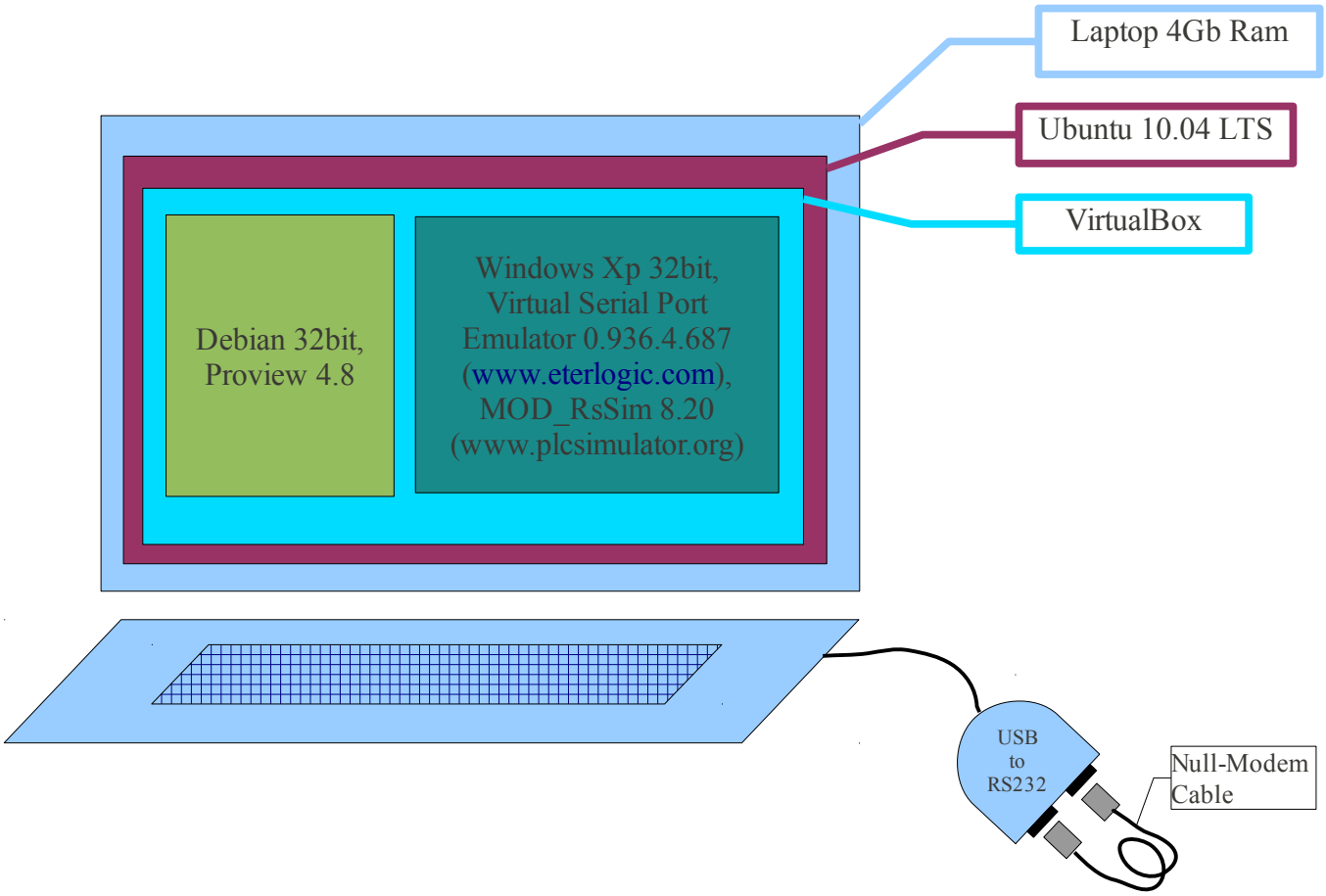
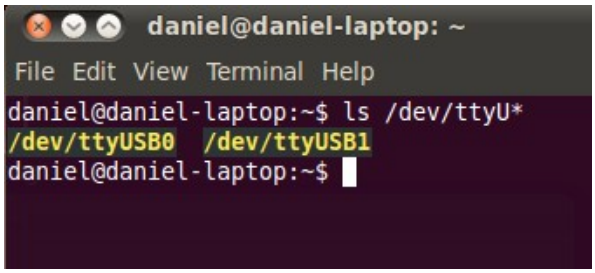
Modbus RTU Proview 4.8 tutorial. **PLEASE EXCUSE MY ENGLISH !!!**

What do I use??

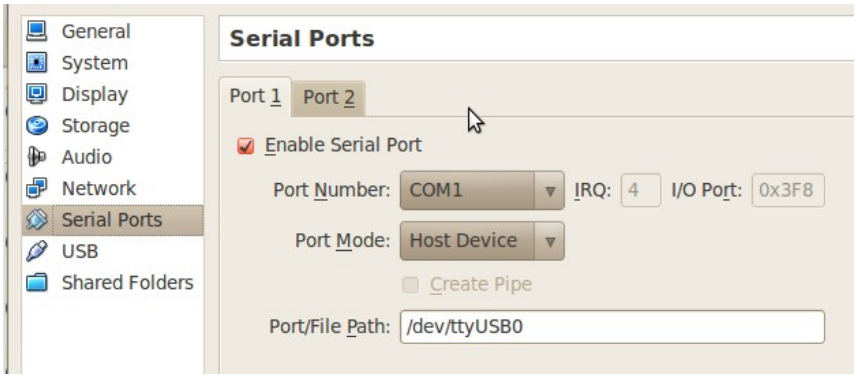


1. **Converter USB to RS232 settings:**

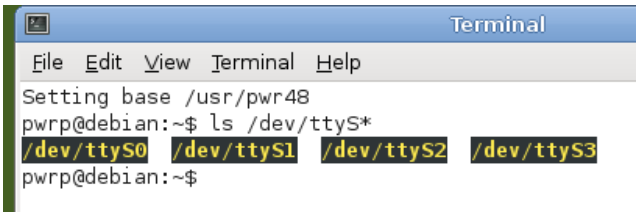
In **Ubuntu**
Serial COM1 → /dev/ttyUSB0
Serial COM2 → /dev/ttyUSB1



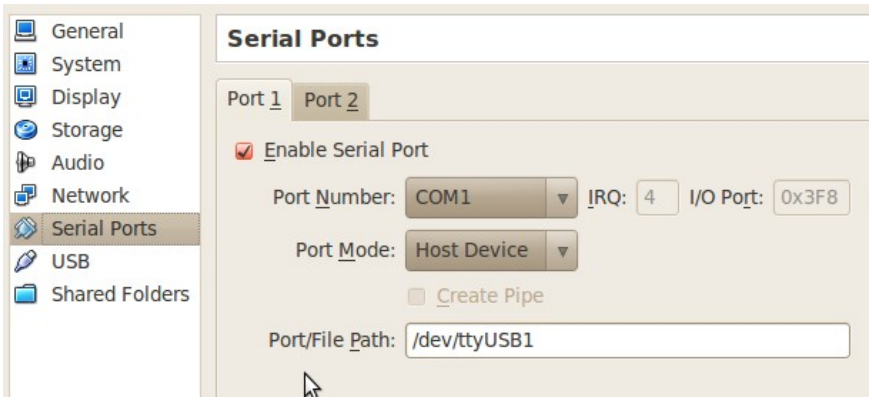
VirtualBox → Debian



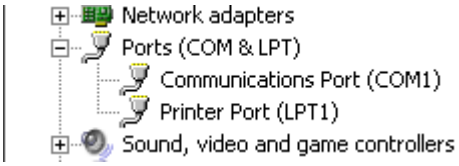
Debian → /dev/ttyS0



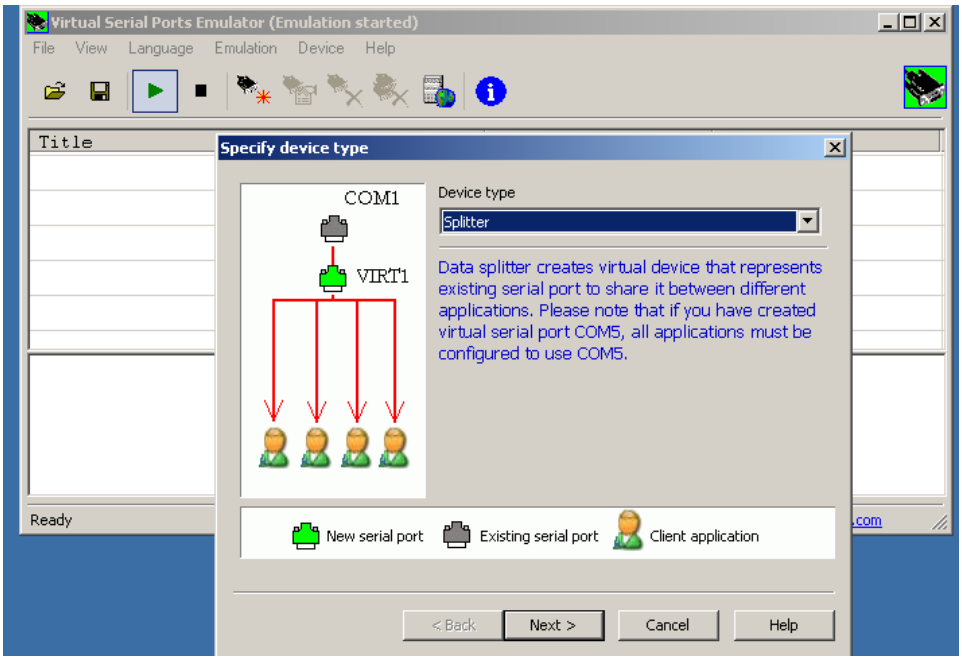
VirtualBox → Windows XP

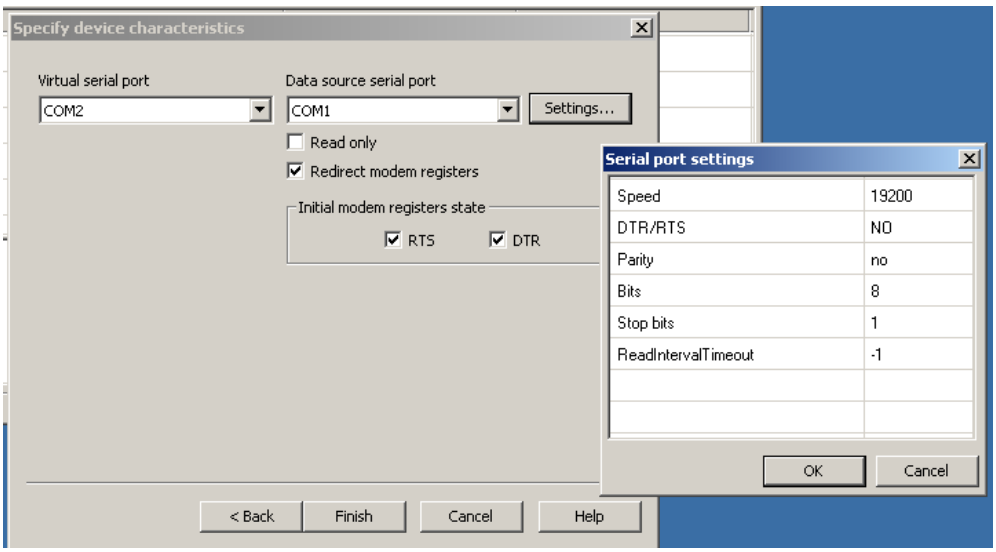


Windows XP → COM1



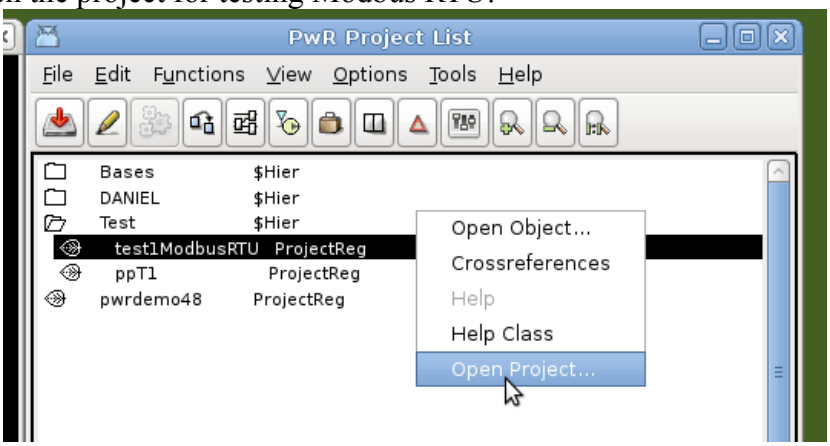
Windows XP → Emulare COM1



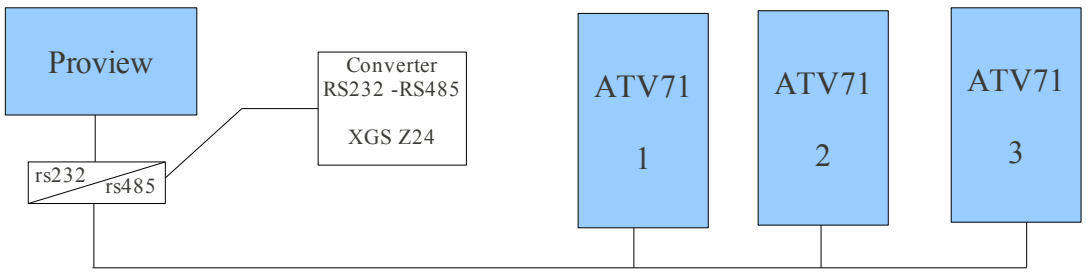


2. Proiect Proview

Create and then open the project for testing Modbus RTU:



Requirements project (the project are real data)
I want to read seven words, starting with the address 3202 (0x0C82, {43203 - modbus}) of the three converters ATV71-type (Schneider-Electric), using Modbus RTU serial protocol.

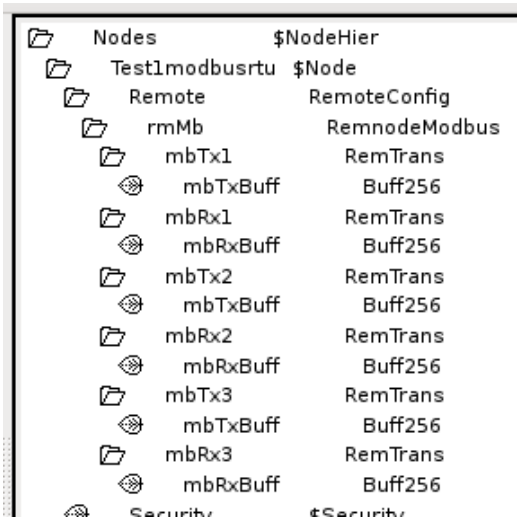


Now, let's see the project. First, please study the tutorial from the following link:
http://www.proview.se/index.php?option=com_joomlaboard&Itemid=24&func=view&id=268&catid=3#268
and chapter 12.2.6 from designer's guide manual.

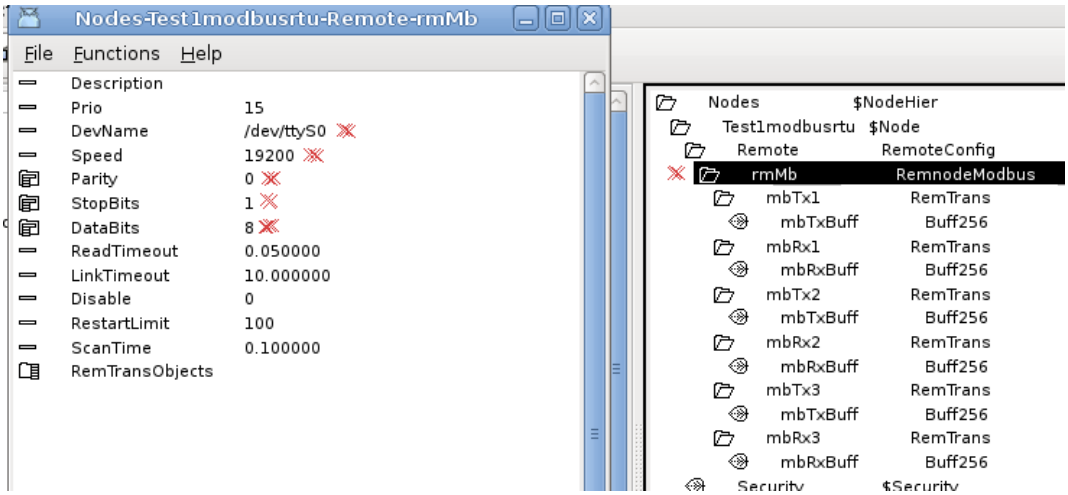
Proview

How to configure Modbus RTU:

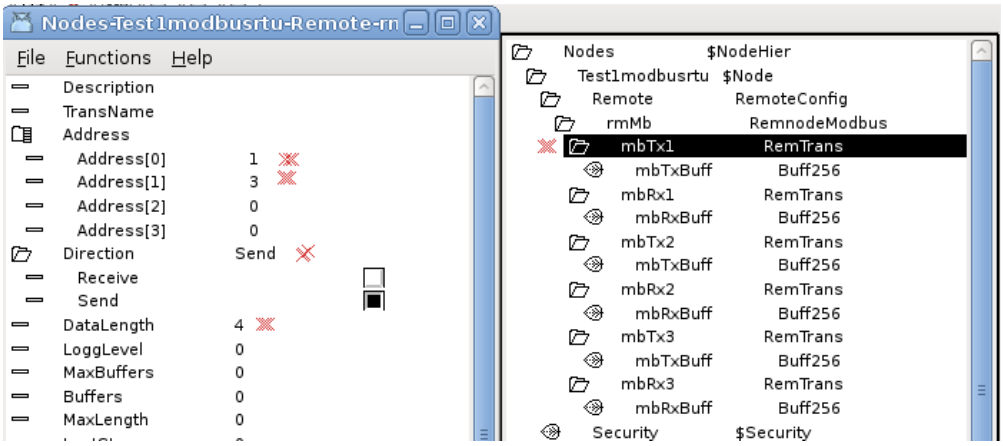
- Place a RemoteConfig in the node-hierarchy;



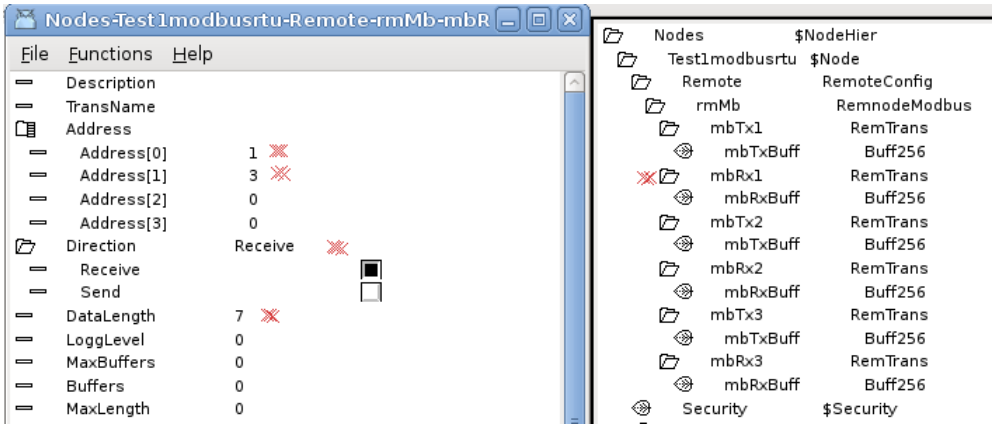
- Below the RemoteConfig-object you place a RemnodeModbus object for each communication link you will have in your system. Configure serial port as shown below;



- Below the RemnodeModbus-object you place a RemTrans-object;
  SENDING → mbTx {Address[0] = 1 (slave-address), Address[1] = 3 (modbus function code)}



RECEIVING → mbRx {Address[0] = 1 (slave-address), Address[1] = 3 (modbus function code)}



Please note that mbTx → Address[0] and mbRx → Address[0] must coincide. Same mbTx → Address[1] and mbRx → Address[1] !

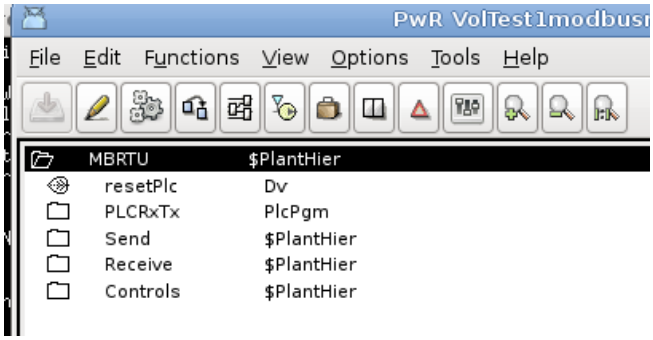
- The Buffers-objects are the data area for the message. Place a Buff256-object for each RemTrans-object.

Settings for RemTrans items:
mbTx1 → Address[0] = 1, Address[1] = 3, Direction = Send, DataLenght = 4;
mbRx1 → Address[0] = 1, Address[1] = 3, Direction = Receive, DataLenght = 10;
mbTx2 → Address[0] = 2, Address[1] = 3, Direction = Send, DataLenght = 4;
mbRx2 → Address[0] = 2, Address[1] = 3, Direction = Receive, DataLenght = 10;
mbTx3 → Address[0] = 3, Address[1] = 3, Direction = Send, DataLenght = 4;
mbRx3 → Address[0] = 3, Address[1] = 3, Direction = Receive, DataLenght = 10.

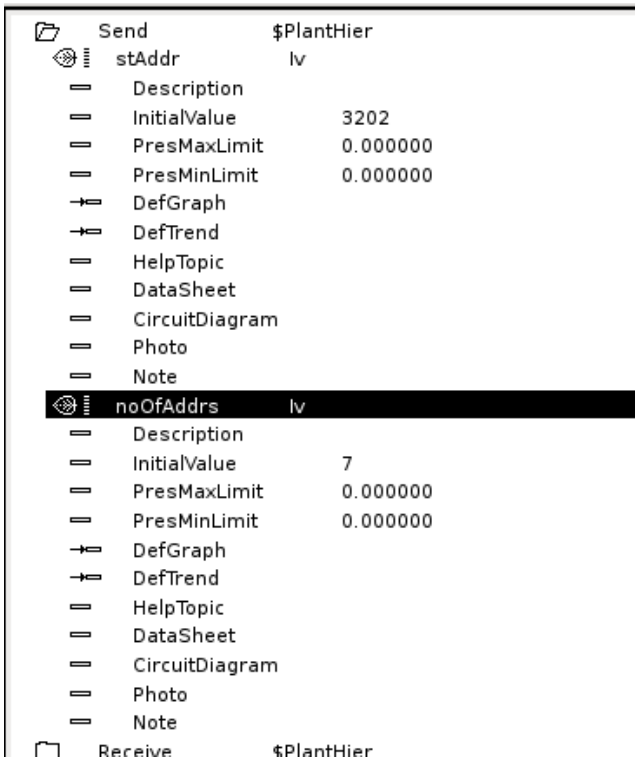
I notice that the DataLenght item for "Direction = Receive", can be set to any value !

Now that we have established communication parameters, let's have a look on the project variables.

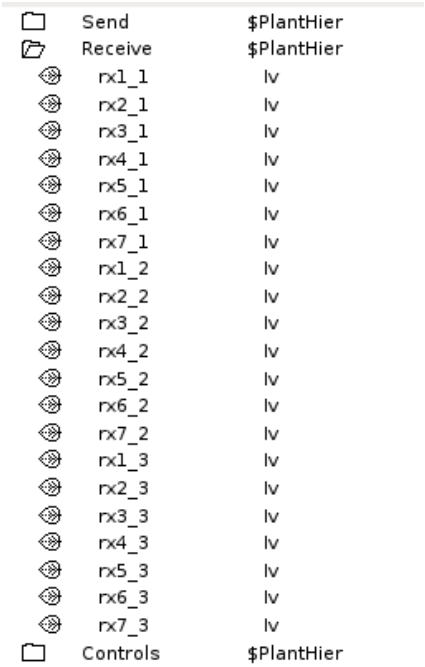
Project hierarchy:



Send → stAddr (the Data Address of the first register requested) → InitialValue = 3202;
→ noOfAddrs (the total number of registers requested) → InitialValue = 7.



Receive → rx(1..7)_(1..3)



Control communication → send, occ, err.

	Controls	$PlantHier
	sendDv1	Dv
	occDv1	Dv
	errDv1	Dv
	sendDv2	Dv
	occDv2	Dv
	errDv2	Dv
	sendDv3	Dv
	occDv3	Dv
	errDv3	Dv

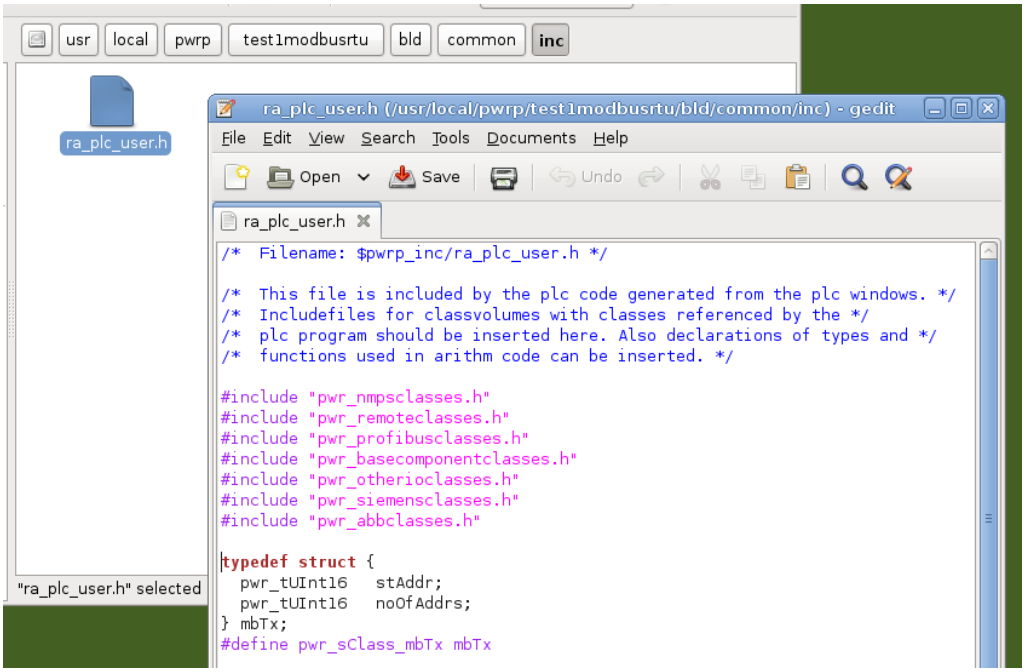
PLCRxTx → plc program.

First, let's see how the protocol works:

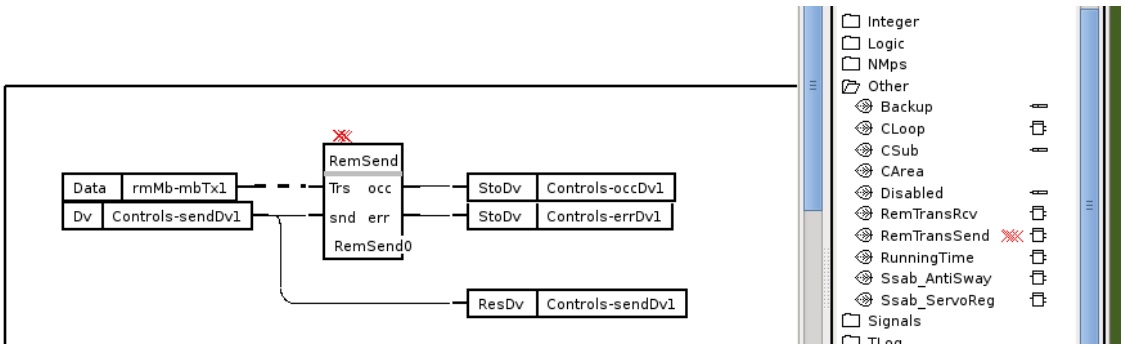
1. **Send (mbTx)**:

01	03	0C	82	00	07	A7	70
Address[0] (slave-address)	Address[1] (modbus function code)	stAddr (the Data Address of the first register requested)		noOfAddrs (the total number of registers requested)		CRC	

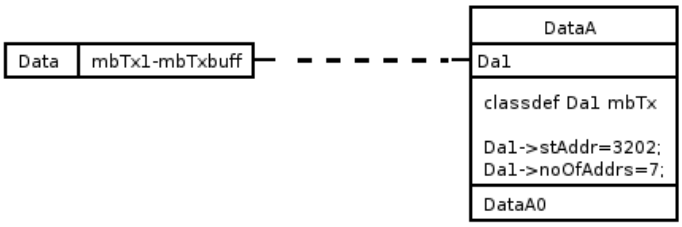
We need to create a data structure for the send messages. That will be defined in the file ra_plc_user.h in $pwrp_inc-directory. This file is automatically included when you compile the plc-code.



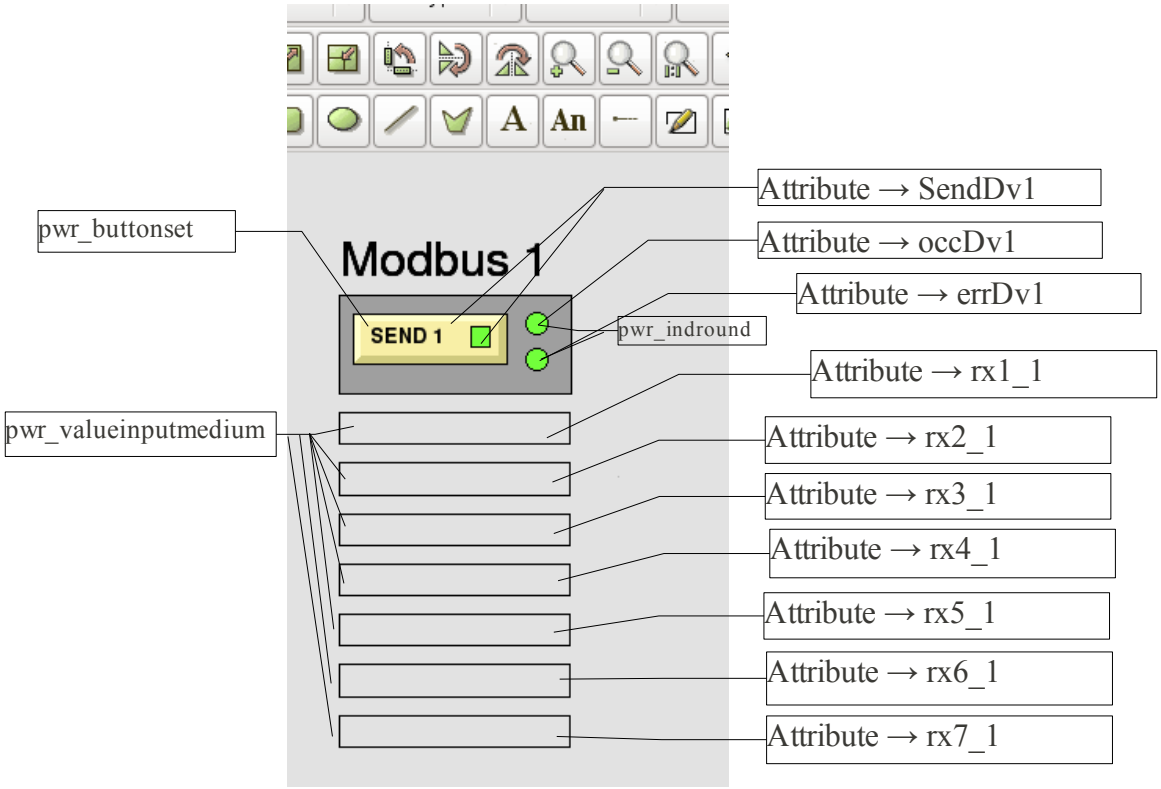
The plc code



RemTransSend must have a subwindow. The send-buffer for the message to send is connected to a DataArithm.



In the end , I created a XttGraph to test the serial communication.

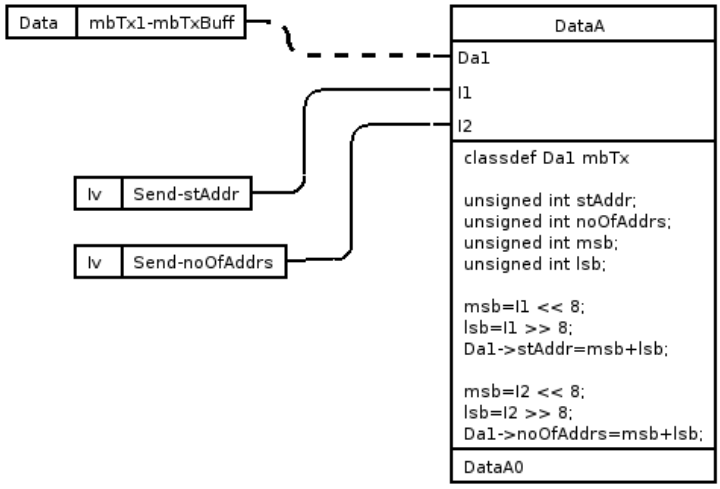


Now let's see what Proview send to the slaves.

01	03	82	0C	07	00	AF	81
Address[0] (slave-address)	Address[1] (modbus function code)	stAddr (the Data Address of the first register requested)		noOfAddrs (the total number of registers requested)		CRC	

Not good at all!
Note though that Modbus works with Big Endian, so you need to **byte-swap**.

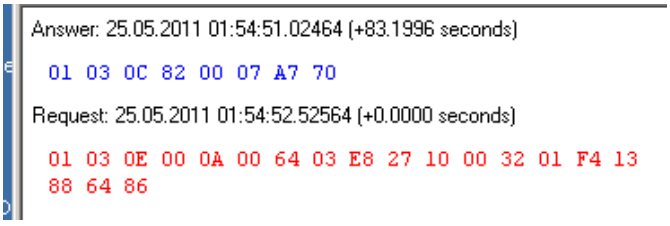
We need to correct the plc code.
The code from DataArithm-object will be:

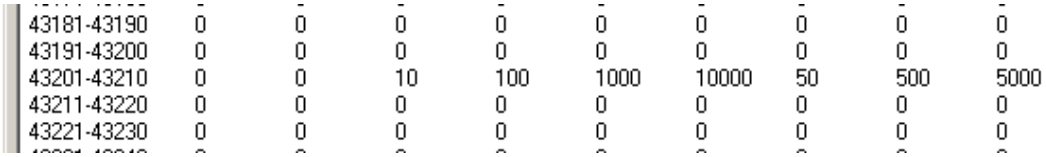


Now everything's fine.

2. Receive(mbRx):

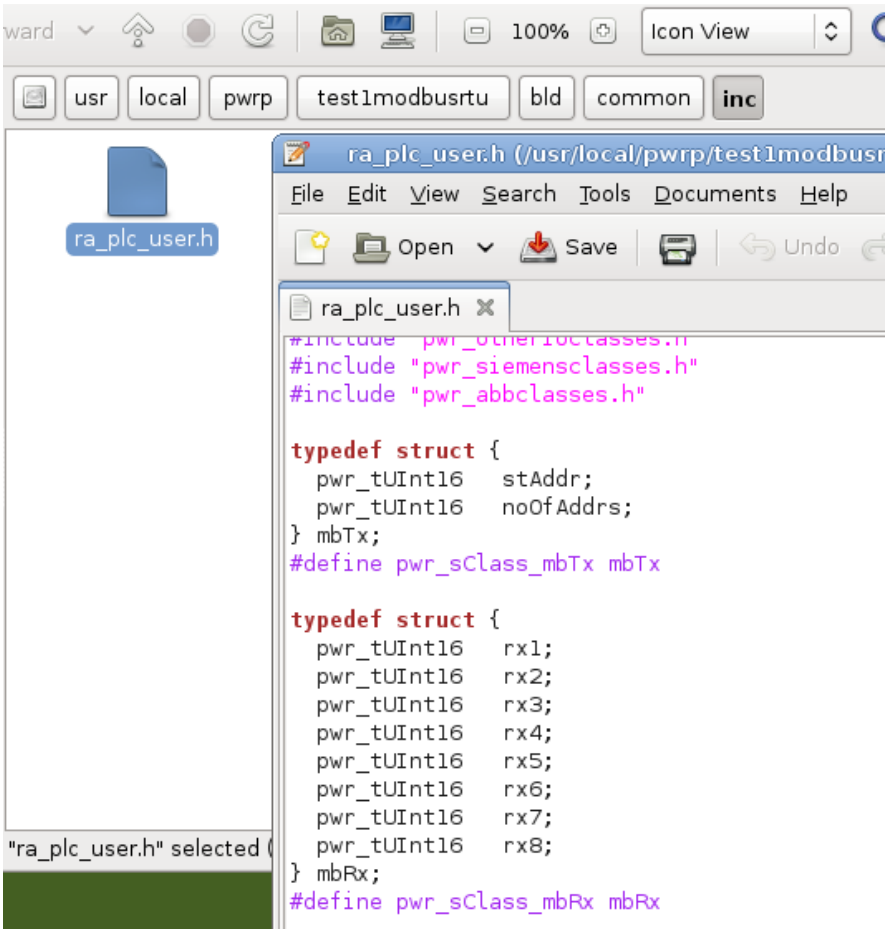
01	03	0E	00	0A	00	64	03	E8	27	10	00	32	01	F4	13	88	64	86
a	b	c	d		e		f		g		h		i		j		CRC	



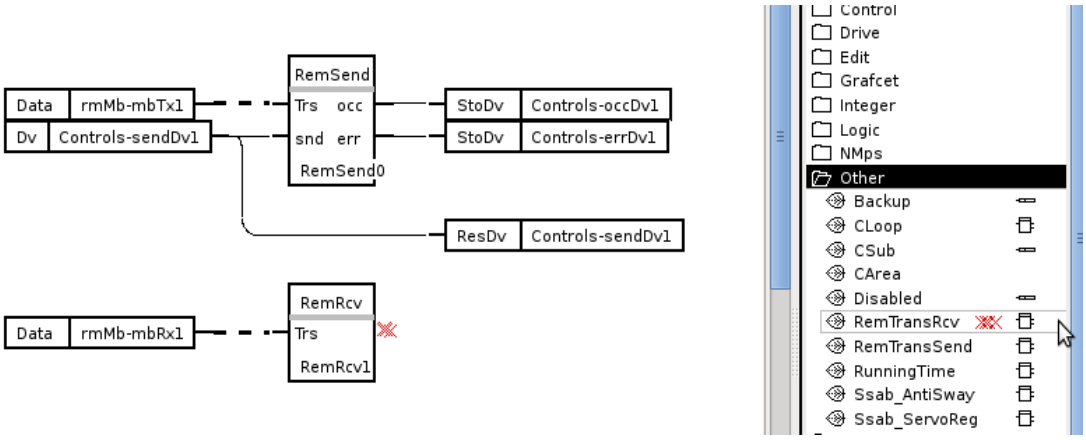


a- the slave address;
b- the function code;
c- the number of data bytes to follow (7 registers x 2 bytes each = 14 bytes);
d- the contents of register 0x0C82 (3202 in decimal, {43203 modbus});
e- the contents of register 0x0C83 (3203 in decimal, {43204 modbus});
f- the contents of register 0x0C84 (3204 in decimal, {43205 modbus});
g- the contents of register 0x0C85 (3205 in decimal, {43206 modbus});
h- the contents of register 0x0C86 (3206 in decimal, {43207 modbus});
i- the contents of register 0x0C87 (3207 in decimal, {43208 modbus});
j- the contents of register 0x0C88 (3208 in decimal, {43209 modbus});

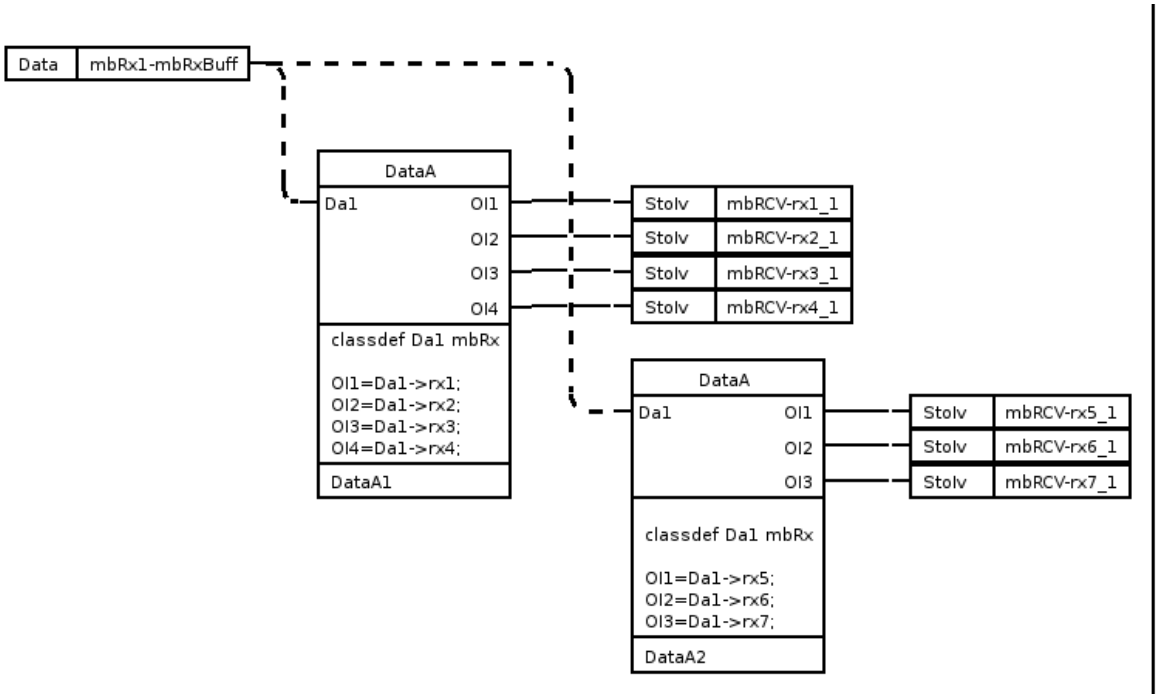
We need to create a data structure for the receive messages. That will be defined in the file ra_plc_user.h in $pwrp_inc-directory. This file is automatically included when you compile the plc-code.



The plc code:

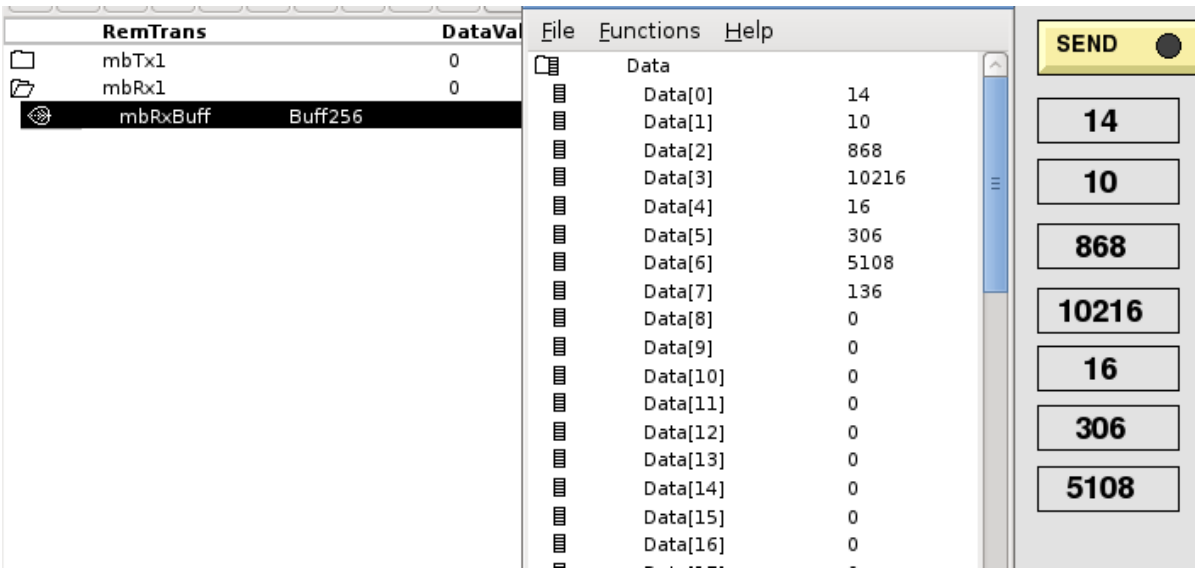


RemTransRcv must have a subwindow. The send-buffer for the message to send is connected to a DataArithm.



After I compile the application and test the communication, I received the following answer:

01	03	00	0E	00	0A	03	64	27	E8	00	10	01	32	13	F4	00	88
1	3	14		10		868		10216		16		306		5108		136	
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r

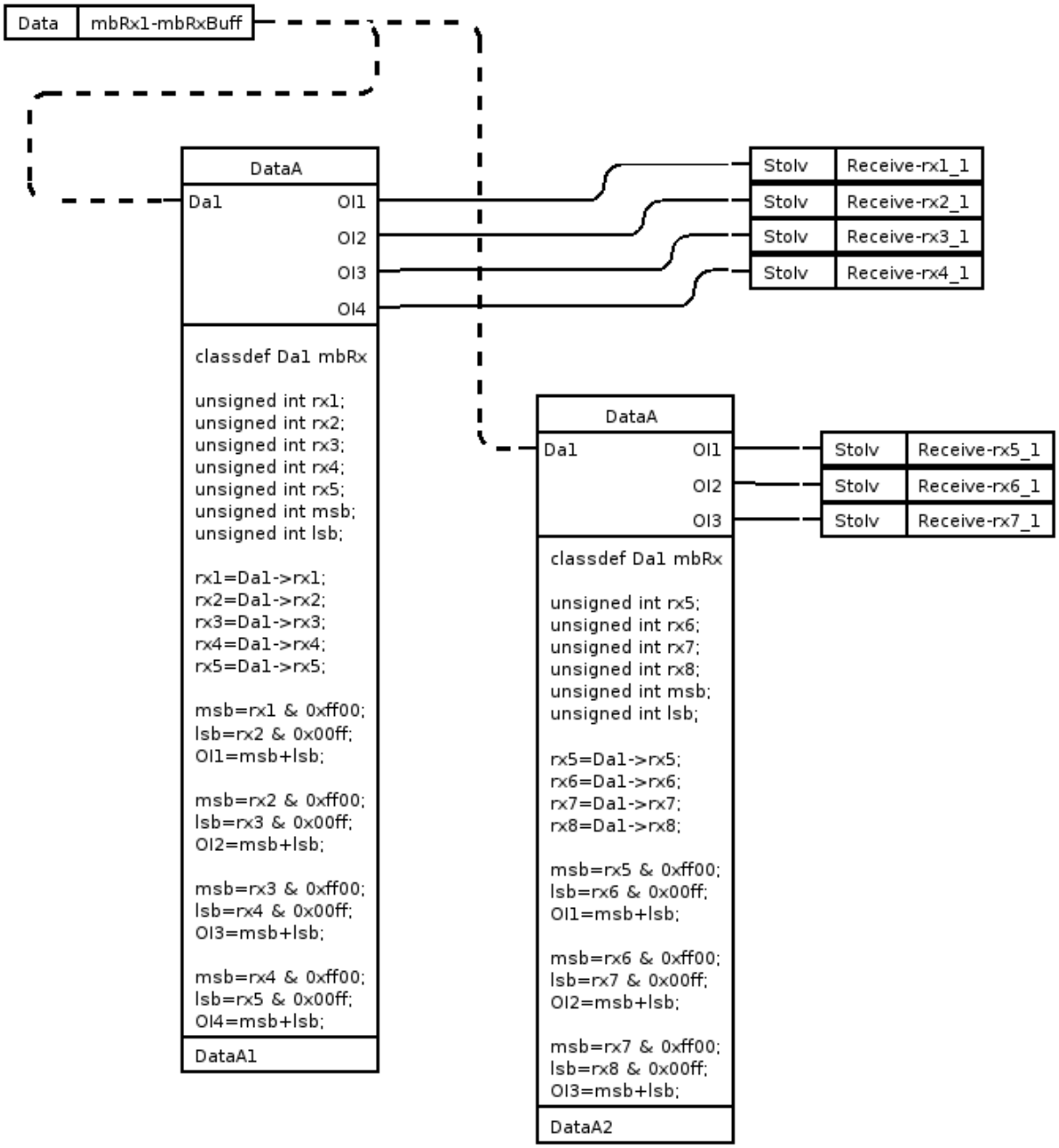


Now we have two problems. The byte-swap and the "d" (the number of data bytes) which is part of modbus protocol.

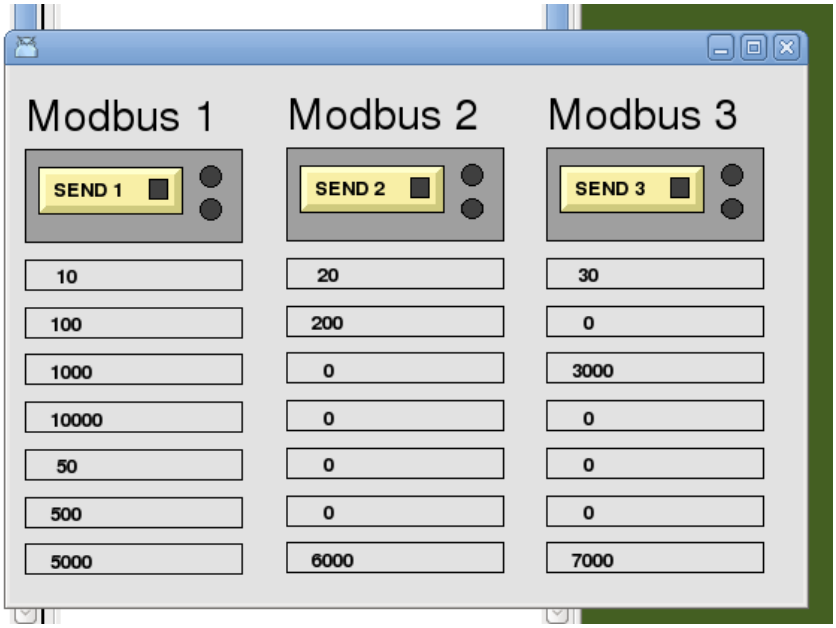
We need to correct the plc code.
The code from DataArithm-object will be:

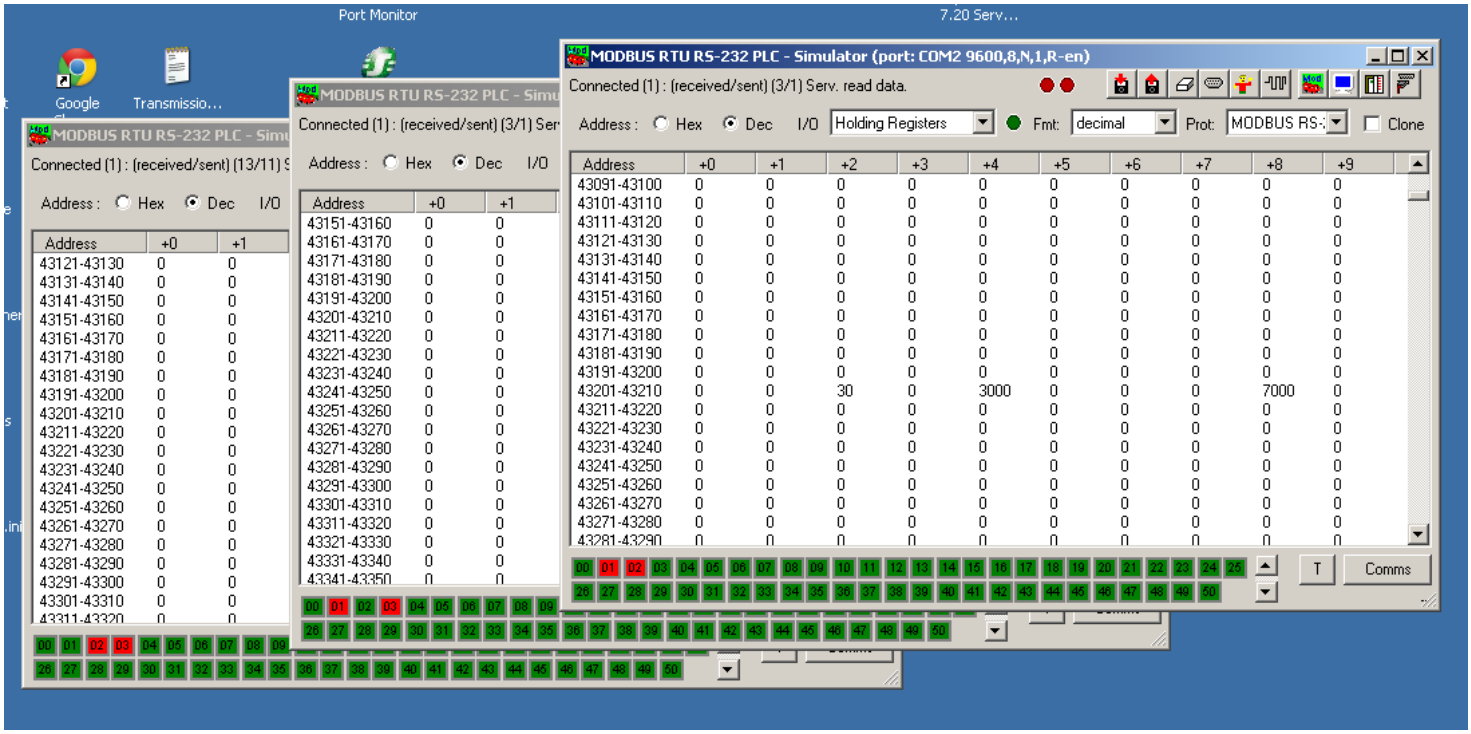
DataA1		DataA2	

classdef Da1 mbRx
 	
unsigned int rx1;
unsigned int rx2;
unsigned int rx3;
unsigned int rx4;
unsigned int rx5;
unsigned int msb;
unsigned int lsb;
 	
rx1=Da1->rx1;
rx2=Da1->rx2;
rx3=Da1->rx3;
rx4=Da1->rx4;
rx5=Da1->rx5;
 	
msb=rx1 & 0xff00;
lsb=rx2 & 0x00ff;
OI1=msb+lsb;
	msb=rx2 & 0xff00;
	lsb=rx3 & 0x00ff;
	OI2=msb+lsb;
	
	msb=rx3 & 0xff00;
	lsb=rx4 & 0x00ff;
	OI3=msb+lsb;
	
	msb=rx4 & 0xff00;
	lsb=rx5 & 0x00ff;
	OI4=msb+lsb;
		classdef Da1 mbRx
		
		unsigned int rx5;
		unsigned int rx6;
		unsigned int rx7;
		unsigned int rx8;
		unsigned int msb;
		unsigned int lsb;
		
		rx5=Da1->rx5;
		rx6=Da1->rx6;
		rx7=Da1->rx7;
		rx8=Da1->rx8;
		
		msb=rx5 & 0xff00;
		lsb=rx6 & 0x00ff;
		OI1=msb+lsb;
			msb=rx6 & 0xff00;
			lsb=rx7 & 0x00ff;
			OI2=msb+lsb;
			
			msb=rx7 & 0xff00;
			lsb=rx8 & 0x00ff;
			OI3=msb+lsb;



Finally, we will write code for the three inverters and we will see the result.





Everything works great.
Hope you enjoy this tutorial.

Daniel - 2011