



universität
wien

MASTERARBEIT

Titel der Masterarbeit

„Implementation of a Curve Based Realtime Photovoltaic
String Simulation “

verfasst von

Reinhard Mondl, BSc

angestrebter akademischer Grad

Master of Science (MSc)

Wien, 2015

Studienkennzahl lt. Studienblatt:

A 066876

Studienrichtung lt. Studienblatt:

Masterstudium Physik

Betreut von:

Ass. Prof. Dr. Viktor Schlosser

Abstract

In the field of work of setting standards for photovoltaic solar panels testing procedures are a main topic for investigating influencing effects on the modules durability. For many of this testing procedures simulation of working conditions is of importance. Simulation of working conditions in the lab has the big advantage that experiments can be done under controlled conditions whereas in the fields its almost impossible to get same conditions at different times to compare testing results. Without the possibility of controlled working conditions it's hard to analyze the yielded measurement data with respect to effects of definite disturbances. So therefore simulation of working conditions is a big aid in analyzing dependencies of photovoltaic modules on environmental conditions as well as technical failures.

In this thesis an experimental setup for testing curve behavior of photovoltaic arrays which include defect modules is presented. Therefore a Power Hardware In the Loop RealTime simulation is used to simulate the proper modules of a single photovoltaic string whereas defect modules are implemented in the circuit as real hardware components. For string simulation a software-hardware combination is used composed of a **Matlab Simulink** model of the string and a voltage controlled current source .

The capability of the setup with respect to simulate the current-voltage characteristics of modules and strings is shown. Further some comparative current-voltage characteristics and thermography of the investigated modules front side under various operating conditions are shown.

Zusammenfassung

Im Bereich der Standardisierung von Photovoltaik-Modulen stellt die Simulation von Arbeitsbedingungen eine Notwendigkeit dar. Durch testen der Module unter realen Bedingungen kann eine Entscheidung über die Qualifikation der jeweiligen Technologie für den realen und dauerhaften Einsatz getroffen werden. Um vergleichbare Resultate im Test von Modulen zu bekommen müssen diese möglichst ähnlichen Bedingungen ausgesetzt werden. Dies stellt jedoch im freien Feld aufgrund des stets wechselnden Wetters eine Unmöglichkeit dar. Daher ist man hier auf die Simulation von Umweltbedingungen angewiesen mittels welcher die Experimente reproduzierbar werden. Der Einsatz reproduzierbarer Arbeitsbedingungen ermöglicht zusätzlich eine genauere Untersuchung der zeitlichen Entwicklung von Moduldefekten und wie diese von den Umweltbedingungen abhängt.

In dieser Arbeit wird ein Testaufbau präsentiert um den Einfluss von einzelnen defekten Modulen auf die Leistungsabgabe von ganzen Strängen zu untersuchen. Dazu werden die intakten Module des Stranges mittels einer Power-Hardware-In-the-Loop Echtzeit-Simulation ersetzt während defekte Module als reale Komponenten in den simulierten Strang implementiert werden.

Für die Strang-Simulation wird eine Software-Hardware-Kombination verwendet welche sich aus einem `Matlab Simulink` Model und einer spannungsgesteuerten Stromquelle zusammensetzt.

Die Tauglichkeit des Testaufbaues bezüglich der Simulation von Strom-Spannung-Kennlinien von Modulen und Strängen wird dargestellt. Des Weiteren werden vergleichbare Strom-Spannung-Kennlinien und Infrarot-Aufnahmen der Modul-Vorderseiten unter verschiedenen Simulationsbedingungen gezeigt.

Contents

1	Introduction	9
2	Basics of Photovoltaic Solar Cells	11
2.1	Band Structure	11
2.2	Semiconductors	12
2.3	pn-Junction	13
2.4	Photovoltaic Solar Cells	14
2.5	Air Mass	15
2.6	Standard Conditions (STC)	16
3	Power Hardware In the Loop Simulation of a Photovoltaic Array	17
3.1	Terms	17
3.1.1	Hardware In the Loop (HIL)	17
3.1.2	Power Hardware In the Loop (PHIL)	17
3.2	Simulation Setup	18
3.2.1	Simulation Software	19
3.2.1.1	The Simulink Model	21
3.2.1.2	Module Modeling	23
3.2.1.3	Modeling via PVP MC-Library	24
3.2.1.4	Modeling via Current-Voltage-Curve	26
3.2.1.5	IV-Curve Datafile Format	26
3.2.1.6	The Graphical User Interface	27
3.2.2	Data Acquisition (DAQ, Signal Acquisition)	30
3.2.3	Temperature Measurement	30
3.2.4	Irradiance Measurement	31
3.2.5	PVAS Voltage Measurement	32
3.2.6	Signal Amplification	32
3.2.7	Power Signal Amplifier (PhotoVoltaicArraySimulator PVAS)	34
3.2.8	Photovoltaic Modules	35
3.2.9	Electrical Load	35
3.3	Configuration of Target and host-PC	36
3.3.1	Configuration of target-PC (xPC-Target)	37
3.3.2	Configuration of host-PC	38
4	Experimental Setup and Illustrative Tests	39
4.1	Modules Under Investigation	40
4.2	Simulation Configurations	44
4.3	Measurement Results and Discussion	44
5	Conclusion	64
	Bibliography	65

A	Appendix	66
A.1	List of Used Abbreviations	66
A.2	Amplifier Circuitry Schematics	67
A.3	Gauging of Amplifier Boards, Input to Output	70
A.4	Variable Resistor , Capacitance and Inductivity	72
A.5	Simulink Block Diagrams	73
A.6	Matlab Code Graphical User Interface	76
A.7	Matlab Code Embedded Matlab Function <i>CurveOrParam</i>	109
A.8	Matlab Code Matlab Function <i>Radiation</i>	117
B	Curriculum Vitae	118

List of Figures

2.1	Formation of energy bands	11
2.2	Energy band gap of materials	13
2.3	Depletion region	14
2.4	Principle of solar cell	14
2.5	IV-curve of solar cell	15
2.6	Spectral intensity, air mass	16
3.1	PHIL simulation setup	18
3.2	Scheme of the closed control loop of the PHIL simulation	19
3.3	Blockdiagram of pv_daq_neu	21
3.4	Target screen after downloading the Simulink model	22
3.5	Block diagram of pv_daq_neu/Module	23
3.6	Equivalent circuit of the single diode model of a PV module. ¹	25
3.7	IV-CurveFile	27
3.8	Graphical User Interface	28
3.9	Sequence for simulation and datalog format	29
3.10	Data Acquisition Devices from National Instruments	30
3.11	Reference cell and series resistance	31
3.12	Amplifier circuitry version3	31
3.13	Differential probe	32
3.14	PVAS front-,back-, inside view	34
3.15	Variable resistor	35
3.16	Configuration dialog for the boot drive of the target-PC	37
3.17	Target-PC screen after booting from USB drive	38
4.1	Mounting of infrared cam and anti reflection sheets	39
4.2	Experimental setup in the lab	40
4.3	Tested modules	41
4.4	Comparison IV curves measured, simulated and corrected Atersa module	45
4.5	Comparison IV curves measured, simulated and corrected Auversun modules	46
4.6	IV-diagrams of the Atersa module(DH, 1+1)	48
4.7	IR-images of the Atersa module(DH, 1+1)	49
4.8	IV-diagrams of the Atersa module (DH, 3+1)	50
4.9	IR-images of the Atersa module (DH, 3+1)	51
4.10	IV-diagrams of the Atersa module (DH, 5+1)	52
4.11	IR-images of the Atersa module (DH, 5+1)	53
4.12	IV-diagrams of the Atersa module (DH, 5+1, shadowing)	54
4.13	IR-images of the Atersa module (DH, 5+1, shadowing)	55
4.14	IV-diagrams of the Auversun module (DH, 5+1)	56
4.15	IR-images of the Auversun module (DH, 5+1)	57
4.16	IV-diagrams of the Auversun module (DH, 5+1, shadowing)	58
4.17	IR-images of the Auversun module (DH, 5+1, shadowing)	59
4.18	IV-diagrams of the Auversun module (TC, 5+1)	60
4.19	IR-images of the Auversun module (TC, 5+1)	61
4.20	IV-diagrams of the Auversun module (TC, 5+1, shadowing)	62
4.21	IR-images of the Auversun module (TC, 5+1, shadowing)	63
A.1	Amplifier boards	67

A.2	Circuitry diagrams of the amplifier boards	69
A.3	Blockdiagram of the simulation	73
A.4	Blockdiagram for irradiation and temperature transitions	74
A.5	Blockdiagrams for module modeling and irradiation calibration	75

List of Tables

4.1	Simulation Configurations	43
-----	-------------------------------------	----

Chapter 1

Introduction

Examination and prediction of PhotoVoltaic Power Plant (PVPP) output is an important topic today due to increasing capacities of installed photovoltaic (PV) arrays. There are several long term and short term issues influencing the power output of PVPPs which have to be compensated by other power sources/sinks of the smartgrid. Some of those issues for example are:

- *short-term issues*: changing irradiation and temperature conditions because of weather, shadowing and pollution,...
- *long-term issues*: delamination of antireflection layer and encapsulation, discoloring of encapsulation due to heating of broken cells or soldering joints, potential-induced degradation, cell breakage, ...

The experimental setup described in the following focuses on the long term issues. In the open field it's very hard to examine such issues because there is no control about the environmental conditions which influences the characteristics of the modules. So it's almost impossible to get same conditions over long term examination of photovoltaic array characteristics. This makes it hard to get comparable data of measurings over long time periods to examine long term issues of PV arrays which are also important for prediction of PVPP output and in time maintenance. The idea is to give - by means of the Power Hardware In the Loop (PHIL) simulation - a possibility to analyze in detail the defects of PV modules and their losses under realistic but reproducible operating conditions to support future modeling.

In this master thesis a Power Hardware In the Loop Simulation is set up to examine the behavior of photovoltaic arrays including defect modules with respect to current-voltage characteristics and infrared thermography. While the defect modules are given by real modules with well known defects the rest of the array, containing only proper modules, is simulated by a software controlled high power current source. Due to the possibility to simulate arbitrary current-voltage characteristics also strings of defect modules can be simulated but all modules containing the same defect characteristics.

The controlling software is implemented in **Matlab Simulink**, which provides a graphical programming environment, and *Matlab*. The hardware part of the simulation is represented by an PhotoVoltaic Array Simulator (PVAS, voltage-controlled current source), manufactured by Austrian Institute of Technology, which provides a range of operation from 0 - 800 Volts at 0 - 32 Amperes. The connecting interface between PC (software) and the PVAS is managed by a PCI card (**National Instruments PCI 6229**) in combination with a BNC-connector box (**National Instruments BNC 2110**) for easy connection via BNC-plugs. The software operates the PVAS in a manner that it behaves like a module or a series of modules under preset conditions derived from real irradiance and temperature. The measured voltage drop over the amplifier is used for calculations determining the corresponding current regarding to the preset module IV-characteristics. The software provides a big variety of modules which can be used for simulation. Selection from a big photovoltaic module database (SandiaModuleDatabase) for

supplying the software with necessary parameters for module modeling is also provided as the usage of a measured IV-curve to be simulated.

The software of the PHIL setup is divided in two subparts. The first part is represented by a Matlab Simulink model which model the photovoltaic string under desired conditions. It runs on a separated PC, called target-PC, to provide an optimum simulation performance. The second part of the software is the Graphical User Interface (GUI). It depicts the control panel of the simulation so the user can change parameters on the model or start/stop the simulation. The GUI is executed on the host-PC and is connected to the target-PC via an ethernet connection for exchange of data and commands.

The PHIL setup allows changes in irradiance, module temperature (or ambient temperature) and on the module type which should get simulated. Also an opportunity for logging the data has been implemented.

The setup gets tested with respect to agreement of simulated current-voltage characteristics and measured current-voltage characteristics from real modules. Infrared thermography is performed on the modules after 20min of operation in specific operation points for comparison with data from the fields.

Goal of this thesis is to proper simulate strings under real life conditions with specific defects of single modules. This thesis restricts on the implementation of the PHIL setup and testing of it functionality to enable further investigations on module defects in real life strings.

Chapter 2

Basics of Photovoltaic Solar Cells

¹Photovoltaic cells are a big player in the topic of renewable energies. Even if there are several other techniques for generation of renewable energy, this chapter is restricted on photovoltaic cells for reason of the topic of this thesis. For explaining the concept of photovoltaic cells, it will refer to the silicon cell technology, which is also the most common cell technology. The principle of other photovoltaic cell technologies, such as CIGS or GaAs, is the same.

For an understanding of the photovoltaic principle it is necessary to explain some basic terms

2.1 Band Structure

Solving the Schrödinger equation for an electron in the potential of a single atom core yields discrete energy levels which the electron is able to occupy. The number of these energy levels is infinite but discrete. Each energy level hosts at least one state but has a finite maximum of states, increasing with the energy level. The number of electrons per state is limited to 2 due to Fermi's law, which states that electrons can never have exactly the same state. Accordingly the two electrons possible per state must have opposed spins. So the number of electrons per energy level is equal to the number of states per energy level multiplied by a factor of 2 (spin up/down). Considering a second identical atom, far away from the first one, it will show the same energy levels and states for its electrons. By approaching these two atoms will result in a small shift of the energy levels due to Fermi's law, so each energy level splits up into 2 very close energy levels. Spinning this idea further to the approximation of many atoms, like it is the case in bulk material, the split up of same energy levels results in narrow bands of energy levels, called energy bands. The collectivity of these energy bands is then called the band structure of the specific material. The energy levels/bands get occupied by the electrons in a sense of lowest

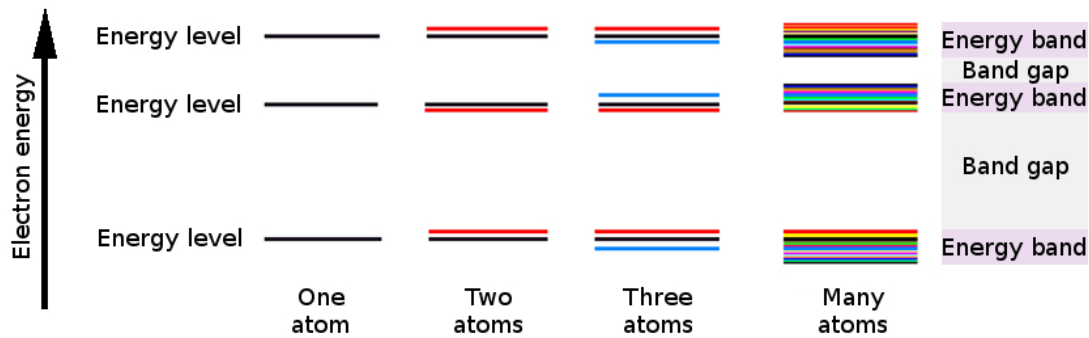


Figure 2.1: The state of an electron in a single atom is restricted to discrete energy levels (very left). When approaching identical atoms, these energy levels must split up due to Fermi's law. The shift in energy of the energy levels is small, compared to the distance between the discrete energy levels of a single atom. Approaching many atoms this will result in bands of energy levels for the hosted electrons (very right).²

¹Information for the chapter **Basics of Photovoltaic Solar Cells** was taken from ref. [9]

²Figure was taken from ref. [10]

energy, in strict accordance of Fermi's law. The gaps between the energy bands are forbidden states for the electrons and are referred to as *band gaps*.

2.2 Semiconductors

Bulk material is usually classified in 3 categories: insulators, metals and semiconductors. The crucial property for this classification is the electric resistivity of the material, which is determined by the band structure and its filling level and varies by many orders of magnitude. As mentioned in section 2.1, the filling of the states for the electrons in the band structure is from lower energies to higher energies. Due to the finite number of electrons per atom, the filling of the states has an upper border, referred to as highest occupied molecular orbital (HOMO). At absolute zero temperature, states beneath this HOMO will be occupied with electrons, states above are left empty. At non-zero temperature also states higher than the HOMO can be occupied due to gaining thermal energy. Then also some states beneath the HOMO will be left unoccupied. The highest totally filled energy band is called valence band whereas the lowest not totally filled band is called conduction band. The combination of possible orbitals (band structure) and their filling levels give rise to the classification:

Insulators: They have a totally filled valence band and an empty conduction band. The band gap between valence and conduction band is far bigger (several eV) than the average thermal energy, which is in the range of 25meV at 300°K. If an external electrical field gets applied to an insulator, the electrons in the valence band are unable to gain energy, because there are no unoccupied higher energy levels close by. Therefore the electrons can not gain velocity fed by the external field and further the material shows no conductivity.

Metals: In metals the HOMO lies in between the conduction band (partially filled). Therefore electrons in the conduction band can easily gain energy from the external field, because there are many unoccupied states close by and therefore metals have a high conductivity.

Semiconductors: In semiconductors the band gap between valence and conduction band is in the range of the average thermal energy up to a few eV. This means that a very few ¹ electrons are able to get excited into the conduction band through thermal energy. This results in free electron states close to the occupied states in the conduction band and the valence band both. So the conductivity is happening in both of the bands. But as the number of electrons excited to the conduction band is very low, the resulting conductivity is too. Typical semiconductors have a band gap of a few eV. Pure silicon has a band gap of 1.1eV at 300°K.

Unoccupied electron states in the valence band are also referred to as holes. Their behavior in external applied fields and with respect to occupation of electron states in the bands is opposed to that of electrons. The drift of holes through the material contributes to an electric current in the same way as electrons with an opposed direction of drift do. Free electrons and holes are also referred to as charge carriers (carriers of the electric current).

PV-cells are typically made from semiconductors because their band gap fits properly to the energy of light which should get transformed by them.

¹The probability $p(E)$ of an electron at temperature T [°K] to get excited to a higher state with energy E [eV] is proportional to $p(E) = \frac{1}{e^{\frac{E-\mu}{k_B T}} + 1}$, with μ [eV] as the chemical potential and k_B [eV/K] as the Boltzmann constant. In most of the cases μ can be approximated by the Fermi energy, which gives the energy of the highest occupied electron state at $T=0$. The number of electrons $n(E)$ at an energy E is the product of the probability $p(E)$ and the density of states $d(E)$: $n(E)=p(E) \cdot d(E)$.

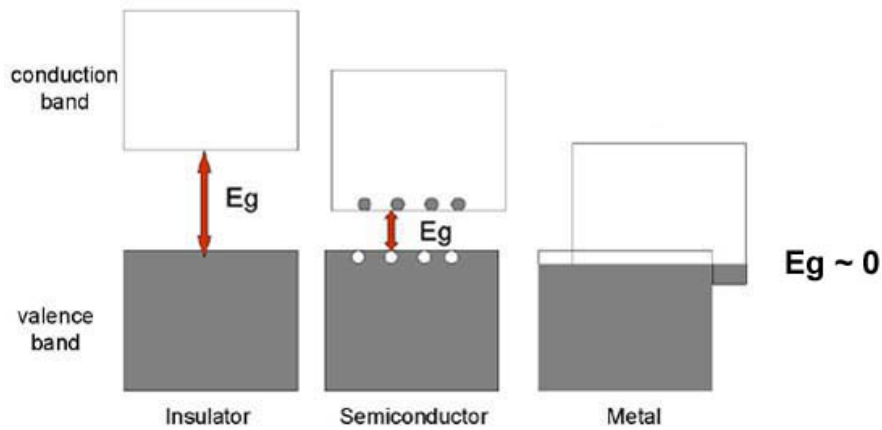


Figure 2.2: The distinction between insulator, semiconductor and metal is their size of the energy band gap E_g . In insulators the band gap is too big for thermal excitation of electrons into the conduction band, whereas in semiconductors it is small enough so that a few electrons can cross the band gap. In metals there is no band gap at all and therefore electrons are free to move.¹

2.3 pn-Junction

Silicon is a group-IV element in the periodic table, therefore it has 4 electrons in the outermost, partially filled shell. Electrons in a partially filled shell tend to create bonds with neighbouring atoms to get their partially filled shell filled up and decrease the energy of the overall system. In pure, bulk silicon every single atom creates 4 bonds to 4 neighbouring atoms. Doping silicon with a group-V element (5 valence electrons) will add extra electrons at the sites of the dopants and is so called n-doped. Because there are only very few unoccupied electron states in the valence band, most of the electrons will get excited to the conduction band and therefore can move free across the bulk material. Their movement from the dopants site will leave positive charged ions which are fixed in the crystal lattice. In a macroscopic scale the bulk material is still electric neutral. Similar happens when doping silicon with group-III elements. The dopants then will add some extra holes (p-doped) at the dopants site. These holes also can move in the lattice like the free electrons even with a different mobility. So in this case negative charged ions will be left at the fixed dopants site.

In undoped material the number of holes in the valence band is equal to the number of electrons in the conduction band. Whereas in doped material this equilibrium gets disturbed. Considering n-doped silicon, the adding of extra electrons decreases the number of holes in the valence band. At the same time the number of electrons in the conduction band gets increased. Thus in n-doped silicon there is an excess of free electrons as a carrier of charge whereas in p-doped silicon there is an excess of holes. Charge carriers with the higher/lower concentration level are referred to as majority/minority carriers respectively.

A pn-junction is created by joining p-doped and n-doped materials. As in the n-doped material there is an excess of electrons and in the p-doped material an excess of holes, by joining them the free electrons and holes will diffuse into the other material. Because of the left behind ion cores an electrical field will build up between the two different doped regions. Free electrons and holes will be swept out fast from this region by the electric field, therefore there is a lack of free carriers in this region and it's called depletion region. The electric current due to the electric field is referred to as drift current whereas the current due to a concentration gradient of the carriers, is referred to as diffusion current. In equilibrium state of the junction these two current will compensate each other, so that there is no net current.

¹Figure was taken from ref. [11]

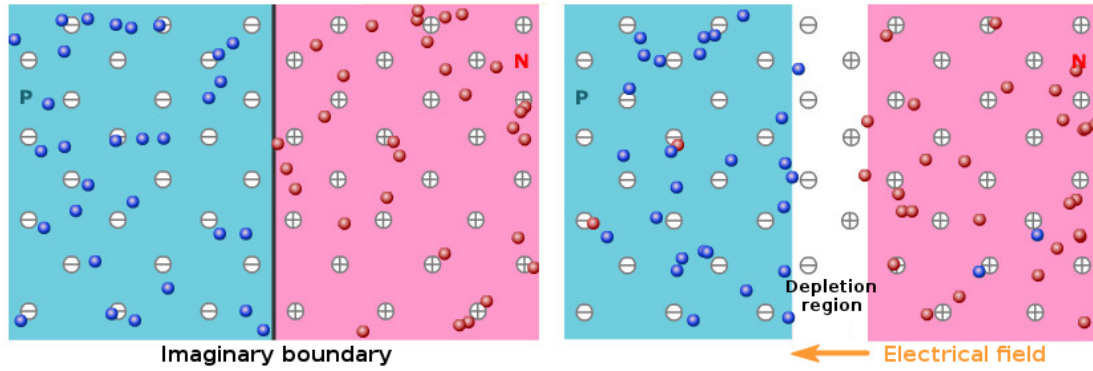


Figure 2.3: Formation of the depletion region: On the left hand side the doped materials get joined with an imaginary boundary between so that carriers are not able to pass into the opposing material. Removing this imaginary boundary the carriers will start to diffuse into the opposing region, leaving behind the fixed ion cores. The resulting electric field between the different doped regions leads to an depletion of carriers in the interfacing area.¹

2.4 Photovoltaic Solar Cells

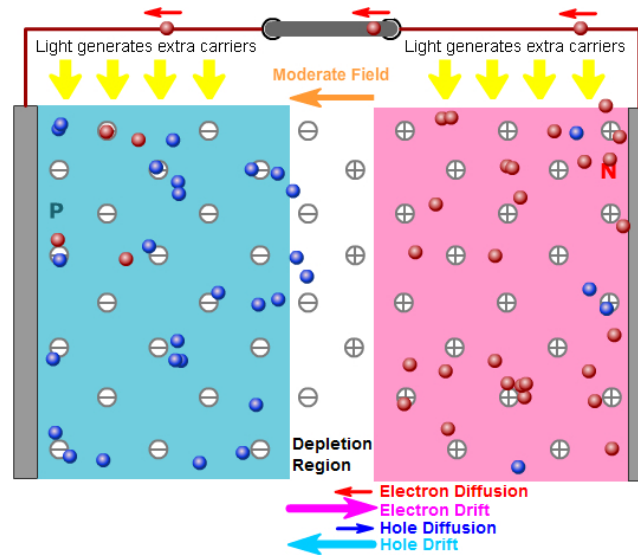


Figure 2.4: Principle of a solar cell: Incident light generates electron-hole pairs inside the cell. These carriers get separated by the intern electric field in the depletion zone according to their charge. This results in an excess of electrons in the n-doped region and holes in the p-doped region. Recombination of electrons and holes occurs through transfer of electrons via an external circuitry. Note: The carriers which get swept through the depletion zone by the electric field are minority carriers. Only if the minority carrier reaches the depletion zone before it recombines and gets swept to the opposing side, then a majority carrier (which gets rejected by the electric field of the depletion zone) is forced to take the path, to the opposing region, over the external circuit and hence will create an electric current there.²

A photovoltaic solar cell then consists of a pn-junction and additional contacts at the free surfaces of the p- and n-doped regions (see Fig.2.4). Incident photons with energies of at least the band gap energy will create free electron-hole pairs inside the silicon. The electric field inside the pn-junction causes them to separate into n-doped and p-doped regions respectively which will result in an additional electric field inside the pn-junction. The excess of electrons in the n-doped region with respect to the equilibrium state and holes in the p-doped region now can get equalized by transferring electrons per external circuitry to the p-doped region, driven by the additional electric field inside the pn-junction.

The relationship between the external current I and voltage V of a solar cell is described by the

¹Figure was taken from ref. [9]

²Figure was taken from ref. [9]

following equation:

$$I = I_L - I_0 \left(e^{\frac{qV}{nkT}} - 1 \right). \quad (2.1)$$

With I_L Light induced current
 I_0 Dark saturation current
 q Electron charge.
 n Diode ideality factor
 k Boltzmann's factor
 T Temperature at the pn-junction.

I_L , the light generated current, is the number of carriers generated per unit time inside the cell due to photon absorption. Not all of this generated carriers will contribute to the solar cells current output because of recombination inside the cell. Recombination describes the process of electron and hole meeting together and equalize each other, means they will cross the band gap into the other energy band, so they are no longer existent for the transport of charge. Recombination therefore is the inverse process to generation of charge carriers. It releases energy which can be either emission of a photon, excitation of other electrons or increase of the thermal energy of the crystal. To minimize this recombination rate is a main issue in designing PV-cells for increasing the cells efficiency. The amount of the lost current, due to recombination inside the cell in the absence of incident light, is described by the dark saturation current I_0 . The ideality factor n is a correction factor to the ideal diode equation to mirror real diodes behavior.

For a given current-voltage curve (IV-curve), the value of the current at zero voltage is referred to as short circuit current I_{SC} , whereas the voltage value at zero current is referred to as open circuit voltage V_{OC} . A typical IV-curve of solar cells is shown in figure 2.5.

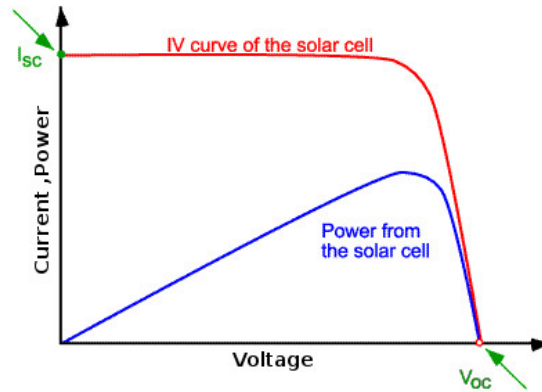


Figure 2.5: Typical current-voltage relation (red) of a solar cell. According to equation 2.1 the short cut circuit current I_{SC} is equal to the light induced current. The according power output as a function of the cell voltage is depicted as blue graph.¹

2.5 Air Mass

The air mass is a measure for the path length of the light passing through the atmosphere, normalized to the path length of light passing through the atmosphere, incident normal to the earths surface. With θ being the angle of incident radiation with respect to earth surfaces normal, the air mass is calculated through

$$AM = \frac{1}{\cos(\theta) + 0.50572 \cdot (96.07995 - \theta)^{-1.6364}} \quad (2.2)$$

This equation takes into account the atmospheric curvature. For small angles the curvature can be approximated by a plane and therefore the equation simplifies to

$$AM = \frac{1}{\cos(\theta)} \quad (2.3)$$

¹Figure was taken from ref. [9]

The air mass is an important factor for calculating the remained intensity of light from the sun after passing through the atmosphere. Loss in intensity is due to absorption and scattering in the earth's atmosphere.

For comparison of performance from solar cells of different techniques and at different locations, standard solar spectra are defined. These standard solar spectra are giving the spectral irradiances of the sun in W/m^2 at specific circumstances. The standardised solar spectrum AM0 gives the spectral intensity of the sun for extra terrestrial use. Whereas AM1.5 gives the spectral intensity for terrestrial use where the sunlight has to pass the earth's atmosphere and therefore absorption and scattering affects the incident light spectra. The AM1.5 is further divided in AM1.5D and AM1.5G which refers to the direct incident light only (AM1.5D) or the global incident light (AM1.5G) respectively. The AM1.5G includes the diffuse scattered light from the environment too.

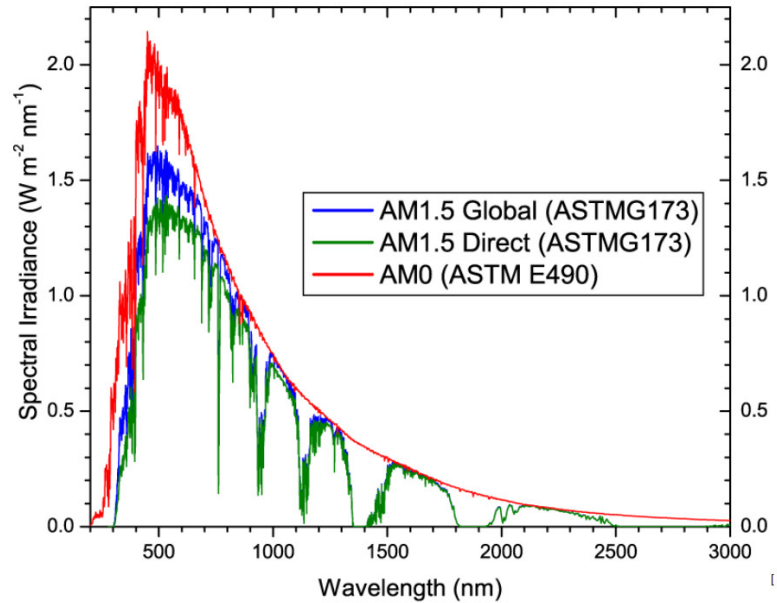


Figure 2.6: Spectral intensity gives the intensity of the incident irradiation as a function of the photons wavelengths. AM0 depicts the sun's spectral intensity outside the earth's atmosphere. Irradiation at the earth's surface is described by the AM1.5G/AM1.5D spectra which include absorption and scattering. The appendices "G" and "D" refer to Global and Direct irradiation.¹

2.6 Standard Conditions (STC)

For enabling comparison and classification of photovoltaic solar cells it is useful to declare referencing operation conditions at which the properties of the cell get determined. These so-called standard conditions are committed to:

- Spectrum of incident radiation according to global air mass 1.5 (AM1.5G) for terrestrial cells
- Intensity of incident radiation of 1000 W/m^2
- Cell temperature of 25°C

¹Figure was taken from ref. [9]

Chapter 3

Power Hardware In the Loop Simulation of a Photovoltaic Array

In the following the PHIL setup will be described. A general overview of the functionality of the experiment will be given as well as detailed explanations about the single components of the setup. First some general used terms are described.

3.1 Terms

3.1.1 Hardware In the Loop (HIL)

Hardware in the loop is a concept for testing and developing devices under real life conditions. It replaces parts of a real system by a software-hardware combination simulating this corresponding part of the system also referred as *rest of system* (ROS). The remaining part of the system is still represented by real system components which are also called *Hardware Under Test* (HUT) or *Device Under Test* (DUT). In most of the cases simulated rest of the system is implemented by a combination of software and hardware whereas a simulation just by hardware is also possible. A model of the software represented ROS runs on a PC in *real time simulation* calculating all the theoretically outputs of the ROS depending on the measured inputs. An appropriate interface transforms the virtually outputs of the ROS model into physical signals connected to the HUT. Therefore the HUT "sees" the same electrical environment as if connected to the real system. The concept of Hardware In the Loop (HIL) has big advantage of rapid and low cost prototyping because by modifying the software the software-hardware combination can be adapted to lots of different simulated systems without the need of an expensive full hardware model of the system.

3.1.2 Power Hardware In the Loop (PHIL)

The concept of *Power Hardware In the Loop* (PHIL) is the same as for HIL except the transformation of virtually outputs to physical signals. This step contains an additional process of amplifying the signals. For example the simulation of a high voltage device cannot be managed from any port of the simulating PC. Therefore an additional high voltage source is needed which gets controlled by the low voltage outputs possible at a conventional PC.

3.2 Simulation Setup

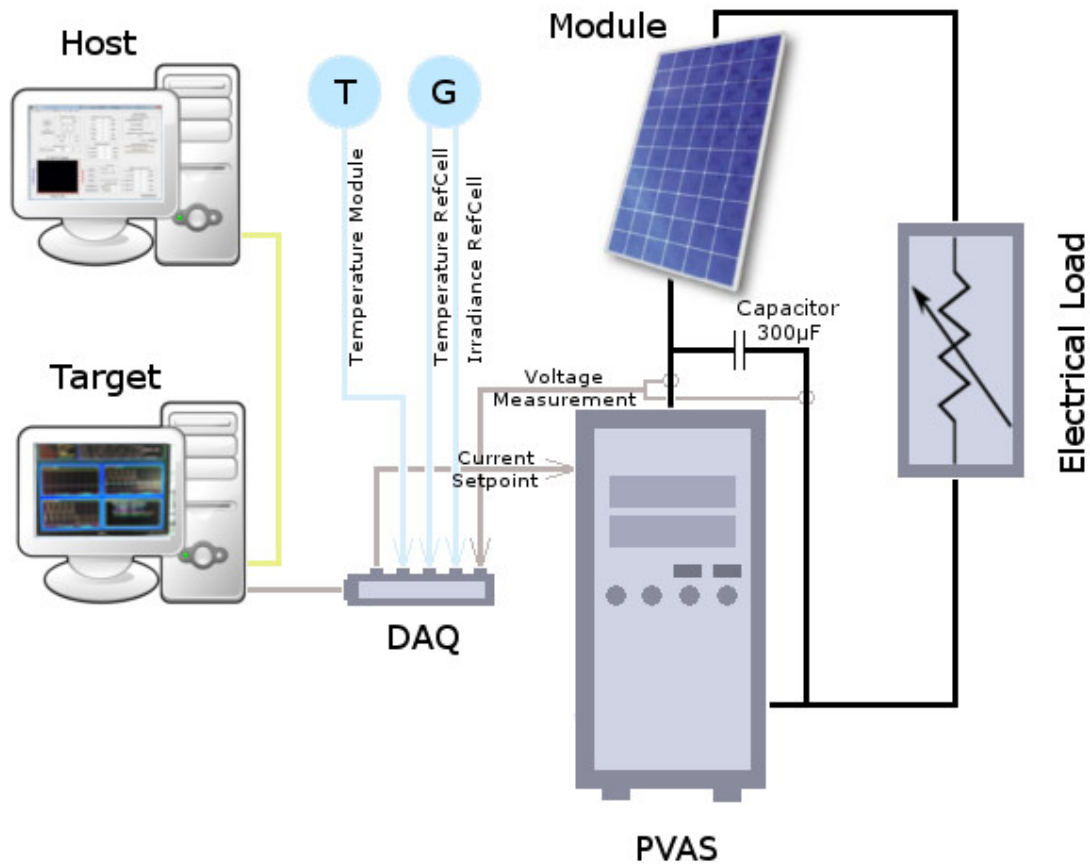


Figure 3.1: The PHIL simulation setup: From the host-PC the user controls the simulation program on the target-PC through ethernet connection. Via PCI-card (target-PC) and DAQ-Box the signals get transferred between the target-PC, PVAS and sensors for irradiation and temperature. The simulated part of the string (represented by simulation program and PVAS) is connected to the real module in series. Additionally an electrical load is connected in series for variation of the operating point. A 300μF capacitor in parallel to the PVAS avoids oscillations of the simulation in a critical region next to the open circuit point of the simulated IV-curve.

Figure 3.1 gives an overview draft of the created PHIL setup. PVAS, module and electrical load get connected in series. The PVAS acts as an voltage controlled ideal current source and forms the simulated part of the module string which is connected in series to the real part of the string represented by a single or a substring of modules. The experiment gets controlled from the host-PC which runs the GUI and also stores the Simulink model. After loading the Simulink model to the target-PC the model can be executed by the target-PC which offers an real time environment provided by **The Mathworks** and included in the Matlab version installed on the host-PC. The real time kernel gets loaded from a USB-drive and needs no other operating system installed on the target-PC. Commands and data are transferred to the target-PC via ethernet. Also requests from the host to the target get answered using the ethernet link. When the Simulink model on the target gets started from the host-PC control signals and measuring variables are in-/outputted via the DAQ-box. The voltage drop over the PVAS and signals from irradiance and temperature measurements are used for calculating a corresponding current value through an appropriate representation of the current-voltage characteristics of the modeled modules. This value then gets transferred to the PVAS which consequently regulates the driven current to the instructed value. Therefore the PVAS acts as a voltage controlled current source. Because the PVAS is in series with the module and the electrical load the driven current causes an voltage drop over the PVAS which is the inverted sum of the voltage drop over the module and the electrical load $V_{PVAS} = -(V_{Module} + V_{ElectricalLoad})$. This voltage drop again is used for the next calculation of current in addition with the other refreshed variables as module temperature and irradiance. This gives the following control loop for this setup:

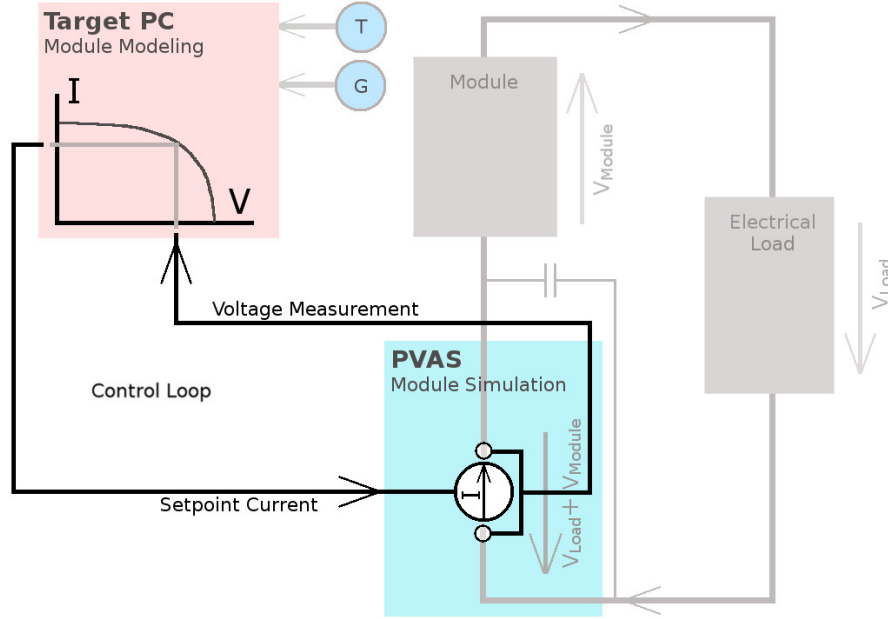


Figure 3.2: Scheme of the closed control loop of the PHIL simulation: The closed control loop is depicted in black. Through a differential probe the voltage drop over the PVAS gets measured. This voltage drop is the negative sum of the voltage drops over electrical load and module. The voltage value then gets transferred to the target-PC where it gets correlated to a current value according to the actual modelled IV-curve. Values from the temperature (T) and irradiance (G) measurement are considered in the IV-curve modelling. The current value gives the new set point for the current to get driven by the PVAS which acts as a voltage controlled current source. The new current value will cause a new voltage drop over the PVAS which again gets determined by the differential probe.

The control loop is depicted as black lines whereas additional parts of the setup are shown in gray. The IV-curve gets updated every simulation step which has a time interval of $100\mu\text{s}$. The loop is also depicted in figure 3.1 as gray lines to show the regarding connections in the setup scheme.

The capacitor of $300\mu\text{F}$ in figure 3.1 is necessary to avoid oscillatory instabilities for operating points at $\frac{I_{PVAS}}{V_{PVAS}} > \frac{I_{SC}}{V_{OC}}$ with V_{PVAS} and I_{PVAS} as the voltage drop and current over the PVAS and V_{OC} and I_{SC} as open circuit voltage and short circuit current of the simulated string at actual conditions, respectively.

For reasons of complete data generation and monitoring the voltage drop over the module was logged with an separated data logger.

3.2.1 Simulation Software

The software for this experiment is based on a former version created by Kishore Kumar Yamana[6] and written in **Matlab Simulink** which provides a graphically programming environment. To fulfill the new requirements the software got modified in terms of code for module modelling, applying live values for irradiance and temperature to the simulation, data logging have been implemented too. The code for module modeling offers now 2 opportunities to do so, modeling from a Matlab library and modeling via measured current-voltage curve, instead of the former fix implemented single diode model.

In Matlab Simulink mathematical and conceptual functions are represented as blocks in a diagram so called block diagram. The path of data transfer between the blocks is determined by connecting lines. For the sake of clarity group of blocks can be combined to a single block. For real time simulation a separated PC, running a RealTime kernel only, is used by reason of speed and steadiness. The kernel which executes the **Simulink**-model is provided by **Matlab**. The PC, running the real time kernel, is called *target-PC* and is connected to a BNC-Box (**National Instruments BNC-2110**), for input and output of signals, via PCI-card (**National Instruments PCI 6229**). The running model on the target-PC can be accessed and partial modified from a Graphical User Interface (GUI) on the host-PC connected to target-PC through

ethernet. The GUI allows control of the Simulink model in terms of start/stop the execution of the Simulink model on target-PC and modification of working conditions and module parameters.

3.2.1.1 The Simulink Model

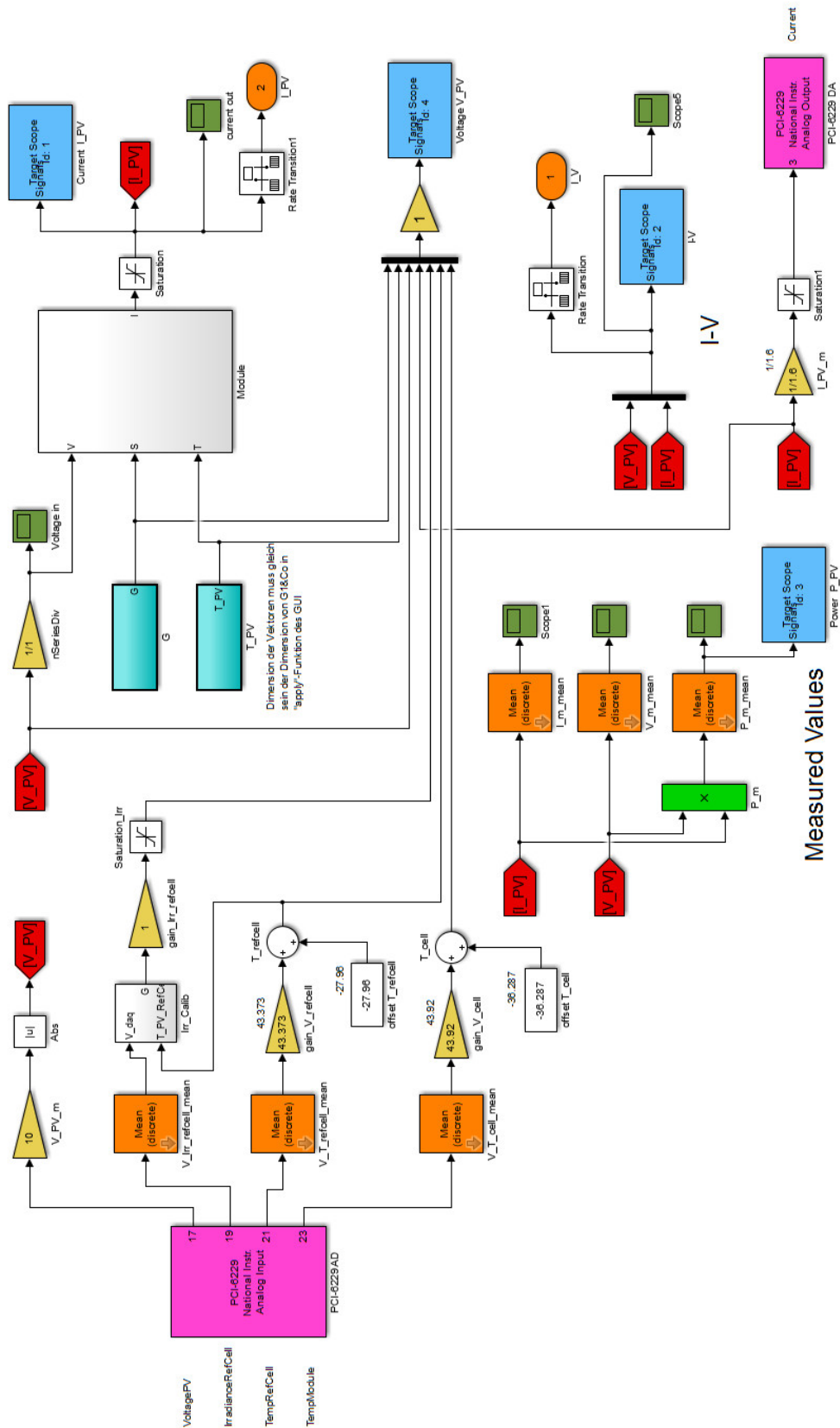


Figure 3.3: Block diagram of the Simulink model: The flow of the data is from the left to the right as indicated by the arrows. Single blocks are representing tasks or mathematical functions. Some of them are containing Matlab-code or further block diagrams.

Figure 3.3 shows the block diagram of the Simulink model which gets executed on the target-PC. The *PCI-6229 AD* block on the upper left (magenta) represents the input block. Signals applied to the DAQ-box are accessed by this block. All the blocks in the column neighbouring the input block are for signal conversions and calibrations. For example transferring the voltage signal of port 21 into a temperature value. The two cyan blocks in the upper middle *G* and *T_PV* are providing irradiance and temperature for the module modeling. They can be actual values from connected reference cell and temperature sensor or default values set from the GUI. The default use of the provided temperature value from the *T_PV* block is for the module temperature (backsheet attached temperature sensor). However, it can also be used as the ambient temperature input and the module temperature is calculated. Therefore the code in the *Module/CurveOrParam* block has to be slightly modified as commented in the lines of definition of *Tamb*, *Tmodule*, *Tcell*. The read in voltage value used for the module modeling (port 17 of the input block) is related to the voltage drop over the simulated PV-string (PVAS). The output of the *Module* block is the corresponding current value according to the irradiance, temperature and voltage values. This value for the current will be transformed to a voltage signal and outputted to the DAQ-box via the *PCI-6229 DA* block in the lower right (magenta, output block). The outputted voltage signal will control the PVAS's current output simulating the PV string. The driven current then again causes a new voltage drop over the PVAS according to the adjusted electrical load and therefore gives a new voltage value to be read in by the input block (port 17). So the control loop is closed. A single iteration of the loop must take less than $100\mu s$ which is the time step of the simulation.

The light blue blocks named *Target Scopes* are output blocks for the target-PC. Each of it causes a window on the target-PC's display showing the related value in a plot. The green scopes are only relevant for non real time mode so when testing the Simulink model on host-PC without using *RealTimeWorkshop*. When the Simulink model gets downloaded to the target, the target-PC's screen appears like this:



Figure 3.4: Target screen after downloading the Simulink model. Window 1: graph of the setpoint for current outputted from the DAQ-box; window 2: graph of set point for current and measured voltage drop over the PVAS inputted by the DAQ-box; window 3: power calculated from setpoint for current and PVASs voltage drop; window 4: numerical values from the running simulink model as irradiance, module/ambient temperature, voltage drop over PVAS and set point for current and read in and calibrated values from the sensors via DAQ-box as irradiance, temperature of reference cell and temperature of module/ambiance.

The two windows on the top are showing some actual simulation parameters like name of the loaded program file and sample time on the left side and command lines on the right side. The four windows below are oscilloscope-like and will draw graphs of current, current & voltage and power during the simulation (windows 1,2 and3) and numerical values of the actual graphs as

well as sensed values from the DAQ-box (window 4).

3.2.1.2 Module Modeling

The modeling of the module itself is accomplished by the *Module* block. Inside the *Module* block there is a further block diagram.

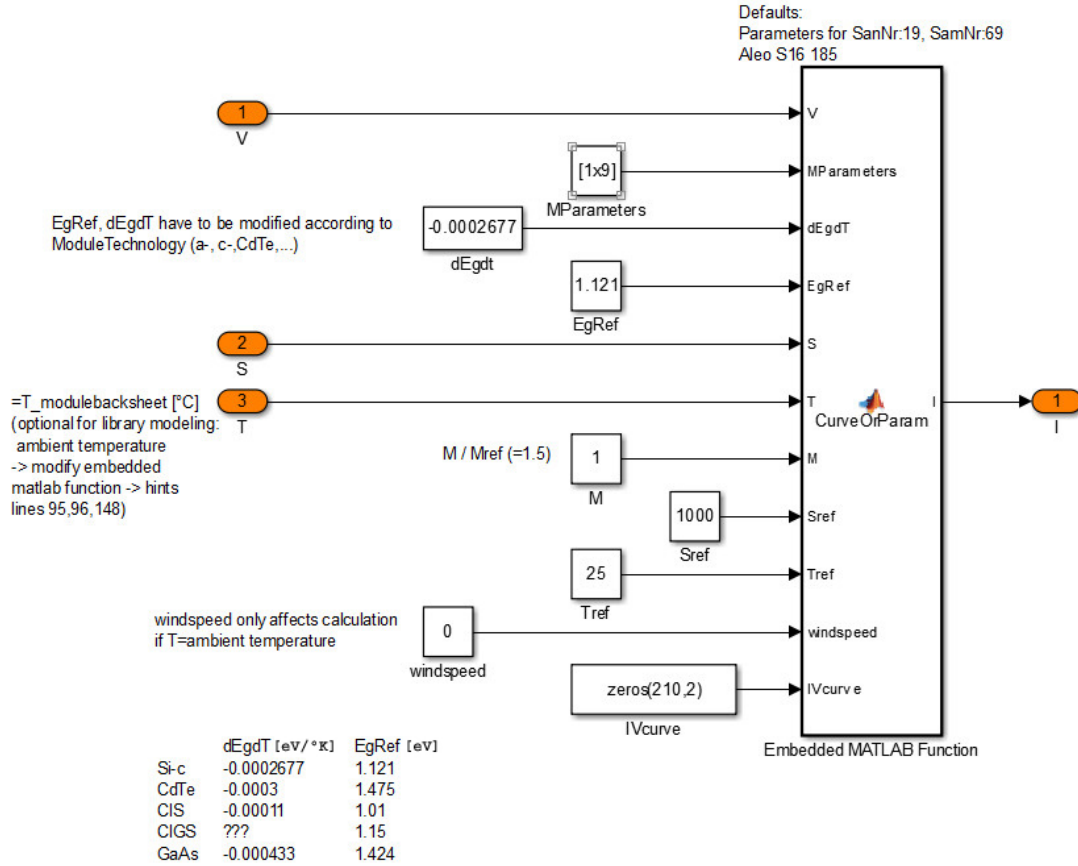


Figure 3.5: Blockdiagram of pv_daq_neu/Module: The orange blocks are transferred variables from and to the overlying block diagram. The small white blocks serving parameters for the modeling. Inside the *CurveOrParam* block there is conventional Matlab code for modeling the modules.

The block *Embedded MATLAB Function (CurveOrParam)* contains conventional matlab code modeling the PV-module. The code is designed so that it offers two opportunities to model a PV-module.

- modeling using a Matlab library provided by *PVPerformance Modeling Collaborative (PVPMC)*[7]
- modeling using a measured current-voltage curve (IV-curve) of the module

The modeling with PVPMCs library requires some parameters of the modules which are usually not available. PVPMC therefore offers excel sheets containing the required parameters for a huge number of modules but mostly for modules on the american market.

The second alternative allows to model the modules from a measured IV-curve. Transitions to other irradiance and temperature conditions than the measured ones are calculated by using the international standard IEC 60891 [2].

The small blocks on the left side of figure 3.5 provide some necessary parameters for the modeling. The orange blocks called *V*, *S* and *T* represent transfer values from the overlying block model. Blocks *MPParameters*, *dEgdt*, *EgRef*, *M*, *Sref*, *Tref* and *wind speed* are parameters for modeling via PVPMC library. The block *IVcurve* supplies all the parameters and data for modeling via IV-curve.

V [V] gives the measured voltage drop over the simulated string (PVAS) which is used for determining the corresponding current according to the module characteristics.

S, T [$W/m^2, ^\circ C$] are the values for irradiance and temperature measured from sensors connected to the DAQ-box. Temperature can be the value of the ambient temperature or module back-sheet depending on measurement. Default mode is the module back sheet temperature. In this configuration modeling via IV-curve as well as modeling via library is possible. If using ambient temperature the code of the module modeling block *Embedded MATLAB Function (CurveOrParam)* has to be modified slightly. Some lines have to be commented while others have to be uncommented. These lines are commented in the code where T_{amb} , T_{module} and T_{cell} gets defined. When using ambient temperature only modeling via library will provide good results. In modeling via IV-curve there is no correction for calculating module temperature from ambient temperature. So ambient temperature will be used as module temperature which can differ by tens of Celsius degrees and therefore will produce a maybe relevant error.

$MParameters$ gives some extracted module parameters from the PVPMC database (excel sheet) like light generated current [A], dark saturation current [A], modified diode ideality factor [V], series- and shunt resistance [Ω] and temperature coefficients [$x/^\circ K$].

$dEgdT$ [$eV/^\circ K$] gives the temperature dependency of the energy band gap. It's an material specific parameter and has to be changed manually for the various cell technologies. A small table in the bottom left of the block diagram provides values for the most common cell technologies (see fig.3.5).

$EgRef$ [eV] gives the energy band gap at reference conditions. It's an material specific parameter and has to be changed manually for the various cell technologies. A small table in the bottom left of the block diagram provides values for the most common cell technologies (see fig.3.5).

M gives an airmass modifier which is given by the ratio from local airmass over reference airmass of 1.5. So a local airmass of 1.5 gives an airmass modifier of 1.

$Sref$ [W/m^2] gives reference value of irradiation for parameters of PVPMC database ($1000 \frac{W}{m^2}$).

$Tref$ [$^\circ C$] gives reference value of temperature for parameters of PVPMC database ($25^\circ C$).

$windSpeed$ [m/s] gives the wind speed for correction of module temperature when ambient temperature is used as the input T to the *Embedded MATLAB Function*-block.

$IVcurve$ gives parameters and data points for modeling via measured IV-curve. This array is created by the GUI function *load module*. The specific format of this array is described in section 3.2.1.5.

The blocks $MParameters$, $dEgdT$, $EgRef$, M , $Sref$, $Tref$ and $windSpeed$ only affects the modeling via PVPMC library whereas $IVcurve$ only affects modeling via IV-curve.

Modeling in detail:

3.2.1.3 Modeling via PVPMC-Library

Modeling via PVPMC library uses only parameters given by the manufacturer in combination with empirical equations. The method described here was developed by the *Sandia National Laboratory* [7]. It starts with the single diode model for a module (Duffie and Beckman [3]; Nelson [5]) which has the following equivalent circuit:

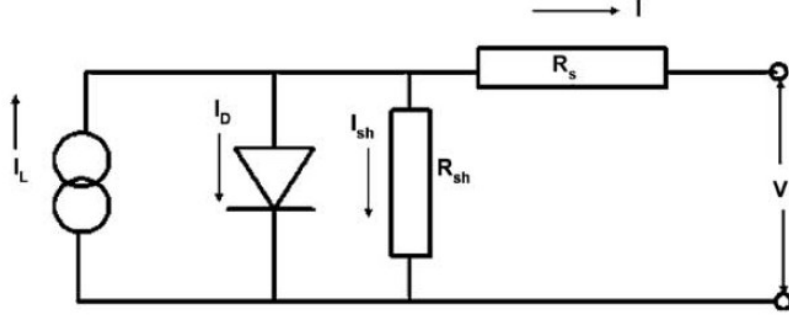


Figure 3.6: Equivalent circuit of the single diode model of a PV module.¹

The corresponding current-voltage relationship is

$$I = I_L - I_0 \left(e^{\frac{V+I \cdot R_s}{a}} - 1 \right) - \frac{V + I \cdot R_s}{R_{sh}} \quad (3.1)$$

with

$$a = \frac{N_S n k T}{q} \quad (3.2)$$

- and R_S Series resistance [Ω]
- R_{Sh} Shunt resistance [Ω]
- a Modified diode ideality factor
- N_S Number of cells in series inside the module

The first two terms to the right side in equation 3.1 are the terms for the ideal diode equation of a single solar cell as mentioned in section 2.4 ($I = I_L - I_0(e^{\frac{qV}{nkT}} - 1)$) with $V \rightarrow \frac{V+I \cdot R_s}{N_S}$ as the included series resistance has to be considered. The division by the integer N_S considers the fact that the modules voltage output is due to a series of N_S single cells inside the module, each with a voltage drop of $\frac{V+I \cdot R_s}{N_S}$. These first two terms of equation 3.1 correspond to the light induced current I_L and diode current I_D in figure 3.6 where $I_D = I_0(e^{\frac{V+I \cdot R_s}{a}} - 1)$. The third term to the right in equation 3.1 considers the drop in current of the modules output due to the shunt resistance of the module.

Equation 3.1 includes 5 parameters to determine the current for some voltage value, therefore its called *5-parameter model*. The usually given parameters from the manufacturers are short circuit current I_{SC} , open circuit voltage V_{OC} , maximum power point current and voltage I_{MPP} and V_{MPP} and the temperature coefficients of short circuit current $\alpha_{I_{SC}}$ and open circuit voltage $\beta_{V_{OC}}$ (also the Nominal Operating Condition Temperature NOCT is given but not needed in this model). All these parameters are given at standard conditions STC which are defined for an irradiance of $1000 \frac{W}{m^2}$ and a cell temperature of $25^\circ C$. To calculate the 5 reference parameters at STC $I_{L,ref}$, I_0,ref , $R_{S,ref}$, $R_{Sh,ref}$ and a_{ref} 5 independent equations are needed. 3 Equations are yielded from short circuit, open circuit and maximum power point conditions applied to equation 3.1. Fourth equation is yielded from the fact that the derivative of power at the maximum power point is zero. The last equation comes from the expression for the temperature dependency of the open circuit voltage. For solving the equations simultaneously a nonlinear equation solver (EES, Klein 2005 [12]) is used. Once calculated this 5 reference parameters the parameters at other operating conditions I_L , I_0 , R_S , R_{Sh} and a can be calculated by use of some empirical equations. This is what's implemented in the Matlab-code of the block diagram. The empirical equations assume a constant series resistance R_S , whereas the shunt resistance R_{Sh} is approximated as a function of the irradiation: $R_{Sh} = R_{Sh,ref} \cdot \frac{S_{ref}}{S}$. $R_{Sh,ref}$ is the shunt resistance at reference conditions, S and S_{ref} are the irradiation and irradiation at reference conditions. Sandia National Laboratory offers a table with the 5 calculated reference parameters for a huge amount of modules which then are used for modeling the modules IV-curve in the Simulink model. The reference parameters can be calculated for any arbitrary module, which manufacturer parameters are known, by using a non-linear equation solver to solve the 5 equations simultaneously as

¹Figure was taken from ref. [1]

mentioned above.

(For more detail refer to *De Soto et al, Improvement and validation of a model for photovoltaic array performance* [1])

3.2.1.4 Modeling via Current-Voltage-Curve

For the modeling via IV-curve a set of measured current-voltage (IV) pairs from the desired module is used. The implemented code can handle up to 200 IV pairs per module. Interpolation between the discrete data points will be done linearly. For modeling at other irradiation and temperature conditions then the ones from the measurement a procedure according to International Standard IEC 60891 [2] is used. This procedure includes the following 2 equations for transforming current and voltage values of the data points.

$$I_2 = I_1 + I_{SC} \cdot \left(\frac{G_2}{G_1} - 1 \right) + \alpha \cdot (T_2 - T_1) \quad (3.3)$$

$$V_2 = V_1 - R_S \cdot (I_2 - I_1) - \kappa \cdot I_2 \cdot (T_2 - T_1) + \beta \cdot (T_2 - T_1) \quad (3.4)$$

with	I_1, V_1	Coordinates of points on the measured characteristics in [A],[V] respectively
	I_2, V_2	Coordinates of the corresponding points on the new characteristics in [A],[V] respectively
	G_1	Irradiance at the measured characteristics in [W/m ²]
	G_2	Irradiance at the new characteristics in [W/m ²]
	T_1	Module temperature at the measured characteristics in [°C]
	T_2	Module temperature at the new characteristics in [°C]
	I_{SC}	Measured short-circuit current in [A] of the test specimen at G_1 and T_1
	α, β	Current and voltage temperature coefficients of the test specimen in the standard or target irradiance for correction and within the temperature range of interest in [A/°C], [V/°C] respectively
	R_S	Internal series resistance of the test specimen in [Ω]
	κ	Curve correction factor in [V/A°C]

Parameters like α and β are available from the manufacturers data sheet and are supposed to be constant over the whole range of operation. R_S and κ need to be determined from measurements as described in International Standard IEC 60891 [2].

The written Matlab function applies zero-crossing extrapolation to both axes, if necessary after transformation, using first four and last four data pairs respectively for a linear regression. Further a *Saturation-Block* in the block diagram limits all current values to avoid negative values. Calculated negative values will be set to zero.

3.2.1.5 IV-Curve Datafile Format

To use measured IV-curve data files for modeling a strict file format has to be provided to the simulation software. The data files are stored in *.xlsx* format and are arranged in two columns (column A and B of the sheet). The number of rows is limited to 210 although columns 1 to 9 are reserved for some parameters and additional informations. So rows 10 to 210 are reserved for IV data points. The data points have to be monotonically increasing in voltage from 0 to V_{OC} , beginning with row number 10. They also have to be unique in voltage values because a division by difference of two neighbouring voltage points is executed for interpolation between data points. Voltage values are entered in the first column (column A). The corresponding current values are entered in the second column (column B). Extrapolation of the data points to zero-crossing is done automatically by linear regression of the first four and last four voltage-current values. Therefore the first four and last four data pairs should be representative for the IV-curve next to V_{OC} and I_{SC} .

Rows 1 to 7 contain some parameters yielded from the IV-curve measurement or from manufacturers data sheet and are as described in section 3.2.1.4. An example of the required format shown for a module named 'M3_Photon':

	A	B	C	D	E	F	G	H	I	J	K
1	Name	Photon									
2	Rs [Ω]	0,27									
3	κ [V/(A°C)]	0,0013									
4	α [A/°C]	0,002									
5	β [V/°C]	-0,0769									
6	G1 [W/m²]	900									
7	T1 [°C]	25									
8											
9											
10	0,710229	6,9732									
11	3,96634	6,95518									
12	4,23324	6,95518									
13	4,51538	6,95518									
14	4,90429	6,94918									
15	5,18643	6,95819									
16	5,4457	6,94617									
17	5,70497	6,93115									
18	6,29214	6,94317									

Figure 3.7: Format of the IV-CurveFile: The first 9 rows are reserved for parameters required for the modelling. Starting with row 10 the voltage-current pairs are entered. Left column (A) lists the voltage values starting with the lowest one, the right column (B) lists the corresponding current values.

The name of the module will be shown in the GUI after loading. Of course entries in column A and row 1 to 9 are not obligatory but just for description. It's also possible to use less than 200 data points for IV-curve representation. If so, remained cells are left empty but always have to be started with row 10. The criterion for the program if to use library or IV-curve modeling is the entry in column B, row 10 (so it's the current of the first data point). If it is empty or zero the program will switch to modeling via Matlab-library if not IV-curve modeling will be used.

3.2.1.6 The Graphical User Interface

For start and control of the Simulink model a Graphical User Interface (GUI) is used. It also is based on a former version of Kishore Kumar Yamana [6]. The GUI is executed on the host-PC and exchanges data with the target-PC via ethernet. The GUI allows the following operations:

- Transfer of the Simulink model from the host-PC to the target-PC
- Selection of the module to be simulated
- Starting/stopping the Simulink program on the target-PC
- Control of the simulated irradiance, (module-)temperature and wind speed
- Activation of the live-data acquisition for irradiance and (module-)temperature
- Data logging
- Execution of predefined sequences for irradiance and (module-)temperature
- Displaying of actual model parameters and calculated IV-curve

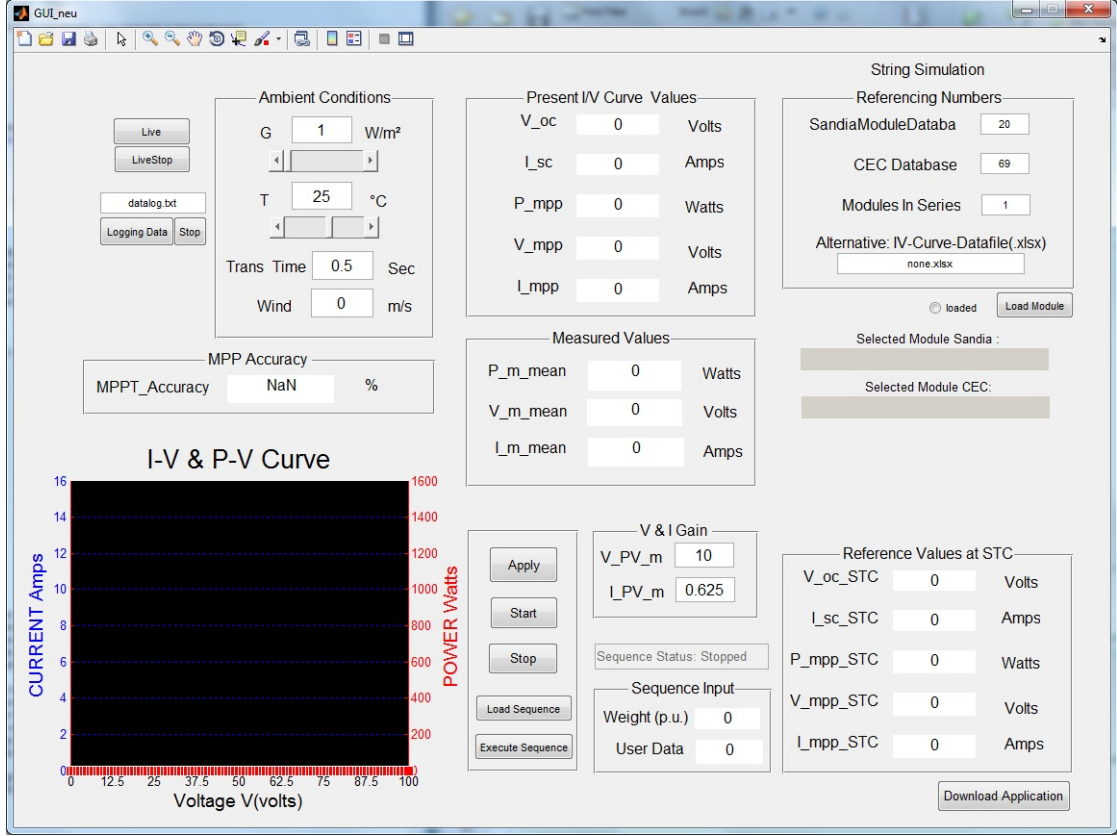


Figure 3.8: Appearance of the Graphical User Interface (GUI) on the host-PC after starting the program. From the GUI the user can start, stop and control the simulation.

A screenshot of the GUI as it appears on the host-PC's screen is depicted in figure 3.8. The initial action before the GUI can act on anything is to download the simulation software to the ready target-PC. This is done by clicking the *Download Application*-button. As a default, parameters for the *ALEO S16 185* module will get loaded. After downloading is finished parameters can be set on the Simulink model via the GUI. The box in the upper right called *Referencing Numbers* is for module selection. The entries at *SandiaModuleDatabase* and *CEC Database* referring to the number of row from the PVPVC module sheets in which the desired module is listed. The sheets are found in the working folder of the Simulink program. For proper modeling it's necessary that the numbers refer to the same module in both sheets. The Simulink program then will extract the required parameters from the sheets automatically. For the case of modeling via IV-curve the textedit *Alternative: IV-Curve-Datafile* is used. By typing in the filename including the extension (needs to be .xlsx) which contains the measured IV-curve this file will be used for modeling the module. The format of such an IV-curve file has to fulfill the requirements described in section 3.2.1.5. If modeling via parameters from Sandia- and CEC database is intended the string has to be set to *none.xlsx*. Within the textedit *Modules In Series* the desired number of modules in series to be simulated is specified. By clicking the *Load Module*-button all required data will be loaded to the target-PC. The *loaded* indicator next to the *Load Module*-button indicates if the loading has been finished (blue color). For checking the selections on the modules in the sheets the names of the modules will be shown in the text fields named *Selected Module Sandia/CEC*.

The box in the lower right called *Reference Values at STC* is showing some reference values under Standard Conditions (STC, in this case irradiance of 1000W/m^2 and ambient temperature of 25°C) for the selected module. For modeling via library they are taken from the PVPVC module sheet. For modeling via IV-curve they get calculated for standard conditions using equations 3.3 and 3.4 and linear interpolation or extrapolation respectively.

The boxes *Present I/V Curve Values* and *Measured Values* in the upper middle are showing actual values of the simulation. Although the name of the box *Measured Values* is not quite correct because only V_{m_mean} is a real measured value whereas I_{m_mean} is the calculated set point value and therefore P_{m_mean} is a mixture of both. There is no assurance that the set

point for the current (I_{m_mean}) is really driven by the PVAS if there are some errors. The boxes in the lower middle: *V&I Gain* are the gains for input signal of voltage and output signal of current. The voltage gain is determined by the gain of the measurement tool which probes the voltage drop of the PVAS. In this case a differential probe with a magnification of 1/10 was used, so the input signal had to be multiply by factor 10 (which is also default value when starting the GUI). The current gain is determined by the PVAS2 signal input port which is configured to have a ratio of 1.6A/V (3.2A/V for PVAS3). So gain for the current is $\frac{1}{1.6} = 0.625$ for the PVAS2 (default) and $\frac{1}{3.2} = 0.3125$ for the PVAS3. The gains are getting applied to the Simulink model by pressing the *Apply*-button when the simulation is stopped. This also rescales the plot axes in the window *V-I & P-V Curve* to an appropriate range. Using the buttons *Start* and *Stop* the simulation gets started/stopped. This means too that signals get in- and outputted from the connected DAQ-box. The *Apply*-button applies the adjusted values for irradiance (G), temperature (T), transition time and wind from the box *Ambient Conditions* to the simulation. During simulation by pressing the *Apply*-button the old values get continuously changed to the new values. *Trans Time* (Transition Time) determines the time frame for changing from old to new values (Transient Mode). Transition between this values will appear as linear function (linear ramp) with a slope of $\frac{NewValue-OldValue}{TransTime}$. Temperature T can be the ambient temperature as well as the module temperature. Depending on your temperature measurement (ambient or module) the modeling code for the module in the Simulink block *Embedded MATLAB Function* (*CurveOrParam*) has to be modified (see section 3.2.1.2). During simulation the window in the lower left *I-V & P-V Curve* will show a plot of the current IV-curve according to the actual parameters. The overlying box *MPP Accuracy* will give the reached Maximum Power Point Tracking Accuracy (MPPT Accuracy) yielded from the ratio of actual power to maximum power at current conditions. This feature eases to find the MPP when adjusting the manual variable resistor during simulation or helps to examine the MPP-tracking accuracy when using an inverter instead of the manual variable resistor. Buttons *Live* and *Live Stop* are for using live values for irradiance and temperature acquired from the sensors by the DAQ-box. Live values will be refreshed every 0.5sec. The buttons *Load Sequence* and *Execute Sequence* allow the user to execute a predefined profile of irradiance and temperature values over time stored in a Matlab-file called *Seq.mat* (Sequence Mode). When loading the sequence file by clicking the button an additional window appears showing a plot of the stored sequence (figure 3.9a). A red bar will mark the current position in the sequence.

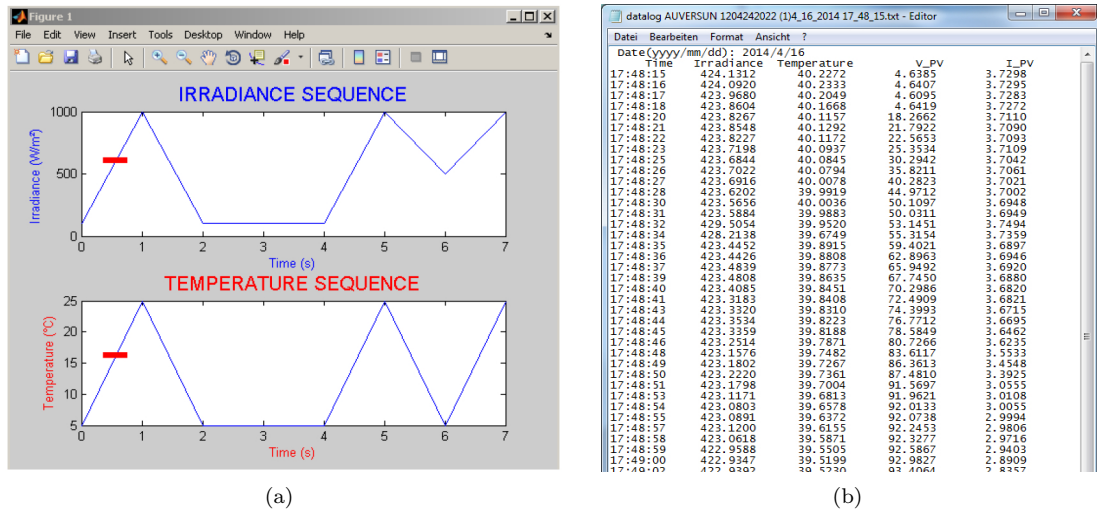


Figure 3.9: a) Irradiance and temperature plot of a sequence. b) Example of the format from logged data.

Logging Data enables to log some simulation parameters to a .txt-file. The default parameters which will be logged are irradiance, temperature, voltage and current. The logging period is 1 sec and every entry will be signed with a time stamp (figure 3.9b). The data will be stored in the current working folder named according to the string in the textedit above the logging-buttons. The extension .txt also has to be entered with the filename.

3.2.2 Data Acquisition (DAQ, Signal Acquisition)

The communication of signals between hardware and software is accomplished by a PCI-card from **National Instruments**, PCI6229, which is installed inside the target-PC. For easier handling it is connected to a BNC-connector box also from **National Instruments**, BNC-2110, so all the signals can be quick and easily connected through BNC-connectors.

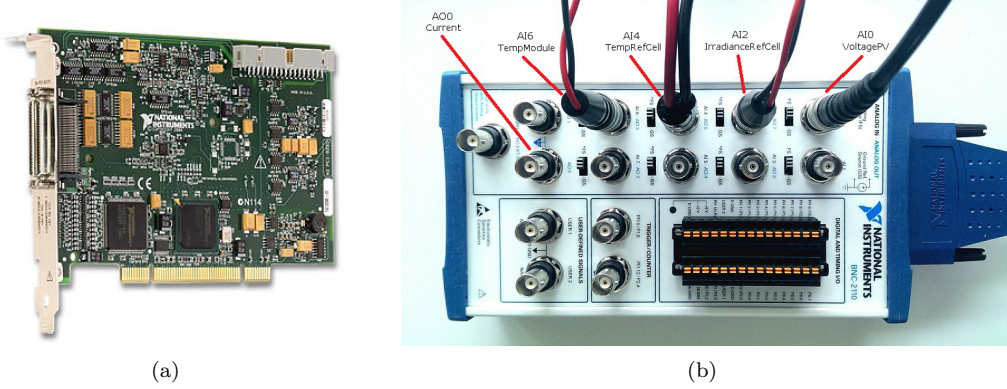


Figure 3.10: Signal transformation and transfer is accomplished by data acquisition devices from National Instruments a) PCI-card PCI6229. b) BNC-connector box BNC-2110. They are connected to each other through a shielded 68-pin cable.

The maximum voltage range for the in-/output signals of the PCI-card and BNC-connector box is $\pm 10V$. Minimum sample time of the PCI-6229 is $4\mu s$ per channel. Because of better noise reduction the sample time has been set to $7\mu s$. For this PHIL-experiment 4 analog inputs and 1 analog output are used. The pin-assignment is the following:

Analog Inputs

AI0 Voltage signal for voltage drop over the PVAS, so it's the voltage drop over the simulated string. This signal is used for calculation of the corresponding current according to IV-curve of the module.

AI2 Reference cell voltage signal for irradiance

AI4 Reference cell voltage signal for cell temperature

AI6 Voltage signal for the temperature of the modules back sheet or ambient temperature

Analog Output

AO0 Voltage signal for control of PVAS's current output ($1.6A/V$)

The switches below the BNC-connectors at the BNC-connector box offers two different configurations *FS* and *GS*. For this experimental setup all of the used channels have to be set to *FS* which stands for *Floating Source*. This means that the incoming signal is a floating signal so it is not referenced to ground. *GS* on the other hand is for *Grounded Sources*.

3.2.3 Temperature Measurement

Temperature sensing is accomplished by PT100 sensors. They have a well defined temperature dependent resistance. By applying a constant current and measuring the voltage drop one gets the value for resistance and therefore the temperature of the sensor. The constant current of $100\mu A$ is supplied by a precision current source/sink with a very low temperature coefficient (REF200U). The appropriate supply and measurement circuits are designed and build by Bernhard Kubicek (circuit diagram see appendix A.2). The same boards also includes circuitry for only signal amplification without supplying a PT100 sensor. The temperature sensing for the experiment is done by mounting a PT100 sensor to the back sheet of the real module in the simulation using thermally conductive paste, thermally insulating tape and aluminium tape for fixing. If a string of proper modules should be simulated take care that the sensor is placed at a spot on the back sheet which is representable for a faultless module (like no active bypass diode or cell

with hotspot). In case that ambient temperature is used for modeling then place the sensor at an appropriate shaded spot next to the module.

For measuring the temperature of the reference cell the integrated PT100 sensor is used. The temperature influences the calibration of the signal from the solar cell to determine the irradiation (*Irr_Calib*-block in figure 3.3).

3.2.4 Irradiance Measurement

Irradiance sensing is done by a reference photovoltaic cell which gives a defined current of 100mA at standard conditions of $1000 \frac{W}{m^2}$ and 25°C. The irradiation dependency of the photocurrent is supposed to be linear and passes zero for vanishing irradiation. For getting a voltage signal to the DAQ-box a resistor of 0.1 Ohms is connected in series to the reference cell. The voltage drop then gets amplified by an amplifier circuitry (Bernhard Kubicek, circuit diagram see appendix A.2).

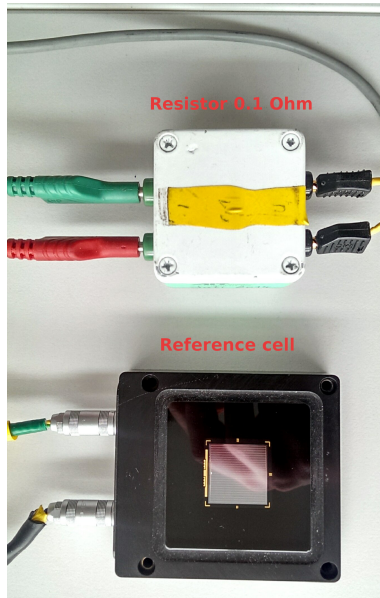


Figure 3.11: For measurement of irradiation a reference cell is used, connected in series to a resistor of 0.1Ω for determining the current due to the voltage drop. The two connectors at the reference cell are for output of the generated current and output of the integrated PT100 sensor to determine cell temperature.

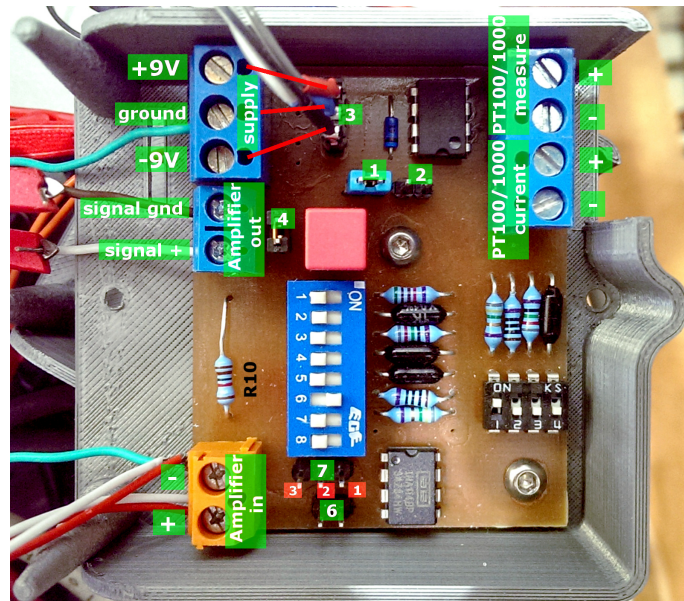


Figure 3.12: Amplifier board (version 3). Signals from the sensors gets amplified by an amplifier board to yield a better resolution of the signal measurement at the PCI-card's data acquisition ports.

3.2.5 PVAS Voltage Measurement

For calculating the current output of the simulated string the voltage drop of the PVAS is needed. The maximum voltage range of the DAQ-box is limited to $\pm 10V$ so the voltage signal has to be transformed to a lower voltage signal. Therefore a differential probe is used. It has a maximum voltage range of 1000V and offers 2 amplification factors ($\frac{1}{10}$ and $\frac{1}{100}$) to lower the signals. The amplification factor has to be set according to the maximum voltage drop of the PVAS in the PHIL experiment so that the range of the DAQ-box is utilized to a maximum but not exceeded. The amplification factor have to be considered in the Simulink model too to recalculate the correct voltage drop. Therefore the appropriate gain in the textedit V_{PV_m} in the box $V\&I$ Gain of the GUI has to be set. For example, an amplification factor of $\frac{1}{10}$ on the differential probe needs to be considered in the simulation by setting V_{PV_m} to 10 in the GUI. Confirmation of the value by use of the *Apply*-button will set the according value in the Simulink model.



Figure 3.13: A differential probe with BNC-plug and banana-plug is used to convert the high voltage signal from the PVAS output to a low voltage signal for the DAQ-box. It provides attenuation factors of 10 and 100.

3.2.6 Signal Amplification

As the PT100 sensors for temperature measurement have to get supplied with a current and the yielded signals need to get amplified, amplifier circuits are used. Also the signal from the reference cell (series resistor) for measuring the irradiation has to get amplified as all of these signals are in a range less than 1Volt. To use the full resolution capability of the PCI-card, the signals get amplified by analogue circuits. These amplifier circuits are designed and built by Bernhard Kubicek. They act as amplifier as well as supply for the sensors. There are three versions of boards (v1, v2 and v3) which differ in some components and circuit design but are equal in functionality. For the PHIL experiment only v2 and v3 boards were used. All of the boards get supplied by two 9V battery blocks providing a symmetrically voltage supply ($\pm 9V$). In figure 3.12 the third version of circuitry is depicted which is used for amplifying the irradiance signal. The blue multi-switch in the middle (8-fold switch) of the board is for setting the gain. The multi-switch on the right side (black 4-fold switch) determines the reference resistance for comparison of voltage drop through PT-sensors. The reference resistors have very low temperature sensitivity. The difference in voltage drop over reference resistor and PT-sensor serve as input signal for the amplifying part of the circuitry. In terms of avoiding negative input signals for negative temperatures at the PT-sensors a less resistance then the PT-resistance has to be chosen. So for the measurement via PT100 sensors the reference resistance of 82Ω has to be selected. This corresponds to switch number 1 of the 4-fold switch (switch1 $\rightarrow 82\Omega$, switch2 $\rightarrow 100\Omega$, switch3 $\rightarrow 820\Omega$, switch4 $\rightarrow 1000\Omega$).

Jumpers 1,2,4,6 and 7 are for various configurations of the boards:

Filter To activate the filter for the amplifiers output jumper 4 is set as closed and jumper 2 is set as opened.

Negative Voltage Protection To avoid negative voltage drop at the amplifiers output (which can cause damage to the amplifier) a protection mode has been implemented. To activate this protection mode jumper 2 has set as opened and jumper 1 as closed.

Activation of PT-Sensing If using the board for temperature sensing via PT-sensors the amplifier input has to be connected to the reference resistors by setting jumper 6 as closed and jumper 7 pin 1/2 as closed (for pin assignment see figure 3.12, red boxes). If PT-sensors are not used, the amplifiers input has to be connected to the external source by setting jumper 6 as opened and jumper 7 pin 2/3 as closed.

Hint: v2 boards don't have a 3-pin jumper 7 instead they have a 2-pin jumper. The 2-pin jumper correspond to pin 1 and 2 of the 3-pin jumper. So for PT-sensing the 2-pin jumper has to get closed and for not PT-sensing it has to be opened. Further the minus pole of the amplifier input has to be connected to the ground pole of the supply (see green wire in figure 3.12, this has the same result as setting jumper 7 pin 2/3 as closed).

Jumper 3 is directly connected to the supply block on the upper left as indicated by the red bars and is for quick connection to the voltage supply.

For calibration of the signals in the Simulink model information about gain and offset of the amplifier circuits is needed. Therefore all the boards got gauged as documented in appendix A.3. Because there are some spreads in gain and offset among the boards it's important to use the appropriate board for the related signal calibration in the Simulink model. The configuration of the PHIL-experiment with respect to the assignment of amplifiers and signals is the following:

Irradiance Measurement Amplification of the irradiance signal is done by the v3 board (blue 8-fold switch) and has the following configuration (see figure 3.12):

8-fold switch, switch 6 → on

4-fold switch, switch 1 → on

Jumper 1,4 and 7 (pin 2/3) closed

Temperature Reference Cell Amplification of the built in PT100 signal is done by a v2 board (resistor R10 blue colored, position of R10 → see figure 3.12 or appendix A.2):

8-fold switch, switch 7 → on

4-fold switch, switch 1 → on

Jumper 1,4,6 and 7 (pin 1/2) closed

Temperature Module Back sheet Amplification of PT100 signal for module back sheet temperature (or ambient temperature) is done by a v2 board (resistor R10 brown colored, position of R10 → see figure 3.12 or appendix A.2):

8-fold switch, switch 7 → on

4-fold switch, switch 1 → on

Jumper 1,4,6 and 7 (pin 1/2) closed

3.2.7 Power Signal Amplifier (PhotoVtaicArraySimulator PVAS)

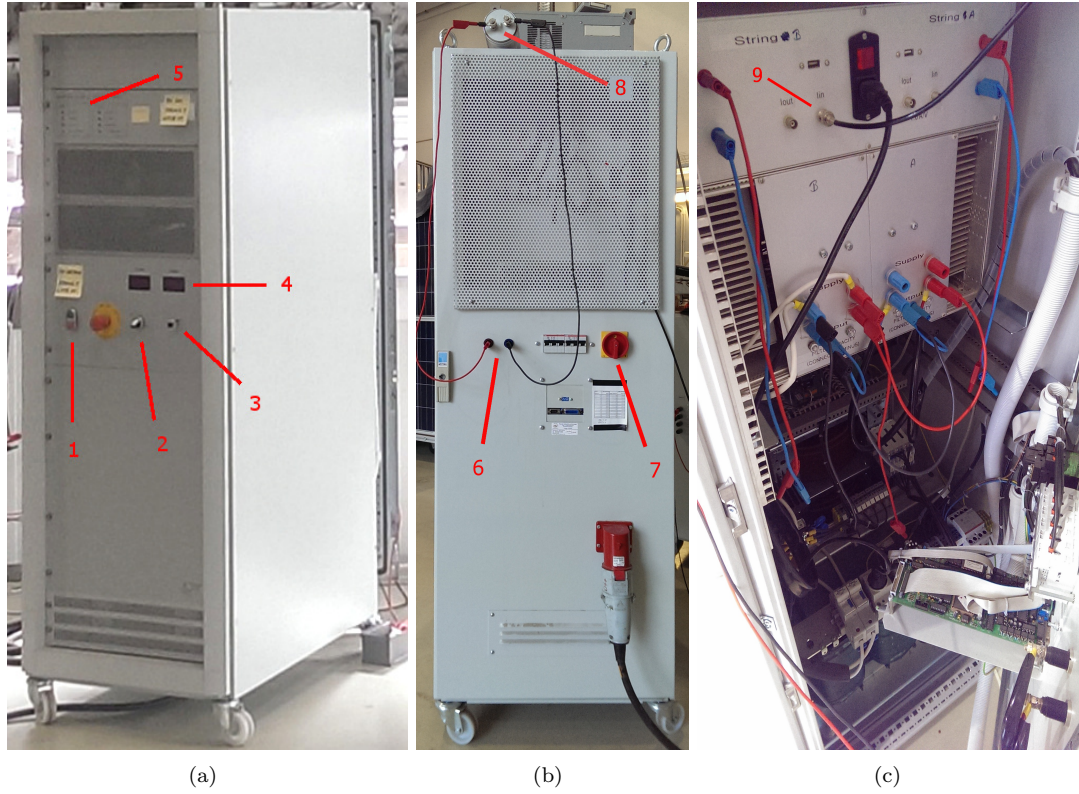


Figure 3.14: The used power hardware for the simulation of a PV-string is the PhotoVtaic Array Simulator (PVAS) from the Austrian Institute of Technology (AIT). It acts as a voltage controlled current source. a) Front side view of the PVAS2. b) Backside. c) Inside.

For simulating a PV string a controllable high power source is used. The PCI-card only supplies with $\pm 10\text{V}$ at $\pm 5\text{mA}$ whereas a PV string usually reaches up to some hundreds of Volts at 10 Amperes. Therefore the PVAS2 is used which can provide up to 800V at 16A when simulating a singular string. When simulating two strings in parallel, 32A can be supplied (PVAS3 supplies wit 800V and up to 32A at single string). The PVAS gets controlled by a 0-10V signal determined by the Simulink model and outputted from the DAQ-box. The connector (BNC-plug) for the control signal from the DAQ-box is inside the PVAS (9). An input signal to this plug of 0-10V leads to an output signal of the PVAS2 of 0-16A (single string) or 0-32A (double string) with a ratio of 1.6A/V and 3.2A/V respectively (PVAS3 0-10V $\rightarrow$ 0-32A , with ratio of 3.2A/V). Therefore the PVAS acts as a voltage controlled current source. The mapping from input to output appears linearly without offset, so $I_{out} = 1.6 \cdot V_{in}$ for a single string and $I_{out} = 3.2 \cdot V_{in}$ for two strings in parallel (and PVAS3).

The main switch of the PVAS is positioned on the backside (7) whereas the common on/off switch (1) is positioned on the front panel of the PVAS . With the potentiometer (3) the maximum supply voltage of the amplifier is adjusted which value is shown on the right display (4). When preparing a simulation the supply voltage has to be set to a value higher than the maximum voltage to be simulated because the maximum voltage drop over the PVAS will be limited by the supply voltage. So for simulating a string of 10 modules with open circuit voltage of 30V per module, the supply voltage has to be set to at least $10 \times 30\text{V} = 300\text{V}$. To guarantee maximum voltage supply, the supply voltage should be set to 15V above the necessary voltage for simulation. On the left display (4) the actual driven current through the PVAS is shown. On the top of the front panel (5) some LEDs acts as indicators for the most common errors. The output of the PVAS is positioned on the backside (6). As the PVAS was not connected to an inverter for electrical load an additional capacity (8) had to be connected to the PVAS's output in parallel (as can be seen in figure 3.14b and figure 3.1). The used capacity is a $300\mu\text{F}$ capacitor and is necessary to avoid oscillations of the setup when operating the PVAS close to the open circuit

voltage of the simulated IV-curve.
(For more details on the PVAS2 (PVAS3) contact Philip Svec)

3.2.8 Photovoltaic Modules

The photovoltaic modules are part of the real system components in the PHIL experiment. In general there is no restriction on the number of modules which get connected in series to the PVAS. Care needs to be taken about the correct polarity when connecting PVAS and module(s).

3.2.9 Electrical Load

The electrical load is also part of the real system components. It's not absolutely necessary to include this component in the PHIL experiment. If one is just interested in examinations at short circuit conditions no electrical load is needed. Otherwise if behavior at non short circuit condition is of interest an electrical load is used for adjusting the operating point. Common electrical loads are then inverters or like in this case a variable resistor. A variable resistor has advantage to choose an operating point independent from the maximum power point or to examine the strings behavior over the complete voltage range.

The used variable resistor consists of 5 independent coiled resistors. Each resistor can be varied from 0-200 Ω and has a specified maximum power dissipation of 400W. The variation of the resistances is done by crank handles (as can be seen in figure 3.15), varying the length of the effective resistor coils and therefore their resistivities. BNC-plugs on the resistors backside allow them to get connected in series yielding an accumulated resistance of 0-1000 Ω and a maximum power consumption of 2000W. (Values for capacitance and inductivity of the resistor are given in appendix A.4)



Figure 3.15: A variable resistor as electrical load is used for the tests to have control over the operating point of the simulation.

3.3 Configuration of Target and host-PC

The instructions for configuration of the PC's as described here are appropriate for use of the following soft- and hardware. For use of other devices and versions refer to the built in help function of Matlab.

host-PC:

Windows 7 Enterprise, Service Pack 1, 64-bit
Intel i7-3770 @ 3.4 GHz, 4 GB RAM
Networkadapter PCI Digitus DN-10110 (Realtek RTL8169/8110)
Matlab R2013b (Version 8.2), 64-bit, including

- Simulink
- RealTimeWorkshop
- xPC Target Toolbox

target-PC:

Windows XP Professional, Version 2002, Service Pack 3
Intel Pentium4 @ 3.8 GHz, 3.5 GB RAM
Matlab RealTime kernel via USB-stick
PCI-card National Instruments PCI6229
Networkadapter Broadcom NetXtreme 57xx Gigabit Controller

Even if the operating system on the target-PC is not necessary for running the realtime kernel, it was already installed on the PC and was used for determining the installed hardware specifications.

3.3.1 Configuration of target-PC (xPC-Target)

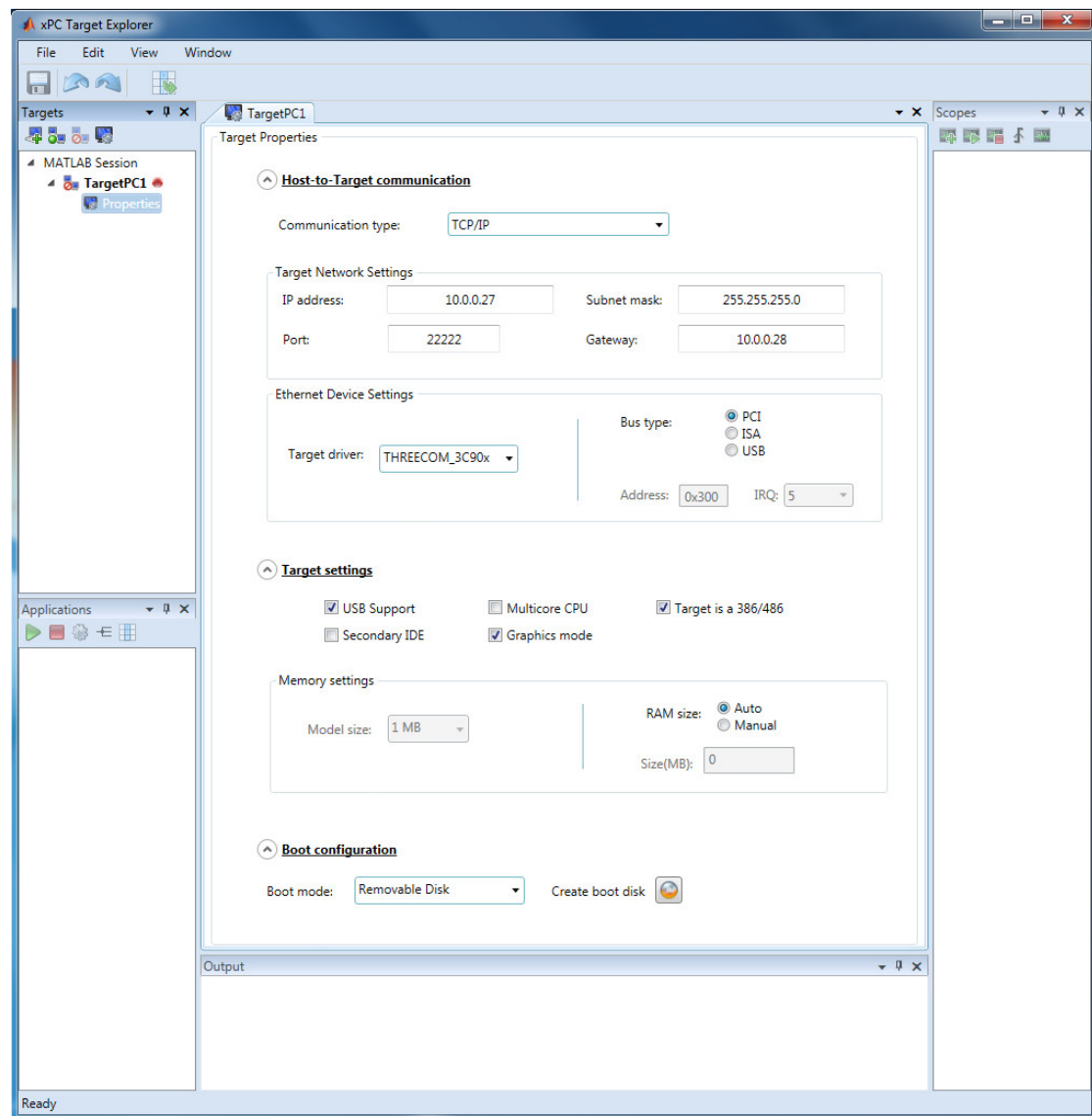


Figure 3.16: Screenshot of the configuration dialog for the boot drive of the target-PC. The *Target Driver* drop-down menu offers a few general drivers for ethernet-cards. Even if the used ethernet-card isn't a THREECOM, it's the appropriate driver for it. The checkbox *Target is a 386/486* needs to be set due the architecture of the built in CPU. More detailed information about the configurations is given in Matlab Help Browser. A click on the button *Create Boot Disk* will create the boot disk (plugged-in USB-drive) for the target-PC according to the entered configuration.

For running the Simulink model in RealTime mode the target-PC need some extra configuration. A RealTime kernel provided by Matlab has to be installed on a bootable drive. Further the BIOS settings have to be configured to :

- Booting from USB device is enabled
- Multi-threading is disabled.

An ethernet connection to the host-PC is necessary. Matlab provides a few options to run RealTime kernel on the target-PC. In this case the boot mode *Removable Disk* was chosen with an conventional USB drive (AIT USB drive 4GB) as the removable medium. To create the boot drive type in *xpcexplr* in the Matlab command window. As a consequence a new window opens called *xPC Target Explorer*. In the *targets*-window on the upper left expand the *TargetPC#* item by double clicking. Then set the properties in the *Target Properties*-window as shown in figure

3.16. Finally plug in the USB drive, click the *create boot disk*-button and select the according drive. Check the ethernet connection to the host-PC and start the target-PC.

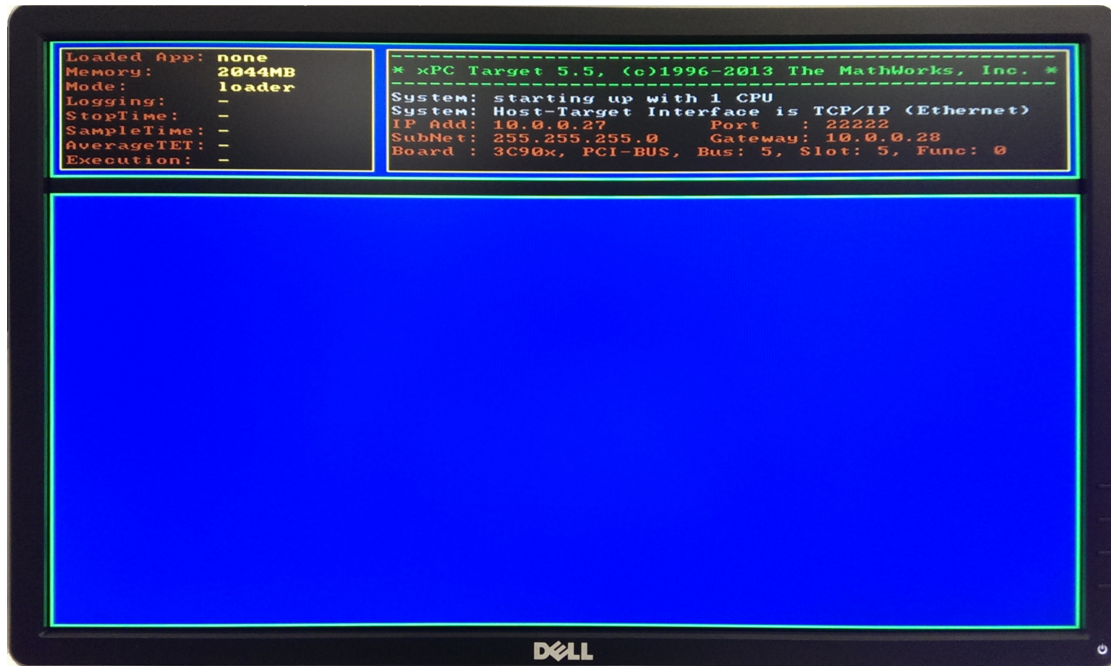


Figure 3.17: Appearance of the target-PC's screen after booting from USB drive. The upper left window show some general system- and simulation settings and parameters, the right one is kind of a command line during simulation.

After a successful start of the target-PC the target-Monitor should look like figure 3.17. The target-PC is now ready for getting commands from the host-PC.

3.3.2 Configuration of host-PC

For the host-PC the installation of a version of Matlab is necessary including the toolboxes *Simulink*, *RealTimeWorkshop* and *xPC Target*. For the connection of host and target-PC via ethernet the configuration of the network adapter is set to:

IP-adress: 10.0.0.28

Gateway: 10.0.0.27

Subnetmask: 255.255.255.0

Chapter 4

Experimental Setup and Illustrative Tests

The PHIL setup as described in the preceding chapter was set up in the laboratory. The real module was placed in a stationary sun simulator able to provide up to $950W/m^2$ of solar spectrum (class C). Additionally it's possible to regulate the modules temperature by air conditioning of a controlled air stream over the module surfaces. A PT100 sensor is used for the control loop of the sun simulator's air stream temperature, to regulate the module temperature. The PT100 sensor gets mounted onto the modules back sheet. The module inside the sun simulator is mounted on a tray, kind of a big drawer, to shift the module inside and outside the sun simulator.

Values from the Simulink model as time, irradiance, module temperature, module voltage and module current were logged through the GUI during the experiments. Additional voltage values were logged through a data logger as voltage drop of module, voltage drop of PVAS, voltage drop of series resistance, for measuring the current in the string, and voltage drop of an additional pyranometer for control of irradiation. For some of the simulated operation conditions thermography (infrared imaging) of the modules have been performed. The infrared (IR) imaging was

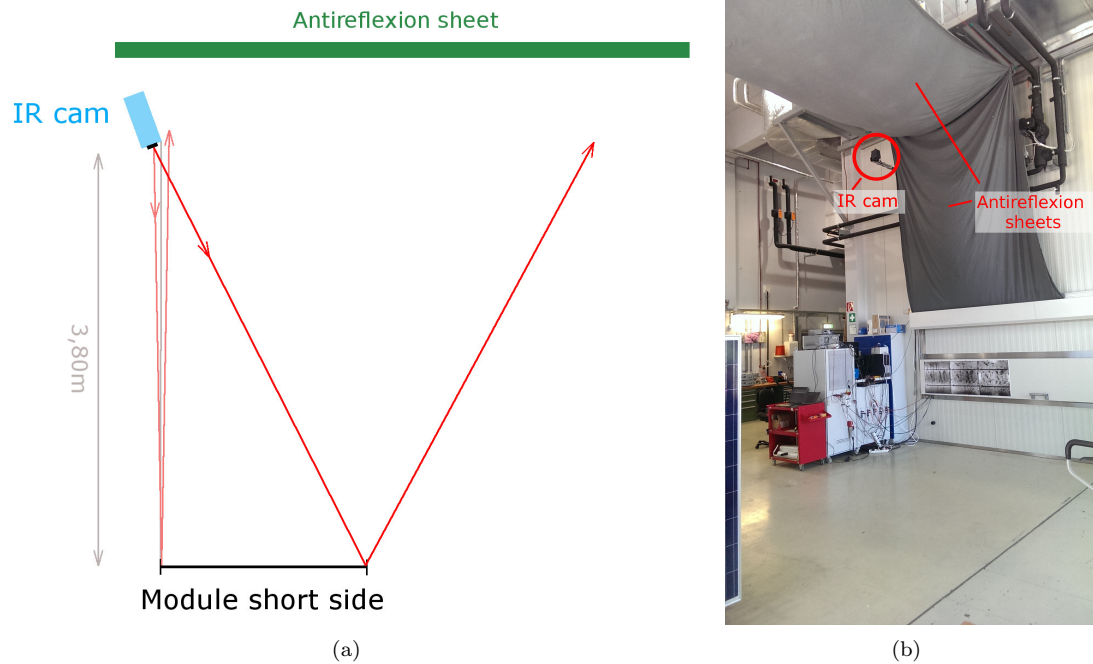


Figure 4.1: Mounting of the infrared cam and the anti reflection sheets. a) Position of the infrared cam above the modules front side. b) Mounting of the anti reflection sheets.

done from the front side of the module using the smallest angle of incidence possible so that

there is no reflection of the cam itself on the IR-image. Therefore the IR-cam is mounted in line with the modules surface normal at the middle of the modules alongside frame as indicated in figure 4.2. The distance of the IR-cam to the modules surface for capturing the whole module is 3,80 meters. To avoid reflections from the environment and installations on the ceiling some anti reflection sheets (grey cotton blankets) were mounted beside and above the IR-capturing scenery (see figures 4.1 and 4.2). The used IR cam is of type *VarioCAM hr head 600* with a geometrical resolution of 640×480 pixels and a thermal resolution of 0.03 K at 30°C. IR capturing of the modules was performed for 10min respectively outside the stationary sun simulator with no driven current from the PVAS. Before IR capturing the module was operated for at least 15min at the maximum power point to ensure a state next to thermal equilibrium of the module. The frame rate of the IR cam was set to 2Hz (cycle of 0.5sec). Capturing a sequence of IR images from the module instead of a single shot has advantage that the initial temperature of the module, right before leaving the sun-simulator, can be determined through analysis of the thermal decay behavior. Although possible this calculation wasn't performed here because it is out of the scope of this work. The average time between start to remove the module from the sun-simulator and the first full sized IR image of the module, captured by the cam, was about 30 seconds.

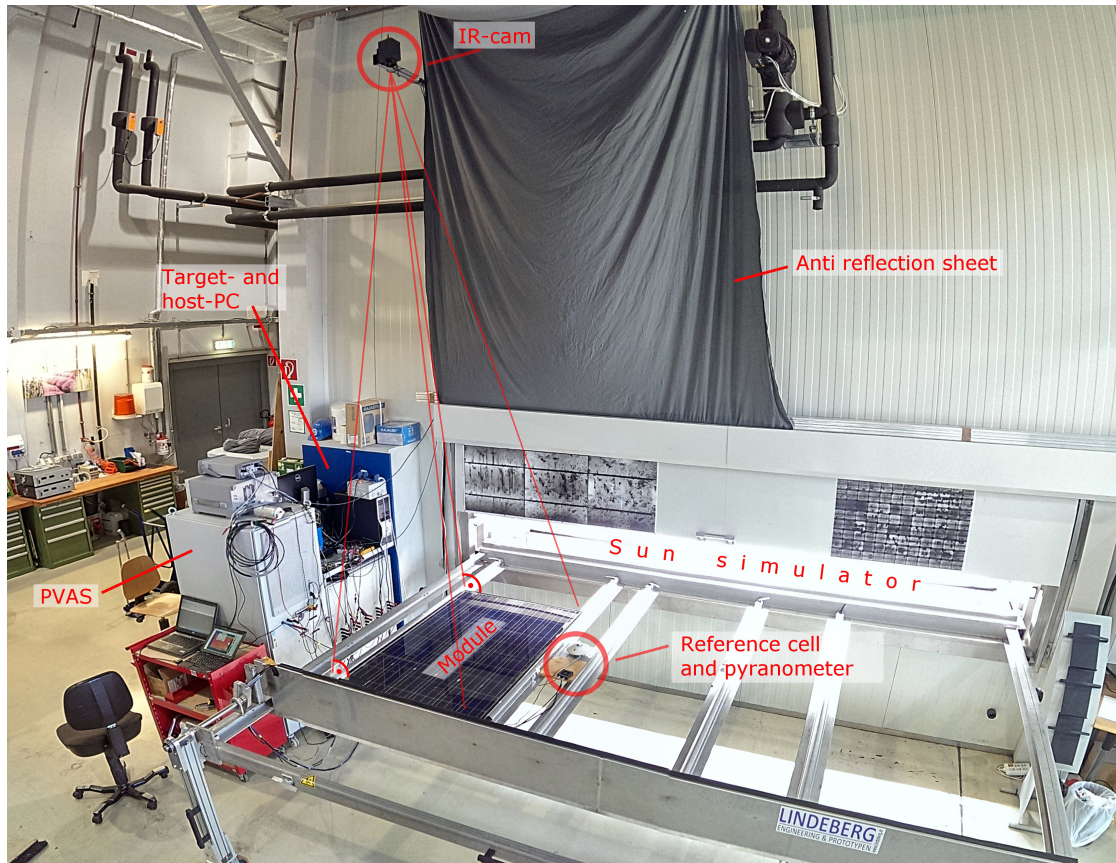


Figure 4.2: The experimental setup in the lab: The module (device under test) is mounted on the drawer of the sun-simulator. To the left there are laptops for IR-thermography control and data logging, PVAS2, target- and host-PCs. IR-thermography is applied from the top view outside the sun-simulator using cotton blankets (grey) as anti-reflection sheets. Sensors for irradiation measurement inside the sun-simulator during simulation are mounted on the right side next to the module.

4.1 Modules Under Investigation

For the testing experiments modules were used showing distinct defects. The chosen modules are output from another project of AIT (SOPHIA) for testing durability of encapsulation and interconnects. They were exposed to testings like damp/heat cycles, temperature cycles, mechanical load cycles or longtime outdoor usage. All of them showed distinct deviation from proper

current-voltage characteristic. The used modules are all of c-Si type:

ATERSA A-235P (Ser.Nr. 1031204010013, DH85/85) This module got cycled at 85°C damp heat and 85% humidity for 4000 hours in summary.

AUVERSUN AV240P60 (Ser.Nr. 1204242019, DH85/85) Cycled at 85°C damp heat and 85% humidity for 4000 hours in summary.

AUVERSUN AV240P60 (Ser.Nr. 1204242022, TC40/85) Cycled from -40°C to +85°C for 1200 cycles in summary.

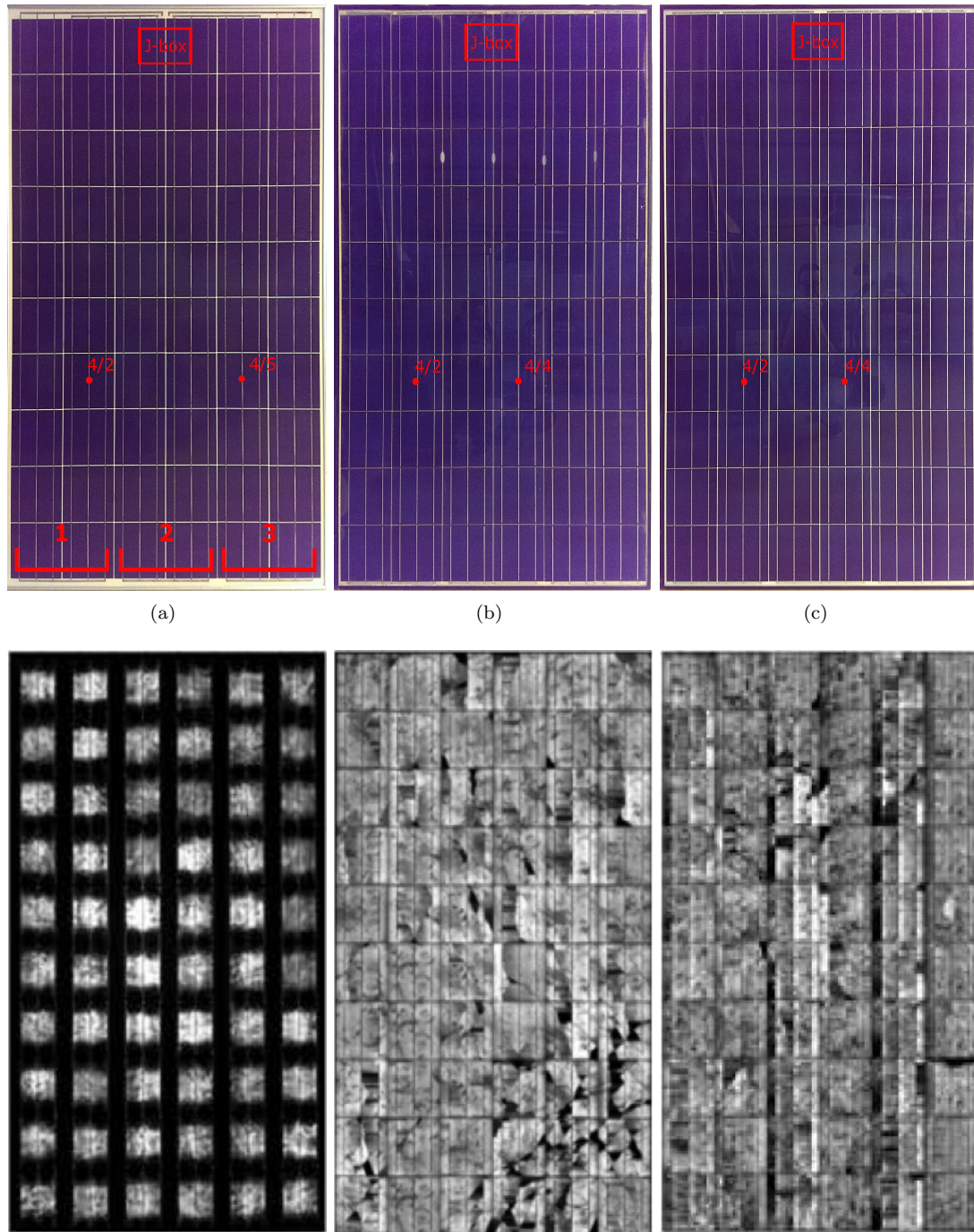


Figure 4.3: First row: Photographs of the test-modules a) Atersa A-235P DH85/85. b) Auversun AV240P60 DH85/85. c) Auversun AV240P60 TC40/85. The red colored overlays depicting the positions of the junction box, used numbering for the strings inside the module and the points of temperature measurement (row/column) for the experiments through PT100 sensors. Second row shows the according electroluminescence images of the modules.

For a more detailed description on the cycling tests refer to the *SOPHIA*-report [8]. The red spots overlayed in figure 4.3 showing the points of temperature measurements for each module. The nomenclature is according to row/column of the regarding cell in the module starting from the bottom left. Every time 2 sensors were mounted on the same cell, one for the simulation and one for the control loop of the sun simulator. They were mounted on the modules back sheet using thermally conductive paste, thermally insulating tape and aluminium tape. Therefore they are visible in the front side IR images.

Table 5.1: Simulation Configurations (Hints, see following page)

ATERSA A235P, Ser.Nr. 1031204010013, #5 DH85/85				
mod in ser	T _{mod}	G	shadowing	filename
1+1	25	500	no	atersa11_25_500_no
1+1	25	800	no	atersa11_25_800_no
1+1	25	930	no	atersa11_25_930_no
1+1	35	500	no	atersa11_35_500_no
1+1	35	800	no	atersa11_35_800_no
1+1	35	930	no	atersa11_35_930_no
1+1	45	500	no	atersa11_45_500_no
1+1	45	800	no	atersa11_45_800_no
1+1	45	930	no	atersa11_45_930_no
3+1	25	500	no	atersa31_25_500_no
3+1	25	800	no	atersa31_25_800_no
3+1	25	930	no	atersa31_25_930_no
3+1	35	500	no	atersa31_35_500_no
3+1	35	800	no	atersa31_35_800_no
3+1	35	930	no	atersa31_35_930_no
3+1	45	500	no	atersa31_45_500_no
3+1	45	800	no	atersa31_45_800_no
3+1	45	930	no	atersa31_45_930_no
5+1	25	500	no	atersa51_25_500_no
5+1	25	800	no	atersa51_25_800_no
5+1	25	930	no	atersa51_25_930_no
5+1	35	500	no	atersa51_35_500_no
5+1	35	800	no	atersa51_35_800_no
5+1	35	930	no	atersa51_35_930_no
5+1	45	500	no	atersa51_45_500_no
5+1	45	800	no	atersa51_45_800_no
5+1	45	930	no	atersa51_45_930_no
5+1	45	800	string1	atersa51_45_800_str1
5+1	45	800	string2	atersa51_45_800_str2
5+1	45	800	string3	atersa51_45_800_str3

4/5 (42°C)
4/5 (42°C)

AUVERSUN AV240P60, Ser.Nr. 1204242019, #4 DH 85/85				
mod in ser	T _{mod}	G	shadowing	filename
5+1	25	800	no	auversun51_25_800_no
5+1	25	930	no	auversun51_25_930_no
5+1	45	800	no	auversun51_45_800_no
5+1	45	930	no	auversun51_45_930_no
5+1	45	800	string1	auversun51_45_800_str1
5+1	45	800	string2	auversun51_45_800_str2
5+1	45	800	string3	auversun51_45_800_str3

AUVERSUN AV240P60, Ser.Nr. 1204242022, #16 TC -40/85				
mod in ser	T _{mod}	G	shadowing	filename
5+1	25	800	no	auversunTC51_25_800_no
5+1	25	930	no	auversunTC51_25_930_no
5+1	45	800	no	auversunTC51_45_800_no
5+1	45	930	no	auversunTC51_45_930_no
5+1	45	800	string1	auversunTC51_45_800_str1
5+1	45	800	string2	auversunTC51_45_800_str2
5+1	45	800	string3	auversunTC51_45_800_str3

4/4 (44°C)

4/4 (25°C)

4/4 (45°C)

= IR capturing

4/5 (42°C)
= Cell for temperature measurement [row/column (setpoint)]
cell in row 4 and column 5 was used for temperature measurement
and temperature setpoint of the sun simulator was set to 42 °C.

4.2 Simulation Configurations

The simulation, including the real module, was tested under several operation conditions. The irradiation was set to 500, 800 and 930W/m² whereas temperature of the module back sheet was varied to 25, 35 and 45°. Also the number of simulated modules in series was varied from 1, 3 to 5 modules. All the simulations were done with only one single real module in series with the simulated string, using the *Modeling via Current-Voltage-Curve* method (see section 3.2.1.4). The investigated simulation configurations are tabulated in table 5.1. In the first column the string constellation is given. For example 3+1 stands for 3 modules in series were simulated via the PVAS and 1 additional real module was connected in series during irradiation in the sun simulator. The second column gives the used set point of the sun simulators control loop for the modules back sheet temperature. Column 3 is showing the approximated irradiance in the sun simulator during measurement. The exact measured values are available in the according data logs and differ up to $\pm 5\%$ from the tabulated values. Column number 4 gives the shadowing conditions. "no" means no shadowing at all, "string 1" for example means that one single cell of string number 1 was totally shadowed. Numbering of the strings is according to figure 4.3. The last column gives the corresponding filename of the data log file. Rows which are red colored indicating that after the simulation IR imaging was performed and therefore IR data is available. Some rows showing extra hints on the right side of the tabular, like "4/5 (42°C)". If there is no hint then the chosen spot for measurement of the modules back sheet temperature was cell 4/2 (see section 4.1). If there is any hint then for this simulation the according cell was used in combination with a slightly shifted set point of the modules back sheet temperature which is given in the subsequent brackets. For example "4/5 (42°C)" in the second last row of the left tabular means that for this simulation cell 4/5 was used for temperature measurement and the set point for the control loop of the sun-simulator was changed to 42°C. The change in sensors positions was done for getting more comparable results because during shadowing of a string it heats up more and therefore if the sensor is placed on this string the sun simulators control loop would cool down the rest of the module to lower temperature than the original set point. Further the shift of the set point temperature is then necessary because of the air stream in the sun simulator which causes a temperature gradient like up to 6°C over the module from one side to the other. The direction of the air stream always was, with reference to figure 4.3, from right to the left side, so from string 3 to string 1. Modules therefore showing a positive temperature gradient in the same direction, means getting warmer in direction of the air stream.

4.3 Measurement Results and Discussion

At first a comparison is done to show if theoretical and simulated values match in a sufficient degree. Therefore IV-curves from measurements of the modules at standard conditions at the beginning of their lifetimes are compared to measured values from the simulation of these modules. The simulations were performed using the *Modeling via Current-Voltage-Curve-Method* (see section 3.2.1.4) as the used modules are not listed in the PVPMC-database and therefore no parameters for the simulation via *Modeling via PVPMC-Library* (section 3.2.1.3) are available.

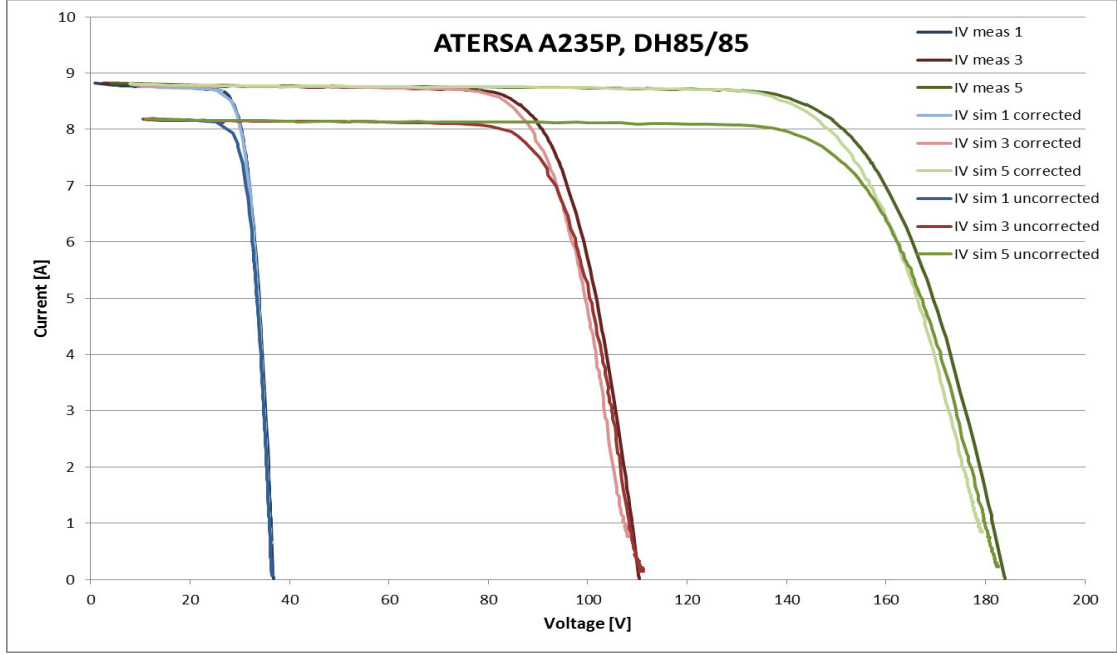
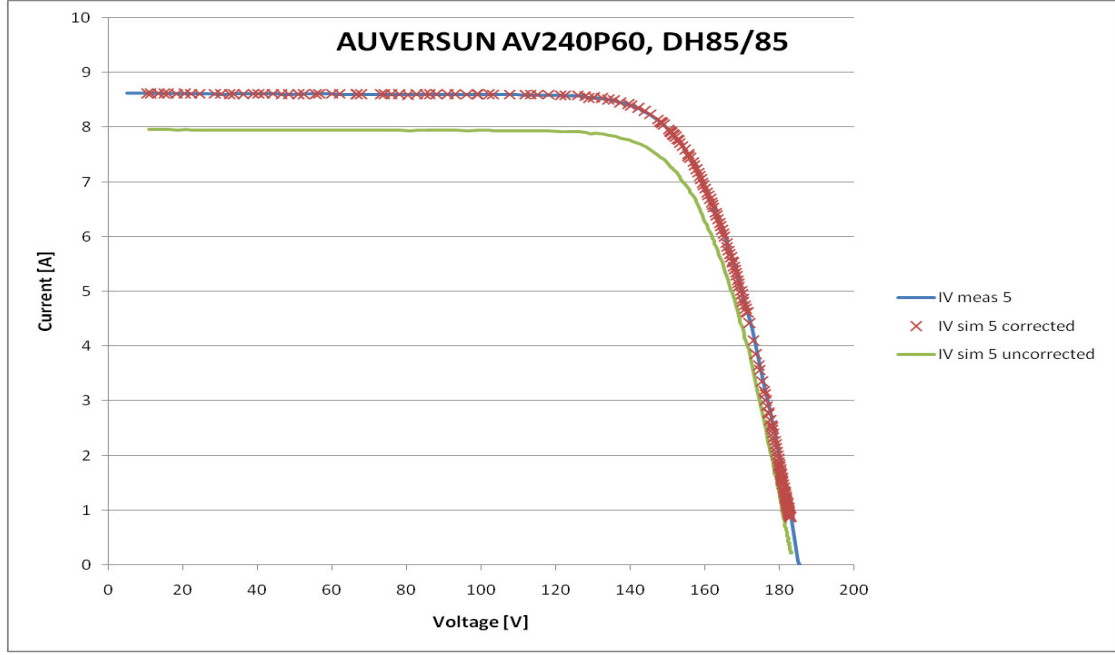


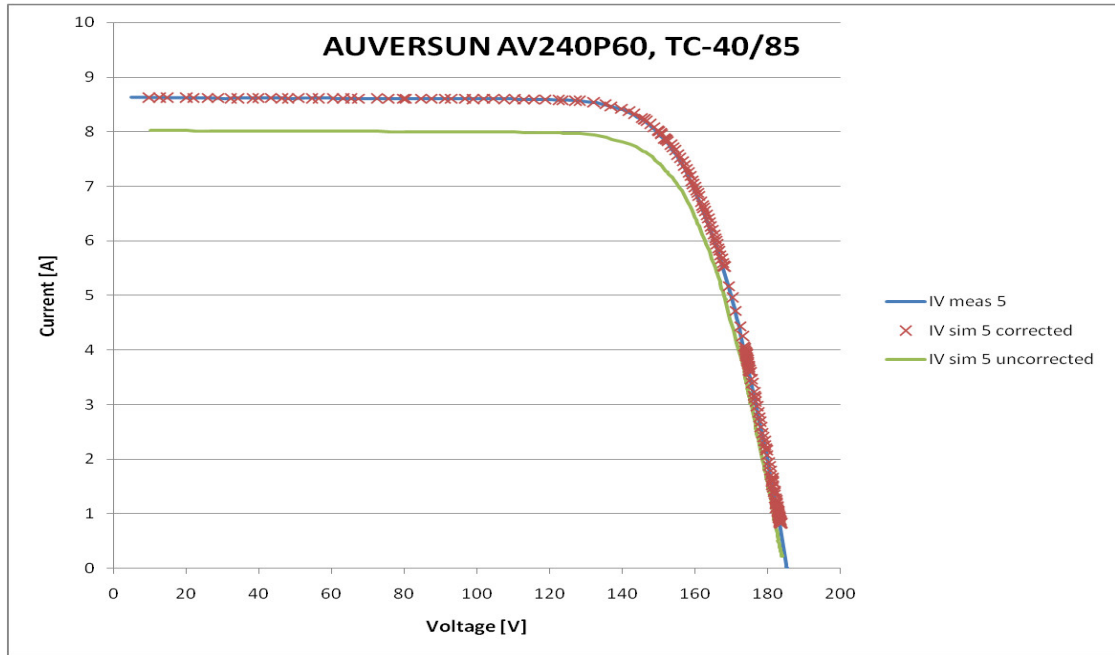
Figure 4.4: Comparison of measured, simulated and corrected IV curve values for the Atersa module. Blue, red and green curves showing results for 1,3 and 5 simulated modules in series. *IV meas #* depicts the measured IV curves at standard conditions (STC @ $1000W/m^2$, $25^\circ C$) from the module at the beginning of its lifetime. Voltage values have been multiplied by the number of modules in series. *IV sim # uncorrected* are the curves directly measured at the PVAS with irradiation in the sun-simulator of $\sim 930W/m^2$ and a temperature of $25^\circ C$, whereas *IV sim # corrected* are the curves corrected to STC, according to IEC 60891, for comparison with the measured values.

For comparison of the values from the simulation of multiple modules in series with the original module IV-curve, voltage values from the original module IV-curve are multiplied by the number of modules in series in the simulation. Voltage and current values from the simulation have been measured directly at the PVAS. Because irradiation and temperature in the sun-simulator didn't fit the standard conditions exactly, correction according to the IEC 60891 [2] standard has been applied to get standard condition values again. So the correction is in the same way as implemented in the simulink model and therefore shouldn't add an extra error to the simulated IV-curve. Finally the original IV-curve and the simulated/corrected IV-curve are compared to each other.

The measured IV-curves are shown in figure 4.4. The deviations from measured, simulated and corrected curves are most likely from measurement errors with respect to temperature. The modules showing a big temperature gradient over the whole area because of the air stream in the sun simulator. The temperature drop from one side of the module to the other is up to $6^\circ C$ (as can be seen in the IR images below) and therefore the temperature measurement of the module is of less accuracy. When varying the initial temperature of correction in a range of $0.4^\circ C$ the curves will match so that they cannot be distinguished in the plot.



(a) Comparison of measured, simulated and corrected IV curves, Auversun module (DH 85/85)



(b) Comparison of measured, simulated and corrected IV curves , Auversun module (TC -40/85)

Figure 4.5: Comparison of measured, simulated and corrected IV curves for 5 Auversun modules connected in series. Measured IV curve @ STC, simulated/uncorrected IV curve @ $930W/m^2, 25^\circ C$, simulated/corrected IV curve @ STC. Measured and simulated/corrected curves match satisfying.

The comparisons of measured, simulated and corrected IV-curves in figure 4.4 and 4.5 are showing satisfying precision in reproducing the original modules IV-curve through the implemented setup.

For illustration of effects of the operation conditions on the modules IV-curves and IR-images further measurements had been performed including a defect module as real component in the simulated string.

In the following some of the yielded IV-curves and IR-images are depicted. The according simulation conditions can be derived from table 5.1 or from the filename directly, which is composed as "Modulename#SimModules#RealModules_SetpointModuletemp_SetpointIrradiation_ShadowingCondition".

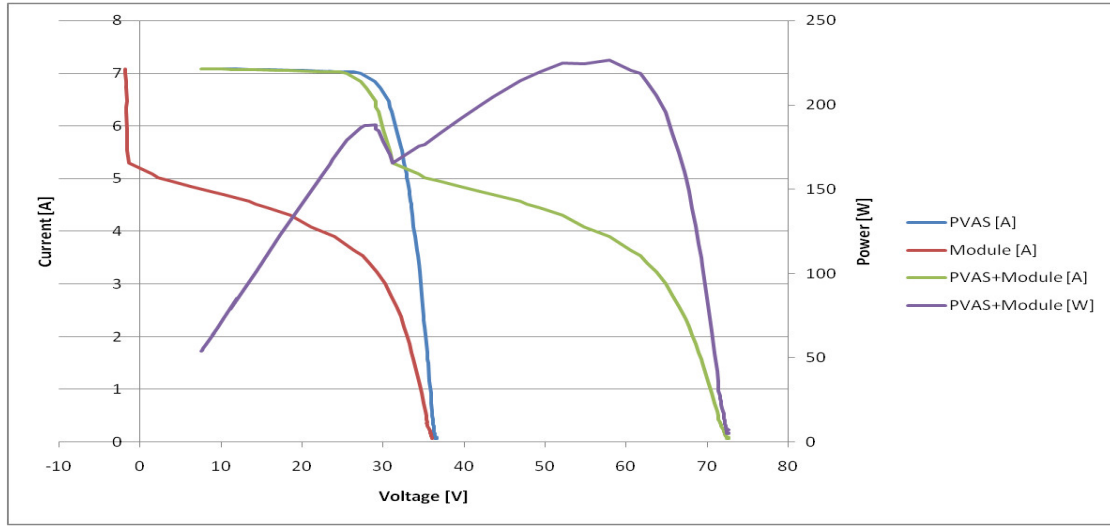
Because set point and measured values for irradiation and temperature differ by up to 5% (inaccuracy of the sun-simulators control system) the exact values of irradiation and module temperature at starting time of the experiment are mentioned in the captions of the diagrams.

IR-pictures of the modules are showing a temperature gradient, with a temperature drop in the range of 4-6°C, along their short frame side. This temperature drop is due to the air stream (air conditioning) inside the sun-simulator. The cooled air hits the module on the air entry side of the module (blue colored in the IR pictures), while passing the modules surface the air gets heated and therefore the opposing side of the module doesn't get cooled as the air entry side. An indicator for the direction of the air stream inside the sun simulator with respect to the module is depicted in figure 4.7(a) and applies for all of the IR-pictures. The hotspots on the right side in the middle of every module (big white spot) is due to the junction box mounted on the modules back surfaces, decreasing the cooling efficiency of the air stream at this area. Also the mounted PT100 sensors on the modules back sheets have the same effect and therefore cause heated spots as indicated in the IR-pictures, they are not necessarily indicating defects of the modules. For simulating shading of modules a black painted cardboard was used to shade single cells of the modules as can be seen in the according IR-images (figures 4.13, 4.17, 4.21).

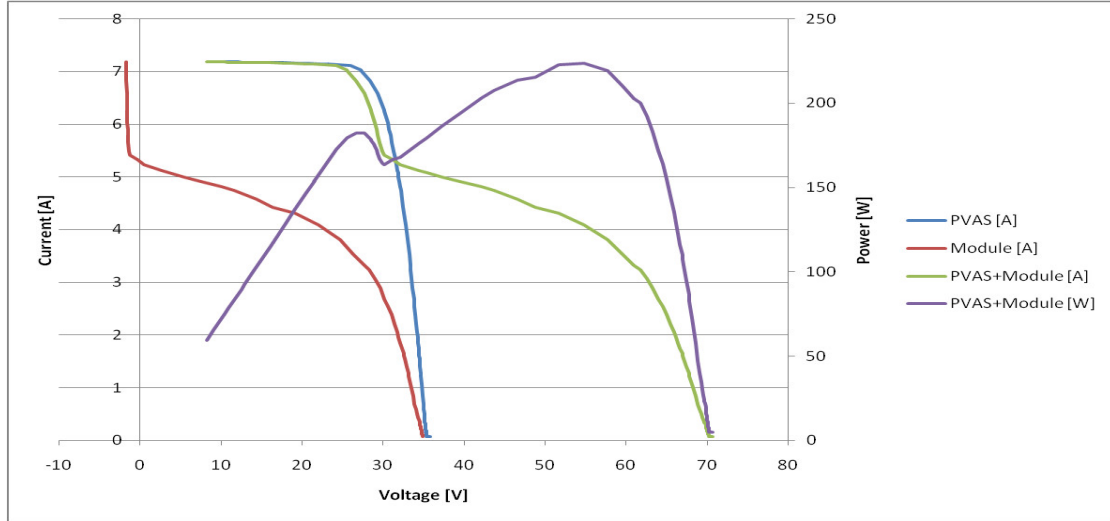
(Also some dynamic live condition experiments have been performed in outdoors. The ambient conditions contained free varying irradiation in the range of 200-1000W/m², temperature variation of the modules back sheets between 50-60°C and partially shadowing due to trees. The simulation setup worked stable during the dynamical outdoor tests. The complete set of data is available and stored digitally on AIT servers.)

The following diagrams depicting the measured data from the simulations. The purple graph P_{ges} depicts the overall systems power (PVAS+module) as a function of the overall systems voltage and therefore refers to the right diagram axes. All the other graphs are IV-curves and refer to the left diagram axes.

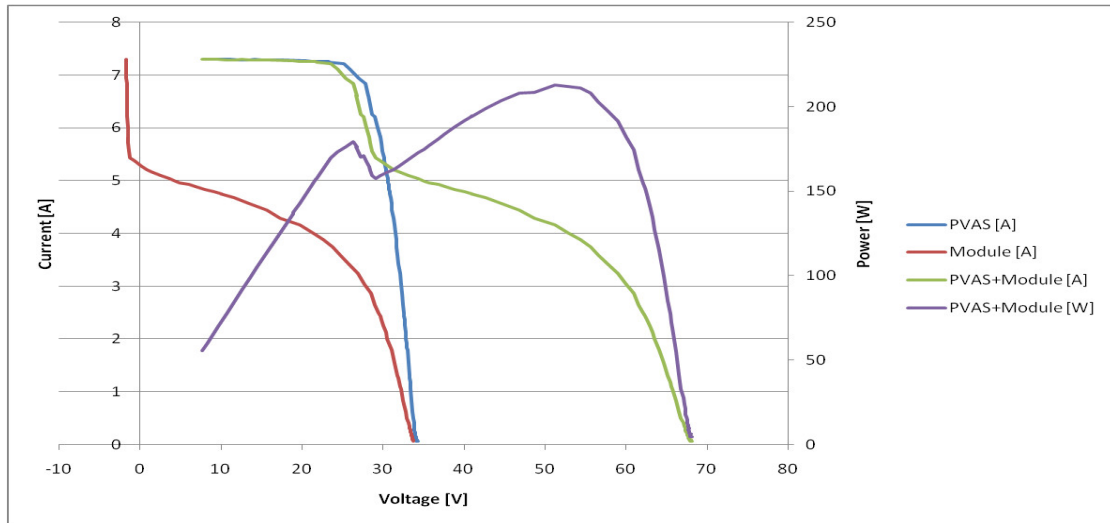
ATERSA A235P (Ser.Nr. 1031204010013, DH85/85)



(a) Atersa11.25-800.no, $G=808W/m^2$, $T_{Mod} = 25^{\circ}C$.

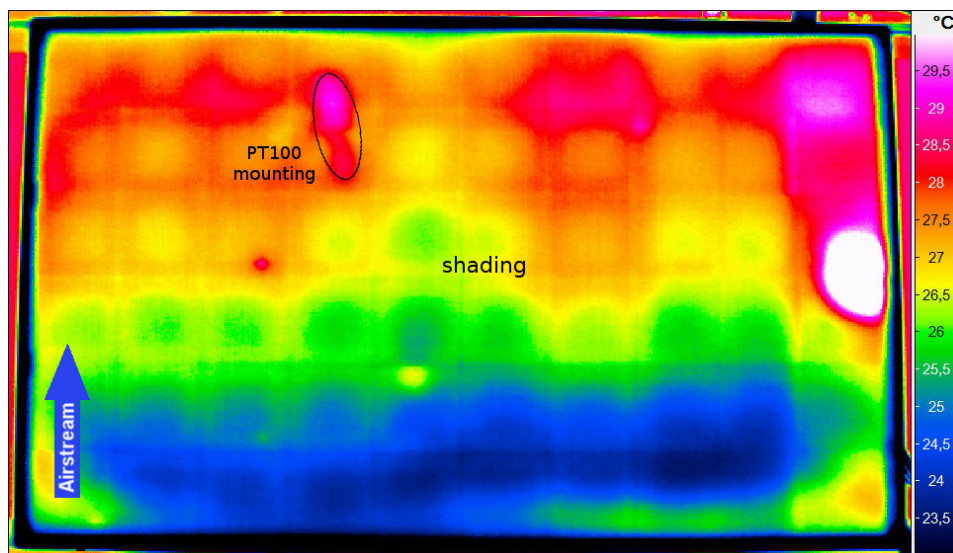


(b) Atersa11.35-800.no, $G=818W/m^2$, $T_{Mod} = 35,1^{\circ}C$.

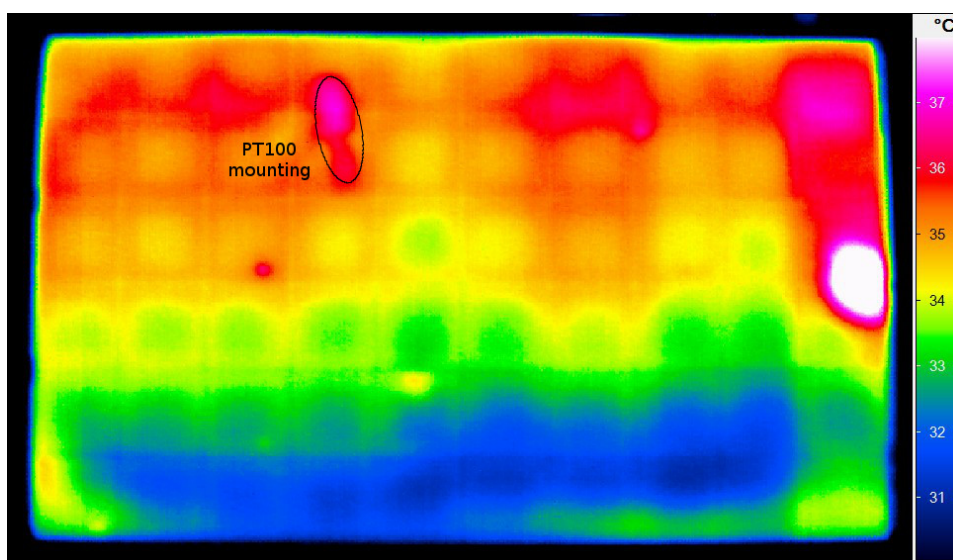


(c) Atersa11.45-800.no, $G=826W/m^2$, $T_{Mod} = 45,2^{\circ}C$.

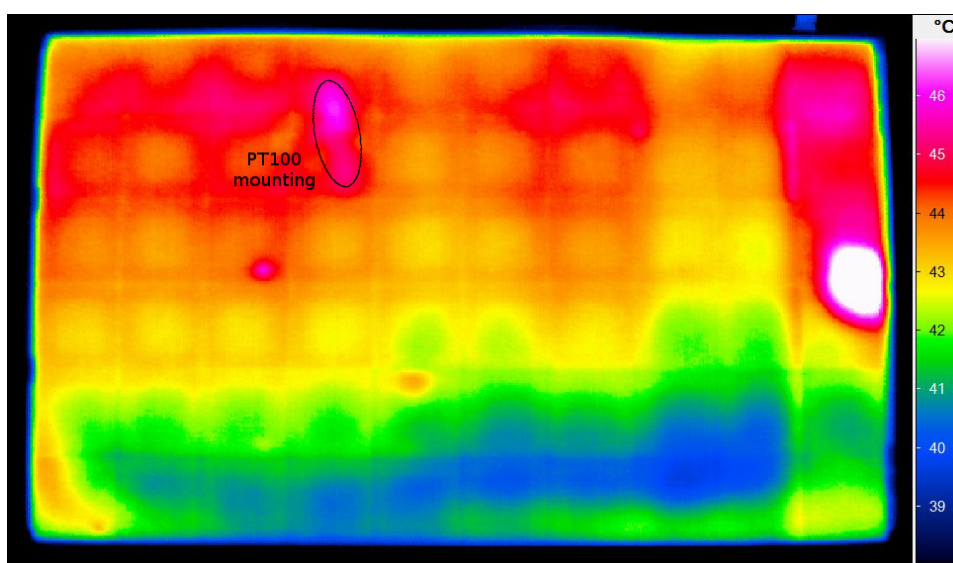
Figure 4.6: IV-diagrams of the Atersa module(DH, 1+1)



(a) Atersa11_25_800_no

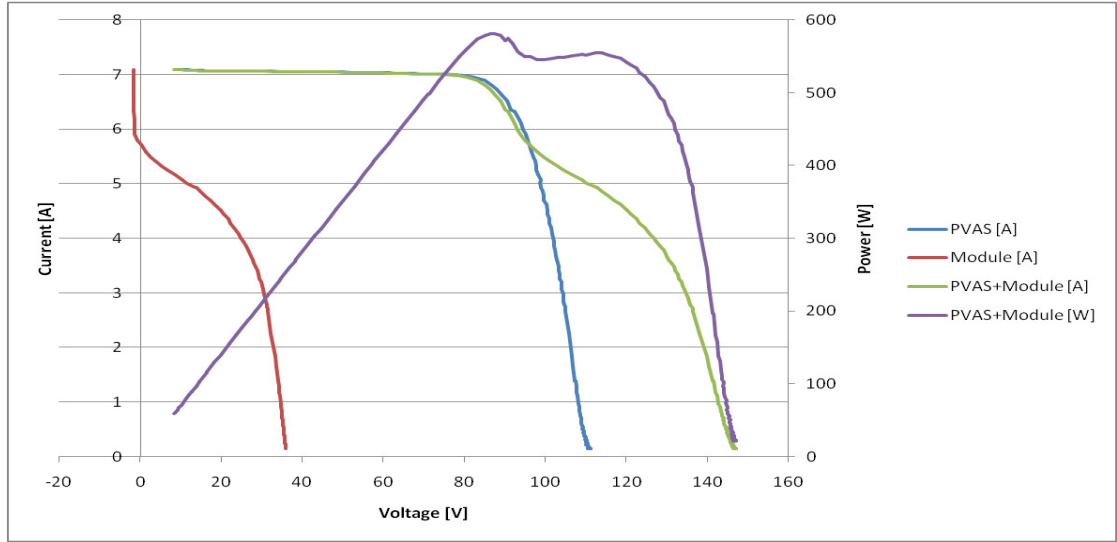


(b) Atersa11_35_800_no

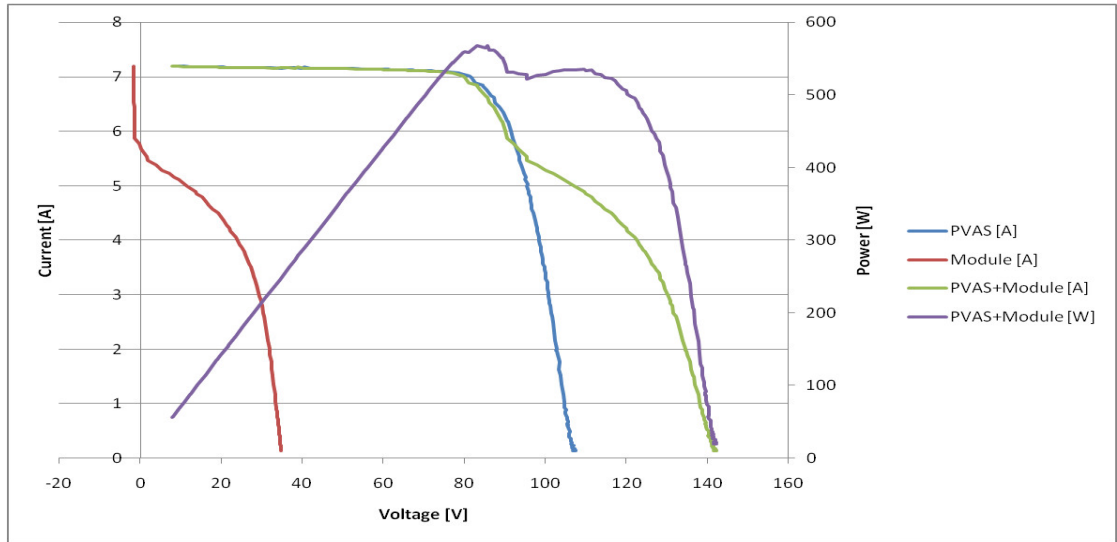


(c) Atersa11_45_800_no

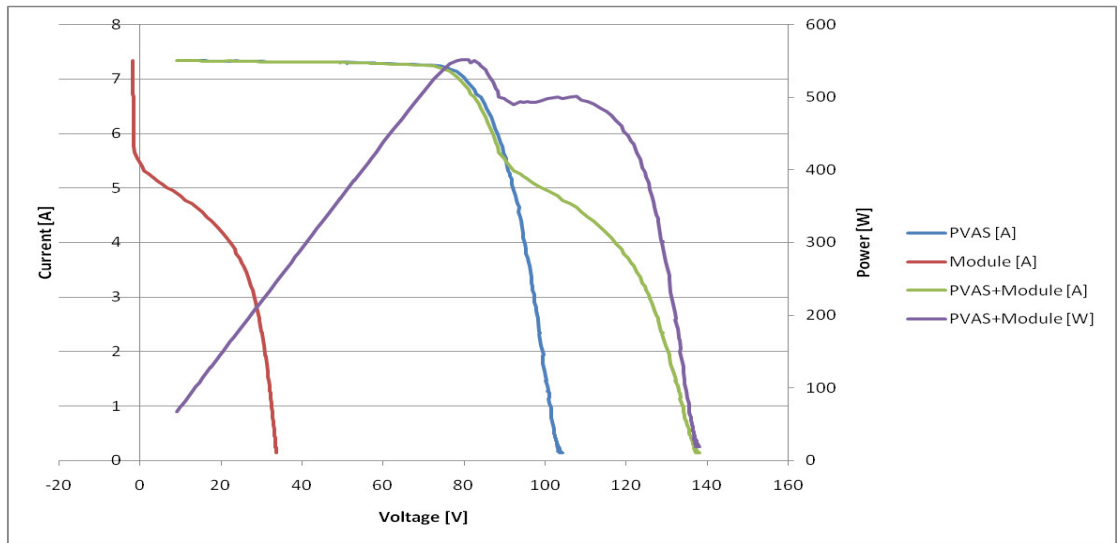
Figure 4.7: IR-images of the Atersa module(DH, 1+1)



(a) Atersa31_25_800_no, $G=806\text{W}/\text{m}^2$, $T_{Mod} = 25,8^\circ\text{C}$.

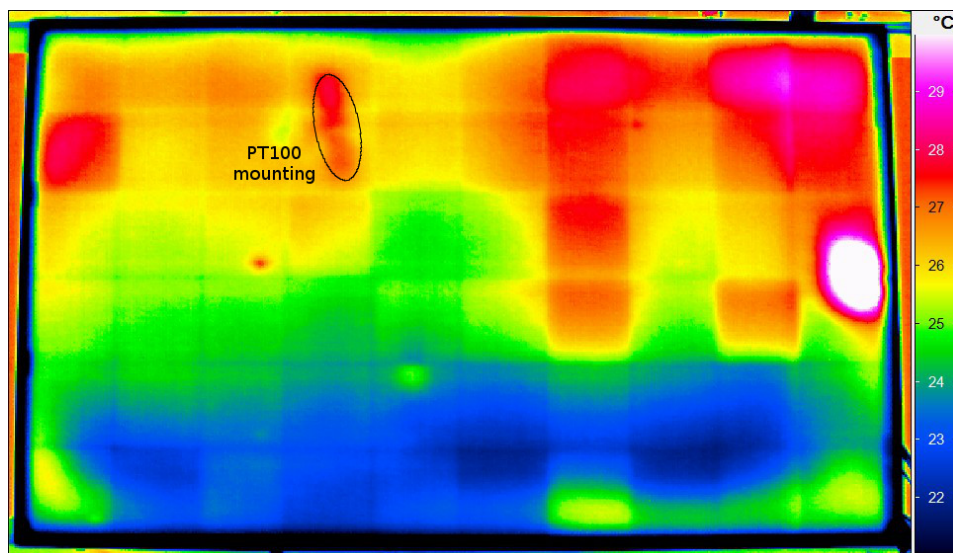


(b) Atersa31_35_800_no, $G=814\text{W}/\text{m}^2$, $T_{Mod} = 35,2^\circ\text{C}$.

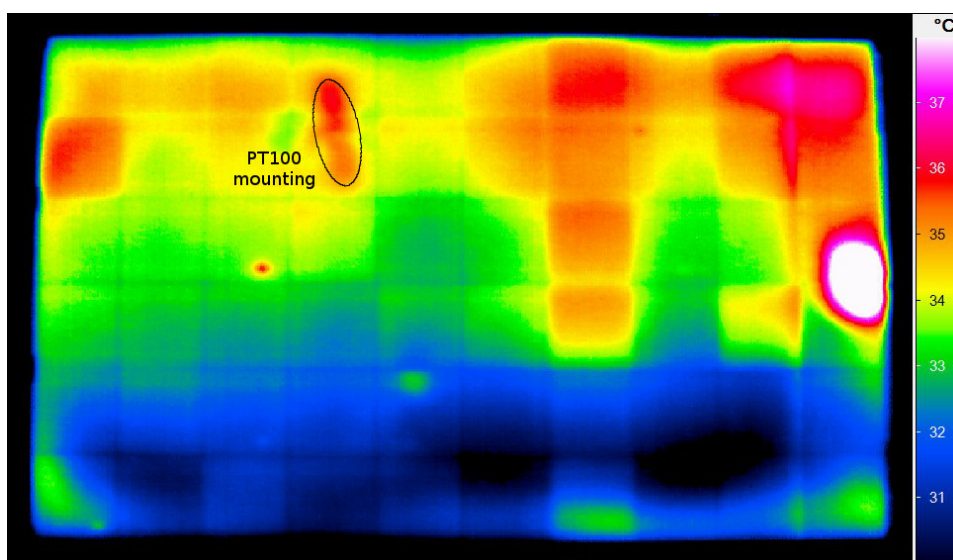


(c) Atersa31_45_800_no, $G=829\text{W}/\text{m}^2$, $T_{Mod} = 45,3^\circ\text{C}$.

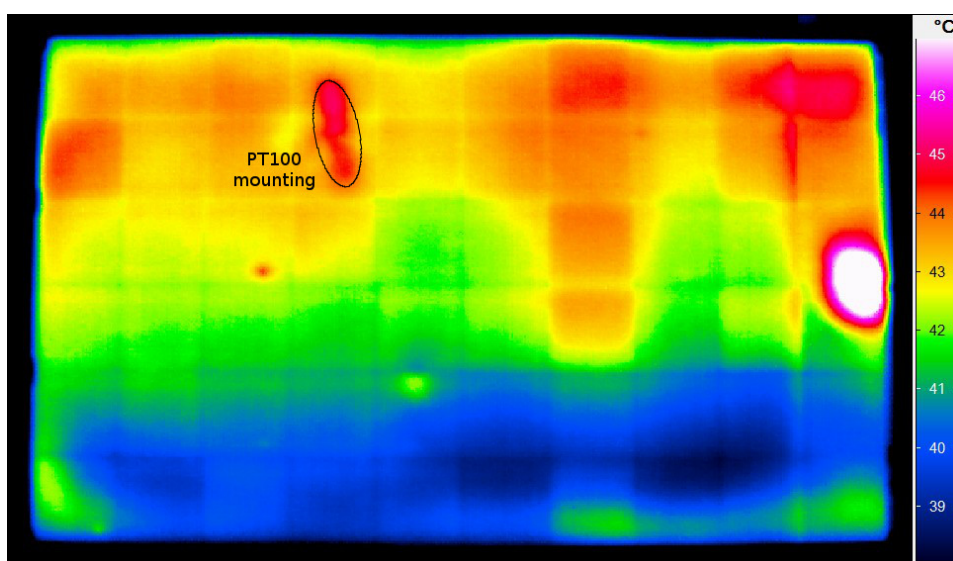
Figure 4.8: IV-diagrams of the Atersa module (DH, 3+1)



(a) Atersa31_25_800_no

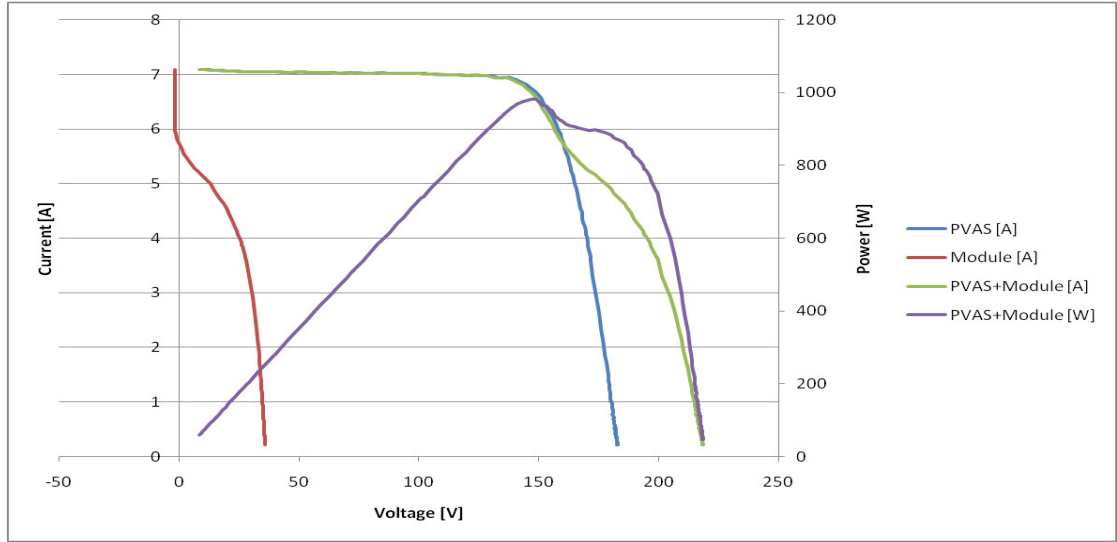


(b) Atersa31_35_800_no

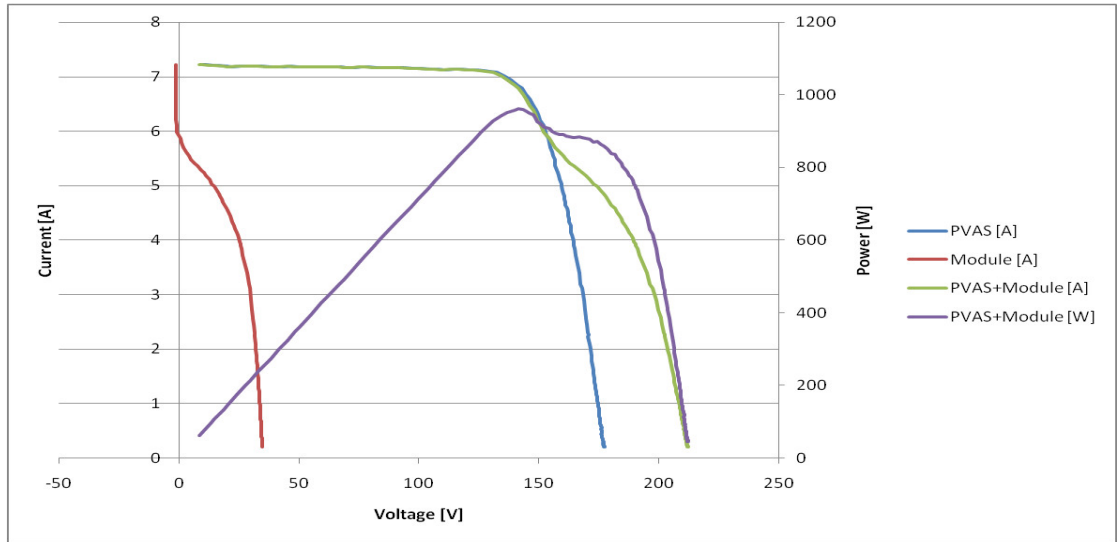


(c) Atersa31_45_800_no

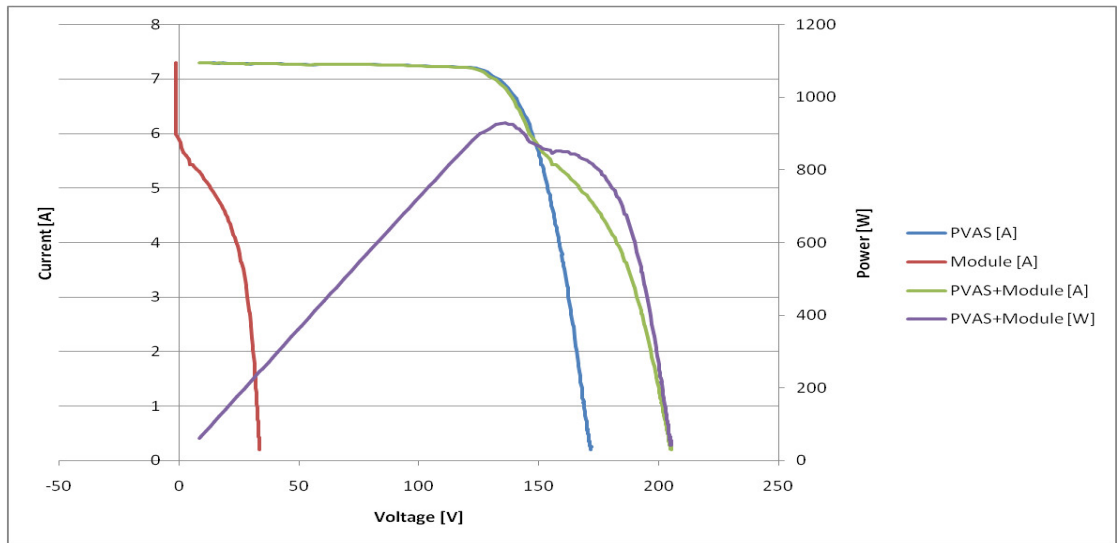
Figure 4.9: IR-images of the Atersa module (DH, 3+1)



(a) Atersa51_25_800_no, $G=805W/m^2$, $T_{Mod} = 25, 2^{\circ}C$.

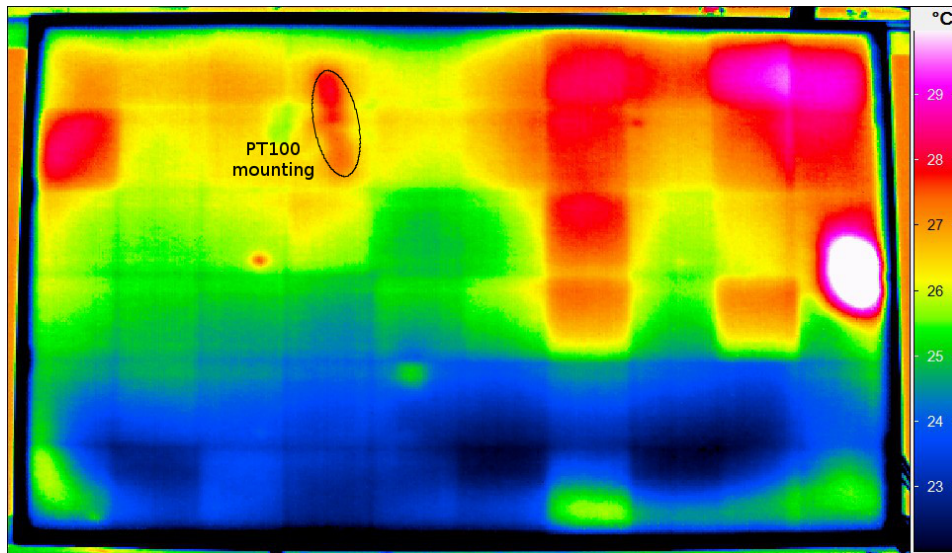


(b) Atersa51_35_800_no, $G=817W/m^2$, $T_{Mod} = 35, 1^{\circ}C$.

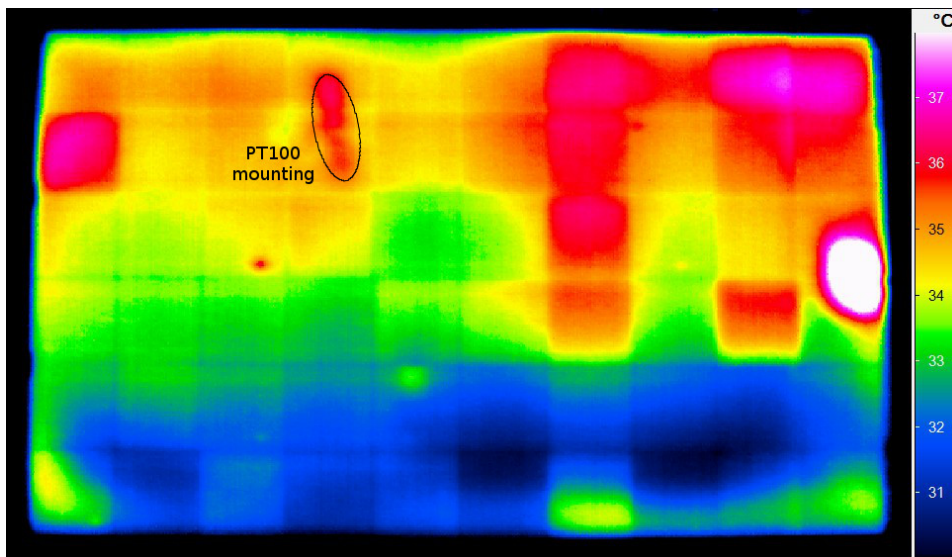


(c) Atersa51_45_800_no, $G=824W/m^2$, $T_{Mod} = 45, 1^{\circ}C$.

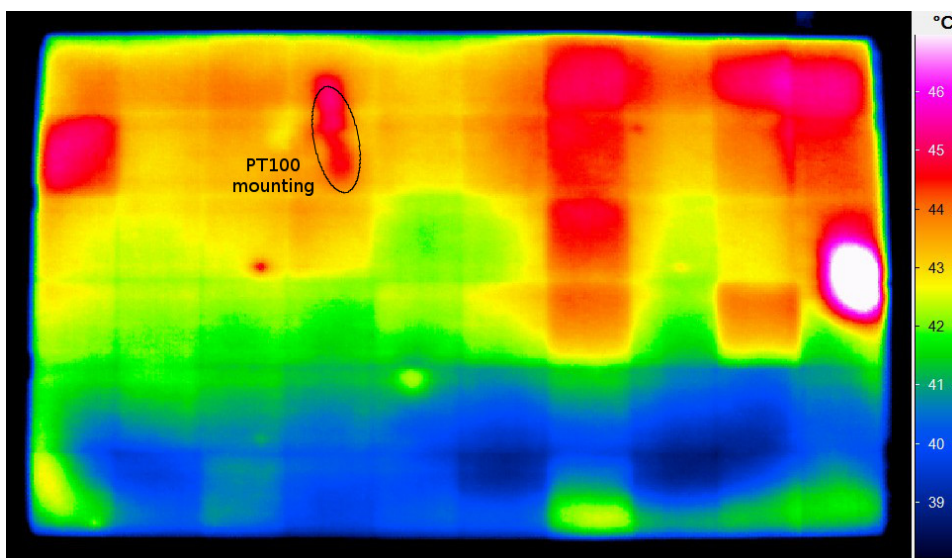
Figure 4.10: IV-diagrams of the Atersa module (DH, 5+1)



(a) Atersa51_25_800_no

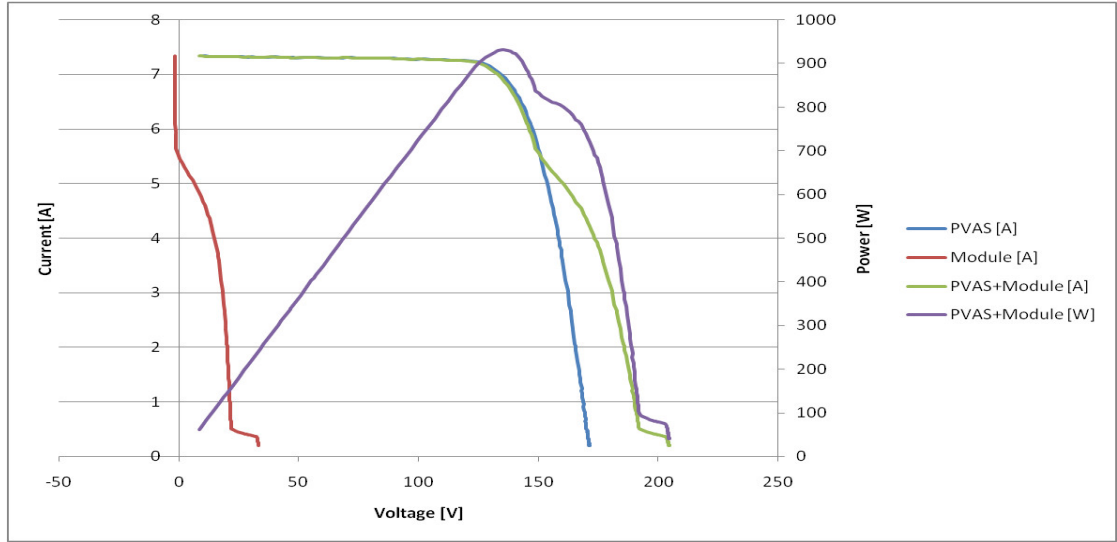


(b) Atersa51_35_800_no

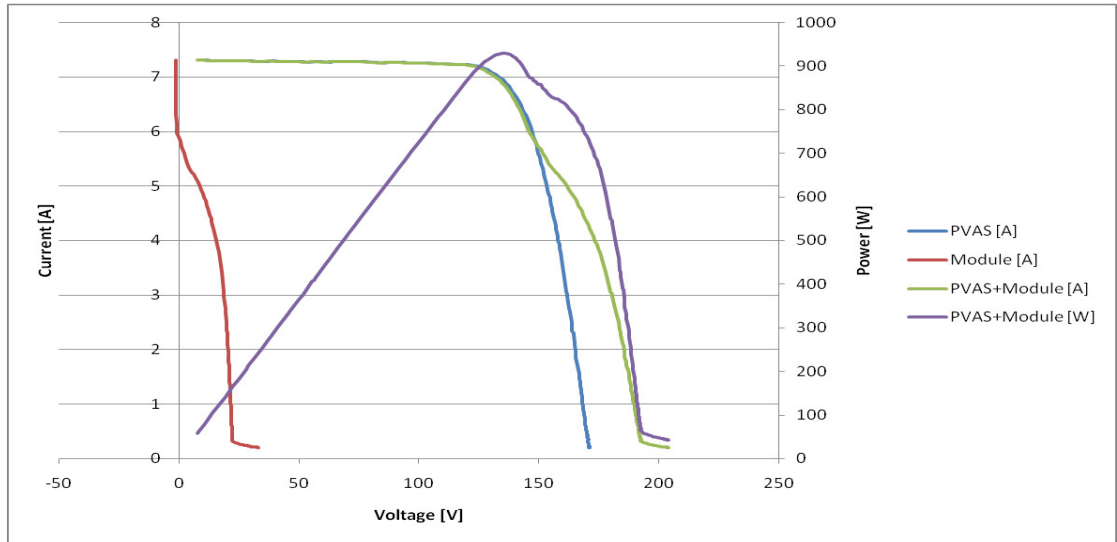


(c) Atersa51_45_800_no

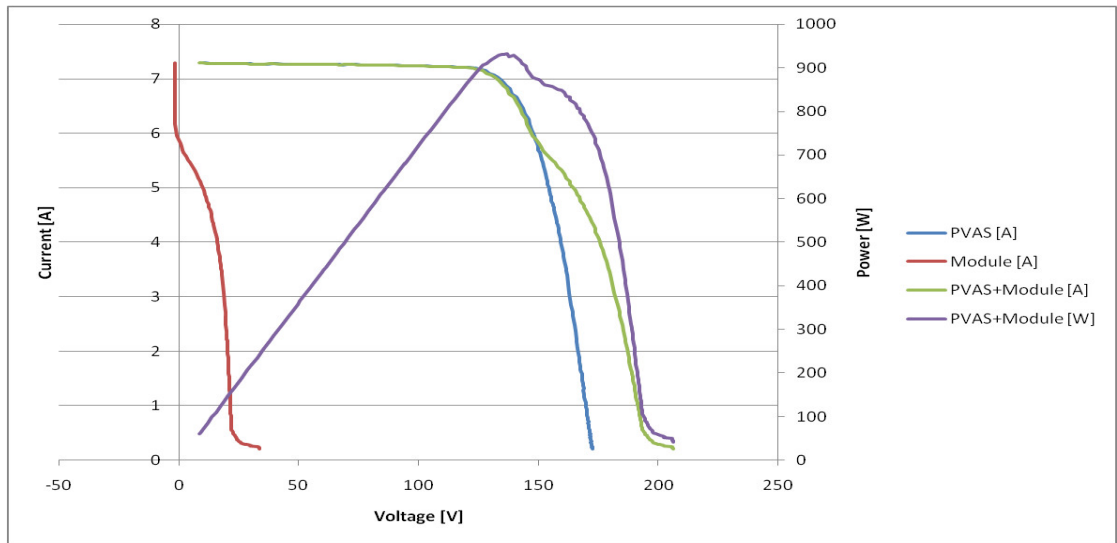
Figure 4.11: IR-images of the Atersa module (DH, 5+1)



(a) Atersa51.45.800_str1, $G=828\text{W}/\text{m}^2$, $T_{Mod} = 44,7^\circ\text{C}$.

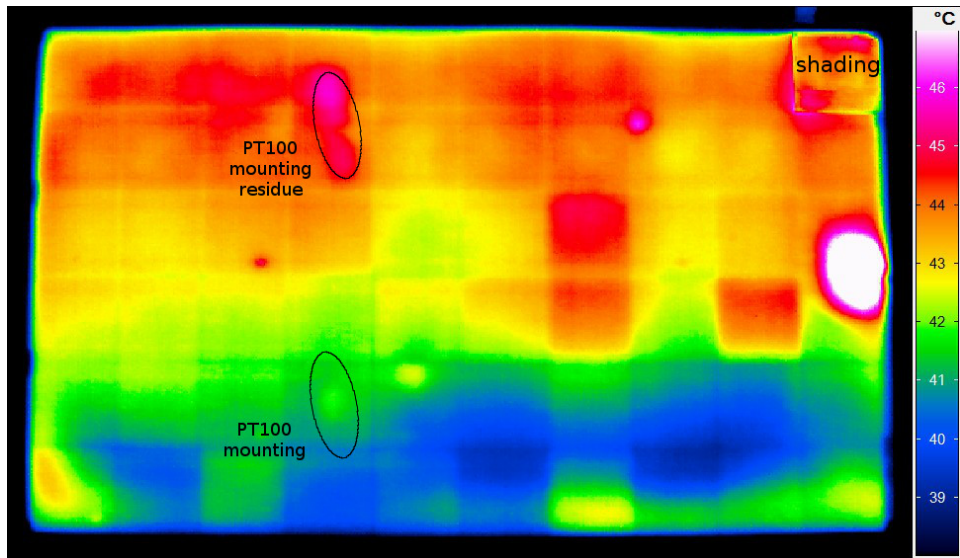


(b) Atersa51.45.800_str2, $G=825\text{W}/\text{m}^2$, $T_{Mod} = 44,3^\circ\text{C}$.

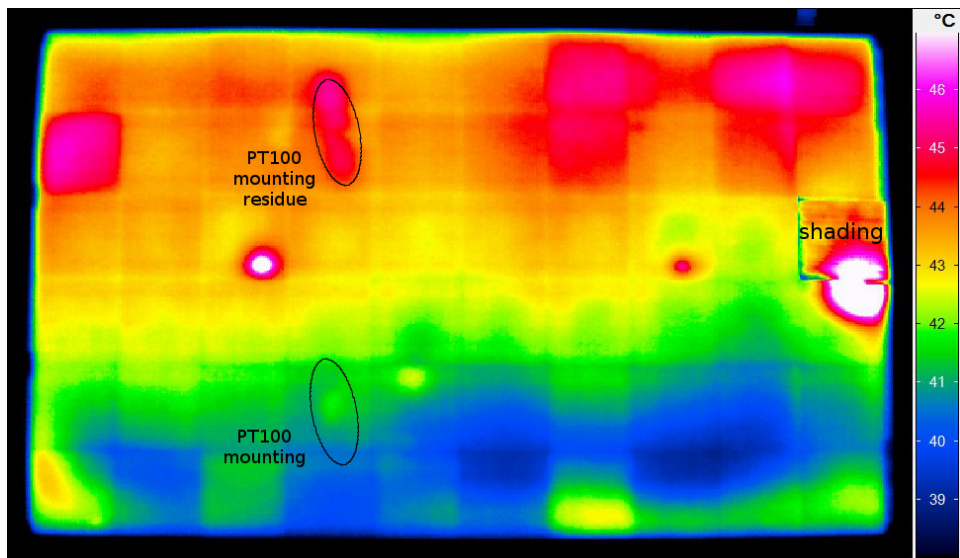


(c) Atersa51.45.800_str3, $G=823\text{W}/\text{m}^2$, $T_{Mod} = 43^\circ\text{C}$.

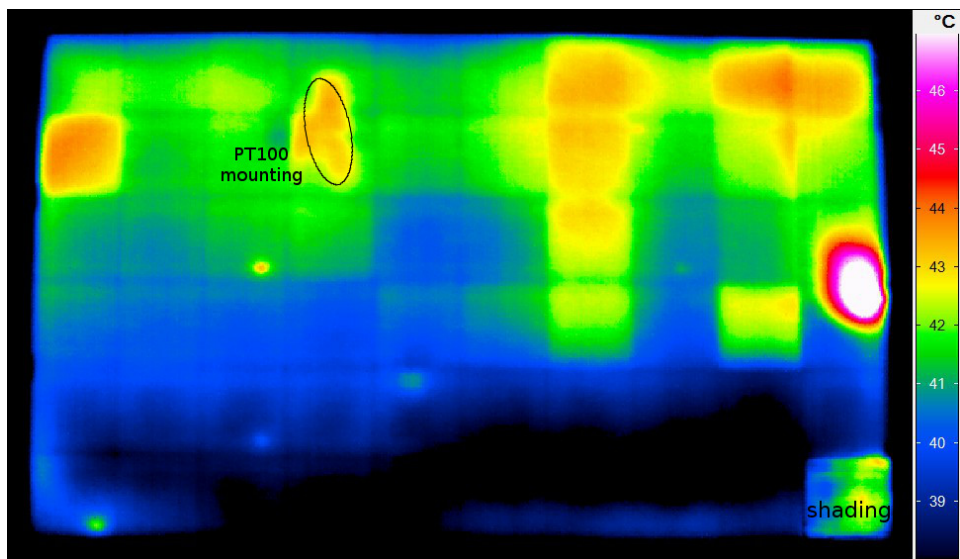
Figure 4.12: IV-diagrams of the Atersa module (DH, 5+1, shadowing)



(a) Atersa51.45.800_str1



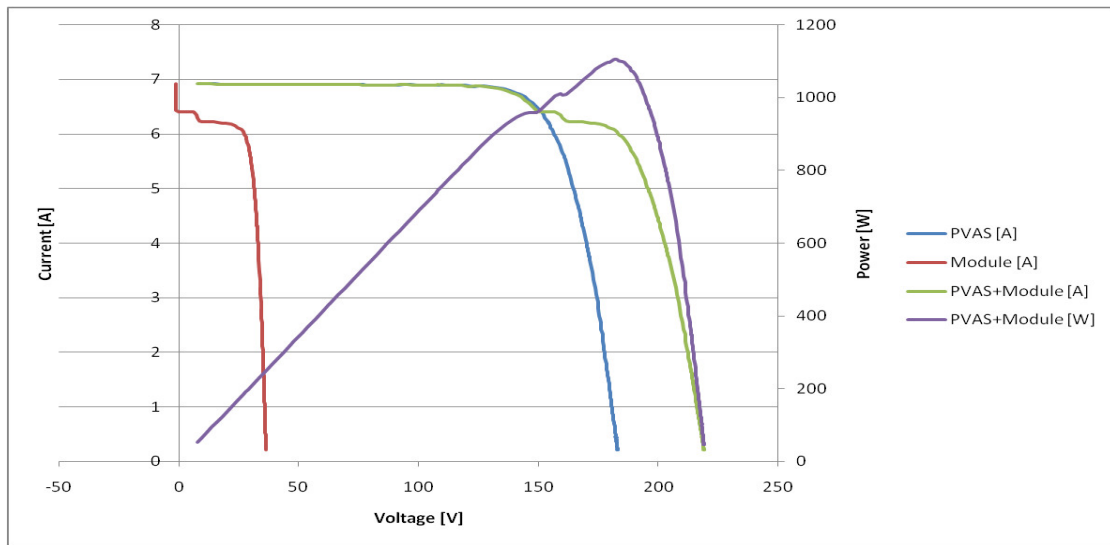
(b) Atersa51.45.800_str2



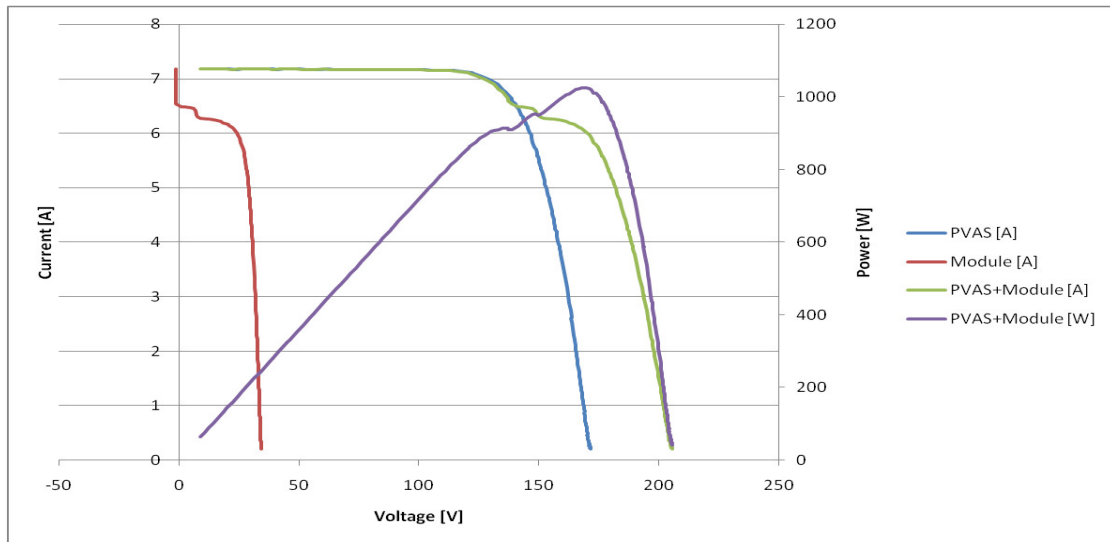
(c) Atersa51.45.800_str3

Figure 4.13: IR-images of the Atersa module (DH, 5+1, shadowing)

Auversun AV240P60 (Ser.Nr. 1204242019, DH85/85)

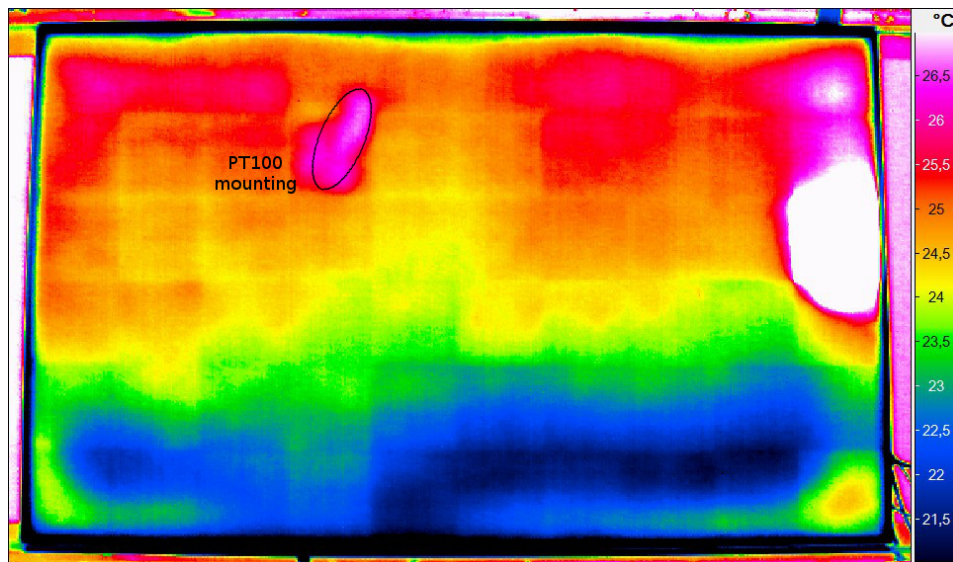


(a) Auversun51.25.800_no, $G=801W/m^2$, $T_{Mod} = 25^{\circ}C$.

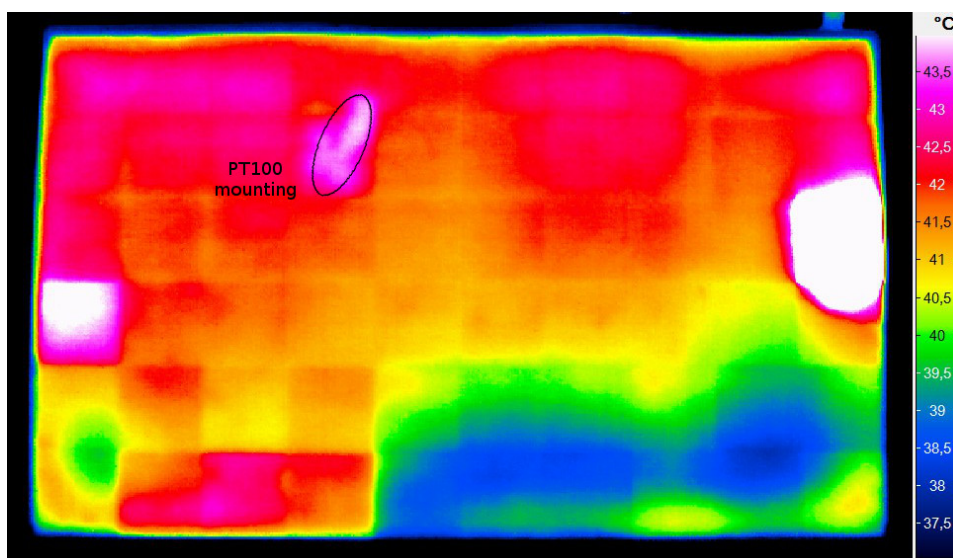


(b) Auversun51.45.800_no, $G=824W/m^2$, $T_{Mod} = 44,2^{\circ}C$.

Figure 4.14: IV-diagrams of the Auversun module (DH, 5+1)

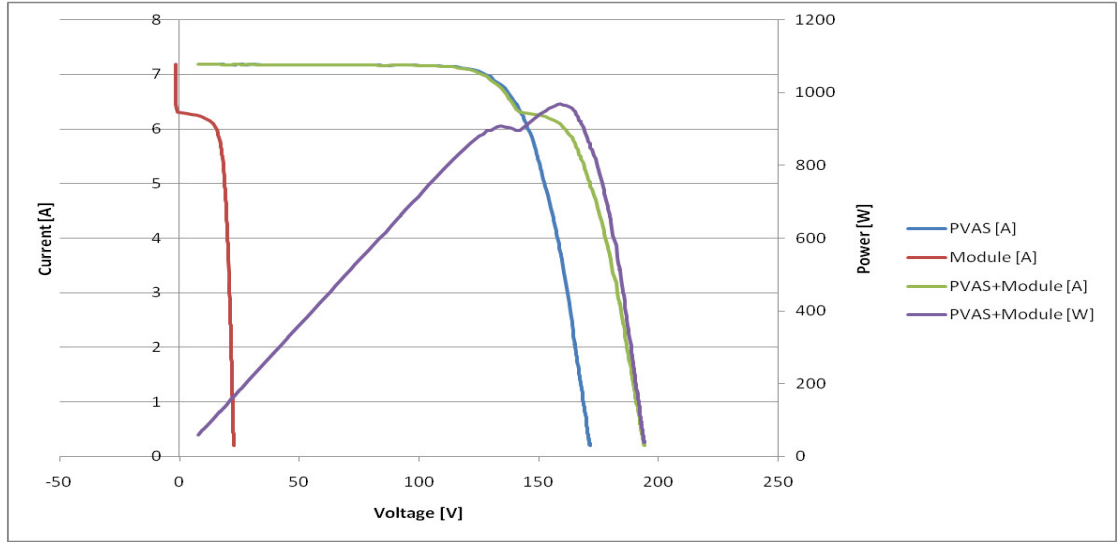


(a) Auversun51_25_800_no

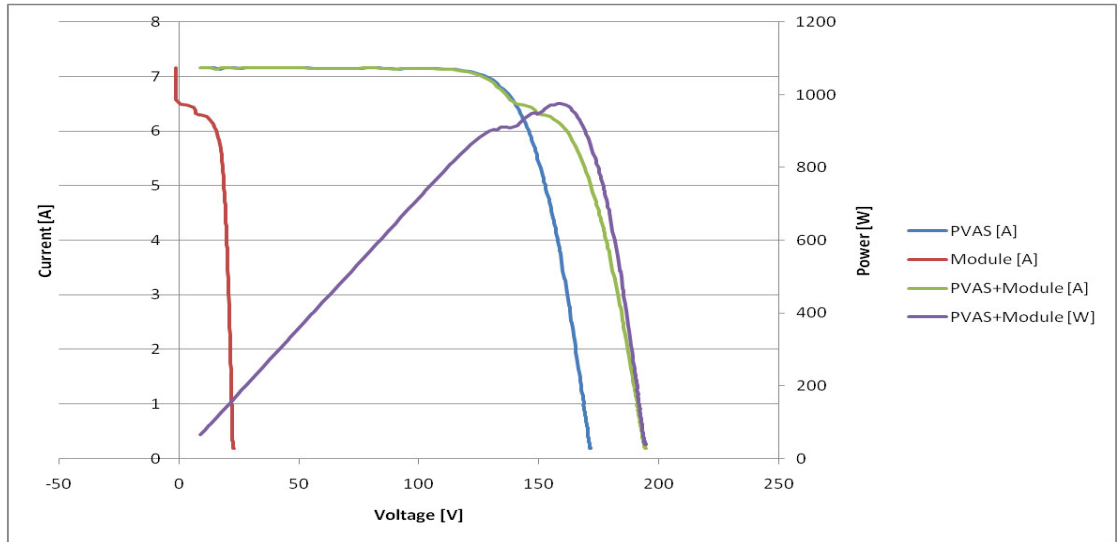


(b) Auversun51_45_800_no

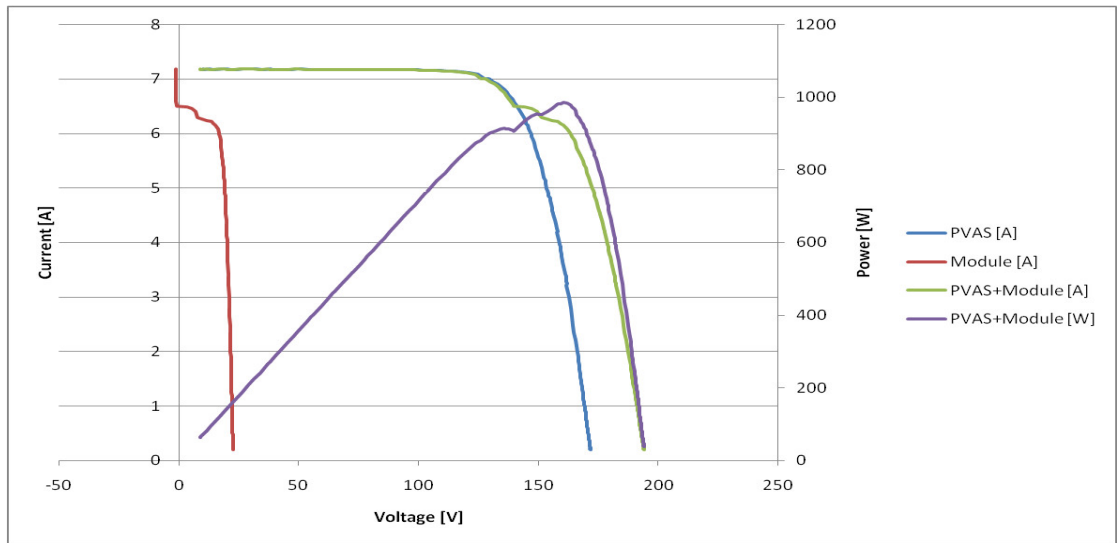
Figure 4.15: IR-images of the Auversun module (DH, 5+1)



(a) Auversun51.45.800-str1, $G=822W/m^2$, $T_{Mod} = 46, 4^\circ C$.

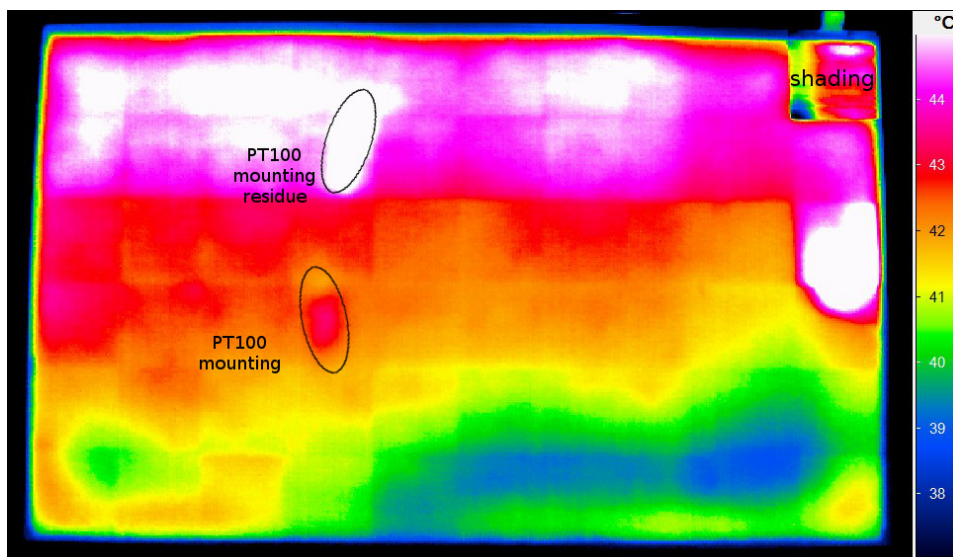


(b) Auversun51.45.800-str2, $G=821W/m^2$, $T_{Mod} = 44, 4^\circ C$.

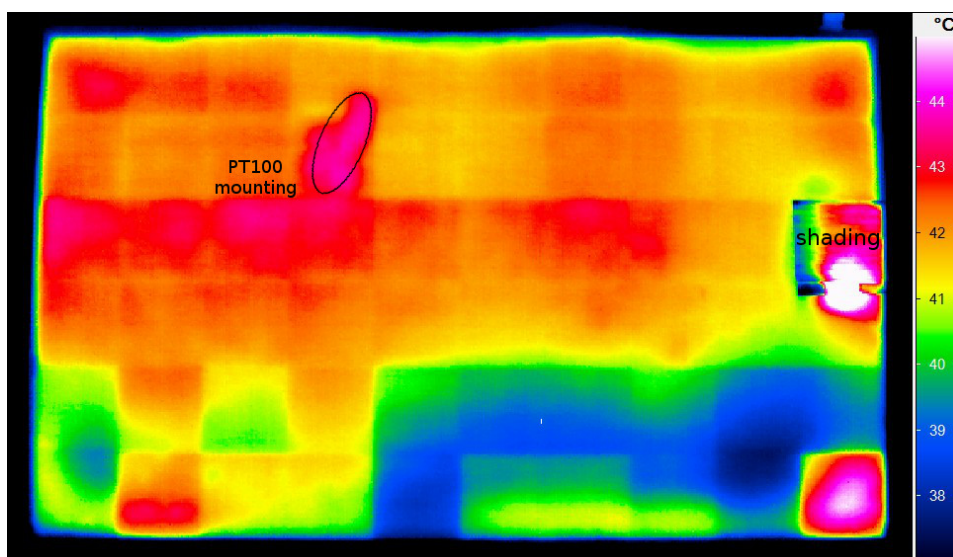


(c) Auversun51.45.800-str3, $G=823W/m^2$, $T_{Mod} = 44, 2^\circ C$.

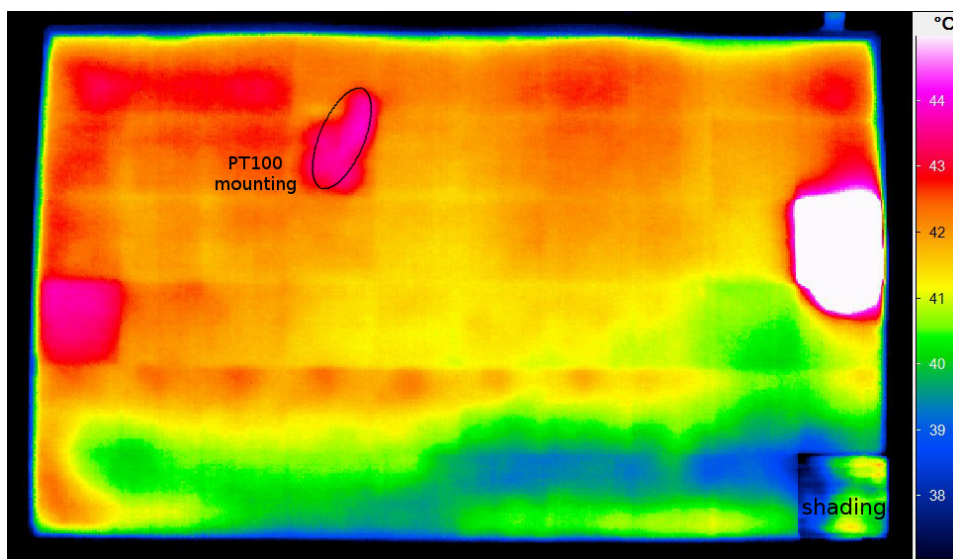
Figure 4.16: IV-diagrams of the Auversun module (DH, 5+1, shadowing)



(a) Auversun51.45.800_str1



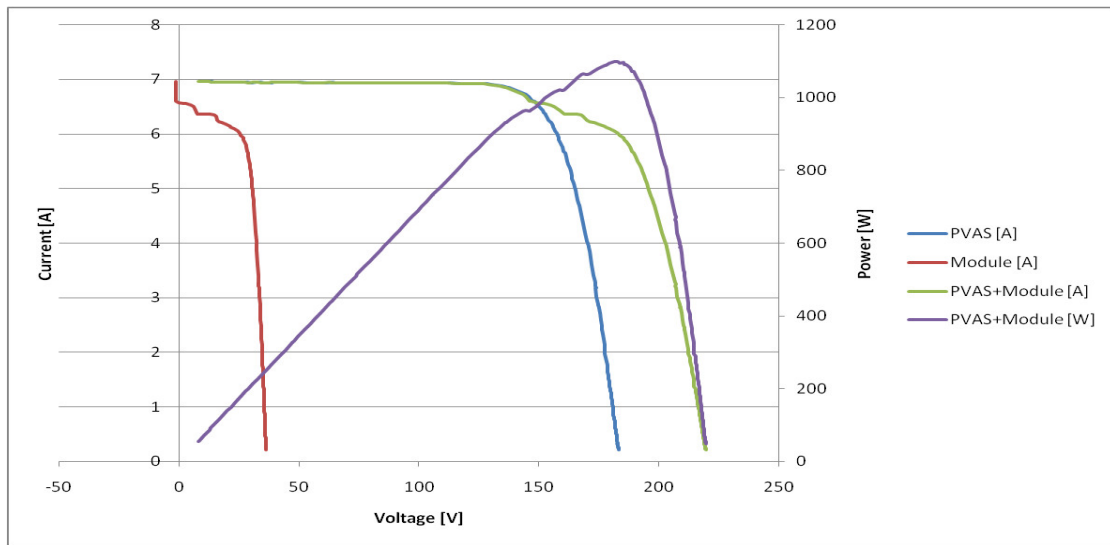
(b) Auversun51.45.800_str2



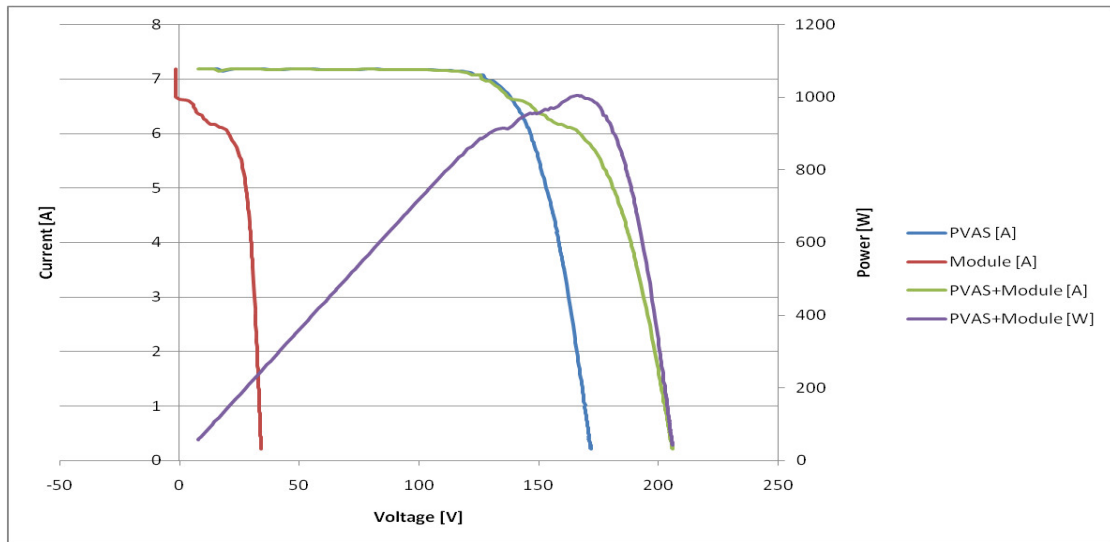
(c) Auversun51.45.800_str3

Figure 4.17: IR-images of the Auversun module (DH, 5+1, shadowing)

Auversun AV240P60 (Ser.Nr. 1204242022, TC-40/85)

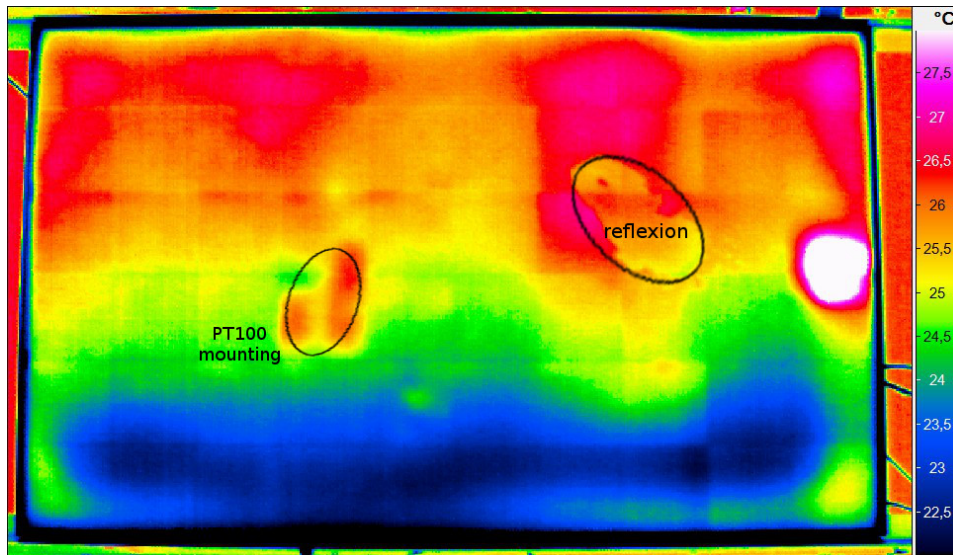


(a) AuversunTC51_25_800_no, $G=807W/m^2$, $T_{Mod} = 24,3^{\circ}C$.

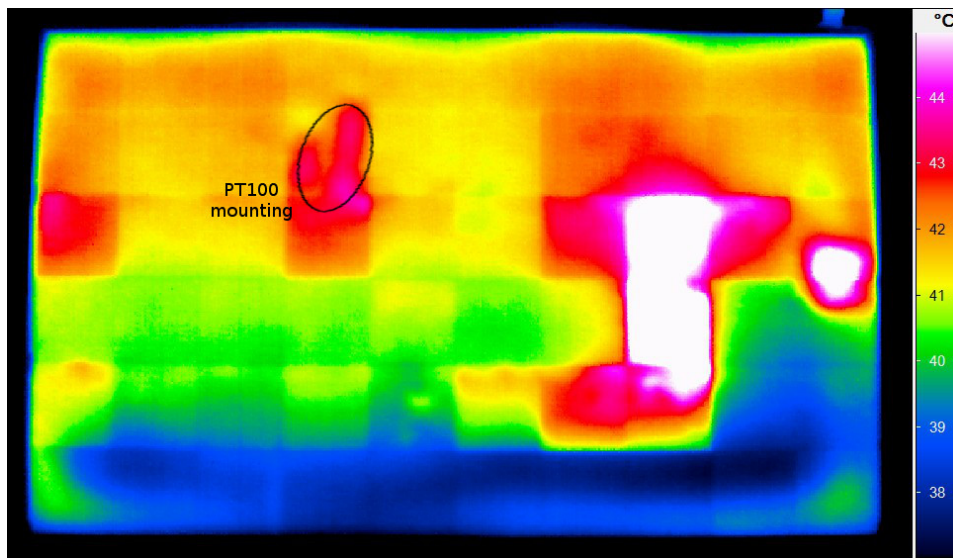


(b) AuversunTC51_45_800_no, $G=824W/m^2$, $T_{Mod} = 44,4^{\circ}C$.

Figure 4.18: IV-diagrams of the Auversun module (TC, 5+1)

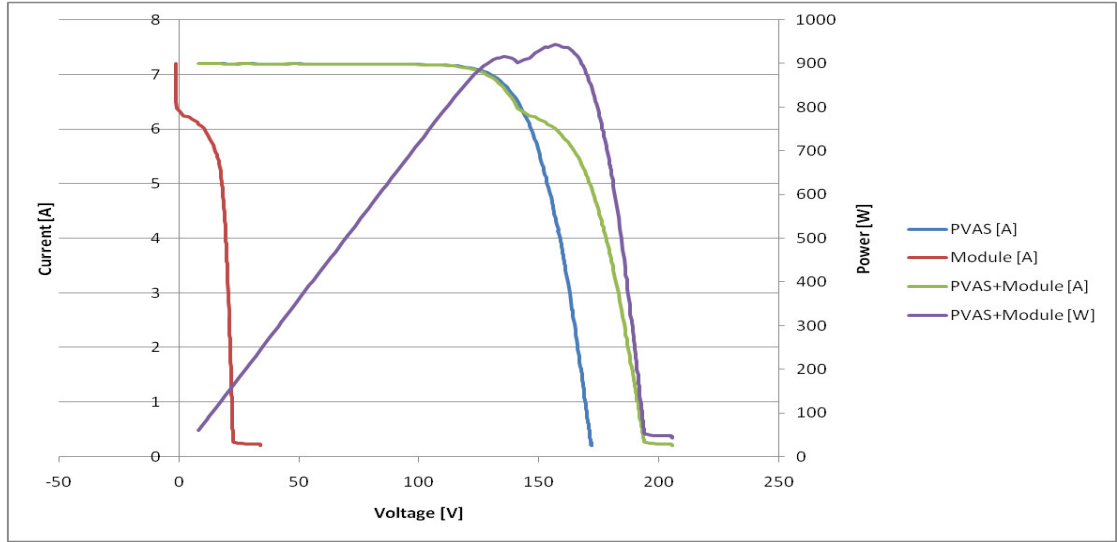


(a) AuversunTC51_25_800_no. The area in the upper right, looks like a water stain, is a reflection from the anti reflection sheet. During simulation condensing water from installations on the ceiling dropped down onto the sheets and caused a locally decrease of infrared radiation from the sheet.

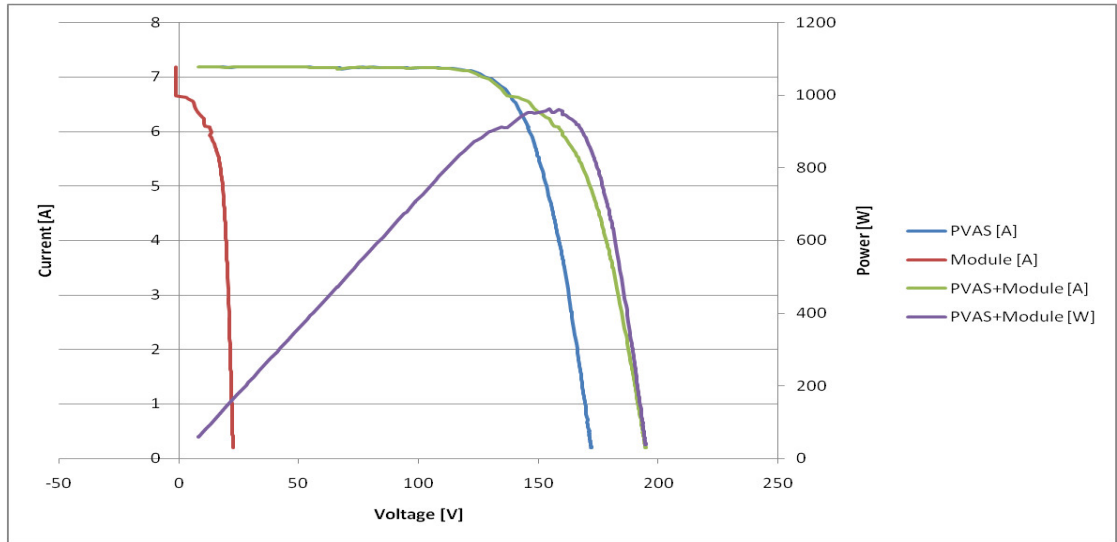


(b) AuversunTC51_45_800_no

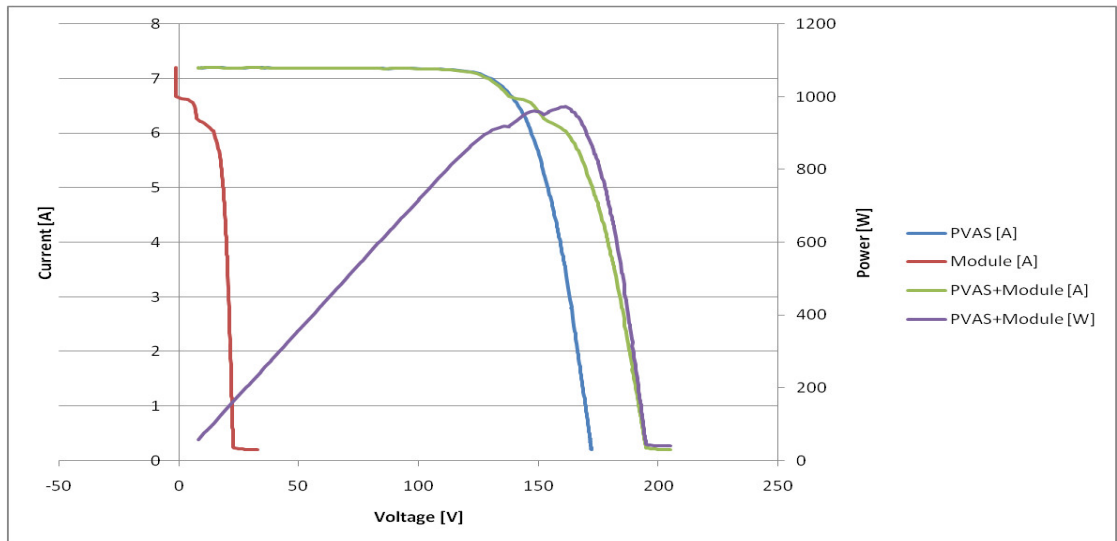
Figure 4.19: IR-images of the Auversun module (TC, 5+1)



(a) AuversunTC51.45.800_str1, $G=825\text{W}/\text{m}^2$, $T_{Mod} = 44, 2^\circ\text{C}$.

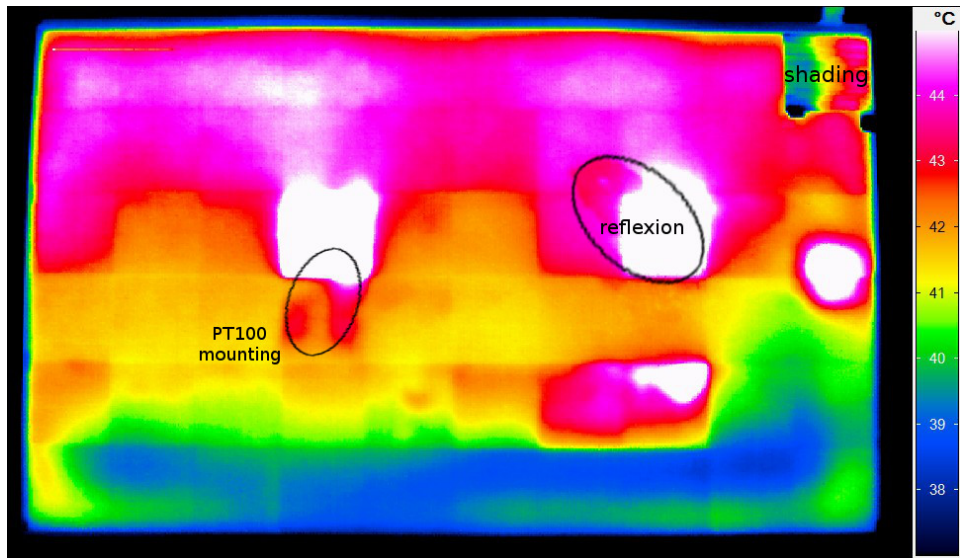


(b) AuversunTC51.45.800_str2, $G=822\text{W}/\text{m}^2$, $T_{Mod} = 44, 5^\circ\text{C}$.

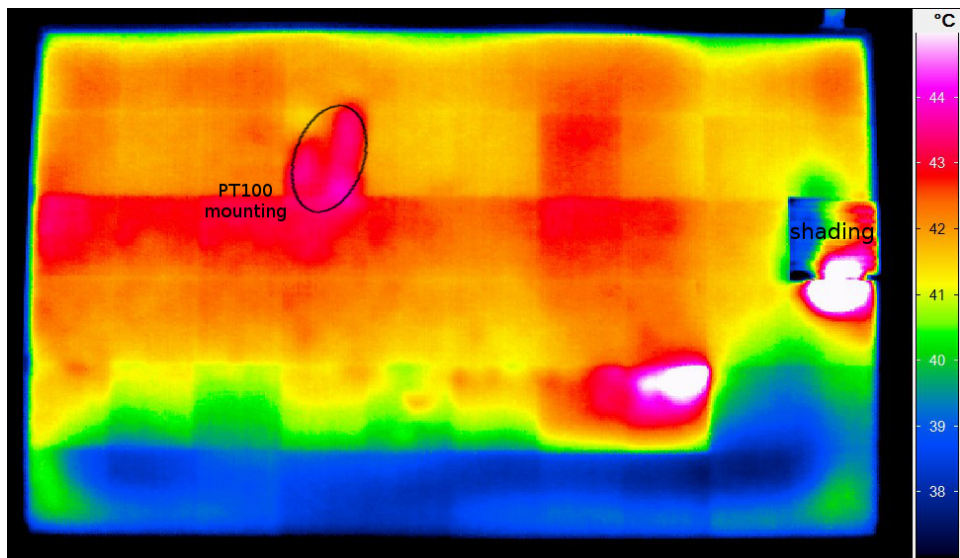


(c) AuversunTC51.45.800_str3, $G=825\text{W}/\text{m}^2$, $T_{Mod} = 44, 3^\circ\text{C}$.

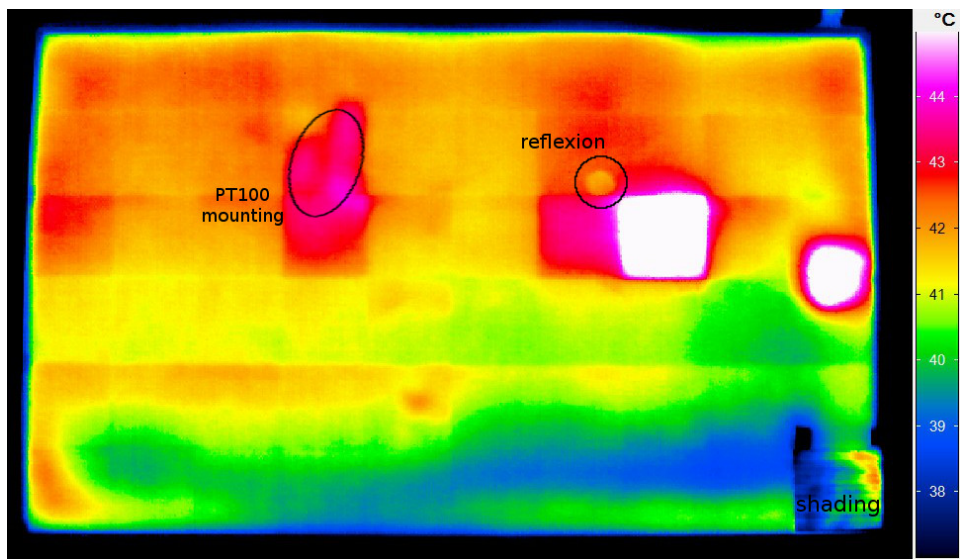
Figure 4.20: IV-diagrams of the Auversun module (TC, 5+1, shadowing)



(a) AuversunTC51.45.800_str1



(b) AuversunTC51.45.800_str2



(c) AuversunTC51.45.800_str3

Figure 4.21: IR-images of the Auversun module (TC, 5+1, shadowing)

Chapter 5

Conclusion

The installed setup has shown to work properly in simulating a photovoltaic string. The experiments gave satisfying results with respect to the IV-curve simulation. The target-PC's configuration combined with the simulation time step of $100\mu s$ performed well so the simulation never stopped because of execution time exceedance. The use of a manually variable resistor, instead of an inverter, in the circuit, caused oscillations of the setup in regions of the simulated IV-curve next to the open circuit voltage. A combination of the variable resistor and an $300\mu F$ capacitor was able to avoid oscillations or instabilities of the setup during all of the experiments. Installed cotton sheets provided a suitable IR-capturing setup to cancel inhomogeneous reflections of the environment on the modules surface, which would have been captured by the IR-camera. Test of IR-capturing without the cotton sheets showed heavy reflections from installations on the ceiling such as ventilation system and lighting. Response of the simulation to changes in operation conditions, such as irradiance, temperature and shadowing showed behavior as expected from real modules. The accuracy of the setup is of a degree high enough to perform further investigations on the effect of defect modules in a photovoltaic string. For a more accurate investigation with respect to IR imaging, maybe some modifications to the used sun simulator are necessary to get more homogeneous temperature distribution over the modules area. Instead outdoor irradiation of the module could be performed including string simulation at outdoor conditions to get more life like conditions with respect to the module temperature.

For a more accurate modeling of the PV modules, the representations of the series and shunt resistances in the simulink model could be extended to be functions of temperature and irradiance both.

Finally this setup has shown to be able to ease future investigations on photovoltaic modules and further increase the understanding of behavior of photovoltaic arrays to finally get more efficiency in the goal of producing renewable energy systems.

Bibliography

- [1] De Soto W., Klein S.A., Beckman W.A., 2004. Improvement and Validation of a Model for Photovoltaic Array Performance, Solar Energy Laboratory, University of Wisconsin-Madison. Solar Energy 80(2006) 78-88.
- [2] International Standard, IEC60891, Edition2, 2009-12.
- [3] Duffie J.A., Beckman W.A., 1991. Solar Engineering of Thermal Processes, second ed. John Wiley & Sons Inc., New York.
- [4] O. Madelung, "Semiconductors: Data Handbook, 3rd ed." ISBN 3-540-40488-0
- [5] Nelson J., 2003. The Physics of Solar Cells. Imperial College Press, London.
- [6] Kishore Kumar Yamana, Master thesis, Real Time Power Hardware in the Loop Simulation - Theory and Applications in the Field of Electrical Energy Systems, Austrian Institute of Technology, Vienna.
- [7] <https://pvpmc.sandia.gov>
- [8] Project SOPHIA, PhotoVoltaic European Research Infrastructure, GA N° 262533, D9.4 - Aging testing on test samples completed
- [9] www.pveducation.org
- [10] www.physics.udel.edu
- [11] <http://c125.chem.ucla.edu>
- [12] Klein, S., 2005. EES-Engineering Equation Solver, F-Chart Software. Available from: www.fchart.com.
- [13] King, D. et al, 2004, "Sandia Photovoltaic Array Performance Model", SAND Report 3535, Sandia National Laboratories, Albuquerque, NM

I have tried to provide the information about the owner of pictures and figures in this thesis and obtained their approval. In the case of copyright infringement I ask you kindly for your declaration.

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

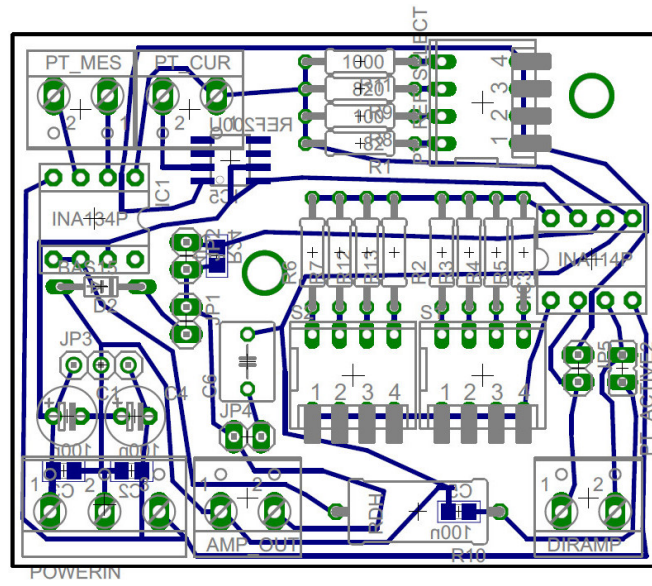
Appendix A

Appendix

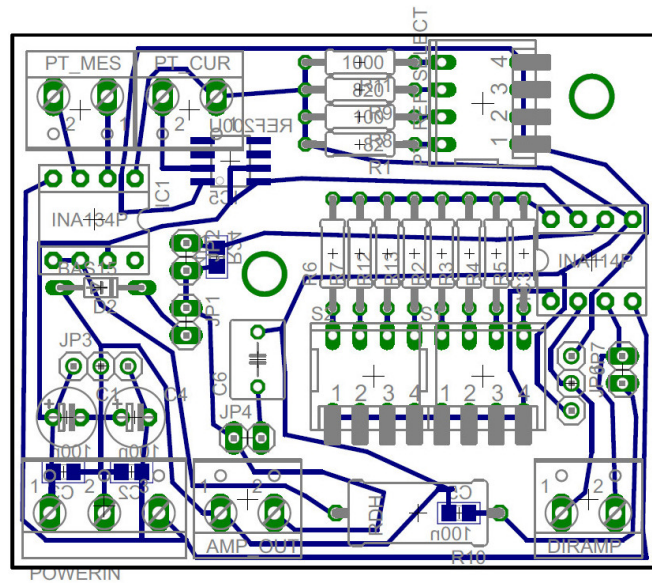
A.1 List of Used Abbreviations

AIT	Austrian Institute of Technology
AM	Air Mass
CIGS	Copper-Indium-Gallium-Diselenid
DAQ	Data AQuisition
DUT	Device Under Test
eV, meV	ElectronVolts, milliElectronVolts
FS	Floating Source
GaAs	Gallium-Arsenide
GS	Grounded Source
GUI	Graphical User Interface
HIL	Hardware In the Loop
HOMO	Highest Occupied Molecular Orbital
IR	InfraRed
IV	Current-Voltage (german abbreviation)
MPP	Maximum Power Point
NOCT	Nominal Operating Condition Temperature
PHIL	Power Hardware In the Loop
PV	PhotoVoltaic
PVAS	PhotoVoltaic Array Simulator
PVPMC	PhotoVoltaic Performance Modeling Collaborative
PVPP	PhotoVoltaic Power Plant
ROS	Rest Of the System
STC	STandard Conditions

A.2 Amplifier Circuitry Schematics

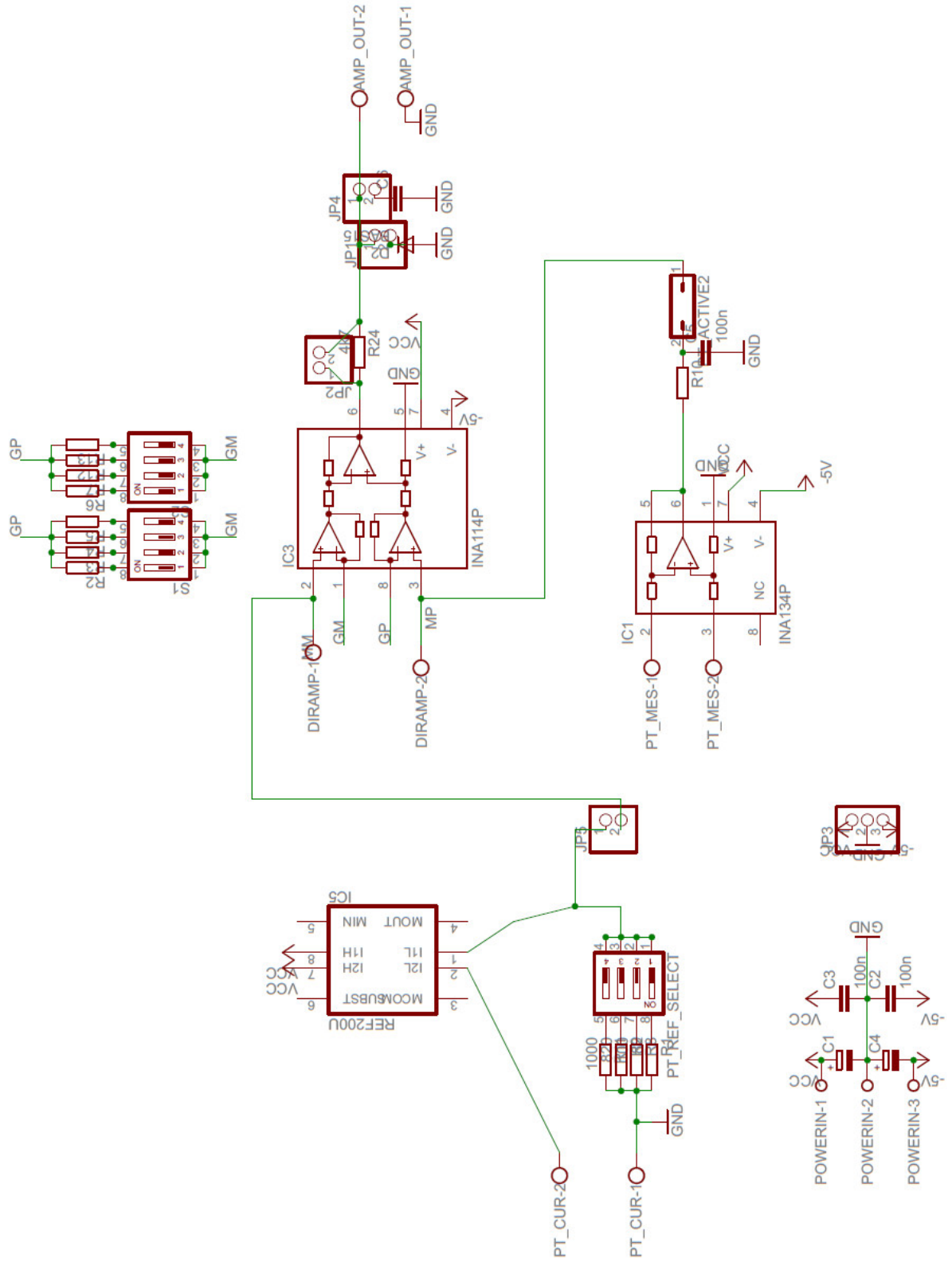


(a) Amplifier board version2

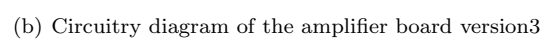


(b) Amplifier board version3

Figure A.1



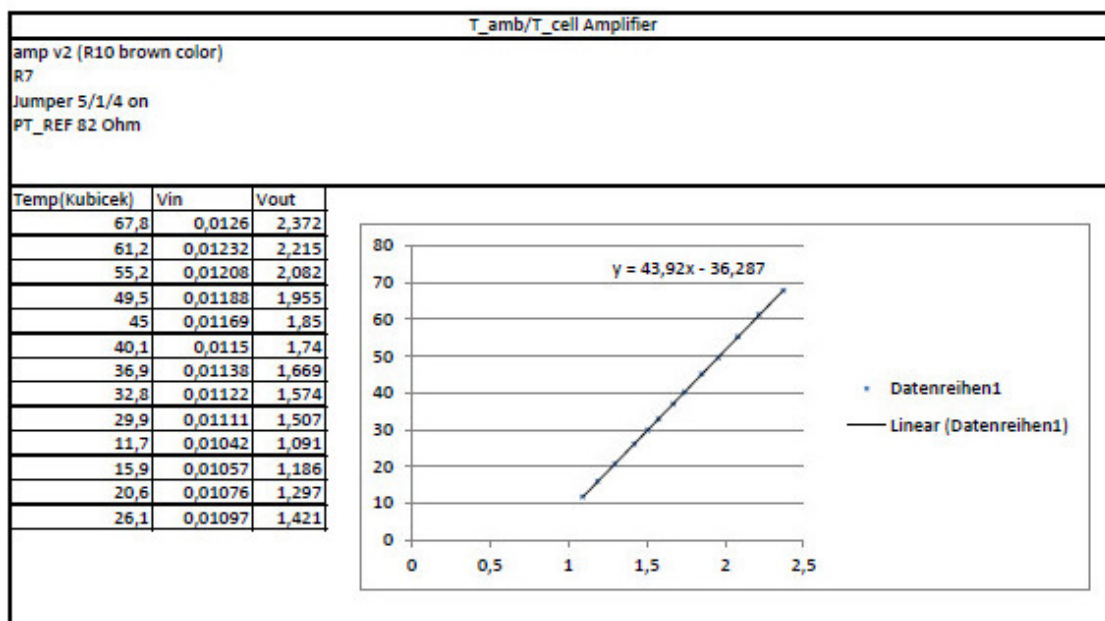
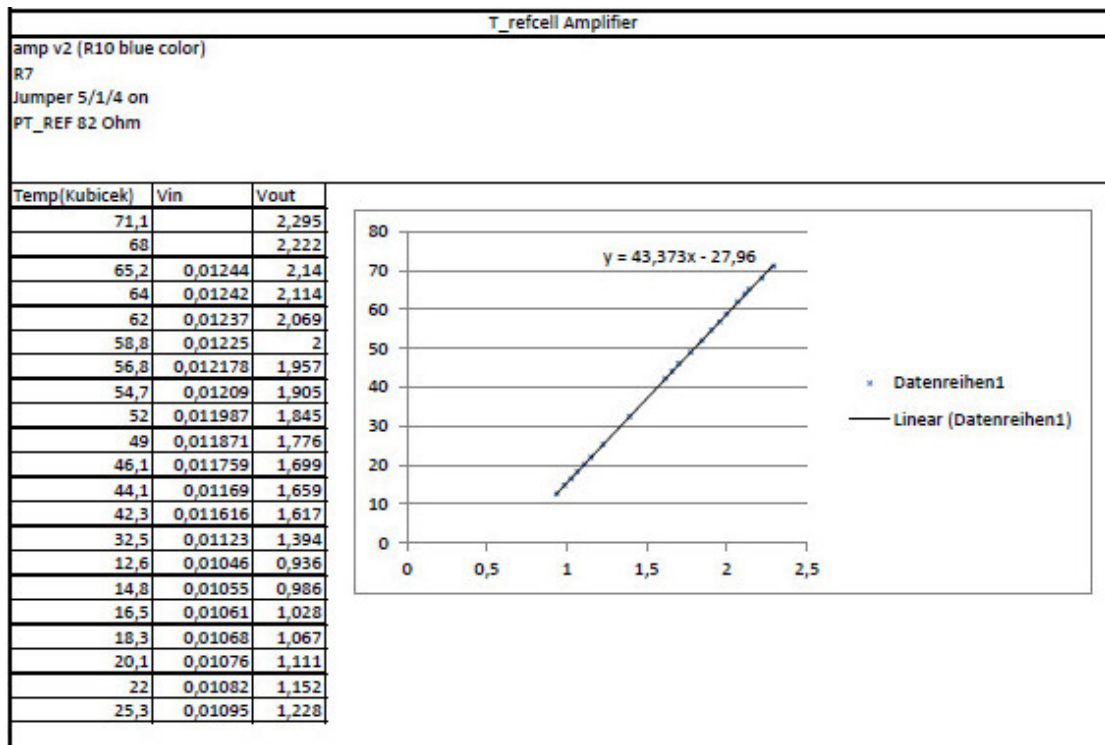
(a) Circuitry diagram of the amplifier board version2

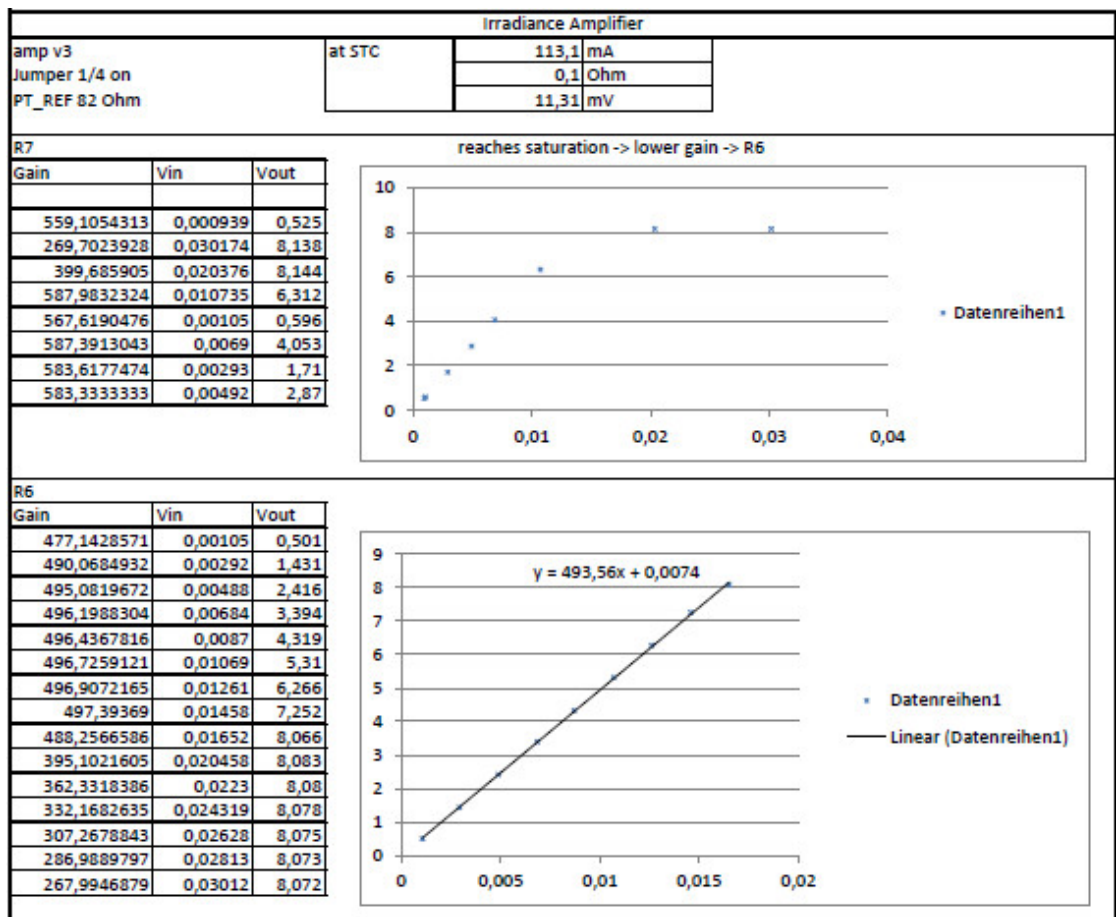


69

A.3 Gauging of Amplifier Boards, Input to Output

Input-output characteristics of the amplifier boards v2 and v3. Temperature sensing was done with a commercial temperature sensor from *checktemp1*, *hanna* (property of Bernhard Kubicek).



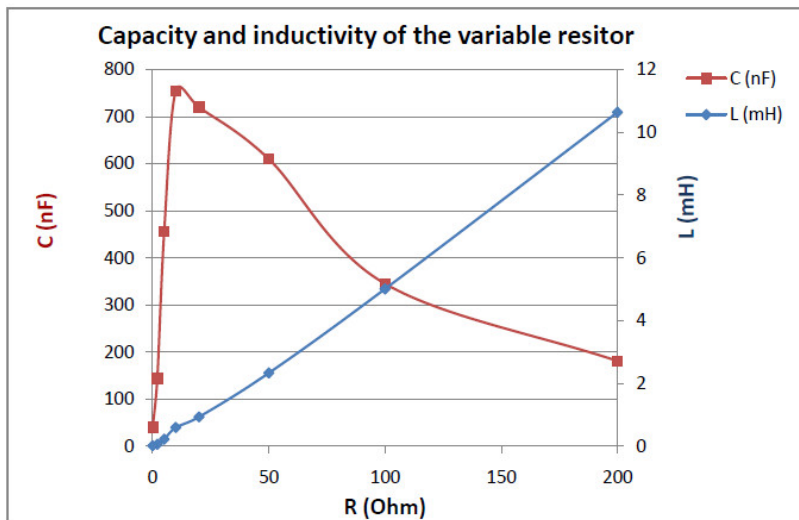


A.4 Variable Resistor , Capacitance and Inductivity

Ruhstrat 5 x 200 Ohm variable resistor

Values per phase

R (Ohm)	L (mH)	C (nF)	ω_0 (Hz)
200	10,64	181	3,63E+03
100	5,01	345	3,83E+03
50	2,33	610	4,22E+03
20	0,931	720	6,15E+03
10	0,594	755	7,52E+03
5	0,218	457	1,60E+04
2	0,055	143	5,68E+04
0,04	0,014	40	2,13E+05



A.5 Simulink Block Diagrams

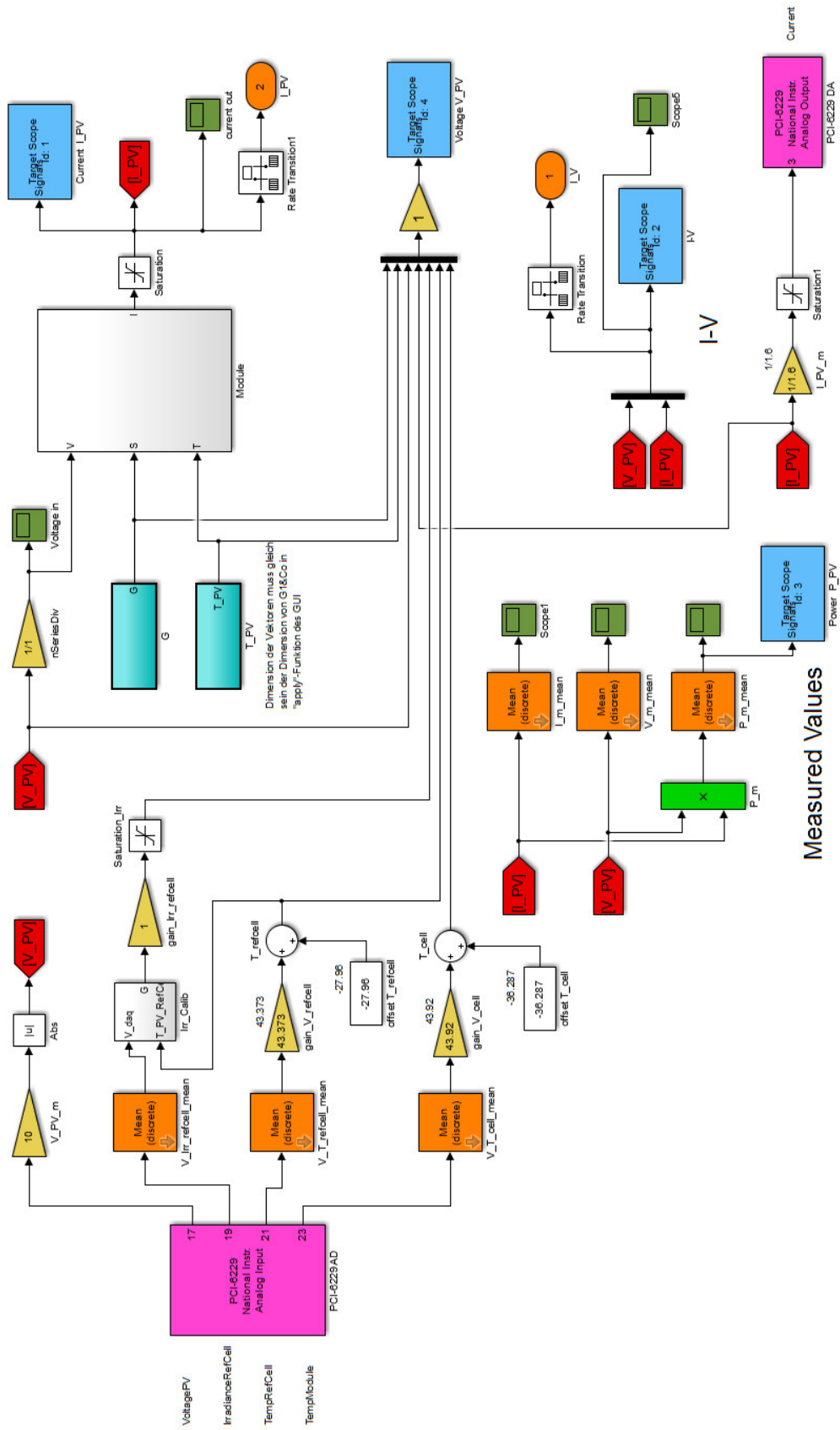
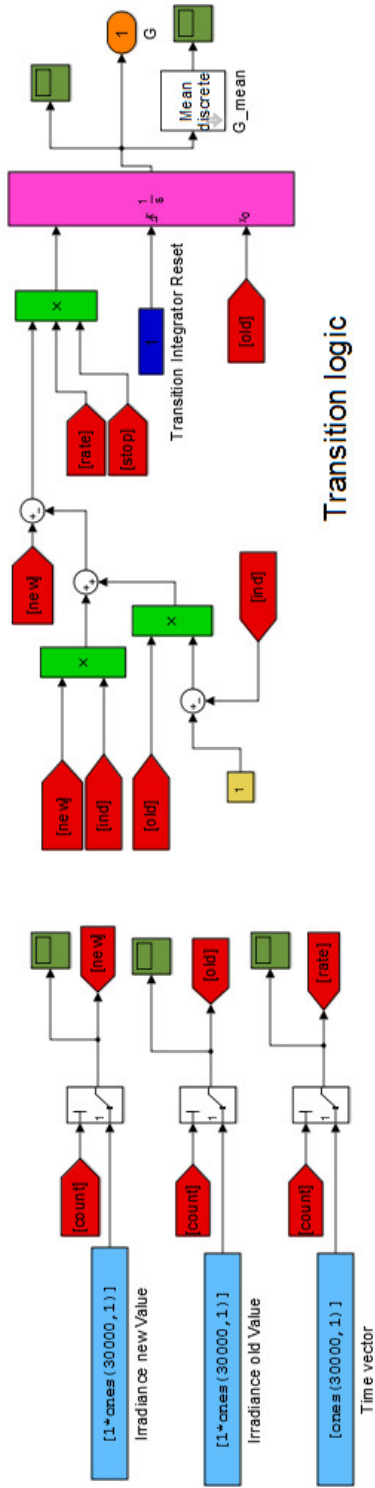
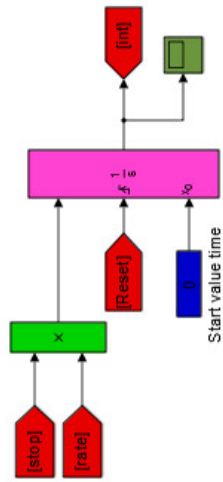


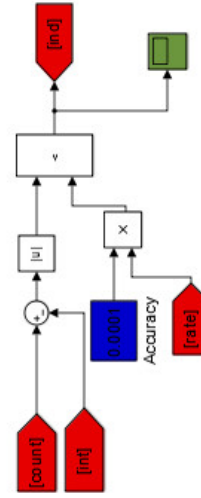
Figure A.3: pv-daq_neu



Transition logic

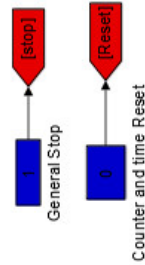


Time Evaluation

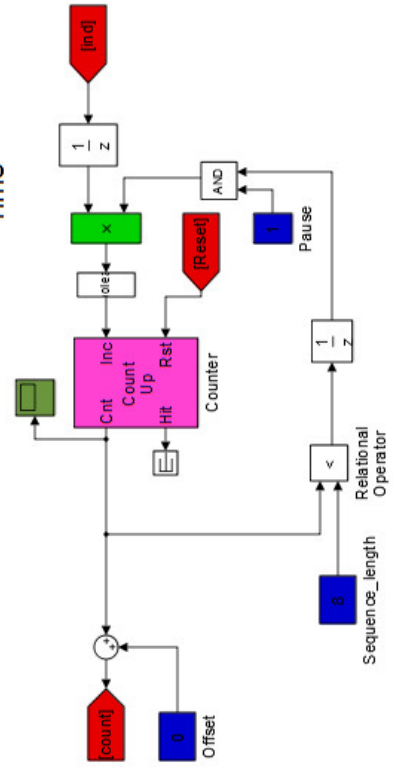


Final value detection logic

Counter and time Reset

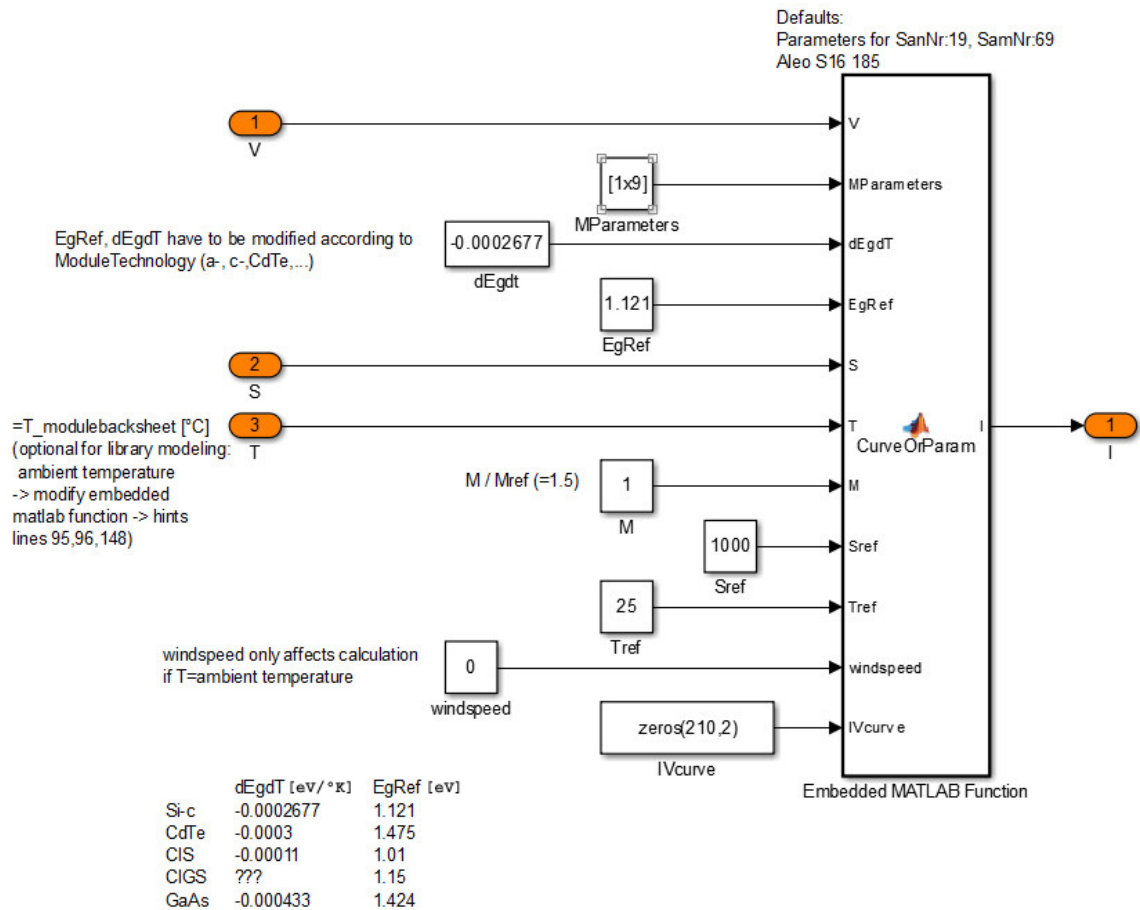


Counter Logic

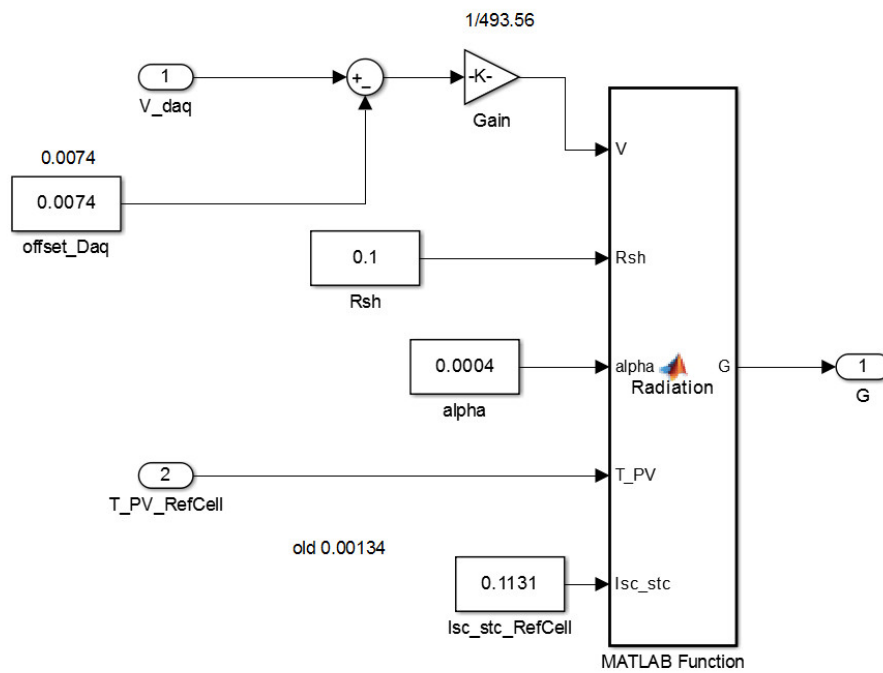


Counter Logic

Figure A.4: pv_daq_neu_G, same as pv_daq_neu_T



(a) pv_daq_neu.Module



(b) pv_daq_neu.Irr_Calib

Figure A.5

A.6 Matlab Code Graphical User Interface

PV Array Simulator GUI_neu code

```
%----- Don not edit this Function (Initialization code by default)-----%
function varargout = GUI_neu(varargin)

% GUI_neu M-file for GUI_neu.fig
%     GUI_neu, by itself, creates a new GUI_neu or raises the existing
%     singleton*.
%
%     H = GUI_neu returns the handle to a new GUI_neu or the handle to the
%     existing singleton*.
%
%     GUI_neu('CALLBACK',hObject,eventData,handles,...) calls the local
%     function named CALLBACK in GUI_neu.M with the given input arguments.
%
%     GUI_neu('Property','Value',...) creates a new GUI_neu or raises the
%     existing singleton*. Starting from the left, property value pairs
%     are applied to the GUI_neu before GUI_neu_OpeningFcn gets called. An
%     unrecognized property name or invalid value makes property
%     application stop. All inputs are passed to GUI_neu_OpeningFcn via
%     varargin.
%
%     *See GUI_neu Options on GUIDE's Tools menu. Choose "GUI_neu allows
%     only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_neu

% Last Modified by GUIDE v2.5 21-Feb-2014 17:14:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @GUI_neu_OpeningFcn, ...
    'gui_OutputFcn',  @GUI_neu_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT --- Executes just before GUI_neu is
% made visible.
```

Opening function of the GUI_neu

```
function GUI_neu_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn. hObject handle to
% figure eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA) varargin
% command line arguments to GUI_neu (see VARARGIN)

%----- Iniatizing the vallues for gui_neu in the opening function-----%

% Set the tolerance of the fminbnd function to be 1e-8 instead of the
% default 1e-4
defaultoptions = optimset('fminbnd');
options = optimset(defaultoptions, 'TolX', 1e-8);

if isempty(find_system('Name','pv_daq_neu')),
    open_system('pv_daq_neu'); %Opens the Pv_daq Simulink model...
end %...with subsystems

%--- logfile indentifier, 0=closed, 1=opened
handles.fopen=0;

%--- Initializing STC(Standard Test conditions) and some general parameters
handles.G_STC=1000;
handles.T_STC=25;
handles.M=1; % AirmassModifier (M=Mx/Mref=1 -> AM1.5, Mref=1.5)
handles.dEgdt=-0.0002677; % Temp.dependency of Bandgap
handles.EgRef=1.121; % energy-bandgap at reference temp

%--- Setting the Irradiance and temperature slider values to initial values
%as displayed on GUI_neu-----%
G=1; T_PV=25; wind=0; % Initialising the irradiance & Temperature Values
set(handles.slider1,'Value',G); % G=1
set(handles.slider2,'Value',T_PV); % T_PV=25
set(handles.wind,'String',num2str(wind));
set(handles.radiobutton1,'Value',0);

%--- Saving the Initial Values of Irradiance & Temperature to re-use in the
%apply callback function--%
save Irradiance G
save temp T_PV
save wind

% Initializing the timer to Update the plot,data & sequence in the GUI_neu
%Updates the XPC Data Plot On GUI for a period of every 200ms.
handles.plot_timer = timer('TimerFcn', {@xPCUpdatePlot, hObject},...
    'Period',0.2, 'ExecutionMode','fixedSpacing', 'BusyMode','drop');
% Updates the XPC Data that is displayed on GUI for every 200ms.
handles.data_timer = timer('TimerFcn', {@xPCUpdateData, hObject},...
    'Period',0.2, 'ExecutionMode','fixedSpacing', 'BusyMode','drop');
% Updates the XPC sequence Data that is displayed on GUI for every 200ms.
handles.seq_timer = timer('TimerFcn', {@xPCUpdate_seq_data, hObject},...
    'Period',0.2, 'ExecutionMode','fixedSpacing', 'BusyMode','drop');
```

```

% Updates the Irr&Temp data that is read from DAQ and displayed in
% edit_text for every 500ms.
handles.live_timer = timer('TimerFcn',{@liveUpdate, hObject},...
    'Period',0.5, 'ExecutionMode','fixedSpacing', 'BusyMode','drop');
% Logs data to .txt file every 1000ms.
handles.log_timer = timer('TimerFcn',{@log, hObject},...
    'Period',1.0, 'ExecutionMode','fixedSpacing', 'BusyMode','drop');

%---Initializing current & power axes on GUI_neu---%
ax1=handles.current_axes;
ax2=handles.power_axes;
ax.Position=get(ax1,'Position');
ax.Parent=get(ax1,'Parent');

set([ax1 ax2], ax);
set(ax1, 'YColor','b',...
    'Color','k',...
    'XTick',[], ...
    'Box','on', ...
    'YAxisLocation','left',...
    'XLimMode','manual', ...
    'YLimMode','manual'); % Tick color blue

set(ax2, 'YColor','r', ...
    'Color','none',...
    'XTick',[], ...
    'Box','on', ...
    'YAxisLocation','right', ...
    'XLimMode','manual', ...
    'YLimMode','manual'); % Tick color red
set(get(ax1,'Ylabel'),'Color','b','String',...
    'CURRENT Amps','FontSize', 14); % Setting Current Labelling
set(get(ax2,'Ylabel'),'Color','r','String',...
    'POWER Watts','FontSize', 14); % Setting Voltage labelling
linkaxes([ax1 ax2],'x'); %Linking the current & power axes

%----Initialising the current Vs voltage and power Vs voltage plots
% and marking of current and power mean on GUI_neu ----%
hl1 =line(NaN,NaN,'Color','w','LineWidth',...
    3,'Parent',ax1); % Current plot
handles.hl1=hl1;
hl2 = line(NaN,NaN,'Color','r','LineWidth',...
    3,'Parent',ax2); % Power plot
handles.hl2=hl2;
% Marking the I_mean w.r.to V_mean on current curve
hl3=line([NaN;NaN],[NaN;NaN],'Color',...
    'r','LineWidth',5,'Parent',ax1);
handles.hl3=hl3;
% Marking the P_mean w.r.to V_mean on power curve
hl4=line([NaN;NaN],[NaN;NaN],'Color',...
    'w','LineWidth',5,'Parent',ax2);
handles.hl4=hl4;
% Marking the I_mpp on current curve
hl5=line(NaN,NaN,'marker','s',...

```

```

        'markersize',9,...
        'markeredgecolor','r',...
        'markerfacecolor','w',...
        'linestyle','none','Parent',ax1);
handles.hl5=hl5;
% Marking the P_mpp on power curve
hl6=line(NaN,NaN,'marker','s',...
        'markersize',9,...
        'markeredgecolor','w',...
        'markerfacecolor','r',...
        'linestyle','none','Parent',ax2);
handles.hl6=hl6;
% Set Grid on
grid(handles.current_axes,'on');

% Reading the Voltage gain value from GUI
V_PV_m = str2double(get(handles.voltage_gain,'String'));
v=V_PV_m*10; % Voltage Gain is multiplied with daq max Voltage.
set(ax1,'xlim',[0 v]); % Set Xlimits to Voltage Gain factor mutlipl. by 10.
% Reading the Current Gain value on GUI
I_PV_m = str2double(get(handles.current_gain,'String'));
set(ax1,'YLim',[0 10/I_PV_m]); % Setting the Current limits on I-V plot
set(ax2,'YLim',[0 (10/I_PV_m)*v]); % Setting Power Limits on P-V plot
xlims = get(ax2,'XLim'); % Get xlims on power axes
ylims = get(ax1,'YLim'); % Get ylims on current axes
ylims2 = get(ax2,'YLim'); % Get xlims on power axes
yinc = (ylims(2)-ylims(1))/8;
yinc2 = (ylims2(2)-ylims2(1))/8;
set(ax2,'XTick',[0:8]*xlims(end)/8); % Setting voltage Ticks
set(ax1,'YTick',ylims(1):yinc:ylims(2)); % Setting Current Ticks
set(ax2,'YTick',ylims2(1):yinc2:ylims2(2)); % Setting Power Ticks
set([ax1 ax2], ax);
grid(handles.current_axes,'on'); % Set Grid on
% Setting the Voltage-labelling for I-V and P-V characteristic curve
xlabel(handles.power_axes,'Voltage V(volts)',...
        'Color','k','FontSize', 14);
title(handles.power_axes,'I-V & P-V Curve',...
        'Color','k','FontSize', 20); % Setting the Title
set(handles.status,'Enable','off');
%update the handles structure
handles.output = hObject;

set(handles.output,'toolbar','figure'); %Creates toolbar for GUI
guidata(hObject, handles);

% UIWAIT makes GUI_neu wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_neu_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT); hObject
% handle to figure eventdata reserved - to be defined in a future version

```



```
% of MATLAB handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Irradiance Slider callback On GUI_neu.

```
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
%        slider

G = get(handles.slider1, 'Value'); % Get the Irradiance slider value from GUI
% Displaying the Irradiance slider value into the GUI_neu edit text
% component
set(handles.irradiance_editText, 'String', num2str(G));

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to slider1 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject, 'BackgroundColor'), ...
    get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', [.9 .9 .9]);
end
```

Temperature slider callback on GUI_neu.

```
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
%        slider

% Get the temperature slider value from GUI
T_PV = get(handles.slider2, 'Value');
% Displaying the Irradiance slider value into the GUI_neu edit text
% component
set(handles.temp_editText, 'String', num2str(T_PV));
```

```

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

% hObject    handle to slider2 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

```

Irradiance Edit Text Callback On GUI_neu

```

function irradiance_editText_Callback(hObject, eventdata, handles)
% hObject    handle to irradiance_editText (see GCBO) eventdata reserved -
% to be defined in a future version of MATLAB handles    structure with
% handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of irradiance_editText as
% text
%         str2double(get(hObject,'String')) returns contents of
%         irradiance_editText as a double

% Get the Irradiance edit text value
G = get(handles.irradiance_editText,'String');

%convert from string to number if possible, otherwise returns empty
G = str2num(G);
%if user inputs something is not a number, or if the input is less than 0
%or greater than 2000, then the slider value defaults to 0
if (isempty(G) || G < 0 || G > 2000)
    set(handles.slider1,'Value',0);
    set(handles.irradiance_editText,'String','0');
else
    set(handles.slider1,'Value',G);
end

% --- Executes during object creation, after setting all properties.
function irradiance_editText_CreateFcn(hObject, eventdata, handles)

% hObject    handle to irradiance_editText (see GCBO) eventdata reserved -
% to be defined in a future version of MATLAB handles    empty - handles
% not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Temperature Edit Text Callback on GUI_neu

```
function temp_editText_Callback(hObject, eventdata, handles)
% hObject    handle to temp_editText (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of temp_editText as text
%        str2double(get(hObject,'String')) returns contents of
%        temp_editText as a double

% Get the Temperature edit text value
T_PV = get(handles.temp_editText, 'String');

% convert from string to number if possible, otherwise returns empty
T_PV = str2num(T_PV);
% if user inputs something is not a number, or if the input is less than
% -200 or greater than 200, then the slider value defaults to 0
if (isempty(T_PV) || T_PV < -200 || T_PV > 200)
    set(handles.slider2, 'Value', 25);
    set(handles.temp_editText, 'String', '0');
else
    set(handles.slider2, 'Value', T_PV);
end
% --- Executes during object creation, after setting all properties.
function temp_editText_CreateFcn(hObject, eventdata, handles)

% hObject    handle to temp_editText (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'), ...
    get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end
```

Tansient Time Callback on GUI_neu

```
function trans_Time_Callback(hObject, eventdata, handles)
% hObject    handle to trans_Time (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of trans_Time as text
%        str2double(get(hObject,'String')) returns contents of trans_Time
%        as a double

% Get the transient time value from trans_time edit text on GUI
T= str2num(get(hObject, 'String'));
% checks to see if input is empty. if so, default trans_time to 0.1
```

```

% if the tras_time is less than 0.01 or greater than 100000 set it to 0.1
if (isempty(T)|| T < 0.01 || T >100000)
    set(hObject,'String','0.1')
end
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function trans_Time_CreateFcn(hObject, eventdata, handles)

% hObject    handle to trans_Time (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Power at STC Callback on GUI_neu

```

function power_Callback(hObject, eventdata, handles)
% hObject    handle to power (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of power as text
%        str2double(get(hObject,'String')) returns contents of power as a
%        double

% Get the power at mpp value from P_MPP_STC edit text on GUI
P_MPP_STC= str2num(get(hObject,'String'));
% checks to see if input is empty. if so, default input1_editText to zero
% set P_MPP_STC to zero if the mpp power at stc edit text is empty
if (isempty(P_MPP_STC))
    set(hObject,'String','0')
end

% --- Executes during object creation, after setting all properties.
function power_CreateFcn(hObject, eventdata, handles)

% hObject    handle to power (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    empty - handles not created
% until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Voltage at STC callback On GUI_neu

```
function voltage_Callback(hObject, eventdata, handles)
% hObject    handle to voltage (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage as text
%        str2double(get(hObject,'String')) returns contents of voltage as a
%        double

% Get the voltage at mpp value from V_MPP_STC edit text on GUI
V_MPP_STC= str2num(get(hObject,'String'));

% checks to see if input is empty. if so, default input1_editText to zero
% set V_MPP_STC to zero if the mpp voltage at stc edit text is empty
if (isempty(V_MPP_STC))
    set(hObject,'String','0')
end
% --- Executes during object creation, after setting all properties.
function voltage_CreateFcn(hObject, eventdata, handles)

% hObject    handle to voltage (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Voltage Gain Callback on GUI_neu

```
function voltage_gain_Callback(hObject, eventdata, handles)
% hObject    handle to voltage_gain (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voltage_gain as text
%        str2double(get(hObject,'String')) returns contents of voltage_gain
%        as a double

% Get the voltage gain value from V_PV_m edit text on GUI
V_PV_m= str2num(get(hObject,'String'));

% checks to see if input is empty. if so, default input1_editText to zero
% set V_PV_m to zero if the voltage gain edit text is empty
if (isempty(V_PV_m))
    set(hObject,'String','0')
end
```

```

% --- Executes during object creation, after setting all properties.
function voltage_gain_CreateFcn(hObject, eventdata, handles)

% hObject    handle to voltage_gain (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles     empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Current gain Callback on GUI_neu

```

function current_gain_Callback(hObject, eventdata, handles)
% hObject    handle to current_gain (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles     structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of current_gain as text
%       str2double(get(hObject,'String')) returns contents of current_gain
%       as a double

% Get the Current gain value from V_PV_m edit text on GUI
I_PV_m= str2num(get(hObject,'String'));

% checks to see if input is empty. if so, default input1_editText to zero
% set I_PV_m to zero if the current gain edit text is empty
if (isempty(I_PV_m))
    set(hObject,'String','0')
end

% --- Executes during object creation, after setting all properties.
function current_gain_CreateFcn(hObject, eventdata, handles)

% hObject    handle to current_gain (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles     empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Status Message Callback On GUI_neu

```

function status_Callback(hObject, eventdata, handles)
% hObject    handle to status (see GCBO) eventdata reserved - to be

```



```

% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of status as text
%         str2double(get(hObject,'String')) returns contents of status as a
%         double

% --- Executes during object creation, after setting all properties.
function status_CreateFcn(hObject, eventdata, handles)

% hObject    handle to status (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Start Button Callback On GUI_neu

--- Executes on button press in start.

```

function start_Callback(hObject, eventdata, handles)

% hObject    handle to start (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)
handles.tg = xpc;           % Create xPC target object
start(handles.tg);          % command to start the xpc target
% Starts data timer which updates data that is displayed on GUI
start(handles.data_timer);
% Starts plot timer which updates Plot that is displayed on GUI
start(handles.plot_timer);
set(handles.stop,'Enable','on'); % Enable the Stop push button on GUI
%---- These Push or edit call backs are disabled on GUI_neu once the xpc
%target starts---%
set(handles.start,'Enable','off'); % Disable Start push button on GUI
set(handles.download,'Enable','off'); % Disable Download Pushbutton on GUI
set(handles.load,'Enable','off'); % Disable Load Sequence Button on GUI
set(handles.execute,'Enable','off'); % Disable Execute Sequence Button on GUI
set(handles.voltage_gain,'Enable','off'); % Disable Voltage gain edit on GUI
set(handles.current_gain,'Enable','off'); % Disable current gain edit on GUI
set(handles.LoadModule,'Enable','off'); % Disable Load Module Button on GUI

```

Stop Button Callback On GUI_neu

--- Executes on button press in stop.

```

function stop_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to stop (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)

handles=guidata(hObject);
handles.tg=xpc;
stop(handles.tg) % Stops the xpc target running
stop(handles.data_timer); % Stops data timer
stop(handles.plot_timer); % Stops plot timer
stop(handles.seq_timer); % Stops sequence timer
stop(handles.live_timer);
stop(handles.log_timer);
if handles.fopen==1
    fclose(handles.fid);
    handles.fopen=0;
end
set(handles.stop, 'Enable', 'on'); % Enable Stop push button on GUI
set(handles.start, 'Enable', 'on'); % Enable Start push button on GUI
set(handles.apply, 'Enable', 'on'); % Enable Apply push button on GUI
% Enable the Download Application Pushbutton on GUI
set(handles.download, 'Enable', 'on');
set(handles.load, 'Enable', 'on'); % Enable Load Sequence Button on GUI
set(handles.execute, 'Enable', 'on'); % Enable Execute Sequence Button on GUI
set(handles.status, 'Enable', 'off'); % Disable Status text on GUI
set(handles.voltage_gain, 'Enable', 'on'); % Enable Voltage gain edit on GUI
set(handles.current_gain, 'Enable', 'on'); % Enable current gain edit on GUI
set(handles.trans_Time, 'Enable', 'on');
set(handles.irradiance_editText, 'Enable', 'on');
set(handles.temp_editText, 'Enable', 'on');
set(handles.apply, 'Enable', 'on');
set(handles.status, 'String', ...
    'Sequence Status: Stopped'); % Displays status message of sequence
set(handles.LoadModule, 'Enable', 'on'); % Enable Load Module Button on GUI
set(handles.datalog, 'Enable', 'on');
set(handles.Live, 'Enable', 'on');
set(handles.slider1, 'Enable', 'on');
set(handles.slider2, 'Enable', 'on');
guidata(hObject, handles);
% Save the updated handles structure

```

Download Button Callback On GUI_neu

--- Executes on button press in download.

```

function download_Callback(hObject, eventdata, handles)

% hObject    handle to download (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
set(handles.radiobutton1, 'Value', 0);
rtwbuild('pv_daq_neu') % Downloads simulink pv_daq model on the xPC target
%close_system('pv_daq',0); xpcexplr
handles.tg=xpc;
%set(handles.radiobutton1, 'Value', 0);

```

```

% reads/creates selected module parameters
offset=-1;
SanNr=str2num(get(handles.SanDB, 'String'))+offset;
SamNr=str2num(get(handles.SamDB, 'String'));
SanPar=pvl_sapmmoduledb(SanNr, 'SanModDB');
CECModuleDB=load('CECModuleDatabaseSAM2012.11.30.mat');
SamPar=CECModuleDB.CECModuleDB(SamNr);
ModuleNameSAM=CECModuleDB.ModuleNames(SamNr);
ModuleNameSan=SanPar.name;
Data=[SamPar.a_ref, SamPar.IL_ref, SamPar.IO_ref, SamPar.Rsh_ref, ...
      SamPar.Rs_ref, SamPar.a_wind, SamPar.b_wind, SamPar.delT, ...
      SamPar.AlphaIsc];

% VGain=SanPar.Voc0*(1+0.2)*str2num(get(handles.nSeries, 'String'))/10;

% sets parameters on target
setparam(handles.tg, getparamid(handles.tg, 'Module/MParameters', 'Value'), ...
      Data);
setparam(handles.tg, getparamid(handles.tg, 'Module/dEgdt', 'Value'), ...
      handles.dEgdt);
setparam(handles.tg, getparamid(handles.tg, 'Module/EgRef', 'Value'), ...
      handles.EgRef);
setparam(handles.tg, getparamid(handles.tg, 'Module/M', 'Value'), handles.M);
setparam(handles.tg, getparamid(handles.tg, 'Module/Sref', 'Value'), ...
      handles.G_STC);
setparam(handles.tg, getparamid(handles.tg, 'Module/Tref', 'Value'), ...
      handles.T_STC);
% setparam(handles.tg, getparamid(handles.tg, 'V_PV_m', 'Gain'), VGain);

% displays selected modules&Characteristics
set(handles.ModuleNameSAM, 'String', ModuleNameSAM);
set(handles.ModuleNameSan, 'String', ModuleNameSan);
set(handles.V_oc_STC, 'String', SanPar.Voc0);
set(handles.I_sc_STC, 'String', SanPar.Isc0);
set(handles.P_mpp_STC, 'String', SanPar.Imp0*SanPar.Vmp0);
set(handles.V_mpp_STC, 'String', SanPar.Vmp0);
set(handles.I_mpp_STC, 'String', SanPar.Imp0);
set(handles.voltage_gain, 'String', num2str(getparam(handles.tg, ...
      getparamid(handles.tg, 'V_PV_m', 'Gain'))));

handles.SanPar=SanPar;
handles.SamPar=SamPar;
nSeries=str2double(get(handles.nSeries, 'String'));
handles.nSer=nSeries;
set(handles.radiobutton1, 'Value', 1);
handles.IVcurve_ind=0; %IVcurve_indicator, 1=on

% set(handles.tg, 'ShowSignals', 'off')
% set(handles.tg, 'ShowParameters', 'off')
% getsignal(handles.tg, getsignalid(handles.tg, 'Module/IrrIn_mean/Gain'))

guidata(hObject, handles);

```

Update Data That Are Displayed On GUI_neu For Every 200 ms

```
function xPCUpdateData(obj,event, hObject)

handles = guidata(hObject);
handles.tg=xpc;
%---Reading the signal index from target using name of the block and
%reading back the Signal value of that block from target& rounding of the
%values to two digits after decimal point---%
% V_OC=num2str(round(getsignal(handles.tg,...
%     getsignalid(handles.tg,... 'PV_Generator Model/V_OC'))*100)/100);
%     % Open Circuit Voltage
% I_SC=num2str(round(getsignal(handles.tg,...
%     getsignalid(handles.tg,... 'PV_Generator Model/I_SC'))*100)/100);
%     % Short Circuit current
V_m_mean=num2str(round(getsignal(handles.tg,...
%     getsignalid(handles.tg,...
%     'V_m_mean/Product'))*100)/100); % Measured mean Voltage
I_m_mean=num2str(round(getsignal(handles.tg,...
%     getsignalid(handles.tg,...
%     'I_m_mean/Product'))*100)/100); % Measured mean Current
P_m_mean=num2str(round(getsignal(handles.tg,...
%     getsignalid(handles.tg,...
%     'P_m_mean/Product'))*100)/100); % Measured mean Power

% assignin('base', 'I_m_mean', I_m_mean);

%---Displaying the read back values from xpc on GUI_neu---
% set(handles.V_oc,'String',V_OC); set(handles.I_sc,'String',I_SC);
set(handles.voltage_mean_staticText,'String',V_m_mean);
set(handles.current_mean_staticText,'String',I_m_mean);
set(handles.power_mean_staticText,'String',P_m_mean);

guidata(hObject,handles);
```

Update The Plot On GUI_neu For Every 200 ms

```
function xPCUpdatePlot(obj, eventdata, hObject)

handles = guidata(hObject);
handles.tg=xpc;
%----- Reading the index values in the form of an array (named as constant)
%with block names such as Irradiance mean G_mean,
% Temperature mean T_PV_mean,Power mean P_m_mean,Voltage mean V_m_mean
% Current mean I_m_mean from target
Constant=getsignal(handles.tg,[getsignalid(handles.tg,'G/G_mean/Product')...
%     getsignalid(handles.tg,'T_PV/T_PV_mean/Product')...
%     getsignalid(handles.tg,'P_m_mean/Product')...
%     getsignalid(handles.tg,'V_m_mean/Product')...
%     getsignalid(handles.tg,'I_m_mean/Product')]);
G=Constant(1);T_PV=Constant(2);P_m_mean=Constant(3);
```

```

V_m_mean=Constant(4);I_m_mean=Constant(5);

wind=getparam(handles.tg,getparamid(handles.tg,'Module/windspeed','Value'));
% Reading the voltage gain value displayed on GUI
V_PV_m = str2double(get(handles.voltage_gain,'String'));
% Voltage range of the PV; 10->InputVoltageRange from DAQ
V_PV=0:0.1:(10*V_PV_m);
nSeries=str2double(get(handles.nSeries,'String'));

if handles.IVcurve_ind==1
    IVcurve=handles.IVcurve;
    Rs=IVcurve(2,2);
    k=IVcurve(3,2);
    alpha=IVcurve(4,2);
    beta=IVcurve(5,2);
    G1=IVcurve(6,2);
    T1=IVcurve(7,2);
    Isc=IVcurve(8,2); %shortcut current at reference conditions
    l=IVcurve(9,2);

    % original curve
    x1=IVcurve(10:10+l-1,1); % Voltage DataSet (0 -> Voc)
    y1=IVcurve(10:10+l-1,2); % Current DataSet (Isc -> 0)

    % corrected curve according to 1) & 2) in Reference [2] (IEC 60891)
    y2=y1+Isc*(G/G1-1)+alpha*(T_PV-T1);
    x2=x1-Rs*(y2-y1)-k*y2*(T_PV-T1)+beta*(T_PV-T1);

    % interpolations to zero crossing, original curve use linear regression
    % over 4 value pairs for extrapolation to zerocrossing
    regsize=4; % regression size (number of pairs for linear regression)
    epsilon=realmin; % smallest possible floating point number...
    % (double precision)

    if x2(1)>0
        %y2(1)=(y2(2)-y2(1))/(x2(2)-x2(1))*(-x2(1))+y2(1);
        xi=x2(1:regsize); yi=y2(1:regsize);
        xi_avg=sum(xi)/regsize; yi_avg=sum(yi)/regsize;
        denom1=sum((xi-xi_avg).*xi);
        if denom1==0
            denom1=epsilon;
        end
        ai=sum((yi-yi_avg).*xi)/denom1;
        bi=sum(yi-ai*xi)/regsize;
        Isc=bi;
        x2(2:l+1)=x2(1:l);
        y2(2:l+1)=y2(1:l);
        x2(1)=0;
        y2(1)=Isc;
        l=l+1;
    end
    if y2(l)>0
        % x2(l)=(x2(l)-x2(l-1))/(y2(l)-y2(l-1))*y2(l)+x2(l-1);
        xv=x2(l-regsize+1:l); yv=y2(l-regsize+1:l);
        xv_avg=sum(xv)/regsize; yv_avg=sum(yv)/regsize;
    end
end

```

```

        denom2=sum((xv-xv_avg).*xv);
        if denom2==0
            denom2=epsilon;
        end
        av=sum((yv-yv_avg).*xv)/denom2;
        bv=sum(yv-av*xv)/regsize;
        Voc=-bv/av;
        y2(l+1)=0;
        x2(l+1)=Voc;
        if x2(l)==x2(l+1)
            x2(l+1)=x2(l+1)+epsilon; % avoids division through 0...
                                     % at interpolation
        end
    end
end

x2=x2*nSeries;
x2max=max(x2);
z1=find(V_PV<=x2max,1,'last');
f2=fit(x2,y2,'linearinterp');
I_PV=0*V_PV;
I_PV(1:z1)=f2(V_PV(1:z1));
z2=(I_PV<0);
I_PV(z2)=0;
z3=find(I_PV==0,1,'first');
P_PV = I_PV.*V_PV; %Photo voltaic POWER
% Calculating max Power & Voltage from the Plot
[max_power,max_voltage] = max(P_PV);
% Calculating the Accuracy which is the ratio of p_m_mean & Mpp power
MPPT_Accuracy=num2str(round((P_m_mean/max_power)*100*100)/100);
max_current=find(P_PV==max(P_PV(:))); % Calculating max urrent from Plot

%--- Calculating the mpp power & voltage from P_V Curve-----%
% Mpp Power rounding of to 2digits after decimal point
P_mpp=num2str(round(max_power*100)/100);
% Mpp Voltage rounding of to 2digits after decimal point
V_mpp=num2str(round(V_PV(max_voltage)*100)/100);
% Mpp Current rounding of to 2 digits after decimal point
I_mpp=num2str(round(I_PV(max_current)*100)/100);
set(handles.V_oc,'String',num2str(round(V_PV(z3)*1000)/1000));
set(handles.I_sc,'String',num2str(f2(0)));

else

    IV=fcnGUI(handles.SanPar,handles.SamPar,G,T_PV,wind,handles.G_STC,...
        handles.M,handles.EgRef,handles.dEgdt,V_PV);
    IV.V=IV.V*nSeries;
    IV.Vmp=IV.Vmp*nSeries;
    IV.Voc=IV.Voc*nSeries;
    IV.Pmp=IV.Pmp*nSeries;

    I_PV=IV.I;
    V_PV=IV.V;
    P_PV = I_PV.*V_PV; %Photo voltaic POWER
    MPPT_Accuracy=num2str(round((P_m_mean/IV.Pmp)*100*100)/100);

```



```

max_current=find(P_PV==max(P_PV(:)));% Calculating max current from Plot

% Mpp power rounding of to 2digits after decimal point
P_mpp=num2str(IV.Pmp); %num2str(round(max_power*100)/100);
% Mpp Voltage rounding of to 2digits after decimal point
V_mpp=num2str(IV.Vmp); %num2str(round(V_PV(max_voltage)*100)/100);
% Mpp Current rounding of to 2digits after decimal point
I_mpp=num2str(IV.Imp); %num2str(round(I_PV(max_current)*100)/100);

set(handles.V_oc, 'String', num2str(IV.Voc));
set(handles.I_sc, 'String', num2str(IV.Isc));
end
Vgain=str2double(get(handles.voltage_gain, 'String'));
barlength=40*Vgain/100;
set(handles.h11, 'XData', V_PV, 'YData', I_PV);
set(handles.h12, 'XData', V_PV, 'YData', P_PV);
% Set the marker on the I_V curve at I_mean with respective to voltage mean
set(handles.h13, 'XData', [V_m_mean+barlength;V_m_mean-barlength],...
    'YData', [I_m_mean;I_m_mean])
% Set the marker on the P_V curve at P_mean with respective to voltage mean
set(handles.h14, 'XData', [V_m_mean+barlength;V_m_mean-barlength],...
    'YData', [P_m_mean;P_m_mean]);
% Set the marker on I_V Curve at mpp_current of the curve
set(handles.h15, 'XData', V_PV(max_current),...
    'YData', I_PV(max_current));
% Set the marker on P_V Curve at mpp_power of the curve
set(handles.h16, 'XData', V_PV(max_current),...
    'YData', P_PV(max_current));
set(handles.P_mpp, 'String', ...
    P_mpp); % Displaying the mpp_power calculated from curve on GUI
set(handles.V_mpp, 'String', ...
    V_mpp); % Displaying the mpp_voltage calculated from curve on GUI
set(handles.I_mpp, 'String', ...
    I_mpp); % Displaying the mpp_current calculated from curve on GUI
set(handles.mppt_accuracy_staticText, 'String', ...
    MPPT_Accuracy); % Displaying the Accuracy calculated on GUI

guidata(hObject, handles);

```

Apply Button Callback On GUI_neu

--- Executes on button press in apply.

```

function apply_Callback(hObject, eventdata, handles)

% hObject    handle to apply (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)
handles = guidata(hObject);
handles.tg=xpc;

if strcmp(get(handles.voltage_gain, 'enable'), 'on');    % rescale plot axes
    ax1=handles.current_axes;

```

```

ax2=handles.power_axes;
ax.Position=get(ax1, 'Position');
ax.Parent=get(ax1, 'Parent');
% Reading the Voltage gain value from GUI
V_PV_m = str2double(get(handles.voltage_gain, 'String'));
v=V_PV_m*10; % Voltage Gain is multiplied with daq max Voltage.
set(ax1, 'xlim', [0 v]); % Set Xlimits to Voltage Gain factor mutlipl. by 10.
% Reading the Current Gain value on GUI
I_PV_m = str2double(get(handles.current_gain, 'String'));
set(ax1, 'YLim', [0 10/I_PV_m]); % Setting the Current limits on I-V plot
set(ax2, 'YLim', [0 (10/I_PV_m)*v]); % Setting Power Limits on P-V plot
xlims = get(ax2, 'XLim'); % Get xlims on power axes
ylims = get(ax1, 'YLim'); % Get ylims on current axes
ylims2 = get(ax2, 'YLim'); % Get xlims on power axes
yinc = (ylims2(2)-ylims(1))/8;
yinc2 = (ylims2(2)-ylims2(1))/8;
set(ax2, 'XTick', [0:8]*xlims(end)/8); % Setting voltage Ticks
set(ax1, 'YTick', ylims(1):yinc:ylims(2)); % Setting Current Ticks
set(ax2, 'YTick', ylims2(1):yinc2:ylims2(2)); % Setting Power Ticks
set([ax1 ax2], ax);
grid(handles.current_axes, 'on'); % Set Grid on
end

% Reading Trans_time value on GUI
T= str2double(get(handles.trans_Time, 'String'));
wind=str2double(get(handles.wind, 'String'));
setparam(handles.tg, getparamid(handles.tg, 'Module/windspeed', 'Value'), wind);

% -----Gets the Transient time from GUI_neu for Temperature &
% Irradiance-----%
load Irradiance % Loading the Irradiance Value that is saved as G
load temp % loading the Temperature Value that is saved as T_PV

%---Initializing the size of vector values for Irradiance ,Temperature and
%transition time---%
G1=zeros(30000,1);T_PV1=zeros(30000,1);
G2=zeros(30000,1);T_PV2=zeros(30000,1);
Time=ones(30000,1);
%---Defining the Vector elements for Irradiance,Temperature &
%TransientTime-----%
G1_new=[get(handles.slider1, 'Value');get(handles.slider1, 'Value')];
G1(1:length(G1_new),1)=G1_new; % Irradince new value
G2_old=[G;get(handles.slider1, 'Value')]; % G is the saved irradiance value
G2(1:length(G2_old),1)=G2_old; % Irradiance old value
Time1=[1/T;1/T]; % T is Trans_time
Time(1:length(Time1),1)=Time1; % Transition time for both G & T_PV

T_PV1_new=[get(handles.slider2, 'Value');get(handles.slider2, 'Value')];
T_PV1(1:length(T_PV1_new),1)=T_PV1_new; % Temperature new value
T_PV2_old=[T_PV;get(handles.slider2, 'Value')]; % T_PV is saved temp. value
T_PV2(1:length(T_PV2_old),1)=T_PV2_old; % Temperature old value

%---Setting sequence length block parameter value to one for both
%Irradiance & Temp --%

```

```

setparam(handles.tg,getparamid(handles.tg,'G/Sequence_length','Value'),...
    (length(Time1)));setparam(handles.tg,...
    getparamid(handles.tg,'T_PV/Sequence_length','Value'),(length(Time1)));

% ---Setting the General stop block parameter to zero in Irradiance &
% Temperature---%
G_General_Stop=getparamid(handles.tg,'G/General Stop','Value');
T_General_Stop=getparamid(handles.tg,'T_PV/General Stop','Value');
setparam(handles.tg,G_General_Stop,...
    0);setparam(handles.tg,T_General_Stop,0);

% ----Reading the Irradiance subsystem Simulink block Parameters index
% values from target----%
setparam(handles.tg,getparamid(handles.tg,...
    'G/Irradiance new Value','Value'),G1); % Irradiance New Value
setparam(handles.tg,getparamid(handles.tg,...
    'G/Irradiance old Value','Value'),G2); % irradiance Old value
setparam(handles.tg,getparamid(handles.tg,...
    'G/Time vector','Value'),Time); % Irradiance Trans_Time
G_Counter_reset=getparamid(handles.tg,...
    'G/Counter and time Reset','Value'); % Counter and Time Integrator Reset
G_Integrator_reset=getparamid(handles.tg,...
    'G/Transition Integrator Reset','Value'); % Transition Integrator Reset
%----In the Irradiance simulink, reset the Counter and Integrator and then
%setting general stop to one----%
setparam(handles.tg,[G_Counter_reset G_Counter_reset...
    G_Integrator_reset G_Integrator_reset G_General_Stop ],{1,0,1,0,1});

% ----Reading the Temperature Simulink block Parameters index values from
% target----%
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Temperature new Value','Value'),T_PV1);% Temperature New Value
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Temperature old Value','Value'),T_PV2);% Temperature Old Value
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Time vector','Value'),Time); % Temperature Trans_Time
T_Counter_reset=getparamid(handles.tg,...
    'T_PV/Counter and time Reset','Value');% Counter and Time Integrator Reset
T_Integrator_reset=getparamid(handles.tg,...
    'T_PV/Transition Integrator Reset','Value');% Transition Integrator Reset
%----In the Temperature simulink, reset the Counter and Integrator and then
%setting general stop to one----%
setparam(handles.tg,[T_Counter_reset T_Counter_reset...
    T_Integrator_reset T_Integrator_reset T_General_Stop],{1,0,1,0,1});

%---Reading the values on GUI_neu to change it on target---%

V_PV_m = str2double(get(handles.voltage_gain,'String')); % Voltage Gain
I_PV_m = str2double(get(handles.current_gain,'String')); % Current Gain

%----Reading the parameter index values to change in target-----%
v_pv_m=getparamid(handles.tg,...
    'V_PV_m','Gain'); % Voltage Gain
i_pv_m=getparamid(handles.tg,...

```

```

        'I_PV_m','Gain'); % Current Gain

% setting the parameter values on target
setparam(handles.tg,[v_pv_m,i_pv_m],{V_PV_m,I_PV_m});

% Saving Irradiance & Temperature Slider Values on GUI_neu to store it
%& TO re-use in the irradiance & Temp old values as G & T_PV.
G=get(handles.slider1,'Value');
save Irradiance G
T_PV =get(handles.slider2,'Value');
save temp T_PV

guidata(hObject,handles);

```

Load Sequence Button Callback On GUI_neu

--- Executes on button press in load.

```

function load_Callback(hObject, eventdata, handles)

% hObject    handle to load (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)
%
%x = uigetfile('*.mat', 'Choose the GUI_neu file to load');
handles = guidata(hObject);
handles.tg=xpc;
figure;
load Seq % Load the sequence file (mat) from matlab to run on GUI
seq.Time;seq.Temp;seq.G;seq.Weight;seq.User;
% Saving the sequence values to handles structure to re-use in another
% callback
handles.W=seq.Weight;
handles.User=seq.User;
handles.T=seq.Time;
handles.Temp=seq.Temp;
handles.G=seq.G;
subplot(2,1,1)
% Plot Irradiance Versus time for the given sequence
handles.h7=plot(handles.T,handles.G);
handles.h8=line([NaN;NaN],[NaN;NaN],...
    'Color','w','LineWidth',5);
title('IRRADIANCE SEQUENCE','Color',...
    'b','FontSize',15);
ylabel('Irradiance (W/m²)', 'Color','b');
xlabel('Time (s)', 'Color','b');
subplot(2,1,2)
% Plot Temperature versus time for the given sequence
handles.h9=plot(handles.T,handles.Temp);
handles.h10=line([NaN;NaN],[NaN;NaN], 'Color','w','LineWidth',5);
title('TEMPERATURE SEQUENCE', 'Color','r','FontSize',15)
ylabel('Temperature (°C)', 'Color','r');
xlabel('Time (s)', 'Color','r');

```

```
guidata(hObject, handles);
```

Execute Sequence Button Callback On GUI_neu

--- Executes on button press in execute.

```
function execute_Callback(hObject, eventdata, handles)

% hObject    handle to execute (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
handles = guidata(hObject);
handles.tg=xpc;
load Seq
seq.Time;seq.Temp;seq.G;seq.Weight;seq.User;
% Saving the sequence values to handles structure to re-use in another
% callback
handles.W=seq.Weight;
handles.User=seq.User;
handles.T=seq.Time;
handles.Temp=seq.Temp;
handles.G=seq.G;
%starts plot timer
seq.Weight=handles.W;
seq.User=handles.User;
T=handles.T;
T_PV=handles.Temp;
G=handles.G;
% Initialising the size of Vectors of Irradiance & Temperature new & old
% values and Transtime
G1=zeros(30000,1);T_PV1=zeros(30000,1);
G2=zeros(30000,1);T_PV2=zeros(30000,1);
Time=ones(30000,1);
G1_new=G;
G1(1:length(G1_new)-1,1)=G1_new(2:length(G1_new),1);%Irradiance New value
G1(length(G1_new),1)=G1(length(G1_new)-1,1);
G2_old=G;
G2(1:length(G2_old),1)=G2_old;%Irradiance Old value
i=1;
while ((i+1)<length(handles.T)+1)

    Time1(i)=1/(T(i+1)-T(i));
    i=i+1;
end;

Time1=Time1';
Time(1:length(Time1),1)=Time1;%Trans_Time for both Temperature & Irradiance

T_PV1_new=T_PV;
%Temperature new value
T_PV1(1:length(T_PV1_new)-1,1)=T_PV1_new(2:length(T_PV1_new),1);
```

```

T_PV1(length(T_PV1_new),1)=T_PV1(length(T_PV1_new)-1,1);
T_PV2_old=T_PV;
T_PV2(1:length(T_PV2_old),1)=T_PV2_old;
%---Setting sequence length block parameter value to length of the time
%vector plus one for both Irradiance & Temp ---%
setparam(handles.tg,getparamid(handles.tg,'G/Sequence_length','Value'),...
    (length(Time1))+1);setparam(handles.tg,...
    getparamid(handles.tg,'T_PV/Sequence_length','Value'),(length(Time1))+1);

% ---Setting the General stop block parameter to zero in Irradiance &
% Temperature simulink blocks---%
G_General_Stop=getparamid(handles.tg,'G/General Stop','Value');
T_General_Stop=getparamid(handles.tg,'T_PV/General Stop','Value');
setparam(handles.tg,G_General_Stop,...
    0);setparam(handles.tg,T_General_Stop,0);

% ----Reading the Irradiance subsystem Simulink block Parameters index
% values from target----%
setparam(handles.tg,getparamid(handles.tg,...
    'G/Irradiance new Value','Value'),G1);% Irradiance New Value
setparam(handles.tg,getparamid(handles.tg,...
    'G/Irradiance old Value','Value'),G2);% irradiance Old value
setparam(handles.tg,getparamid(handles.tg,...
    'G/Time vector','Value'),Time); %Irradiance Trans_Time
G_Counter_reset=getparamid(handles.tg,...
    'G/Counter and time Reset','Value');
G_Integrator_reset=getparamid(handles.tg,...
    'G/Transition Integrator Reset','Value');
%---In the irradiance simulink, Reset the Counter and Integrator and then
%setting general stop to one----%
setparam(handles.tg,[G_Counter_reset G_Counter_reset...
    G_Integrator_reset G_Integrator_reset G_General_Stop ],{1,0,1,0,1});

% ----Reading the Temperature Simulink block Parameters index values from
% target----%
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Temperature new Value','Value'),T_PV1);%Temperature New Value
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Temperature old Value','Value'),T_PV2);%Temperature Old Value
setparam(handles.tg,getparamid(handles.tg,...
    'T_PV/Time vector','Value'),Time); %Temperature Trans_Time
T_Counter_reset=getparamid(handles.tg,...
    'T_PV/Counter and time Reset','Value');
T_Integrator_reset=getparamid(handles.tg,...
    'T_PV/Transition Integrator Reset','Value');
%---In the Temperature simulink, reset the Counter and Integrator and then
% setting general stop to one----%
setparam(handles.tg,[T_Counter_reset T_Counter_reset...
    T_Integrator_reset T_Integrator_reset T_General_Stop],{1,0,1,0,1});

%----- Starting the sequence to run on GUI_neu-----%
handles.tg=xpc;
start(handles.tg); % Starts the xpc target
start(handles.data_timer);% Starts update data timer that is displayed on GUI

```



```

start(handles.plot_timer);% Starts update Plot timer on GUI
start(handles.seq_timer); % Starts update seq data timer that's displayed on GUI
set(handles.stop, 'Enable', 'on'); % Enable the Stop push button on GUI
set(handles.start, 'Enable', 'off'); % Disable the Start push button on GUI
set(handles.apply, 'Enable', 'off'); % Disable the Apply push button on GUI
set(handles.power, 'Enable', 'off'); % Disable MPP Power edit at STC on GUI
set(handles.voltage, 'Enable', 'off'); % Disable MPP Voltage edit at STC on GUI
set(handles.download, 'Enable', 'off'); % Disable Download Pushbutton on GUI
set(handles.load, 'Enable', 'off'); % Disable Load Sequence Button on GUI
set(handles.status, 'Enable', 'off'); % Disable Status text on GUI
set(handles.execute, 'Enable', 'off'); % Disable Execute Sequence Button on GUI
set(handles.voltage_gain, 'Enable', 'off'); % Disable Voltage gain edit on GUI
set(handles.current_gain, 'Enable', 'off'); % Disable current gain edit on GUI

guidata(hObject, handles);

```

Sequence Data Update that are displayed on GUI_neu

```

function xPCUpdate_seq_data(obj, eventdata, hObject)

handles = guidata(hObject);
set(handles.status, 'String', 'Sequence Status: Running');
seq.Weight=handles.W;
seq.User=handles.User;
%-----Read the Signal index of G_mean & T_mean & those signal values-----%
G_mean=getsignal(handles.tg, getsignalid(handles.tg, 'G/Integrator'));
T_mean=getsignal(handles.tg, getsignalid(handles.tg, 'T_PV/Integrator'));
%---updates the sequence data of the Irradiance & Temperature displayed on
%GUI_neu for every 200 ms---
set(handles.irradiance_editText, 'String', num2str((round(G_mean)*100)/100));
set(handles.temp_editText, 'String', num2str((round(T_mean)*100)/100));
h8=handles.h8;
h10=handles.h10;
%---Read the signal index of Trans_time of G or T & its signal value ---%
T_t=getsignal(handles.tg, getsignalid(handles.tg, 'T_PV/Integrator2'));
set(h8, 'XData', [T_t-0.2;T_t+0.2], 'YData', [G_mean;G_mean], ...
    'color', 'r', 'Linewidth', 5);
set(h10, 'XData', [T_t-0.2;T_t+0.2], 'YData', [T_mean;T_mean], ...
    'color', 'r', 'Linewidth', 5);
seq_count=handles.tg.getsignal('T_PV/Counter/p1');
if(seq_count~=0)
    if(seq_count+1<length(handles.T(seq_count)))
        set(handles.weight, 'String', num2str(seq.Weight(seq_count)));
        set(handles.userdata, 'String', num2str(seq.User(seq_count)));
        set(handles.trans_Time, 'String', ...
            num2str(handles.T(seq_count+1)-handles.T(seq_count)));
    end;
end;
guidata(hObject, handles);

% --- Executes on mouse press over figure background.

```

```

function figure1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% --- Executes on button press in Live.
function Live_Callback(hObject, eventdata, handles)
% hObject    handle to Live (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA),

%starts reading the voltage-signals from xPC and updating it on GUI_neu.
handles=guidata(hObject);
handles.tg=xpc;
%set(handles.tg, 'ShowSignals', 'off') set(handles.tg, 'ShowParameters', 'on')
set(handles.trans_Time, 'Enable', 'off', 'String', 0.5);
set(handles.irradiance_editText, 'Enable', 'off');
set(handles.temp_editText, 'Enable', 'off');
set(handles.apply, 'Enable', 'off');
set(handles.slider1, 'Enable', 'off');
set(handles.slider2, 'Enable', 'off');
set(handles.Live, 'Enable', 'off');
start(handles.live_timer);

% ---Updates live data on GUI.
function liveUpdate(obj,eventdata,hObject)
%reads the voltage signals for Irradiance&Temperature (NIDAQ) from the
%xPC-Target and updates the Irr&Temp edit_text in GUI_neu.
handles=guidata(hObject);
handles.tg=xpc;
G=getsignal(handles.tg, getsignalid(handles.tg, 'Saturation_Irr'));
Temp=getsignal(handles.tg, getsignalid(handles.tg, 'T_cell'));
set(handles.irradiance_editText, 'String', num2str(G));
set(handles.temp_editText, 'String', num2str(Temp));
set(handles.slider1, 'Value', G);
set(handles.slider2, 'Value', Temp);
apply_Callback(hObject, eventdata, handles);
guidata(hObject, handles);

% --- Executes on button press in LiveStop.
function LiveStop_Callback(hObject, eventdata, handles)
% hObject    handle to LiveStop (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
handles=guidata(hObject);
stop(handles.live_timer);
set(handles.trans_Time, 'Enable', 'on');
set(handles.irradiance_editText, 'Enable', 'on');
set(handles.temp_editText, 'Enable', 'on');
set(handles.apply, 'Enable', 'on');
set(handles.slider1, 'Enable', 'on');
set(handles.slider2, 'Enable', 'on');
set(handles.Live, 'Enable', 'on');

```

```

function SanDB_Callback(hObject, eventdata, handles)
% hObject    handle to SanDB (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SanDB as text
%        str2double(get(hObject,'String')) returns contents of SanDB as a
%        double

% --- Executes during object creation, after setting all properties.
function SanDB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SanDB (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    empty - handles not created
% until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function SamDB_Callback(hObject, eventdata, handles)
% hObject    handle to SamDB (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SamDB as text
%        str2double(get(hObject,'String')) returns contents of SamDB as a
%        double

% --- Executes during object creation, after setting all properties.
function SamDB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SamDB (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    empty - handles not created
% until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nSeries_Callback(hObject, eventdata, handles)
% hObject    handle to nSeries (see GCBO) eventdata reserved - to be

```

```

% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nSeries as text
%         str2double(get(hObject,'String')) returns contents of nSeries as a
%         double

% --- Executes during object creation, after setting all properties.
function nSeries_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nSeries (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function wind_Callback(hObject, eventdata, handles)
% hObject    handle to wind (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    structure with handles and user
% data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of wind as text
%         str2double(get(hObject,'String')) returns contents of wind as a
%         double

% --- Executes during object creation, after setting all properties.
function wind_CreateFcn(hObject, eventdata, handles)

% hObject    handle to wind (see GCBO) eventdata reserved - to be defined
% in a future version of MATLAB handles    empty - handles not created
% until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Load Module Button Callback On GUI_neu

--- Executes on button press in LoadModule.

```

function LoadModule_Callback(hObject, eventdata, handles)
% hObject    handle to LoadModule (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles

```

```

% and user data (see GUIDATA)

% This function loads the required data/parameters to the target depending
% if a measured IV-Curve or parameters from the San/SAM(CEC)-Database are
% used. If a measured IV-Curve is used the data has to be stored in .xlsx
% format with some formatting restrictions. Max ArraySize 210x2 (row,column).
% V-values have to be unique (no values appearing double,because difference
% between neighbouring V-values is used as denominator). Also the first and
% last two IV-pairs should be representable for behavior at Isc and Voc
% (are used for linear interpolation to zero crossings -> Isc, Voc). For
% example look at 'M3_Photon.xlsx'
%
% Required format of the IV-Curve-Datafile (row,column):
%      (1,2) ModuleName
%      (2,2) Rs - SeriesResistance
%      (3,2) Kappa - CurveCorrectionFactor
%      (4,2) alpha - Temperature Coefficient for Isc
%      (5,2) beta - Temperature Coefficient for Voc
%      (6,2) G1 - Reference Irradiation at IV-Measurement
%      (7,2) T1 - Reference Temperature at IV-Measurement
%      (8,2) Isc - Reference ShortcutCurrent
%      (10:210,1) Voltage Datapoints (max 200 datapoints)(0 -> Voc)
%      (10:210,2) Current Datapoints (max 200 datapoints)(Isc -> 0)
%
handles.tg=xpc;
handles.guidata(hObject);

set(handles radiobutton1,'Value',0); % sets "loaded ready"-indicator to 0

% gets number of modules in series for other calculations and plots
nSeries=str2double(get(handles.nSeries,'String'));
handles.nSer=nSeries;

% reads filename of measured IV-Curve data from GUI 'IV-Curve-Datafile'
IVcurveFile=get(handles.IVcurveFile,'String');

% if filename IVcurveFile='none.xlsx' model is build from
% San-/SAM(CEC)-Parameters specified by SanNr and SAM(CEC)Nr otherwise
% measured IV-CurveFile 'modulename.xlsx' is used for building model
% ATTENTION: special format of file is used.

% executes if IV-CurveFile is used (Modulename.xlsx)
if ~strcmp(IVcurveFile,'none.xlsx') % string comparison
    [~,~,raw]=xlsread(IVcurveFile); % reads .xlsx-file,'~' will be discarded
    [p,~]=size(raw); % gets length of array
    ModuleName=raw(1,2); % extracts ModuleName from file
    IVcurve=zeros(210,2); % initializing IV-Curve-Array
    l=p-9; % number of V-I-Valuepairs,...
    % first 9 rows are used for ModuleParameters

    % creates DataArray IVcurve for target
    IVcurve(10:p,1:2)=cell2mat(raw(10:p,1:2));
    IVcurve(2:7,2)=cell2mat(raw(2:7,2));

```

```

IVcurve(9,2)=1;
% uses linear regression over first 4 value pairs (regsize)
% to calculate Isc
regsize=4; % regression size (number of pairs for linear regression)
% smallest possible floating point number (double precision)
epsilon=realmin;
xi= IVcurve(10:10+regsize-1,1); yi= IVcurve(10:10+regsize-1,2);
xi_avg=sum(xi)/regsize; yi_avg=sum(yi)/regsize;
denom1=sum((xi-xi_avg).*xi);
if denom1==0
    denom1=epsilon;
end
ai=sum((yi-yi_avg).*xi)/denom1;
bi=sum(yi-ai*xi)/regsize;
Ishortcut=bi;
IVcurve(8,2)=Ishortcut;

% writes DataArray to target
setparam(handles.tg,getparamid(handles.tg,'Module/IVcurve','Value'),...
    IVcurve);
setparam(handles.tg,getparamid(handles.tg,'nSeriesDiv','Gain'),...
    1/nSeries);

% writes data in handles-struct for use by other functions (plot)
handles.IVcurve=IVcurve;
% Indicator (1 indicates that measured IV-Curve is used)
handles.IVcurve_ind=1;

% 20.6.2014
Rs=IVcurve(2,2);
k=IVcurve(3,2);
alpha=IVcurve(4,2);
beta=IVcurve(5,2);
G1=IVcurve(6,2);
T1=IVcurve(7,2);
Isc=IVcurve(8,2);
l=IVcurve(9,2);

%original curve
x1=IVcurve(10:10+l-1,1);
y1=IVcurve(10:10+l-1,2);

%corrected curve
G=1000; T_PV=25;
y2=y1+Isc*(G/G1-1)+alpha*(T_PV-T1);
x2=x1-Rs*(y2-y1)-k*y2*(T_PV-T1)+beta*(T_PV-T1);

% interpolation to zero crossing, corrected curve
% use linear regression over 4 value pairs for extrapolation to zerocrossing
regsize=4; % regression size (number of pairs for linear regression)
if x2(1)>0
    xi=x2(1:regsize); yi=y2(1:regsize);
    xi_avg=sum(xi)/regsize; yi_avg=sum(yi)/regsize;
    denom1=sum((xi-xi_avg).*xi);

```



```

        if denom1==0
            denom1=epsilon;
        end
        ai=sum((yi-yi_avg).*xi)/denom1;
        bi=sum(yi-ai*xi)/regsize;
        Isc=bi;
        x2(2:l+1)=x2(1:l);
        y2(2:l+1)=y2(1:l);
        x2(1)=0;
        y2(1)=Isc;
        l=l+1;
    end
    if y2(l)>0
        xv=x2(l-regsize+1:l); yv=y2(l-regsize+1:l);
        xv_avg=sum(xv)/regsize; yv_avg=sum(yv)/regsize;
        denom2=sum((xv-xv_avg).*xv);
        if denom2==0
            denom2=epsilon;
        end
        av=sum((yv-yv_avg).*xv)/denom2;
        bv=sum(yv-av*xv)/regsize;
        Voc=-bv/av;
        y2(l+1)=0;
        x2(l+1)=Voc;
        if x2(l)==x2(l+1)
            x2(l+1)=x2(l+1)+epsilon;
        end
    end
end

% determining characteristic values at STC (Voc,Isc,MPPs)
% Reading the voltage gain value displayed on GUI
V_PV_m = str2double(get(handles.voltage_gain,'String'));
% Voltage range of the PV; 10->InputVoltageRange from DAQ
V_PV=0:0.1:(10*V_PV_m);
f2=fit(x2,y2,'linearinterp');
x2max=max(x2);
z1=find(V_PV<=x2max,1,'last');
I_PV=0*V_PV;
I_PV(1:z1)=f2(V_PV(1:z1));
z2=(I_PV<0);
I_PV(z2)=0;
z3=find(I_PV==0,1,'first');
P_PV = I_PV.*V_PV; %Photo voltaic POWER
[max_power,max_voltage] = max(P_PV); % Calculating max Power&Voltage...
% from Plot
max_current=find(P_PV==max(P_PV(:))); % Calculating max current from Plot

%---- Calculating the mpp power & voltage from P_V Curve-----%
% Mpp Current rounding of to 2digits after decimal point
P_mpp=num2str(round(max_power*100)/100);
% Mpp Current rounding of to 2digits after decimal point
V_mpp=num2str(round(V_PV(max_voltage)*100)/100);
% Mpp Current rounding of to 2digits after decimal point
I_mpp=num2str(round(I_PV(max_current)*100)/100);

```

```

% displays selected modules&Characteristics on GUI
set(handles.ModuleNameSAM, 'String', ModuleName);
set(handles.ModuleNameSan, 'String', ModuleName);
set(handles.V_oc_STC, 'String', num2str(round(V_PV(z3)*1000)/1000));
set(handles.I_sc_STC, 'String', num2str(f2(0)));
set(handles.P_mpp_STC, 'String', P_mpp);
set(handles.V_mpp_STC, 'String', V_mpp);
set(handles.I_mpp_STC, 'String', I_mpp);

else
% indicates that no reference IV-curve is used for simlation
setparam(handles.tg, getparamid(handles.tg, 'Module/IVcurve', 'Value'), ...
    zeros(210, 2));

% reads selected module parameters
% offset for moduleNumber in SAN Database, matlabArrayNumber = xlsNumber-1
offset=-1;
SanNr=str2num(get(handles.SanDB, 'String'))+offset;
SamNr=str2num(get(handles.SamDB, 'String'));

% creates array of parameters from SandiaModuleDatabase.xls
SanPar=pvl_sapmmoduledb(SanNr, 'SanModDB');
% creates array of parameters from CECModuleDatabase.mat
CECModuleDB=load('CECModuleDatabaseSAM2012.11.30.mat');
SamPar=CECModuleDB.CECModuleDB(SamNr);

% extracts modulenames from arrays
ModuleNameSAM=CECModuleDB.ModuleNames(SamNr);
ModuleNameSan=SanPar.name;

% creates array of parameters for writing to target
Data=[SamPar.a_ref, SamPar.IL_ref, SamPar.IO_ref, SamPar.Rsh_ref, ...
    SamPar.Rs_ref, SanPar.a_wind, SanPar.b_wind, SanPar.delT, ...
    SanPar.AlphaIsc];

%sets parameters on target
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/MParameters', 'Value'), Data);
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/dEgdt', 'Value'), handles.dEgdt);
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/EgRef', 'Value'), handles.EgRef);
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/M', 'Value'), handles.M);
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/Sref', 'Value'), handles.G_STC);
setparam(handles.tg, ...
    getparamid(handles.tg, 'Module/Tref', 'Value'), handles.T_STC);
setparam(handles.tg, ...
    getparamid(handles.tg, 'nSeriesDiv', 'Gain'), 1/nSeries);
% setparam(handles.tg, getparamid(handles.tg, 'nSeries', 'Gain'), nSeries);
set(handles.voltage_gain, 'String', num2str(getparam(handles.tg, ...
    getparamid(handles.tg, 'V_PV_m', 'Gain'))));

```

```

% displays selected modules&Characteristics on GUI
set(handles.ModuleNameSAM, 'String', ModuleNameSAM);
set(handles.ModuleNameSan, 'String', ModuleNameSan);
set(handles.V_oc_STC, 'String', SanPar.Voc0);
set(handles.I_sc_STC, 'String', SanPar.Isc0);
set(handles.P_mpp_STC, 'String', SanPar.Imp0*SanPar.Vmp0);
set(handles.V_mpp_STC, 'String', SanPar.Vmp0);
set(handles.I_mpp_STC, 'String', SanPar.Imp0);

handles.SanPar=SanPar;
handles.SamPar=SamPar;
handles.IVcurve_ind=0;      %IVcurve_indicator, 1=on

end

set(handles.radiobutton1, 'Value', 1); % indicates that module loaded completely

guidata(hObject, handles);

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)

% hObject    handle to radiobutton1 (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

```

Datalog Button Callback On GUI_neu

--- Executes on button press in datalog.

```

function datalog_Callback(hObject, eventdata, handles)
% hObject    handle to datalog (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA) logging data from target
handles.tg=xpc;

logfile=get(handles.datalogStr, 'String'); % gets logfile-name from GUI
fid=fopen(logfile, 'a+'); % creates fid... file identifier
handles.fid=fid; % stores fid in handles struct for other functions
c=fix(clock); % gets clock and round down seconds

% prints header for clock/date and desired variables/parameters
fprintf(fid, '\n %-10s %u/%u/%u\n', 'Date(yyyy/mm/dd):', c(1), c(2), c(3));
fprintf(fid, '%12s %12s %12s %12s %12s\n', 'Time', 'Irradiance', ...
    'Temperature', 'V_PV', 'I_PV');

set(handles.datalog, 'Enable', 'off'); % disables 'datalog' button
handles.fopen=1; % indicates that logfile is opened, for other functions
start(handles.log_timer); % starts timer for log function
guidata(hObject, handles);

```

```

% logs data to .txt file
function log(obj,eventdata,hObject)

handles.tg=xpc;
handles=guidata(hObject);

if handles.fopen==1      % checks if datafile is opened for read/write
    c=fix(clock);        % gets clock and round down seconds

    % get signals from target
    signals=getsignal(handles.tg,...
        [getsignalid(handles.tg,'G/G_mean/Product')...
        getsignalid(handles.tg,'T_PV/T_PV_mean/Product')...
        getsignalid(handles.tg,'V_m_mean/Product')...
        getsignalid(handles.tg,'I_m_mean/Product')]);
    G=signals(1);
    T=signals(2);
    V_PV=signals(3);
    I_PV=signals(4);

    % prints clock and desired variables/parameters to the logfile
    % specified by handles.fid
    fprintf(handles.fid,'      %2u:%2u:%02u %12.4f %12.4f %12.4f %12.4f\n',...
        c(4),c(5),c(6),G,T,V_PV,I_PV);
end

```

Stop Log Button Callback On GUI_neu

--- Executes on button press in StopLog.

```

function StopLog_Callback(hObject, eventdata, handles)
% hObject    handle to StopLog (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)
handles=guidata(hObject);

if handles.fopen==1      % fopen=1 if datalogfile is opened for read/write
    stop(handles.log_timer); % stops datalog-timer
    fclose(handles.fid);    % close datalogfile
end
set(handles.datalog,'Enable','on'); % enables datalog-button
handles.fopen=0;                % indicates that file is closed
guidata(hObject,handles);

function datalogStr_Callback(hObject, eventdata, handles)
% hObject    handle to datalogStr (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of datalogStr as text
%        str2double(get(hObject,'String')) returns contents of datalogStr
%        as a double

```

```

% --- Executes during object creation, after setting all properties.
function datalogStr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to datalogStr (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function IVcurveFile_Callback(hObject, eventdata, handles)
% hObject    handle to IVcurveFile (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    structure with handles
% and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of IVcurveFile as text
%       str2double(get(hObject,'String')) returns contents of IVcurveFile
%       as a double

% --- Executes during object creation, after setting all properties.
function IVcurveFile_CreateFcn(hObject, eventdata, handles)
% hObject    handle to IVcurveFile (see GCBO) eventdata reserved - to be
% defined in a future version of MATLAB handles    empty - handles not
% created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Published with MATLAB® R2013b

A.7 Matlab Code Embedded Matlab Function *CurveOrParam*

```
function I = CurveOrParam(V,MParameters,dEgdT,EgRef,...
    S,T,M,Sref,Tref,windspeed,IVcurve)
% V - VoltageValue for Current-Calculation
% MParameters - ModuleParameters from PVPMC- and SystemAdvisorModel-Datasbase
% dEgdT - Temperature-Dependency of EnergyBandGap
% EgRef - EnergyBandGap at reference conditions
% S - Total incident irradiance [W/m²]
% T - ModuleBacksheetTemperature (or AmbientTemp) [°C],
% M - Relative AirMassModifier (Mx/Mref],
%     Mref=1.5, Mx=local airmass, M=1 means AM1.5
% Sref - Reference Irradiance [1000 W/m²]
% Tref - Reference Temperature [25°C]
% windspeed - Ambient windspeed [m/s],
% IVcurve - DataSet for measured IV-Curve (210x2 Array)
%     IVcurve(1:9,2)= Reference Values for measured IV-Curve
%     IVcurve(10:210,1:2)= measured V-I-ValuePairs
%     IVcurve(10,2)~ first current value from measured IV-Curve
%     If IVcurve(10,2)=0, Model is build from San-/SAM-Parameters
%     If IVcurve(10,2)~=0, Model is build from measured IV-Curve
%
I=nan; % Initialization of I

% Calculation of I if a measured IV-curve is used for Module-Representation
% Method according to IEC 60891 [2]

if IVcurve(10,2)~=0
    Rs=IVcurve(2,2); % SeriesResistance
    k=IVcurve(3,2); % Kappa, CurveCorrectionFactor
    alpha=IVcurve(4,2); % Temperature Coefficient Isc
    beta=IVcurve(5,2); % Temperature Coefficient Voc
    G1=IVcurve(6,2); % Reference Irradiation at IV-Measurement
    T1=IVcurve(7,2); % Reference Temperature at IV-Measurement
    Isc=IVcurve(8,2); % Reference ShortcutCurrent
    l=IVcurve(9,2); % number of IV-Pair Values

    % original curve
    x1=IVcurve(10:10+l-1,1); % Voltage DataSet (0 -> Voc)
    y1=IVcurve(10:10+l-1,2); % Current DataSet (Isc -> 0)

    % corrected curve according to 1) & 2) in Reference [2] (IEC 60891)
    y2=y1+Isc*(S/G1-1)+alpha*(T-T1);
    x2=x1-Rs*(y2-y1)-k*y2*(T-T1)+beta*(T-T1);

    % interpolation to zero crossing (Isc & Voc for corrected curve)
    % using linear regression over 4 value pairs for extrapolation
    % to zerocrossing
    regsize=4; % regression size (number of pairs for linear regression)
    %smallest possible floating point number (double precission)
    epsilon=realmin;
    if x2(1)>0
        xi=x2(1:regsize); yi=y2(1:regsize);
        xi_avg=sum(xi)/regsize; yi_avg=sum(yi)/regsize;
```



```

        denom1=sum((xi-xi_avg).*xi);
        if denom1==0
            denom1=epsilon;
        end
        ai=sum((yi-yi_avg).*xi)/denom1;
        bi=sum(yi-ai*xi)/regsize;
        Isc=bi;
        x2(2:l+1)=x2(1:l);
        y2(2:l+1)=y2(1:l);
        x2(1)=0;
        y2(1)=Isc;
        l=l+1;
    end
    if y2(l)>0
        xv=x2(l-regsize+1:l);    yv=y2(l-regsize+1:l);
        xv_avg=sum(xv)/regsize;    yv_avg=sum(yv)/regsize;
        denom2=sum((xv-xv_avg).*xv);
        if denom2==0
            denom2=epsilon;
        end
        av=sum((yv-yv_avg).*xv)/denom2;
        bv=sum(yv-av*xv)/regsize;
        Voc=-bv/av;
        y2(l+1)=0;
        x2(l+1)=Voc;
        if x2(l)==x2(l+1)
            x2(l+1)=x2(l+1)+epsilon; % avoids division through 0...
                                     % at interpolation
        end
        l=l+1;
    end

    ind=nan;
    % determine Voltage-Intervall, if V > max(x2)=Voc then I=0
    for i=1:l
        if x2(i)<=V;    % 7.4.2014: < -> <=
            ind=i;
        else
            break
        end
    end

    % Interpolation between discrete IV-Pairs
    if ind<l    % 7.4.2014: l-1 -> l
        slope=(y2(ind+1)-y2(ind))/(x2(ind+1)-x2(ind));
        I=y2(ind)+slope*(V-x2(ind));
    else    % if V >= Voc then I=0
        I=0;
    end

else
    % Calculation of I through single-diode-model using PVPMC-Database/-Library

```

```

% and 5 parameter model [1]

% Reference parameters from CEC-database
a_ref=MParameters(1); % modified diode ideality factor at...
                        % reference conditions (RC)
IL_ref=MParameters(2); % Light-generated current (or photocurrent) at RC
I0_ref=MParameters(3) ; % diode reverse saturation current at RC
Rsh_ref=MParameters(4); % shunt resistance at RC
Rs_ref=MParameters(5); % series resistance at RC

% Parameters from PVPMC-database
a=MParameters(6);      % wind parameter (upper T-limit at low speed)
b=MParameters(7);      % wind parameter (dropping rate)
deltaT=MParameters(8); % parameter giving Temp-difference...
                        % cell-module backsheet at RC
AlphaIsc=MParameters(9); % Isc temperature coefficient

% redefining S and Sref
E=S;
E0=Sref;

% Tamb=T; % uncomment if T=ambient temperature,
           % comment if T=module backsheet temperature
Tmodule=T; % comment if T=ambient temperature,
           % uncomment if T=module backsheet temperature

% ----- Code from PVPMC-library, pvl_sapmcelltemp -----

% [Tcell Tmodule] = pvl_sapmcelltemp(E, E0, a, b, windspeed, Tamb, deltaT)
%
%
% PVL_SAPMCELLTEMP Estimate cell temperature from irradiance, windspeed,
% ambient temperature, and module parameters (SAPM)
%
% Syntax
% Tcell = pvl_sapmcelltemp(E, E0, a, b, windspeed, Tamb, deltaT)
% [Tcell Tmodule] = pvl_sapmcelltemp(E, E0, a, b, windspeed, Tamb, deltaT)
%
% Description
% Estimate cell and module temperatures per the Sandia PV Array
% Performance model (SAPM, SAND2004-3535), when given the incident
% irradiance, wind speed, ambient temperature, and SAPM module
% parameters.
%
% Inputs
% E - Total incident irradiance in W/m^2. E must be a scalar or a vector
%     of the same size as windspeed, and Tamb. Must be >=0.
% E0 - Reference irradiance used when determining delta T, in W/m^2. E0
%      must be a scalar. Must be >=0;
% a - SAPM module parameter for establishing the upper limit for module
%      temperature at low wind speeds and high solar irradiance (see SAPM
%      eqn. 11). Must be a scalar.
% b - SAPM module parameter for establishing the rate at which the module

```

```

%     temperature drops as wind speed increases (see SAPM eqn. 11). Must be
%     a scalar.
%     windspeed - Wind speed in m/s at a height of 10 meters. windspeed must
%     be a scalar or a vector of the same size as E and Tamb. Must be >=0;
%     Tamb - Ambient dry bulb temperature in degrees C. Tamb must be a scalar
%     or a vector of the same size as windspeed and E. Must be >= -273.15.
%     deltaT - SAPM module parameter giving the temperature difference
%     between the cell and module back surface at the reference irradiance,
%     E0. Must be a numeric scalar >=0.
%
% Outputs
%     Tcell - A column vector of cell temperatures in degrees C.
%     Tmodule - A column vector of module back temperature in degrees C.
%
% References
%     [13] King, D. et al, 2004, "Sandia Photovoltaic Array Performance Model",
%     SAND Report
%     3535, Sandia National Laboratories, Albuquerque, NM
%
% See also PVL_SAPM
% -----
% end

%     Tmodule = E.*(exp(a+b.*windspeed))+Tamb; % uncomment if T=ambient temp;
%                                           % comment if T=module backsheet temp
%     Tcell = Tmodule + E./E0.*deltaT;

% ----- Code end  PVPMC-library, pvl_sapmcelltemp -----

% ----- code from PVPMC-library, pvl_calcp_params_desoto -----

% Silicon (Si):
%     EgRef = 1.121
%     dEgdT = -0.0002677
%     M = polyval([-0.000126 0.002816 -0.024459 0.086257 0.918093], AMa)
%     Source = Reference 1
% Cadmium Telluride (CdTe):
%     EgRef = 1.475
%     dEgdT = -0.0003
%     M = polyval([-2.46E-5 9.607E-4 -0.0134 0.0716 0.9196], AMa)
%     Source = Reference 4
% Copper Indium diSelenide (CIS):
%     EgRef = 1.010
%     dEgdT = -0.00011
%     M = polyval([-3.74E-5 0.00125 -0.01462 0.0718 0.9210], AMa)
%     Source = Reference 4
% Copper Indium Gallium diSelenide (CIGS):
%     EgRef = 1.15
%     dEgdT = ???
%     M = polyval([-9.07E-5 0.0022 -0.0202 0.0652 0.9417], AMa)
%     Source = Wikipedia
% Gallium Arsenide (GaAs):
%     EgRef = 1.424
%     dEgdT = -0.000433

```

```

%      M = unknown
%      Source = Reference 4
%
% Reference parameter should be taken and input from CEC database or similar

% k=1.3806488e-23; %Boltzman's constant in units of J/K
k = 8.617332478e-5; % Boltzmann constant in units of eV/K

Tref_K=Tref+273.15; % Reference cell temperature in Kelvin
Tcell_K=Tcell+273.15; % cell temperature in Kelvin

% We divide by S later, so make S a very small number for any S == 0.
% S=G;
if (S==0) ;
    S= 1E-10;
end

E_g=EgRef.*(1+dEgdT*(Tcell_K-Tref_K)); % Equation 10 in [1]
nNsVth=a_ref.*(Tcell_K./Tref_K); % Equation 8 in [1]
IL=S./Sref.*M.*(IL_ref+AlphaIsc.*(Tcell_K-Tref_K)); % Equation 9 in [1]
I0=I0_ref.*((Tcell_K./Tref_K).^3).*exp((EgRef./(k.*Tref_K))-...
    (E_g./(k.*Tcell_K)));
Rsh=Rsh_ref.*(Sref./S); % Equation 12 in [1]
Rs =Rs_ref;
% ----- code end  PVPMC-library, pvl_calcp_params_desoto -----

% ----- code from PVPMC-library, pvl_singlediode -----
% ----- code from PVPMC-library, pvl_singlediode, I_from_V -----
x = Rs.*I0.*Rsh.*...
    exp(Rsh.*(Rs.*(IL+I0)+V)./(nNsVth.*(Rs+Rsh)))./...
    (nNsVth.*(Rs + Rsh));
% ----- code end  PVPMC-library, pvl_singlediode, I_from_V -----

% ----- code from PVPMC-library, pvl_singlediode, wapr_vector -----
% function [w] = wapr_vec(x,varargin)+++++
% x=argW;
% Approximating the W function
% -----
%
% wapr.m is the Matlab version of Algorithm 743 (written in
% FORTRAN), Collected Algorithms from ACM.
% Web reference: http://www.netlib.org/toms/743
%
% wapr.m written by D. A. Barry, 23 June 2003 (d.a.barry@ed.ac.uk)
%
% wapr_vec.m modifies wapr.m to improve performance (using vector
% operations rather than a loop) but also restricts inputs to positive
% real numbers.
%
% wapr_vec.m by C. W. Hansen, 27 Sept. 2012 (cwhanse@sandia.gov)
%
% Archival journal references:
%

```

```

% Barry, D. A., P. J. Culligan-Hensley, and S. J. Barry. 1995.
% Real values of the W-function. Association of Computing
% Machinery Transactions on Mathematical Software. 21(2): 161-171.
%
% Barry, D. A., S. J. Barry, and P. J. Culligan-Hensley. 1995.
% Algorithm 743: WAPR: A FORTRAN routine for calculating real
% values of the W-function. Association of Computing Machinery
% Transactions on Mathematical Software. 21(2): 172-181.
%
% -----
% Only consider real non-negative arguments
if any(any(imag(x) ~= 0)) || any(any(x<0))
    fprintf(1, '\r%s\r', 'Complex and non-negative arguments not supported');
    return
end

%
%
% nbits is the number of bits (less 1) in the mantissa of the
% floating point number number representation of your machine.
% It is used to determine the level of accuracy to which the W
% function should be calculated.
%
% Most machines use a 24-bit matissa for single precision and
% 53-56 bits for double precision. The IEEE standard is 53
% bits. The Fujitsu VP2200 uses 56 bits. Long word length
% machines vary, e.g., the Cray X/MP has a 48-bit mantissa for
% single precision.
%
% Matlab uses the IEEE standard mantissa of 53 bits, thus nbits
% is set to 52. Uses can check this result by running the following
% section of code, although this is only necessary if Matlab departs
% from the IEEE standard.
%v=10;
%i=0;
%while v > 1
%    i=i+1;
%    b=2^(-i);
%    v=1+b;
%end
%nbits=i-1;
%j=-log10(b);
%if m == 1
%    fprintf(1, '\r%s%3.0f%s\r%s%3.0f%s\r', ' nbits is', nbits, ...
%    '.', ' Expect', j, ' significant digits from wapr.')
% end

nbits=52;

% nbits should not need to change while Matlab uses
% the IEEE standard described above.
% Various constants
em=-exp(-1);
em9=-exp(-9);
c13=1/3;

```

```

c23=2*c13;
em2=2/em;
e12=-em2;
tb=.5^nbits;
tb2=sqrt(tb);
x0=tb^(1/6)/2;
x1=(1-17*tb^(2/7))*em;
an3=8/3;
an4=135/83;
an5=166/39;
an6=3167/3549;
s2=sqrt(2);
s21=2*s2-3;
s22=4-3*s2;
s23=s2-2;

% Calculate W for each element in the argument vector x

% initialize flags and output

w=NaN;
itflag=0*x;

% calculate W using different methods for different ranges of argument x

if abs(x)<=x0;
    w=x./(1+x./(1+x./(2+x./(0.6+0.34*x))));
    itflag=1;
elseif abs(x)>x0 & x<=x1;
    delx = x - em;
    reta=sqrt(e12*delx);
    w=reta./(1+reta./(3+reta./(reta./(an4+reta./(reta*an6+an5))+an3)))-1;
    itflag=1;
end

if x>x1 & x <= 20 ;
    reta=s2*sqrt(1-x/em);
    an2=4.612634277343749*sqrt(sqrt(reta+1.09556884765625));
    w=reta./(1+reta./(3+(s21*an2+s22).*reta./(s23*(an2+reta))))-1;
end

if x > 20;
    zl=log(x);
    w=log(x./log(x./zl.^exp(-1.124491989777808 ./(.4225028202459761+zl))));
end

if itflag==0 ;
    zn=log(x./w)-w;
    temp=1+w;
    temp2=temp+c23*zn;
    temp2=2.*temp.*temp2;
    w=w.*(1+(zn./temp).*(temp2-zn)./(temp2-2.*zn));
end

```

```

% ----- code end PVPMC-library, pvl_singlediode, wapr_vector -----

    inputterm = w; %wapr_vec(argW);

% ----- code from PVPMC-library, pvl_singlediode, I_from_V -----
% function [I]=I_from_V(Rsh, Rs, nNsvth, V, Io, Iphi)
%
% % calculates I from V per Eq 2 Jain and Kapoor 2004
% % uses Lambert W implemented in wapr_vec.m
% % Rsh, nVth, V, Io, Iphi can all be vectors
% % Rs can be a vector, but should be a scalar
%
% argW = Rs.*Io.*Rsh.*...
%       exp(Rsh.*(Rs.*(Iphi+Io)+V)./(nNsvth.*(Rs+Rsh)))./...
%       (nNsvth.*(Rs + Rsh));
% inputterm = wapr_vec(argW);
%
% % Eqn. 4 in Jain and Kapoor, 2004
% I = -V./(Rs + Rsh) - (nNsvth./Rs) .* inputterm + ...
%     Rsh.*(Iphi + Io)./(Rs + Rsh);
% end
I = -V./(Rs + Rsh) - (nNsvth./Rs) .* inputterm + ...
    Rsh.*(IL + IO)./(Rs + Rsh);
% ----- code end PVPMC-library, pvl_singlediode, I_from_V -----
% ----- code end PVPMC-library, pvl_singlediode -----

```

end

% Reference:

```

% [1] W. De Soto et al., "Improvement and validation of a model for
%      photovoltaic array performance", Solar Energy, vol 80, pp. 78-88, 2006.
% [2] IEC60891 (Edition 2.0, 2009-12), "International Standard"

```

Published with MATLAB® R2013b

A.8 Matlab Code Matlab Function *Radiation*

```
function G = Radiation(V,Rsh,alpha,T_PV,Isc_stc)

% Standard Conditions
G_stc=1000;
T_stc=25;

% According to "International Standard , IEC 60891", page 6
G = V/Rsh*G_stc/Isc_stc*(1+alpha*(T_PV-T_stc));
```

Published with MATLAB® R2013b

Appendix B

Curriculum Vitae

Personal Information

Name	Reinhard Mondl
Place of birth	Mistelbach, Austria
Languages	German, English

Educational History

Since Sept. 2012	Master study in physics at the University of Vienna
Oct. 2008 - Jun. 2012	Bachelor study in physics at the University of Vienna
Nov. 2007 - Jun. 2008	Studienberechtigungsprüfung, VHS Ottakring, Vienna
Sept. 1997 - Jun. 2001	Engineering school, Vienna, Schellinggasse
Sept. 1993 - Jun. 1997	Main school, Wolkersdorf im Weinviertel
Sept. 1989 - Jun. 1993	Primary school, Unterolberndorf im Kreuttal