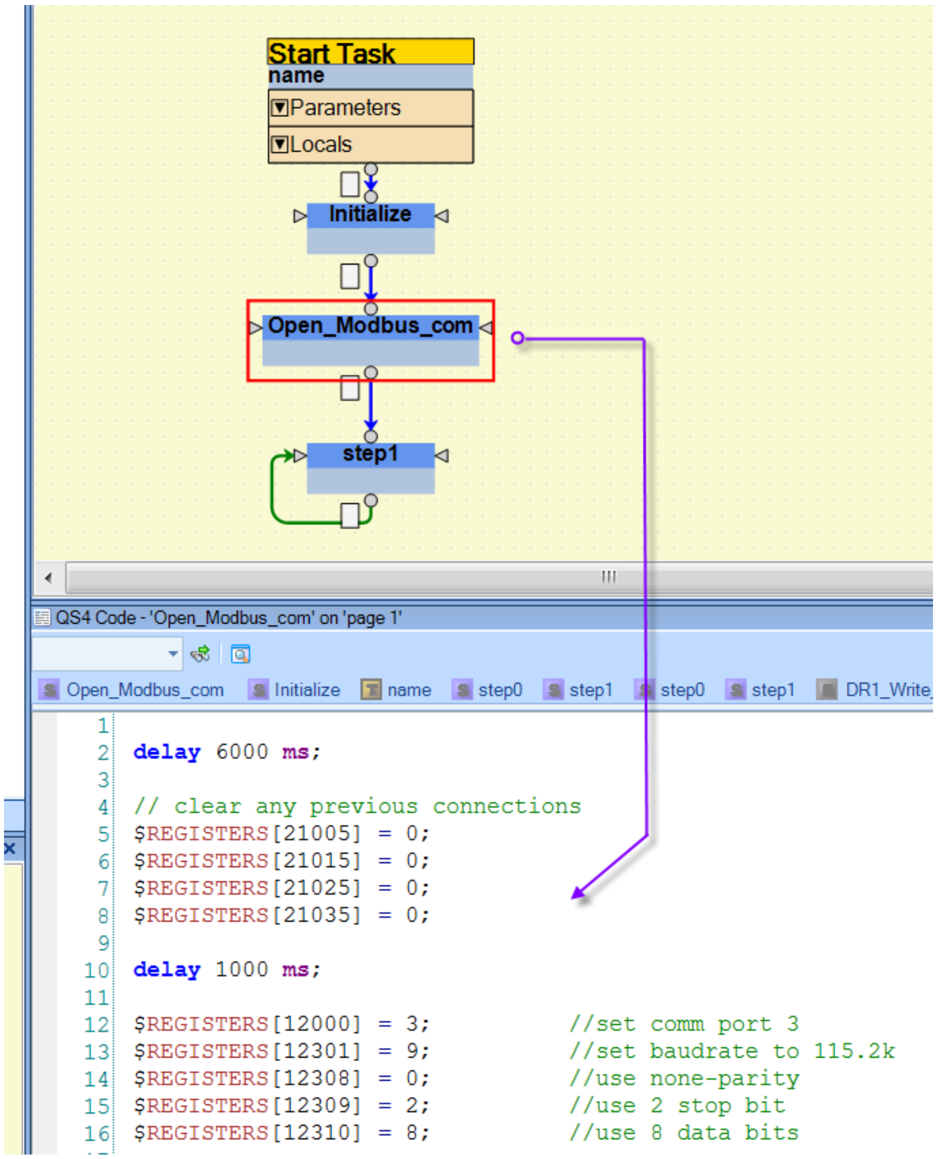
The Register Remapping Start Offset Register assigns the registers within the 5300 to use to access the Mitsubishi drive parameters.

The following example uses QuickBuilder to set up the Modbus connection.

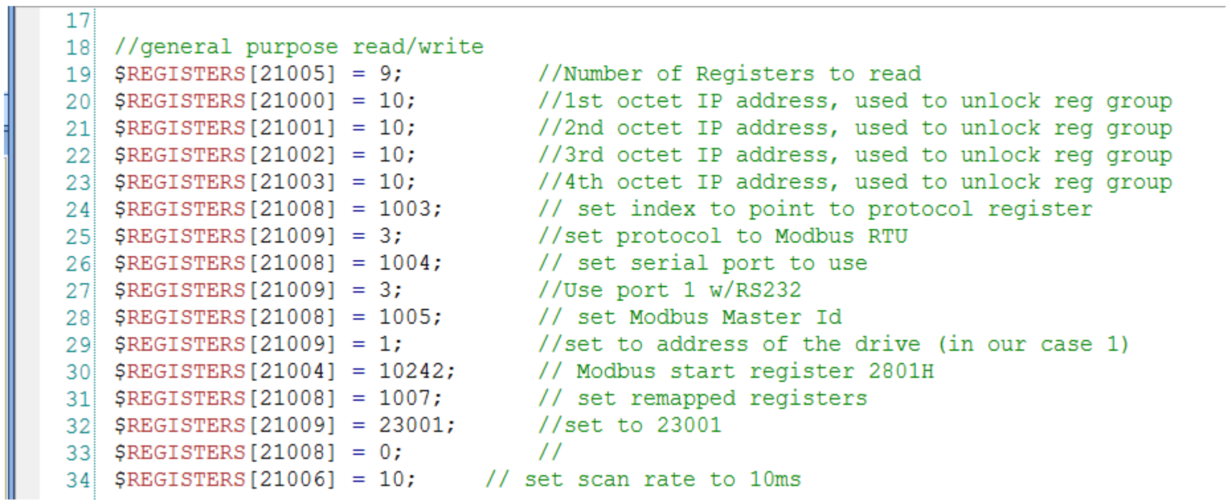
Note that we add the Modbus Register Settings after the Communication Port Settings and move profile initialization.



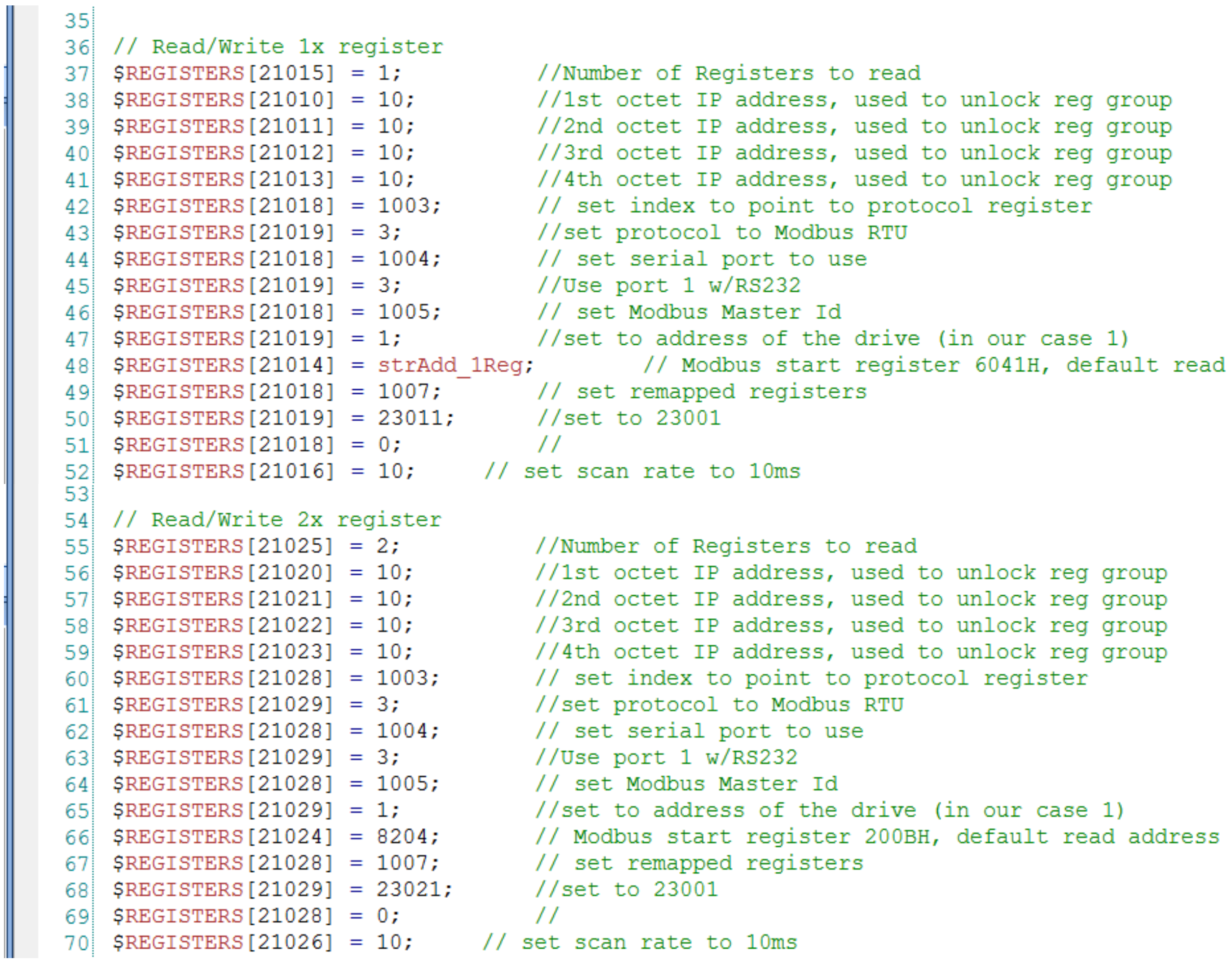
```
//Point Table constants
NoEntries = 7;
PosData = 0;
Accel = 500;
Decel = 500;
Dwell = 0;
Auxiliary = 0;
Mcode = 0;
strAdd_1Reg = 24642; //6041h, default 1xReg address
//strAdd_2Reg =
//strAdd_3Reg
```



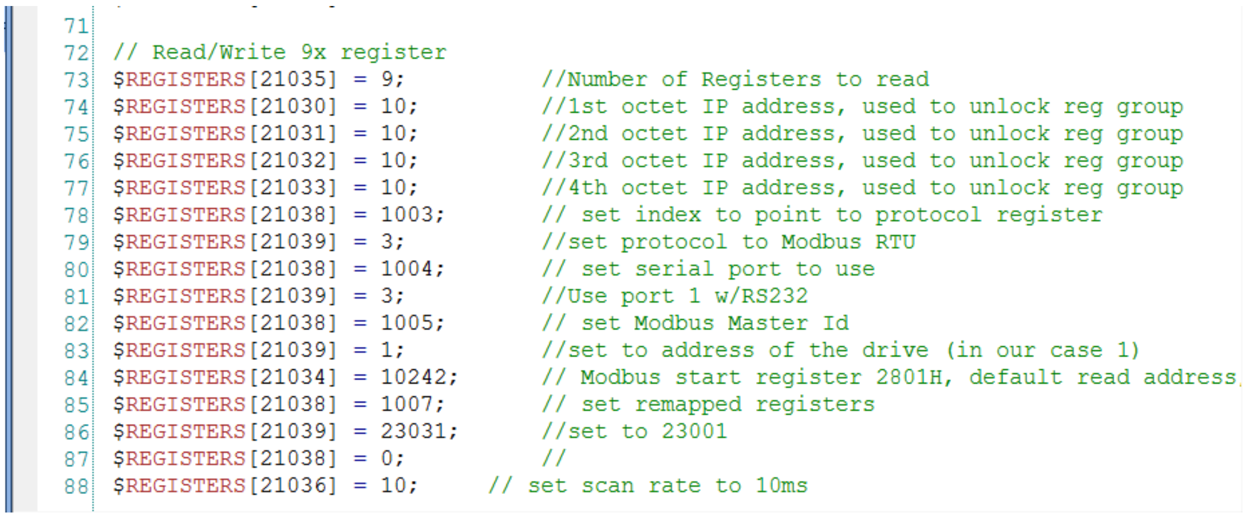
```

delay 6000 ms;

// clear any previous connections
$REGISTERS[21005] = 0;
$REGISTERS[21015] = 0;
$REGISTERS[21025] = 0;
$REGISTERS[21035] = 0;

delay 1000 ms;

$REGISTERS[12000] = 3;          //set comm port 3
$REGISTERS[12301] = 9;          //set baudrate to 115.2k
$REGISTERS[12308] = 0;          //use none-parity
$REGISTERS[12309] = 2;          //use 2 stop bit
$REGISTERS[12310] = 8;          //use 8 data bits
```



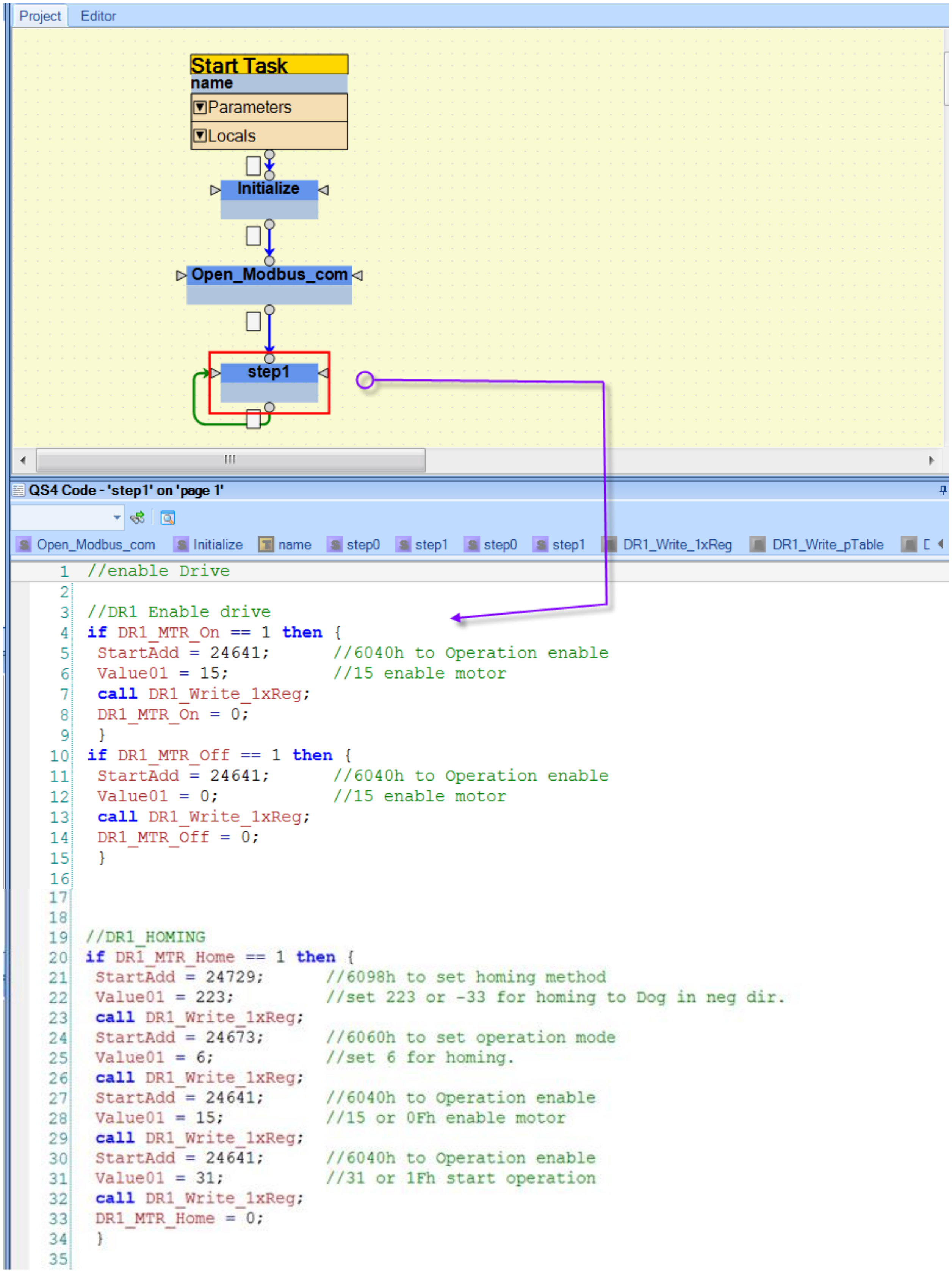
```

//general purpose read/write
$REGISTERS[21005] = 9;          //Number of Registers to read
$REGISTERS[21000] = 10;         //1st octet IP address, used to unlock reg group
$REGISTERS[21001] = 10;         //2nd octet IP address, used to unlock reg group
$REGISTERS[21002] = 10;         //3rd octet IP address, used to unlock reg group
$REGISTERS[21003] = 10;         //4th octet IP address, used to unlock reg group
$REGISTERS[21008] = 1003;       // set index to point to protocol register
$REGISTERS[21009] = 3;          //set protocol to Modbus RTU
$REGISTERS[21008] = 1004;       // set serial port to use
$REGISTERS[21009] = 3;          //Use port 1 w/RS232
$REGISTERS[21008] = 1005;       // set Modbus Master Id
$REGISTERS[21009] = 1;          //set to address of the drive (in our case 1)
$REGISTERS[21004] = 10242;      // Modbus start register 2801H
$REGISTERS[21008] = 1007;       // set remapped registers
$REGISTERS[21009] = 23001;      //set to 23001
$REGISTERS[21008] = 0;          //
$REGISTERS[21006] = 10;     // set scan rate to 10ms
```



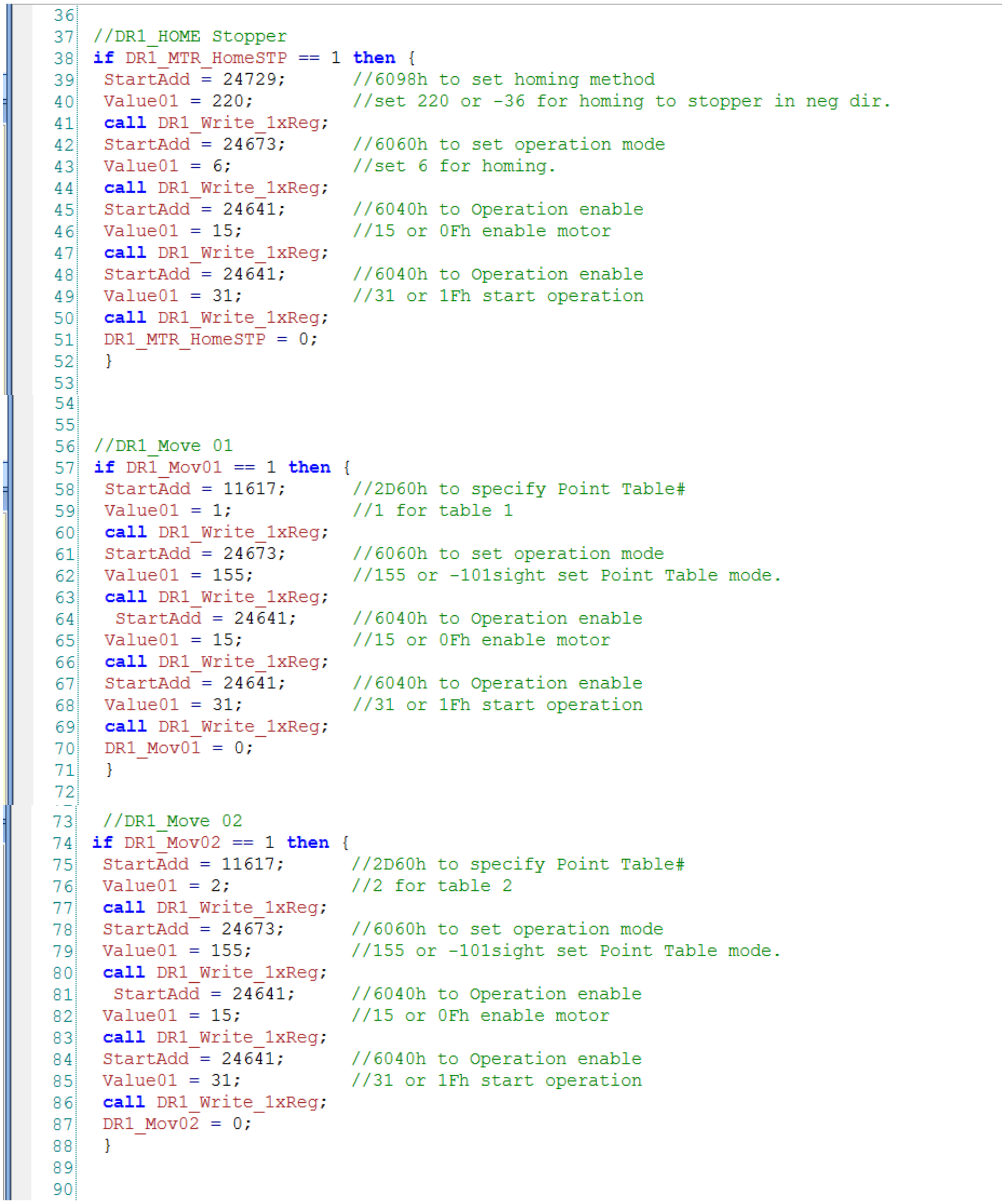
```

// Read/Write 1x register
$REGISTERS[21015] = 1;          //Number of Registers to read
$REGISTERS[21010] = 10;         //1st octet IP address, used to unlock reg group
$REGISTERS[21011] = 10;         //2nd octet IP address, used to unlock reg group
$REGISTERS[21012] = 10;         //3rd octet IP address, used to unlock reg group
$REGISTERS[21013] = 10;         //4th octet IP address, used to unlock reg group
$REGISTERS[21018] = 1003;       // set index to point to protocol register
$REGISTERS[21019] = 3;          //set protocol to Modbus RTU
$REGISTERS[21018] = 1004;       // set serial port to use
$REGISTERS[21019] = 3;          //Use port 1 w/RS232
$REGISTERS[21018] = 1005;       // set Modbus Master Id
$REGISTERS[21019] = 1;          //set to address of the drive (in our case 1)
$REGISTERS[21014] = strAdd_1Reg;        // Modbus start register 6041H, default read
$REGISTERS[21018] = 1007;       // set remapped registers
$REGISTERS[21019] = 23011;      //set to 23001
$REGISTERS[21018] = 0;          //
$REGISTERS[21016] = 10;     // set scan rate to 10ms

// Read/Write 2x register
$REGISTERS[21025] = 2;          //Number of Registers to read
$REGISTERS[21020] = 10;         //1st octet IP address, used to unlock reg group
$REGISTERS[21021] = 10;         //2nd octet IP address, used to unlock reg group
$REGISTERS[21022] = 10;         //3rd octet IP address, used to unlock reg group
$REGISTERS[21023] = 10;         //4th octet IP address, used to unlock reg group
$REGISTERS[21028] = 1003;       // set index to point to protocol register
$REGISTERS[21029] = 3;          //set protocol to Modbus RTU
$REGISTERS[21028] = 1004;       // set serial port to use
$REGISTERS[21029] = 3;          //Use port 1 w/RS232
$REGISTERS[21028] = 1005;       // set Modbus Master Id
$REGISTERS[21029] = 1;          //set to address of the drive (in our case 1)
$REGISTERS[21024] = 8204;       // Modbus start register 200BH, default read address
$REGISTERS[21028] = 1007;       // set remapped registers
$REGISTERS[21029] = 23021;      //set to 23001
$REGISTERS[21028] = 0;          //
$REGISTERS[21026] = 10;     // set scan rate to 10ms
```



```

// Read/Write 9x register
$REGISTERS[21035] = 9;          //Number of Registers to read
$REGISTERS[21030] = 10;         //1st octet IP address, used to unlock reg group
$REGISTERS[21031] = 10;         //2nd octet IP address, used to unlock reg group
$REGISTERS[21032] = 10;         //3rd octet IP address, used to unlock reg group
$REGISTERS[21033] = 10;         //4th octet IP address, used to unlock reg group
$REGISTERS[21038] = 1003;       // set index to point to protocol register
$REGISTERS[21039] = 3;          //set protocol to Modbus RTU
$REGISTERS[21038] = 1004;       // set serial port to use
$REGISTERS[21039] = 3;          //Use port 1 w/RS232
$REGISTERS[21038] = 1005;       // set Modbus Master Id
$REGISTERS[21039] = 1;          //set to address of the drive (in our case 1)
$REGISTERS[21034] = 10242;      // Modbus start register 2801H, default read address
$REGISTERS[21038] = 1007;       // set remapped registers
$REGISTERS[21039] = 23031;      //set to 23001
$REGISTERS[21038] = 0;          //
$REGISTERS[21036] = 10;     // set scan rate to 10ms
```



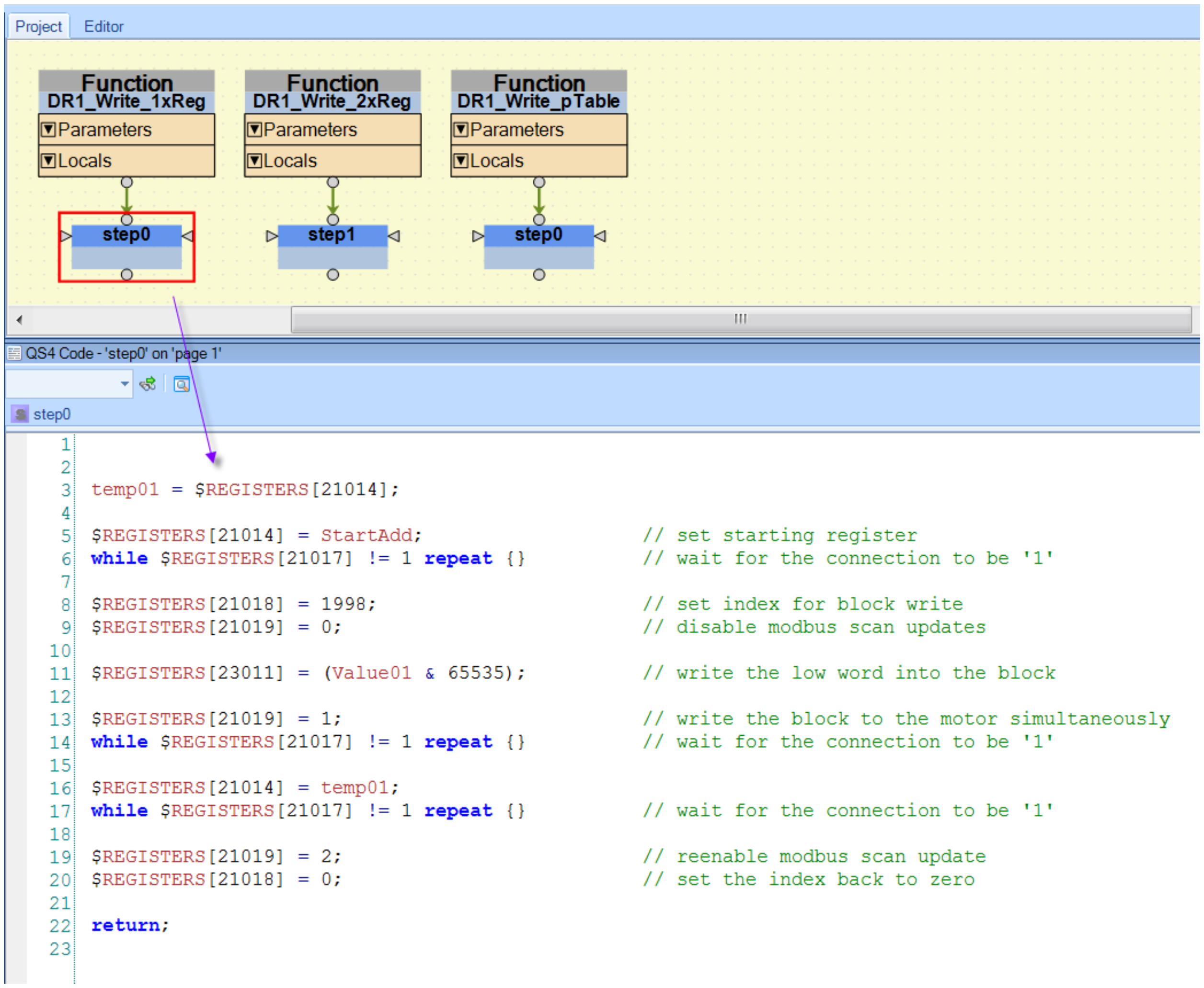
```
//enable Drive

//DR1 Enable drive
if DR1_MTR_On == 1 then {
 StartAdd = 24641;        //6040h to Operation enable
 Value01 = 15;            //15 enable motor
 call DR1_Write_1xReg;
 DR1_MTR_On = 0;
 }
if DR1_MTR_Off == 1 then {
 StartAdd = 24641;        //6040h to Operation enable
 Value01 = 0;             //15 enable motor
 call DR1_Write_1xReg;
 DR1_MTR_Off = 0;
 }



//DR1_HOMING
if DR1_MTR_Home == 1 then {
 StartAdd = 24729;        //6098h to set homing method
 Value01 = 223;           //set 223 or -33 for homing to Dog in neg dir.
 call DR1_Write_1xReg;
 StartAdd = 24673;        //6060h to set operation mode
 Value01 = 6;             //set 6 for homing.
 call DR1_Write_1xReg;
 StartAdd = 24641;        //6040h to Operation enable
 Value01 = 15;            //15 or 0Fh enable motor
 call DR1_Write_1xReg;
 StartAdd = 24641;        //6040h to Operation enable
 Value01 = 31;            //31 or 1Fh start operation
 call DR1_Write_1xReg;
 DR1_MTR_Home = 0;
 }

```



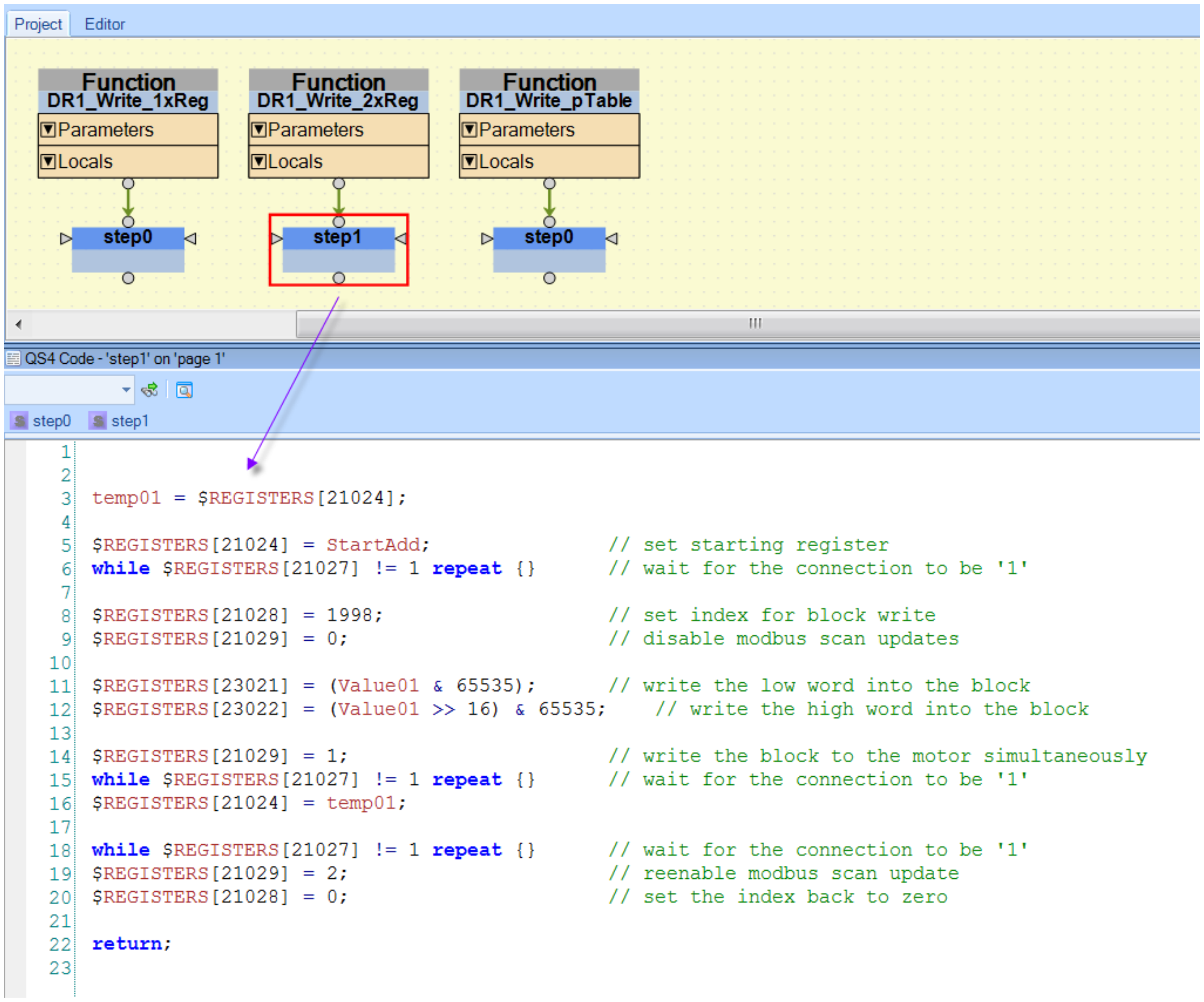
```
//DR1_HOME Stopper
if DR1_MTR_HomeSTP == 1 then {
 StartAdd = 24729;       //6098h to set homing method
 Value01 = 220;          //set 220 or -36 for homing to stopper in neg dir.
 call DR1_Write_1xReg;
 StartAdd = 24673;       //6060h to set operation mode
 Value01 = 6;            //set 6 for homing.
 call DR1_Write_1xReg;
 StartAdd = 24641;       //6040h to Operation enable
 Value01 = 15;           //15 or 0Fh enable motor
 call DR1_Write_1xReg;
 StartAdd = 24641;       //6040h to Operation enable
 Value01 = 31;           //31 or 1Fh start operation
 call DR1_Write_1xReg;
 DR1_MTR_HomeSTP = 0;
 }



//DR1_Move 01
if DR1_Mov01 == 1 then {
 StartAdd = 11617;       //2D60h to specify Point Table#
 Value01 = 1;            //1 for table 1
 call DR1_Write_1xReg;
 StartAdd = 24673;       //6060h to set operation mode
 Value01 = 155;          //155 or -101sight set Point Table mode.
 call DR1_Write_1xReg;
  StartAdd = 24641;      //6040h to Operation enable
 Value01 = 15;           //15 or 0Fh enable motor
 call DR1_Write_1xReg;
 StartAdd = 24641;       //6040h to Operation enable
 Value01 = 31;           //31 or 1Fh start operation
 call DR1_Write_1xReg;
 DR1_Mov01 = 0;
 }

 //DR1_Move 02
if DR1_Mov02 == 1 then {
 StartAdd = 11617;       //2D60h to specify Point Table#
 Value01 = 2;            //2 for table 2
 call DR1_Write_1xReg;
 StartAdd = 24673;       //6060h to set operation mode
 Value01 = 155;          //155 or -101sight set Point Table mode.
 call DR1_Write_1xReg;
  StartAdd = 24641;      //6040h to Operation enable
 Value01 = 15;           //15 or 0Fh enable motor
 call DR1_Write_1xReg;
 StartAdd = 24641;       //6040h to Operation enable
 Value01 = 31;           //31 or 1Fh start operation
 call DR1_Write_1xReg;
 DR1_Mov02 = 0;
 }
```

The code above remaps the Basic Group of Mitsubishi Parameters in the Mitsubishi drive to the Model 5300's 23000 registers.

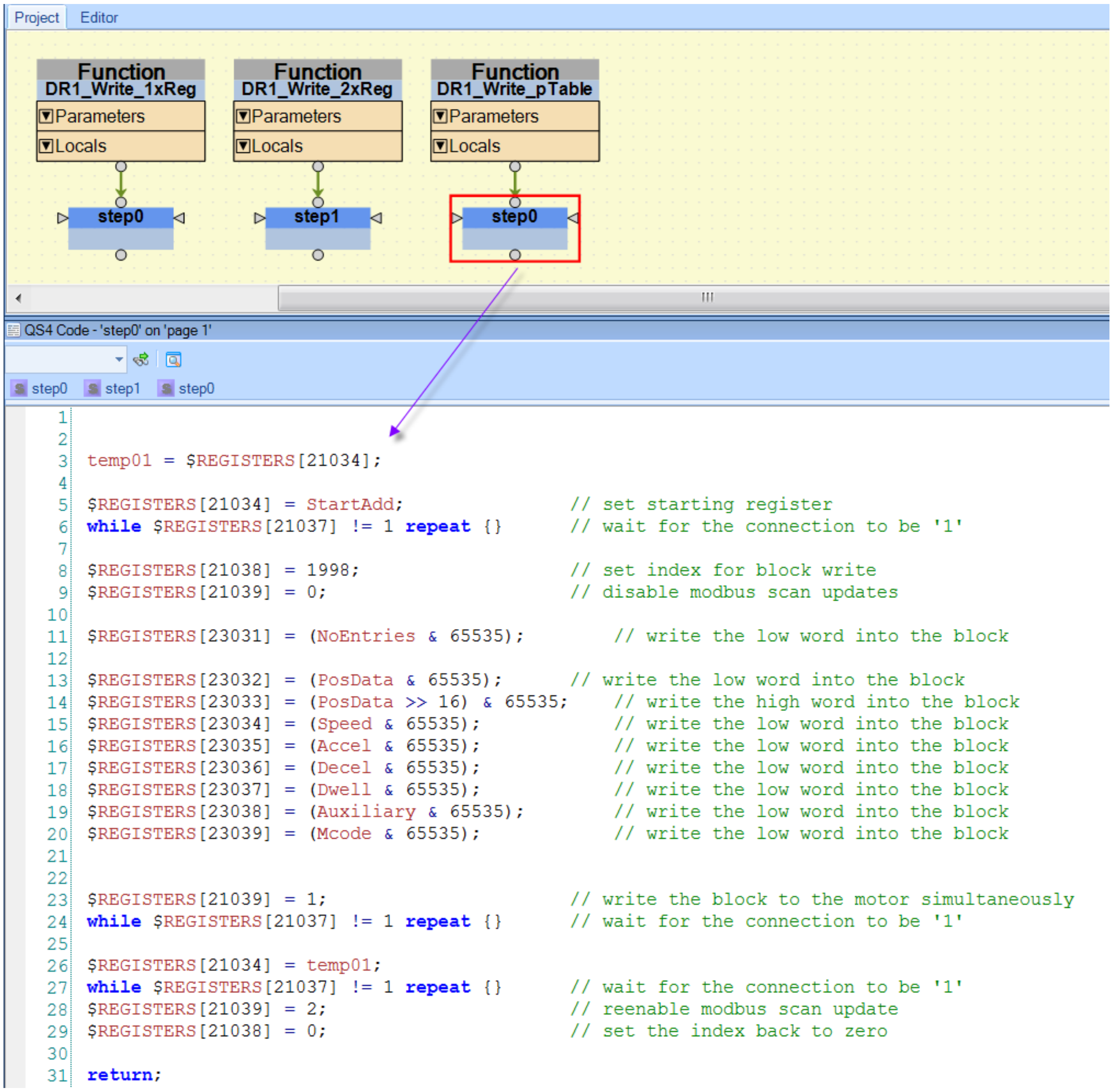
The Mitsubishi MR-JE-_A supports various methods of Modbus reads and writes. Achieving this can be accomplished by configuring several sets of "21**xx**0" registers.
In this example, register series 21**01**0 configures Modbus for a single-register read/write. Register series 21**02**0 configures Modbus for a double-register read/write. Lastly, register series 21**03**0 configures Modbus for a block-register read/write. Functions are called to execute each configuration prior to reading and writing the desired data. Each function is configured as follows:



```

temp01 = $REGISTERS[21014];

$REGISTERS[21014] = StartAdd;                          // set starting register
while $REGISTERS[21017] != 1 repeat {}                 // wait for the connection to be '1'

$REGISTERS[21018] = 1998;                              // set index for block write
$REGISTERS[21019] = 0;                                 // disable modbus scan updates

$REGISTERS[23011] = (Value01 & 65535);                 // write the low word into the block

$REGISTERS[21019] = 1;                                 // write the block to the motor simultaneously
while $REGISTERS[21017] != 1 repeat {}                 // wait for the connection to be '1'

$REGISTERS[21014] = temp01;
while $REGISTERS[21017] != 1 repeat {}                 // wait for the connection to be '1'

$REGISTERS[21019] = 2;                                 // reenable modbus scan update
$REGISTERS[21018] = 0;                                 // set the index back to zero

return;

```



```

temp01 = $REGISTERS[21024];

$REGISTERS[21024] = StartAdd;                    // set starting register
while $REGISTERS[21027] != 1 repeat {}           // wait for the connection to be '1'

$REGISTERS[21028] = 1998;                        // set index for block write
$REGISTERS[21029] = 0;                           // disable modbus scan updates

$REGISTERS[23021] = (Value01 & 65535);           // write the low word into the block
$REGISTERS[23022] = (Value01 >> 16) & 65535;        // write the high word into the block

$REGISTERS[21029] = 1;                           // write the block to the motor simultaneously
while $REGISTERS[21027] != 1 repeat {}           // wait for the connection to be '1'
$REGISTERS[21024] = temp01;

while $REGISTERS[21027] != 1 repeat {}           // wait for the connection to be '1'
$REGISTERS[21029] = 2;                           // reenable modbus scan update
$REGISTERS[21028] = 0;                           // set the index back to zero

return;

```



```


temp01 = $REGISTERS[21034];

$REGISTERS[21034] = StartAdd;                    // set starting register
while $REGISTERS[21037] != 1 repeat {}           // wait for the connection to be '1'

$REGISTERS[21038] = 1998;                        // set index for block write
$REGISTERS[21039] = 0;                           // disable modbus scan updates

$REGISTERS[23031] = (NoEntries & 65535);             // write the low word into the block

$REGISTERS[23032] = (PosData & 65535);           // write the low word into the block
$REGISTERS[23033] = (PosData >> 16) & 65535;         // write the high word into the block
$REGISTERS[23034] = (Speed & 65535);                 // write the low word into the block
$REGISTERS[23035] = (Accel & 65535);                 // write the low word into the block
$REGISTERS[23036] = (Decel & 65535);                 // write the low word into the block
$REGISTERS[23037] = (Dwell & 65535);                 // write the low word into the block
$REGISTERS[23038] = (Auxiliary & 65535);             // write the low word into the block
$REGISTERS[23039] = (Mcode & 65535);                 // write the low word into the block


$REGISTERS[21039] = 1;                           // write the block to the motor simultaneously
while $REGISTERS[21037] != 1 repeat {}           // wait for the connection to be '1'

$REGISTERS[21034] = temp01;
while $REGISTERS[21037] != 1 repeat {}           // wait for the connection to be '1'
$REGISTERS[21039] = 2;                           // reenable modbus scan update
$REGISTERS[21038] = 0;                           // set the index back to zero

return;
```