Modbus TCP Master and Slave quickstart guide for

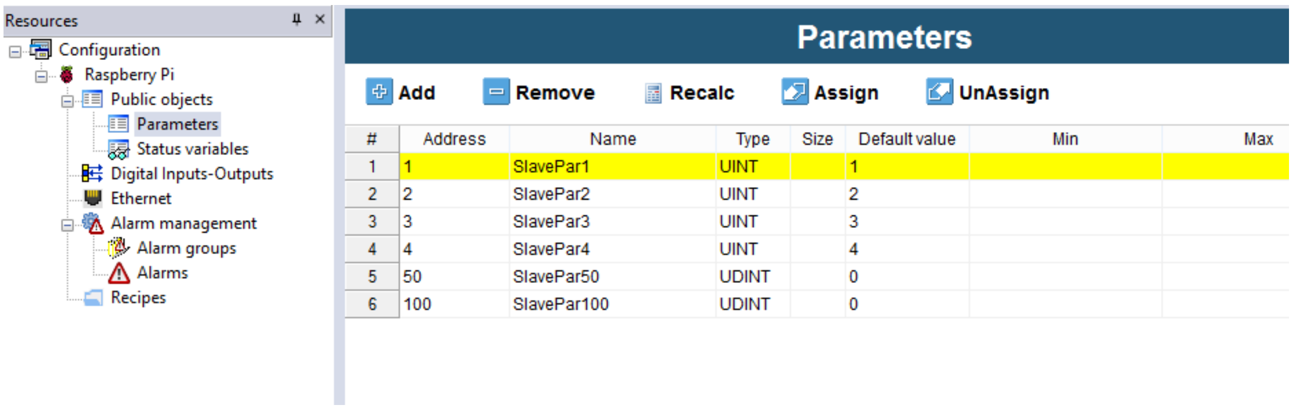
Axel Automation Suite

with RaspberryPi

RaspSlave – Modbus TCP slave project

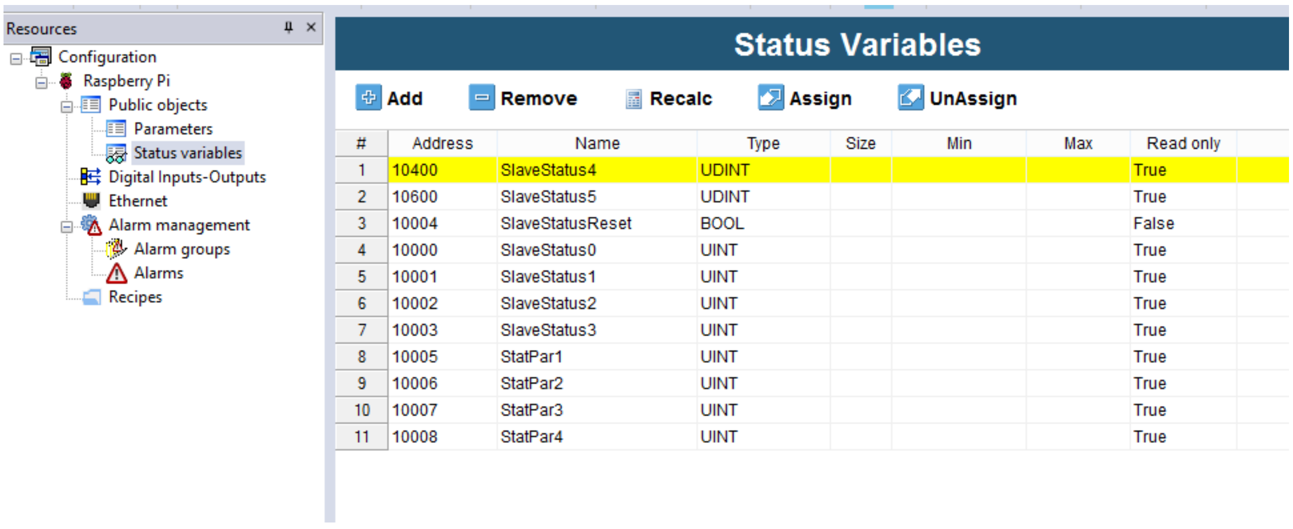
Database definition

First of all define the database Parameters (persistent)



#	Address	Name	Type	Size	Default value	Min	Max
1	1	SlavePar1	UINT		1		
2	2	SlavePar2	UINT		2		
3	3	SlavePar3	UINT		3		
4	4	SlavePar4	UINT		4		
5	50	SlavePar50	UDINT		0		
6	100	SlavePar100	UDINT		0		

And Status Variables (volatile)

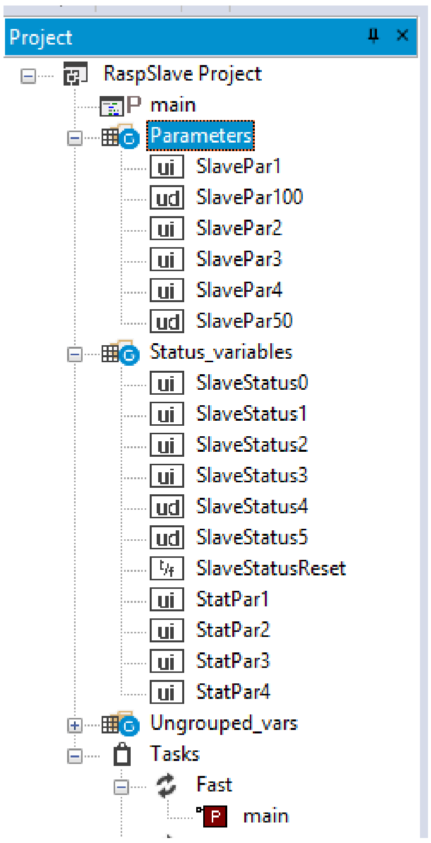


#	Address	Name	Type	Size	Min	Max	Read only
1	10400	SlaveStatus4	UDINT				True
2	10600	SlaveStatus5	UDINT				True
3	10004	SlaveStatusReset	BOOL				False
4	10000	SlaveStatus0	UINT				True
5	10001	SlaveStatus1	UINT				True
6	10002	SlaveStatus2	UINT				True
7	10003	SlaveStatus3	UINT				True
8	10005	StatPar1	UINT				True
9	10006	StatPar2	UINT				True
10	10007	StatPar3	UINT				True
11	10008	StatPar4	UINT				True

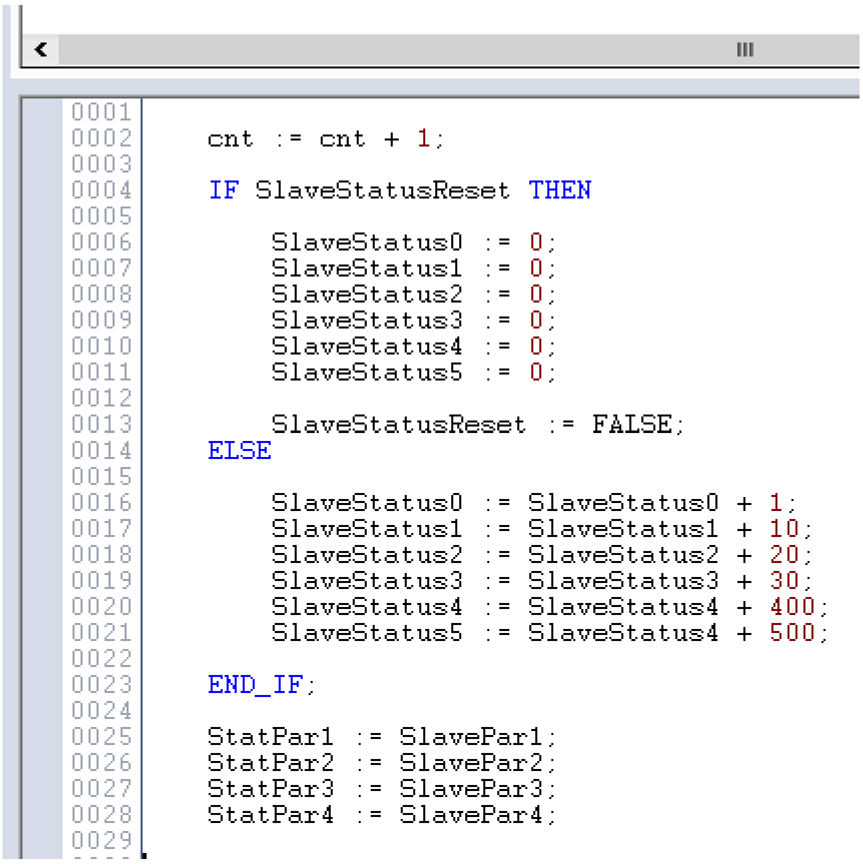
These public objects can be accessed via Modbus TCP specifying the address.

For each Public object a variable of the specified type is created.

Write the code

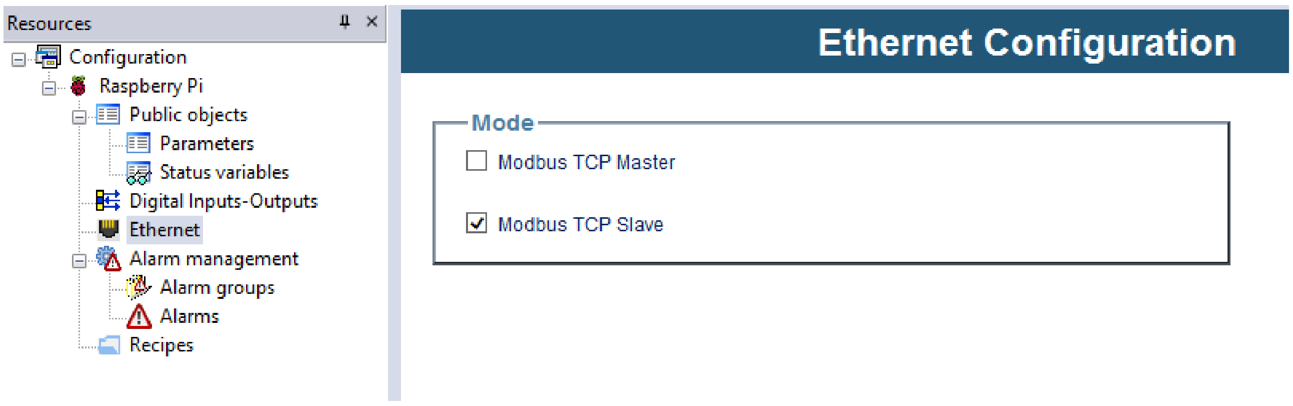


This variables can be used in the PLC project. This is the code associated to the main program in task Fast



```

    cnt := cnt + 1;

    IF SlaveStatusReset THEN

        SlaveStatus0 := 0;
        SlaveStatus1 := 0;
        SlaveStatus2 := 0;
        SlaveStatus3 := 0;
        SlaveStatus4 := 0;
        SlaveStatus5 := 0;

        SlaveStatusReset := FALSE;
    ELSE

        SlaveStatus0 := SlaveStatus0 + 1;
        SlaveStatus1 := SlaveStatus1 + 10;
        SlaveStatus2 := SlaveStatus2 + 20;
        SlaveStatus3 := SlaveStatus3 + 30;
        SlaveStatus4 := SlaveStatus4 + 400;
        SlaveStatus5 := SlaveStatus4 + 500;

    END_IF;

    StatPar1 := SlavePar1;
    StatPar2 := SlavePar2;
    StatPar3 := SlavePar3;
    StatPar4 := SlavePar4;

```

Enable Modbus TCP slave communication


Resources
- Configuration
  - Raspberry Pi
    - Public objects
      - Parameters
      - Status variables
    - Digital Inputs-Outputs
    - Ethernet
    - Alarm management
      - Alarm groups
      - Alarms
    - Recipes

Ethernet Configuration

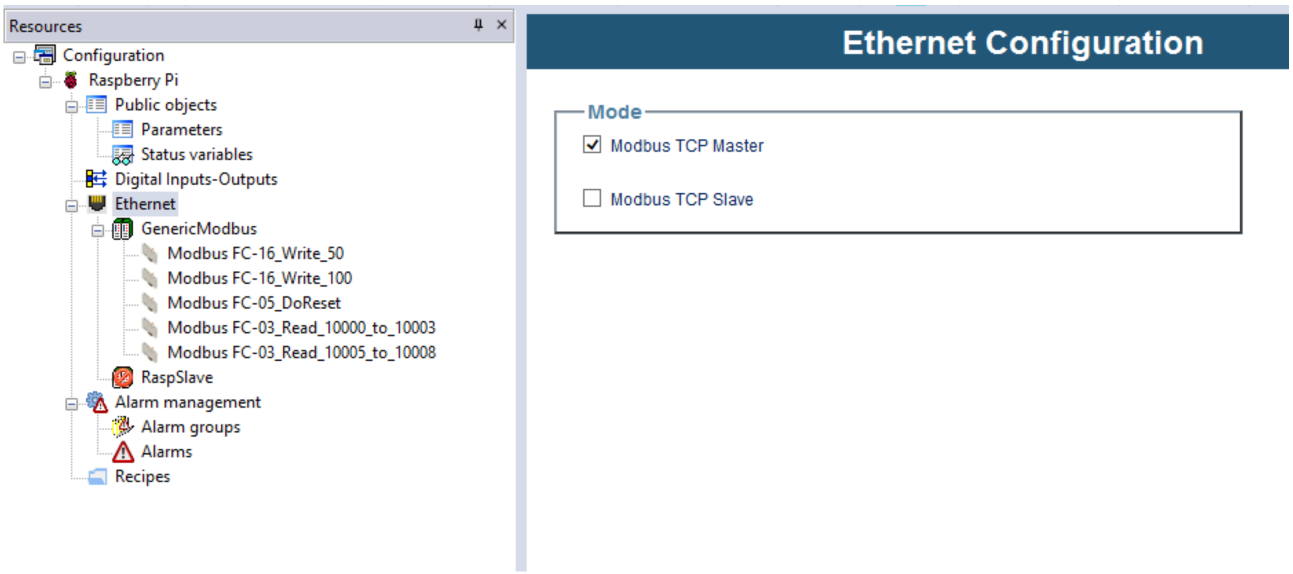
Mode
- [ ] Modbus TCP Master
- [✓] Modbus TCP Slave

Download the project into target

RaspMaster – Modbus TCP master project

Enable Modbus TCP master

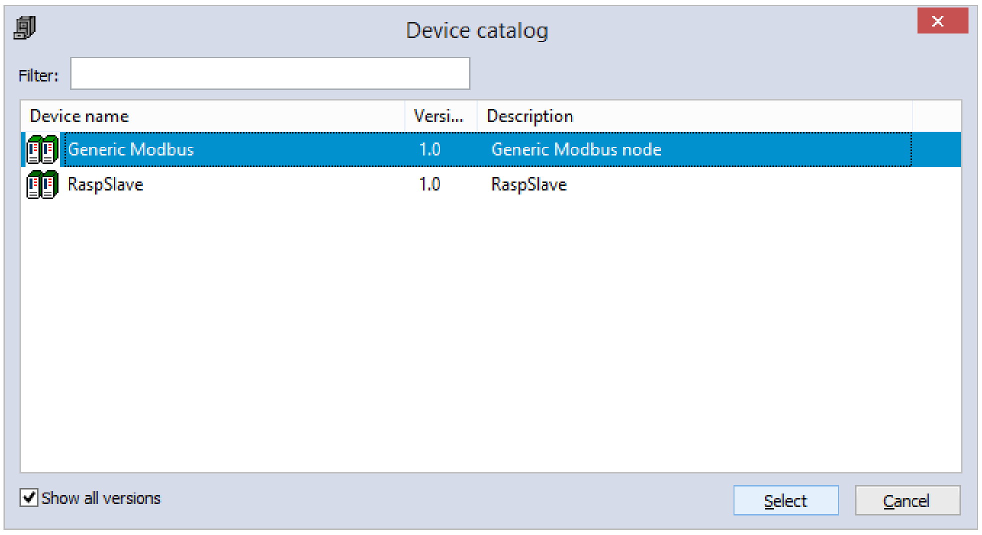
Select Ethernet


Resources
- Configuration
  - Raspberry Pi
    - Public objects
      - Parameters
      - Status variables
    - Digital Inputs-Outputs
    - Ethernet
      - GenericModbus
        - Modbus FC-16_Write_50
        - Modbus FC-16_Write_100
        - Modbus FC-05_DoReset
        - Modbus FC-03_Read_10000_to_10003
        - Modbus FC-03_Read_10005_to_10008
      - RaspSlave
    - Alarm management
      - Alarm groups
      - Alarms
    - Recipes

Ethernet Configuration

Mode
- [✓] Modbus TCP Master
- [ ] Modbus TCP Slave

Right click Add and select Generic Modbus (or Developer > View Catalog and Drag & Drop)


Device catalog

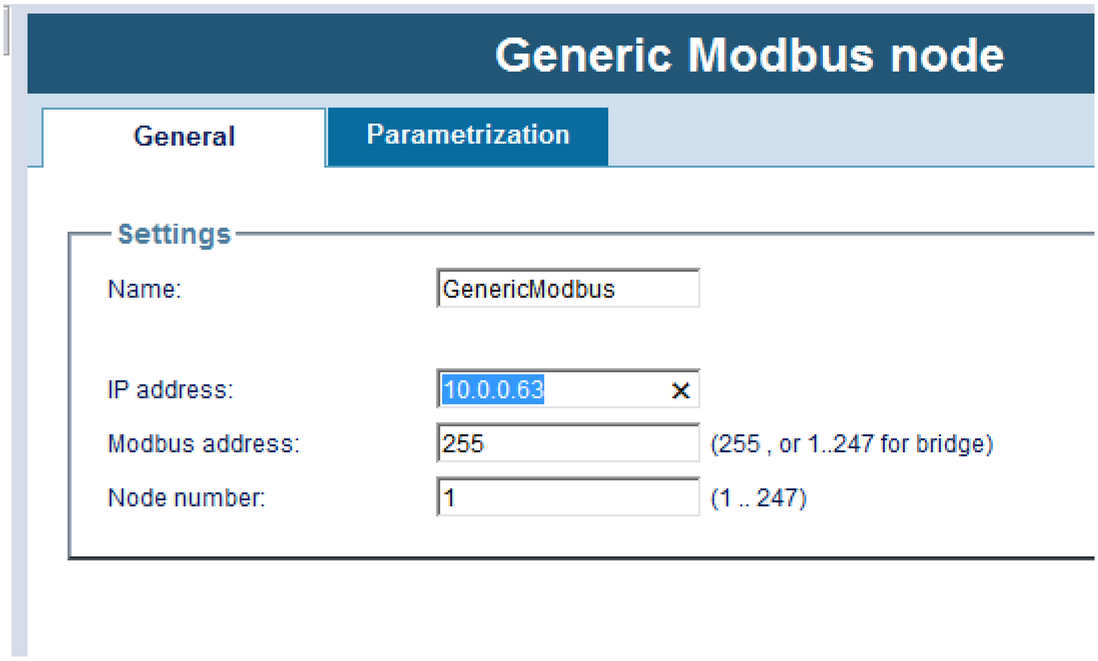
Filter:

Device name	Versi...	Description
Generic Modbus	1.0	Generic Modbus node
RaspSlave	1.0	RaspSlave

[✓] Show all versions

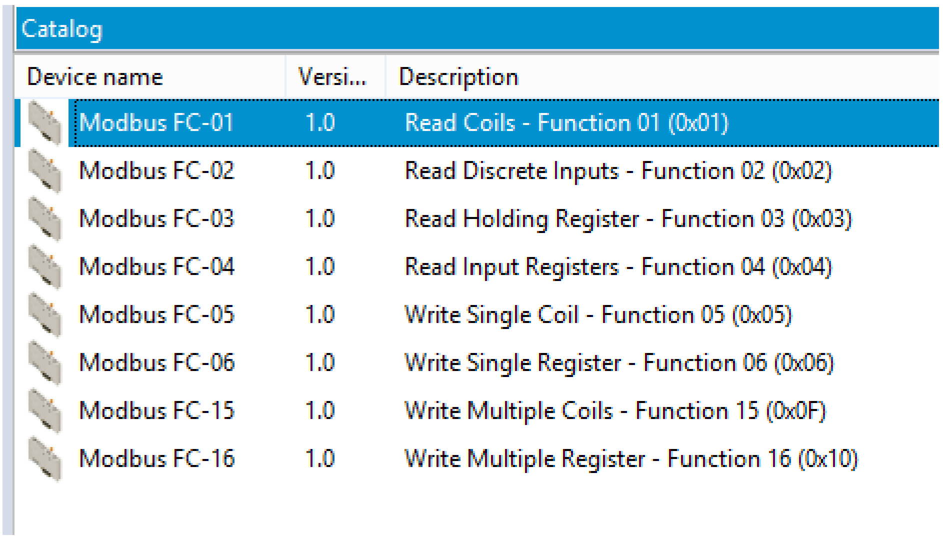
Generic Modbus

Specify the TCPIP address of the slave

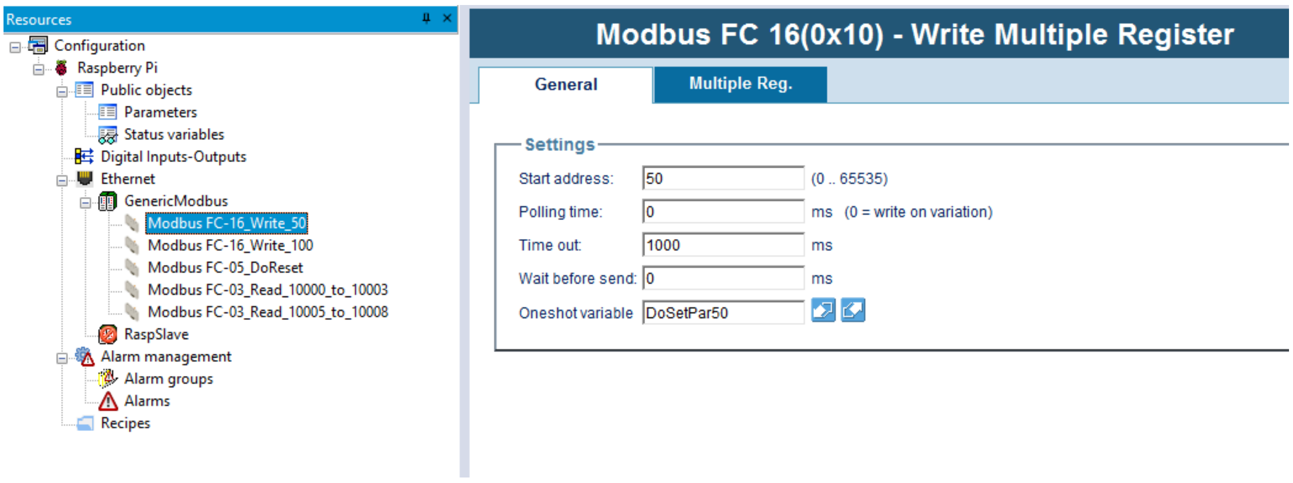


Name:	GenericModbus
IP address:	10.0.0.63
Modbus address:	255 (255 , or 1..247 for bridge)
Node number:	1 (1 .. 247)

Drag Modbus commands from Catalog window



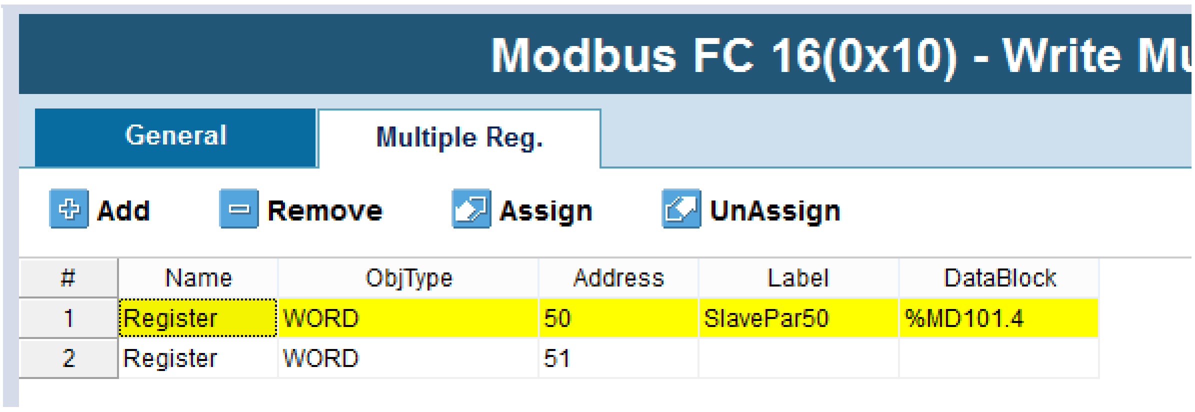
Device name	Versi...	Description
Modbus FC-01	1.0	Read Coils - Function 01 (0x01)
Modbus FC-02	1.0	Read Discrete Inputs - Function 02 (0x02)
Modbus FC-03	1.0	Read Holding Register - Function 03 (0x03)
Modbus FC-04	1.0	Read Input Registers - Function 04 (0x04)
Modbus FC-05	1.0	Write Single Coil - Function 05 (0x05)
Modbus FC-06	1.0	Write Single Register - Function 06 (0x06)
Modbus FC-15	1.0	Write Multiple Coils - Function 15 (0x0F)
Modbus FC-16	1.0	Write Multiple Register - Function 16 (0x10)



Start address:	50 (0 .. 65535)
Polling time:	0 ms (0 = write on variation)
Time out:	1000 ms
Wait before send:	0 ms
Oneshot variable	DoSetPar50

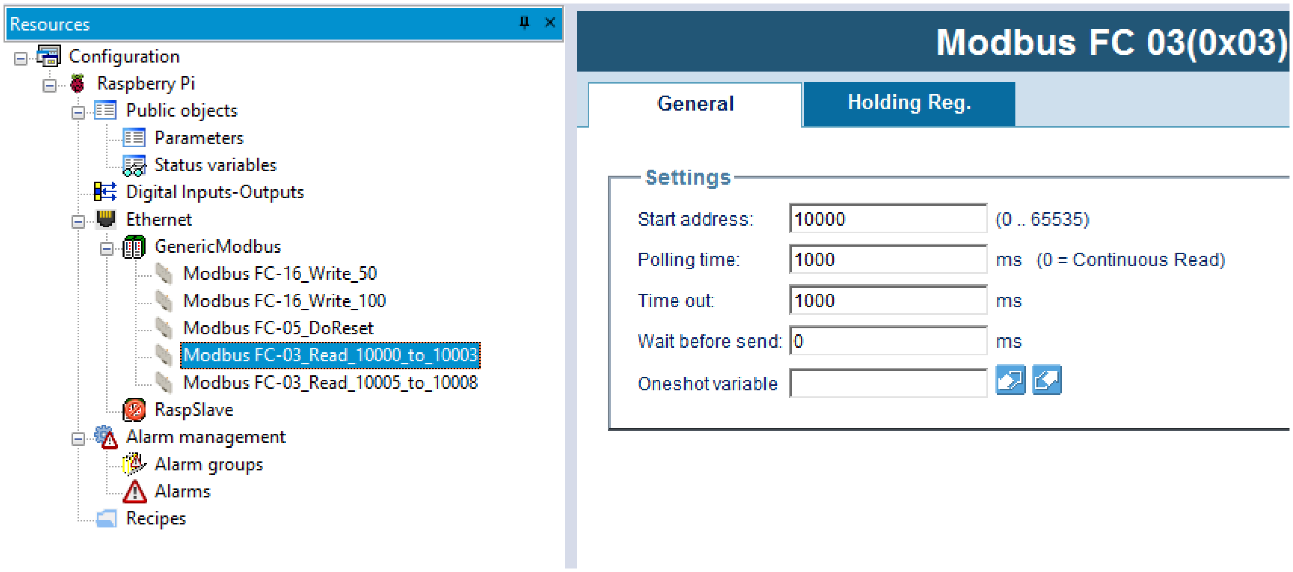
Sart Address is the first register in the command, in this case SlavePar50 of the RaspSlave dictionary.

This is a two WORDS parameter so you should add another (empty) row



#	Name	ObjType	Address	Label	DataBlock
1	Register	WORD	50	SlavePar50	%MD101.4
2	Register	WORD	51		

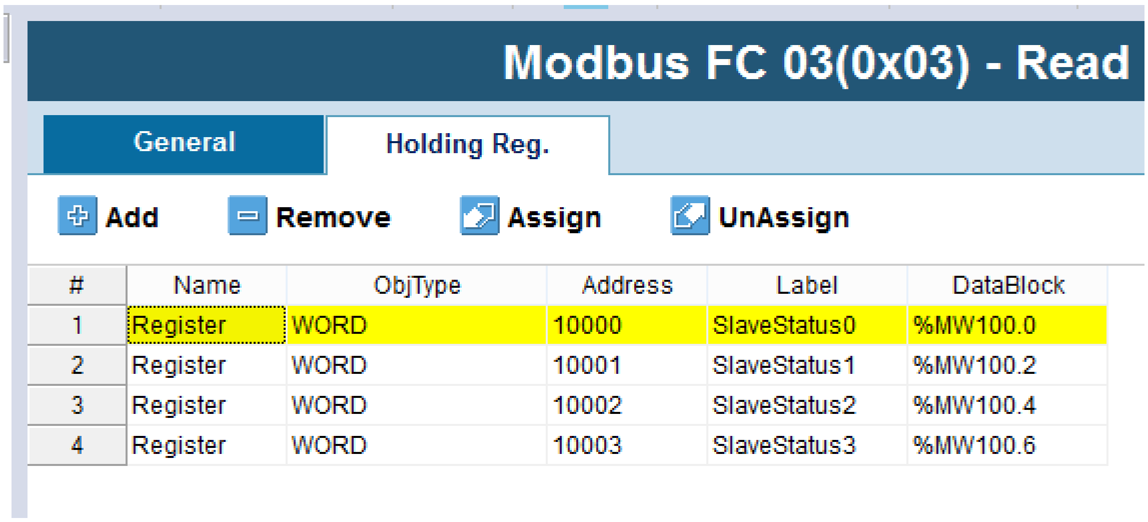
This is a sample of read multiple registers:



Start address:	10000	(0 .. 65535)
Polling time:	1000	ms (0 = Continuous Read)
Time out:	1000	ms
Wait before send:	0	ms
Oneshot variable		

Start address is 10000 so StatusVariables SlaveStatus1 in the RaspSlave target.

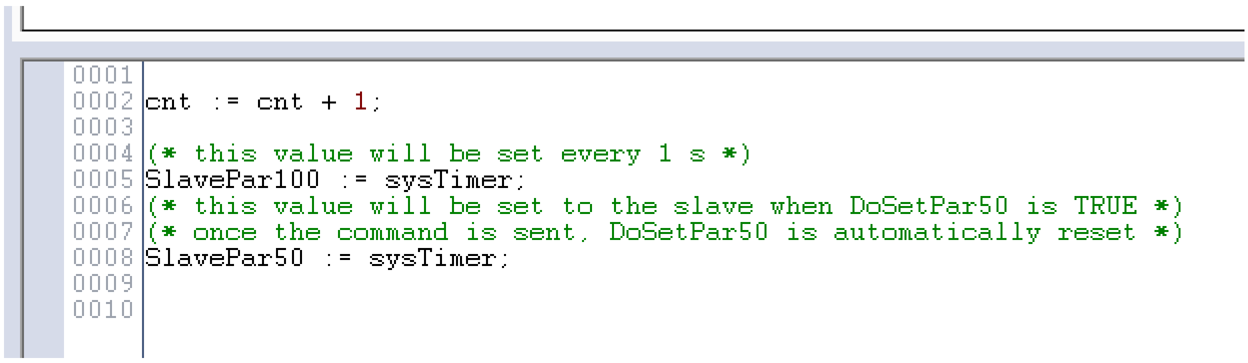
Then it is indicated to read 4 consecutive register and put values in SlaveStatus0-3 variables



#	Name	ObjType	Address	Label	DataBlock
1	Register	WORD	10000	SlaveStatus0	%MW100.0
2	Register	WORD	10001	SlaveStatus1	%MW100.2
3	Register	WORD	10002	SlaveStatus2	%MW100.4
4	Register	WORD	10003	SlaveStatus3	%MW100.6

Please use read holding register even if the status variables are set as read only in this version!

Associated code to the fast task

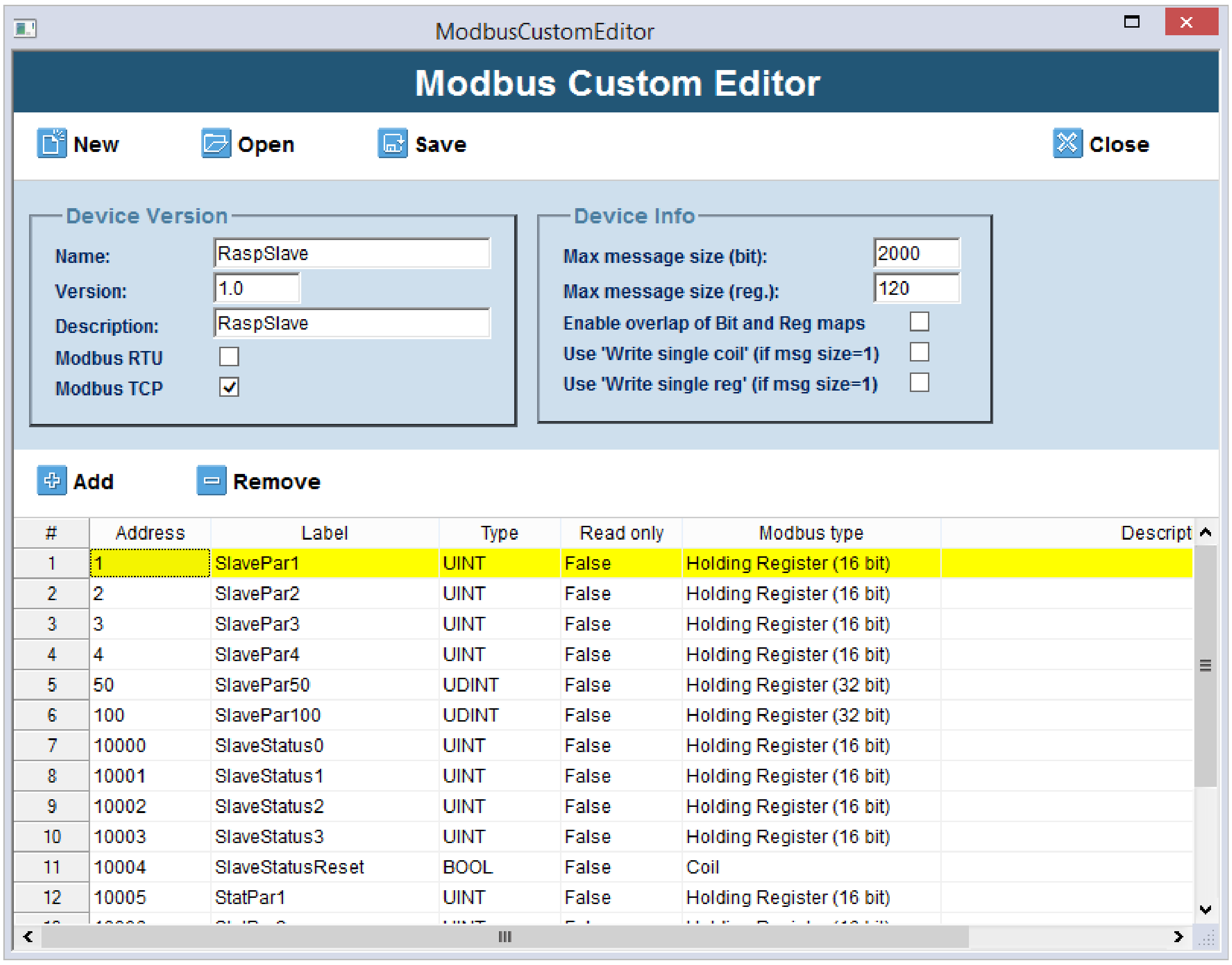


```

cnt := cnt + 1;

(* this value will be set every 1 s *)
SlavePar100 := sysTimer;
(* this value will be set to the slave when DoSetPar50 is TRUE *)
(* once the command is sent, DoSetPar50 is automatically reset *)
SlavePar50 := sysTimer;


```

Modbus Custom Editor

The same data exchange can be automated using Developer > Modbus Custom Editor

You can describe here the slave database



Device Version

Name: RaspSlave
Version: 1.0
Description: RaspSlave
Modbus RTU ☐
Modbus TCP ✓

Device Info

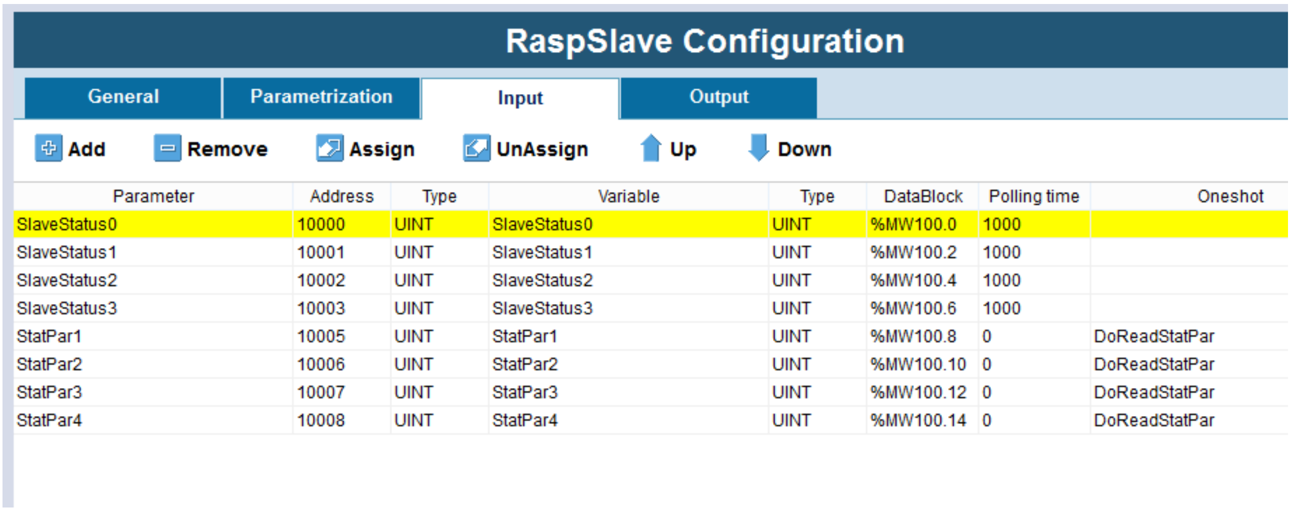
Max message size (bit): 2000
Max message size (reg.): 120
Enable overlap of Bit and Reg maps ☐
Use 'Write single coil' (if msg size=1) ☐
Use 'Write single reg' (if msg size=1) ☐

#	Address	Label	Type	Read only	Modbus type	Descript
1	1	SlavePar1	UINT	False	Holding Register (16 bit)	
2	2	SlavePar2	UINT	False	Holding Register (16 bit)	
3	3	SlavePar3	UINT	False	Holding Register (16 bit)	
4	4	SlavePar4	UINT	False	Holding Register (16 bit)	
5	50	SlavePar50	UDINT	False	Holding Register (32 bit)	
6	100	SlavePar100	UDINT	False	Holding Register (32 bit)	
7	10000	SlaveStatus0	UINT	False	Holding Register (16 bit)	
8	10001	SlaveStatus1	UINT	False	Holding Register (16 bit)	
9	10002	SlaveStatus2	UINT	False	Holding Register (16 bit)	
10	10003	SlaveStatus3	UINT	False	Holding Register (16 bit)	
11	10004	SlaveStatusReset	BOOL	False	Coil	
12	10005	StatPar1	UINT	False	Holding Register (16 bit)	

Give it a name RaspSlave and then you can easily make data exchange.

You will find it in the Catalog and Drag & Drop under the Ethernet Modbus TCP master node

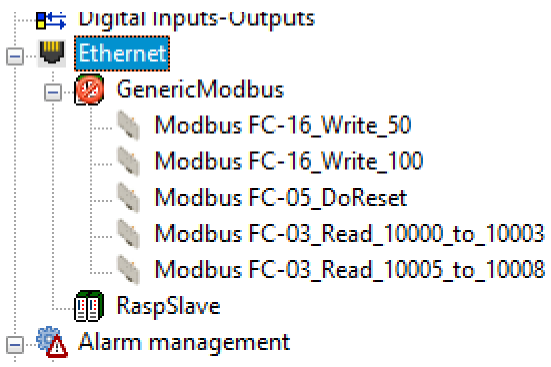
Then Input/Output settings will be easy to do:


RaspSlave Configuration

General | Parametrization | Input | Output

Add | Remove | Assign | UnAssign | Up | Down

Parameter	Address	Type	Variable	Type	DataBlock	Polling time	Oneshot
SlaveStatus0	10000	UINT	SlaveStatus0	UINT	%MW100.0	1000	
SlaveStatus1	10001	UINT	SlaveStatus1	UINT	%MW100.2	1000	
SlaveStatus2	10002	UINT	SlaveStatus2	UINT	%MW100.4	1000	
SlaveStatus3	10003	UINT	SlaveStatus3	UINT	%MW100.6	1000	
StatPar1	10005	UINT	StatPar1	UINT	%MW100.8	0	DoReadStatPar
StatPar2	10006	UINT	StatPar2	UINT	%MW100.10	0	DoReadStatPar
StatPar3	10007	UINT	StatPar3	UINT	%MW100.12	0	DoReadStatPar
StatPar4	10008	UINT	StatPar4	UINT	%MW100.14	0	DoReadStatPar


```
Digital Inputs-Outputs
Ethernet
    GenericModbus
        Modbus FC-16_Write_50
        Modbus FC-16_Write_100
        Modbus FC-05_DoReset
        Modbus FC-03_Read_10000_to_10003
        Modbus FC-03_Read_10005_to_10008
    RaspSlave
Alarm management
```

In the RaspMaster project both ways to do the same Modbus communication with the RaspSlave is provided. Download the project enabling GenericModbus or RaspSlave.

Library

Modbus.pll library with diagnostic structure can be linked to the project from

C:\Program Files (x86)\Axel PC Tools\Catalog\RaspPI_1p1\Libraries