K-DUCER Modbus TCP Programmer Manual

Note: we now have a C# library and Siemens S7 function block library that abstract away all Modbus TCP implementation details from the user.
If you are using C# please refer to:
https://kolver.com/us/blog-detail/kolver-s-new-nuget-net-library and
https://www.nuget.org/packages/Kolver.Kducer
If you are using Siemens S7 PLCs:
https://kolver.com/us/blog-detail/siemens-s7-plc-integration-now-available
https://kolver.com/upl/Siemens%20S7%20library%20and%20example.zip
We also have a URCap for Universal Robots:
https://kolver.com/us/blog-detail/universal-robots-kolver-introducing-our-new-urcap-plugin
https://kolver.com/upl/URCAP-KDUCER.zip

Resource Packet Contents

The *KDUCER ModBus TCP resources* packet contains the following files and documents:

- **KDU-1A_Modbus_Map_RevXY.xlsx**: table of all MODBUS data and addresses for **KDU-1A** controller
- **KDU-NT_Modbus_Map_RevXY.xlsx** table of all MODBUS data and addresses for **KDU-NT** controller
- **AllenBradley PLC MODBUS resources:** example project integrating the K-DUCER with a Micro820 PLC via MODBUS, plus guides for implementing a MODBUS TCP client with CompactLogix and ControlLogix PLCs
- **Siemens PLC MODBUS resources**: K-DUCER function block library for S7 PLCs and example project using it
- **Script examples:** various examples in python and powershell demonstrating the functionality described here

Table of Contents

Usage 2
K-DUCER MODBUS map 3
Writing values to HOLDING REGISTERS 3
Method 1: modify settings in volatile memory only 4
Method 2: modify settings in volatile and permanent memory 5
Special holding registers 5
General implementation guidelines and tips 6
Typical program loops 7
MODBUS TCP code examples and literature 9

Introduction

The recommended way to interface with the K-DUCER unit is through the MODBUS TCP protocol on the ethernet port (CN5).
MODBUS communication protocol provides a Client-Server interface between devices connected on an ethernet TCP/IP network.
The MODBUS protocol specifications are open source and freely available online at modbus.org, however most automation engineers will not need to worry about the implementation details because MODBUS is already supported and implemented by most ethernet-capable PLCs and industrial PCs.

Usage

Enable MODBUS TCP via the General Settings menu > Communication Protocol. The K-DUCER should be connected to the same LAN network as the client device, and it must be left in the main operation screen, outside of any configuration menu.
Note: the K-DUCER will respond to the *ping* command over TCP/IP when configured correctly.

The K-DUCER implements a MODBUS server, which responds to MODBUS requests. The automation device (PLC, industrial PC, …) must implement a MODBUS client, which sends MODBUS requests to the server (K-DUCER).

The MODBUS server (K-DUCER) only responds to requests and never initiates any communication independently, in accordance with the MODBUS protocol.
A MODBUS request is simply a message requesting to read or write one or more *bits* or *bytes* of data at a particular address. The list of all accessible data and their addresses is called the MODBUS map.

MODBUS requests are categorized into *function codes*. Different function codes are used to access different types of data (bits-coils or byes-registers). There are also convenience function codes used to access a range of multiple data addresses at once.

All program, sequence, and general settings can be modified via MODBUS requests. However, Kolver recommends pre-configuring the K-DUCER programs and settings via the K-Expand software, via touch screen, or via kdu backup file from USB, and only utilizing the MODBUS TCP protocol for screwdriver control, program switching, and data acquisition.

K-DUCER MODBUS map

The K-DUCER, MODBUS data is organized and accessed as follows:

Data Category	Contents	Access	Associated MODBUS function codes
COILS (*bits*)	A mirror copy of the CN3 output pins 23 to 43 represented as bits; Writeable coils mimicking the functionality of CN3 input pins 13 to 20, providing screwdriver motor control capability	Read/ Write	01 (read coils) 05 (write single coil) 15 (write multiple coils)
INPUT REGISTERS (*bytes*)	Data related to the last screwdriving results including closing torque and angle; torque/angle charts; current screwdriving state and errors; connected screwdriver info	Read only	04 (read input registers)
HOLDING REGISTERS (*bytes*)	Current selected program; All program settings; All sequence setting; Current selected sequence; General settings; Remote programming mode enter/exit flag;	Read/ Write	03 (read holding registers) 06 (write single register) 16 (write multiple registers)
DISCRETE INPUTS	A mirror copy of the CN3 input pins 1 to 20 represented as bits	Read only	02 (read discrete inputs)

The full MODBUS map can be found in the attached document **KDucer_Modbus_Map_Rev29.xlsx.**

Writing values to HOLDING REGISTERS

Starting with KDU firmware version 38* (i.e., mainboard versions G.00.38, H.00.38, M.00.38 when in the General Settings => Info button screen), there are two ways to write values to the Modbus holding registers.

*Kolver can upgrade any KDU-1A controller to firmware version v38 for free

Method 1: modify settings in volatile memory only

This is the recommended method for all applications that do not require retaining values written to holding register after a powering off the KDU controller.

Target applications

- applications where the operator is not allowed to modify the program/sequence values directly on the KDU controller
- applications where the KDU is reprogrammed via Modbus when the application first boots up
- applications that provide an HMI to re-program the KDU controller that is intended to substitute the touch screen interface on the KDU controller

In these types of applications, it is not important to retain the program/sequence settings upon rebooting the KDU, because the application will re-write the KDU configuration when it boots.

The advantage of this method over method 2 is that it is about 10x faster to modify one or more parameters, by virtue of not having to save the new parameters onto permanent memory.

The disadvantage is that the settings are lost when the operator enters the KDU configuration menu, or when rebooting the controller. In both cases the settings in permanent memory are loaded.

Usage

Simply write the desired values via Modbus function codes 6 or 16.
As soon as a write register command is received, the KDU will temporarily enter the "REMOTE PROGRAMMING.. PLEASE WAIT" screen.
The KDU will remain in this screen for 200 milliseconds. The 200ms timer is reset if another write register command is received while in this screen.
When the timer expires, the modified register values will be applied and the KDU will return to the main screen.

Method 2: modify settings in volatile and permanent memory

Method 2 is used to save settings to permanent memory (retained when the controller is powered off) and is the only method available with KDU firmware version 37 and prior.

Target applications

Any application where it is required that the KDU retains the values written to the holding register when the operator enters the configuration menu or after rebooting the controller.
In both of these cases, the application could still use method 1 as long as the values are re-written after the power-off event or the user configuration event.

Usage

- Write the value "1" to address 7790 to enter "Remote Programming Mode" using MODBUS function code 06 "write single register"
    - The controller will display "REMOTE PROGRAMMING… PLEASE WAIT" indefinitely
- Change the desired holding register values using MODBUS function codes 06 or 16
- Write the value "2" to address 7790 to apply the changes and exit "Remote Programming Mode"
    - The controller will exit the "REMOTE PROGRAMMING" screen and return to the main screen
    - This step takes about 2 seconds
- It is also possible to write value "0" to exit remote programming mode without applying any of the changes (the holding register values are restored in this case)

Special holding registers

Some holding registers can be written to without triggering the "REMOTE PROGRAMMING… PLEASE WAIT" screen (method 1) or without having to enter Remote Programming Mode (method 2).

These registers are:

- 7373 (7372 base-0) Current Program (if the general setting "remote progr" is set to "CN5 TCP")
- 7372 (7371 base-0) Current Sequence (if the general setting "remote seq" is set to "CN5 TCP")
- 7380 (7379 base-0) Barcode
- 7790 (7789 base-0) Remote Programming Status (to enter and exit Remote Programming Mode for method 2)

General implementation guidelines and tips

- **Make sure your program can gracefully handle Modbus Exception Codes**:

Exception Code	MODBUS name	Comments
01	Illegal Function Code	The function code is unknown by the server
02	Illegal Data Address	Dependant on the request
03	Illegal Data Value	Dependant on the request
04	Server Failure	The server failed during the execution
05	Acknowledge	The server accepted the service invocation but the service requires a relatively long time to execute. The server therefore returns only an acknowledgement of the service invocation receipt.
06	Server Busy	The server was unable to accept the MB Request PDU. The client application has the responsibility of deciding if and when to re-send the request.

  - Exception code 6 "Server Busy" will be used by the KDU in the following situations:
    - Attempting to write to a holding register while the screwdriver motor is running
    - Attempting to write to a holding register while a user is navigating the configurating menus on the KDU touch screen interface
- **Take advantage of the following settings on the KDU controller**:
  - REMOTE PROG => CN5 TCP: enables switching program # via Modbus without triggering/entering remote programming mode, and disables the program change button on the KDU touch screen so your operators can't override the program selection
  - REMOTE SEQ => CN5 TCP: same as above, for sequence selection
  - LOCK IF CN5 NOT CONNECTED: locks the KDU controller and disables the screwdriver if/when the Modbus TCP connection is not open. Use this if you want the KDU controller to only be usable when your Modbus Client is connected.

- **Avoid opening and closing the TCP connection for every Modbus request**, as this will slow down your program substantially. Instead, open the TCP connection on startup, leave it open indefinitely, and close/reopen only in case of a disconnection event
- The KDU client can receive asynchronous Modbus requests and will process them in the order in which they are received. However, the aggregate/average request rate should not exceed 1 request every 5 (five) milliseconds or you may accumulate requests faster than the KDU can process them and eventually the KDU may drop the connection if too many requests are accumulated. When in doubt, **keep cyclical request rates at 10ms or longer, and/or use synchronous requests**
- When instantiating the Modbus TCP Client object, **configure the TCP timeouts to around 1 second**. The default TCP timeouts on most PC and PLCs are tuned for internet connections (15 to 30 seconds, or even worse, "infinite" timeout) which will slow down your program in the event of a disconnection from the KDU controller
- If cyclically reading multiple coils or multiple registers, **read all values in a single pass as opposed to issuing multiple read commands for a single value**

Typical program loops

These are tips/guidelines meant to complement the code examples provided in this packet.

To monitor for new screwdriving results

The KDU will only store the latest screwdriving result data on the input registers. This means that if the registers are not polled frequently enough, a new result could overwrite an older result before your program has had the opportunity to see it. Fortunately, this is nearly impossible if the polling rate is kept reasonably fast, and the special input register "New Result Available" address 295 (294 base-0) should make this easy to implement.

For example:

Cyclically poll Input Register 295 (294 base-0) "New Result Available" every 50 milliseconds*.
The value of this register always resets to zero after every read. This means that you will only see a value "1" once for each unique screwdriving result.

When the value returned is 1, it means there is new screwdriving result data ready on the other Input Registers, and only at this point your program should proceed to read the other input registers of interest.
Most likely your program will need to read multiple input register values. Use Modbus function code 4 “Read Input Registers” with an address range that encompasses all the values of interest to get everything in a single pass as opposed to issuing multiple reads of each single register. If your registers of interest are not contiguous, it’s still recommended to use a single pass read and ignore the unneeded values as opposed to using multiple requests.
Because the maximum number of registers that can be read is 125 (Modbus protocol limitation), it may not be possible to get all the values of interest in a single read, for example if retrieving the torque-angle / torque-time graph data. In this case it is of course necessary to issue multiple read requests to get all the data.

*in the case of extremely fast back-to-back results, for example when using sequence auto-transition mode with very short rundowns, you can increase the polling frequency to 10 milliseconds or any value in between. It is not recommended to poll faster than this as it is unlikely to be necessary nor beneficial.

To remote-control the screwdriver

Alternate between cyclically checking for a new result (tightening finished) as per the paragraph above, and cyclically writing the remote lever command via the REMOTE LEVER coil address 33 (32 base-0).

The REMOTE LEVER coil must be cyclically written-to. If it is not written to for 0.5 seconds, the controller will behave as if the screwdriver lever was released (the motor will stop and the “release lever error” will be raised if said error is enabled in the program parameters).

When the new result is detected, after reading the result registers of interest, you can write “0” (zero) to the REMOTE LEVER coil to re-enable the REMOTE LEVER coil for the next tightening. Alternatively, you can wait at least 0.5 seconds from the last time the REMOTE LEVER coil was written, to re-enable it, as per the same release-lever mechanism described above.

MODBUS TCP code examples and literature

We provide sample projects illustrating K-DUCER screwdriver control built by Kolver for various devices, as well as generic MODBUS TCP guides and literature produced by the manufacturers of these devices.
We also recommend searching youtube for a multitude of freely available videos illustrating how to implement MODBUS TCP communication with various control systems.

PC (multiplatform) with Python, C#, and PowerShell

If you are using C# please refer to:
https://kolver.com/us/blog-detail/kolver-s-new-nuget-net-library and
https://www.nuget.org/packages/Kolver.Kducer

Included with the packet are the following scripts:

- *KDUCER_modbus_obtain_screwdriving_results_python.py*
- *KDUCER_modbus_obtain_screwdriving_results_PowerShell.ps*
    - This stand-alone PowerShell script requires no external libraries and allows you to save the tightening result from one K-Ducer controller. Run multiple instances of this script to connect to multiple K-Ducers.
- *KDUCER_reprogram_via_modbus_python.py*
    - This script illustrates how to modify various program parameters on the K-Ducer via modbus
- *KDUCER_modbus_remote_screwdriving_control_example_python.py*
    - This script illustrates remote actuation and control of the screwdriver. It follows the same steps and logic outlined in the section below “example screwdriving cycle”.

Siemens PLCs

Please refer to:
https://kolver.com/us/blog-detail/siemens-s7-plc-integration-now-available
https://kolver.com/upl/Siemens%20S7%20library%20and%20example.zip
This library has been developed and tested by Kolver for use with S7 series PLCs.

Allen Bradley PLCs

We provide an example project, *KDUCER_modbus_example_Micro820*, built and tested for a Micro820 PLC using Connected Components Workbench.
The project illustrates K-DUCER screwdriver control and data retrieval and follows the steps outlined in the section below "example screwdriving cycle".

Additionally, the following Rockwell-produced application guides and libraries are provided:

- 101037 Modbus TCP Add-On instructions for ControlLogix and CompactLogix controllers, AOI Version 2.04.00 (https://www.rockwellautomation.com/resources/downloads/samplecode/101037.exe)
- at002 AllenBradley EtherNetIP Socket Interface

More updated versions of these AOIs and guides may be found at https://www.rockwellautomation.com/search/ by searching for "modbus tcp" ("sample code" tab)

Universal Robots

We have a URCap: https://kolver.com/us/blog-detail/universal-robots-kolver-introducing-our-new-urcap-plugin
https://kolver.com/upl/URCAP-KDUCER.zip

Example Screwdriving Cycle

The sample projects and programs provided by Kolver for integrating the K-DUCER with the corresponding devices using MODBUS TCP all implement the screwdriving cycle illustrated here.

WARNING: running any of the attached examples with a properly configured K-DUCER will cause the screwdriver to run!
Do not run these examples if you are not familiarized with operating the screwdriver. Use all appropriate precautions and read the K-DUCER operator manual first.

The screwdriving cycle implemented in the examples loops through the following operations:

Step number	What it does	How it's done
1	Runs the screwdriver until it stops automatically	Writes "high" to COIL address 33 (REMOTE_LEVER) continuously every 50-100mS. Reads INPUT REGISTER address 138 to obtain the current screwdriving state and determine when the cycle is complete
2	Reads closing torque and angle	Reads INPUT REGISTER addresses 149 and 151 to obtain the closing torque and angles
3	Selects program #1 if the screw result was OK, program #2 if the screw result was NOK	Writes "1" or "2" to HOLDING REGISTER address 7373 to select the next program to run. Selection is determined by comparing the last screwdriving state to values 13/14 (screw OK/angle OK).