



BACHELOR THESIS

Maximum Power Point Tracking: Algorithm and Software Development

Authors:

Stefan Moring - 1525379

Anton Pols - 4001648

Supervisor:

Dr. J. Popovic

Delft University of Technology
Faculty of EEMCS

June 27, 2012

Preface

This thesis is part of a design project conducted for the Nuon Solar Team. The goal of the project was to design a Maximum Power Point Tracker with a very high efficiency. The project has been conducted with a total of six persons, divided into three subgroups. This thesis describes the research, design and prototyping of the algorithms and other software run on the controller.

We would like to thank dr. Jelena Popovic, supervisor of our bachelor graduation project, for her technical as well as non-technical support and Milos Acanski MSc, for his contributions to the project and help with the models. Also, we would like to thank Kasper Zwetsloot for his extremely valuable help and comments while measuring our prototype and we would like to thank the Nuon Solar Team for giving us this incredible opportunity.

Delft, June 27, 2012
Stefan Moring
Anton Pols

Summary

Currently, the Nuon Solar Team is building their new solar car, which makes use of solar panels. Solar panels have a Maximum Power Point (MPP), which, when operated at that point, ensures the maximum available power is obtained from them. In this thesis, different options were explored to solve the problem of tracking the MPP of a solar panel. The focus of this thesis was the software part of tracking the MPP and the goal of this thesis was to implement the most efficient algorithm that works in fast changing levels of irradiance and when the solar panels are partially shaded.

In order to realize this goal, we first did a literature survey to learn about the available algorithms and their respective advantages and disadvantages. Subsequently, we chose the algorithms which had potential and we could realistically implement. Those algorithms were P&O, InCond and their adaptive variants. We simulated those algorithms for their efficiencies in Simulink and implemented them onto a microcontroller. Lastly, we made an experimental setup and measured the algorithms for their efficiencies.

The results showed that based on the simulations, the adaptive InCond algorithm is the most efficient algorithm, also in fast changing levels of irradiance. As the simulation did not simulate partially shaded solar panels, we can not make any conclusions about the performance of the different algorithms in that case. In the experimental setup, we verified that the controller and all implemented algorithms worked correctly. However, we were not able to verify all the simulation results, as we could only test the sudden shading condition. The MPPT was able to reliably track the MPP of a solar panel, depending on what algorithm was used. Some algorithms were more susceptible to noise than others, and eventually we concluded that the adaptive P&O algorithm performed best in the experimental setup, because it is least susceptible to noise and has the advantages of an adaptive algorithm. We did not measure the efficiency of the MPPT. However, based on the simulations and the measured efficiencies of the other subsystems of the MPPT, we are confident that we succeeded in designing and implementing an MPPT algorithm with an efficiency of at least 95%.

Contents

Preface	i
Summary	ii
1 Introduction	1
2 Brief of Requirements	3
2.1 Required Conditions	3
2.2 Preferred Conditions	3
2.3 Optional Conditions	4
3 Design of the MPPT	5
3.1 Top Level Design of the MPPT	5
3.1.1 Solar Panel	5
3.1.2 DC-DC Converter	6
3.1.3 MPPT Controller	7
3.1.4 Load	8
3.2 Controller Topology	8
3.2.1 Analog to Digital Converters	8
3.2.2 Algorithm	9
3.2.3 PWM Generator	9
3.2.4 Communication	9
3.3 Design Process	9
4 State of the Art MPPT Algorithms	11
4.1 Hill Climbing Algorithms	11
4.1.1 Perturb & Observe	12
4.1.2 $\frac{dP}{dV}$ Feedback Control	12
4.1.3 Incremental Conductance	13
4.1.4 Drawbacks of the Hill Climbing Algorithms	13
4.1.5 Making the algorithms adaptive	15
4.2 Fuzzy Logic Control	15
4.3 Current Sweep Method	17
4.4 DIRECT Method	18

4.5	Summary	19
5	Simulation of the MPPT Algorithms Performance	20
5.1	Simulation Model	20
5.1.1	Ideal Model	20
5.1.2	MPPT Algorithm Model	21
5.1.3	Simulation Conditions	22
5.2	Simulation Results	22
5.2.1	Steady State Analysis	23
5.2.2	Increasing/Decreasing Irradiance	23
5.2.3	Sudden shading/exposure of the solar panel	24
5.3	Conclusion	26
6	Implementation Platform	27
6.1	Requirements	27
6.1.1	Analog to Digital Conversion	27
6.1.2	PWM Generator	28
6.1.3	Operating Frequency	28
6.1.4	Communication	28
6.2	Microcontroller Comparison	28
6.2.1	68HC11	28
6.2.2	8051	29
6.2.3	Atmega168/Pic16F87X	29
6.3	Conclusion	29
7	Implementation of the MPPT Controller	30
7.1	Configuration of the microcontroller	30
7.1.1	PWM Generator	30
7.1.2	Analog to Digital Conversion	31
7.1.3	Timing	31
7.2	Implementation of the different algorithms	32
7.2.1	P&O	32
7.2.2	InCond	33
7.2.3	Making the Algorithms Adaptive	33
7.3	Implementation of the CAN Bus Protocol	33
8	Experimental Evaluation	34
8.1	Software Verification	34
8.1.1	Analog to Digital Converters	34
8.1.2	PWM Signal	34
8.1.3	Timer Interrupt Frequency	35
8.1.4	Algorithm Verification	35
8.1.5	CAN Bus Controller	35
8.2	Experimental Setup	36
8.2.1	Experimental Hardware Setup	36
8.2.2	Test Conditions	36

8.3 Results of the Experimental Evaluation	37
8.3.1 Influence of the System Parameters	37
8.3.2 Algorithm Performance	38
9 Conclusion and Recommendations	45
9.1 Conclusion	45
9.2 Recommendations	46
Bibliography	47
Appendix A: Matlab Simulation Code	51
Appendix B: Simulink Simulation Block Scheme	53
Appendix C: C Code files	54
C.1 MPPTFunctions.c	54
C.2 MPPTmain.c	57
C.3 can.c	60
Appendix D: Brief of Requirements	64
D.1 Use of the MPPT	64
D.2 Ecological Requirements	64
D.2 Technical Requirements	65
D.4 Production and Installation Requirements	65
D.5 Requirements on Product End-of-Life	66

Chapter 1

Introduction

Solar panels generate power by using the photovoltaic effect: electrons are transferred between different energy bands in the atom by means of irradiation. The solar panel has a characteristic p-v characteristic where a global maximum is present. This means that for a different operating point of the solar panel, a different output power is obtained. The maximum power is obtained when the solar panel operates at the voltage where the global maximum of the p-v characteristic is present. Therefore, only for one specific operating point, the maximum power output is obtained from the solar panel. This point in the p-v characteristic is called the Maximum Power Point (MPP). This MPP changes when the irradiation and temperature changes or when the solar panel is partially shaded [1].

To track the constantly changing MPP a device is needed, this device is called the Maximum Power Point Tracker (MPPT). The MPPT consists of two main parts, a microcontroller to track the MPP and a converter to convert the generated voltage to a desired level for the load. An algorithm runs on the microcontroller to track the MPP. There are a lot of different algorithms to track the MPP [2], but they all do not work in fast changing levels of irradiance or when the solar panel is partially shaded [3]. This is a problem for us, because we have to develop an MPPT for the Nuon Solar Team. This team builds a solar car, so it is important for them that the MPP is tracked in an environment where there are fast changing levels of irradiance and the solar panel is partially shaded.

In this thesis, we explore different options to solve these problems. Our goal is to implement the most efficient algorithm that works in fast changing levels of irradiance and when the solar panels are partially shaded. Especially the efficiency of the algorithm is important, because we want to make an MPPT with a very high efficiency. Furthermore, the implementation complexity of the algorithm should not be too high and it must be executable on a microcontroller.

First, we state the Brief of Requirements (Chapter 2) and we describe the de-

sign of the MPPT and the subsystems relevant to us (Chapter 3). Secondly, we discuss the different algorithms available and their advantages and limitations (Chapter 4) and we simulate the different algorithms (Chapter 5). Thirdly, we discuss the requirements of the microcontroller and choose an appropriate one (Chapter 6) and implement the algorithms on the chosen microcontroller (Chapter 7). Fourthly, we measure the efficiency of the algorithms and evaluate the results (Chapter 8). Finally, we present conclusions and recommendations based on our experience (Chapter 9).

Chapter 2

Brief of Requirements

In this chapter, we briefly list the requirements required by the Nuon Solar Team. The requirements are divided into three parts: required, preferred and optional. The full brief of requirements can be found in Appendix D.

2.1 Required Conditions

- The MPPT must have an efficiency of minimal 95%.
- The MPPT has to be installed in fivefold in Nuna's car, resulting in a nominal input power between 20W and 100W and an absolute maximum of 200W.
- The MPPT accepts input voltage between 60V and 120V and provides an output voltage between 120V and 160V.
- The MPPT provides plug-and-play connectivity, thus every team member of Nuna can install it.
- The MPPT needs to work independent, but must be able to react to externally given commands.
- The MPPT is maintenance-free during operation.

2.2 Preferred Conditions

- The MPPT needs to communicate with the car using CAN-bus communication on a send-only basis, providing additional information on at least:
 - Power in [W]
 - Power out [W]
 - Input Voltage [V]

- Output Voltage [V]
 - Output Current [A]
- An external reset switch is needed for resetting the MPPT during an infinite loop or error.
- Extra safety systems, such as fuses to prevent overcharging or limit short circuit damage are applied to the design.
- Input and output and other high-voltage connections need to be properly shielded.

2.3 Optional Conditions

- The condition of the MPPT can be controlled visually using multiple leds.
- The design is robust, light and compact.
- The MPPT will work in the conditions given in Australia during the World Solar Challenge.
- All the used components meet the specifications as provided by Veolia World Solar Challenge.
- The case and connections are dust-sealed and are shock and vibration proof.

Chapter 3

Design of the MPPT

In order to successfully design a controller for an MPPT, first, we must get a clear picture of the general lay-out of an MPPT and have some general knowledge of his subsystems (Section 3.1). Subsequently, we cover the controller of the MPPT in more depth (Section 3.2) and we discuss the design process, we used for designing our MPPT controller (Section 3.3).

3.1 Top Level Design of the MPPT

Figure 3.1 depicts a block diagram of a system consisting of a load powered by a solar panel equipped with an MPPT. There are four blocks in this scheme: a solar panel, a DC-DC power converter, an MPPT controller and a load. In our case, the solar panel will be the solar panels of the NUNA and the load will be the batteries of the NUNA. The DC-DC Power Converter transforms the voltage of the solar panels to the desired voltage of the batteries. It also determines the operating point of the solar panels. In the following subsections, we will elucidate these subsystems, the solar panel in 3.1.1, the DC-DC converter in 3.1.2, the MPPT controller in 3.1.3 and the load in 3.1.4.

3.1.1 Solar Panel

The solar panel is the source of all the energy available to the NUNA. As can be seen from the model of a solar panel depicted in figure 3.2, the internal impedance consists mainly of a diode. Because the internal impedance determines the ideal operating point and the diode is a non-linear circuit element, the i-v characteristic of a solar panel is non-linear. The i-v characteristic of a typical solar panel is shown in figure 3.3 by the dotted line. The resulting p-v characteristic is depicted by the green line. As can be seen from the figure, the p-v characteristic has a maximum at a certain voltage. This voltage is known as the Maximum Power Point (MPP).

The MPP of a solar panel is dependent on several factors, namely the amount

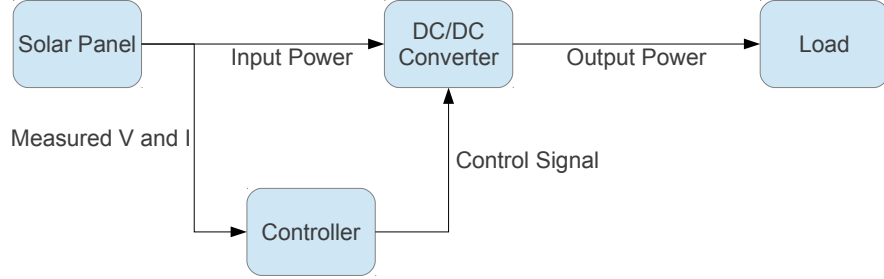


Figure 3.1: MPPT block scheme

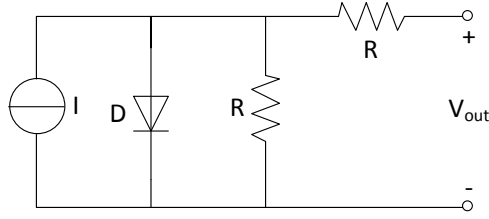


Figure 3.2: Equivalent model of a solar panel [3]

of irradiance, the temperature and the shading of the panel. All these factors have an impact on the location of the MPP. Therefore, the MPP of a solar panel is constantly changing and an MPPT is needed to obtain the maximum power output.

3.1.2 DC-DC Converter

The DC-DC converter converts the input DC voltage to another DC voltage. The converter used is a boost converter, as the voltage required to charge the batteries is higher than the output voltage of the solar panel. Figure 3.4 shows the schematic of the boost converter. The three main components are the inductor, the MOSFET switching device and the diode[4]. When the switch is open, the current from the source charges the inductor. When the switch is closed, the energy stored in the inductor adds to the source and thus increases

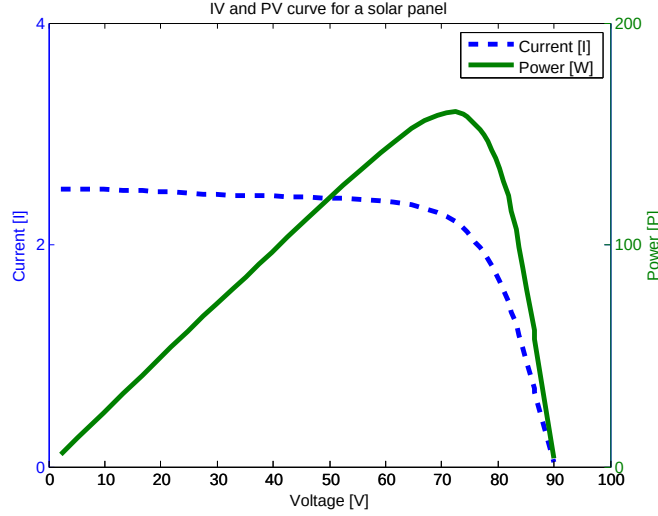


Figure 3.3: The i-v and p-v characteristic of a solar panel

the output voltage. The duty cycle of a PWM signal determines the ratio between the input and output voltage. In case of an ideal switching device and when losses are neglected, the ratio between V_{in} and V_{out} can be calculated with formula 3.1. In this formula, D represents the duty cycle of the PWM signal and has an value between 0 and 1[4]. As can be seen in formula 3.1, V_{out} increases as the duty cycle increases. We shall use this property to adjust the operating point of the MPPT.

$$\frac{V_{out}}{V_{in}} = \frac{1}{1 - D} \quad (3.1)$$

3.1.3 MPPT Controller

The MPPT controller executes the algorithm to find the MPP. The input of the controller is the measured output voltage and current of the solar panel. This value is not the actual value of the output voltage and current, but the actual value has been converted to a value between 0 and 5 V. Based on these inputs, the algorithm performs its calculations. The output of the controller is the adjusted duty cycle of the PWM, which drives the DC-DC converter's switching device. A different duty cycle causes a different operating point. In addition to these calculations, the controller also has to send the measured output voltage and current to the operating system of the NUNA. These values are used to track how much energy is generated and to spot any failures or errors in the system.

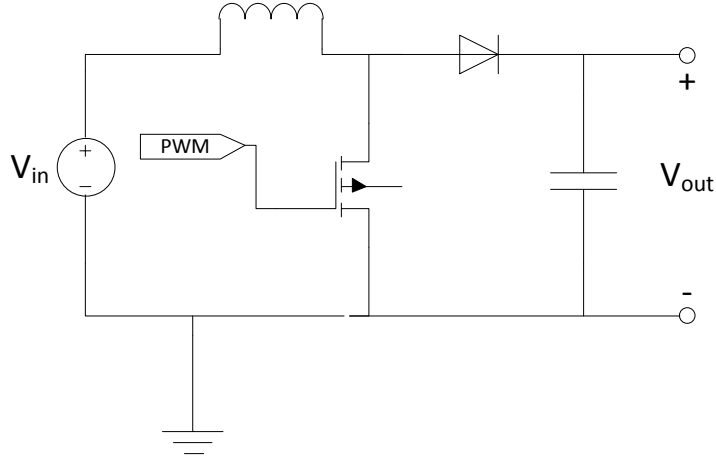


Figure 3.4: Boost Converter Topology [4]

3.1.4 Load

The load of the DC-DC converter consists of a battery pack. Connected in parallel to this battery pack, are the electrical motors which drive the car and all the electronics which are used to control the car. As the load is of no further importance to the controller itself, it will not be discussed any further.

3.2 Controller Topology

Now that the top level topology of the MPPT is known, we look further into the required functions of the controller itself. The controller is required to have three functions and one would be nice to have function: the analog to digital converter (3.2.1), the algorithm (3.2.2) and the PWM output generator (3.2.3) are all required functions, communication with the NUNA operating system is a would be nice to have function (3.2.4).

3.2.1 Analog to Digital Converters

The main task of the Analog to Digital Converter (ADC) is to measure the input signals, which are between 0 and 5 V, that represents the output voltage and current of the solar panel and convert them into binary numbers with which the controller can perform its calculations. These measurements and conversions

should be as accurate as possible, as inaccurate measurements and conversions could lead to inaccurate tracking of the MPP.

3.2.2 Algorithm

The main operation of the controller is performing the calculations to determine the next operating point. The inputs of this process are the binary numbers provided by the ADCs, those binary numbers represent the solar panel's output voltage and current. Based on the calculations, the controller output should be a signal which tells the PWM generator the value of the duty cycle.

3.2.3 PWM Generator

The input of the PWM generator module is the desired duty cycle of the PWM signal. Based on this input, the PWM generator creates a PWM signal with this desired duty cycle and this PWM signal is subsequently sent to the switching device of the DC-DC converter.

3.2.4 Communication

Another task of the controller is that it should take care of communications with the operating system of the NUNA. As can be read in the brief of requirements, found in Appendix D, the Nuon Solar Team requires the MPPT to be compatible with the CAN bus protocol. The CAN bus protocol is commonly used in the automotive industry for reading all the sensor values which are used in a car engine. The values the MPPT should send to the operating system of the NUNA can be found in Appendix D.

3.3 Design Process

We divided the process of the design of our MPPT into three parts. At first, we subdivided the system into two main parts, the hardware and the software part. As we were responsible for the software part, subsequently, we did a literature survey to find out which algorithms were available to track the MPP and what the advantage and disadvantage of each respective algorithm was. Then, we simulated the different algorithms for their respective efficiencies. Based on these simulation results, we concluded which algorithm would perform best. Subsequently, we implemented all the simulated algorithms onto a microcontroller and tested them in practice to verify the results of the simulation. Based on this experimental evaluation, we chose the best performing algorithm to be implemented into our MPPT design.

To be able to keep track of the progress, we executed different steps, namely the literature survey, the simulation and the experimental evaluation. We made several deadlines and made a planning in the form of a Gantt-chart. This chart can be seen in figure 3.5. It should be noted that the chapters found under the heading 'Writing' have changed during the design process.

Week numbers (separated in half weeks)	17.1	17.2	18.1	18.2	19.1	19.2	20.1	20.2	21.1	21.2	22.1	22.2	23.1
Practical Work													
Program Models in Matlab/Simulink													
Simulating													
Program Models in C													
Preparations for Experimental Evaluation													
Measurements													
Writing													
Literature Study (Stefan + Anton)													
Preface (Anton) / Design (Stefan + Anton)													
Microcontroller (Stefan)													
Simulation (Stefan)													
Experimental Evaluation (Anton)													
Algorithm Comparison (Stefan + Anton)													
Evaluation (Anton) / Conclusion (Stefan)													

Figure 3.5: Planning of the design of the MPPT controller

Chapter 4

State of the Art MPPT Algorithms

An efficient tracker algorithm is needed for the MPPT to obtain the maximum power from the solar panels. A tracker algorithm tries to track the maximum power point of the solar panel as best as possible, i.e. as fast and reliably as possible.

Over the years, several tracker algorithms have been developed, each with his own advantages and disadvantages. In this chapter, we will discuss the most used and some promising algorithms, each section will cover a different algorithm, namely Hill Climbing Algorithms in Section 4.1, Fuzzy Logic Control in Section 4.2, the Current Sweep method in Section 4.3 and the DIRECT method in Section 4.4, Section 4.5 is a summary of all the considered algorithms. The goal of the chapter is to perform a literature review of the existing MPPT algorithms in order to be able to evaluate them and choose the most suitable one for our application.

The reported efficiencies of the algorithms discussed, are defined in equation 4.1, where P_{sim} is the power obtained from the solar panel when the simulated algorithm is used (excluding losses of the MPPT itself) and P_{max} is the maximum power that theoretically can be obtained from the solar panel under the given operating conditions.

$$\eta = \frac{P_{act}}{P_{max}} * 100\% \quad (4.1)$$

4.1 Hill Climbing Algorithms

The most common types of algorithms are Hill Climbing algorithms. Hill Climbing means that the algorithm takes steps over the p-v characteristic to find the MPP. Of this type, there are three algorithms which are very common, Perturb & Observe (4.1.1), $\frac{dP}{dV}$ Feedback Control (4.1.2) and Incremental Conductance

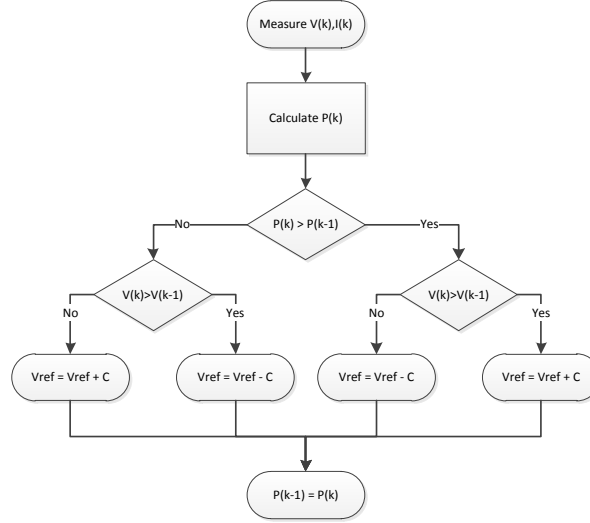


Figure 4.1: Flowchart of the P&O algorithm

(4.1.3). In 4.1.4, we will discuss the drawbacks of these types of algorithms and in 4.1.5 we will discuss a method to overcome some of these drawbacks.

4.1.1 Perturb & Observe

The most widely used algorithm is the Perturb & Observe (P&O) algorithm. The P&O algorithm perturbs the duty cycle which controls the power converter, in this way it takes steps over the p-v characteristic to find the MPP. This perturbation causes a new operating point with a different output power [5],[2],[6],[7],[8],[3]. In case this output power is larger than the previous output power, this point is set as the new operating point. In case it is lower, the same power point is adjusted to a lower or higher working voltage, depending on the previous step direction. A flowchart of the P&O algorithm is found in figure 4.1. In several studies, it has been shown that P&O has led to efficiencies as high as 96.5% in [8], and 99.5% in [3].

4.1.2 $\frac{dP}{dV}$ Feedback Control

The $\frac{dP}{dV}$ algorithm is based on the derivative of the power with respect to the voltage. Both the current and the voltage are measured, from which the power and its derivative is calculated. The algorithm is based on the fact that the

derivative equals zero at the MPP, see equation 4.2 [7].

$$\begin{aligned} \frac{dP}{dV} &= 0 && \text{at MPP} \\ \frac{dP}{dV} &> 0 && \text{left of MPP} \\ \frac{dP}{dV} &< 0 && \text{right of MPP} \end{aligned} \quad (4.2)$$

Based on the equations described in 4.2, the width of the PWM signal is adjusted, such that the operating point moves towards the MPP.

4.1.3 Incremental Conductance

Incremental Conductance (InCond) is a more elaborate version of the dP/dV algorithm. Because the power equals $V \cdot I$, the derivative of power with respect to voltage equals [5],[6],[7],[8],[3]:

$$\frac{dP}{dV} = V \frac{dI}{dV} + I \frac{dV}{dV} = V \frac{dI}{dV} + I \quad (4.3)$$

Combining 4.3 with the set of equations given in 4.2, the following set of rules is found:

$$\begin{aligned} \frac{dI}{dV} &= -\frac{I}{V} && \text{at MPP} \\ \frac{dI}{dV} &> -\frac{I}{V} && \text{left of MPP} \\ \frac{dI}{dV} &< -\frac{I}{V} && \text{right of MPP} \end{aligned} \quad (4.4)$$

With the help of these equations, the next operating point is chosen. A flowchart of the InCond algorithm is found in figure 4.2.

A PI controller is an effective way of implementing the InCond algorithm [1]. As an input for this controller, an error signal is constructed, see formula 4.5.

$$e = \frac{dI}{dV} + \frac{I}{V} \quad (4.5)$$

The PI controller makes this error signal go to zero, which means the operating point is at the MPP. The efficiency of this algorithm is higher than P&O, [8] states an efficiency of 98.5% in simulation and 98.2% in measurements.

4.1.4 Drawbacks of the Hill Climbing Algorithms

The three algorithms mentioned above have certain, similar drawbacks. First, all of them can not distinguish a local maximum from a global maximum. In case of shaded conditions, these local maxima do occur in the p-v characteristic of a solar panel and operating at a local maximum could mean reduced power output, as is shown in [1].

Secondly, all algorithms mentioned have irregular behavior in case of rapidly changing irradiation conditions [3],[9].

In both these cases, it is possible the algorithm loses track of the MPP and reduced power output will occur.

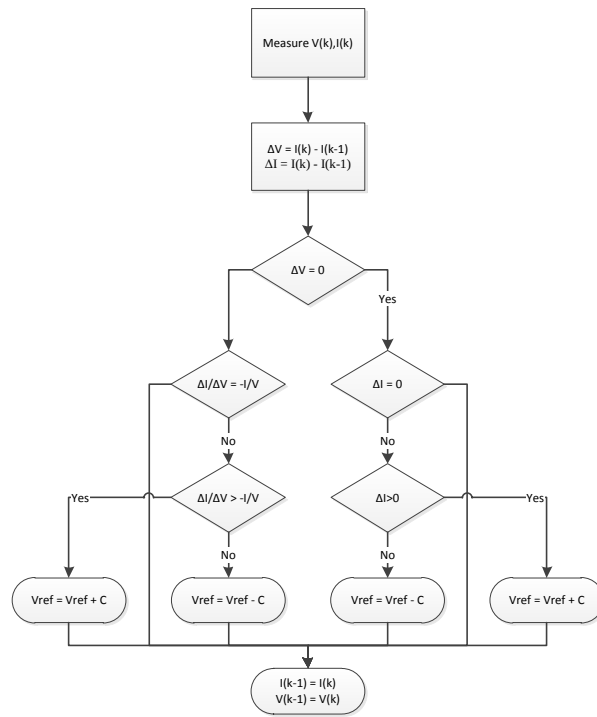


Figure 4.2: Flowchart of the InCond algorithm

4.1.5 Making the algorithms adaptive

All the hill climbing algorithms previously discussed, can be made adaptive by adding some extra rules. There are multiple ways of doing this. In [3], InCond is made adaptive by looking at the value of dI . The following set of rules is used:

$$\begin{array}{llll}
 dI & \text{Oscillating} & \rightarrow & \Delta V_{ref} = 0.1V \\
 dI & < 0.001A & \rightarrow & \Delta V_{ref} = 0.05V \\
 dI & < 0.005A & \rightarrow & \Delta V_{ref} = 0.2V \\
 dI & < 0.01A & \rightarrow & \Delta V_{ref} = 0.5V \\
 dI & < 0.015A & \rightarrow & \Delta V_{ref} = 1.5V \\
 & \text{Otherwise} & \rightarrow & \Delta V_{ref} = 3.0V
 \end{array} \tag{4.6}$$

The set of rules in 4.6 state that the higher the change in current, the larger the stepsize should be. The drawback of these rules is that it is not generic, the values that are used have to be changed each time another solar panel is used.

$\frac{dP}{dV}$ feedback control can also be made adaptive as noted in [10]. It remembers the previous value of $\frac{dP}{dV}$ and subsequently, determines the new stepsize according to the sign of the product of the new and previous value of $\frac{dP}{dV}$.

When the product is negative, the operating point oscillates around the maximum of the p-v characteristic, because $\frac{dP}{dV}$ differs from sign on each side of the maximum of the p-v characteristic.

When the product is positive, we look at the sign of the current derivative to determine whether the operating point is located to the right or to the left of the MPP.

Subsequently, we take the difference between the current and previous value of $\frac{dP}{dV}$. As the p-v characteristics can be considered linear far from the MPP, the difference between the current and previous value of $\frac{dP}{dV}$ will be small and the stepsize should be large. The value of $\frac{dP}{dV}$ changes much near the MPP, so the difference between the current and previous value of $\frac{dP}{dV}$ will be large and the stepsize should be small.

4.2 Fuzzy Logic Control

Another interesting tracker algorithm is fuzzy logic. Fuzzy logic control uses member functions and a rule table to decide the next operating point. The decisions are based on an error function, defined as follows[7], [11],[12],[3]:

$$E(k) = \frac{P(k) - P(k-1)}{V(k) - V(k-1)} = \frac{\Delta I}{\Delta V} + \frac{I}{V} \tag{4.7}$$

The change in the error function equals:

$$\Delta E(k) = E(k) - E(k-1) \tag{4.8}$$

After these values are calculated, they are classified into one of five or more classes: Negative Big (NB), Negative Small (NS), Zero (ZE), Positive Small

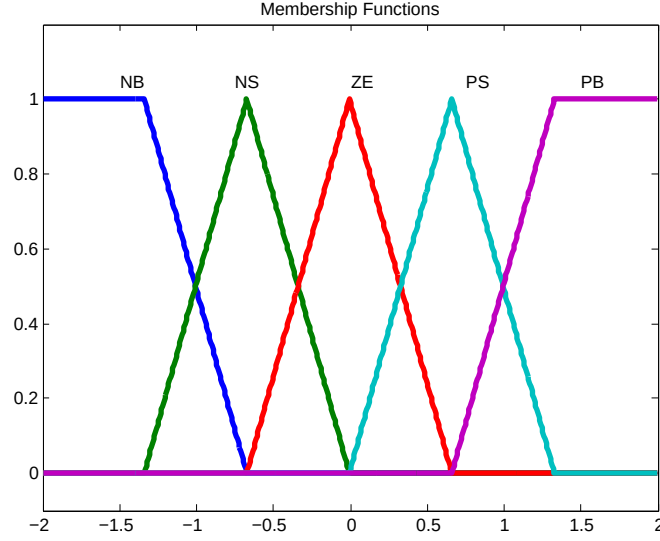


Figure 4.3: Example of membership functions [7]

(PS) and Positive Big (PB). This is done by using membership functions, which are depicted in figure 4.3.

Table 4.1: Fuzzy Logic Control Rule Table [7]

E\de	NB	NS	ZE	PS	PB
NB	NB	NS	ZE	PS	PB
NS	NB	NS	ZE	PS	PB
ZE	NB	NS	ZE	PS	PB
PS	NB	NS	ZE	PS	PB
PB	NB	NS	ZE	PS	PB

After the classification of the values, a decision is made based on a rule table, an example is shown in table 4.1. After this decision, defuzzification occurs and the new operating point is calculated. According to [12], [3], good design practice and information about specifics of the solar panel will make fuzzy logic control faster and more efficient. According to [3], an efficiency of 99.5% can be achieved.

The main drawback of this method is the fact that the algorithm needs to be tuned. The member functions are defined according to parameters from the solar panel and the rest of the system. A lot of knowledge about the system is needed to achieve high efficiencies. This problem is solved by [13] by using adaptive fuzzy logic control. However, the complexity of fuzzy logic control, especially the defuzzification stage, is very high, and adaptive fuzzy logic control has an

even higher complexity.

4.3 Current Sweep Method

Another method to obtain the MPPT from a solar panel is the current sweep method. The current sweep method obtains the I-V characteristic of the solar panel at its current operating condition[14],[7],[3]. The method manipulates the solar panel current as a predetermined function of time:

$$i(t) = f(t) \quad (4.9)$$

Subsequently, the solar panel voltage is measured and the instantaneous power is calculated by the following formula:

$$p(t) = v(t) * i(t) = v(t) * f(t) \quad (4.10)$$

Maximum power will be obtained when the derivative of $p(t)$ with respect to t equals zero, e.g.:

$$\frac{dp(t)}{dt} = v(t) * \frac{df(t)}{dt} + f(t) * \frac{dv(t)}{dt} = 0 \quad (4.11)$$

We can choose $f(t)$ equal to $k * df/dt$, where k is a real constant and $df(t)/dt$ does not equal zero in the range of the current sweep waveform. Equation 4.11 then simplifies to:

$$\frac{dp(t)}{dt} = k * \frac{dv(t)}{dt} + v(t) = 0 \quad (4.12)$$

This implies that we can determine the maximum power point by solving the relatively easy equation 4.12. Another advantage of equation 4.12 is that it is expressed in the voltage of the maximum power point and we do not need to worry about the current at the maximum power point. The solution of the differential equation for $f(t)$ is unique and equals to:

$$f(t) = c * e^{\frac{t}{k}} \quad (4.13)$$

If we take c as the maximum solar panel current and take k as a negative real number, then $f(t)$ equals to the formula describing the discharging of a capacitor through a resistor. Thus, we can generate our desired current sweep waveform by discharging a capacitor.

The current sweep method manipulates the solar panel current, so during the current sweep there will be reduced power output. The current sweep determines the i-v characteristic of the solar panel and the maximum power point voltage is determined. The controller holds this computed voltage as the operating voltage of the solar panel until the next current sweep determines a new maximum power point voltage. So, the current sweep is not performed continuously, but only periodical. It only makes sense to perform a current sweep if the increase in generated power is greater than the loss of power by performing the current sweep[14]. This is the main consideration for determining the period of the current sweep method.

4.4 DIRECT Method

The DIRECT algorithm is an algorithm that can be used to calculate the global minimum of a Lipschitzian function[15]. A Lipschitzian function $f(t)$ satisfies the following condition:

$$|f(x) - f(x')| \geq K|x - x'| \quad \forall x, x' \in [l, u] \quad (4.14)$$

where K is a positive constant and $[l, u]$ is a closed interval where $f(x)$ is defined. All functions with a bounded first derivative satisfy this condition, so we can also use the DIRECT algorithm to calculate the global maximum of a function, because calculating the global maximum of the function $f(x)$ is the same as calculating the global minimum of the function $-f(x)$, which will also have a bounded first derivative if the first derivative of $f(x)$ is bounded. [16] proves that the p-v characteristic of a solar panel satisfies this condition and thus we can use the DIRECT algorithm to calculate the global maximum of the p-v characteristic. The main advantage of the DIRECT algorithm is that it can calculate the global maximum of the p-v characteristic, where most algorithms discussed, like P&O and Incremental Conductance, can only calculate a local maximum.

The DIRECT algorithm works as follows:

1. Define the center of the closed interval $[l, u]$ and call it $c1$, and then calculate the value of the function $f(x)$ at this point and store it as $fmin$.
2. Divide the closed interval in three equal parts. This will ensure that $c1$ stays at the center of the in-between interval. Calculate the centers of the other two intervals and calculate the value of $f(x)$ at those points. If the lowest of those calculated values is lower than $fmin$, store that value as $fmin$.
3. Identify the set of potentially optimal intervals. An interval is potentially optimal if its center lies on the lower right of the convex hull of the centers of all intervals.
4. Divide all the potentially optimal intervals in three equal parts. This will ensure that the center of the potential optimal interval stays the center of the in-between interval. Calculate the centers of the other two intervals and calculate the value of at those points. If the lowest of those calculated values is lower than $fmin$, store that value as $fmin$.
5. Check how many times you have identified the set of potentially optimal intervals. If this value equals a predetermined iteration limit, stop the algorithm and $fmin$ will be the minimum of the function. If the value is less than the predetermined iteration limit, go back to step 3.

So we can use the DIRECT algorithm to track the global maximum of the p-v characteristic of the solar panel and thus obtain the maximum power generated by the solar panels. The DIRECT algorithm also has a fast tracking speed and

a better performance compared to P&O and InCond, especially in the presence of sudden high level changings of irradiance[16].

4.5 Summary

There are a wide variety of algorithms available to track the MPPT of a solar panel. We discussed the frequently used and most promising ones.

The hill climbing algorithms are algorithms that take steps over the p-v characteristic to find the MPP. There are many different hill climbing algorithms. We discussed the P&O algorithm, the dP/dV feedback control algorithm and the InCond algorithm. All hill climbing algorithms have the drawback that they cannot distinguish a local maximum from a global maximum and that they cannot track the MPP in rapidly changing irradiation conditions. An advantage of the hill climbing algorithms is that their complexity is low and their efficiency is comparable to other MPPT tracking algorithms. By making the hill climbing algorithms adaptive, we increase their convergence speed and reduce oscillations around the MPP, thus increasing their efficiency.

Fuzzy logic control uses member functions and a rule table to decide the next operating point. The drawback of fuzzy logic control is that it needs to be tuned and a lot of knowledge about the system is needed to achieve high efficiencies. This drawback can be overcome by making use of adaptive fuzzy logic control. However, this comes at the cost of a very high complexity. An advantage of fuzzy logic control is that slightly higher efficiencies than hill climbing algorithms can be achieved, however, this comes at the cost of a higher complexity.

The current sweep method obtains the I-V characteristic of the solar panel at its current operating condition. Based on this I-V characteristic, it can find the MPP. A disadvantage of this method is that during the current sweep, there will be reduced power output. Furthermore, the complexity is high and convergence speed is slow[3]. An advantage of the method is that it can locate a global maximum. For these reasons, the current sweep method should not be used to track the MPP continuously, but should be used as a complement to other methods.

The DIRECT method can be used to track the global maximum of a p-v characteristic. It has a fast tracking speed and better performance compared to P&O and InCond, especially in the presence of sudden high level changing of irradiance. However, it has a high implementation complexity, and therefore we chose not to pursue the implementation of the DIRECT method.

Chapter 5

Simulation of the MPPT Algorithms Performance

In order to get an indication of which algorithms work best, we have to simulate the different algorithms. First, we have to build an effective simulation model. We have built our simulation model in Simulink and we discuss this model in Section 5.1. Secondly, we discuss our simulation results in Section 5.2 and in Section 5.3 we make a conclusion based on the simulation results. It should be noted that the algorithms based on fuzzy logic control and current sweep are omitted from the results, as we did not succeed in obtaining good results with this model. Therefore, we chose not to pursue the implementation of the fuzzy logic control and current sweep algorithms.

5.1 Simulation Model

In this section, first, we discuss the ideal model which our simulation model is based upon (5.1.1). Secondly, we discuss how we implemented the different algorithms in this simulation model (5.1.2) and the simulation conditions (5.1.3).

5.1.1 Ideal Model

For the simulation, a model without a solar panel or boost converter is used, because we do not want to take into account the losses and inefficiencies of the boost converter. Furthermore, if we would have used a solar panel as a source for our model, it would have been nearly impossible to calculate the 'real' MPP for varying irradiance, so we would not have a reference MPP. Therefore, we chose to use a simple DC voltage source with two resistors in series, both of them variable. The variable resistors emulate the varying internal resistance and the configurable DC-DC converter resistance. With this model, shown in figure 5.1, we can simulate the changes of irradiance on the solar panel. The higher the

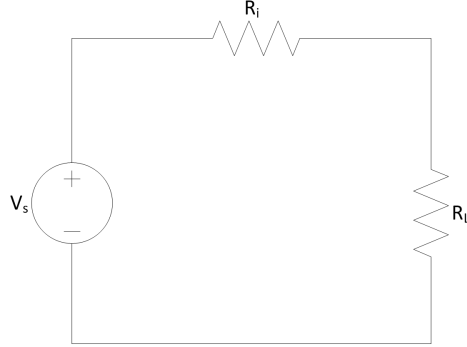


Figure 5.1: Schematic drawing of the simulation circuit

irradiance, the lower the internal impedance of the solar panel. Therefore, we can simulate the irradiance by adjusting R_i .

In our model, it is easy to calculate the ideal maximum power output, which is necessary for the calculation of the efficiency. In figure 5.1, a schematic drawing of the circuit is shown. From the maximum power transfer theorem, it follows that R_i should equal R_L to obtain the maximum power transfer [17]. From Kirchoff's voltage law, it follows that when R_i equals R_L , the voltage V_{out} equals half the open circuit voltage V_{oc} . Now that we have obtained an ideal operating point and resistance, we can calculate the maximum possible power transfer:

$$P_{ideal} = \frac{V_{ideal}^2}{R_i} = \frac{\left(\frac{V_{oc}}{2}\right)^2}{R_i} = \frac{V_{oc}^2}{4R_i} \quad (5.1)$$

With 5.1, we can calculate the MPP in Simulink, since it is independent of R_L and V_{oc} is a predefined value. Because the internal resistance R_i is user-defined, the efficiency can be calculated for a varying internal resistance.

5.1.2 MPPT Algorithm Model

The model of the MPPT algorithm is embedded in a triggered subsystem. We have chosen for a triggered subsystem to simulate the update frequency of the microcontroller. This frequency determines how often the duty cycle is updated. The subsystem consists of three parts: the delays, the algorithm and the integrator. The delays are used to obtain the previous values of the voltage and current, which are used by both the InCond and the P&O algorithms. The algorithm itself is programmed in an Embedded System Function. This embedded system has as an output a control signal which indicates by how much the value of the resistance should be changed. Subsequently, the integrator integrates the control signal and thus, a value for the R_L is obtained. Furthermore, the initial

value of the resistor is set at the ideal value. The code used to run the simulation can be found in Appendix A, the block scheme of this simulation model, made in Simulink, can be found in Appendix B.

5.1.3 Simulation Conditions

The MPPT's performance is simulated for multiple conditions: steady state, increasing/decreasing the resistance and a step increase/decrease of the resistance. This subsection describes the different conditions and explains why they are used.

Steady State Analysis The steady state analysis is used to simulate the performance of the MPPT when the sky is clear and no shading is present. In this condition, the actual output power will oscillate around the MPP. The amplitude of this oscillation will determine the efficiency of the algorithm, the lower the amplitude the better. This condition is simulated by keeping R_i in figure 5.1 at a constant value.

Increasing/Decreasing Irradiance This condition simulates changing irradiance. This will happen when it is partly clouded and the amount of sunlight falling on the solar panel increases or decreases over time, or when the temperature of the solar panel changes. The performance of the different algorithms will be determined by how fast the algorithms can change the operating point of the solar panel. They should be able to track the MPP, even if the rate of change is high. This condition is simulated by increasing or decreasing R_i in figure 5.1.

Sudden shading/exposure of the solar panel This condition is used to simulate a sudden shading or a sudden exposure to sunlight of the solar panel. This could happen when objects, like buildings or trees standing next to the road, are blocking the sun. The performance will be determined by how fast the operating point of the solar panel is adjusted to the new MPP. This condition is simulated by setting the value of R_i in figure 5.1 with a step function, a step increase for a sudden shading and a step decrease for a sudden exposure of the solar panel.

The efficiencies of the different algorithms are calculated as follows:

$$\eta = \frac{P_{act}}{P_{max}} * 100\% \quad (5.2)$$

5.2 Simulation Results

In this section, we discuss the simulation results for the different simulation conditions.

5.2.1 Steady State Analysis

As can be seen from table 5.1, the algorithms perform almost exactly the same. The adaptive algorithms seem to perform a little bit less, this is probably because they need a little more calculations, making the algorithms a little bit slower. Because of this, the output voltage is updated later, and hence the efficiency is a little lower. However, the differences are negligible.

Table 5.1: Steady State Simulation - Efficiency in %

Algorithm	Stepsize				
	0.005	0.01	0.05	0.1	0.2
P&O	>99.99	>99.99	>99.99	99.96	99.87
InCond	>99.99	>99.99	>99.99	99.97	99.86
Adaptive P&O	>99.99	>99.99	99.98	99.91	99.66
Adaptive InCond	>99.99	>99.99	99.98	99.91	99.86

5.2.2 Increasing/Decreasing Irradiance

Increasing Irradiance From simulating an increasing irradiance, it follows that the best performing algorithms are the adaptive algorithms. The adaptive algorithms perform better when the slope of the irradiance is steeper. This is expected, as the adaptive algorithms can adjust their operating point faster than the other algorithms. The results of the simulation are shown in tables 5.2 - 5.5.

Table 5.2: Increasing Irradiance - P&O Efficiency in %

Slope [Ω/s]	Stepsize				
	0.005	0.01	0.05	0.1	0.2
1	99.67	99.69	99.93	99.58	97.64
5	95.36	95.42	99.45	99.75	99.12
10	97.87	88.96	97.73	99.43	99.47
20	89.95	96.25	92.80	97.87	99.33
30	82.21	90.18	87.15	95.34	98.78

Decreasing Irradiance In the case of decreasing irradiance, we see that the efficiencies of all algorithms are lower than in the case of increasing irradiance. It is notable that the P&O and InCond algorithms have higher efficiencies than their adaptive counterparts. The results of the simulation are shown in tables 5.6 - 5.9.

Table 5.3: Increasing Irradiance - InCond Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	99.64	99.62	99.87	99.73	99.19
5	95.24	95.18	99.11	99.41	99.13
10	97.76	88.60	97.08	99.00	98.90
20	89.68	95.99	91.98	96.90	98.56
30	81.88	89.76	86.28	94.47	97.89

Table 5.4: Increasing Irradiance - Adaptive P&O Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	>99.99	>99.99	>99.99	99.97	99.89
5	99.82	99.82	99.86	99.92	99.94
10	99.72	99.38	99.45	99.67	99.91
20	99.44	99.16	98.13	98.90	99.70
30	99.00	98.87	96.46	97.72	99.25

Table 5.5: Increasing Irradiance - Adaptive InCond Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	>99.99	>99.99	99.98	99.97	99.90
5	99.81	99.81	99.79	99.86	99.93
10	99.71	99.34	99.32	99.62	99.81
20	99.42	99.14	97.90	98.92	99.50
30	98.99	98.84	96.15	97.96	99.13

Table 5.6: Decreasing Irradiance - P&O Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	99.75	99.37	99.97	99.97	99.94
5	99.22	98.45	99.54	99.82	99.85
10	97.96	96.80	97.42	99.43	99.70
20	91.88	89.83	75.64	97.40	99.17
30	72.86	70.12	54.91	91.42	97.51

5.2.3 Sudden shading/exposure of the solar panel

Sudden shading When simulating a sudden shading of the solar panel, the adaptive algorithms significantly outperform the other algorithms. This is as expected as the adaptive algorithms can adjust their operating point faster

Table 5.7: Decreasing Irradiance - InCond Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	99.75	99.38	99.98	99.98	99.97
5	99.22	98.45	99.56	99.86	99.91
10	97.96	96.80	97.60	99.52	99.78
20	91.88	89.83	75.64	97.55	99.34
30	72.86	70.12	54.91	91.42	97.97

Table 5.8: Decreasing Irradiance - Adaptive P&O Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	99.96	99.97	99.95	99.89	99.53
5	95.80	99.78	99.77	99.74	99.45
10	91.02	90.24	99.61	99.37	99.01
20	77.79	76.09	98.79	98.77	97.58
30	54.63	52.79	93.60	93.33	88.82

Table 5.9: Decreasing Irradiance - Adaptive InCond Efficiency in %

	Stepsize				
Slope [Ω/s]	0.005	0.01	0.05	0.1	0.2
1	99.97	99.98	99.97	99.97	99.88
5	96.06	99.83	99.88	99.87	99.71
10	91.12	90.68	99.74	99.72	99.63
20	78.26	76.82	99.35	99.27	98.94
30	54.80	53.48	97.09	97.87	94.69

than the other algorithms. P&O and InCond perform almost equally. Also, the adaptive algorithms perform mutually the same. The results of the simulation are shown in table 5.10.

Table 5.10: Sudden Shading Simulation - Efficiency in %

	Stepsize				
Algorithm	0.005	0.01	0.05	0.1	0.2
P&O	96.98	97.85	99.29	99.45	99.23
InCond	96.98	97.85	99.29	99.68	99.15
Adapt P&O	99.33	99.56	99.58	99.45	98.63
Adaptive InCond	99.22	99.48	99.55	99.77	98.00

Sudden Exposure In the case of a sudden exposure to the sunlight of the solar panel, the adaptive algorithms perform better than the other two algorithms. Only when the stepsize becomes too large, the adaptive P&O algorithm's efficiency collapses. Therefore, the clear winner in this simulation is the adaptive InCond algorithm. P&O and InCond perform equally well. The results of this simulation are shown in table 5.11.

Table 5.11: Sudden Exposure Simulation - Efficiency in %

Algorithm	Stepsize [Ω/s]				
	0.005	0.01	0.05	0.1	0.2
P&O	93.13	94.24	98.04	98.58	98.54
InCond	93.13	94.24	98.41	99.00	98.60
Adaptive P&O	97.76	98.27	98.19	97.72	86.75
Adaptive InCond	97.79	98.33	98.85	99.26	98.36

5.3 Conclusion

First, we conclude that the difference in performance between P&O and InCond is negligible, except in the case of steady state. This is as expected as the P&O algorithm keeps searching for a better operating point, which makes it oscillate around the MPP. On the other hand, InCond stops searching when it has reached the MPP and only starts changing its operating point when the input voltage or current is changed. Therefore, overall, the InCond algorithm performs better than the P&O algorithm.

Secondly, we conclude that the adaptive algorithms in general perform better than their non-adaptive counterparts, except in the case of Decreasing Irradiance. The adaptive InCond algorithm performs better than the adaptive P&O algorithm in the case of decreasing step, and therefore is preferred above the adaptive P&O algorithm, based on our simulations.

Finally, it should be noted that the absolute values of the results of our simulations are not comparable to the values given in Chapter 4, as those simulations used other models for their simulation than the one we used. However, all simulations give an indication of the differences between the efficiencies of the different algorithms.

Chapter 6

Implementation Platform

The operations of the MPPT algorithm will be executed on a microcontroller. The microcontroller is preferred over other platforms, such as FPGAs, ASICs or DSPs, because of the peripherals often present on microcontrollers. First, we will discuss the features a microcontroller needs to have to be suited for our application in Section 6.1. Secondly, we will compare some different microcontrollers in Section 6.2 and finally, we will make a choice for a specific microcontroller in Section 6.3.

6.1 Requirements

Our MPPT controller needs some essential features, namely accurate measurements need to be performed (6.1.1), the DC-DC converter needs to be controlled (6.1.2), everything needs to be timed correctly (6.1.3) and the microcontroller needs to be able to communicate with the operating system of the NUNA by making use of the CAN bus protocol (6.1.4). We will discuss all those essential features in the subsections below.

6.1.1 Analog to Digital Conversion

A good Analog to Digital Converter (ADC) is necessary for accurate measurements, the most important specification of an ADC is the resolution. The resolution is determined by the length of the register in which the measured value is stored. The most common register lengths are 8 and 10 bits [18],[19]. The voltage range that needs to be measured is between 0 and 120 Volts and with register lengths of 8 and 10 bits, this leads to a resolution of 469 mV and 117 mV, respectively. The current range that needs to be measured is between 0 and 3.33 Ampère. This leads to a resolution of 13.0 mA and 3.25 mA, respectively. Both these resolutions will suffice for our MPPT controller.

Another requirement is that at least two ADCs are present on the microcontroller, because both input voltage and current need to be measured. However,

in the future the output current and voltage need also to be determined, and therefore another two ADCs is an advantage.

6.1.2 PWM Generator

The DC-DC converter is set at a certain operating point with help of a Pulse Width Modulation (PWM) signal[4]. The duty cycle of this PWM signal determines the operating point of the converter. Therefore, a PWM generator is an essential feature to be present on the microcontroller. The DC/DC converter needs a specific PWM frequency to operate correctly. Furthermore it is desired that the resolution of the PWM signal is as high as possible. The accuracy is determined by the value in the compare register and the clock frequency. How these values are related is discussed in Chapter 7.

6.1.3 Operating Frequency

The microcontroller needs a clock frequency to operate correctly. The main thing the operating frequency determines is the speed at which the calculations are performed. Another important thing the clock frequency dictates is the resolution of the PWM signal. How these two values are related is discussed in Chapter 7, but it is sufficient to know that the resolution of the PWM signal will increase when the clock frequency increases.

6.1.4 Communication

As CAN bus support is required to send the measured values back to the operating system of the NUNA, communication peripherals are needed. These peripherals have the entire communication protocol pre-programmed and as they are executed parallel to the main program, very little execution time is added. Some microcontrollers even have the CAN bus protocol implemented, but ISP or USART protocols will suffice, as external controllers are available, such as the MCP2515 [20].

6.2 Microcontroller Comparison

In this section, we will discuss different microcontrollers to find the one that is best suited for our application. It should be noted that only microcontrollers which are available in DIP packages are considered, as the prototype is made on a breadboard.

6.2.1 68HC11

The 68HC11 is developed by Motorola and is an 8-bit microcontroller. It was introduced in 1985 and has been used frequently in embedded systems ever since. The controller has a maximum clock frequency of 3 MHz, 8 ADCs with a register length of 8 bits and PWM generation is possible with 16-bit resolution.

The generation of the PWM signal has to be done in the program code itself as no separate PWM generator is present[19].

6.2.2 8051

A comparable microcontroller is the 8051. The 8051 is a 16-bit microcontroller designed by Intel in the 1980s. This microcontroller has been used a lot in amateur projects and therefore, a large quantity of information about its applications is available online. The 8051 has a maximum clock frequency of 12 MHz, 8 ADCs are available with a register length of 8 bits and a PWM signal can be generated with a register of 16 bits. Just like the 68HC11, the generation of this PWM signal has to be done in the code[21].

6.2.3 Atmega168/Pic16F87X

The Atmega168 is an 8-bit microcontroller developed by Atmel in 1996. It is very popular among amateur programmers and therefore, a lot of information about it is available. The same applies to the PIC16F87X series, which has been introduced by Microchip Technologies in 1993. Both microcontrollers have the same most important features needed for our application. Both have a maximum clock frequency of 20 MHz, eight 10-bit ADC channels are available and both have a separate PWM generator with a resolution of 16 bits[18],[22].

6.3 Conclusion

Based on the specifications of the different microcontrollers listed in table 6.1, we chose to use the Atmega168 microcontroller. This controller has a higher clock frequency than both the 8051 and 68HC11, which makes it more suitable. The Atmega168 is preferred over the PIC16F87X because of the available material and knowledge about the Atmega168.

It should be noted that better candidates are available, such as the at90can series or the xmega32 series, which have built in CAN bus support and can run at higher frequencies. However, these microcontrollers come only in quad flat packages, which can not be used on a breadboard.

Table 6.1: Comparison of the different microcontrollers [19],[21],[18],[22]

Microcontroller	68HC11	8051	Atmega168	PIC16F87X
Clock Frequency	3 MHz	0-12 MHz	0-20 MHz	0-20 MHz
No. of ADCs	8	8	8	8
ADC Register Length	8	8	10	10
PWM Register Length	16	16	16	16
Communication	USART, SPI	USART	USART, SPI, I ² C	USART, SPI, I ² C
Programming Language	Asm,C	Asm,C	Asm,C	Asm,C

Chapter 7

Implementation of the MPPT Controller

In this chapter, we will explain how we implemented the different algorithms onto the microcontroller. First, we explain how we configured the microcontroller to make it suited for our application (Section 7.1). Secondly, we explain how we implemented the different algorithms onto the microcontroller (Section 7.2). The code of the implementation and configuration of the algorithms and explanation thereof, can be found in Appendix C.2. Lastly, we explain how we implemented the CAN bus controller (Section 7.3).

7.1 Configuration of the microcontroller

In this section, we explain how we configured the microcontroller to make it suited for our application. First, we describe how we configured the PWM generator (7.1.1). Secondly, we explain how we configured the ADCs (7.1.2) and how we handled the timing (7.1.3).

The formulas and calculations given in this section, can be found in [18]. An block scheme of the microcontroller can be found in figure 7.1.

7.1.1 PWM Generator

The Atmega168 supports three types of PWM generators: Fast PWM, 'Phase Correct PWM' and 'Phase and Frequency Correct PWM'. The two 'correct' PWM generators are useful when multiple PWM signals are generated. As we only use one PWM signal, the simpler Fast PWM mode can be used.

The PWM signal is generated by a counter. The value of this counter is constantly compared to a certain value, and when the counter equals this value, the PWM signal is set to 0. After that, when the counter reaches its top value and overflows, the PWM signal is set to 1. Obviously, the compare value should never be higher than the top value, because then the PWM signal will always

be 1. The PWM signal is characterized by the two values mentioned above: the compare value and the top value. The top value is the most important value, it determines both the resolution and the frequency of the PWM signal. The frequency of the PWM signal is calculated with 7.1.

$$f_{PWM} = \frac{f_{clk}}{Top Value + 1} \quad (7.1)$$

The frequency of the signal should be 100 kHz, because of the design of the DC-DC Converter[4]. With the clock frequency set to the maximum of 20 MHz that leads to a top value of 199.

With this top value, the resolution of the PWM signal can be calculated. Because the compare value cannot exceed the top value, the duty cycle is distributed over the possible values of the compare value. The resolution can be calculated with 7.2.

$$R_{PWM} = \frac{1}{Top Value + 1} \quad (7.2)$$

With a top value of 199, this leads to a resolution of 0.005 or 0.5 %.

7.1.2 Analog to Digital Conversion

To measure the output voltage and current of the solar panel, two Analog to Digital converters (ADCs) are needed. The maximum value an ADC can measure is equal to the supply voltage, which is 5V in our case. As the maximum solar panel voltage is 120 V, the solar panel voltage must be scaled down with a factor of 24. The scaling is implemented outside the MPPT controller and we will not cover it in this thesis. The interested reader can read more about the scaling process of the output voltage and current in [23]. It is sufficient to know that the output voltage and current are converted to values between 0 and 5 V. These values are measured by the ADCs and converted into either an 8-bit or an 10-bit value, which can be used for calculations. The 10-bit ADCs offer a higher resolution, so we use those instead of 8-bit ADCs. With formula 7.3, this leads to a resolution of 117 mV for the voltage. The maximum output current of the solar panel will be 3.33 A. The resolution of the current is calculated in 7.4 and equals 3.25 mA.

$$R_V = \frac{V_{max}}{2^{10}} \quad (7.3)$$

$$R_I = \frac{I_{max}}{2^{10}} \quad (7.4)$$

7.1.3 Timing

It is needed to regulate the frequency at which the duty cycle of the PWM is updated, because the DC-DC converter has a transient time and if the duty

cycle is updated more often than that transient time, unwanted transient responses occur. The transient time is very small, in the order of microseconds, and therefore the duty cycle is updated every 10 milliseconds. By using this frequency, we are certain that no unwanted transients will occur.

The regulating of the update frequency is achieved by using a timer. When the counter overflows, it produces an interrupt. When this interrupt occurs, the values of the ADCs are read and the new duty cycle of the PWM signal is calculated and updated.

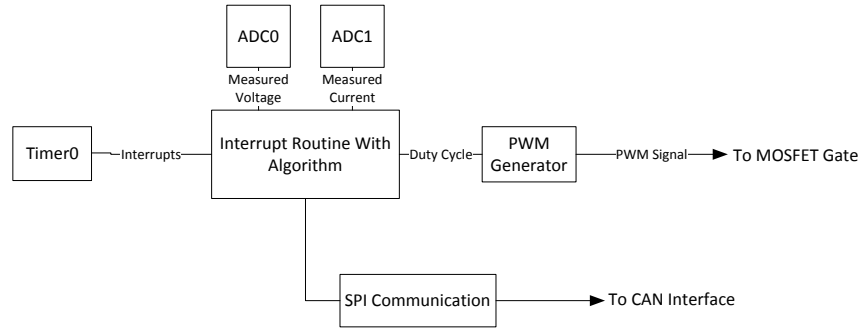


Figure 7.1: Microcontroller block scheme

7.2 Implementation of the different algorithms

In this section, we will discuss how we implemented the different tracking algorithms on the microcontroller. We implemented the P&O algorithm (7.2.1), the InCond algorithm (7.2.2) and we made both those algorithms adaptive (7.2.3). All our implemented algorithms are dependent on the measured output voltage and current of the solar panel. These values are produced by the ADCs. Also, all our implemented algorithms return an integer that determines the change of the duty cycle of the PWM signal, we call this integer the stepsize.

7.2.1 P&O

The P&O function we implemented, returns a 1 or -1 as the stepsize. Its inputs are the calculated current and previous power of the solar panel and the previous stepsize. The power of the solar panel is calculated from the measured output voltage and current of the solar panel. Based on these inputs, the function calculates the stepsize. This is done by making use of an if-statement.

7.2.2 InCond

The InCond function we implemented, returns a 1, 0 or -1 as the stepsize. Its inputs are the measured output voltage and current of the solar panel and the difference between the current and previous measured output voltage and current of the solar panel. Based on these inputs, the function calculates the stepsize. This is done by making use of an if-statement.

7.2.3 Making the Algorithms Adaptive

The algorithms are made adaptive by adding some extra code around the calling of the algorithm functions, namely the P&O function for adaptive P&O and the InCond function for adaptive Incond. In this extra code, the current and previous stepsize are buffered by saving them in an array. Subsequently, the signs of the previous stepsize and the current stepsize are compared. If the signs and thus the directions are equal, the stepsize is increased. If the signs differ, the stepsize is made equal to the value that is returned by the called algorithm function.

7.3 Implementation of the CAN Bus Protocol

The implementation of the CAN bus protocol is achieved by using an external CAN bus controller, because the Atmega168 does not have built in CAN bus support. The controller we chose, is the Microhip MCP2515. This controller converts an SPI interface into an CAN bus interface[20]. The MCP2515 is configured to send at a datarate of 500 kbits/s, as required by the Nuon Solar Team. The output of the MCP2515 varies between 0 and 5 Volts. As the CAN bus protocol requires an 12V signal, the output of the MCP2515 needs to be converted. This is done by the Microchip MCP2551[24]. This IC is an CAN bus transceiver, converting his input voltage levels to the correct voltage. The code used for the CAN bus communication can be found in Appendix C.3.

Chapter 8

Experimental Evaluation

In this chapter, we discuss our experiences with evaluating our controller experimentally. First, we verify that the software works correctly (Section 8.1). Secondly, we discuss our experimental setup (Section 8.2). Thirdly, we explain the results of our experimental evaluation (Section 8.3).

8.1 Software Verification

Before the actual MPPT efficiencies could be measured, the functions of the controller needed to be verified to be working correctly. We verified the ADCs (8.1.1), the PWM signal (8.1.2), the timer interrupt frequency (8.1.3), the algorithms (8.1.4) and the CAN bus controller (8.1.5).

8.1.1 Analog to Digital Converters

The ADCs were tested by attaching variable resistors to his input pins. With these variable resistors, it was possible to set a voltage between 0 and 5 Volts at the ADCs inputs. Subsequently, these voltage values were displayed on an LCD display. The values returned by the ADCs should be between 0 and 1023, as they have 10 bit registers. This was the case, and therefore, the ADCs were verified to be working correctly.

8.1.2 PWM Signal

Two things of the PWM signal needed to be verified, namely the frequency of the PWM signal, which should be 100 kHz, and the function that updates the duty cycle of the PWM signal.

The frequency of the PWM signal was easily verified by connecting an oscilloscope to the PWM generator output pin. This way, we measured an frequency of 100.7 kHz. The small deviation of the measured frequency to the desired frequency does not harm the system.

We tested the function that updates the duty cycle of the PWM signal by connecting a variable resistor to one of the ADCs input pins and using the generated value by the ADC to update the duty cycle. The oscilloscope showed that the duty cycle was correctly updated, with it varying between 0 and 100%.

8.1.3 Timer Interrupt Frequency

The timer interrupt frequency determines how often the duty cycle of the PWM signal is updated. We verified the timer interrupt frequency to be working correctly by toggling an output pin every time an interrupt occurred. As the interrupt frequency should be 100 Hz, the resulting frequency of the toggled signal should be 50 Hz. We measured the frequency of the toggled signal to be 50.1 Hz.

8.1.4 Algorithm Verification

We verified the P&O and InCond algorithms by connecting two variable resistors to the ADCs. These two variable resistors simulated the measured voltage and current. We only had to test these two algorithms, because their adaptive counterparts are just an expansion of them, so if P&O and InCond work correctly, so will adaptive P&O and adaptive InCond.

Subsequently, we displayed the resulting stepsizes calculated by the different algorithms, on an LCD screen. All the algorithms worked correctly, except for one thing: when the stepsize was negative, -1 for both P&O and InCond, the algorithms actually returned an value of 255. This was due to the register sizes, the algorithm initially returned a signed 8 bit value, which was then converted to an integer of 16 bits. This conversion resulted in a value of 255. By changing the return values of the algorithm functions into integers, this problem was solved. With this change to the algorithm functions, all algorithms, including the adaptive, operated as expected.

8.1.5 CAN Bus Controller

The CAN bus controller was tested with a device called the CANView USB. This device can be used to read the sent CAN bus messages from the controller on a computer, and to send messages to the connected devices [25]. After trying to send several messages to the controller via the SPI interface, and trying to send messages to the CANView USB, we found that it did not work as expected. We suspect the problem to be in the SPI communication, as the MCP2515 did not get into configuration mode and it was not able to read out registers via the SPI interface. However, since CAN bus implementation is a would be nice to have function of the microcontroller and due to time constraints, further tests were not executed. Therefore, CAN bus communication is at the moment not functional.

8.2 Experimental Setup

After all the functions of the controller were verified, we tested the complete MPPT. The MPPT was built as depicted in figure 3.1. First, we explain how we built our experimental setup(8.2.1). Secondly, we detail on the test conditions(8.2.2).

8.2.1 Experimental Hardware Setup

As can be seen from this figure, a solar simulator is used. The solar simulator simulates the sun by mimicking its light spectrum with the use of a variety of lamps. The use of a solar simulator has some advantages over the use of a DC power supply, as explained below.

A DC power supply is a regulated source, which limits current and voltage. When the voltage of the power supply is regulated, maximum power is obtained when the current is maximized. However, the current of a power supply is limited and the tracking algorithms will not function properly. As the voltage is stable, ΔV will be zero and the current limiting causes irregular current behaviour. Therefore, the algorithms on the controller will not function properly. However, a solar panel does not have these characteristics, and behaves in a way that is expected by the controller. For these reasons, a solar panel irradiated by a solar simulator was chosen as the source.

To know whether the algorithm tracks the MPP, we need to know the MPP of the used solar panel. The solar simulator has the option to measure the i-v curve of the solar panel and calculate the p-v curve and MPP from this i-v curve. The i-v curve of a solar panel is dependent on the temperature, and because the solar panel gets hot while under the solar simulator, the MPP changes constantly. Therefore, we decided to obtain two i-v curves, p-v curves and MPPs: one when the solar panel is at room temperature and one when the solar panel has been under the solar simulator for a long time. Both p-v curves are shown in figure 8.1. From these curves, it follows that the MPP should lie between 60.7 V and 72.4 V.

To verify that the controller operates between these two voltages, the voltage is measured at the output of the solar panel. Also, the PWM signal is displayed on a scope, so we are able to review the operating of the algorithm. Finally, the solar panel's output current is measured, so we are able to calculate and track the output power.

8.2.2 Test Conditions

Due to time constraints, we were not able to evaluate the different algorithms as extensively as we did the simulations. The only condition that we could test, was the steady state condition, because we only had a constant power source. However, we were able to test some properties of the different algorithms, such as convergence speed and the amount of oscillation, while we changed the update frequency of the PWM signal and the stepsize. It proved too difficult for

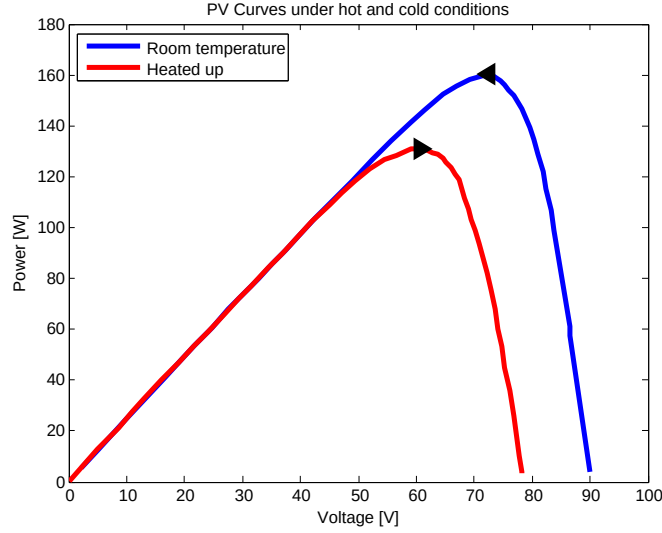


Figure 8.1: P-v characteristics of a solar panel in hot and cold conditions

us to give a reliable value for the measured efficiency of the different algorithms, this is mainly because we were too short on time.

The effects of different update frequencies and stepsizes were tested by simply changing the update frequency and stepsize values in our code. The oscillations of the different algorithms were tested by waiting for the algorithm to converge to the MPP and then evaluate the oscillation of the duty cycle on the oscilloscope. The convergence speed was tested by setting the initial duty cycle far from the MPP and subsequently, track on the scope how fast the different algorithms converged to the MPP.

8.3 Results of the Experimental Evaluation

First, we discuss the influence of the system parameters on the performance of the system(8.3.1). Secondly, we detail on the performance of the different algorithms(8.3.2).

8.3.1 Influence of the System Parameters

Stepsize The system was tested with three different stepsizes, which were acquired by trial and error: 0.005, 0.015 and 0.025. We found out that the stepsize is a critical part of the algorithm, when chosen wrong, it can make the entire controller unstable in operation.

When a stepsize of 0.005 was chosen, the difference in output voltage and current of the solar panel was too small to obtain a reading with which the mi-

controller could reliably calculate the next operating point. This led to an instability of the system, causing it to converge to a duty cycle of 0 or 1. The effects of a bigger stepsize resulted in similar behaviour as it did in the simulations. It caused the controller to oscillate around the MPP with a larger amplitude and changed the operating point of the solar panel more aggressively. However, when the stepsize became too big, the controller converged to a duty cycle of either 0 or 1. We think this is because of a flaw in the MPPT, it does not operate properly at the whole range of the duty cycle, see below.

Operating Range When the MPPT is operating at a duty cycle of less than 20% or higher than 80%, the duty cycle converged to 0% or 100%, respectively. We found out that this was due to the non-linearity of the DC-DC converter, and therefore, it is an issue of the hardware part of the MPPT, namely the boost converter. When the duty cycle is less than 20%, one step change causes a very small change in the voltage. This change in voltage is smaller than the resolution of the ADC, making the controller think it is converging in the right direction. When the duty cycle is larger than 80% and a step change is made, the change in current is very small, making the controller think the current did not change at all and he is converging in the right direction. Therefore, the MPPT will only work correctly in a duty cycle range from 20% to 80%.

Update Frequency of the PWM Signal By adjusting the update frequency of the PWM signal between 2 Hz and 100 Hz, we found out that the frequency has a major influence on the stability of the system. The lower the update frequency, the more stable the system was. This is probably due to the transients which occur when the operating point of the solar panel is changed. The maximum update frequency at which the controller operated correctly varied widely over time, we think this is because of the noise of the system.

Summary The main issue we had with our experimental setup was that the system is very sensitive to noise. As can be seen from figure 8.2, the difference between the output current we measured with the oscilloscope (green) and the output current the measuring circuit of the MPPT measured (purple) is not negligible. It should be noted that a lot of noise is present on the purple signal. This causes the read values of the ADC to be inaccurate and the outcome of the algorithms to deviate from what it should be. However, we were able to verify some results the simulations predicted. Bigger stepsizes cause larger oscillations, but also make the algorithm converge faster.

8.3.2 Algorithm Performance

By adding buffers and filters to the ADC pins, we solved the earlier mentioned problems with noise. Due to time constraints, we were not able to test the different algorithms as extensively as we wanted. However, we were able to

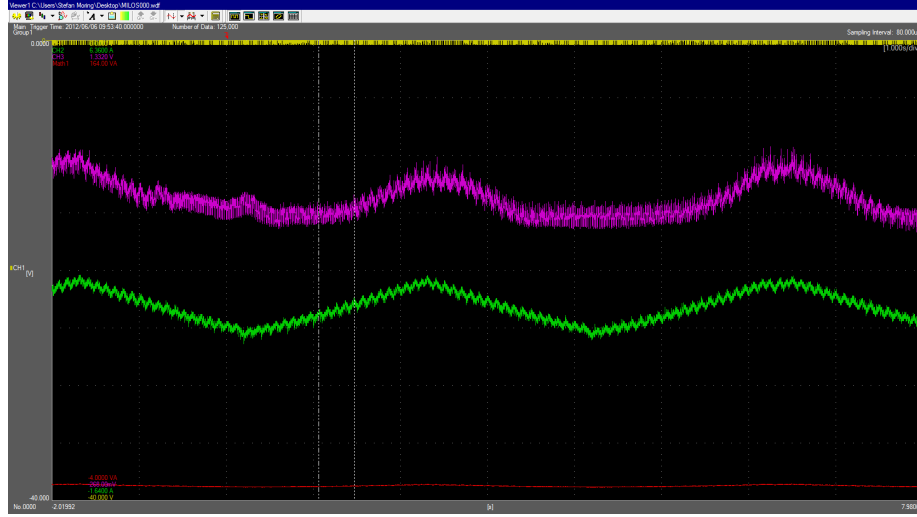


Figure 8.2: Output current measured by the oscilloscope (green) and output current measured by the measuring circuit of the MPPT (purple)

discover some differences between them. We have tested only one condition, namely sudden shading. We tested this in two ways: first by initializing the duty cycle at 70%, a point far away from the MPP, and secondly by actually shading the solar panel using a piece of cardboard. These two tests were performed with an update frequency of only 1 Hz, so we could easily track the behaviour of solely the algorithm and no other factors influenced the performance.

P&O P&O appeared to be the most stable algorithm we tested. Due to its simplicity, it tracked the MPP under most circumstances, because it is the least sensitive algorithm to noise. Even when a lot of noise was present, the algorithm was able to track the MPP, as long as the stepsize was large enough. In noisy conditions, the algorithm had the tendency to oscillate its way to the MPP and in less noisy conditions, it tracked the MPP exactly as predicted by the simulations. Figure 8.3 shows the performance of P&O after being initialized at 70%. This figure shows that the algorithm needs 24 steps to converge to the MPP. Figure 8.4 shows the result of the test with the piece of cardboard. As can be seen from this figure, the output power falls as the piece of cardboard is shoved between the solar simulator and the solar panel. The controller then recovers to the new MPP. The converging of this point takes 12 steps.

InCond The performance of the InCond algorithm was less than expected. We think this is the case, because the algorithm is more sensitive to noise. Because the algorithm is based on the derivative of the output current with respect to the output voltage, the effect of noise is larger than with P&O. Small disturbances in the current or voltage measurements can result in a large

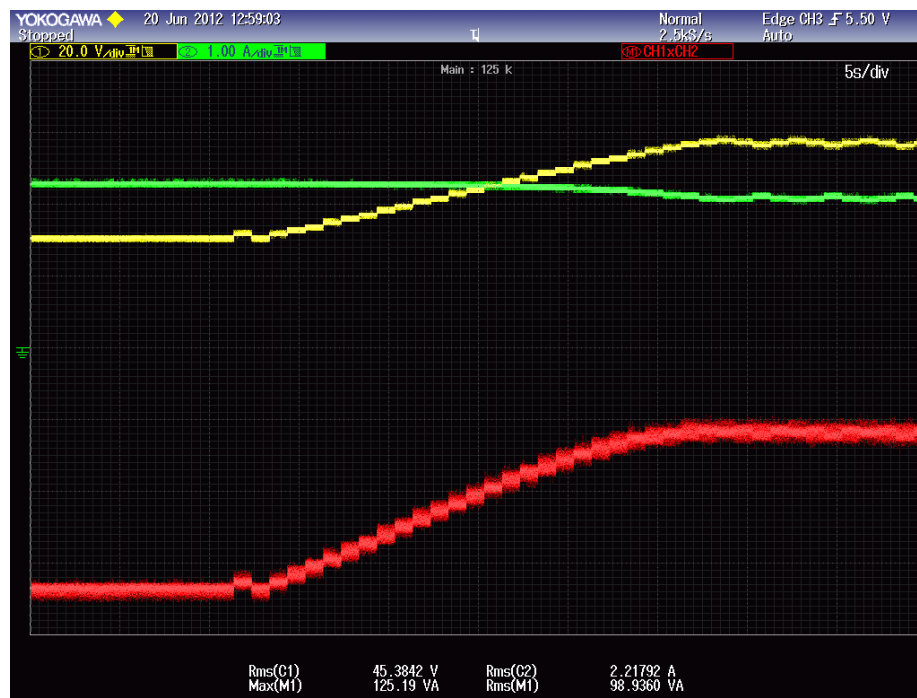


Figure 8.3: 70% test P&O, Voltage (yellow), Current (green), Power (red), 5 s/div



Figure 8.4: Shading test P&O, Voltage (yellow), Current (green), Power (red), 5 s/div



Figure 8.5: 70% test adaptive P&O, Voltage (yellow), Current (green), Power (red), 5 s/div

deviation of the derivative. It should be noted that occasionally, when the system was stable, i.e. when little noise was present, the algorithm performed very well with smaller oscillations and faster convergence speeds than the P&O algorithm.

Adaptive Algorithms The adaptive InCond was even more instable as the regular InCond algorithm. This is probably due to the aggressiveness of the algorithm. Adaptive P&O however performed much better than the regular P&O algorithm. Figure 8.5 shows the output power after being initialized at 70%. The converging to the MPP takes only 8 steps, making it three times faster than the regular P&O. Figure 8.6 shows the output power after shading the solar panel with a piece of cardboard. In this figure the overshoot is clearly visible. The output power does not directly converge, but has a few dips in between. These dips are caused by an overshoot in the duty cycle. As can be seen from this figure, the converging takes 12 steps, which is as fast as the regular P&O.

Summary After the problems with the noise were solved, we were partly able to verify part of the simulations. We have seen the adaptive P&O algorithm



Figure 8.6: Shading test adaptive P&O, Voltage (yellow), Current (green), Power (red), 5 s/div

performing up to three times faster than the regular P&O algorithm. However, still some problems with the noise need to be solved. The current sensor still picks up too much noise, making the InCond algorithms instable. This might be easily resolved by reducing the wirelength and adding better filters. Overall, with this setup, it was the adaptive P&O algorithm which performed best.

Chapter 9

Conclusion and Recommendations

9.1 Conclusion

The aim of this thesis was to implement an MPPT algorithm in a controller for an MPPT that implemented the most efficient algorithm that works in fast changing levels of irradiance and when the solar panels are partially shaded. Also, the efficiency of the algorithm had to be as high as possible, as the MPPT had to have an efficiency of at least 95%, and the implementation complexity of the algorithm could not be too high, in order to fit within the time constraints of the project.

We succeeded in implementing a controller for an MPPT. We verified that the controller and the implemented algorithms work correctly. After adding filters and buffers to the ADC inputs, the controller was able to track the MPP reliably. Due to time constraints, we chose to focus on the P&O algorithm, the InCond algorithm and their adaptive variants and simulated and implemented these hill climbing algorithms. The conclusion based on the simulations, was that the adaptive InCond algorithm is the most efficient algorithm, also in fast changing levels of irradiance. We had no time to simulate a solar panel that is partially shaded, so we can not make any conclusions about the performance of the different algorithms in that case.

In the experimental setup, we verified that the controller and all implemented algorithms worked correctly. However, we were not able to verify the simulation results. First, due to time constraints and available equipment, we were only able to evaluate shading conditions. Secondly, while using the InCond algorithm, the MPPT was not able to reliably track the MPP of a solar panel, because the system was noisy. It suffered especially from a noisy output of the current sensor, so the ADC could not obtain reliable input data. The P&O algorithms however performed exactly as expected, because of their lesser sensitivity to noise. Therefore, we concluded that the adaptive P&O algorithm performed

best in the experimental setup, because it is least susceptible to noise and was faster than the regular P&O.

Due to time constraints, we were not able to measure the efficiency of the MPPT. However, based on the simulations and the measured efficiencies of the other subsystems of the MPPT, we are confident that we succeeded in designing and implementing an MPPT algorithm with an efficiency of at least 95%.

9.2 Recommendations

First, there are more advanced and efficient tracking algorithms available than the hill climbing algorithms, especially in the case of fast changing levels of irradiance and when the solar panels are partially shaded. Simulating and implementing these algorithms is expected to deliver a higher MPPT algorithm efficiency.

Secondly, improvements can be made in the simulation model by enabling the evaluation of partially shaded solar panels.

Thirdly, the MPPT efficiency in fast changing levels of irradiance and with partial shading of the solar panels should be measured. This way, it would become more clear how the MPPT would perform in the real world.

Finally, and most importantly, the issue of noise in the current sensor circuitry should be resolved. We believe that this can be resolved by using higher order filters or reducing the wire length between the sensor and the microcontroller. Subsequently, when the noise problem is solved, accurate measurements can be conducted and it can be checked whether the adaptive InCond algorithm performs better than the other algorithms, as predicted by the simulations, and whether the MPPT meets the efficiency requirement of at least 95%.

Bibliography

- [1] V. Agarwal H. Patel. Maximum power point tracking scheme for pv systems operating under partially shaded conditions. *IEEE Trans. Ind. Electron.*, 55:1689–1698, 2008.
- [2] M. E. Ropp D. P. Hohm. Comparative study of maximum power point tracking algorithms. *Prog. Photovolt: Res. Appl.*, 11:47–62, 2003.
- [3] D. S. Morales. Maximum power point tracing algorithms for photovoltaic applications. Master’s thesis, Aalto University, 2010.
- [4] F. Nassiri Nia A.T. Sluimer. Converter design for nuna maximum power point tracker, To Be Published.
- [5] K.H. Hussein et. al. Maximum photovoltaic power tracking an algorithm for rapidly changing atmospheric conditions. *IEE Proc. Gener. Transm. Distrib.*, 142(1):59–64, 1995.
- [6] C. Shen C. Hua. Study of maximum power tracking techniques and control of dc/dc converters for photovoltaic power systems. In *Power Electronics Specialists Conference*, 1998.
- [7] P.L. Chapman T. Eser. Comparison of photovoltaic array maximum power point tracking techniques. *IEEE Trans. Energy Convers.*, 22(2):439–449, 2007.
- [8] M. E. Ropp D. P. Hohm. Comparative study of maximum power point tracking algorithms using an experimental, programmable, maximum power point tracking test bed. In *Conference Record of the Twenty-Eighth IEEE Photovoltaic Specialists Conference - 2000*, pages 1699–1702, 2000.
- [9] T. Kawamura et al. Analysis of mppt characteristics in photovoltaic power system. *Solar Energy Materials and Solar Cells*, 47:47–62, 1997.
- [10] C Zhang et. al. A modified mppt method with variable perturbation step for photovoltaic system. In *6th Power Electronics and Motion Control Conference*, pages 2096 – 2099, 2009.
- [11] S. Yurkovich K. M. Passino. *Fuzzy Control*. Addison-Wesley, 1998.

- [12] A. Zerguerras C. L. M. S. Cheikh, G. F. Tchoketch Kebir. Maximum power point tracking using a fuzzy logic control scheme. *Revue des Energies Renouvelables*, 10(3):387–395, 2007.
- [13] S. Premrudeepreechacharn N. Patcharaprakiti. Maximum power point tracking using adaptive fuzzy logic control for grid-connected photovoltaic system. In *2002 IEEE Power Engineering Society Winter Meeting*, 2002.
- [14] M. Ermis M. Bodur. Maximum power point tracking for low-power photovoltaic solar panels. In *7th Mediterranean Electrotechnical Conference*, 1994.
- [15] B. E. Stuckman D. R. Jones, C. D. Perttunen. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application*, 79(1):157–181, 1993.
- [16] K. Low T. L. Nguyen. A global maximum power point tracking scheme employing direct search algorithm for photovoltaic systems. *IEEE Trans. Ind. Electron.*, 57(10):3456–3467, 2010.
- [17] P. Lin R. A. DeCarlo. *Linear Circuit Analysis Second Edition*. Oxford University Press, 2001.
- [18] Atmel Corp. Atmega48/88/168 technical datasheet, revision 2545t-04/11. 2011.
- [19] Freescale Semiconductor Corp. M68hc11e technical datasheet, revision 5.1. 2005.
- [20] Microchip Corp. Mcp2515 stand-alone can controller with spi interface technical datasheet, revision e. 2007.
- [21] Intel Corp. Mcs-51 technical datasheet, revision t-49-19-07. 1988.
- [22] Microchip Corp. Pic16f87x technical datasheet, revision c. 2000.
- [23] Y.K.L.M. Prevoo M.G.P. Hovens. Maximum power point tracking for electric automotive applications, To Be Published.
- [24] Microchip Corp. High-speed can transceiver technical datasheet, revision d. 2003.
- [25] RM Michaelides. Rm canview usb. <http://www.rmcan.com/index.php?id=62&L=1>.

List of Figures

3.1	MPPT block scheme	6
3.2	Equivalent model of a solar panel [3]	6
3.3	The i-v and p-v characteristic of a solar panel	7
3.4	Boost Converter Topology [4]	8
3.5	Planning of the design of the MPPT controller	10
4.1	Flowchart of the P&O algorithm	12
4.2	Flowchart of the InCond algorithm	14
4.3	Example of membership functions [7]	16
5.1	Schematic drawing of the simulation circuit	21
7.1	Microcontroller block scheme	32
8.1	P-v characteristics of a solar panel in hot and cold conditions . .	37
8.2	Output current measured by the oscilloscope (green) and output current measured by the measuring circuit of the MPPT (purple)	39
8.3	70% test P&O, Voltage (yellow), Current (green), Power (red), 5 s/div	40
8.4	Shading test P&O, Voltage (yellow), Current (green), Power (red), 5 s/div	41
8.5	70% test adaptive P&O, Voltage (yellow), Current (green), Power (red), 5 s/div	42
8.6	Shading test adaptive P&O, Voltage (yellow), Current (green), Power (red), 5 s/div	43

List of Tables

4.1	Fuzzy Logic Control Rule Table [7]	16
5.1	Steady State Simulation - Efficiency in %	23
5.2	Increasing Irradiance - P&O Efficiency in %	23
5.3	Increasing Irradiance - InCond Efficiency in %	24
5.4	Increasing Irradiance - Adaptive P&O Efficiency in %	24
5.5	Increasing Irradiance - Adaptive InCond Efficiency in %	24
5.6	Decreasing Irradiance - P&O Efficiency in %	24
5.7	Decreasing Irradiance - InCond Efficiency in %	25
5.8	Decreasing Irradiance - Adaptive P&O Efficiency in %	25
5.9	Decreasing Irradiance - Adaptive InCond Efficiency in %	25
5.10	Sudden Shading Simulation - Efficiency in %	25
5.11	Sudden Exposure Simulation - Efficiency in %	26
6.1	Comparison of the different microcontrollers [19],[21],[18],[22] . . .	29

Appendix A: Matlab Simulation Code

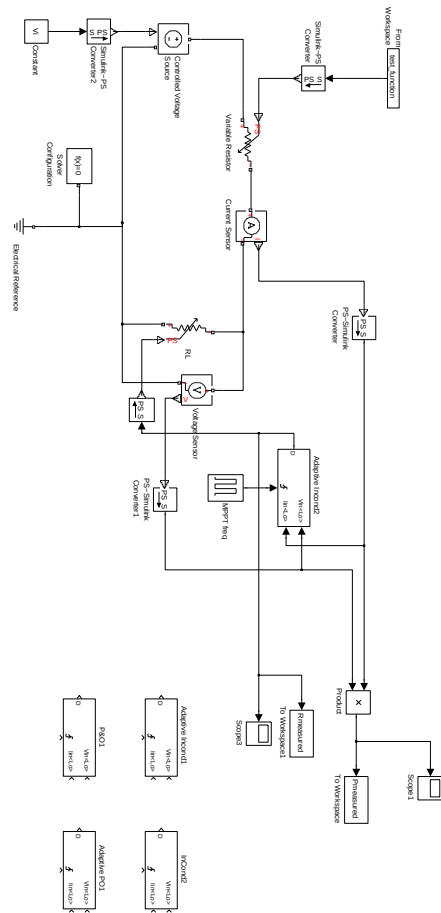
```
1 clc
2 clear all
3 close all
4 tic
5
6 % Internal Resistance (Ohm)
7 Vi = 180;
8 f_update = 100;
9 gain = 10;
10 max_step = 1000;
11 % min_steps {1,2,10,20,40} = 0.005 0.01 0.05 0.1 0.2
12 min_steps_vect = [1 2 10 20 40];
13
14 sample_time = 0.0001;
15 % End time of simulation (s)
16 sim_time = 1;
17 % Generate Test function
18 t = 0.0000:sample_time:sim_time;
19 t = t';
20
21
22 % Steady State
23 % f = 20.001 * ones(length(t),1);
24
25 %% Ramp Function
26 slopes = -[1 5 10 20 30];
27
28 % Steps Increase
29 % f = 0*t + 10+0.001;
30 % f((end-1)/2:end) = 20.001;
31
32
33 % % % Steps Decrease
34 % f = 0*t + 20+0.001;
35 % f((end-1)/2:end) = 10.001;
36
37
38
39
```

```

40
41 %Start simulation
42 efficiency = [];
43 slope_eff = [];
44 errors = [];
45 for n = slopes;
46 n
47     f = n * t + 35;
48 mppt_init_value = f(1);
49
50 for k = min_steps_vect;
51     k
52     min_steps = k;
53     test_function = [t f];
54     set_param('SimpleSim', 'SimulationCommand', 'start')
55     %Wait for it to finish
56     while strcmp(get_param('SimpleSim', 'SimulationStatus'), '↵
        running')
57         pause(1);
58     end
59
60     Pideal = Vi^2./(4*f);
61     % Retrieve measured signals
62     Pout = Pmeasured.signals.values;
63     Tout = Pmeasured.time;
64     wrong_measurements = Pout > Pideal;
65     errors = [errors sum(wrong_measurements)];
66     Pout(wrong_measurements) = [];
67     Pideal(wrong_measurements) = [];
68
69     Pout = Pout/max(Pideal);
70     Pideal = Pideal/max(Pideal);
71     efficiency = [efficiency sum(Pout(1000:end))/sum(Pideal(1000:↵
        end)) * 100];
72
73 end
74 slope_eff = [slope_eff; efficiency];
75 efficiency = [];
76 end
77
78
79 figure(1)
80 subplot(211)
81 plot(t,[f Rmeasured.signals.values])
82 subplot(212)
83 t(wrong_measurements) = [];
84 Tout(wrong_measurements) = [];
85 plot(t,Pideal,Tout,Pout);
86
87 efficiency

```

Appendix B: Simulink Simulation Block Scheme



Appendix C: C Code files

C.1 MPPTFunctions.c

```
1 /*
2  * mpptfunctions.c
3  *
4  * Created on: May 10, 2012
5  * Author: Stefan Moring
6  */
7
8 #include <avr/io.h>
9 #include "mpptfunctions.h"
10 #include <util/delay.h>
11
12 // Define values for the PWM generator and timer0
13 #define PWMTOP.VALUE 0x00C7
14 #define TIMER_TOP.VALUE 0xC2
15
16 // P&O algorithm
17 signed int po(unsigned int P, unsigned int Pold, signed int  $\leftarrow$ 
    step_old)
18 {
19     if ((P > Pold && step_old > 0) | (P < Pold && step_old < 0))
20         return 1;
21     else
22         return -1;
23 }
24
25 // Incremental Conductance Algorithm
26 signed int incond(int V, int dV, int I, int dI)
27 {
28     if (dV == 0 && dI == 0)
29         return 0;
30     else if ((dV == 0) && (dI > 0))
31         return 1;
32     else if (dV == 0 && dI < 0)
33         return -1;
34     else if (dI / dV == -I / V)
35         return 0;
36     else if (dI / dV > -I / V)
37         return 1;
38     else if (dI / dV < -I / V)
39         return -1;
```

```

40
41     else
42         return 0; // should never happen
43 }
44
45 // Function to sweep through the spectrum, not tested!
46 void sweep()
47 {
48     unsigned int I;
49     unsigned int V;
50     int t = 0, P = 0;
51     for (int i = 0; i <= PWM_TOP_VALUE; i++) {
52         _delay_ms(50);
53         OCR1A = i;
54
55         read_V(&V);
56         read_I(&I);
57
58         if (V * I > P) {
59             P = V * I;
60             t = i;
61         }
62     }
63
64     OCR1A = t;
65 }
66
67 // Function to increase the duty cycle
68 void set_duty(int d)
69 {
70     unsigned int temp = OCR1A + d;
71     if (temp > PWM_TOP_VALUE)
72         return;
73     else
74         OCR1A = temp;
75 }
76
77 // Function to initialize timer0
78 void init_timer()
79 {
80     // set timer counter to 0;
81     TCNT0 = 0;
82     // CTC Mode
83     TCCR0A |= (1 << WGM01);
84     // Prescale timer to f_io/1024
85     TCCR0B |= (1 << CS02) | (0 << CS01) | (1 << CS00);
86     // Count to TIMER_TOP_VALUE
87     OCR0A = TIMER_TOP_VALUE;
88     // Enable timer0 interrupt
89     TIMSK0 = (1 << OCIE0A);
90 }
91
92 // Function to initialize the ADC channels
93 void init_adc()
94 {
95     // Set ADC5 and ADC4 as input
96     DDRC &= ~( (1 << PC5) | (1 << PC4) );

```

```

97 // prescale adc clock to clk/64
98 ADCSRA |= (1 << ADPS2) | (1 << ADPS1);
99 // set ref voltage to avcc
100 ADMUX |= (1 << REFS0);
101 // Set resolution to 10bits, output in ADCH reg
102 ADMUX &= ~(1 << ADLAR);
103 // set to freerunning
104 ADMUX |= (1 << ADSC);
105 // Set ADC channel to ADC5 (pin 28)
106 ADMUX &= ~(1 << MUX3) | (1 << MUX1);
107 ADMUX |= (1 << MUX2) | (1 << MUX0);
108 // enable adc
109 ADCSRA |= (1 << ADEN);
110 ADCSRA |= (1 << ADSC); // start a2d conversion
111 }
112
113 // Function to initialize the PWM generator
114 void init_pwm()
115 {
116     // Set initial duty cycle
117     OCR1A = 5 * PWM_TOP_VALUE / 10;
118     // Set maximum count value
119     ICR1 = PWM_TOP_VALUE;
120     // Set pin 15/B5(OC1A) as output
121     DDRB |= (1 << DDB1);
122     // Non-inverted output on OC1A
123     TCCR1A |= (1 << COM1A1);
124     // Fast PWM with TOP in ICR1
125     TCCR1A |= (1 << WGM11);
126     TCCR1B |= (1 << WGM13) | (1 << WGM12);
127     // Set clock to no prescaling
128     TCCR1B |= (0 << CS02) | (1 << CS10);
129 }
130
131 // Function to measure the voltage
132 void read_V(unsigned int *V)
133 // Arguments *V & *dV are the address of the registers where the ↔
134 // measured values should be stored
135 {
136     // Set ADC channel to ADC4
137     ADMUX |= (1 << MUX0);
138     // Start conversion
139     ADCSRA |= (1 << ADSC);
140     // Wait for conversion to finish
141     while (!(ADCSRA & (1 << ADIF)));
142     // Return measured value in &V
143     *V = ADC >> 6;
144 }
145
146 // Function to measure the current
147 void read_I(unsigned int *I)
148 {
149     // Set ADC channel to ADC5
150     ADMUX &= ~(1 << MUX0);
151     // Start conversion
152     ADCSRA |= (1 << ADSC);

```

```

153     // Wait for conversion to be done
154     while (!(ADCSRA & (1 << ADIF))) ;
155     // Return measured value in &I
156     *I = ADC >> 6;
157 }
158
159 // Function to measure the voltage and calculate dV
160 void read_dV(unsigned int *V, signed int *dV)
161 // Arguments *V & *dV are the address of the registers where the ←
    measured values should be stored
162 {
163     unsigned int t = 0;
164     read_V(&t);
165     // Calculate and return dV and I in &dI and &I
166     *dV = t - *V;
167     *V = t;
168 }
169
170 // Function to measure the current and calculate dI
171 void read_dI(unsigned int *I, signed int *dI)
172 // Arguments *I & *dI are the address of the registers where the ←
    measured values should be stored
173 {
174     unsigned int t = 0;
175     read_I(&t);
176     // Calculate and return dI and I in &dI and &I
177     *dI = t - *I;
178     *I = t;
179 }

```

C.2 MPPTmain.c

```

1
2 /*
3  * MPPTmain.c
4  *
5  * Created on: May 3, 2012
6  * Author: Stefan Moring
7  */
8
9 #include <avr/io.h>
10 #include <util/delay.h>
11 #include <avr/interrupt.h>
12
13 #include "mpptfunctions.h"
14 #include "can.h"
15
16
17 //Define the operating modes used in the switch function
18 #define PO 1
19 #define PO_ADAPT 2
20 #define INCOND 3
21 #define INCOND_ADAPT 4
22 #define DUTY_CONTROL 5
23 #define PO_STABLE 6
24 #define INCOND_STABLE 7
25

```

```

26 // Variables used by the MPPT algorithms
27 unsigned int V = 0, I = 0, Vold = 0, Iold = 0;
28 signed int dV = 0, dI = 0;
29 unsigned int P = 0, Pold = 0;
30 signed int step = 0, stepold = 0;
31
32 // Buffer used by the adaptive algorithms
33 int adapt_buffer[2] = { 0, 0 };
34
35 // Buffers used by the stable algorithms
36 unsigned int Pbuf[4], Vbuf[4], Ibuf[4];
37 signed int dVbuf[4], dIbuf[4];
38
39 // Variables used in the main code
40 char OPERATING_MODE = 0;
41 int count = 0;
42 int STEPSIZE = 1;
43
44 int main(void) {
45     cli();
46     OPERATING_MODE = INCOND;
47     init_adc();
48     init_timer();
49     init_pwm();
50     sei();
51
52     while (1);
53
54     // Should never happen
55     return 0;
56 }
57
58
59
60 // Interrupt routine for timer0
61 ISR(SIG_OUTPUT_COMPARE0A)
62 {
63     // Divide interrupt frequency, count == 0 for 100 Hz, 1 for 50←
64     // Hz, 2 for 25 Hz etc.
65     if (count == 2)
66     {
67         count = 0;
68         switch (OPERATING_MODE)
69         {
70             case PO:
71                 Pold = P;
72                 stepold = step;
73                 read_V(&V);
74                 read_I(&I);
75                 P = I * V;
76                 step = po(P, Pold, stepold);
77                 set_duty(step * STEPSIZE);
78                 break;
79             case PO_STABLE:
80
81

```

```

82         Pold = P;
83         stepold = step;
84         // Measure V and I four times
85         for (int i = 0; i < 4; i++)
86         {
87             read_V(&V);
88             read_I(&I);
89             Pbuf[i] = I * V;
90         }
91         // Calculate average P
92         P = Pbuf[3] + Pbuf[2] + Pbuf[1] + Pbuf[0];
93         P = P / 4;
94         step = po(P, Pold, stepold);
95         set_duty(step * STEPSIZE);
96         break;
97
98     case INCOND:
99
100         read_dV(&V, &dV);
101         read_dI(&I, &dI);
102         step = incond(V, dV, I, dI);
103         set_duty(step * STEPSIZE);
104         break;
105
106     case INCOND_STABLE:
107         // Measure V,dV,I and dI four times
108         for (int i = 0; i < 4; i++)
109         {
110             read_dV(&Vbuf[i], &dVbuf[i]);
111             read_dI(&Ibuf[i], &dIbuf[i]);
112         }
113
114         // Calculate Averages
115         V = Vbuf[3] + Vbuf[2] + Vbuf[1] + Vbuf[0];
116         dV = dVbuf[3] + dVbuf[2] + dVbuf[1] + dVbuf[0];
117         I = Ibuf[3] + Ibuf[2] + Ibuf[1] + Ibuf[0];
118         dI = dIbuf[3] + dIbuf[2] + dIbuf[1] + dIbuf[0];
119         V = V / 4;
120         dV = dV / 4;
121         I = I / 4;
122         dI = dI / 4;
123         step = incond(V, dV, I, dI);
124         set_duty(step * STEPSIZE);
125         break;
126
127     case PO_ADAPT:
128
129         adapt_buffer[0] = adapt_buffer[1];
130         stepold = step;
131         Pold = P;
132         Vold = V;
133         read_V(&V);
134         read_I(&I);
135         P = I * V;
136         step = po(P, Pold, stepold);
137         // if signs are equal, accumulate, if not control ←
            signal equals the result of incond(..)

```

```

138         if (((adapt_buffer[0] > 0) && (step > 0)) || ((←
            adapt_buffer[0] < 0) && (step < 0)))
139             adapt_buffer[1] = adapt_buffer[0] + step;
140         else
141             adapt_buffer[1] = step;
142
143         set_duty(STEPSIZE * adapt_buffer[0]);
144         break;
145
146     case INCOND_ADAPT:
147
148         adapt_buffer[0] = adapt_buffer[1];
149
150         read_dV(&V, &dV);
151         read_dI(&I, &dI);
152         step = incond(V, dV, I, dI);
153         // if signs are equal, accumulate, if not control ←
            signal equals the result of incond(..)
154         if (((adapt_buffer[0] > 0) && (step > 0)) || ((←
            adapt_buffer[0] < 0) & (step < 0)))
            adapt_buffer[1] = adapt_buffer[0] + step;
155         else
156             adapt_buffer[1] = step;
157
158         set_duty(STEPSIZE * adapt_buffer[0]);
159         break;
160
161     case DUTY_CONTROL:
162
163         read_V(&V);
164         double d = (double) V / 1023;
165         d = d * 199;
166         OCR1A = (int) d;
167         break;
168
169     default:
170         // if operating mode not set, set to P&O (Should ←
            never happen)
171         OPERATING_MODE = PO;
172     }
173 }
174 }
175 else
176     count++;
177 }

```

C.3 can.c

```

1  /*
2  * spi.c
3  *
4  * Created on: May 9, 2012
5  * Author: Stefan Moring
6  */
7
8
9
10

```

```

11 #include <avr/io.h>
12 #include <util/delay.h>
13 #include "can.h"
14
15
16
17 // Function to initialise the SPI
18 void init_spi(void)
19 {
20     // Set SS (4), MISO (6) and SCK (7) as output
21     SPI_DDR |= (1 << SS_PIN) | (1 << SCK_PIN) | (1 << MISO_PIN);
22     // Set MOSI (5) as input
23     SPI_DDR &= ~(1 << MOSI_PIN);
24     // Enable SPI
25     SPCR |= (1<<SPE)|(1<<MSTR);
26     // Prescale clock to Fclk/4 (5 MHz)
27     SPSR |= (0 << SPI2X) | (0 << SPR1) | (0 << SPR0) | (1<<CPOL)↵
        |(1<<CPHA);
28     SPI_PORT |= (1 << SS_PIN);
29 }
30
31 // Function to send data over SPI
32 char spi_send(char d)
33 {
34     SPDR = d;
35     while( !( SPSR & (1<<SPIF) ) );           // Wait for data ↵
        register to be ready
36     return SPDR;
37 }
38
39 // Function to write to the MCP2515
40 void mcp_write(char address, char data )
41 {
42
43     SPI_PORT &= ~(1<<SS_PIN);                // Toggle SS pin
44
45     spi_send(MCP_WRITE);                     // Request write
46     spi_send(address);                       // Send address to write to
47     spi_send(data);                          // Send data to write
48
49     SPI_PORT |= (1<<SS_PIN);                  // Toggle SS pin
50 }
51
52 char mcp_read(char address)
53 {
54     SPI_PORT &= ~(1<<SS_PIN);                // Toggle SS pin
55
56     spi_send(MCP_READ);                      // Request read
57     spi_send(address);                       // Send address to read
58     char data = spi_send(0xFF);              // retrieve data
59
60     SPI_PORT |= (1<<SS_PIN);                  // Toggle SS pin
61     return data;
62 }
63
64 void mcp_bit_mod(char address, char mask, char data)
65 {

```

```

66     SPI_PORT &= ~(1 << SS_PIN);    // Toggle SS pin
67
68     spi_send(MCP_BIT_MOD);          // request bit modification
69     spi_send(address);              // send address of register ↔
70         to modify
71     spi_send(mask);                 // send modification mask
72     spi_send(data);
73     SPI_PORT &= ~(1 << SS_PIN);    // Toggle SS pin
74 }
75
76 void mcp_init()
77 {
78     char config;
79     init_spi();
80     SPI_PORT &= ~(1<<SS_PIN);        // Toggle SS pin
81     spi_send(MCP_RESET);             // Reset to enter config mode
82     _delay_ms(1);                   // Wait for MCP to settle
83     SPI_PORT |= (1<<SS_PIN);         // Toggle SS pin
84     _delay_ms(10);                  // Wait for MCP to settle
85
86
87
88
89
90
91
92     mcp_write(CNF1,0x00);            // set for 500 kbps, 0% error↔
93         , 75% sampling point
94     mcp_write(CNF2,0xb6);
95     mcp_write(CNF3,0x04);
96
97
98
99     mcp_write( RXB0CTRL, (1<<RXM1)|(1<<RXM0) );    // accept↔
100         all messages
101     mcp_write( RXB1CTRL, (1<<RXM1)|(1<<RXM0) );
102
103     mcp_write(RXMOSIDH,0);           // dit ↔
104         weet ik nog niet
105     mcp_write(RXMOSIDL,0);
106     mcp_write(RXMOEID8,0);
107     mcp_write(RXMOEID0,0);
108     mcp_write(RXM1SIDH,0);
109     mcp_write(RXM1SIDL,0);
110     mcp_write(RXM1EID8,0);
111     mcp_write(RXM1EID0,0);
112
113     config = 0x0F;                   // Set BF Pins as interrupt
114     mcp_write(BFPCTRL, config);
115
116     config = 0x00;
117     mcp_write(CANINTE,config);       // Enable receive ↔
118         interrupts

```

```

118     mcp_write( TXRTSCTRL, config);
119
120     mcp_bit_mod(CANCNTRL,0xE0,0);           // Exit Config Mode
121
122
123
124     config = 0b00000010;
125
126     PORTC &= ~(1 << PC3);
127     _delay_ms(200);
128     PORTC |= (1 << PC3);
129     _delay_ms(200);
130     for(int i = 0; i < 8; i++)
131     {
132         if(config && (1 << i))
133             PORTC |= (1 << PC3);
134         else
135             PORTC &= ~(1 << PC3);
136         _delay_ms(100);
137     }
138
139     PORTC |= (1 << PC3);
140     _delay_ms(200);
141     PORTC &= ~(1 << PC3);
142     _delay_ms(200);
143
144
145
146     //while ((mcp_read(CANSTAT) & 0xE0) != 0x40);
147     PORTC |= (1<< PC3);
148
149
150
151
152 }
153
154
155
156 void mcp_send_message(int ID, char *data, char msglength)
157 {
158
159     mcp_write(TXBOSIDH,(char) (ID >> 3));
160     mcp_write(TXBOSIDL,(char) (ID << 5));
161
162     mcp_write(TXBODLC,msglength);
163
164     for(int i = 0; i < msglength; i++)
165         mcp_write(TXBODO + i,data[i]);
166
167     SPI_PORT &= ~(1<<SS_PIN);
168     spi_send(RTS | 0x01);
169     SPI_PORT |= (1<<SS_PIN);
170
171 }

```

Appendix D: Brief of Requirements

D.1 Use of the MPPT

- [D.1.1] The MPPT will maximize the output power of a connected solar cell under all conditions specified in the Usage Characteristics.
- [D.1.2] The MPPT will be installed fivefold in the Nuna 6 solar racing vehicle, this will be called the system.
- [D.1.3] The system can be installed, removed and operated by any Nuon Solar team member, without requiring prior knowledge of its inner workings.
- [D.1.4] The MPPT provides plug-and-play connectivity.
- [D.1.5] Each MPPT has an external reset switch.
- [D.1.6] The system will communicate with the vehicle.
- [D.1.7] The MPPTs operating status can be visually inspected.
- [D.1.8] The system will be robust, stable and safe.
- [D.1.9] The system will be constructed as light and compact as possible.
- [D.1.10] The system can be easily integrated into Nuna 6 and its future successors.
- [D.1.11] Each MPPT will have a life expectancy under normal operating conditions of at least 3 years.
- [D.1.12] The MPPT is maintenance free during its expected lifetime.

D.2 Ecological Requirements

- [D.2.1] The system wastes as little energy as possible.
- [D.2.2] The production process has minimal environmental impact.
- [D.2.3] All components are RoHS compliant.

D.3 Technical Requirements

- [D.3.1] A total of 5 on board MPPTs will provide Nuna 6 with more power under equal conditions than the current power point tracking system.
- [D.3.2] A fault in one of the MPPTs will not lead to loss of power to Nuna 6, other than the power controlled by the faulty MPPT itself.
- [D.3.3] The MPPT accepts input voltages between 80V and 100V. Safety margins are taken at 60V and 120V.
- [D.3.4] The MPPT connects to the Nuna 6 power bus, providing an output voltage between 110V and 160V.
- [D.3.5] Should the output voltage rise above 160V, indicating a full battery on the Nuna vehicle, the system will stop tracking to prevent overcharging.
- [D.3.6] The systems nominal input power is between 20W and 100W with an absolute maximum of 200W.
- [D.3.7] The average efficiency of each MPPT will be greater than 95%.
- [D.3.8] The system operates without malfunction with environmental temperatures ranging between -10°C and 80°C .
- [D.3.9] The system operates without malfunction with environmental humidity ranging between 5% and 95%.
- [D.3.10] Each MPPT communicates with the vehicle via the vehicles CAN-bus on a send-only basis, providing (next to all communication necessary for safe and reliable CAN-bus operation) at least: power in [W], power out [W], input voltage [V], output voltage [V], output current [A].
- [D.3.11] The MPPT has at least 4 status LEDs, showing operating status and/or possible error codes.
- [D.3.12] The MPPT provides In-System Programming (ISP) using a JTAG interface.
- [D.3.13] All components meet the specifications as provided by Veolia World Solar Challenge.
- [D.3.14] High-voltage connections must be properly shielded.
- [D.3.15] All critical high-power connections contain correctly dimensioned fuses.
- [D.3.16] Casings and enclosures must be constructed properly and provide adequate sealing, strength and durability to protect enclosed components.
- [D.3.17] All connections will be dust-sealed, shock and vibration proof.

D.4 Production and Installation Requirements

- [D.4.1] The MPPT and all its components are integrated in a enclosure for easy mounting in the Nuna 6.
- [D.4.2] Installation of the system is possible using standard tools available to the Nuon Solar team.
- [D.4.3] Each enclosure has a label for identification purposes.
- [D.4.4] External connections provide standard connectors for easy installation in the vehicle: power bus connectors: Green Power Connector PC 4 HV/4-STF-7.62 Phoenix, CAN-bus connectors: Orange CAN Connector BL 3,5/4 Weidmüller.

D.5 Requirements on Product End-of-Life

- [D.5.1] The MPPT complies with EU WEEE guidelines for electronic waste disposal.
- [D.5.2] The system must be removable from the vehicle.
- [D.5.3] Any batteries in the system must be easy to remove to be processed separately.