DANISH
TECHNOLOGICAL
INSTITUTE

Focus on the control and simulation setup

# Appendix B:

# Description of the OPSYS test rig software

**Danish Technological Institute**

**Title:**
Appendix 3: Description of the OPSYS test rig at Teknologisk Institut

**Prepared for:**
EUDP

**Prepared by:**

Danish Technological Institute
Teknologiparken
Kongsvang Allé 29
8000 Aarhus C
Refrigeration and Heat Pump Technology

March 2023
Author: Troels Stevns Pedersen

# Table of Contents

1. Introduction ..... 4
2. Control and communication setup ..... 4
3. Hardware Components ..... 5
3.1. Router ..... 5
3.2. SIP ..... 5
3.3. Trend (IQ4E) ..... 5
3.4. PreHeat gateway ..... 5
3.5. PC ..... 5
4. Software components ..... 6
4.1. Python ..... 6
4.2. Modbus server ..... 6
4.2.1. Modbus clients (controller and simulation) ..... 7
4.3. House simulation ..... 8
4.4. Controller ..... 9
4.5. SIP ..... 9
4.6. Trend / CTS963 ..... 10
4.7. The simulation pc ..... 10
4.7.1. Python files ..... 10
5. Test procedure ..... 12
5.1. Start-up procedure ..... 12
5.1.1. Pre-test procedure ..... 12
6. List of known issues ..... 12
7. Data processing scripts ..... 13
8. Guide on how to operate the OPSYS 2 test rig ..... 14
8.1. Modbus automatic execution in windows task schedule ..... 14
8.2. Start test ..... 16
8.3. Analysis of test results ..... 16
Appendix 1 Quick guide on how to operate the OPSYS test rig ..... 12

# 1. Introduction

The purpose of this document is to disseminate knowledge regarding the test rig used in the project "*P2003908_Optimering af gulvvarme og vp systemer*" and "P200618_OPSYS 2,0", hereafter referred to as OPSYS.

The document is an update of the documentation made in the original Opsys project.

# 2. Control and communication setup

This section provides a description of the control and communication setup for the test rig. Concerning control, communication and data acquisition, the test setup comprises of four different entities.

1. The Simulation pc
2. The SIP
3. The Trend controllers
4. Router

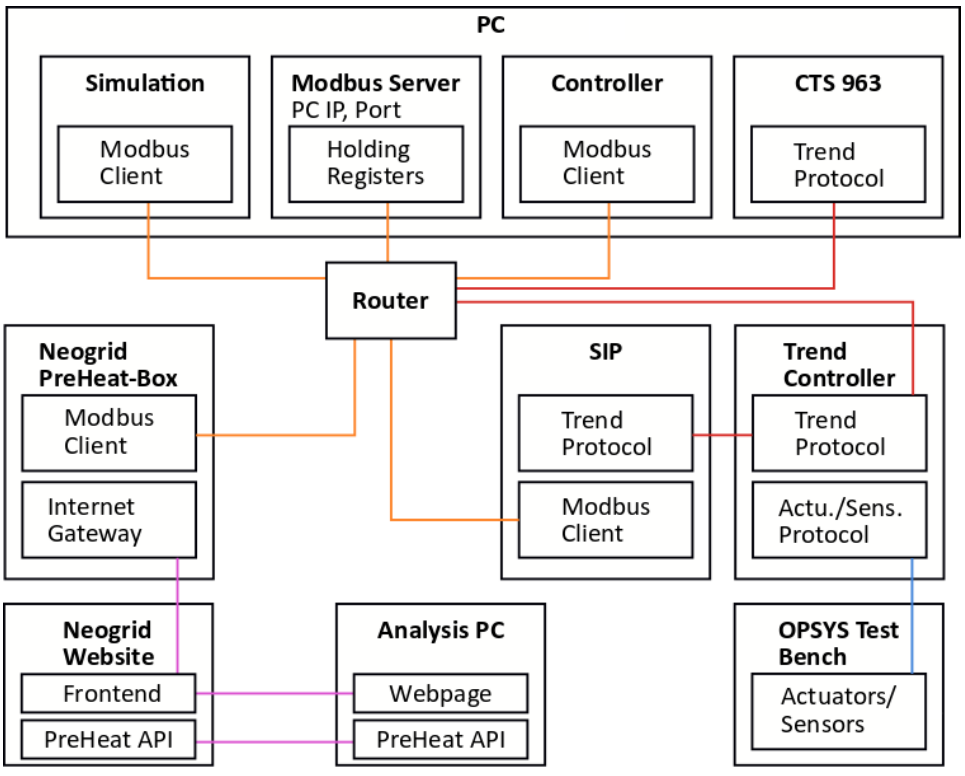



**Figure 1: Communication and data acquisition setup**

The four entities are all depicted in Figure 1, which illustrates the communication and data acquisition setup. The colors of the lines indicate the type of communication, (orange) LAN connection, (red) Trend Protocol, (purple) Internet communication and (blue) Individual sensor signals.

# 3. Hardware Components

This section lists and describes the hardware components, which supports the communication and software infrastructure.

| | IP-address | Hostname |
|---|---|---|
| **Router** | 192.168.0.1 | |
| **SIP** | 192.168.0.56 | SIP |
| **Trend (IQ4E)** | 192.168.0.54 | TREND_0B_C4_84 |
| **PreHeat Gateway** | | |
| **PC** | 192.168.0.52 | |

De ovenstående IP adresser skal følges, ellers opstår der fejl i kommunikationen.

## 3.1. Router

The 4G router handles the communication to the internet and between the nodes on the local network.

## 3.2. SIP

The task of the SIP is to create a Modbus interface between the Python code running in the Simulation pc and the Trend controllers. The SIP acts as a ModbusTcp client, and it ensures that data signals from the Trend controllers are written into the data store maintained by the Modbus server. In addition, the SIP also ensures that data is transferred from the datastore of the Modbus server to the Trend controllers. The mapping between the Modbus addresses and the Trend controller input and output is configured using the web based configuration tool on the SIP.

## 3.3. Trend (IQ4E)

Most of the dynamic equipment is connected to the Trend. Whereby control algorithms of valves and pumps is done here, but received information from PC by what should be perfomed.

## 3.4. PreHeat gateway

Gateway from NeoGrid which can provide information regarding electricity price and also weather forecast to tell if there might be a big PV production or not. Only used for temperature reading at the testrig, DHW relays and stop signal for HP are also connected but not working stable.

## 3.5. PC

The computer where modbus server is running, setting the base for interconnection between all the different components. Further is the control and simulation of the OPSYS rig itself also executed and logged from here.

# 4. Software components

This section covers sub-processes which are executed to carry out simulation on the OPSYS test rig.

Python is used as the general test environment to handle simulation and controlling test rig, so first are the required environment packages defined. And then afterwards is the software presented more detailed; Modbus, Simulation, Controller & Sip.

## 4.1. Python

Following packages are needed to have installed in the python environment at which everything is executed:

| Package | Link |
|---|---|
| Scipy (1.5.2) | |
| Matplotlib | |
| pyfmi (2.8.1) | https://anaconda.org/conda-forge/pyfmi |
| pandas (1.1.3) | https://anaconda.org/conda-forge/pandas |
| Gurobi (9.0.3) | |
| cvxpy (1.1.6) | https://www.cvxpy.org/install/index.html |
| Nose (1.3.7) | |
| neurolab (0.3.5) | https://pypi.org/project/neurolab/ |
| xlsxwriter (1.3.7) | https://anaconda.org/conda-forge/xlsxwriter |
| Pymodbus (2.3.0) | https://anaconda.org/conda-forge/pymodbus<br><br>https://github.com/riptideio/pymodbus/releases/tag/v1.3.0rc2 |

## 4.2. Modbus server

The Modbus server handles the communication between the simulation, controller and SIP.

**Hardware:** OPSYS PC

**Folder:** ..\Opsys\test\

**Files:** run_modbus_server.bat, modbus_server.py

**Dependencies:** Python 3.7, pymodbus 2.3.0, modbus_config_vx_y.csv

- The Modbus server has been configured to start when the PC turns on. This is done using the Task Scheduler in Windows, see appendix for configuring this. It has to be updated when new pc password is set.
- Alternatively, the **run_modbus_server.bat** can be executed.
- The Modbus server is hosted on the standard modbus port **502**.

### 4.2.1. Modbus clients (controller and simulation)

The three Modbus clients are the simulation, controller and SIP there is exchanging information by the modbus server. This section is focusing with the simulation and controller interface. The SIP is handled in Section 4.5

The base class for handling the interface to the Modbus is called ***OpsysModbusInterface***. This class is inherited by both classes: **ModbusLead** (simulation) and **ModbusFollower** (controller). The reason for calling the classes lead and follower is explained further later in this section.

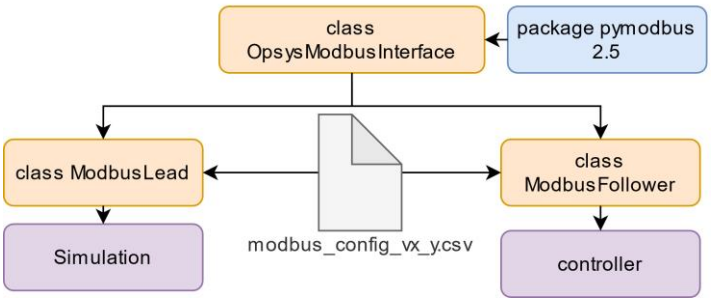



**Figure 2 shows the connection diagram for the modbus interface used by the simulation and controller.**

The file, named **modbus_config_vx_y.csv**, contains the description of the interface to the sip. These are the registers which the simulation and controller need to read and write to, for communication with the test-rig. The first column is the name used by the sip, the second is the register and the third is whether is for reading or writing. In this setup no two-direction registers are used, a register is either for writing or reading when communicating with the SIP.

The main information flows on the modbus, are seen in Figure 3. Signal names in the diagram reflects the names given in Trend.

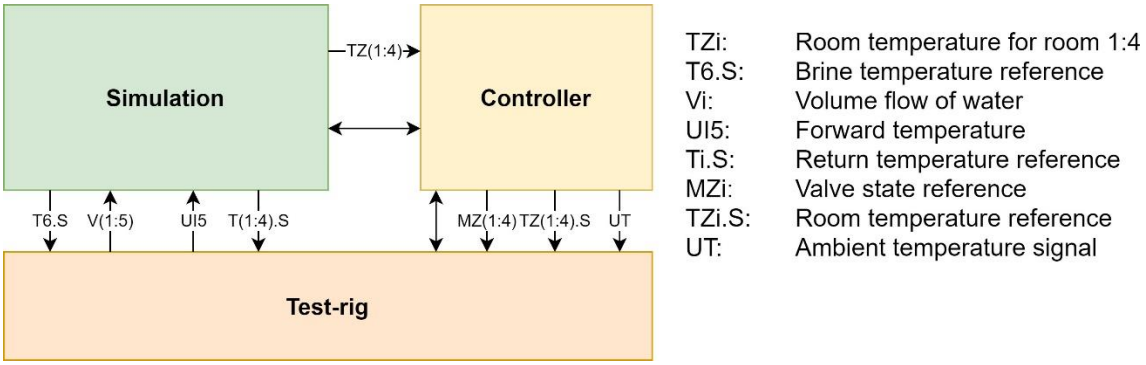



**Figure 3: Main information flows on modbus**

In this version of the test-rig infrastructure the simulation has been separated from the controller. This is done to introduce a certain level of authenticity. Had the house been a house and not a simulation, the controller would be a stand-alone program, hence it has been separated in the test-rig. This of course creates a need for proper timing.

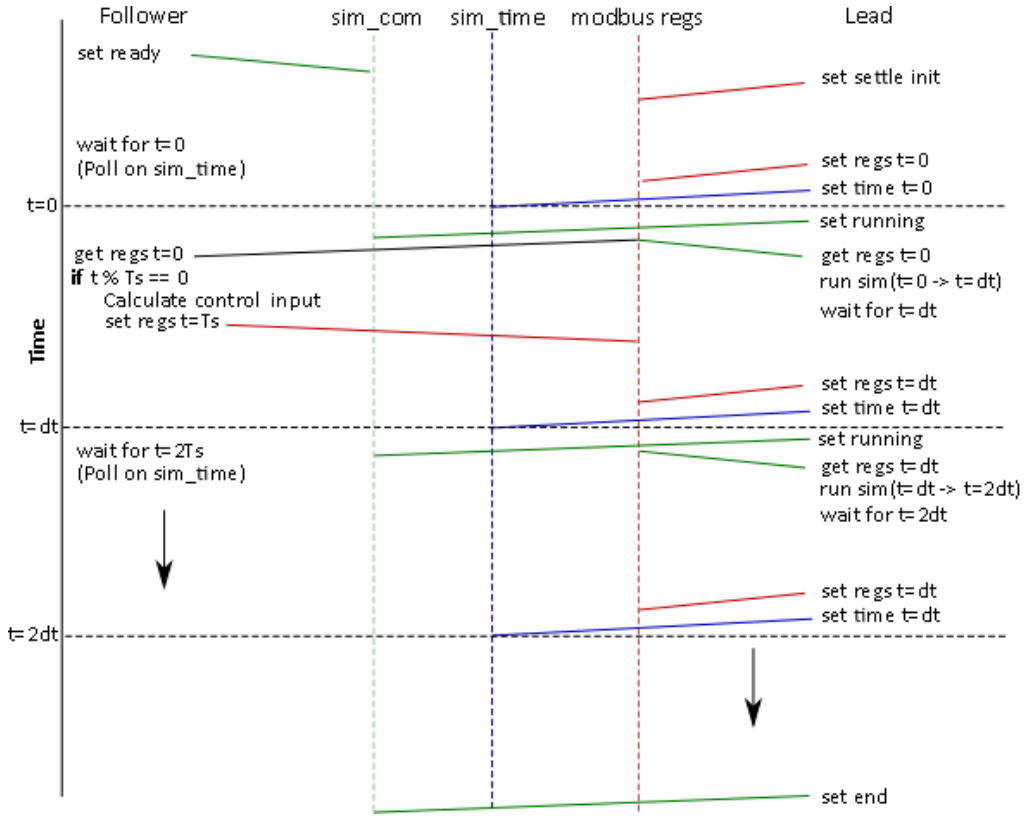



**Figure 4 illustration of communication between the follower and lead**

Requirements for communication between simulation and controller:

- The software should not crash due to short term network problems.
- The **simulation leads** by publishing the simulation time on the Modbus.
- The **controller follows** according to simulation time.

## 4.3. House simulation

The house simulation provides the energy demand for the test-rig.

**FMU Folder:** ..\Opsys\data\input\fmu\

**Simulation Folder:** ..\Opsys\test\

**Files:** fmu_full_test.py, fmu_full_test_config.py

**Dependencies:**

4 complete houses [(Opsys or Original) x (1970 or 2010)] have earlier in OPSYS 1 been modelled in Dymola and exported as an FMU, as can be seen as a black box with required input values and then gives output values;

| Input | Output |
|---|---|
| T_forward | T_room1 |
| V_FLOW_Z1_1 | T_room2 |
| V_FLOW_Z1_2 | T_room3 |
| V_FLOW_Z2 | T_room4 |

| | |
|---|---|
| V_FLOW_Z3 | T_return1 |
| V_FLOW_Z4 | T_return2 |
| | T_return3 |
| | T_return4 |

Settings to the FMU model can be adjusted in the config python script.

## 4.4. Controller

The controller adjusts the test rig to meet the heating demands the house simulation requires.

**Folder:** ..\Opsys\test\

**Files:** Ctrl_3.py, fmu_full_test_config.py

**Dependencies:**

Mostly all control is done from Ctrl_3.py, DHW, PV and battery control has been implemented here aswell.

The battery model is stored in battery_milp.py, which is an Mixed-Integer Linear Problem.

Data from PV panels are generated in the PV model stored under input files and generators.

## 4.5. SIP

The task of the SIP is to create a Modbus interface between the Python code running in the Simulation pc and the Trend controllers. The SIP acts as a ModbusTcp client, and it ensures that data signals from the Trend controllers are written into the data store maintained by the Modbus server running in Python on the Simulation pc. In addition, the SIP also ensures that data is transferred from the datastore of the Modbus server to the Trend controllers. The mapping between the Modbus addresses and the Trend controller input and output is configured using the web based configuration tool on the SIP.

The configuration consist of two highly important tabs: Map points and vIQ. Map points is used to configure which Modbus registers the different signals are mapped to.

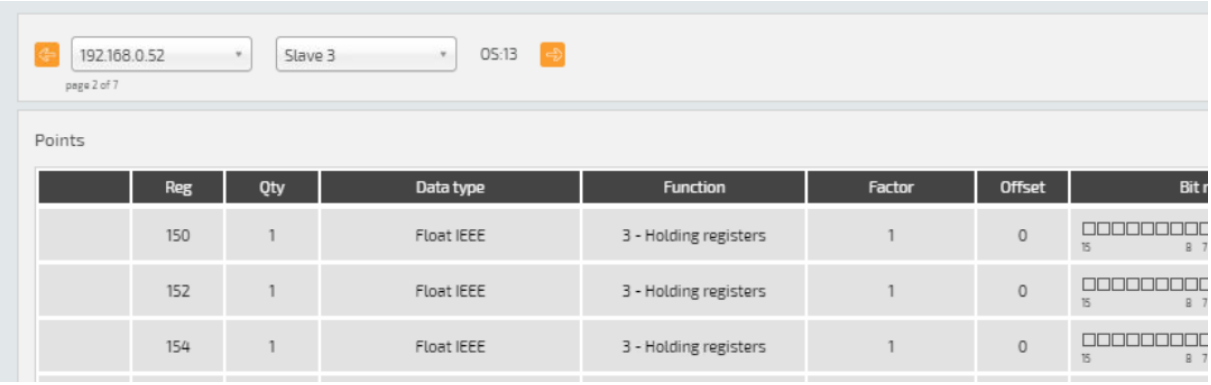



**Figure 5 shows the Map Points tab in the SIP configuration page. It is important that the IP-address seen in the upper left corner match the IP of the Modbus server.**

In the tab vIQ, it is possible to see an overview of the connected modules. It is also possible to see the live values coming from the controller, simulation and test-rig. This property is useful in case one needs to debug the system. The sub-tabs of interest are Sensors and Knobs.

## 4.6. Trend / CTS963

The task of the Trend controllers is to provide the low-level controls of all the electronic valves on the test rig along with the logging of all the relevant measurements and set point for the various control loops. In addition, the Trend controllers provides an interface to the heat pump by presenting five inputs and five outputs that can be used for future improvements of the test rig setup. One of the outputs from the Trend controllers is currently being used to set the ambient temperature for the heat pump and thereby indirectly controlling the forward temperature of the heat pump.

CTS963 is the computer software from which the Trend system is directly monitored, and user can define if specific values should be automatic adjusted by the controller or set a fixed value the user define.

## 4.7. The simulation pc

Figure 1 shows that the **Simulation pc** is running two different tasks, which are:

1. Executing the python code which handles the following tasks:
    a. Running the simulation of the house
    b. Running the high-level control algorithms
    c. Running the Modbus server to enable communication with the Trend controllers through the SIP Modbus interface box and communication with neogrid box.
    d. Data logging are saved to .pkl files, multiple pkl files are saved to differentiate the data and not have one big file. Pkl files are saved to data/output/fmu_full_test_######## where the last part is the time and date when simulation was started.
2. Optional: Running the CTS 963 software, which handles the following tasks
    a. Providing GUI for the Trend controllers and thereby an interface to the test rig
    b. Data logging the Trend controllers in an internally used database
    c. Data logging each measurement point in a separate csv file. Files are saved in "C:\Log data". Files are only updated when CTS963 is running.
    d. The needed data from trend are communicated to modbus server and save to the pkl files earlier mentioned. This has been done in order to ease the data analysis afterwards with all data in one place.

### 4.7.1. Python files

The python code is split up into five files and some sub modules:

1. **Run_modbus_server.py**
    a. Main file used when making tests on the test rig hardware. To start the modbus server.

b. Uses class/function descriptions located in the other python files.
c. The following configuration parameters can be changed in this file:
    i. IP = '192.168.0.52' #ip address of the host computer
    ii. PORT = 502 #Port modbus server is listening on

2. **fmu_full_test_config.py**
    a. Configuration file for both the simulation and controller.
    b. The following configuration parameters can be changed in this file:
        i. Simulation settings: Sample time, stop time
        ii. File and Folder, output folders and name
        iii. Initial values for forward temperature and flows
        iv. FMU options, among many the FMU model to use
        v. Real time
        vi. Controller, number of rooms, reference temperature, offset, scale and ambient temperature.
        vii. Modbus info
3. **fmu_full_test.py**
    a. Reads data from the shared data store context.
    b. Generates input vectors for the simulation model and propagates it SIM_STEP seconds.
    c. Sends room temperatures and water return temperatures to the Trend controllers through the SIP.
    d. The fmu file, e.g., named **"OpSys_0Experiments_Test_0setup_0models_BR1970_0house_0simulation_0with_0data.fmu"** contain the model description for a 1970's house which uses the weather and load data profiles for a year located in **"all_data_1970.txt"** or **"all_data_noisy_BR10.txt"** (values are added +/- 10% normally distributed random noise).
    e. Note that the fmu should be generated as an fmu 2.0 using co-simulation with Dymola solvers in order to function properly (standard CVODE solver sometimes crashes and should be avoided – Dymola Radau-II solver is better).
4. **ctrl_3.py**
    a. Runs control algorithms, e.g., room flows and forward temperature.
    b. Configuration is done in 'fmu_full_test_config.py'.
    c. Results from test rig are saved to .pkl files there are easy to analyze and work with in the afterwards analysis
5. **Sub-modules**
    a. **Common**
    Common operations for the python environment as main folder, simulation name, and handling of output files and folder.
    b. **DymEx**

    c. **OpsysAlgorithms**
    Algorithms for controlling of test rig. Consisting of 3 python classes/algorithms: 1. simple on/off, 2. Discrete PID and 3. Balance flow concept (Formentlig styring efter 1 rum altid er åben.).
    d. **OpsysInternal**

e. **OpsysLog**
   Handles logging during simulation.
f. **OpsysModbusinterface**
   Handling all the basic communication for controller and simulation with modbus server. Consisting of 5 classes;1. Reg, 2. IOInterface, 3. ModbusInterface, 4. Lead, 5. Follower. "Follower" and "Lead", follows the earlier analogy explained on the follower/lead example used for controller and simulation.
g. **OpsysResultHandler**
   Handling the results from simulation and adds them to data file while simulation is running.
h. **TimeKeeper**
   Handles the simulation time and ensures the time keep in sync in relation to real time, functions for conversion of time to strings etc. Consisting of 3 classes; 1. TimeBase, 2. Repeater, 3. TimeKeeper.

# 5. Test procedure

The section provides a description of a test procedure that ensures that data from the data sources can be used and will cover the time period of simulation.

## 5.1. Start-up procedure

The following procedure must be followed to ensure that the data generated by the Trend controller system during testing is stored correctly.

### 5.1.1. Pre-test procedure

1. **Optional step:** Change relevant settings in the CTS 963, e.g. toggle relevant switches to ensure that input from SimPc is used as input to the Trend controllers.
2. Adapt the fmu_full_test_config.py to match the desired simulation to carry out.
3. Run system until system is ready for the actual test
   - To ensure proper operation of the test rig, it is necessary to start the simulation by running the python script "pymodbus_server.py. This ensures that the test rig receives relevant feedback and can operate correctly.
4. Run test
   - Start the controller.bat
   - Start the simulation.bat

# 6. List of known issues

This section presents a list of known technical issues concerning the test rig and the communication and control setup.

- The full stop signal to heat pump is very unstable, another solution should be found.
- Reading of NTC sensors are not correct, NTC sensors should be replaced with PT1000 sensors and then logged with an Agilent there sends the information to the inhouse Yoda platform.

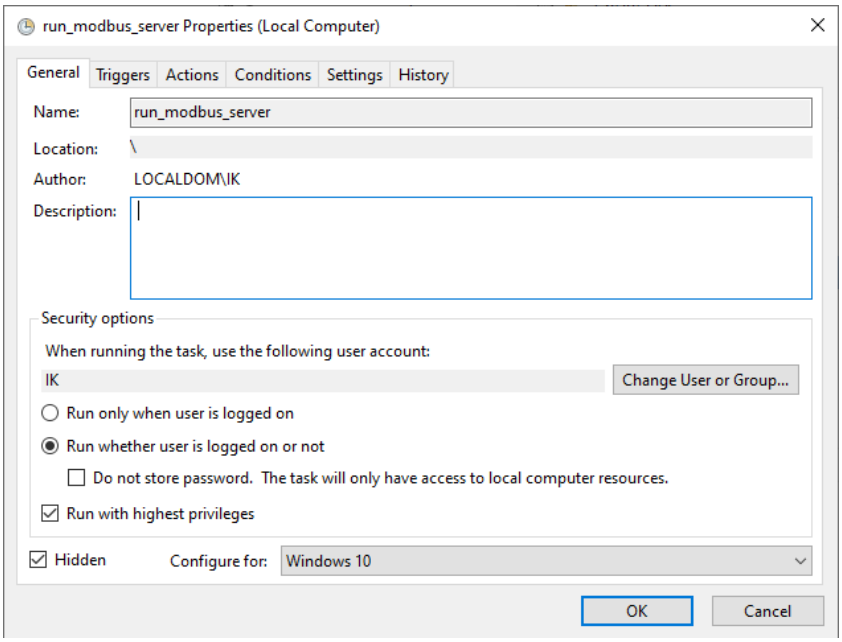
# 7. Data processing scripts

Run the analysis.py in python, data will be loaded and shown in corresponding plots.

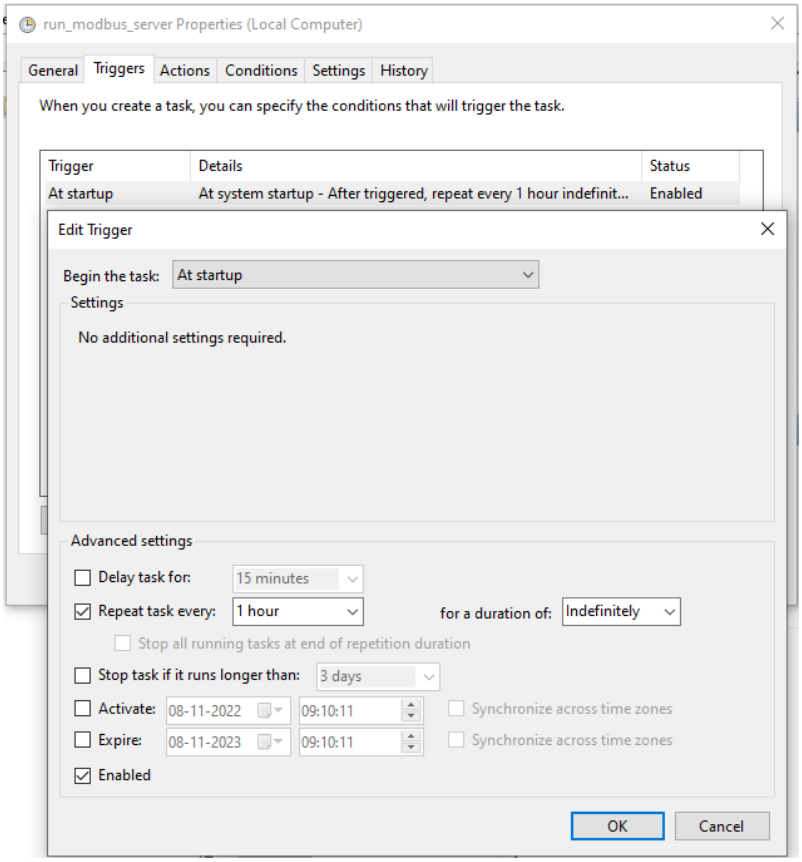
# 8. Guide on how to operate the OPSYS 2 test rig

## 8.1. Modbus automatic execution in windows task schedule

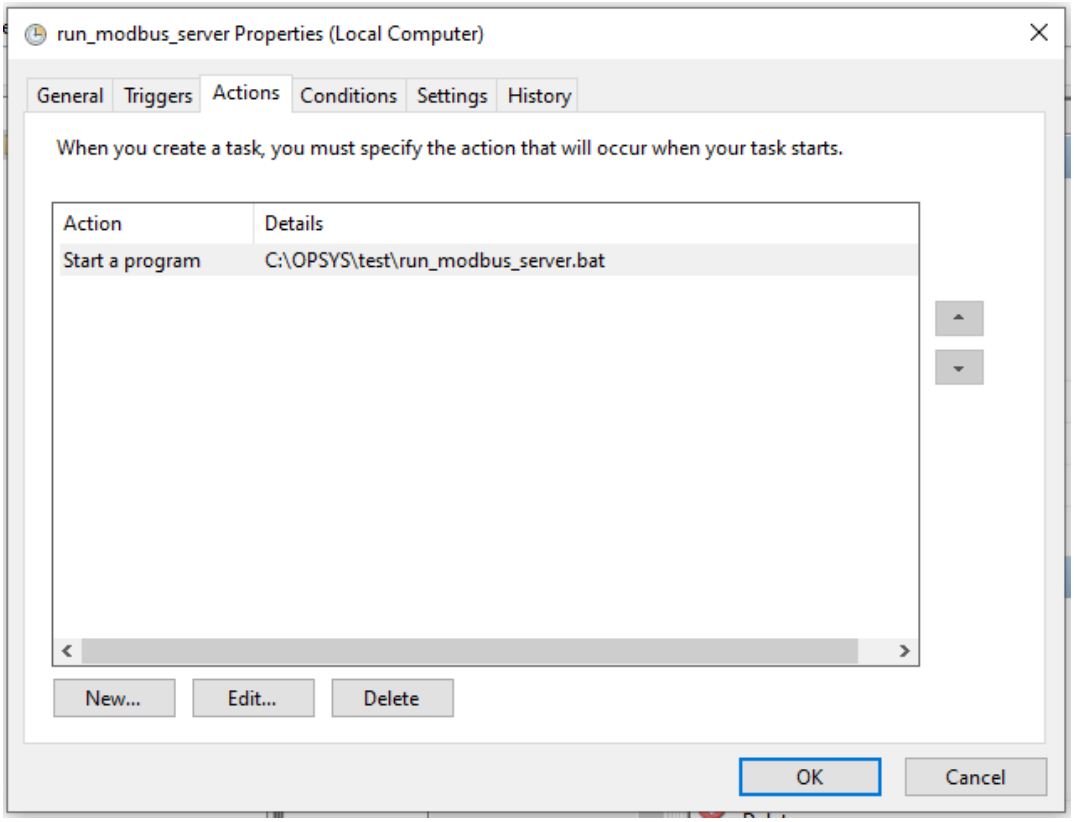
### General



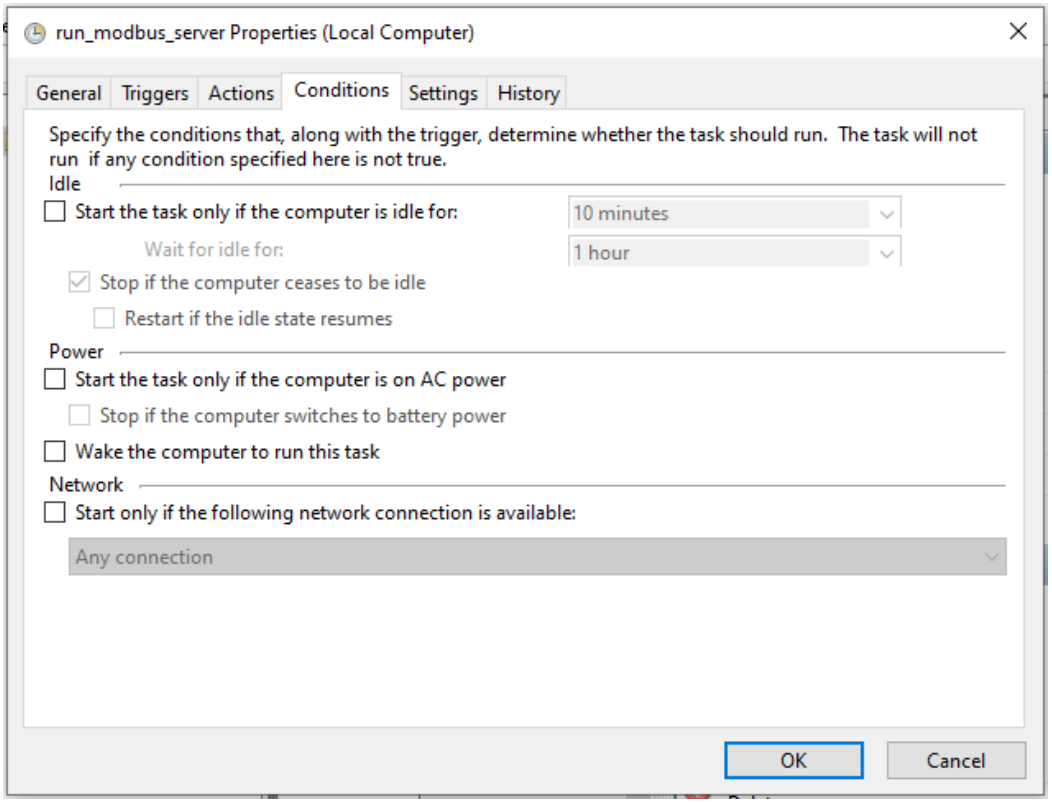

### Triggers



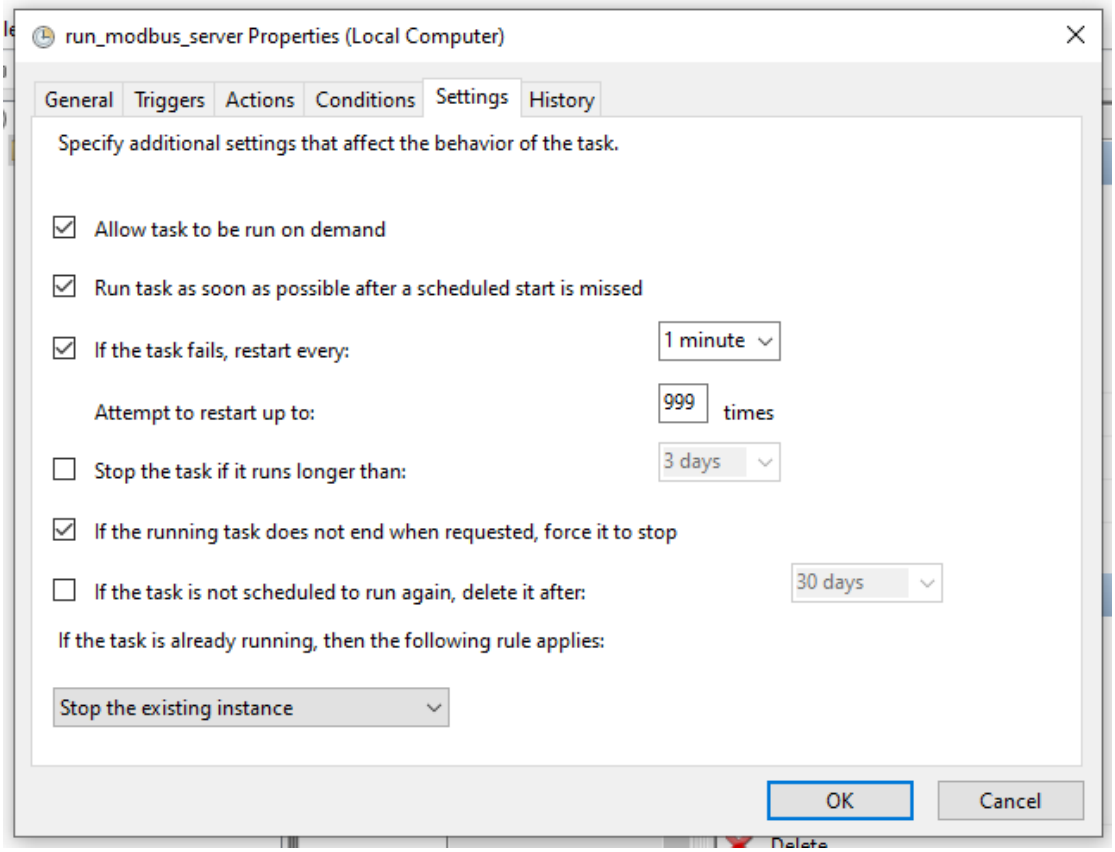
**Actions**




**Conditions**



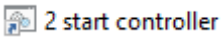
**Settings**



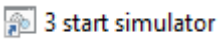

## 8.2. Start test

2.1 - First ensure configuration of all the test constants and data sets are defined correctly in the fmu_full_test_config.py according to the desired test.

2.2 - Execute: 2 start controller




2.3 - Execute: 3 start simulator




## 8.3. Analysis of test results

Run the analysis.py in python, data will be loaded and shown in corresponding plots.