Wago750-841 Ethernet controller with JAMOD

By
Krishna Akkulu

This paper describes an implementation of a Java client program for the Wago 750-841 Ethernet controller. The Wago manual explains that the controller supports a number of network protocols to send and process data via Ethernet TCP/IP. But the Wago manual does not provide any demo or sample programs to control the device from remote machines.

This becomes really hard for programmers who wish to control the Wago Ethernet controller from a remote machine using a C++ or Java program. As my requirement was to control the Wago controller from a Linux box, I chose Java language to work with the device. After extensive research I found the JAMOD library to be the right choice for this device.

I first started working with "TCP Master HOW-TO" program as described in the tutorial at the URL http://jamod.sourceforge.net/development/tcp_master_howto.html. But the TCP Master tutorial did not explain how it generates the Modbus request or what is the right request to be send to the Wago controller.

Finally, I found the ModbusTCPMaster to be the right class to use for controlling the Wago device. Before we go into the details of the program it is very essential to understand how Wago 750-841 Ethernet controller addresses I/O modules. Although the controller supports both digital and analog I/O modules, I selected only digital modules for my presentation.
Figure 1: Shows the Wago 750-841 Ethernet controller packed with various I/O modules.

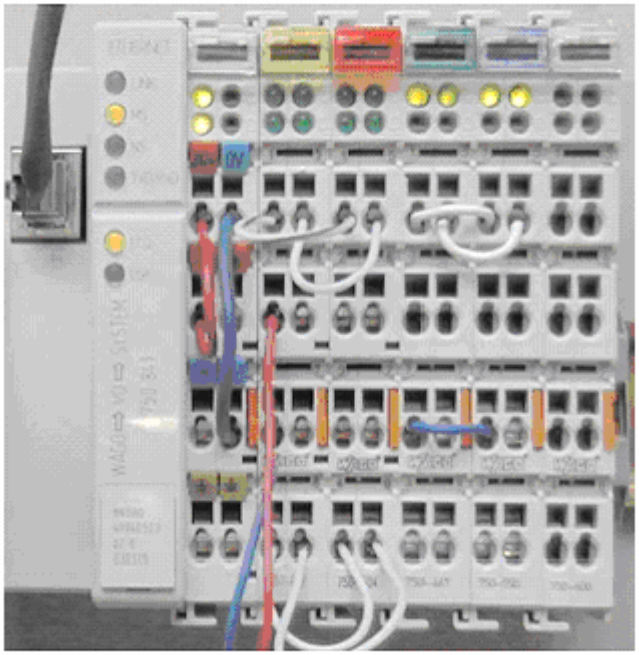

Fig 1: Wago 750-841 Controller with I/O Modules

Let us assume the Wago controller is packed with the I/O modules in the order shown in Table1.

Note:
Although Figure 1 shows only 6 modules, I refer to 7 modules listed in Table 1 in this discussion.

Module Name	Module Type	Number of Channels
750-601	Supply Module DC 24V	
750-502	Fuse(Output) Module	2 output channels
750-430	Input Module	8 Input Channels
750-537	Output Module	8 Output Channels
750-431	Input Module	8 Input Channels
750-530	Output Module	8 Output Channels
750-600	End module	

Table 1: Controller with I/O modules.

From Table 1, we observe that the total numbers of I/O channels are 34, but the controller counts Output and Input module channels separately. For the controller:

Output modules (2+8+8) = 18
Input modules (8+8) = 16

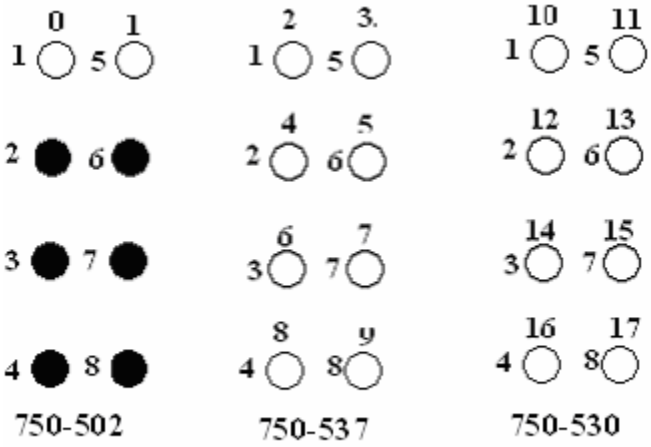

Fig:2 Output modules with Channel numbers

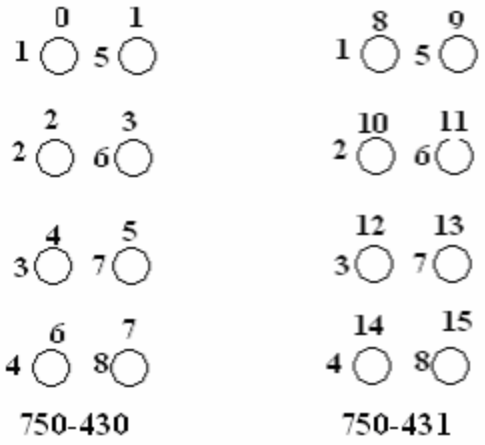

Fig:3 Input modules with Channel numbers

Although the Wago manual does not explain how the controller identifies the channel numbers, it is important to understand the channel numbering system. Fig 2 and Fig 3 display Output and Input modules respectively. In both the modules, the vertical numbers are channel numbers (1 to 8) physically printed on each module. The first column shows 1 to 4 and second column shows 5 to 8. But the Wago Ethernet controller counts the channel numbers differently. For both Output and Input modules, the Wago Contoller starts counting the channel numbers from 0, and it continues counting in horizontal direction instead of vertical direction, and maps these numbers to physical numbers.

Example: If you wish to set a bit for 8th channel in the 3rd output (750-530) module, then you have to pass the channel number as 17 to the program. Similarly, if you wish to find the status for 3rd channel of 2nd input (750-431) module then you need to pass the channel number as 12.

Now we discuss the program by considering two examples.

Example1: *Set a bit ON for 8th channel in 3rd output module*.

Program steps are given below:

1) Create an Instance of ModbusTCPMaster using IPAddress and port numer = 502
   ModbusTCPMaster m_MBMaster =
   new ModbusTCPMaster(IPAddress, portnumber);
2) Connect to Modbus
   m_MBMaster. Connect()
3) Set bit on or off of desired channel.
   Use ***writeCoil****(int unitid, int ref, boolean state)*
   Pass parameters as
   unitid = 0
   ref =17(8th channel in 3rd output module)
   state = true (on the bit)
   = false (off the bit)

m_MBMaster.writeCoil(0, 17, true)

4) Disconnect

m_MBMaster. disconnect()

Example 2: *Get the status for 3rd channel in 2nd input module(750-431).*

For getting status of the Input channel, refer to the steps in *Example 1,* The steps are the same except for the 3rd step. In the 3rd step, replace the call to ***writeCoil*** method with the **`readCoils`**`(int ref, int count) method.`

In the readCoils method, *ref* =0, *count*= maximum number of input coils.

BitVector oBitVector **= readCoils**(int ref, int count)

To find out status of 3rd channel in 2nd Input module, pass 12 as shown below.

Boolean bstatus = oBitVector. getBit(12);

bstatus is ***true*** then bit is ON, else bit is OFF.

Note:- This program requires JAMOD jar file. The file can be downloaded from http://sourceforge.net/project/showfiles.php?group_id=48413

```
/**
*The Program lines are listed below.
*/
package wago.com.impl;

import java.net.*;
import java.util.HashMap;
import java.util.Map;
import net.wimpi.modbus.util.*;
import net.wimpi.modbus.facade.ModbusTCPMaster;
import net.wimpi.modbus.ModbusException;
import com.sony.dcinema.digitalIO.ModbusTCPIfc;

public class WagoOpreations {
        InetAddress m_SlaveAddress;
        int m_port =502;
        ModbusTCPMaster m_MBMaster = null;

        private WagoOpreations(InetAddress ipAddress)
        {
         m_SlaveAddress = ipAddress;
         }

        /*
        * This function connects to Wago Controller by passing IPAddress and
        * Modebus port number.
        */
```

```
 private void connect()
 {
     try  {
          m_MBMaster = new ModbusTCPMaster( SlaveAddress.getHostName(),
                    m_port);
          m_MBMaster.connect();
    }
    catch (Exception ex)
    {
        System.out.println("connection fail"+  ex.printStackTrace());
     }
 }
/*
* This function disconnects with Wago Control .
*/
private void disconnect()
{
      if(m_MBMaster!=null)
     {
         m_MBMaster.disconnect();
         m_MBMaster=null;
    }
 }
 public void setStatus(int coil, boolean bvalue)
 {
     try  {
        if(m_MBMaster!=null)
       {
          m_MBMaster.writeCoil(0, coil, bvalue);
     }
   }catch (ModbusException mex)
   {
       mex.printStackTrace();
    }
    catch (Exception ex)
   {
      ex.printStackTrace();
    }
}
/*
*This function return status of selected channel.
*/
 public boolean getStatus(int coil)
 {
        boolean bValue = false;
         try {
```

```
                    if(m_MBMaster!=null)
                   {
                      BitVector obit=m_MBMaster.readCoils(0,MAX_READ_COILS);
                      bValue= obit.getBit(coil);
                    }
               }
               catch (ModbusException mex)
               {
                   mex.printStackTrace();
                }
               return bValue;
           }
}
```