Monitoring of WVC-600 solar-inverter via 433 MHz RF-interface

Author: Jaap van den Berg (jaap@berghelwig.nl) Date: 1-JAN-2022   Version: 0.2

Disclaimer/Read-Me-First!

The information in this document may be used freely on a non-commercial basis and at own risk without any support of the author. **Also legal and product responsibility issues regarding the use of the described equipment (inverter, RF-device) must be observed!** The author cannot be held responsible for any faults, problems, damage or any legal issues caused by the application of the information of this document.

Introduction/Summary

Since August 2020, the DC-power of my two self-installed solar-panels is converted to 230 Vac by the **WVC-600 solar-inverter**. Some models of this Chinese solar-inverter are equipped with a 433 MHz RF-interface that can provide on request measurement-data (like temperature, panel-voltage, output-power/energy, etc.)

Based on info from the websites https://community.openenergymonitor.org/t/wvc-inverter-mqtt-data-logging-for-all-versions-of-inverters/12352 and https://github.com/krywenko/WVC-inverters , a DIY RF data-communication system has been built. Controlled by a Python-program, this system can derive the measurement-data from the solar-inverter and store this data into a daily generated csv-file, which can be processed by spreadsheet software like Excel.

This document assumes that the reader has at least basic knowledge of Python. If not, knowledge can be obtained/refreshed at a Python tutorial like https://www.w3schools.com/python/default.asp , https://pythonbasics.org/ , etc.

Installation of Python version 3

In order to run the two Python-programs (both test and main program), Python version 3 (3.5 or higher) software must be installed on the computer that is connected to the RF data-communication system.

To avoid disturbing already existing installations, first check if Python version 3 has already been installed by entering the following code in a Python program text-file (e.g. pythonversion.py) and execute it:

```
import sys
print("Python version:")
print (sys.version)
print()
print("Version info:")
print (sys.version_info)
print()
input("Press Enter to continue…")
```

The Python installation software can be downloaded for free from https://www.python.org/ and is available for Windows, Linux, MAC-OS, etc.

A Google search based “linux python3 install” can provide useful additional info for installing Python version 3 on Linux based systems (like https://docs.python-guide.org/starting/install3/linux/).

After installation of the Python version 3 software, the installation of the library-package “**pyserial**” must be verified by the command **“pip list”** or by entering the following code in a Python program text-file (e.g. pyserialversion.py) and execute it:

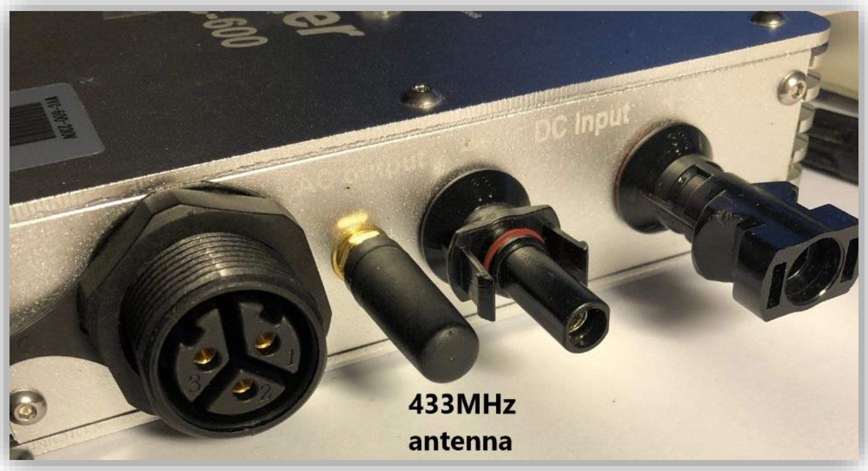
```
import serial
print("Pyserial-version:")
print(serial.__version__)
print()
input("Press Enter to continue...")
```

If not installed, “pyserial” must be installed by the command “**pip install pyserial**” (see also https://pypi.org/project/pyserial/)

Hardware and testing-software

Presence of RF-interface on the solar-inverter.

First of all, the presence of a 433 MHz antenna on a side-panel of the WVC-600 solar-inverter must be verified, before starting the project…



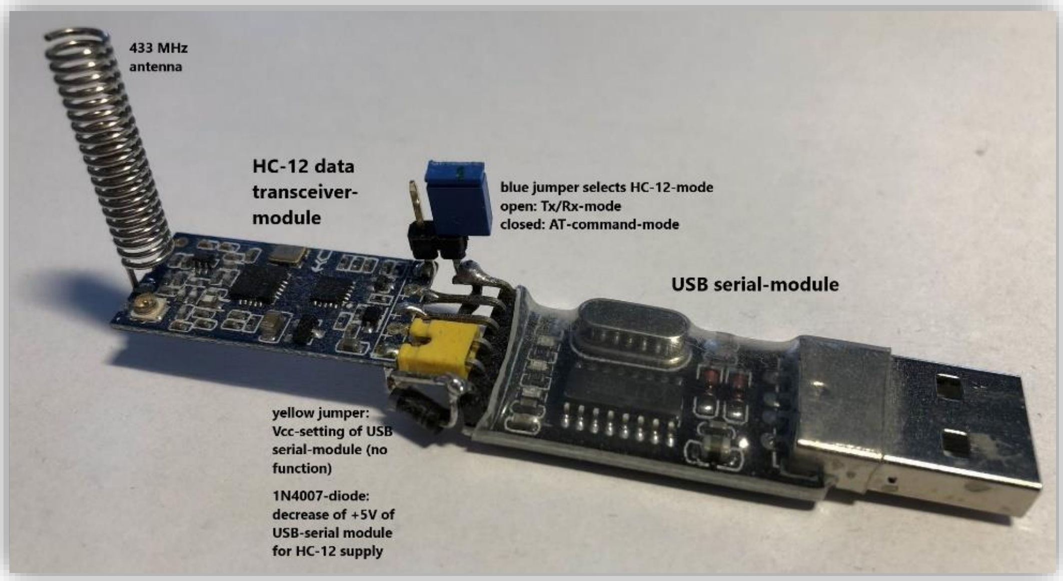
Construction of the RF data-communication system

The RF data-communication system consists of two modules:

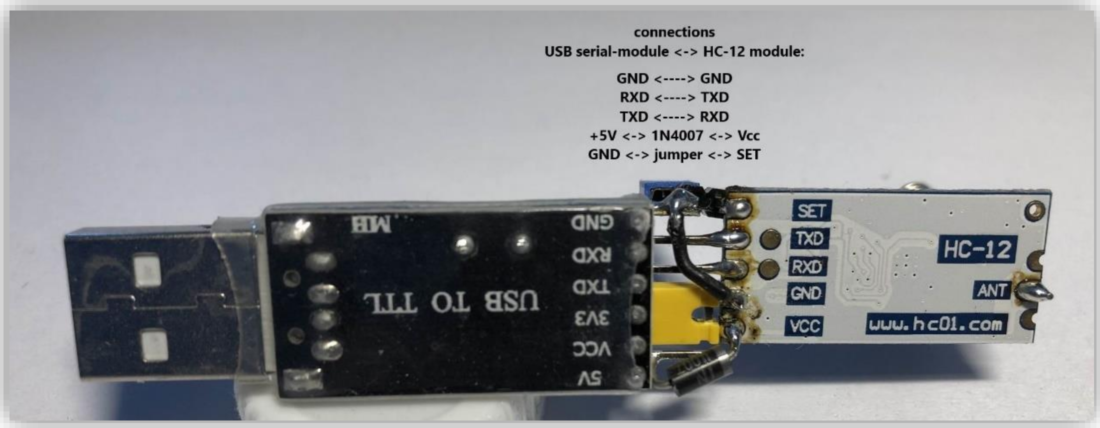
- USB-serial module (like the CH340 (preferred) or the PL2303)
- HC-12 data transceiver-module (documentation: HC-12.pdf)
- These two affordable modules are widely available at aliexpress.com, etc. For my own project, the modules have been bought at:
    - USB-serial module (including free download of driver): https://www.tinytronics.nl/shop/nl/communicatie-en-signalen/serieel/usb/ch340-3.3v-5v-ttl-usb-serial-port-adapter
    - HC-12 data transceiver-module: https://www.tinytronics.nl/shop/nl/communicatie-en-signalen/draadloos/rf/modules/hc-12-si4463-draadloze-serial-port-module-433mhz

The following pictures show the interconnection of these modules:

USB serial-module and HC-12 data transceiver-module


433 MHz antenna
HC-12 data transceiver-module
blue jumper selects HC-12-mode
open: Tx/Rx-mode
closed: AT-command-mode
USB serial-module
yellow jumper: Vcc-setting of USB serial-module (no function)
1N4007-diode: decrease of +5V of USB-serial module for HC-12 supply

USB serial-module and HC-12 data transceiver-module (connections)

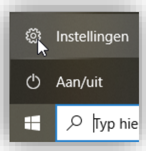


Module connection-issues:

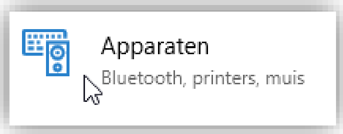
- The 5Vdc-supply of the USB serial-module must be connected via a silicon-diode 1N4007 to the Vcc-pin of the HC-12 module in order to decrease the supply-voltage.
- The yellow jumper connects the “3V3”-pin to the “VCC”-pin of the USB serial-module
- The blue jumper enables the grounding of the “SET”-pin of the HC-12 module to put this module in the AT-command mode. In this mode, the HC-12 module can be configured (regarding RX/TX-frequency, etc.) if necessary. The HC-12 module default configuration (RX/TX-frequency 433.400 MHz at 100mW (+20 dBm), serial communication 9600 bits/sec, 8 bits no parity, 1 stop-bit) complies to the RF-interface of the WVC-600 solar inverter.
- The 433 MHz antenna (part of the HC-12 module delivery) can be soldered to the ANT-output of the HC-12 module. But also a “quarter wave” antenna (based on an app. 17 cm wire) will do nicely.

Testing and configuring of the RF data-communication system

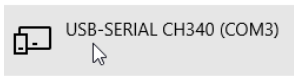
First, set the HC-12-module in the AT-command mode by grounding the “SET”-pin.
After inserting the USB serial-module in a Windows-PC, the Windows-configuration should display an active COM-port (e.g. in this case COM3)



->

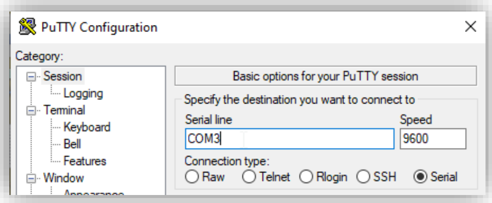


->

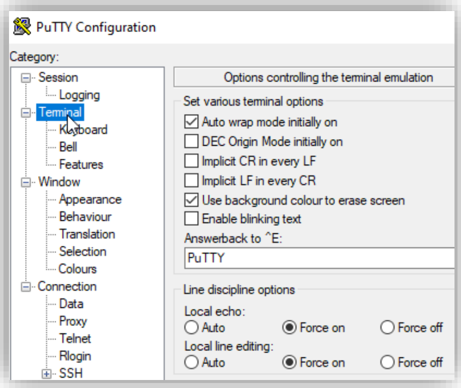


By default, the RF data-communication system should communicate at 9600 bits/sec 8 bits, no parity, 1 stop-bit.

The well-known terminal program PuTTY can be used for serial communication with the system. Make sure to set the correct COM-port, Speed=9600 bits/sec Serial and in the Terminal-menu the”settings “Local echo” and “Local line editing” to “Force on”

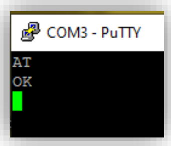

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window

Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line: COM3
Speed: 9600
Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH

Options controlling the terminal emulation
Set various terminal options
[✓] Auto wrap mode initially on
[ ] DEC Origin Mode initially on
[ ] Implicit CR in every LF
[ ] Implicit LF in every CR
[✓] Use background colour to erase screen
[ ] Enable blinking text
Answerback to ^E: PuTTY
Line discipline options
Local echo: ○ Auto ● Force on ○ Force off
Local line editing: ○ Auto ● Force on ○ Force off

Next, click on “Open” and enter (use capital characters only!) “AT” in the black Putty screen. The system should respond with “OK”.


COM3 - PuTTY
```
AT
OK
```

Now, based on the HC-12-documentation HC-12.pdf, enter “AT+RX”. The system should now respond:
OK+B9600
OK+RC001
OK+RP:+20dBm
OK+FU3

This corresponds with the default HC-12-setting:
-Mode FU3
-9600 bits/s (8 bits data, no parity, 1 stop-bit)
-Radio Channel CH001 (433.400 MHz) with output-power +20 dBm (100 mW)

Apart from the output-power (can be decreased if desired/necessary), this default setting must be used to commumicate to the WVC-600 solar-inverter.

Program for testing the communication towards WVC-600 solar-inverter

After successful testing of the AT-command mode, the HC-12 module can now be switched to normal operation by disabling the grounding of the "SET"-pin. The following Python3 progam "wvc600-com-test.py" can be used to test the operation of the RF data-communication system towards the WVC-600 solar-inverter. Before running this program, check at the input-parameters:

- The correct COM-port has been configured
- The correct WVC-600 inverter-id has been entered (see also sticker on inverter)

```
print("  ===============================================================")
print("  |    Python3 program for testing communication towards          |")
print("  |                                                               |")
print("  |    WVC-600 inverter via RF-interface on 433.400 MHz           |")
print("  |                     based on                                  |")
print("  |   USB-serial-module and HC-12 data-transceiver-module         |")
print("  |                                                               |")
print("  | Author: Jaap van den Berg  Date: 31-DEC-2021  Version: 0.1    |")
print("  ===============================================================")

# --------------------------------------------input-parameters----------------------------------------
# Select correct COM-port!!

# if Linux, un-comment following line
# computer_COM_port="/dev/ttyUSB0"

# if Windows, un-comment following line and enter correct COM-port (displayed by Windows configuration)
computer_COM_port="COM3"

# Enter correct WVC-600 inverter-id!! (see sticker on inverter)
wvc600_id = "12345678"
# -----------------------------------------------------------------------------------------------------

print()
print("Program will send a hexadecimal request-code via com-port",computer_COM_port)
print("to WVC-600 inverter with id",wvc600_id)
print("(correct this in -----input-parameters----- of the Python-program if necessary)")

print()
input("Press Enter to continue")

# Import of necessary Python-modules
import serial

# Construction of command to be sent to the WVC-600 inverter
command_firstpart_hex = "f20000000000000000"
command_lastpart_hex = "65"
command_total_hex = command_firstpart_hex + wvc600_id + command_lastpart_hex
command_to_wvc600 = bytes.fromhex(command_total_hex)

print()
print("Hexadecimal request-code sent to the WVC-600-inverter with id",wvc600_id,":")
print(command_total_hex)

# Open serial communication port at 9600 bits/sec (default 8 data-bits, no parity)
serial_comm_port = serial.Serial(port=computer_COM_port, baudrate=9600, timeout=1, write_timeout=1)
serial_comm_port.write(command_to_wvc600)
wvc600_response = serial_comm_port.read(28)
serial_comm_port.close()

if len(wvc600_response) == 28:
  wvc600_hex_output = wvc600_response.hex()
```

```
    print()
    print("WVC-600 inverter-response (hexadecimal): ")
    print(wvc600_hex_output)
    print()
    input("Press Enter to continue")
    print()
    print("WVC-600 inverter measurement-data decoded from response")
    Vdc_hex = wvc600_hex_output[28:32]
    Vdc = int(Vdc_hex,16)/100
    print("Vdc  =",Vdc,"V")
    Vac_hex = wvc600_hex_output[32:36]
    Vac = int(Vac_hex,16)/100
    print("Vac  =",Vac,"V")
    Idc_hex = wvc600_hex_output[36:40]
    Idc = int(Idc_hex,16)/100
    print("Idc  =",Idc,"A")
    Etot_hex = wvc600_hex_output[40:44]
    Etot = int(Etot_hex,16)/100
    print("Etot =",Etot,"kWh")
    Pout_hex = wvc600_hex_output[48:52]
    Pout = int(Pout_hex,16)/10
    print("Pout =",Pout,"W")
    Temp_hex = wvc600_hex_output[52:56]
    Temp = int(Temp_hex,16)/100
    print("Temp =",Temp,"gr.C")
else:
    print()
    print("WVC-600 inverter did not respond correctly!")
    print("Possible causes:")
    print("--Wrong WVC-600 inverter-id used")
    print("--Wrong com-port used")
    print("--Malfunction in data-communication system")
    print("--Insufficient power from the solar panels")

print()
input("Press Enter to continue")
```

The program will first test the response-code of the WVC-600 solar-inverter to the request-code sent by the RF data-communication system. Next, if the response-code is correct, the inverter measurement-data will be decoded from the response-code.

Main software

Installation/operation Python-program “wvc600-mon.py”

When testing of the RF data-communication system has been successful completed, the installation/operation of the main Python-program “wvc600-mon.py” can be started.

Because the WVC-600-inverter is only capable of communication via it’s RF-interface when the solar-panels provide power, the program has the following modes of operation:

- **awaiting-daylight-mode**. The program starts in this mode (or has returned from the night-mode), waiting for the solar-panels to provide enough power to start the WVC-600-inverter by checking communication with WVC-600-inverter every 15 minutes.
- **daylight-mode**. As soon as the WVC-600-inverter is able to communicate, the program goes from the awaiting-daylight-mode into the daylight-mode, periodically collecting the responses of the inverter (determined by the sample period of at least 30 seconds). From these responses, the measurement-data from the inverter is calculated and stored in a daily csv-file.
- **night-mode**. After receiving three error-responses in a row from the WVC-600-inverter (when the solar-panels are no longer producing power due to dusk) or the maximum monitoring time is exceeded, the program closes the daily csv-file, shut down all communication to the inverter and goes from the daylight-mode into the night-mode for the duration the night-time-delay, depending on the month-number (e.g. in December more than 13 hours). After this delay, the program returns to the awaiting-daylight-mode.

Program-text (blue marked text: comment-lines   yellow marked text: Python program code)

```
#===========================================================
#            Python3 program for monitoring
#
#    WVC-600 inverter via RF-interface on 433.400 MHz
#                          based on
#    USB-serial-module and HC-12 data-transceiver-module
#
# Author: Jaap van den Berg  Date: 27-DEC-2021  Version: 0.4
#
#===========================================================

#=====================================================================================
#---- Begin Input-data ---- Begin Input-data ---- Begin Input-data ---- Begin Input-data ---- Begin Input-data
#=====================================================================================
# !!!Check before running program!!!
#-----------------------------------

# Select applicable operating-system serial communication port -check first!-
# Windows: check Windows "configuration" -> "devices" for correct COM-port ("COM1", "COM2", "COM3",...)
# Linux:   check with command "ls -l /dev | grep USB" for correct device (e.g. "/dev/ttyUSB0")
computer_COM_port="/dev/ttyUSB0"

# Linux-note: Due to access-rights for "/dev/ttyUSB..", run program with "sudo python3 <program-name>.py"

# First part of CSV-file-name (without .csv file-extension: will be added including date-stamp) for monitoring-data
#   (csv-file used in append-mode: adding new records to the already existing csv-file)
#   (if no path to folder is included, current program-folder will be used)
#   (!beware: \T, \t, \N, \n, \R, \r are escape-characters, then use \\T, \\n, etc. instead! (example: "c:\\New-folder..) )
csv_filename_firstpart = "/wvc600-archive/wvc600-monitoring-data"

# WVC600-inverter-Id (on the sticker at the front of the WVC-600 inverter)
wvc600_id = "12345678"

# Maximum monitoring-time in daylight-mode (in minutes):
maximum_monitoring_time = 1200

# Time (sample period) between the monitoring-samples in daylight-mode
# (in seconds -preferably no less than 30-):
time_between_samples = 30

# Number of days to run program (to prevent that the program will run forever…):
number_of_days = 400

#========================================================================================
#---- End Input-data ---- End Input-data ---- End Input-data ---- End Input-data ---- End Input-data
#========================================================================================

# Import of necessary Python-modules
import time
import serial
import csv
import sys

# Construction of command to be sent to the WVC-600-inverter
command_firstpart_hex = "f20000000000000000"
command_lastpart_hex = "65"
command_total_hex = command_firstpart_hex + wvc600_id + command_lastpart_hex
command_to_wvc600 = bytes.fromhex(command_total_hex)

# Run the complete program for a maximum number of days
for day in range(number_of_days):

# Start program in awaiting-daylight-mode

# Open serial communication port at 9600 bits/sec (default 8 data-bits, no parity)
  serial_comm_port = serial.Serial(port=computer_COM_port, baudrate=9600, timeout=1, write_timeout=1)

# Set response and power-output from wvc-600-inverter to zero
  wvc600_response = "0"
  Pout = 0

# As long as there is no power-output > 0 response from wvc-600-inverter, stay in
# awaiting-daylight-mode while-loop and check communication with WVC-600-inverter every 15 minutes
  while Pout == 0:
# Send command to wvc-600-inverter
    serial_comm_port.write(command_to_wvc600)
# Receive the 28 bytes response from wvc-600-inverter
    wvc600_response = serial_comm_port.read(28)
    if len(wvc600_response) == 28:
      wvc600_hex_output = wvc600_response.hex()
      Pout_hex = wvc600_hex_output[48:52]
      Pout = int(Pout_hex,16) / 10
    time.sleep(900)
# End of awaiting-daylight-mode while-loop


# Program now continues in daylight-mode

# Construction of the complete daily csv-file name (including date-stamp)
  from time import gmtime, strftime, localtime
  date_stamp=strftime("-%Y-%m-%d", localtime())
  file_name_extension = ".csv"
  csv_filename = csv_filename_firstpart + date_stamp + file_name_extension

# Activation of a new csv-file for a new day that will be used in
# append-mode (adding new records to the already existing csv-file)
# Check if csv_file already exists. If not, create it
  try:
    csv_file=open(csv_filename, 'rt')
    csv_file.close()
    csv_file=open(csv_filename, 'at', newline='', encoding="utf-8")
    writer = csv.writer(csv_file, dialect='excel', delimiter=';', quoting=csv.QUOTE_NONNUMERIC)
  except IOError:
# Creation of csv-file if necessary
    csv_file=open(csv_filename, 'wt', newline='', encoding="utf-8")
    writer = csv.writer(csv_file, dialect='excel', delimiter=';', quoting=csv.QUOTE_NONNUMERIC)

# Wait another 3 minutes to guarantee adequate day-light-level for more stable operation of
# the wvc-600-inverter in daylight-mode
  time.sleep(180)

# Calculation of the maximum number of samples during daylight-mode based on
# maximum monitoring time and time between samples (sample period)
```

```
  maximum_number_of_samples = int(60 * maximum_monitoring_time / time_between_samples)

# Set the number of executed samples and the number of error-responses of WVC-600-inverter to zero
  number_of_executed_samples = 0
  number_of_error_responses = 0

# Start monitoring of WVC-600 inverter in the daylight-mode while-loop, as long as the WVC-600-inverter
# is producing no more than 3 error responses in a row (when the solar-panels end their power output) and
# the number of executed samples is less than the maximum number of samples
  while number_of_error_responses <= 2 and number_of_executed_samples <= maximum_number_of_samples:

# Send command to wvc-600-inverter
    serial_comm_port.write(command_to_wvc600)

# Receive the 28 bytes response from wvc-600-inverter
    wvc600_response_new = serial_comm_port.read(28)

# Check on valid response from WVC-600: if no valid response (length not equal to 28 bytes),
# keep the last response and increase number of error-responses
    if len(wvc600_response_new) == 28:
      wvc600_response = wvc600_response_new
      number_of_error_responses = 0
    else:
      number_of_error_responses = number_of_error_responses + 1

# Increase number of executed samples
    number_of_executed_samples = number_of_executed_samples + 1

# Convert response from bytes to hex
    wvc600_hex_output = wvc600_response.hex()

# Calculation of WVC-600-inverter measurement-data from hex-output

# Calculation of solar-panels Vdc-in
    Vdc_hex = wvc600_hex_output[28:32]
    Vdc = int(Vdc_hex,16) / 100

# Calculation of output-voltage to grid Vac-out
    Vac_hex = wvc600_hex_output[32:36]
    Vac = int(Vac_hex,16) / 100

# Calculation of solar-panels Idc-in
    Idc_hex = wvc600_hex_output[36:40]
    Idc = int(Idc_hex,16) / 100

# Calculation of total energy generated by WVC-600-inverter E-total
    Etot_hex = wvc600_hex_output[40:44]
    Etot = int(Etot_hex,16) / 100

# Calculation of output power to grid P-out
    Pout_hex = wvc600_hex_output[48:52]
    Pout = int(Pout_hex,16) / 10

# Calculation of temperature of WVC-600-inverter Temp
    Temp_hex = wvc600_hex_output[52:56]
    Temp = int(Temp_hex,16) / 100

# Construction of date-stamp:
    from time import gmtime, strftime, localtime
    date_stamp=strftime("%Y-%m-%d", localtime())

# Construction of time-stamp:
    from time import gmtime, strftime, localtime
    time_stamp=strftime("%H:%M:%S", localtime())

# Append calculated measurement data to csv-file:
    writer.writerow([date_stamp,time_stamp,Vdc,Idc,Vac,Pout,Temp,Etot])

# Wait for executing next sample
    time.sleep(time_between_samples)

# End of the day-mode while-loop (after more than 3 error responses in a row or exceeding the maximum number of samples)

# After one minute, close serial communication port and csv-file
  time.sleep(60)
  serial_comm_port.close()
  csv_file.close()
```

```
# Program now continues in night-mode

# Provide a night-time-delay (length in seconds, depending on the actual month-number) in which communication
# to the WVC-600-inverter is shut down in order to avoid needless RF-transmissions during the night

  def night_time_delay(i):
        switcher={
                1:47700,
                2:41400,
                3:34200,
                4:27000,
                5:20700,
                6:17100,
                7:18900,
                8:25200,
                9:32400,
                10:39600,
                11:45900,
                12:49500
              }
        return switcher.get(i,18000)
  from time import gmtime, strftime, localtime
  month_number=int(strftime("%m", localtime()))
  time.sleep(night_time_delay(month_number))

# After this nigh-time-delay, return to awaiting-daylight-mode


# Program will end anyway after the number_of_days of operation
```

Executing the Python program permanently in Linux based systems (e.g. Raspberry Pi)

Based on the name of the Python program **wvc600-mon.py** and stored in the folder **/wvc600-folder** on the Linux based system (like e.g. the Raspberry Pi), the program can be simply executed with the command (e.g. in the shell via Putty terminal):

```
sudo /usr/bin/python3 /wvc600-folder/wvc600-mon.py
```

But after logging out of the shell, the program has to be terminated…

However, on Linux based systems, Python programs can be also executed permanently in the background ("daemonized"). This can be accomplished as follows:

# First create a unit file in the folder /lib/systemd/system/:

```
cd /lib/systemd/system/
sudo nano wvc600-monitor.service
```

# Enter the following lines:

```
[Unit]
Description=WVC600 monitoring service
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /wvc600-folder/wvc600-mon.py

[Install]
```

```
WantedBy=multi-user.target
```

```
# Set the permissions of wvc600-monitor.service:
sudo chmod 644 /lib/systemd/system/wvc600-monitor.service

# Reload-command at any kind of change to any unit file:
sudo systemctl daemon-reload

# Start the service:
sudo systemctl start wvc600-monitor.service

# Stop the service:
sudo systemctl stop wvc600-monitor.service

# View status:
systemctl status wvc600-monitor.service
```

<u>As an option</u>, the service can be enabled to start automatically on system-boot:

```
sudo systemctl enable wvc600-monitor.service
```