Modbus PLC Programming with Panasonic AFPX PLC

Jim Amos

Introduction

For this exercise, I connected a Panasonic AFPX-C14R to an MOONS' STM24QF-3AE integrated stepper motor. I programmed the PLC to command simple moves using Modbus/RTU protocol and RS-232 communication. User input was accomplished by wiring a pushbutton switch to the X4 digital input. I'm using the FPWIN GR free demo version of Panasonic PLC programming software.

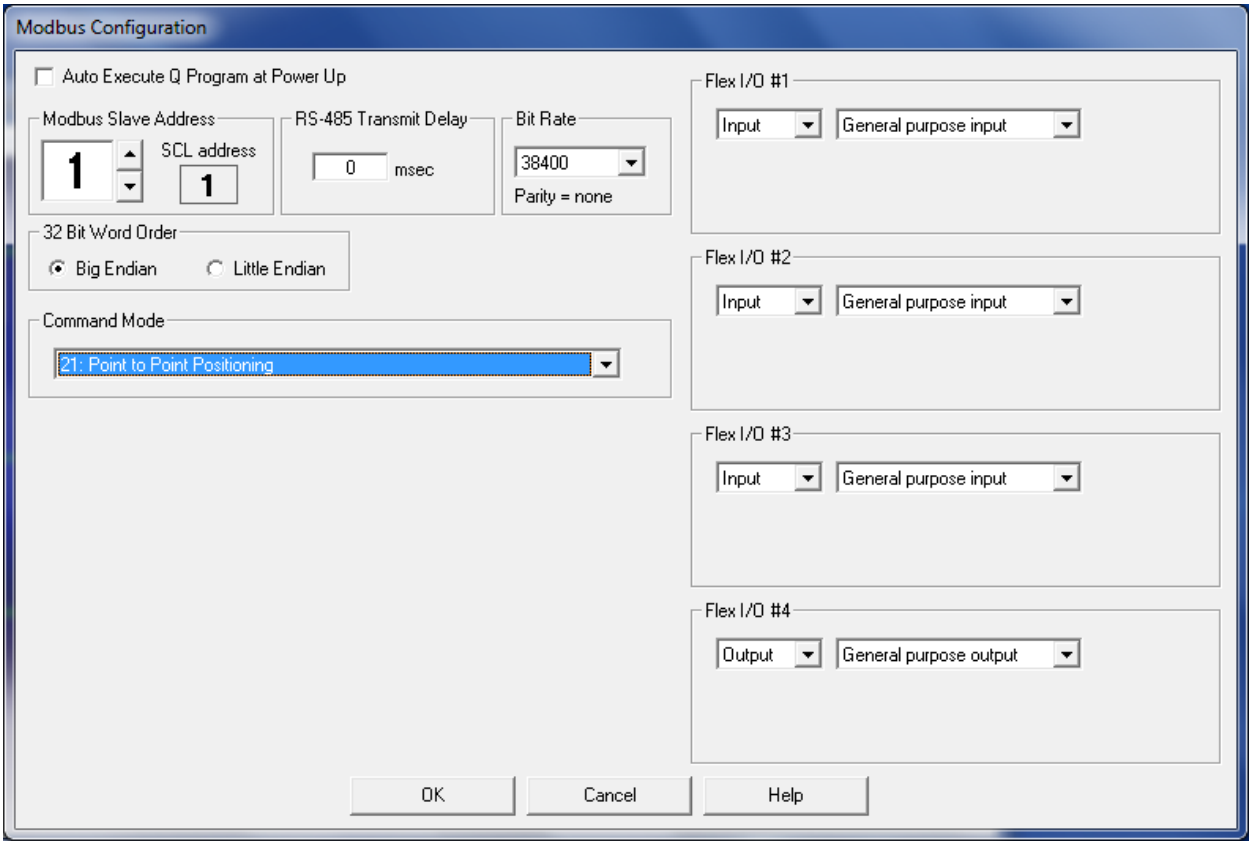
Serial Connection

Modbus/RTU can use RS-232, RS-422 or RS-485 as a physical layer. It can use any bit rate and any choice of parity and stop bits. It is the job of the user to make sure both sides are set the same and properly connected.

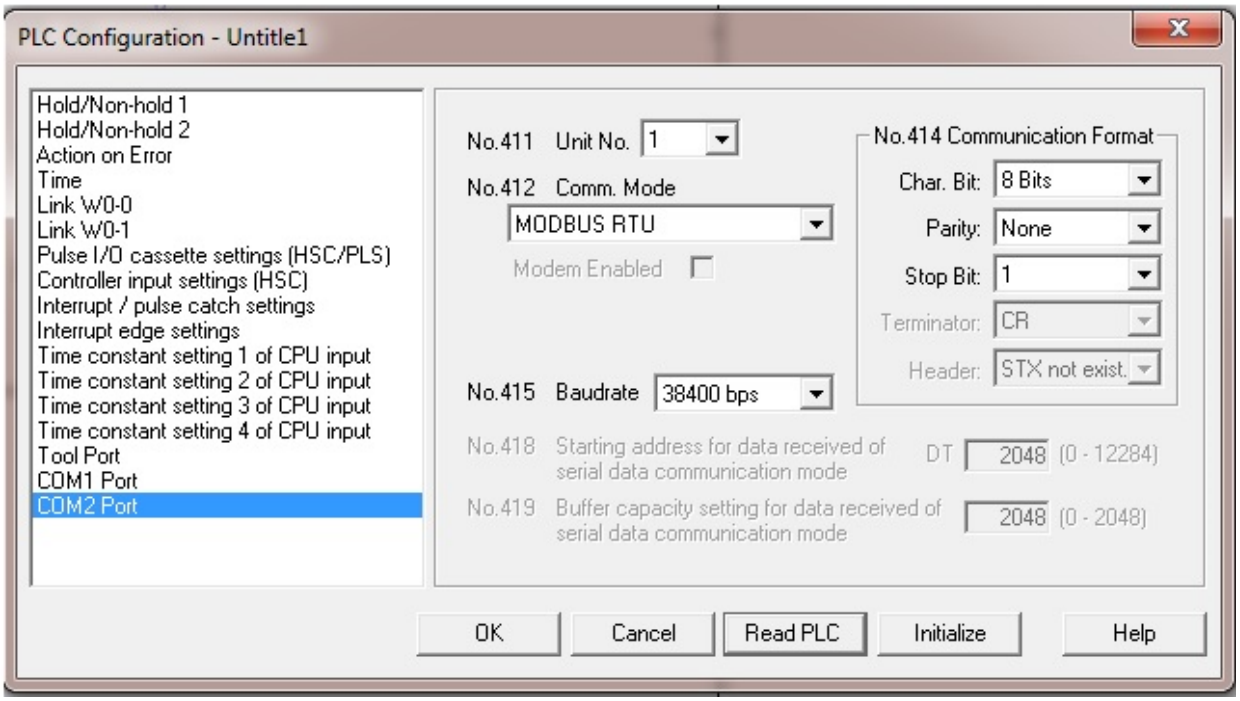
For these examples I used an RS-232, three wire connection (RX, TX, and GND), 38400 bps and no parity. The AFPX PLC uses the optional COM5 Cassette adapter that has two connection points. COM1 is an Ethernet port, COM2 is the screw terminal RS-232 connection point.

Serial Port Settings

On the drive end: use *ST Configurator* to set the drive for Modbus mode, command mode 21 (point to point positioning), 38400 bps. Our drives are always set for "no parity". This is also where you will enter the drive's address; 1 has been chosen for this example.



After closing *ST Configurator*, be sure to power cycle the STM24 so it wakes up at the correct bit rate. On the PLC side, I'm connecting the drive to the AFPX COM2 port on the COM5 adaptor. The PLC comes with a serial programming cable that I connected to an FTDI based USB serial adaptor for programming with FPWIN GR software from Panasonic.



Register Mapping

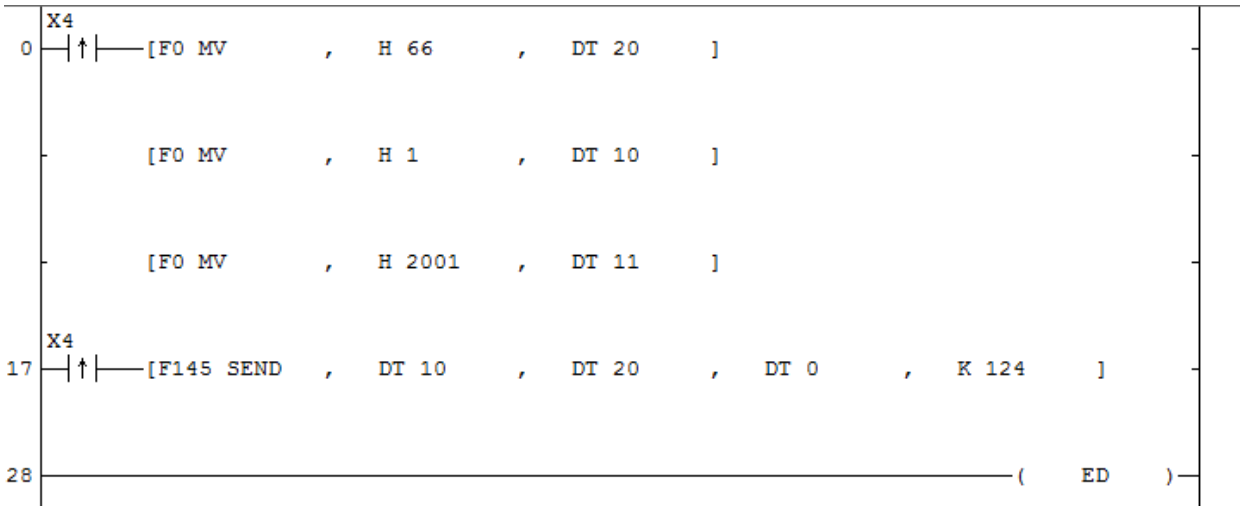
The Modbus protocol is all about moving data from the memory of one device to that of another. You can move as little as one bit or you can move one or more 16 bit words. Most parameters and commands are one 16 bit word. Move distance (DI) is 32 bits in our drives as are some of the monitor values. For the 32 bit values we have to pay attention to word order or endianess.

In our Modbus implementation we default to having the big end of 32 bit values in the first word of memory. That's big endian. My AFPX uses big endian so I don't need to set the endian register to 1 to use little endian.

These are the Modbus registers that we'll be using in the STM24:
40125 = command register
40123 = endian register
40028...40032 = move parameters (AC, DE, VE and DI)
40043...40045 = jog parameters (JA, JL and JS)
40001...40015 = immediate registers for monitoring the drive (AL, SC, IT, IU, etc)

Exercise 1: Point-to-Point Move

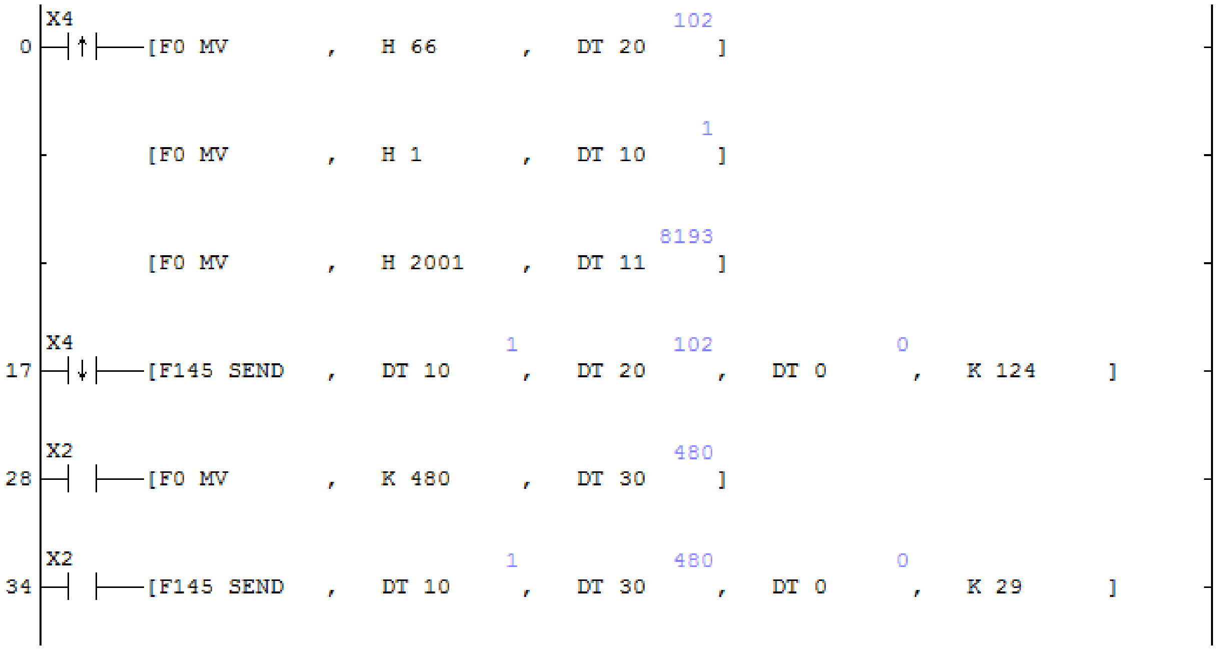
For this first exercise I'll make a simple feed command based on previously stored acceleration, deceleration, velocity and distance parameters in the STM24. I used our SCL Utility to set those parameters. The FP-X has general purpose memory area (D locations) in the range of D0 through D1659. This gives us 1600 16 bit word locations. We'll use those for Modbus transfers.



I'm using the F type high level instruction to load Opcode 0x66 into DT register 20. This is the Opcode for our SCL Feed to Length (FL) command. I'll load the drive's address into register DT10 and load the COM2 (H2) word into register DT11. To launch the move I'll use the Executable instruction (F145) to send the 0x66 to the STM24's command register 40125. For some unknown reason Modbus ads or subtracts registers by one or two sometimes so that's why we see K124 rather than K125.

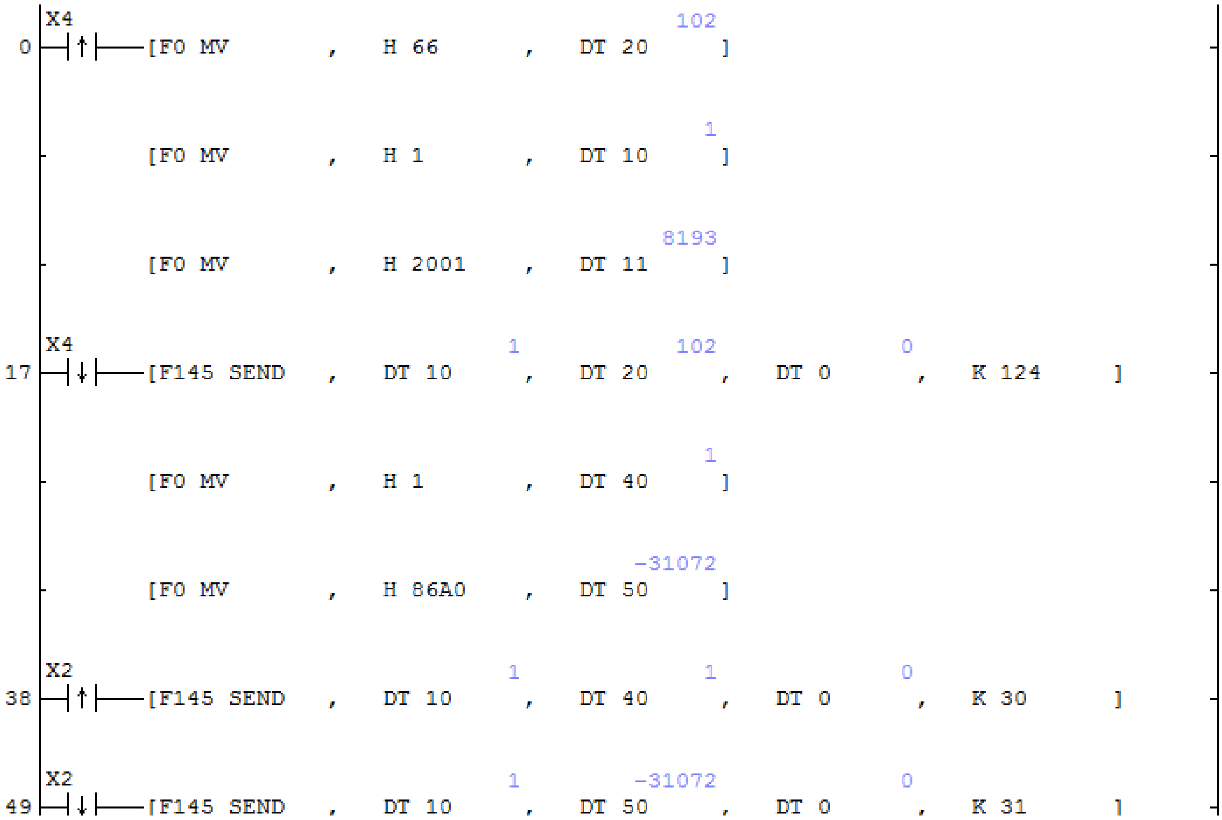
Exercise 2: Speed Change

In the next exercise we'll let the PLC change the speed of the move when X2 is triggered. This loads the drive's V register with a velocity value, which is expressed in units of 0.25 of one RPM. In this case, 480 x 0.25 = 120 RPM. The Modbus register on the drive for velocity is 40030 so we load 240 into K29 rather than K30.



Exercise 3: Change Distance

Now I'm going to let the PLC set the distance.

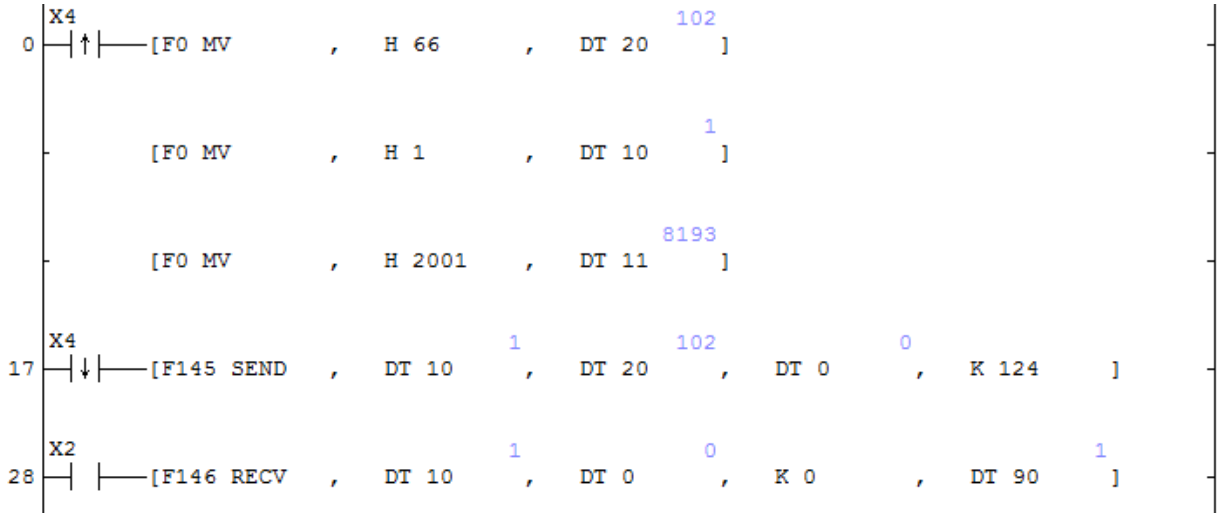


Again I'm using X4 to launch a move based on default values stored in the STM. When X2 rises it sends the first word of a new DI value then the falling edge sends the second word. In this case I sent 100,000

counts (000186A0 hex). There is a high level function (F1 DMV) that can be used to send a 32 bit value, but I wanted to clearly demonstrate the order of endianess.

Exercise 4: Alarm Monitoring

Most applications would want to monitor the health of the drive. The next example shows my X2 input triggering a read of the alarm register 40001. This monitoring would normally be done with an internal timer set to monitor the alarm code every few milliseconds. The results would be compared to stored data using logic to determine corrective actions and or notification to the outside world. For simple demonstration I manually stalled the STM. Its optional encoder detected the stall and changed the alarm code from 0 to 1 which was loaded into the FPX register 90.



Complete project files for the FPWIN GR software are available at www.moonsindustries.com.