MODBUS registers Input and Output handling via port 502.|

MODBUS registers Input and Output handling via port 502

Application Description:
This example shows the use of internal MODBUS Input-Output registers on the Universal-Robots.

The application uses the MODBUS server running at port 502 on the Universal-Robots. The MODBUS server has a range of register available with dedicated content and register 128 to 255 are for general purpose use.
In this example register 0 (Inputs) and register 1 (Outputs) will be used.

A list of the MODBUS and register can be found at this link UR Modbus page

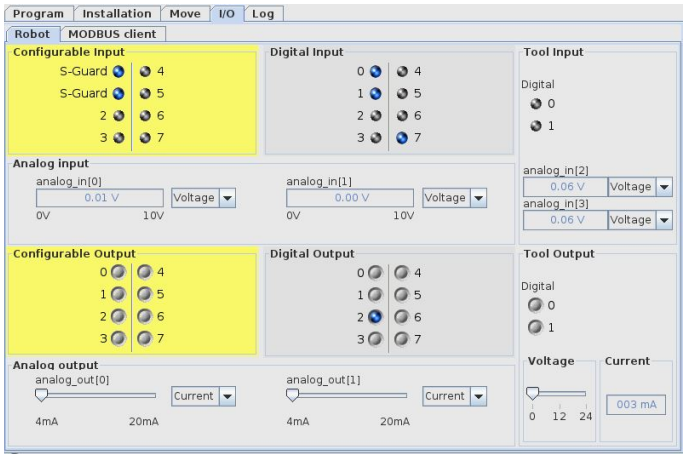
The commands are send from a external host with Python code.

Function description:
The external host (PC) is connected to the UR via Ethernet and access the MODBUS registers on the UR on TCP port 502.

Program description:
The external host first reads the physical hardware inputs and outputs on the UR and displays the results on the host screen.
Then the host sets all outputs on the UR to 0 and wait 1 second.
Then the host set output number 2 (bit 3 i.e. value 04 = 0100) to high.

Program code:
There is no program on UR. Instead the inputs and outputs are set to a random value in order to test the result on the host.

In this case Input 0, 1 and 7 is set to high. And out put 2 is set to high. Rest is set to zero.



Host program in Python code.

```
import socket
import time
import binascii

HOST = "192.168.0.9"            # The Robot IP address
PORT_502 = 502                  # The MODBUS Port on the robot

print "Starting Program"
count = 0
program_run = 0

while (True):
    if 1 == 1:
        if program_run == 0:
            try:
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s.settimeout(10)
                s.connect((HOST, PORT_502))
                time.sleep(0.05)
                print ""
                print "Modbus Read Inputs"
                print "Sending: x00 x01 x00 x00 x00 x06 x00 x03 x00 x00 x00 x01"
                s.send ("\x00\x01\x00\x00\x00\x06\x00\x03\x00\x00\x00\x01")

                msg = s.recv(1024)
                print "Raw received data = ", msg
                print "Received data as x format = ", repr(msg)         #Print the receive data in \x hex format (notice that data above 32 hex will be represented by the ascii code e.g. 50 will show P
                print ""
                msg = msg.encode("hex") #Convert the data from \x hex format to plain hex
                print "Hex data received = ", msg

                print "Strip downto last two bytes"
                byte_1_in = msg[21]
                byte_2_in = msg[20]
                print "byte_1_in = ", byte_1_in
                print "byte_2_in = ", byte_2_in

                byte_1_bin_in = bin(int(byte_1_in, 16))     #convert hex to bin
                print "Binary value of input byte 1 = ", byte_1_bin_in
                byte_1_bin_in = byte_1_bin_in[2:].zfill(4)  #get rid of the 0b and fill with zeros so byte take 4 positions
                print "Remove 0b and make byte occupie 4 positions = ", byte_1_bin_in
                byte_2_bin_in = bin(int(byte_2_in, 16))     #convert hex to bin
                print "Binary value of input byte 2 = ", byte_2_bin_in
                byte_2_bin_in = byte_2_bin_in[2:].zfill(4)  #get rid of the 0b and fill with zeros so byte take 4 positions
                print "Remove 0b and make byte occupie 4 positions = ", byte_2_bin_in
                print ""
                print "Input 0 = ", byte_1_bin_in[3]
                print "Input 1 = ", byte_1_bin_in[2]
                print "Input 2 = ", byte_1_bin_in[1]
                print "Input 3 = ", byte_1_bin_in[0]
                print "Input 4 = ", byte_2_bin_in[3]
                print "Input 5 = ", byte_2_bin_in[2]
                print "Input 6 = ", byte_2_bin_in[1]
                print "Input 7 = ", byte_2_bin_in[0]


                print ""
                print "Modbus Read Outputs"
                print "Sending: x00 x01 x00 x00 x00 x06 x00 x03 x00 x01 x00 x01"
                s.send ("\x00\x01\x00\x00\x00\x06\x00\x03\x00\x01\x00\x01")

                msg = s.recv(1024)
                print "Raw received data = ", msg
                print "Received data as x format = ", repr(msg)         #Print the receive data in \x hex format (notice that data above 32 hex will be represented by the ascii code e.g. 50 will show P
                print ""
                msg = msg.encode("hex") #Convert the data from \x hex format to plain hex
                print "Hex data received = ", msg

                print "Strip downto last two bytes"
```

```
                byte_1_out = msg[21]
                byte_2_out = msg[20]
                print "byte_1_out = ", byte_1_out
                print "byte_2_out = ", byte_2_out

                byte_1_bin_out = bin(int(byte_1_out, 16))       #convert hex to bin
                print "Binary value of Output byte 1 = ", byte_1_bin_out
                byte_1_bin_out = byte_1_bin_out[2:].zfill(4)    #get rid of the 0b and fill with zeros so byte take 4 positions
                print "Remove 0b and make byte occupie 4 positions = ",byte_1_bin_out
                byte_2_bin_out = bin(int(byte_2_out, 16))       #convert hex to bin
                print "Binary value of Output byte 2 = ", byte_2_bin_out
                byte_2_bin_out = byte_2_bin_out[2:].zfill(4)    #get rid of the 0b and fill with zeros so byte take 4 positions
                print "Remove 0b and make output byte occupie 4 positions = ", byte_2_bin_out
                print ""
                print "Output 0 = ", byte_1_bin_out[3]
                print "Output 1 = ", byte_1_bin_out[2]
                print "Output 2 = ", byte_1_bin_out[1]
                print "Output 3 = ", byte_1_bin_out[0]
                print "Output 4 = ", byte_2_bin_out[3]
                print "Output 5 = ", byte_2_bin_out[2]
                print "Output 6 = ", byte_2_bin_out[1]
                print "Output 7 = ", byte_2_bin_out[0]


                time.sleep(10.00)                               #Give time to read the read result before change output

                print ""
                print "Modbus set Outputs to 00"
                print "Sending: x00 x01 x00 x00 x00 x06 x00 x06 x00 x01 x00 x00"
                s.send ("\x00\x01\x00\x00\x00\x06\x00\x06\x00\x01\x00\x00")
                time.sleep(1.00)
                print ""
                print "Modbus set Outputs to 04 i.e. set output 2 on (third bit)"
                print "Sending: x00 x01 x00 x00 x00 x06 x00 x06 x00 x01 x00 x04"
                s.send ("\x00\x01\x00\x00\x00\x06\x00\x06\x00\x01\x00\x04")
                time.sleep(1.00)
                s.close()
                program_run = 0
            except socket.error as socketerror:
                print("Error: ", socketerror)
print "Program finish"
```

Program run:

```
>>>
Starting Program

Modbus Read inputs
Sending: x00 x01 x00 x00 x00 x06 x00 x03 x00 x00 x00 x01
Raw received data = ####
Received data as x format =  '\x00\x01\x00\x00\x00\x05\x00\x03\x02\x00\x83'

Hex data received =  00010000000500030200083
Strip downto last two bytes
byte_1_in =  3
byte_2_in =  8
Binary value of input byte 1 =  0b11
Remove 0b and make byte occupie 4 positions =  0011
Binary value of input byte 2 =  0b1000
Remove 0b and make byte occupie 4 positions =  1000

Input 0 =  1
Input 1 =  1
Input 2 =  0
Input 3 =  0
Input 4 =  0
Input 5 =  0
Input 6 =  0
Input 7 =  1

Modbus Read Outputs
Sending: x00 x01 x00 x00 x00 x06 x00 x03 x00 x01 x00 x01
Raw received data = ####
Received data as x format =  '\x00\x01\x00\x00\x00\x05\x00\x03\x02\x00\x04'

Hex data received =  00010000000500030200004
Strip downto last two bytes
byte_1_out =  4
byte_2_out =  0
Binary value of Output byte 1 =  0b100
Remove 0b and make byte occupie 4 positions =  0100
Binary value of Output byte 2 =  0b0
Remove 0b and make output byte occupie 4 positions =  0000

Output 0 =  0
Output 1 =  0
Output 2 =  1
Output 3 =  0
Output 4 =  0
Output 5 =  0
Output 6 =  0
Output 7 =  0

Modbus set Outputs to 00
Sending: x00 x01 x00 x00 x00 x06 x00 x06 x00 x01 x00 x00

Modbus set Outputs to 04 i.e. set output 2 on (third bit)
Sending: x00 x01 x00 x00 x00 x06 x00 x06 x00 x01 x00 x04
```

Notes:
Data exchange using the Python program with Modbus Data send to UR to read and write UR register.

First request is to read inputs.

Data send
\x00\x01\x00\x00\x00\x06\x00\x03\x00\x00\x00\x01

Which is the same as
00 01 00 00 00 06 00 03 00 00 00 01

Meaning:
00 01 is the sequence number Modbus TCP follows (Sequence no = Transaction identifier (serial no)
00 00 is protocol indentifier
00 06 messages length – means 6 bytes will follow
00 Unit identifier – or slave address
03 is the function code for read
00 00 The address of the first register requested (register 00hex = 0 dec which are inputs).
00 01 The total number of register to read

Data received from UR:

(Notice the data received from the UR is in hex format which might have values below 32 which maybe does not display on the screen depending on the data value because such data below 32 can be control characters. Therefore the data is formatted by the python program to readable data).

First readable poll received: 00010000000500030200083

Meaning:
00 01 is the sequence number that Modbus TCP follow (same as in the request send i.e. replied to)
00 00 Protocol identifier
00 05 messages length – means 5 bytes will follow
03 was the function code requested
02 registers are read

00 83 is data value received. (83 is the interesting data where 3 belongs to first byte of inputs and 8 belongs to second byte of inputs).

Value 3 = 0011 for first byte which means bit 0 and bit 1 has the value "high".

Value 8 = 1000 for second byte which means bit 7 has the value "high".

This is correct according to physical current state of the inputs as shown above in the I/O window on the UR.

Then the python program strips the hex values down to bit value for each inputs and present on the screen.

Input 0 = 1
Input 1 = 1
Input 2 = 0
Input 3 = 0
Input 4 = 0
Input 5 = 0
Input 6 = 0
Input 7 = 1

Second request is to read outputs.

Data send
\x00\x01\x00\x00\x00\x06\x00\x03\x00\x01\x00\x01

Which is the same as
00 01 00 00 00 06 00 03 00 01 00 01

Meaning:
00 01 is the sequence number Modbus TCP follows (Sequence no = Transaction identifier (serial no)
00 00 is protocol indentifier
00 06 messages length – means 6 bytes will follow
00 Unit identifier – or slave address
03 is the function code for read
00 01 The address of the second register requested (register 01hex = 1 dec which are outputs).
00 01 The total number of register to read

Data received from UR:

(Notice the data received from the UR is in hex format which might have values below 32 which maybe does not display on the screen depending on the data value because such data below 32 can be control characters. Therefore the data is formatted by the python program to readable data).

Second readable poll received: 00010000000500030200 04

Meaning:
00 01 is the sequence number that Modbus TCP follow (same as in the request send i.e. replied to)
00 00 Protocol identifier
00 05 messages length – means 5 bytes will follow
03 was the function code requested
02 registers are read
00 04 is data value received. (04 is the interesting data where 4 belongs to first byte of outputs and 0 belongs to second byte of outputs).

Value 4 = 0100 for first byte which means bit 2 has the value "high".

Value 0 = 0000 for second byte which means all bits has the value "low".

This is correct according to physical current state of the inputs as shown above in the I/O window on the UR.

Then the python program strips the hex values down to bit value for each inputs and present on the screen.

Output 0 = 0
Output 1 = 0
Output 2 = 1
Output 3 = 0
Output 4 = 0
Output 5 = 0
Output 6 = 0
Output 7 = 0

First request to write to set output.

Data send.
\x00\x01\x00\x00\x00\x06\x00\x06\x00\x01\x00\x00

Which is the same as
00 01 00 00 00 06 00 06 00 01 00 00

Meaning:
00 01 is the sequence number Modbus TCP follows (Sequence no = Transaction identifier (serial no)
00 00 is protocol indentifier
00 06 messages length – means 6 bytes will follow
00 Unit identifier – or slave address
06 is the function code for write
00 01 The data address of the first register to set (register 01hex = 1 dec).
00 00 The value to set into the register (set value to 0 – this sets all outputs low)

Then the program wait 1 second.

Second request to write.

Data send.
\x00\x01\x00\x00\x00\x06\x00\x06\x00\x01\x00\x04

Which is the same as
00 01 00 00 00 06 00 06 00 01 00 04

Meaning:
00 01 is the sequence number Modbus TCP follows (Sequence no = Transaction identifier (serial no)
00 00 is protocol indentifier
00 06 messages length – means 6 bytes will follow
00 Unit identifier – or slave address
06 is the function code for write
00 01 The data address of the first register to set (register 01hex = 1 dec).
00 04 The value to set into the register (set value to 4 – this sets output 3 high)

Author:
By Zacobria Lars Skovsgaard
Accredited Universal Robots support Centre and Forum.

Also check out the CB 3 forum