Electrical Power System for the CubeSTAR Nanosatellite

by

MARTIN OREDSSON
Department of Physics

THESIS
for the degree of
MASTER OF SCIENCE

(Master i Elektronikk og Datateknologi)

Faculty of Mathematics and Natural Sciences
University of Oslo, September 2010

Abstract

This thesis describes the development of an Electrical Power System (EPS) prototype for the CubeSTAR nanosatellite. Without a power system the other subsystems on the satellite cannot function, therefore a continuous and reliable power source is needed.

CubeSTAR's triple-junction solar cells are characterized and a general introduction to spacecraft EPS is given. A solar simulator circuit that mimics the solar cell behavior is also described. The energy storage cells, which are Lithium Iron Phosphate based, are tested under various conditions. The proposed system employs two redundant and digitally controlled maximum power point trackers. To raise the efficiency of this low-voltage system, synchronous buck converters are used. A PCB implementation that fits into the pre-defined CubeSTAR structure is also presented.

Acknowledgments

Mom and Dad, you deserve your own page, but this will have to do: thank you for your unconditional love, support, and endless supply of Swedish coffee—they are the fundamental components of this work.

I'm very grateful to my supervisor, Associate professor Torfinn Lindem at the Electronics Group, for giving me the opportunity to work with such an interesting topic. By placing himself in the background, he has given me the freedom to explore the the topic of my thesis, and I couldn't have asked for more.

Stein Lyng Nielsen at the Electronics Laboratory has been an invaluable resource. His willingness to share his experience on anything from printed circuit board design to the famous wine districts of Portugal has been one of the highlights the past year.

Tore Andrè Bekkeng also deserves a special mention for being a guiding star throughout the phases of this work. Anything from LaTeX, proofreading, Matlab to PCB design, you've always had the answers and the kindness to share them. I must also acknowledge the Plasma Group and especially Espen Trondsen who have given me a the best possible conditions to do my thesis work. This has included unlimited access to their electronics lab and Espen's practical wisdom—what a privilege! My friend and fellow student Manual Lains has also been helpful in absorbing my bad jokes with minimum fuss.

Finally, my brother and aspiring mechanical engineering wizard, Mattias; thanks man.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 CubeSTAR and the Cubesat Standard 2

1.3 Goals of Present Work 3

1.4 Thesis Outline 4

2 Harvesting Energy from the Sun **5**

2.1 Solar Energy 5

2.2 From Solar- to Electrical Energy 7

2.3 Semiconductor Basics 9

2.4 Electronic Behavior of a Triple Junction Solar Cell 11

2.4.1 Short-Circuit Current 13

2.4.2 Open-Circuit Voltage 14

2.4.3 Maximum Power Point 16

2.4.4 Fill Factor 19

2.4.5 Efficiency 20

2.4.6 Loading the Solar Cell 21

2.5 Angular Response of a Solar Cell 22

2.5.1 Case 1: Minimum Sun 24

2.5.2 Case 2: One side facing the Sun 24

2.5.3 Case 3: Maximum Sun 24

2.5.4 Case 4: Average effective area for a free-tumbling spacecraft 25

3 Spacecraft Electrical Power Systems 29

3.1 Objectives 29

3.2 Power Regulation Topology 30

3.2.1 Direct Energy Transfer (DET) 31

3.2.2 Peak Power Transfer (PPT) 31

3.3 Orbital Considerations 33

3.3.1 Orbit Period and the Eclipse 33

3.3.2 Orbit Temperature 35

3.4 Maximum Power Point Tracking 37

3.4.1 Constant Voltage Method 38

3.4.2 Perturbation and Observation Method 38

3.5 Energy Storage with Lithium Iron Phosphate Battery 41

3.5.1 Chemistry Specific Properties 42
3.5.2 Battery Capacity Calculations 43
3.5.3 Forming a Battery Pack 46
3.5.4 Charging a Lithium-Iron Phosphate Cell 48
3.6 System Reliability 50
3.6.1 Failure Tolerance 50
3.6.2 Fault Protection 50
3.6.3 Redundancy 51
3.6.4 Thermal Design 51

4 System Design **55**

4.1 System Overview 55
4.2 Wiring Up the Solar Cells 56
4.3 Synchronous Buck Converter 57
4.3.1 Adding a Synchronous Rectifier 60
4.3.2 Deriving the LC Filter Component Values 61
4.4 Digital Control Block 66
4.4.1 System State Machine 67
4.4.2 Event Transition Conditions 68
4.5 Large Signal Analysis 68

4.5.1 A Note on Small-signal Loop Response 70
4.6 Sensor to ADC - Analog Interface Design 71
4.6.1 Example: Sensing Battery Voltage 72
4.7 Telemetry and the TWI Bus 73
4.7.1 Slave Reaction to Address Packet 74
4.7.2 Receiving Data 75
4.7.3 Transmitting Data 75
4.8 Powering the Microcontroller 77
4.9 Fault Protection and Monitoring Unit 79

5 PCB Realization **83**

5.1 Physical Dimensions 85
5.2 Electromagnetic Compatibility 88
5.2.1 Conductor Parasitics 88
5.2.2 Noise and Interference 91
5.2.3 Current Return Path and Ground Noise 92
5.3 PCB Layout Noise Reduction Techniques 95
5.3.1 Decoupling 95
5.3.2 Reducing the Loop Areas 96
5.3.3 Implementation of a Passive RC Snubber 98

6 Test Results **109**

6.1 Dark Current-Voltage Measurements on UTJ Solar Cell 109

6.2 UTJ Solar Cell Angular Response 114

6.3 Open Loop Dutycycle Stepping 115

6.4 P&O Characteristic Oscillation 117

6.5 P&O Response Time 117

6.6 MPPT to PID State Transition 118

6.7 Operation Mode Transitions 120

6.8 Over-discharge Abuse Tolerance 123

6.9 $LiFePO_4$ Discharge in -17°C 124

6.10 $LiFePO_4$ vs Lithium Ion Polymer (LiPo) 125

7 Conclusions **127**

Bibliography **131**

A Solar Panel Simulation Circuit **133**

A.1 Motivation 133

A.2 Simulation Goals 133

A.3 SPICE Circuit Implementation 135

A.4 PCB Realization 138

A.5 Results 139

B Triple Junction Solar Cell SPICE Model **141**

B.1 Creating a Subcircuit in SPICE 141

B.2 Excitation Circuit 143

C Discrete PID Controller **147**

C.1 Proportional Term 147

C.2 Integral Term 148

C.3 Derivative Term 149

C.4 Implementation 149

C.4.1 A Note on Loop Tuning 150

D Two Wire Interface (TWI) **153**

D.1 Electrical Characteristics 153

D.2 Start and Stop Conditions 153

D.3 Address 154

D.4 Data Transfer 155

D.5 Clock Stretching 156

E Production Files **157**

E.1 Parts List 158

E.2 Schematics 161

E.3 Gerber Files 177

F C Code for ATXmega128A1 **187**

List of Figures

1.1 CubeSTAR 3D Model 2

1.2 CubeSTAR Subsystems Overview 3

2.1 ASTM standard solar spectrum 6

2.2 Front and rear view of a Spectrolab solar cell 9

2.3 Energy bandgap between valence- and conduction bands . . . 10

2.4 Triple junction solar cell stack-up 11

2.5 Triple junction solar cell equivalent circuit 12

2.6 Triple junction solar cell IV plot 14

2.7 The effect of varying irradiance levels on the I-V curve. 15

2.8 The effect of varying temperature on the I-V curve. 16

2.9 Dual plot of I-V and P-V curve 17

2.10 The effect of internal resistance on the I-V curve. 19

2.11 The effect of internal leakage resistance on the I-V curve. . . . 20

2.12 Constant power load lines intersecting the IV curve 21

2.13 Solar incident angle: Kelly cosine vs Cosine law 23

2.14 Geometry used to calculate average orbital power 26

2.15 Contour plot of the angular response 27

3.1 EPS main functional parts overview 30

3.2 Direct Energy Transfer system block diagram 31

3.3 Peak Power Tracking system block diagram 32

3.4 LEO reference view 33

3.5 Geometry to calculate orbital eclipse and sunlit periods 34

3.6 Low-earth orbit temperature profile 36

3.7 MPPT control block diagram 37

3.8 MPPT: Perturb and Observe flow diagram 40

3.9 Dimensions and picture of LiFePO4 cell 41

3.10 $LiFePO_4$ cycle life performance 42

3.11 Total solar array energy requirements 43

3.12 CubeSTAR batterypack open-view 47

3.13 $LiFePO_4$ charging regime 48

3.14 Equivalent circuit during the constant voltage charging phase. 49

3.15 Component heat removal in space/terrestrial conditions 52

4.1 System overview 56

4.2 Solar array configuration 57

4.3 Synchronous buck converter 58

4.4 Buck converter dutycycle vs V_{out} 59

4.5 Buck converter ON/OFF states 60

4.6 Oscilloscope view of the switch node voltage 61

4.7 Deadtime insertion on the gate driver signals 62

4.8 Buck inductor voltage and current waveforms 63

4.9 System state machine 67

4.10 MPPT operation modes 69

4.11 PID operation mode 70

4.12 Transducer interface design (TID) block diagram. 71

4.13 Implemented TWI slave flow diagram 76

4.14 Microcontroller power scheme 77

4.15 Device: TPS2556 from TI 79

4.16 Device: INA138 from TI 80

4.17 Fault protection and monitoring unit 81

5.1 The first PCB prototype 83

5.2 The second PCB prototype 84

5.3 The third PCB prototype 85

5.4 Four layer PCB stackup 86

5.5 The CubeSTAR module and backpanel templates 87

5.6 Trace inductance and plane capacitance geometries. 89

5.7 Trace resistance vs width 90

5.8 Trace loop inductance vs width 91

5.9 Typical noise path 92

5.10 Return current loop frequency dependency 94

5.11 Transients with and without decoupling 96

5.12 PCB measures taken to reduced noise 97

5.13 Avoiding the gaps in the ground plane 98

5.14 A simplified parasitic model for the switch node. 99

5.15 SR parasitic SPICE model 101

5.16 Parasitic turn-on simulation 102

5.17 Capacitive coupling on the SR gate 103

5.18 Ringing sans snubber 105

5.19 New ringing frequency with capacitor 106

5.20 Damping the ringing with a RC snubber 107

6.1 UTJ diode dark-current test setup 110

6.2 Test: Dark current measurement in -20°C. 110

6.3 Test: Dark current measurement in 3°C. 111

6.4 Test: Dark current measurement in 23°C. 111

6.5 Extrapolation of non-linear regression fits 112

6.6 Comparison of V_{oc} and extrapolated dark-current values . . . 113

6.7 Test: Angular response of two series connect UTJ solar cells. . 114

6.8 Solar simulator IV curve on which the MPPT testing was done. 115

6.9 Test: The effect of dutycycle on current, voltage and power. . 116

6.10 Test: P&O characteristic oscillation 117

6.11 Test: Systen response to varying V_{oc} 118

6.12 Test: System state transition 119

6.13 Test: Charge-discharge-charge cycling 120

6.14 Test setup to test the large signal response. 121

6.15 Battery charging in MPPT mode with dummy load 121

6.16 Dummy load draw on battery charging in PID mode 122

6.17 $LiFePO_4$ over-discharge abuse test 123

6.18 Setup for the -17 degree $LiFePO_4$ discharge test. 124

6.19 Test: $LiFePO_4$ cold discharge characteristics 125

6.20 Test: $LiFePO_4$ vs Lithium Ion Polymer discharge 126

A.1 Solar simulator IV curve 135

A.2 Solar simulator IV and PV curves 136

A.3 Circuit diagram of the solar cell simulation model. 137

A.4 Solar simulator PCB layout 138

A.5 Solar simulator photograph 139

A.6 Solar simulator V_{oc} and I_{sc} range 140

B.1 SPICE circuit used to generate the plots in this section. 144

C.1 PID to Buck block diagram 149

C.2 PID regulator block diagram. 150

D.1 Start and stop conditions. 154

D.2 The first byte after the start condition. 155

D.3 Acknowledge on the TWI bus. 155

D.4 Master read (top) and write (bottom) transaction. 156

E.1 TopSilk Layer 178

E.2 TopStop Layer 179

E.3 TopPaste Layer 180

E.4 TopElec Layer 181

E.5 Ground Layer 182

E.6 Power Layer 183

E.7 BotElec Layer 184

E.8 BotStop Layer 185

E.9 Drill file 186

List of Tables

2.1 Key parameters from the UTJ datasheet 8

2.2 Required energy to ionize different semiconductor materials . . 10

2.3 Solar array power output summary 28

3.1 Lithium Iron Phosphate batteries from A123 Systems 43

3.2 Battery capacity calculations summary. 46

4.1 Theoretical buck converter parameters. 66

4.2 Telemetry data sent on master request. 75

A.1 Spectrolab UTJ Characteristics 134

A.2 Solar cell simulation circuit parameters 139

Chapter 1

Introduction

This thesis gives a theoretical introduction to spacecraft electrical power systems (EPS) and describes the design and development of a prototype EPS for the CubeSTAR nanosatellite. Providing power for electronics that operate in space give rise to a unique set of constraints, and as the only available source of power, the EPS is literally the lifeline of the other systems on the satellite.

By using a single switching regulator with a digital control loop, the maximum amount of solar power can be converted to electrical energy. At the same time, the system controls charging of the energy storage cells that are vital during the solar eclipse periods. Thus, a low parts-count and efficient system is made possible.

1.1 Background and Motivation

This thesis is a part of the CubeSTAR student satellite project at the University of Oslo (UiO). The project is a part of the Norwegian student satellite program (ANSAT) which is mainly funded by the government agency Norwegian Space Centre (NSC). The ANSAT program itself is run by the Norwegian Center for Space-related Education (NAROM) which is based at Andøya Rocket Range (ARR). The scientific mission of CubeSTAR is to demonstrate a new concept of measuring the electron density in the iono-

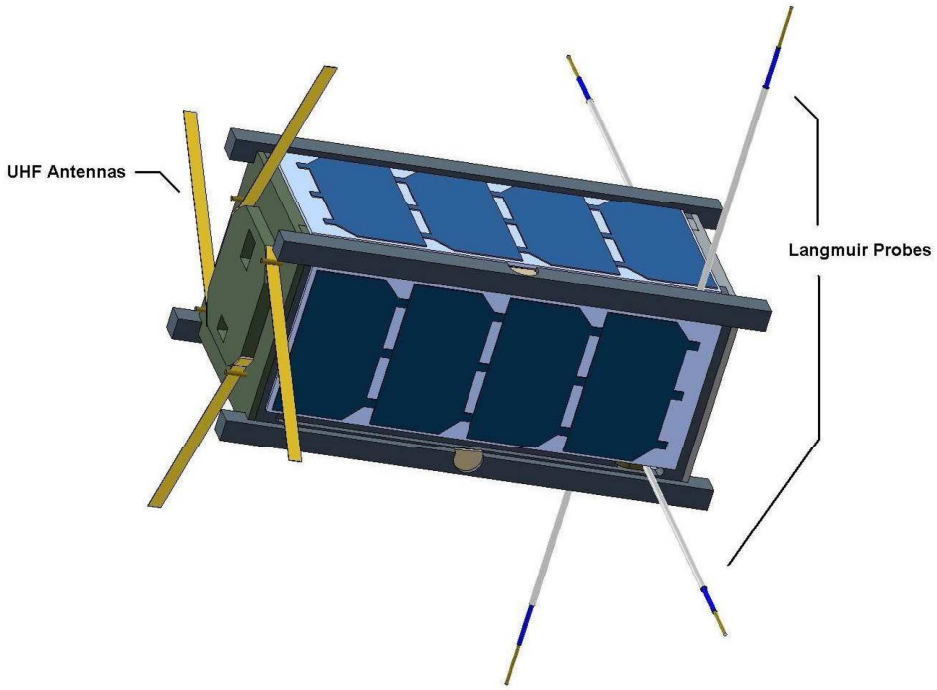


Figure 1.1: The CubeSTAR satellite has dimensions 20cm × 10 cm × 10cm and are covered by solar cells on four sides. The four probes for measuring electron density can be seen on the right side of the structure, while the communication antennas are shown to the left.

spheric plasma, with a new multi-probe system that promises greater spatial resolution than its predecessors. But in addition to that, the CubeSTAR mission has an educational component which is to provide a platform where students from several disciplines can work together towards a common goal of launching their work into space.

1.2 CubeSTAR and the Cubesat Standard

The Cubesat standard, developed by California Polytechnic State University and Standford University in 1999, is a specification of a type of miniaturized satellite with the dimensions of a 10 cm cube and a maximum weight of one kilogram. This is known as a "1U" satellite. It didn't take long though before scaled versions of this 1U structure showed up, and both "2U" (20cm × 10

cm × 10cm) and "3U" (30cm × 10cm × 10cm) structures have been built and launched. The CubeSTAR satellite is built after the 2U specification.

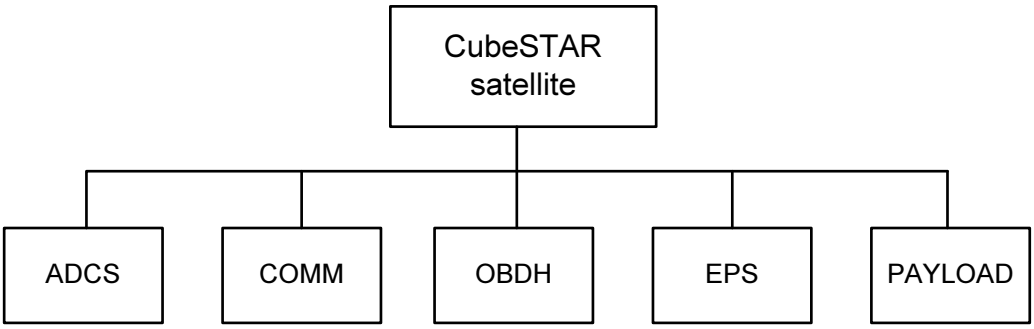


Figure 1.2: The Electrical Power System (EPS) is one of the five CubeSTAR subsystems.

The satellite naturally divides itself into subsystems, which form the basis for separate work groups where each group works on a single subsystem. These range from Attitude Control (ADCS) to stabilize the satellite, communication link (COMM), on-board data handling (OBDH), electrical power system (EPS) and the scientific payload as mentioned. Communication with the satellite is made possible by ground stations at UiO and ARS/Andøya among others.

1.3 Goals of Present Work

The goals of this work have been to provide answers to the following questions:

- How much energy is available to the satellite and how can this energy be utilized most efficiently?
- How should the energy be stored on-board?
- How is the electrical energy distributed to the other subsystems?
- How can the answers to the above questions be implemented on a printed circuit board that is within the physical constraints of the 2U formfactor?

In addition, to gap part of the bridge of discontinuity that often plagues Cubesat projects, an attempt has been made to generate and present information in such a way that it could be picked up by the next wave of students.

1.4 Thesis Outline

The explanatory approach taken here is to follow the flow of energy through the system, from the Sun, through the regulation system and via the batteries, before ending up at the subsystems.

In Chapter 2, the Sun's role as the ultimate energy source is briefly described, before moving on to the semiconductor devices that convert solar energy to electrical energy on most of CubeSTAR's surface; the solar cells. Having gotten the energy aboard, Chapter 3 describes the EPS itself: the maximum power point tracking, the Lithium Iron Phosphate batteries, and system reliability are keywords here.

Arriving at Chapter 4, the background information from the previous chapters is now crystalized into a system design which is described here, but also spills into Chapter 5 which deals with the actual printed circuit board design.

The testing that have been done on various parts of the system is presented in Chapter 6, while Chapter 7 concludes the thesis with a few words on what this all means and how it can be used by future CubeSTAR worker bees.

Chapter 2

Harvesting Energy from the Sun

The CubeSTAR satellite will harvest energy from the Sun through the use of solar cells. This chapter deals with the theoretical background for the conversion from solar- to electrical energy in the context of the chosen triple-junction solar cells.

2.1 Solar Energy

To evaluate the amount of available solar energy for a spacecraft, the spectral irradiance of the Sun at the Earth's mean distance, or one *astronomical unit* (1AU), is often used. Referring to the lack of attenuation in the vacuum conditions above the Earth's atmosphere, this spectrum is known as the Air Mass Zero spectrum, or AM0 for short.

While the atmosphere absorbs certain wavelengths under terrestrial conditions, the solar spectrum in space closely matches a black-body radiator at 5780K. These similarities are illustrated in Figure 2.1 where the Air Mass Zero (AM0) reference[1] spectrum is plotted together with the AM1.5 (terrestrial conditions) and 5780K black-body plots. The peak of the spectrum,

[1]The data for the reference spectrum (ASTM E-490) can be found here: http://rredc.nrel.gov/solar/spectra/am0/

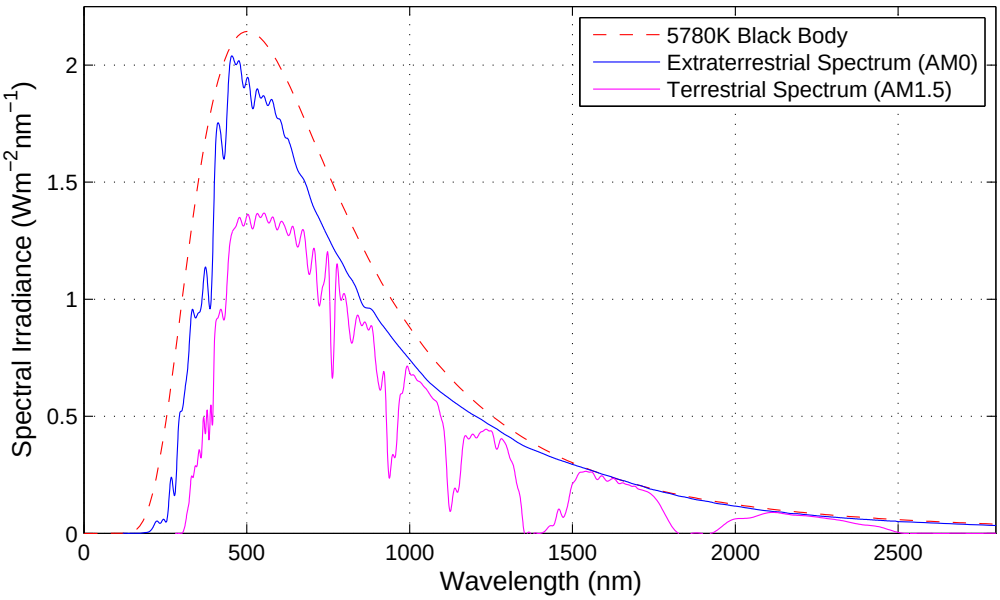


Figure 2.1: While the terrestrial solar spectrum suffers from absorption of various wavelengths in the atmosphere, the conditions in space closely matches those of a black body radiator.

which coincidentally is the part our eyes evolved to see, is at wavelengths from 400 to 700 nanometers.

The photon energy E, in units of electron volts (eV), is

$$E = \frac{hc}{\lambda} \tag{2.1}$$

where $h = 4.135 \times 10^{-15}\text{eV} \cdot \text{s}$ is the Planck constant, $c = 2.998 \times 10^{8}$ m/s is the speed of light in vacuum, and λ is the photon wavelength in meters. As Figure 2.1 shows, at shorter wavelengths (i.e., higher energies) the photon density drops off abruptly, and the Sun emits very little electromagnetic radiation below wavelengths of about 300 nm. In other words, almost no photons carry an energy greater than

$$E_{max} \approx \frac{1240 \cdot \text{eV nm}}{300 \text{ nm}} \approx 4 \text{ eV}$$

A more practical quantification of the incoming solar energy can be found by integrating the energy under the AM0 curve in Figure 2.1. The result is known as the *solar constant* and is a measure of flux, i.e. the amount of incoming electromagnetic radiation per unit area that would be incident on a plane perpendicular to the rays. Ignoring the small (less than 7%) annual variations due to Earth's varying distance to the Sun, the solar constant has been measured to be roughly 1366 Watts per square meter, or 136.6 mW/cm^2.

2.2 From Solar- to Electrical Energy

An incoming solar energy of 1366 watts per square meter would not be of much use without a way to convert it to electrical energy. This job is typically done with solar cells. Although the *photovoltaic effect* was first recognized by French physicist A. E. Becquerel in the year 1839, it wasn't until 1883 that the first solar cell was built by Charles Fritts. His new invention was capable of transforming 1% of the incoming solar energy to electrical energy.

Fast forward 130 years, and luckily, CubeSTAR is mounted with solar cells that convert solar energy to electrical energy with an efficiency of around 28%. The chosen cells for CubeSTAR are Spectrolab's Ultra Triple Junction (UTJ) solar cells, shown in Figure 2.2. For easy reference the most important parameters from the UTJ datasheet are re-hashed in Table 2.1. The meaning and significance of these parameters are explained in Section 2.4.

The cells are delivered in an assembly of solar cell, interconnects and coverglass, known as a CIC, and are approximately 160 microns thick with an area[3] of 26.62 cm^2. The rear side is mounted with a silicon bypass diode that sits anti-parallel relative to the solar cell's anode-to-cathode direction. The role of the bypass diode is to prevent a partially shadowed or damaged individual CIC in a series string from being forced into reverse bias.

When sunlight hits these solar cells, one of three things happens:

Table 2.1: Key parameters from the UTJ datasheet, AM0, 28°

Parameter	**Value**	**Description**
J_{sc}	17.05 mA/cm^2	Short-circuit current density
J_{mp}	16.30 mA/cm^2	Current density at the MPP
V_{oc}	2.665 V	Open-circuit voltage
V_{mp}	2.350 V	Voltage at the MPP
Cff	0.84	Fill factor
η	28.3%	Efficiency
j_{sc}	5 μA/cm^2/°C	Temperature coefficient for J_{sc}
j_{mp}	1 μA/cm^2/°C	Temperature coefficient for J_{mp}
v_{oc}	-5.9 mV/°C	Temperature coefficient for V_{oc}
v_{mp}	-6.5 mV/°C	Temperature coefficient for V_{mp}

1. The photon passes straight through the material.
2. The photon is reflected off the surface.
3. The photon is absorbed in the semiconductor material.

Reflection is related to unfavorable incident angles and surface coating of the cells, and will not be dealt with here. To understand why the solar cell might be transparent to certain photons while absorbing others, a short de-tour into the basic workings of semiconductors is necessary.

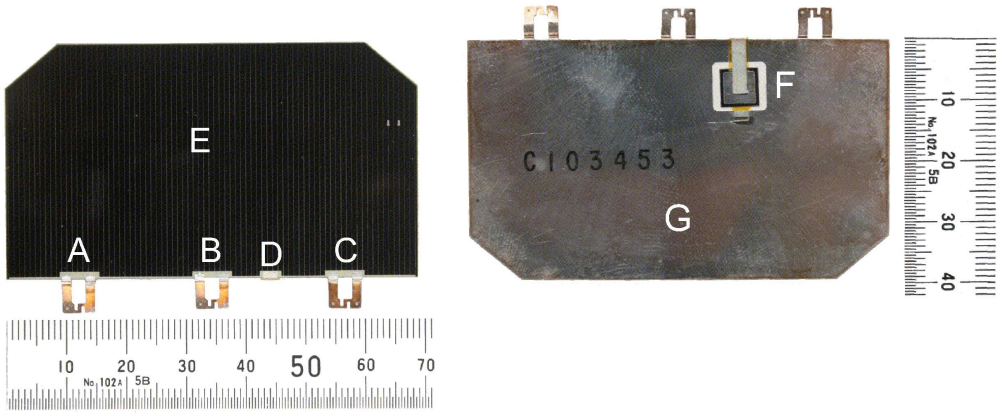


Figure 2.2: Front and rear sides of a CIC from Spectrolab showing the cathode interconnects (A,B,C), the anode (D) of the monolithic bypass diode, the parallel electrodes (E) that terminate at the cathodes, the cell's anode (G), and the integral monolithic bypass diode (F) which protects the cell in the case of reverse bias.

2.3 Semiconductor Basics

In an isolated atomic structure there are discrete energy levels associated with each orbiting electron. When two atoms form a molecule, their atomic orbitals combine and produce a number of molecular orbitals. As the number of atoms increase to form solids, the number of molecular energy levels (orbitals) increase to the point where it is natural to speak of continuous *bands* of energy instead of discrete energy levels. This electronic band structure, which is due to diffraction of the quantum mechanical electron waves in the periodic crystal lattice, is the underlying determinant of a material's electrical properties.

There are two bands of special interest to us. The *valence band* which is the highest occupied band, and the *conduction band* which is the lowest unoccupied band. The energy gap, E_g, between these two bands is known as the material's *band gap*. Suffice for our discussion, it is a measure of the amount of energy required to free an outer shell electron from its orbit around the nucleus to a free (conducting) state.

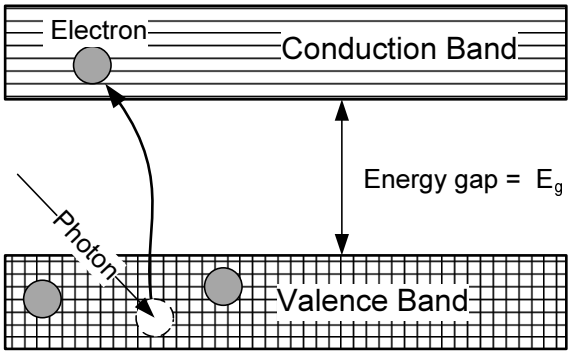


Figure 2.3: The photon energy must be greater or equal to the bandgap in order to free an electron from the valence band and make it a mobile charge carrier.

In fact, a semiconductor is (arbitrary) *defined* as a material with $0 < \mathrm{E_g} < 3$ eV at a temperature of 300K, while insulators and conductors are defined as materials with $\mathrm{E_g} > 3$ eV and $\mathrm{E_g} = 0$ respectively. The voltage across a semiconductor and the band gap energy E_g are closely related through [9, p. 11]

$$V_D \approx \frac{E_g}{q} - 0.4V$$

where we have divided by the elementary charge q in order to get the right units. Table 2.2 lists the band gap energies[9, 20] of a few well-known semiconductor materials[2].

Table 2.2: Required energy to ionize different semiconductor materials.

Semiconductor Material	**Bandgap** E_g	**Voltage** V_D
Silicon (Si)	1.1 eV	0.7V
Germanium (Ge)	0.67 eV	0.27V
Gallium Arsenide (GaAs)	1.41 eV	1.01V
Gallium Indium Phosphide ($\mathrm{GaInP_2}$)	1.85 eV	1.45V

Triple junction solar cells are, simply put, constructed by stacking three

[2] At a temperature of T = 300K

different semiconductor materials on top of each other. If a photon has insufficient energy to knock loose an electron in a layer, it will simply pass through to the next layer. In the cells from Spectrolab, the top $GaInP_2$ layer is transparent to all but the most energetic photons in the ultraviolet and visible part of the spectrum. The second GaAs layer absorbs near-infrared

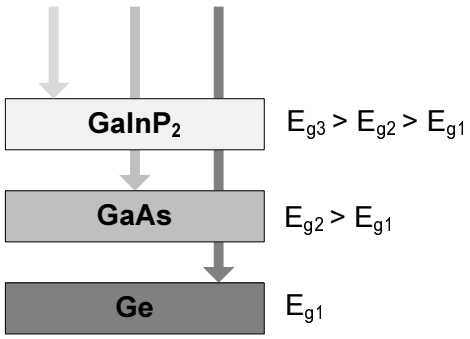


Figure 2.4: The collective ability of triple junction solar cells to absorb light of different wavelengths is the main reason for their superior efficiency compared to conventional solar cells.

light while the bottom Ge layer absorbs all the lower photon energies in the infrared that are above 0.67 eV. By combining semiconductor materials with different bandgap energies this way, higher conversion efficiencies are possible due to the stack's collective ability to match the solar spectrum.

When a layer *does* absorb photons, the solar energy excites electrons into the material's conduction band where it is free to sign up to do work as electrical current. How this photo-generated current and voltage behaves electronically is the topic for the next section.

2.4 Electronic Behavior of a Triple Junction Solar Cell

To understand the electronic behavior of the triple junction solar cells, the equivalent[3] circuit in Figure 2.5 is used. Although the use of three diodes

[3]The monolithic bypass diode is not included, as it does not affect normal operation.

give a better feel for the higher output voltage of triple junction cells, its use here is more illustrative than practical, as it unnecessarily clutters the equations without the benefit of increased information output from the model. Therefore in the following the diode string will be treated as a single diode.

All plots in this section were created with the SPICE model from Appendix B. Although the model ignores more exotic effects such as the parallel non-ohmic current paths caused by recombination (which requires a second parallel diode in the model), it was constructed in such a way as to match the output characteristics of Spectrolab's cells as closely as possible.

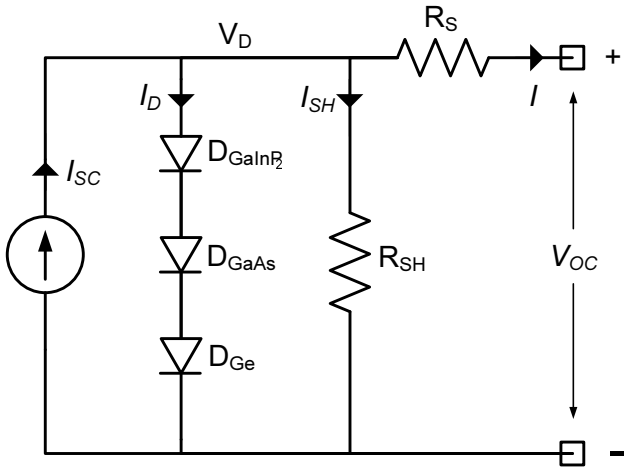


Figure 2.5: An equivalent circuit of a triple junction solar cell. The monolithic bypass diode (not shown) has its anode connected to the negative terminal and its cathode to V_D.

With the simplifications made, the cell acts as a constant current source shunted by a diode. The internal resistance to the current flow, represented by a lumped resistor R_S, is primarily caused by the resistivity of the semiconductor material, but also from the metal grid and contacts on the cell. The other (shunt) resistor R_{SH} represents the leakage current across the junction, and for a high-quality cell we have that $R_{SH} >> R_S$.

2.4.1 Short-Circuit Current

From Kirchoff's Current Law it's clear that the output current I from the cell is

$$I = I_{sc} - (I_D + I_{SH})$$

where the photo-generated current I_{sc} is proportional to the illumination, $I_{SH} = V_D/R_{SH}$ is the current through the shunt resistor, while the current I_D through an isolated photocell is given by the Shockley diode equation;

$$I_D = I_0 \left[\exp\left(\frac{qV_D}{nk_BT} \right) - 1 \right]$$

where q is the elementary charge, V_D is the voltage across the diode(s), k_B is the Boltzmann's constant, T is the absolute temperature, $n = 1$ is the ideality factor for an ideal diode, and finally the temperature- and area (A) dependent reverse saturation current

$$I_0 \propto A \cdot \exp\left(-\frac{E_g}{k_BT} \right) \tag{2.2}$$

The output current then becomes

$$I = I_{sc} - I_0 \left[\exp\left(\frac{qV_D}{nk_BT} \right) - 1 \right] - \frac{V_D}{R_{SH}} \tag{2.3}$$

where the ground leakage current represented by the last term is negligible compared to I_{sc} and I_D.

If the output is shorted, the output voltage is trivially zero, which means that the short-circuit current is the same quantity as the photo-generated current, and thus a measure of the irradiance level, as shown in Figure 2.7.

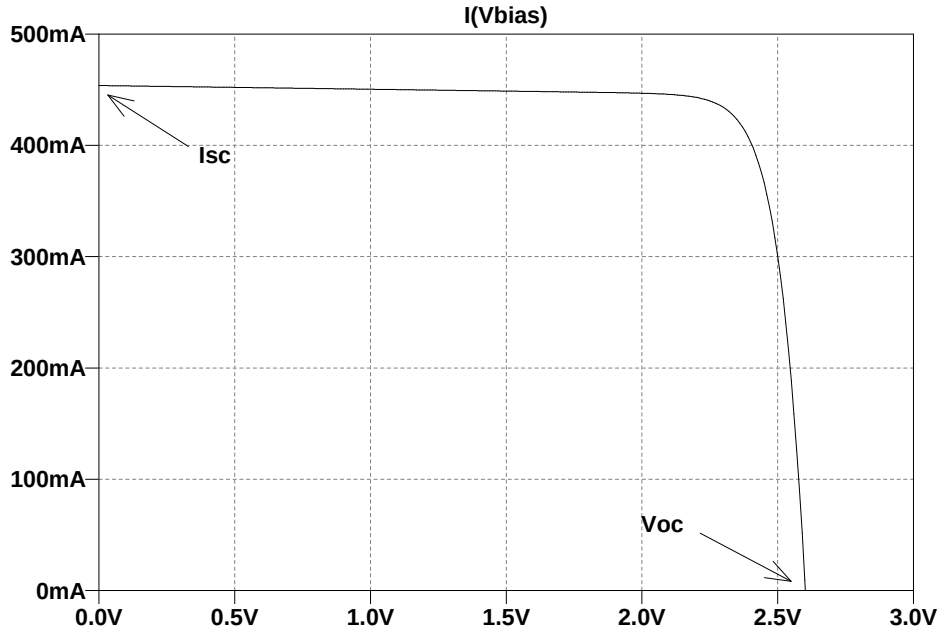


Figure 2.6: The simulated characteristic IV curve of a single triple junction solar cell, plotted here with AM0 conditions in room temperature with Rs = 50mΩ and Rsh = 300Ω.

2.4.2 Open-Circuit Voltage

The cell's open-circuit output voltage V_{OC} is the voltage, V_D, across the current source less the small drop over R_S, or

$$V_D = V_{OC} + IR_S$$

In the open-circuit condition, there can be no current flow, $I = I_{oc} = 0$ and thus

$$I_{oc} = I_{sc} - I_0 \left[\exp\left(\frac{qV_{oc}}{nkT}\right) - 1 \right] - \frac{V_{oc}}{R_{SH}} = 0$$

Ignoring the last term and solving for V_{oc} gives

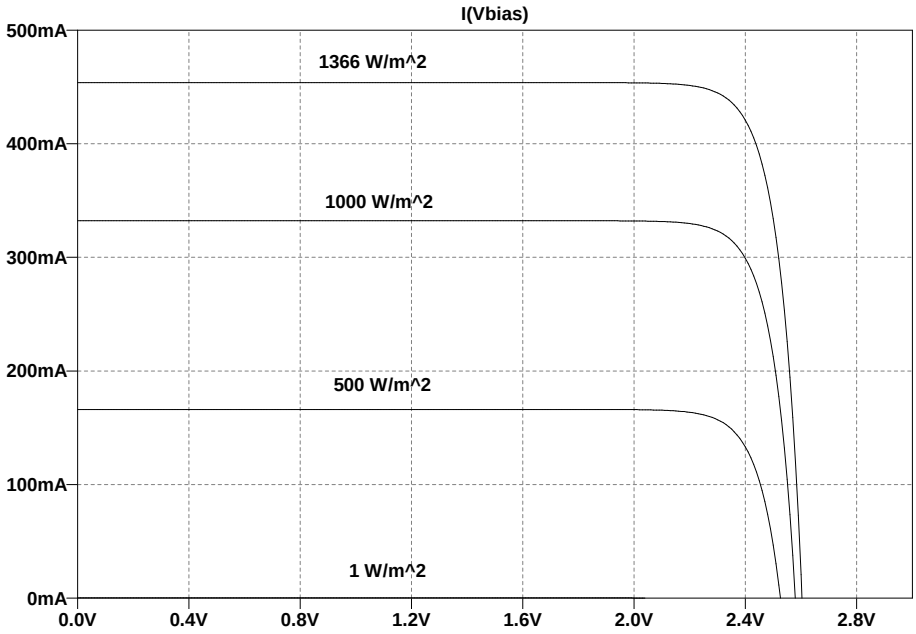


Figure 2.7: The effect of varying irradiance levels on the I-V curve.

$$V_{oc} = V_T \cdot \ln\left(1 + \frac{I_{sc}}{I_0}\right) \approx V_T \cdot \ln\left(\frac{I_{sc}}{I_0}\right) \tag{2.4}$$

where $\mathrm{V}_T = k_B T/q$ is the thermal voltage. From Equation 2.2 we have that I_0 is proportional to $\exp(-E_g/k_B T)$, so

$$V_{oc} \approx V_T \cdot \ln\left(\frac{I_{sc}}{I_0}\right) \approx \frac{k_B T}{q}\left(\ln I_{sc} - \frac{E_g}{k_B T}\right) \propto T \tag{2.5}$$

which shows that the net effect is that V_{oc} increases linearly with increasing temperature. In Figure 2.8, parameters within the SPICE model was adjusted to match the negative temperature coefficient from the datasheet, and as such, the curves represent the theoretical response to varying temperature.

Equation 2.4 also explains why the change in open-circuit voltage was so small relative to the change in I_{sc} under the varying irradiance conditions in Figure 2.7; V_{oc} depends logarithmically on the I_{sc}/I_0 ratio which in turn depends linearly on the irradiance, as we saw earlier. Replacing the current with the current *density* expression above will also reveal that V_{oc} is independent of the cell area.

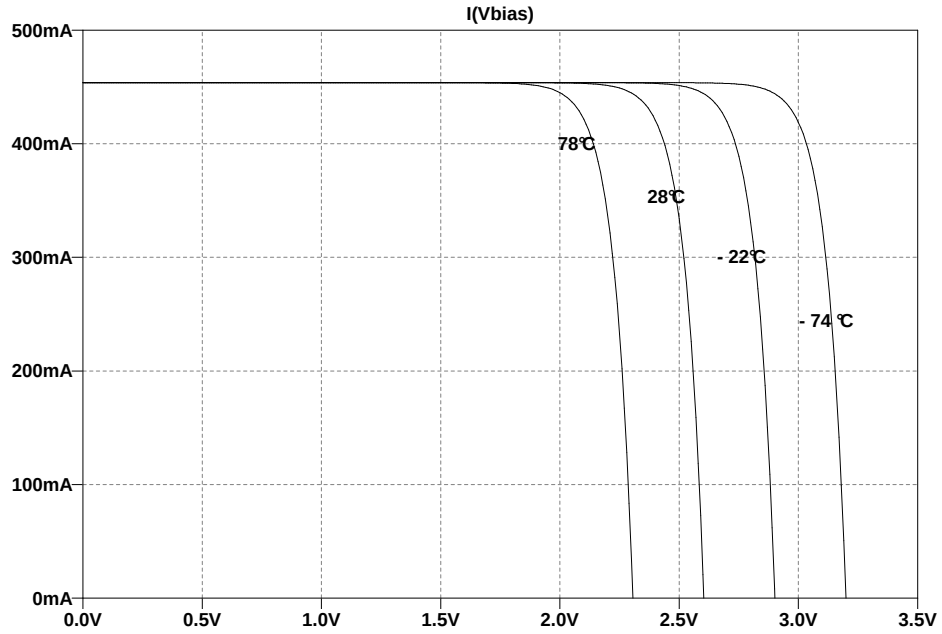


Figure 2.8: The open-circuit voltage is a strong function of temperature, while the temperature's effect on the short-circuit current is small enough to be ignored in this model. The temperature values are retro-fitted values from the datasheet.

2.4.3 Maximum Power Point

Every point on the I-V curve has a power associated with it, given by

$$P = VI = V\left[I_L - I_0\left(\exp\left(\frac{V}{V_T}\right) - 1\right)\right] \tag{2.6}$$

In order to generate power, both V and I must be non-zero, so V $\neq$ V_{oc} and I $\neq$ I_{sc}, and thus the cell must be operated between these two points on the I-V curve.

The *maximum power point* (MPP) of a solar cell is the operating point on the I-V curve where the product of the delivered output current, I_L, and the voltage across the cell, V, is at its maximum. We shall call this point

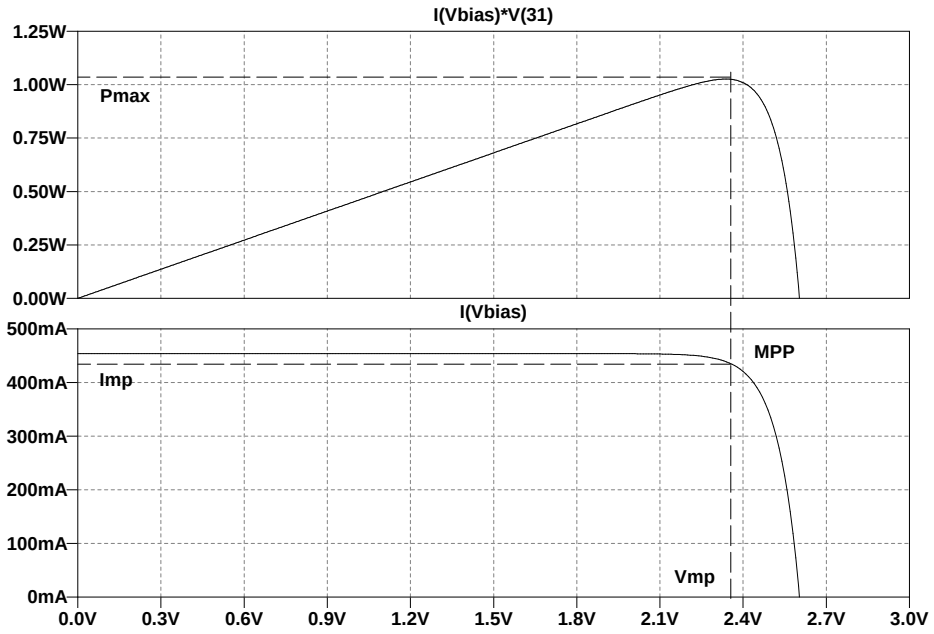


Figure 2.9: Every point on the I-V (bottom) curve has a corresponding point on the P-V (top) curve. To extract maximum power from the cell, it must be operated at the MPP.

$$P_{max} = V_{mp} I_{mp}$$

and take it to be the point where the maximum power is generated, as can be seen in Figure 2.9.

Plugging the pair of V_{mp} and I_{mp} into Equation 2.3 we find that the current at the MPP is,

$$I_{mp} = I_L - I_0 \left[\exp\left(\frac{V_{mp}}{V_T} \right) - 1 \right] \tag{2.7}$$

From Figure 2.9 it is clear that the power derivative with respect to voltage must be zero at the MPP (where $V = V_{mp}$ and $I = I_{mp}$), so

$$\begin{aligned}\frac{dP}{dV} &= I_L - I_0\left[\exp\left(\frac{V_{mp}}{V_T}\right) - 1\right] - \frac{V_{mp}}{V_T} I_0 \cdot \exp\left(\frac{V_{mp}}{V_T}\right) \\ &= I_{mp} - \frac{V_{mp}}{V_T} I_0 \cdot \exp\left(\frac{V_{mp}}{V_T}\right) = 0 \end{aligned} \tag{2.8}$$

Using the result from Equation 2.8 together with Equation 2.4 we find that the voltage at the MPP is

$$\begin{aligned} V_{mp} &= \frac{I_{mp} V_T}{I_0} \exp\left(\frac{V_{mp}}{V_T}\right) \\ &= \frac{I_L V_T - I_0 V_T \cdot \left[\exp\left(\frac{V_{mp}}{V_T}\right) - 1\right]}{I_0 \cdot \exp\left(\frac{V_{mp}}{V_T}\right)} \\ &= \frac{I_L V_T}{I_0 \cdot \exp\left(\frac{V_{mp}}{V_T}\right)} + \frac{V_T}{\exp\left(\frac{V_{mp}}{V_T}\right)} - V_T \\ &= V_T \cdot \exp\left(-\frac{V_{mp}}{V_T}\right)\left(\frac{I_L}{I_0} + 1\right) - V_T \end{aligned} \tag{2.9}$$

Equation 2.9 is a so-called *transcendent* equation whose solution is usually found by graphical or numerical methods. We will not pursue such matters here, and simply state the first of two alternative solutions given by [7, eqn. 3.13—3.15], which is

$$V_{mp} = V_{oc} - 3V_T \tag{2.10}$$

We'll let the power-discussion rest for now, and run through the rest of the datasheet parameters, before picking up the power again in Section 3.4, where the focus will be on how to actually maximize the output power and do something useful with it.

2.4.4 Fill Factor

We will now define the *fill factor* (FF) as the ratio between the actual power output P_{max}, and the product of open-circuit voltage V_{oc} and short-circuit current I_{sc}.

$$FF = \frac{V_{mp}I_{mp}}{V_{oc}I_{sc}} \tag{2.11}$$

The fill factor is a measure of a cell's energy conversion efficiency, and as seen from Figures 2.10 and 2.11, the ratio is strongly dependent on the shunt- and series resistance in the cell.

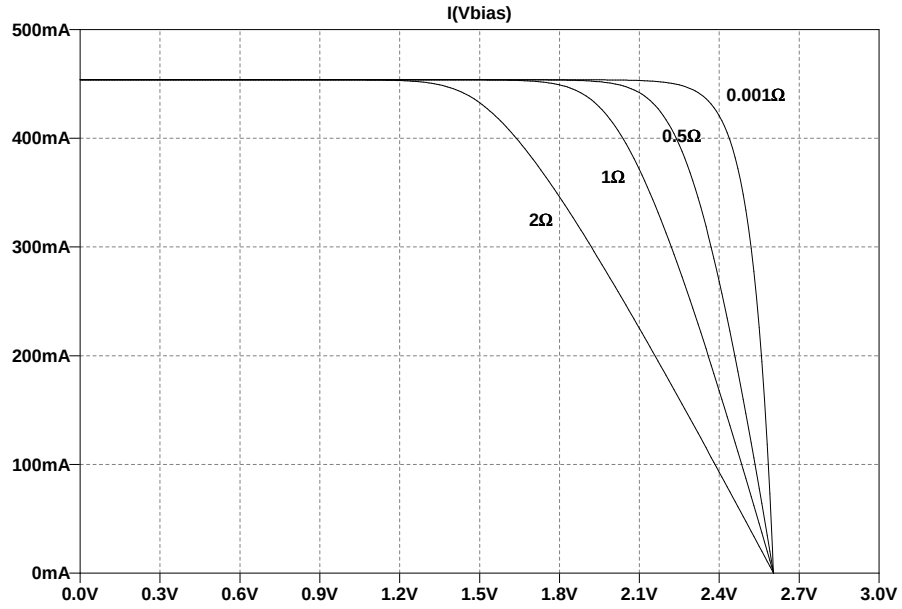


Figure 2.10: The effects of an increasing series resistance (R_S). High quality cells have low series resistance.

As an example of their impact, we can consider the case of a decrease in shunt resistance in Figure 2.11. For every new lower value of R_{SH}, the product of I_{mp} and V_{mp} decreases, while V_{oc} and I_{sc} remain constant, with a lower fill factor as a result.

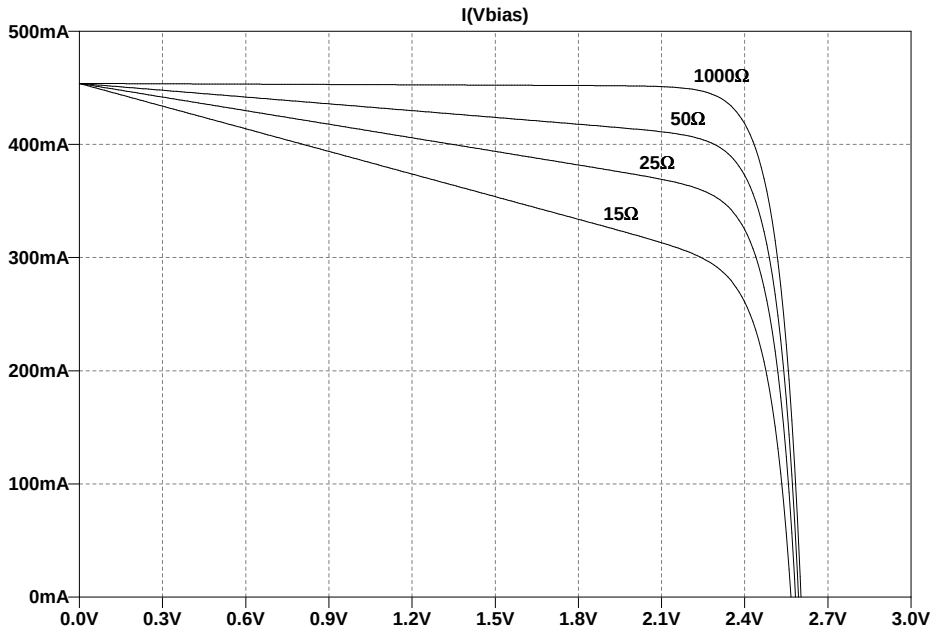


Figure 2.11: The effects of a decreasing shunt resistance (R_{SH}). High quality cells have high shunt resistance.

2.4.5 Efficiency

Lastly, there is the power conversion efficiency, which is a measure of the cell's ability to convert solar energy into electrical energy. Being the ratio of output- to input power, it is dimensionless and given by

$$\eta = \frac{P_m}{P_{in}} = \frac{V_{mp} \cdot I_{mp}}{A \cdot G} = FF\frac{V_{oc} \cdot I_{sc}}{P_{in}} = FF\frac{V_{oc} \cdot I_{sc}}{A \cdot G} \tag{2.12}$$

where $G = 1366$ W/m^2 is the solar constant and $A = 26.6$ cm^2 is the area of a single UTJ cell from Spectrolab. Note that the cell efficiency is referenced to AM0 conditions

2.4.6 Loading the Solar Cell

When a load is directly connected to a non-linear source such as the solar cell, the system's operating point is at the intersection of IV curve and the load line. For a simple linear load with constant resistance, the load line would be a straight line through origo, which, in the general case, would not intersect the IV curve at the MPP but rather at some arbitrary point on the IV curve.

In Figure 2.12, another type of load line can be seen; a *constant power* load. Among the type of loads that are characterized by such a load line, is the switching regulator[4], which aims to regulate its output at a steady voltage regardless of variations in input voltage or load current draw.

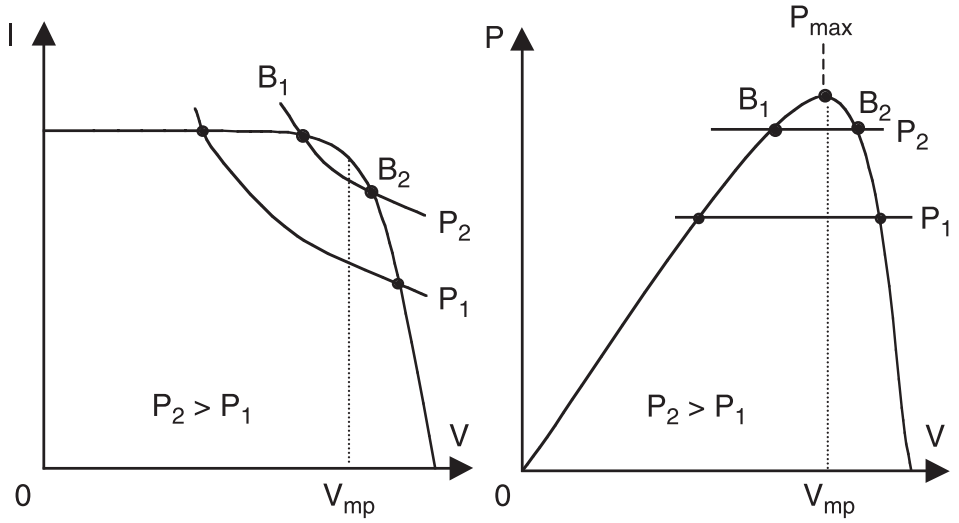


Figure 2.12: Constant power load lines (P_1 and P_2) intersect the IV curve in two places. Only B_2 is stable, and this is where the system naturally operates. *Figure reproduced from* [19].

If the power is $P = VI$ and the operating point moves away from this point,

[4]For example, if the solar cell voltage increases, the control loop of the regulator would reduce its dutycycle, which in turn would reduce the input current of the regulator. Therefor, an increase in the input voltage (i.e., solar cell output voltage) results into a current decrease, and vice versa, making the switching regulator look like a constant power load to the solar cell.

to

$$P + \Delta P = (V + \Delta V)(I + \Delta I) \tag{2.13}$$

then, by ignoring the small term, the change can be expressed as

$$\Delta P = \Delta V I + \Delta I V \tag{2.14}$$

At the maximum power point, ΔP should necessarily be zero and lie on a locally flat neighborhood, so the relationship in the limit between the dynamic and static impedance at the maximum power point, can be stated as

$$\frac{\Delta V}{\Delta I} = -\frac{V}{I} \tag{2.15}$$

When the solar cell is operated *away* from P_{max}, it has been shown in [10] that only point B_2 in Figure 2.12 is stable and any perturbation from it will generate a restoring power in the direction of V_{oc} to take the operation back to B_2. In other words, electrically stable operation of the solar array is characterized by

$$\left[\frac{dP}{dV}\right]_{load} > \left[\frac{dP}{dV}\right]_{source} \tag{2.16}$$

2.5 Angular Response of a Solar Cell

A fundamental parameter in all solar array analysis is the angle (η) between the solar panel normal vector ($\widehat{N}$) and the spacecraft-Sun vector ($\widehat{S}$). In general, the cosine of the angle between the two vectors is given by the the sum of their direction cosines. Thus, by using elementary vector identities the angle for any combination of panel normal and Sun vector can be calculated with

$$\widehat{N} \bullet \widehat{S} = \vec{n}_x \vec{s}_x + \vec{n}_y \vec{s}_y + \vec{n}_z \vec{s}_z = \cos\eta \tag{2.17}$$

This is an important result, because the amount of solar generated current from a cell is proportional to the cosine of the angle between the two vectors—or, at least up until a certain point. Beyond 50°, increased reflection causes the angular response to deviate from the cosine law, and the actual response is more accurately expressed by what is known as the *Kelly cosine*:

$$\rho = -0.369\cos^3\theta + 0.637\cos^2\theta + 0.750\cos\theta - 0.015$$

where θ is the angle between the Sun vector and the solar panel normal.

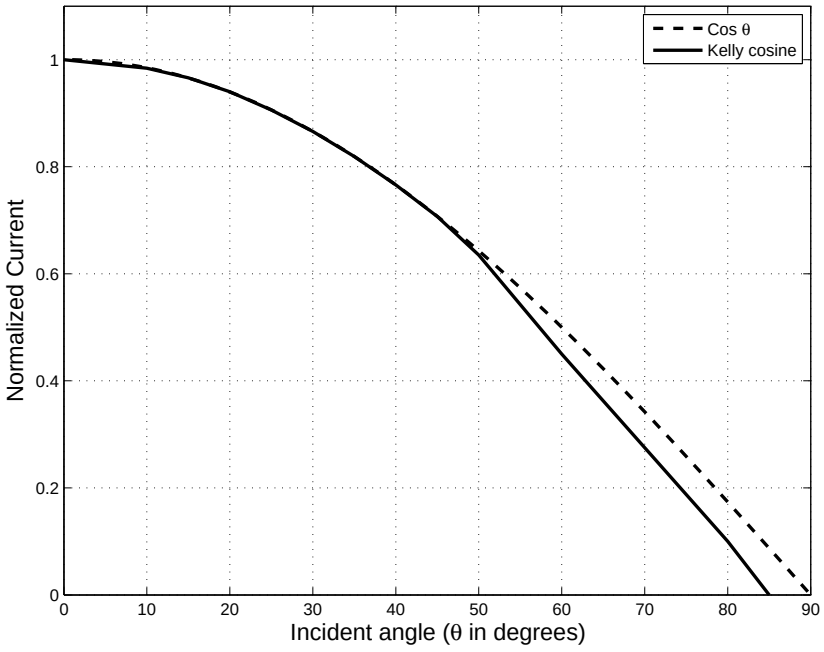


Figure 2.13: Output current from a solar cell is proportional to the cosine of the incident angle—up to about 50° at which point the Kelly cosine is more accurate. No current is produced at angles $>$ 85°.

As seen in Figure 2.13, the actual electrical output drops off slightly steeper than what the cosine function predicts, and no current is produced at angles greater than 85°. However, the deviation is not large enough to outweigh the benefits of simpler calculations, so the Kelly cosine is therefore rejected here in favor of the cosine function.

The input solar energy, I, on a spacecraft surface is given by

$$I = A \cdot K \cdot \cos\eta \tag{2.18}$$

where A is the area in m^2, K is the solar constant (1367 W/m^2), and η is the angle between the surface normal and Sun vector. With Equation 2.18 and Figure 1.1 in mind, four different cases will be considered where an attempt is made to illuminate how the spacecraft's position will affect the effective area $A \cdot \cos\eta$ and thus the power output from the cells.

2.5.1 Case 1: Minimum Sun

This case is trivial since if one of the two sides without mounted solar cells have their normal vector aligned with the Sun vector, the effective solar cell covered area is zero and no power is produced.

$$A_{min} = 0$$

2.5.2 Case 2: One side facing the Sun

If the normal vector of a surface with four mounted solar cells is aligned with the Sun vector the effective area is

$$A_{one} = 4 \times 26.6\text{cm}^2 \times \cos 0 = 106.5\text{cm}^2$$

and the produced power is

$$P_{one} = 1367\text{W/m}^2 \times 0.0106.5\text{m}^2 \times 28\% = 4.08\text{W} \tag{2.19}$$

2.5.3 Case 3: Maximum Sun

The maximum possible effective area is when two sides with mounted cells simultaneously face the Sun such that their individual projected area in the direction of the Sun equals the cosine of the angle. The total area is

$$A_{max} = 2 \times 4 \times 26.6\text{cm}^2 \times \cos 45 = 150.6\text{cm}^2$$

and the produced power is

$$P_{max} = 1367\text{W/m}^2 \times 0.01506\text{m}^2 \times 28\% = 5.76\text{W} \tag{2.20}$$

2.5.4 Case 4: Average effective area for a free-tumbling spacecraft

By letting the Sun vector lie along one of the axis, the projected area in each dimension is sufficiently expressed by the spherical coordinates for the normal vector:

$$\begin{aligned} N_x &= \sin\beta\cos\alpha \\ N_y &= \sin\beta\sin\alpha \\ N_z &= \cos\beta \end{aligned}$$

Negative values of the cosine function are discarded since a surface is only illuminated for angles between 0° and 90°. The total projected area in the direction of the Sun, A_{tot}, is found by integrating the contributions from each face over the 90° rotation about two axis, or

$$A_{tot} = \int_0^{\frac{\pi}{2}} \int_0^{\frac{\pi}{2}} \left(A_1 \sin\beta \cdot \cos\alpha + A_2 \sin\beta \cdot \sin\alpha + A_3 \cos\beta\right) d\beta \cdot d\alpha$$

where $A_{1,2,3}$ represent the areas of each of the three sides that at any given time can face the Sun.

In terms of actual solar cell area, the solar panel configuration from Figure X means one of the three areas will be zero. This fact is taken into account by letting the empty face be represented by $A_{1,2,3}$ in succession and averaging the result over the three trials. But that is equivalent to replacing each

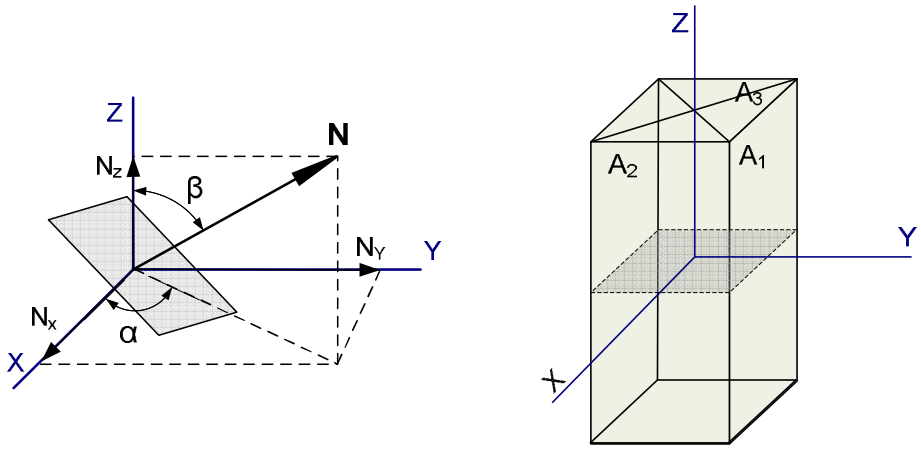


Figure 2.14: Solar panel normal vector and geometry used to estimate the output power for a free tumbling spacecraft.

individual side's area with the average area (i.e., two thirds of the area of one side) of the three exposed sides, or

$$A = A_{1,2,3} = 4 \text{ cells} \times 26.6 \text{ cm}^2 \times \frac{2}{3} = 70.9 \text{ cm}^2 = 0.00709 \text{ m}^2$$

so the numerical value for the total area becomes

$$A_{tot} = A \cdot \int_0^{\frac{\pi}{2}} \int_0^{\frac{\pi}{2}} (\sin\beta \cdot \sin\alpha + \sin\beta \cdot \cos\alpha + \cos\beta)\, d\beta \cdot d\alpha = A \cdot (2 + \frac{\pi}{2})$$

To get a feel for the angular response of A_{tot}/A, the a Matlab script was used to sum[5] up the contributions and average them over the summation limits. It should be noted however, that Figure 2.15 does not illustrate the actual angular response of the spacecraft, but rather that of the equivalent scenario with scaled average sized solar cells on all sides.

[5]The integrals were approximated with sums.

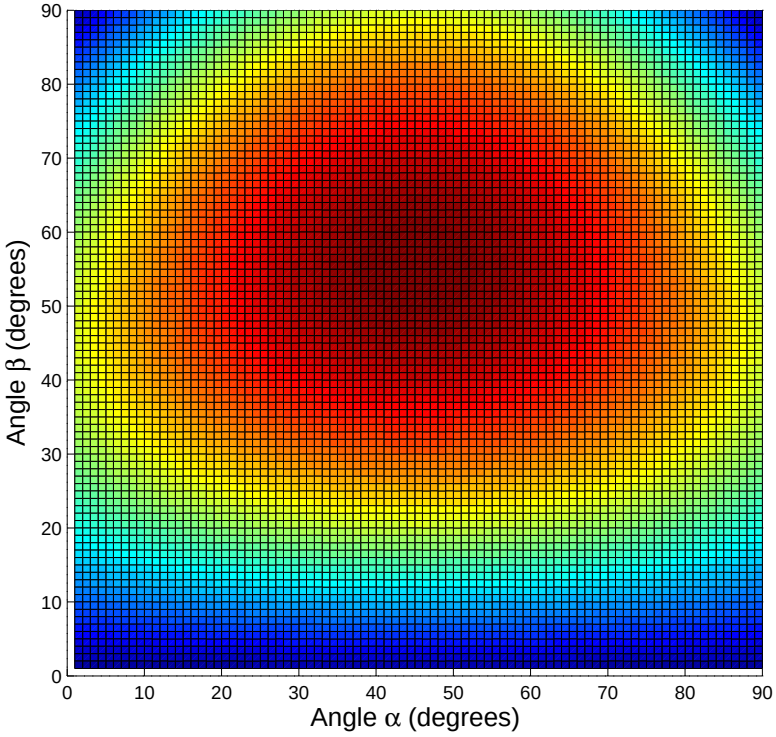


Figure 2.15: Angular response when all sides are covered by imaginary averaged sized solar cells.

The averaged projected area A_{avg} in the direction of the Sun is then found by dividing A_{tot} with the integration limits, or

$$\frac{1}{A_{avg}} = \frac{1}{A_{tot}} \cdot \int_0^{\frac{\pi}{2}} \int_0^{\frac{\pi}{2}} d\beta \cdot d\alpha \quad \Rightarrow \quad A_{avg} = \frac{A_{tot}}{\pi^2/4} \tag{2.21}$$

Thus, the numerical average projected area in the direction of the Sun for a free tumbling spacecraft is

$$A_{avg} = A \cdot \frac{2 + \frac{\pi}{2}}{\pi^2/4} = 0.0071 \text{ m}^2 \cdot \frac{8 + 2\pi}{\pi^2} = 0.01026 \text{ m}^2 \tag{2.22}$$

Finally then, the estimated average input power during the sunlit portion of the orbit is

$$P_{avg} = 1367\text{W/m}^2 \times 0.01026\text{m}^2 \times 28\% = 3.93\text{W} \tag{2.23}$$

for solar cells with 28% efficiency in AM0 conditions.

The four cases considered in this section are summarized in Table 2.3.

Table 2.3: Effective area and produced power under various spacecraft positions.

Case	**Area**	**Power**
1: Minimum area	0	0 W
2: One side only	106.5 cm^2	4.08 W
3: Maximum area	150.6 cm^2	5.76 W
4: Average area	102.6 cm^2	3.93 W

Chapter 3

Spacecraft Electrical Power Systems

3.1 Objectives

The main objective of the electrical power system (EPS) is to provide the other subsystems with a reliable and continuous power source. Typical building blocks of such a system consists of a solar array, energy storage batteries and power processing electronics, which perform:

1. Conversion from solar energy to electrical power
2. Energy storage in electrochemical cells
3. Control and regulation of the spacecraft's electrical power
4. Power distribution to other loads

Having already dealt with the solar array in Chapter 2, the main focus here will be on the second and third points, while the fourth task will be simplified by choosing batteries with an operating voltage that lies in the range of what many of the subsystems are expected to operate within. Thus, in the current setup, a de-centralized distribution scheme is used.

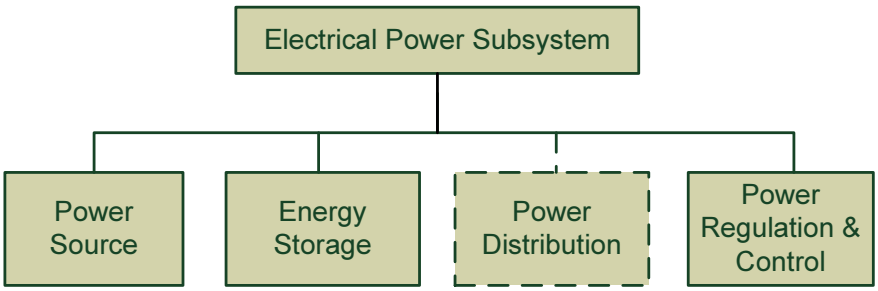


Figure 3.1: Functional overview of the EPS. A de-centralized distribution approach is proposed, where post-regulation is performed at the individual sub-system level if necessary.

3.2 Power Regulation Topology

Electrical power systems used for spacecrafts in LEO can broadly be divided into two types; the *Direct Energy Transfer* (DET) approach and the *Peak Power Transfer* (PPT) approach. All other configurations are variations, derivations or combinations of these two basic types.

Since they both have common building blocks in the form of solar arrays and power distribution units, the distinction between DET and PPT lies in how the power processing electronics conditions the solar array and storage batteries. While a PPT system aims to extract the maximum power from the solar array and hence dissipate very little power internally, a DET system employs a shunt regulator to dissipate any excessive power.

In the following sections, a brief introduction to DET systems will be given before moving on to the PPT systems which will be the main focus. Although both types allow for a regulated or unregulated power bus, only the unregulated types will be evaluated here since a de-centralized regulation approach has been chosen. There is no need to regulate the main bus, as the subsystems themselves will regulate their own supply.

3.2.1 Direct Energy Transfer (DET)

In the DET approach, the power from the solar array is directly transferred to the loads (via the distribution unit). To regulate the bus voltage at a predetermined level, a shunt regulator dissipates any excessive power as heat within the system which may require large heatsinks.

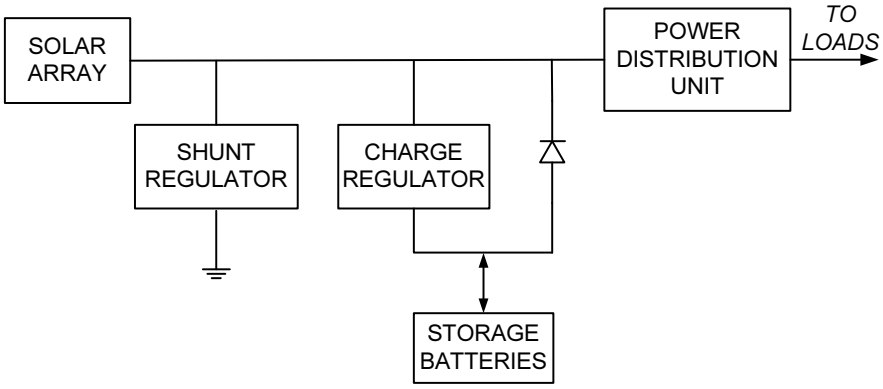


Figure 3.2: The solar array is connected directly to the distribution unit in a DET system. A *regulated* DET system is achieved by replacing the diode with a discharge regulator.

Battery charging is dealt with by a dedicated charge regulator that charges the batteries with a constant current during the sunlit portion of the orbit. For the unregulated DET system shown in Figure 3.2, the batteries discharge through a rectifying diode during the eclipse period, clamping the bus to a diode drop below the battery voltage. A regulated counterpart is also widely used where a dedicated discharge regulator can step up/down the battery voltage to match the desired bus voltage.

3.2.2 Peak Power Transfer (PPT)

In the PPT approach, a regulator is placed in series between the solar array and the loads. By taking control of the operating point on the solar array's I-V curve, the regulator tries to operate the solar array in such a way as to maximize the power output from it. This increases efficiency and simultaneously side-steps the potential thermal dissipation problems seen in DET

systems. Such a regulator is often called a *Maximum Power Point Tracking* (MPPT) regulator, and it is used to both charge the batteries and supply the loads with power.

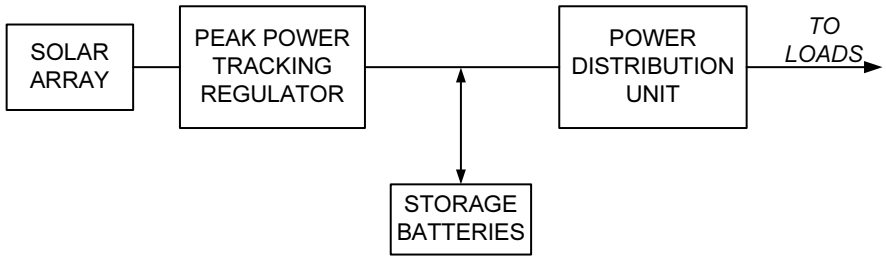


Figure 3.3: A maximum power point tracking regulator, controlling the output of the solar array, can be used to supply power to the loads and for battery charging at the same time.

When the batteries are fully charged, the tracker electronically moves the operating point away (towards the open-circuit condition) from the maximum power point, and in the process it leaves the energy from the sun as heat in the solar array itself, instead of converting it to electrical energy. Since a large portion of the incoming solar energy already is left as heat in the 28% efficient panels during normal operation, an additional few percent does not pose a thermal problem as far as the solar array is concerned. Contrast this to the DET systems, where the excessive energy is dissipated inside of the spacecraft, which may give rise to some of the thermal problems discussed in Section 3.6.4. Yet another benefit with the PPT system is that the battery is connected directly to the loads. This maximizes efficiency during the eclipse, which is when we need it.

Before looking closer at how such a PPT system works, some of the orbit constraints will be introduced. By introducing them here, they will serve to back up the decision to choose a PPT system over a DET system, and also provide the backdrop for the energy storage discussion that follows i Section 3.5.

3.3 Orbital Considerations

Since the final orbit details for CubeSTAR are presently undefined, the orbit considered here will be a typical Low Earth Orbit (LEO) Cubesat orbit, with an altitude of 600 km and an inclination of 98°. To simplify further, only the "minimum Sun" case (i.e., maximum eclipse), where the Sun-Earth vector lies in the orbit plane, will be considered here. For all but the Sun synchronous low earth orbits, the Sun will lie in the orbit plane twice a year, allowing us to calculate the orbital parameters based on a simple argument from geometry. The minimum Sun case also returns valuable information about the energy storage requirements, as it defines the minimum battery capacity needed.

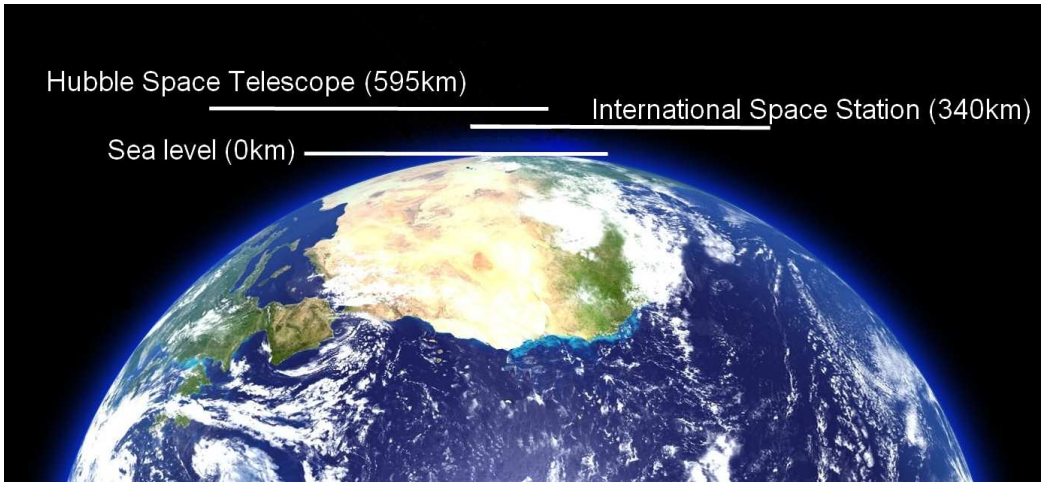


Figure 3.4: Orbital altitude put in perspective; the expected CubeSTAR altitude will be somewhere in between that of the Hubble Space telescope and ISS. By comparison, the Earth's atmosphere (in blue) reaches up to about 100km where the first 10km contain 75% of the planet's air.

3.3.1 Orbit Period and the Eclipse

The orbital period, T, is derived from Newton's formulation of Kepler's third law, but that is a derivation we shall not pursue here. Rather, we will simply reproduce the result from [15]:

$$T \simeq 2\pi\sqrt{\frac{(R_E + A)^3}{\mu}} \simeq 1.6585 \cdot 10^{-4} \times (R_E + A)^{3/2} = 96.7 \text{ min} \qquad (3.1)$$

where $\mu = 3.986005 \times 10^{14}$ m^3/s^2 $= 14.3496 \times 10^8$ km^3/min^2 is the standard gravitational parameter[1] of a celestial body, $R_E = 6378.137$ km is Earth's radius, A= 600 km is the orbit altitude for circular orbits. The sum of $R_E + A$ is the orbit semimajor axis in km.

Equipped with the orbital period, we can find the eclipse period for the minimum Sun case from the simple geometry in Figure 3.5. Since the final orbit of CubeSTAR is unknown as of today, only the maximum eclipse (or minimum Sun) case will be considered here. For all but the Sun synchronous low earth orbits, the sun will lie in the orbit plane twice a year. This situation is illustrated in Figure 3.5 and is valid for circular orbits.

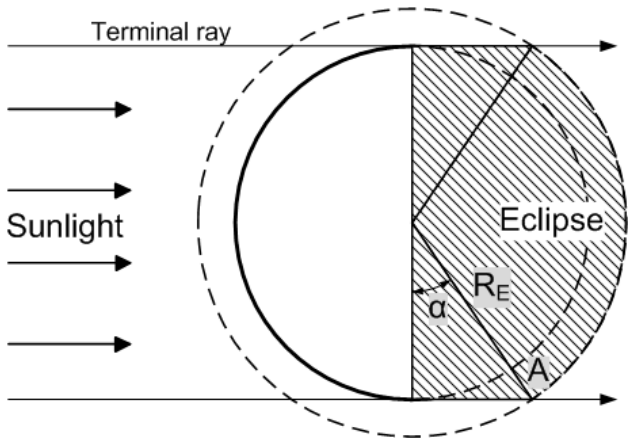


Figure 3.5: Geometry used to calculate the eclipse and sunlit periods for the "minimum sun" case when the sun is in the orbital plane.

When the Sun is in the orbital plane, the rays are parallel and the tangential terminal ray forms a right triangle where the hypotenuse is equal to the sum of Earth's radius and the spacecraft altitude. One of the legs is equal to R_E as shown, and the angle is then

[1] It's the product of Earth's mass and gravitational constant.

$$\alpha = \arccos\left(\frac{R_E}{R_E + A}\right) = \arccos\left(\frac{6378\text{km}}{6378\text{km} + 600\text{km}}\right) = 24^\circ \qquad (3.2)$$

It then follows that

$$\text{Fraction of time in sunlight} = \frac{180 + 2\alpha}{360} = 63.3\ \%$$

and

$$\text{Fraction of time in eclipse} = \frac{180 - 2\alpha}{360} = 36.6\ \%$$

For an orbital period of 96.7 minutes, the sunlit and eclipse fractions are thus 61.2 minutes and 35.4 minutes respectively.

3.3.2 Orbit Temperature

As discussed in Section 2.4, the solar array generates most power when operated under cold conditions. It is clear then from the approximate[2] expected orbital temperatures shown in Figure 3.6 that the solar array will produce most energy upon leaving the eclipse, with the temperature profile rising sharply before leveling out as the sunlight heats the panels. Only the PPT regulator is able to adjust to this temperature variation.

Assuming the temperature profile in Figure 3.6 is representable for our mission, it reveals one of the main weaknesses of the DET system approach. The moment the spacecraft leaves the eclipse, the battery voltage (and thus the main bus) will be at its lowest level, while the cold array could potentially generate its highest power level—if it hadn't been clamped by the battery. Thus, with DET, the moment the EPS needs to produce the most power is the moment it in practice can produce the least relative to its potential. In fact, its in the dual case, when the array is at its warmest (and thus V_{MP}

[2]The shape of this curve depends on panel-specific factors such as absorption coefficients (absorbtivity/emissivity), panel thickness, rigidity, etc.)

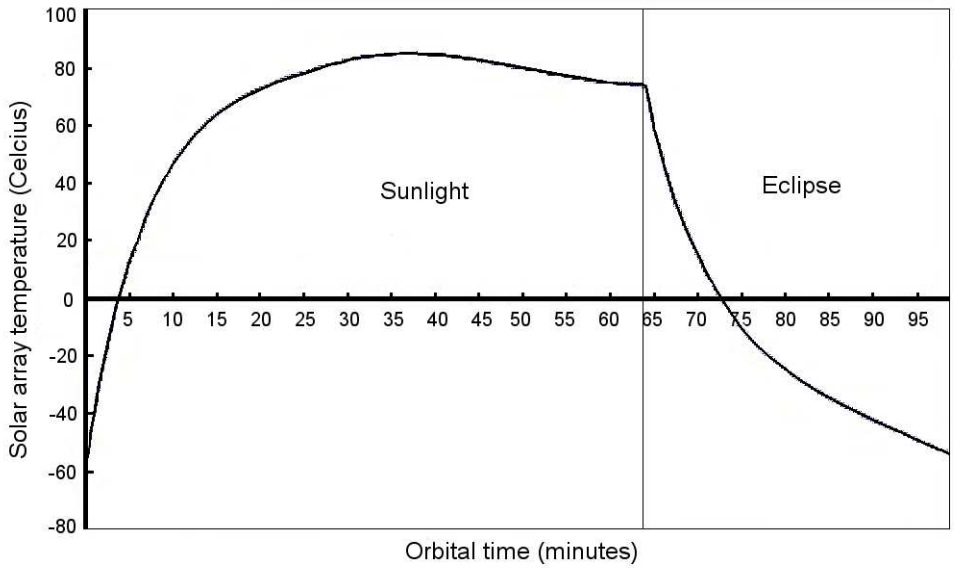


Figure 3.6: Solar array worst-case orbital temperature profile for a satellite in a sun-synchronous low-earth orbit (LEO) at an altitude of 700km.

moves towards the battery voltage) and the battery is nearly fully charged at the end of the orbit, that the DET maximizes solar array performance.

3.4 Maximum Power Point Tracking

A maximum power point tracker can be separated into two blocks; a control block and a power block. The power block, which is discussed in Section 4.3, is typically a switching DC-DC converter that steps the solar array voltage up or down (or both) in accordance to the bus voltage, while the control block, which can be implemented either in hardware[21] or software, measures the solar array parameters and sets the new operating point.

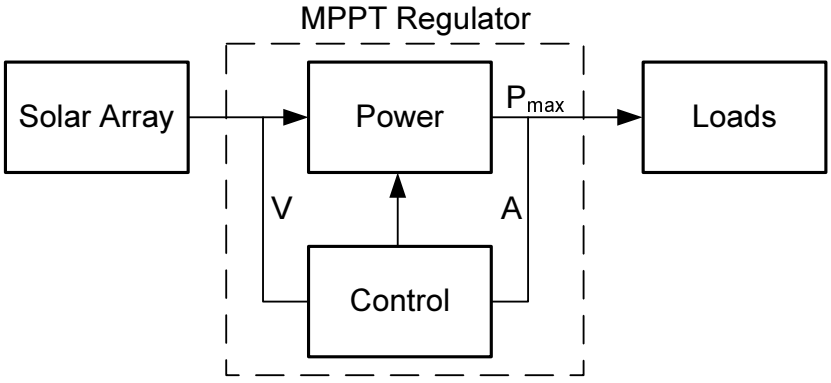


Figure 3.7: A maximum power point tracking regulator sits in series with the solar array, and tries to find the optimal operating point on the array based on the output/input current, voltage and/or a combination of the two.

Several control techniques exists, with varying degree of complexity and accuracy[23, 11]. Perhaps the simplest tracking method is the Constant Voltage method. Although considered at an early stage of development, it was quickly discarded, and its sole purpose here is to provide a backdrop upon which the actual chosen method can be compared to.

Also considered was the Incremental Conductance method, but its added complexity and slower conversion time (with a large power loss as a result[11]) made it a less attractive option than the P&O, and it will not be dealt with here.

3.4.1 Constant Voltage Method

The Constant Voltage (CV) method regulates the solar array terminal voltage and matches it to a fixed reference voltage, which is the assumed MPP of the solar array. The method is simple; just measure the array voltage, compare it to a constant reference, and use the difference (or, error) to drive a power conditioner. While this method requires very little computation, and thus finds the approximated MPP very quickly, it suffers from low accuracy due to the ignored temperature-dependent variations in the solar cell's open-circuit voltage. In fact, the "constant" voltage method, in this context, is inherently flawed, as it is based on a false assumption that the array voltage is constant. A possible solution to this problem is to use a look-up table (based on the temperature vs voltage plot in Figure 2.8) in tandem with panel temperature measurements, and adding this temperature offset to the V_{OC} measurements.

A variation of this method exists, which works by disconnecting the array at regular intervals and measuring the open-circuit voltage. The new operating voltage, i.e. the approximated maximum power point, is then set to around 75% of this value, which should be in the vicinity of the MPP.

$$V_{MPP} = k \times V_{OC} \tag{3.3}$$

where k is a constant approximately equal to 0.75. However, the penalty is that power is lost during the sampling period when the cells are disconnected, again reducing the efficiency of the CV method.

3.4.2 Perturbation and Observation Method

The Perturbation and Observation (P&O) Method is perhaps the most commonly used MPPT method due to its ease of implementation. The tracker operates by periodically incrementing or decrementing (*perturbing*) the solar array voltage and measuring (*observing*) the change in array power. If the change results in an increase in the array output power, the tracker will repeat its last action. In the opposite case, if the result is a decrease, then the next perturbation will be in the opposite direction. This situation is illustrated with the flow diagram in Figure 3.8, where P, V and I is the power,

voltage and current, while D represents the dutycycle control parameter.

One of the side-effects of this method is that the perturbation process is a continuous process—the method will perturb the operating point even if it is at the MPP. Thus, by definition, even if the P&O method finds the MPP, it will never stay there. In fact, and as we'll see later, the operating point will oscillate around the MPP with an amplitude given by the PWM resolution.

A potential weakness of the P&O method is its inability to distinguish a drop in power due to irradiance, versus a drop in power due to a perturbation. The algorithm may fail to interpret the power drop for what it really is, and continue to perturb the terminal voltage in the wrong direction, further reducing the power.

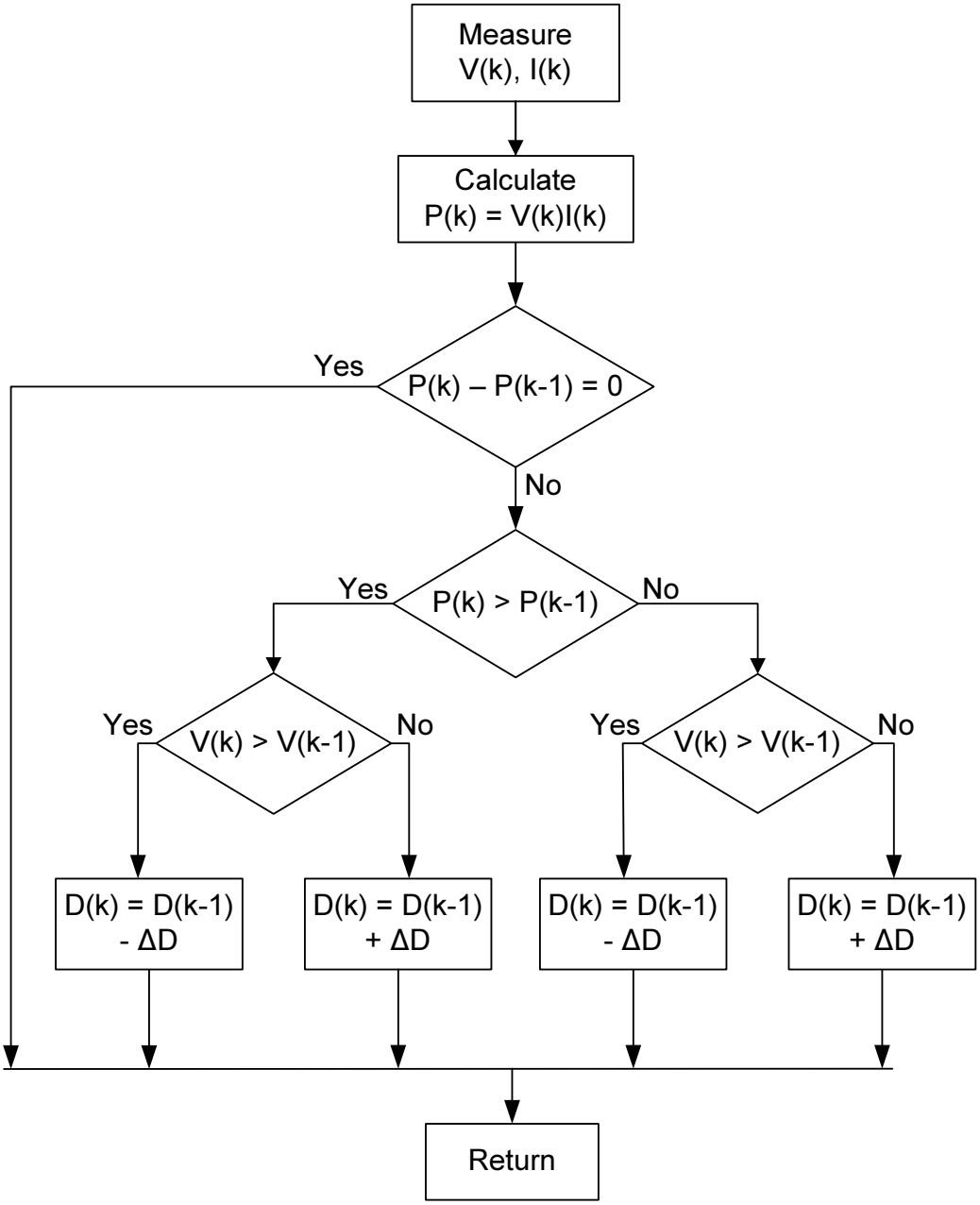


Figure 3.8: Perturb and Observe method flow diagram.

3.5 Energy Storage with Lithium Iron Phosphate Battery

Energy storage batteries provide the power source for peak-power demands and the eclipse periods. The *capacity* of a battery is the total number of ampere-hours that can be withdrawn from a fully charged cell. The product of ampere·second is the amount of electric charge transported in one second by a steady current of one ampere, so the ampere-hour, or Ah, is equal to 3600 coulombs.

Figure 3.9: A123 Systems $LiFePO_4$ cell dimensions and picture.

The chosen batteries in this system are rechargeable 1.1 Ah Lithium Iron Phosphate ($LiFePO_4$) cells from A123 Systems Inc. Based on the calculated orbital eclipse period in Section 3.3, they undergo about 15 charge-discharge cycles per day—or about 5000 cycles a year.

Frequent charge-discharge cycling traditionally has a negative impact on the capacity of a battery, however the projected cycle life of the cells from A123 Systems does not seem to suffer much from such effects, if their own datasheet represents reality. The usual remedy for the detrimental effect of cycling is to avoid deep depths of discharging (DOD) the battery. But even at full discharge (100% DOD), the $LiFePO_4$ cells seemingly perform well. Furthermore, with the short expected lifetime of the CubeSTAR mission, it is unlikely that the frequent cycling will be of any concern.

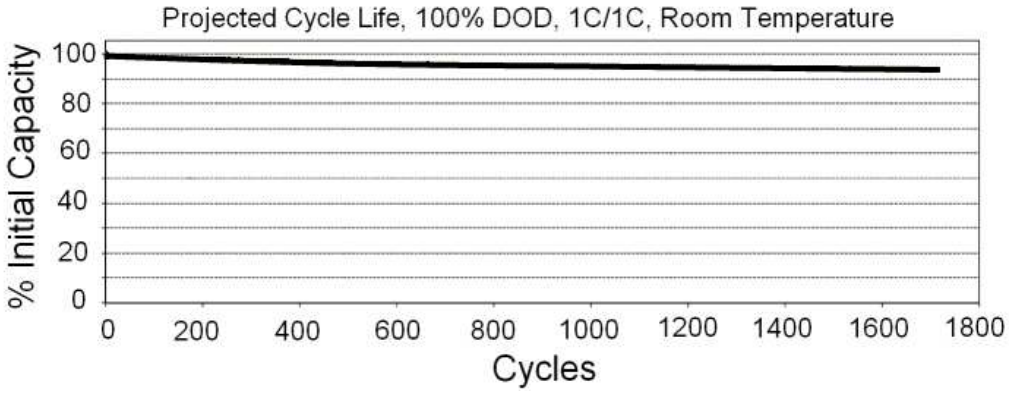


Figure 3.10: A123 Systems $LiFePO_4$ DOD vs Cycles. Picture is copied from A123 Systems Inc own website, and has been edited/emphesized.

3.5.1 Chemistry Specific Properties

Conventional lithium-based batteries generally have anodes made of carbon, while the cathode materials typically are lithium manganese oxide ($LiMn_2O_4$), lithium nickel oxide ($LiNiO_2$) or lithium cobalt oxide ($LiCoO_2$). Lithium iron phosphate ($LiFePO_4$), on the other hand, was first identified as a potential cathode material for rechargeable batteries in 1997, and—once the problem of low electrical conductivity was solved—is now being recognized for its low cost, non-toxicity, excellent thermal stability, safety characteristics, good electrochemical performance, and high specific capacity.

Since it is a lithium-ion-derived chemistry, the $LiFePO_4$ chemistry shares many of the advantages and disadvantages of the traditional lithium-ion chemistry. Two key differences, however, are the current rating and safety—both in huge favor of $LiFePO_4$. Another difference, albeit a small one, is the lower energy density of $LiFePO_4$; they exhibit a lower capacity per size or volume than for example $LiCoO_2$ based cells.

$LiFePO_4$ is an intrinsically safer cathode material than LiCoO2 and manganese spinel. The Fe-P-O bond is stronger than the Co-O bond so that when abused (short-circuited, overheated, etc.), the oxygen atoms are much harder to remove. This stabilization also helps fast ion migration. Breakdown only occurs under extreme heating (generally over 800°C), which prevents the thermal runaway that $LiCoO_2$ is prone to.

Instead of dwelling too much on chemistry, the interesting properties of the $LiFePO_4$ cells are summarized in Table 3.1.

Table 3.1: Lithium Iron Phosphate batteries from A123 Systems

Parameter	**Value**
Nominal capacity	1.1Ah
Nominal voltage	3.3V
Cut-off voltage	2.0V
Maximum continuous discharge	30A
Max charge voltage	3.6V
Volumetric Energy Density	657 kJ/L
Gravimetric Energy Density	324 kJ/kg
Operating temperature range	-30° to +60°
Core cell weight	39 grams

3.5.2 Battery Capacity Calculations

Since about one third of the orbit time is in darkness, some of the energy generated by the solar panels must be diverted to battery charging.

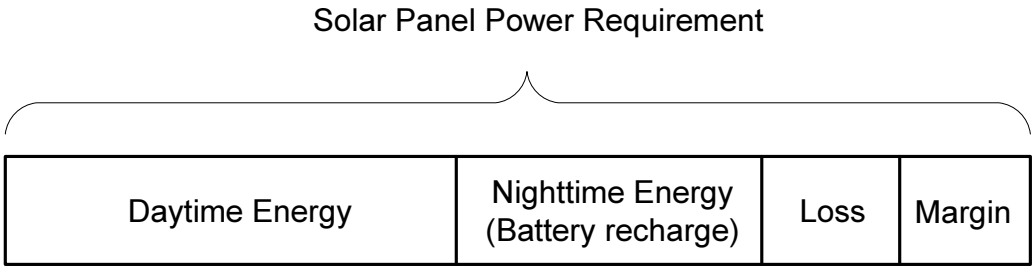


Figure 3.11: Distribution of the total energy from the solar panels.

If we assume a lossless system, the required battery capacity can be determined from an argument of energy balance; the total system energy balance

is the sum of the energy needed to charge the battery and the energy to daytime loads, expressed as

$$E_{sa} = E_n + E_d \tag{3.4}$$

where E_{sa} is the required solar panel energy, E_n and E_d are the energies consumed during night and day, including margins. But energy is the product of power and time, so Equation 3.4 can be written as

$$P_{sa}T_d = P_nT_n + P_dT_d \tag{3.5}$$

where P_{sa} is the average output from the solar array during the sunlit portion of the orbit, P_n and P_d are the average load power consumption during night and day, T_n and T_d are the eclipse and day periods in hours.

If we use the average solar panel output from Table 2.3 and the sunlit and eclipse fractions from Section 3.3 ($T_d = 1$ and $T_n = 0.6$), and assume a constant lossless power consumption $P_{nd(ideal)}$ throughout the orbit, i.e. $P_n = P_d = P_{nd(ideal)}$, then we can solve Equation 3.5 for the average power available to the subsystems throughout the orbit:

$$P_{nd(ideal)} = \frac{P_{sa}T_d}{T_n + T_d} = \frac{3.9\text{W} \cdot 1\text{hrs}}{0.6\text{hrs} + 1\text{hrs}} = 2.44\text{W}$$

Before we can take the system losses into account, it's normal to group the losses into three fractions[8]: X_{a-l} is the power transfer efficiency from the solar array to daytime loads; X_{a-b} is the power transfer efficiency from solar array to battery; X_{b-l} is the power transfer efficiency from the battery to the nighttime loads.

With losses, Equation 3.5 becomes

$$P_{sa} = \frac{P_nT_n}{X_{a-l}X_{b-l}T_d} + \frac{P_d}{X_{a-l}} \tag{3.6}$$

Equation 3.6 should be used with the *maximum* time-averaged electrical loads

(including margins) and P_{sa} is then the *minimum* required power required from the solar array.

We can then take some imaginary losses into consideration by assuming X_{a-l} = 80%, X_{a-b} = 85% and X_{b-l} = 95%. The average power available to the subsystems including loss is

$$P_{nd} = \frac{P_{sa} X_{a-l} X_{a-b} X_{b-l} T_d}{T_n X_{a-l} + X_{a-b} X_{b-l} T_d} = 1.96 \text{ W} \tag{3.7}$$

Raising the solar array to loads and battery efficiency to 90% would yield an available power of 2.15W.

It follows then that the battery energy capacity is the average nighttime power multiplied by the maximum eclipse time divided by the transmission efficiency from battery to loads:

$$E_b = \frac{P_n T_n}{X_{b-l}} \tag{3.8}$$

where E_b is the energy supplied by the battery in single nighttime cycle measured at the battery terminals.

But E_b is usually not equal to the actual battery capacity. Only a small percentage of the total battery capacity is removed at each discharge and the battery is never allowed to discharge completely. The percentage removed, or *depth of discharge* (DOD) during a discharge is

$$\text{DOD} = \frac{E_b}{E_{b(tot)}} \tag{3.9}$$

where $E_{b(tot)}$ is the battery total energy capacity in W·h. If we combine Equation 3.8 with 3.9, the minimum energy capacity requirement for the battery system is

$$E_{b(tot)} = \frac{P_n T_n}{X_{b-l} \text{DOD}} \tag{3.10}$$

Dividing through by V = P/I we're left with the ampere-hour product on the right hand side, which we recognize as the capacity of a battery. The battery capacity requirement in Ampere·hours is thus

$$C = \frac{E_{b(tot)}}{V_{bat}} = \frac{P_n T_n}{V_{bat} X_{b-l} \text{DOD}} \tag{3.11}$$

where V_{bat} in volt is the average battery discharge voltage, P_n in watts is the nighttime power required (including margins) and T_n is the eclipse period in hours.

With a 3.25V discharge voltage and 10% DOD the required capacity is

$$C = \frac{2.44 \times 0.6}{3.25 \times 0.95 \times 0.1} = 4.7\text{Ah} \tag{3.12}$$

A summary, based on the same loss "guestimate" as in Equation 3.7, of the capacity is found in Table 3.2.

Table 3.2: Battery capacity calculations summary.

Parameter	**Value**
Average orbital power available	2.0W
Required battery capacity	4.7Ah
Depth of discharge (DOD)	10%

3.5.3 Forming a Battery Pack

One of the most appealing characteristics of the $LiFePO_4$ cell for our purposes is its flat discharge voltage of around 3.25V. This suites our chosen unregulated power bus well. In fact, it's the main reason the unregulated bus was chosen in the first place. Thus there is no need to connect the batteries in series to raise the battery pack voltage. However, by connecting four 1.1Ah cells in parallel, their collective capacity is increased to 4.4 Ah, which is about the level we need, based on the calculations in Section 3.5.2.

The Electronics Lab and Mechanical Workshop at the Department of Physics (UiO) are working together on the physical design of the battery pack. An early engineering model that was made, is shown in Figure 3.12, where the pack is mounted to form the center of gravity in the structure.

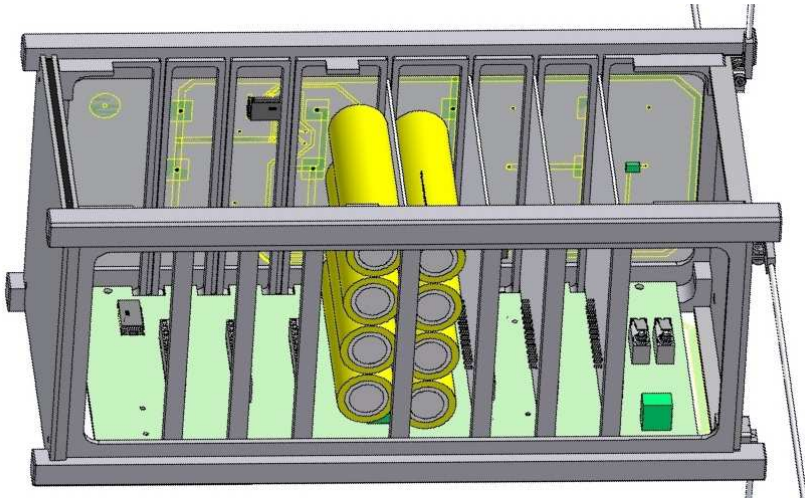


Figure 3.12: An early model made by the Mechanical Workshop at UiO, shows the proposed dual 4.4Ah battery pack.

The pack contains two redundant 4.4 Ah batteries. There are potential issues regarding equalization of the cells, but these should be minor, and should not affect the design of the charging system. Only single-cell testing[3] has been done up to this point and thus the battery pack will not be discussed further, on the assumption that the pack's behavior will not differ much from that of a single cell.

A short note on the pack's potential failure modes is nevertheless worth mentioning; The first kind is a high resistance type where the cell opens (electrically). This kind of failure is less critical in parallel configuration where the resulting decreased runtime offer a more graceful degradation of load capability versus the series case.

An electrical shorted battery cell, on the other hand, would be catastrophic, as it would drain the energy of remaining cells. Considering the extremely

[3]The pack *has* been connected to the prototype, but no systematic tests have been performed yet.

low internal resistance—and high short-circuit current—of the $LiFePO_4$ cells, this type of failure would probably end the mission in spectacular fashion. To prevent this, fuses, circuit breakers or thermal switches could be used, and the surface of the battery terminals should be insulated to avoid accidental contact with other conductors inside the structure[17].

3.5.4 Charging a Lithium-Iron Phosphate Cell

Similar to conventional lithium-based batteries, our chosen cell has two main charging phases; constant current (CC) and constant voltage (CV), as shown in 3.13. Both phases are explained below in the context of the implemented charging system.

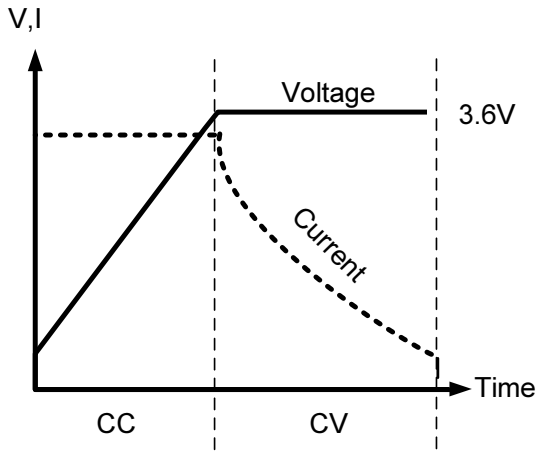


Figure 3.13: All lithium based rechargeable batteries are charged with a "constant current - constant voltage" regime as shown. The battery is said to be fully charged when the current drops below a certain threshold.

Constant Current Phase

When the battery charge regulator operates in the CC phase, it aims to transfer as much energy as possible from the solar array to the battery. To do this, it seeks out the MPP to maximize current flow to the battery, while

assuring that the charge current does not exceed the maximum allowed by the manufacturer—the last point being of little importance due to the inherent current limitations of the solar array and the recommended maximum charge current of 1.5 A. This process continues until the battery's terminal voltage reaches a preset value of 3.6V, as per the cell's datasheet.

Constant Voltage Phase

When the battery bus reaches a preset value of 3.6 V, the battery charge regulator moves away from the MPP toward the open circuit voltage, reducing current flow to the battery, and regulates the bus at a constant voltage of 3.6V. The reduction in charge current, which drops exponentially, is necessary in order to maintain the bus at a steady 3.6V during this phase of the charging.

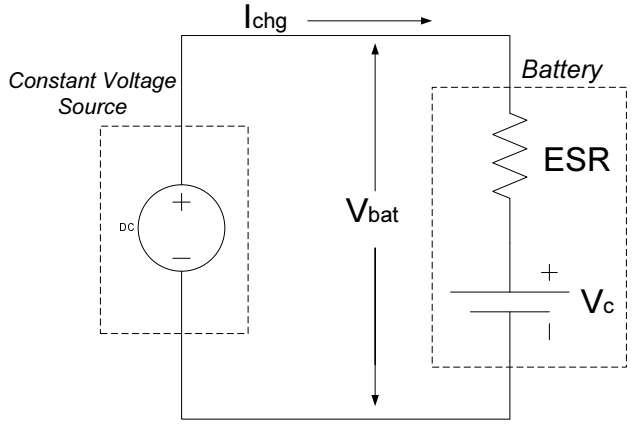


Figure 3.14: Equivalent circuit during the constant voltage charging phase.

The battery is said to be fully charged when $V_{bat} = 3.6V$ and $I_{chg} < C/10$, with C being the charge current. Or, as A123 Systems suggest; the battery is fully charged after 45 min of CV, although this assumes a 1.5A charge current.

3.6 System Reliability

The EPS has a special responsibility as the power supplier to all the other subsystems; a system failure here would end the mission. An inevitable fact of operating a power-system, or any electronics, in space, is that no repair is possible once the system is launched. This means that special attention must be paid to failure tolerance of components and the system's reliability.

3.6.1 Failure Tolerance

In [18], the author summarizes failure tolerance as follows:

> *No single component failure shall result in a significant loss of spacecraft operation.*

Which raises the question; what is a *significant* loss? The answer is mission-specific and the suggestion in [18] of 10% of the total power capability, is, although being a nobel goal, probably not a realistic one, seen from a Cubesat perspective. What the final failure tolerance amounts to will be more clear when moving closer to the final system revisions, but even in the earliest stages of development, the general approach to failure tolerance must be that it is built *into* the system itself, and not be added as a post-hoc "feature".

On a component level, where traditional space power electronics designers are restricted to high-reliability production lines, the limited budgets and shorter expected mission lifetimes of typical Cubesat missions mean that commercial off-the-shelf (COTS) components are often used. Most of the components used in this prototype have been chosen with derated voltage ratings and appropriate temperature ratings, but apart from that, no special considerations regarding component reliability has been taken.

3.6.2 Fault Protection

By continuously monitoring the subsystems, a failure protection unit detects and isolates faults that arise at each subsystem. Anomalous conditions

(e.g., an overcurrent condition) can invoke fault responses in the main on-board computer that contain preprogrammed instructions such as going into a "safe-mode", or similar. Such responses are currently outside the scope of the EPS, and should be implemented in the main on-board computer.

A failed load typically implies a short circuit. Without protection, a shorted subsystem would present a low-impedance path for the batteries—giving the $LiFePO_4$ battery-pack a golden opportunity to show-off their extremely high-current capability. The resulting problem of a drained battery pack would probably be the least of problems, considering how this current rush would cause a likely fatal system error.

3.6.3 Redundancy

Unfortunately, the tight restrictions on PCB real-estate and weight in the Cubesat satellite class, make redundancy difficult. However, by arranging the solar cells as discussed in Section 4.2, the power-system naturally divides itself into two equal and redundant parts. Although MOSFETs, diodes and the passive components can be made more fault tolerant by adding identical components in parallel, there are still unresolved issues regarding the digital control system. What happens if the microcontroller and/or it's power supply fails?

3.6.4 Thermal Design

All non-ideal components dissipate power in the form of heat which is transported away from the component by one of the three fundamental modes of heat transfer; conduction, convection or radiation. The big elephant in the room here is the lack of air convection due to the vacuum conditions in space. Although usually taken for granted, the principal means of cooling PCB mounted components in terrestrial electronics is by air convection. For the terrestrial case, the junction temperature t_j can be modeled as in Figure 3.15, with

$$t_j = P \times (R_{\Theta j-c} + R_{\Theta c-c} \mid\mid R_{\Theta c-s} \mid\mid R_{\Theta c-r})$$

where P is the power to dissipate and $R_{\Theta j-c}$ is the equivalent junction-to-case resistance while $R_{\Theta c-c,s,r}$ are the case-to-environment equivalent resistances representing convection (c), conduction (s) and radiation (r).

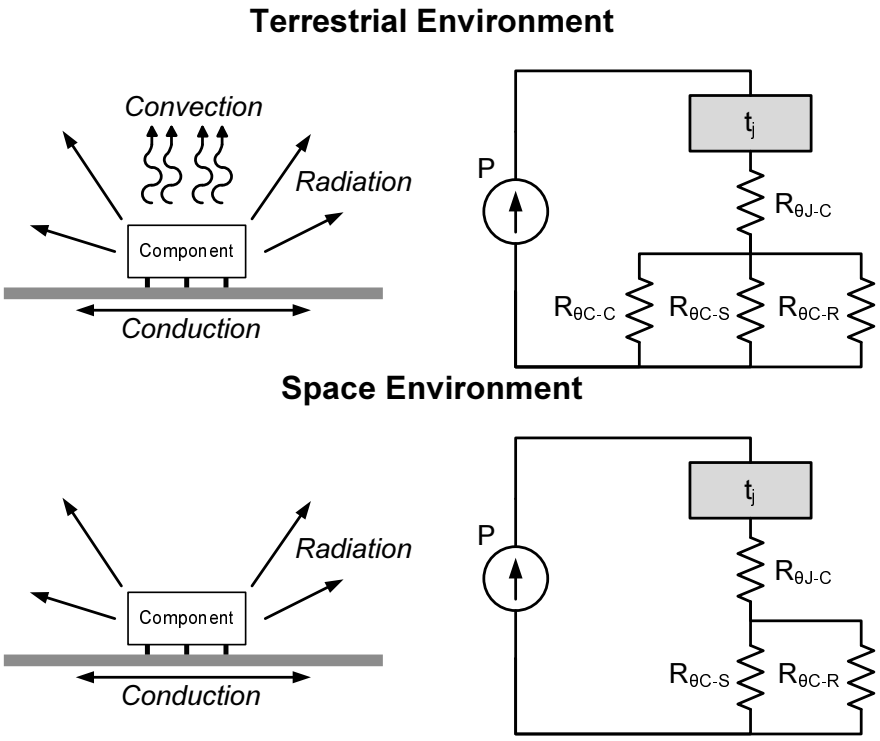


Figure 3.15: The three different means for a component to get rid of heat energy: convection, conduction and radiation. The situation can be modeled with an equivalent circuit as shown (right).

Meanwhile, in space, the only way of transferring heat energy is by radiation or conduction, increasing the junction temperature in Figure 3.15 to

$$t_j = P \times (R_{\Theta j-c} + R_{\Theta c-s} \mid\mid R_{\Theta c-r})$$

Considering how the thermal resistance associated with air convection can be between a fifth and a tenth of the radiation and conduction resistances[18], a component operating at 40°C in a terrestrial environment can experience

a case temperature exceeding 200°C in vacuum! Because of this, special attention must be paid to junction temperatures and expected dissipation when selecting components.

Chapter 4

System Design

4.1 System Overview

Two redundant regulators each serve two opposite faced sides and aim to either optimize the charge current into the batteries, or regulate the bus voltage, depending on the system state. The regulation mode is determined by the battery voltage. As long as the battery is below 3.6V, the regulator aims to maximize the power output from the solar panels. This is equivalent to maximizing the charging current, since the battery voltage is constant between the two discrete points in time where the regulator makes its decision. When (or if) the battery reaches 3.6V, the regulation mode changes to a constant voltage mode where a PID regulator regulates the battery voltage at 3.6V, thus moving the operating point of the solar cells away from the MPP.

A de-centralized post regulation scheme has been chosen for the subsystems. The reasoning behind this can be summed up by the discharge plots of the $LiFePO_4$ cells in Chapter 6. Boasting an almost flat discharge curve, the idea is that this voltage will be stable enough for some of the subsystems, and if not, it is up to each individual subsystem to step up or down the voltage as necessary. However, the subsystems will not be connected directly to the battery bus, but will be routed through an overcurrent protection switch that will shut down any ill-behaving systems.

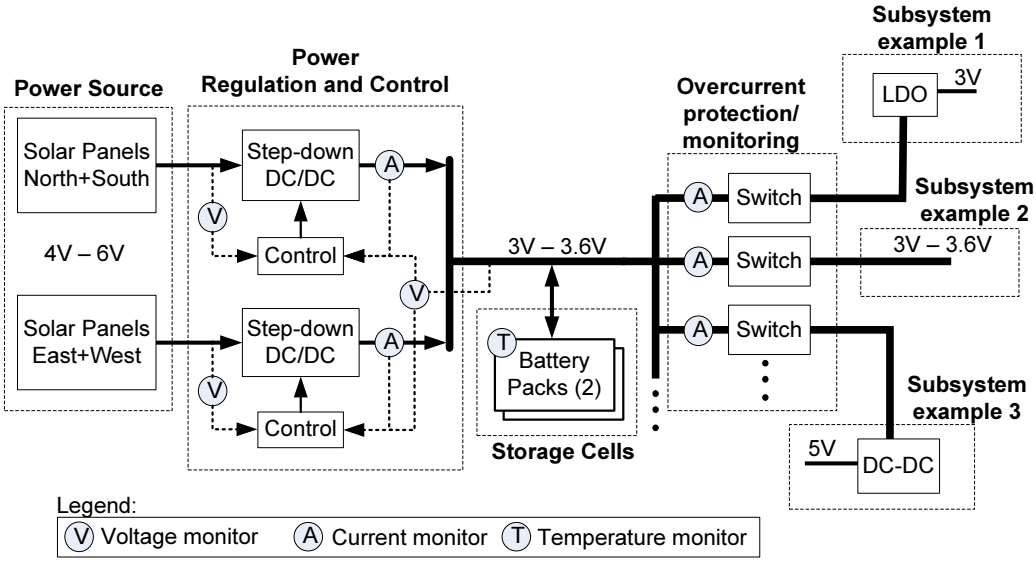


Figure 4.1: Two redundant regulators each service two opposite faced sides and aim to either optimize the charge current into the batteries, or regulate the bus voltage, depending on the system state. De-centralized post regulation has been chosen for the subsystems.

In the following sections, each of the blocks in Figure 4.1 will be explained (except the other subsystems).

4.2 Wiring Up the Solar Cells

Since only one of two opposite facing side can face the Sun at any time, they share a single regulator as shown in Figure 4.2. Two and two cells are connected in series, raising the voltage to around 5 V. With four cells on each side, two such series strings are connected in parallel, similar to the configuration of the US Air Force built PSIREX picosatellite[13]. This boosts the current to a maximum of around 900mA, but the orbital average will be well below this as the photo-generated current falls off with the cosine to the incident angle as described in Section 2.5. Each side is protected with a diode to prevent the shaded side from loading down the sunlit and power producing side. Low drop Schottky diodes are used here to minimize the $V_f I$ loss over the diode.

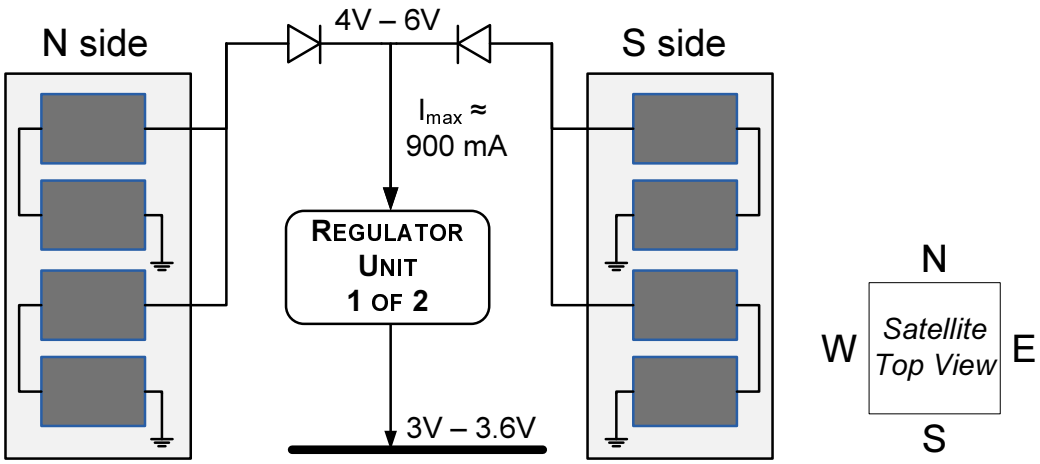


Figure 4.2: Two opposite facing sides are connected in series via protection diodes to prevent the shaded side to load down the sunlit and power producing side.

4.3 Synchronous Buck Converter

Two buck converters are used to step down the solar panel voltages to a lower battery voltage. They are electrically identical, so the following description applies to both converters.

If we forget about the synchronous rectifier (SR) for now, the basic idea is to apply a pulse-width modulated (PWM) signal to the high-side switch and average the resulting square wave with a large LC filter.

Operated as a switch, the mosfet is characterized by a very low drain-to-source channel resistance while in the *on* state, and a very high resistance—for all practical purposes, an open-circuit—while off. By driving the switch with a PWM signal, the ratio of on-time to total cycle time, better known as the *dutycycle*, can be used to set the voltage level on the output.

The relation between the dutycycle and output voltage can be understood by looking at the average voltage at the switch (SW) node in isolation. As with any waveform, the average is found by integrating the peak input voltage seen at the SW node over a switching period T_{sw}, or

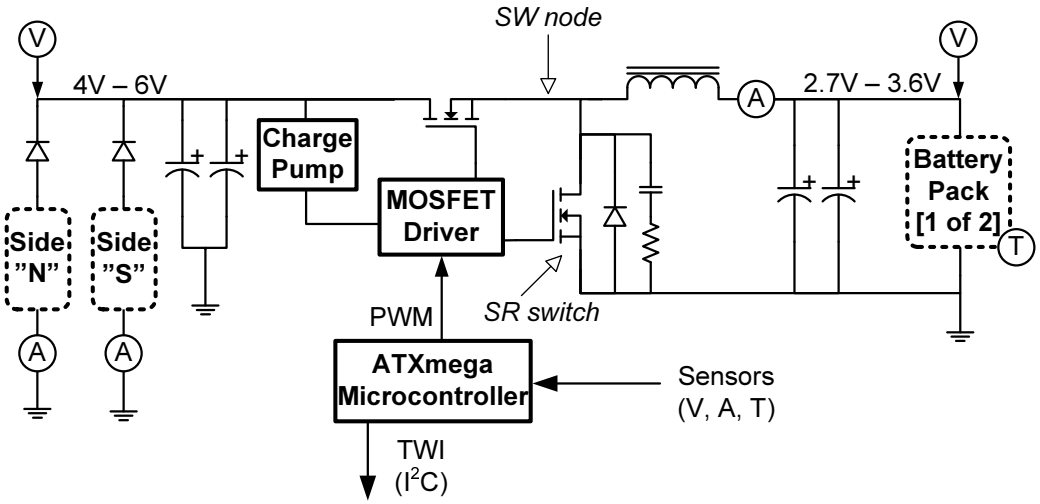


Figure 4.3: Overview over the implemented buck converter. To drive the N-type high-side mosfet, a simple charge pump is used in tandem with a driver that is fed by a PWM signal from a microcontroller.

$$V_{SW(avg)} = \frac{1}{T_{sw}} \int_0^{T_{sw}} V_{in}\, dt = \frac{T_{on}}{T_{sw}}\, V_{in} \tag{4.1}$$

where the total cycle time $T_{sw} = T_{on} + T_{off}$ can be replaced by the switch *on* time T_{on} in the integral limit, since V_{in} is zero during the switch off-time T_{off}. If the open/close frequency is fixed, we can define the dutycycle D as the ratio of on-time to total time, or

$$D = \frac{T_{on}}{T_{on} + T_{off}} = \frac{T_{on}}{T_{sw}} \tag{4.2}$$

and thus D is necessarily a number between 0 and 1. For an (ideal) buck, also known as a *step-down* converter, the output voltage is directly proportional to the dutycycle through

$$V_{out} = D \cdot V_{in} \tag{4.3}$$

which is to say that the output on a buck is always lower than the input.

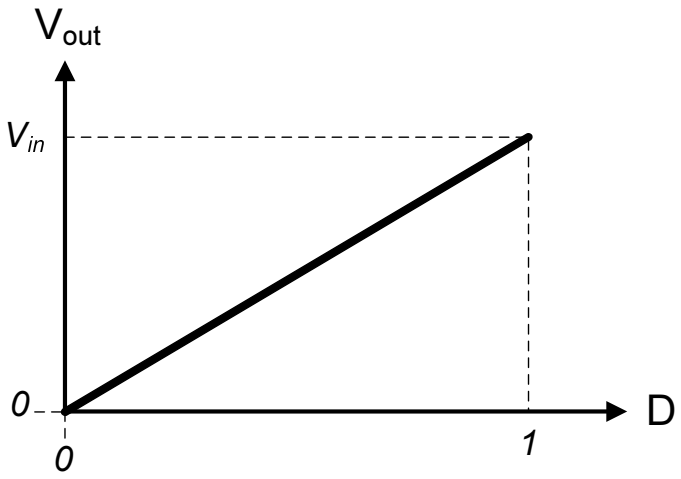


Figure 4.4: In contrast to other converter topologies the buck converter has a linear relationship between dutycycle and output voltage.

However, chopping the input voltage on and off produces a square wave output output with a high harmonic content, and is not very useful as far as providing power to the typical load is concerned. To remedy this, a lowpass LC filter is used to remove the higher harmonics and thus averages the square waveform from the switch. But the introduction of the inductor leads to a new challenge; the voltage across an inductor is related to the rate of change of current by

$$V_L(t) = L \cdot \frac{dI_L(t)}{dt} \tag{4.4}$$

and by abruptly turning off the input, i.e. a large di/dt, the voltage at SW will quickly be driven down in an attempt by the inductor to maintain the previous current. This reversal of the inductor voltage polarity is known as an *inductive kickback*. By adding a diode at the switch node, the voltage at the SW node is clamped a diode drop below ground during the off state and the inductor now has a path through which it can maintain current flow. We have a buck converter.

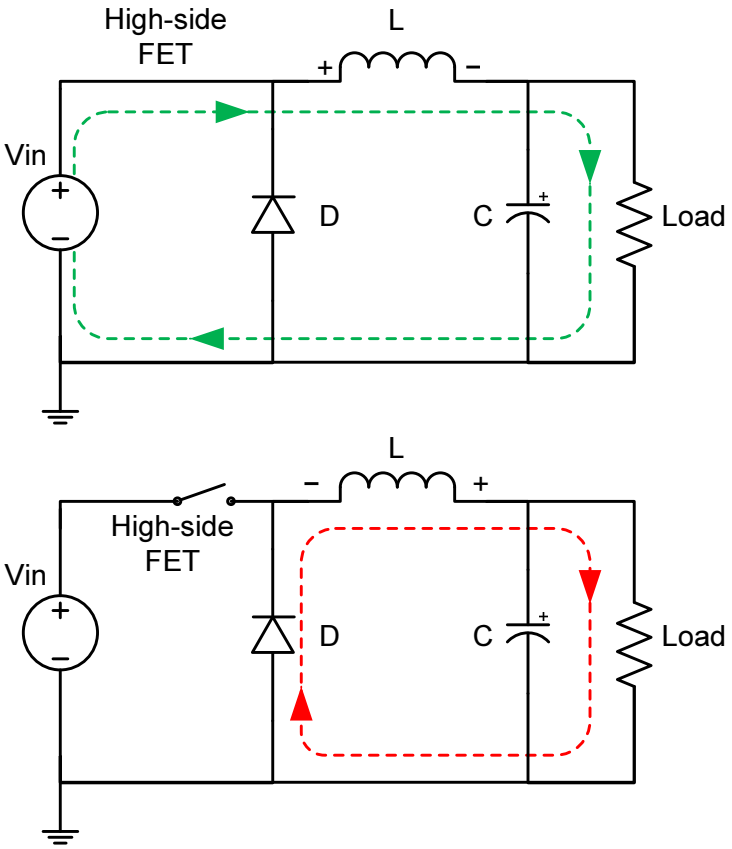


Figure 4.5: Energy is stored in the inductor during the ON state (top) as the current through it ramps up. During the OFF state (bottom) the free-wheeling diode provides a current as the inductor releases its energy into the load.

4.3.1 Adding a Synchronous Rectifier

Unfortunately, adding the free-wheeling diode comes with a prize. From the loss calculations in [16] it's clear that the diode's conduction loss ($I \cdot V_f$) represents by far the biggest loss in the converter. Even with the use of a Schottky diode with low forward drop, as the one used here, the low output voltage of the converter means that the loss in the diode remains relatively large even with the most efficient diodes. A common work-around for low voltage converters is to add a second mosfet that acts as a synchronous rectifier, and basically takes over the duties of the free-wheeling diode[16].

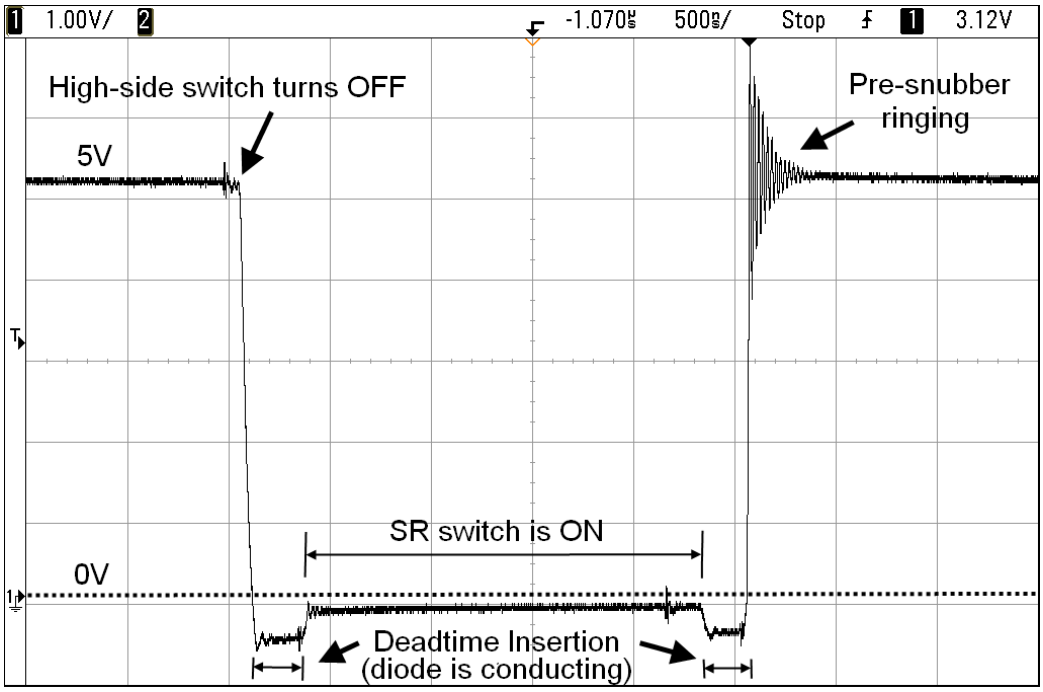


Figure 4.6: SW node voltage. The turning on of the low resistance SR channel is delayed by a small time known as "deadtime", during which the diode conducts. Evidence of the falling inductor current can be seen by the slightly positive voltage gradient while the diode/SR conducts. The snubber is discussed in Section 5.3.3

As shown in Figure 4.6, even though the diode has lost some of its prominent status, it still has a role to play at the beginning and end of each OFF state. By adding the SR and driving it with an inverted PWM signal with respect to the high-side, we now risk that both switches are on simultaneously. This would short the input through the switches to ground, and must be avoided. And it can—by adding a certain amount of *deadtime* between the switching of the two mosfets, we ensure the necessary leeway needed for safe operation. This is shown in Figure 4.7.

4.3.2 Deriving the LC Filter Component Values

The inductor voltage during the *on* period is the difference between the input and output voltage of the converter and is given by

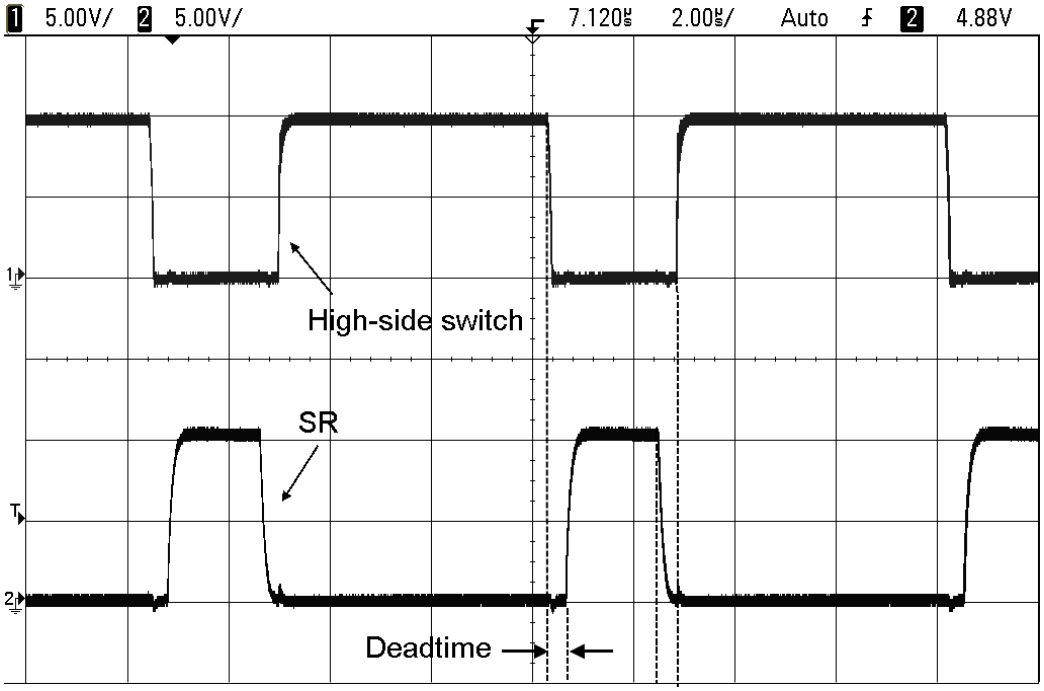


Figure 4.7: Oscilloscope screendump showing the gate drive signals on the two mosfets. A software adjustable amount of deadtime is inserted to prevent shoot-through.

$$V_L = V_{in} - v_{out}(t) \simeq V_{in} - V_{out} \tag{4.5}$$

where the last approximation neglects the relatively small time-varying ripple voltage. Equipped with this knowledge we can easily find the inductor current via Equation 4.4. A simple re-arrangement reveals how the inductor current changes with an essentially constant slope m_1:

$$m_1 = \frac{dI_L(t)}{dt} = \frac{V_L(t)}{L} = \frac{V_{in} - V_{out}}{L}$$

The situation during the *off* period, when the inductor voltage $V_L(t) = -V_{out}$, is analogous to the *on* period, and thus the slope m_2 is given by

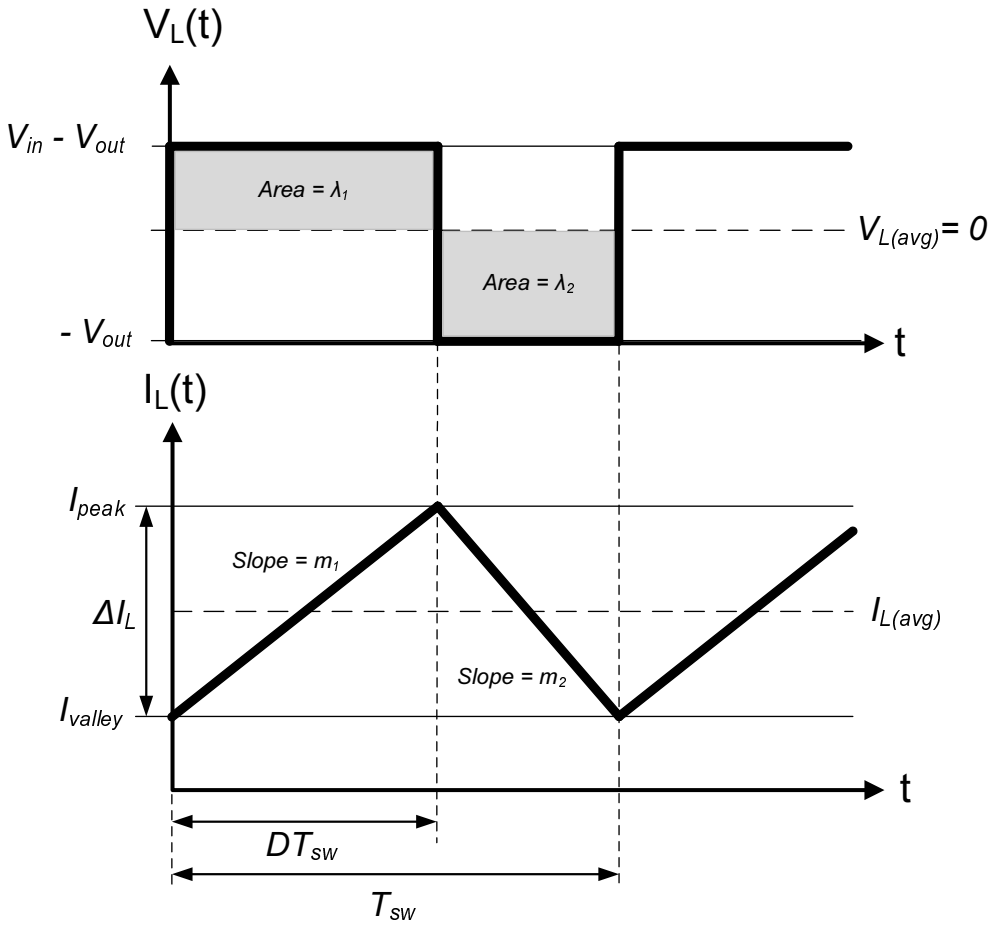


Figure 4.8: Ideal output inductor voltage (top) and current (bottom) waveforms during a switching cycle, illustrating the principle of inductor volt-second balance where $\lambda_1 = \lambda_2$ during steady-state conditions.

$$m_2 = \frac{dI_L(t)}{dt} = \frac{V_L(t)}{L} = -\frac{V_{out}}{L}$$

From Figure 4.8 the change in inductor current, or the output ripple current amplitude, is given by

$$\Delta I_L = \left(\frac{V_{in} - V_{out}}{L}\right) DT_{sw} \tag{4.6}$$

which brings us to the point where we can express the inductance as

$$L = \frac{V_{in} - V_{out}}{\Delta I_L} DT_{sw} \tag{4.7}$$

A couple of key points can be made regarding inductor selection at this stage:

- A large inductance induces a low current ripple.
- The peak inductor current is greater than the average current.
- Choosing a large inductor means a lower switching frequency can be used. This reduces overall AC losses in the circuit, but with a small penalty of a larger DC resistance in the inductor windings.
- A small inductance offers less opposition to current changes and thus quickens the load-transient response[1]
- A large inductor means continuous inductor current flow over a wider load range.

By choosing a switching frequency of 127 kHz and following the guidelines in [12] and allowing for 10% ripple current, the inductor value was chosen to be

[1] A discussion on the difference between a continuous (CCM) and dis-continuous (DCM) inductor current has been left out for the sake of brevity, but, in short, for very light loads in DCM, the output voltage to dutycycle relation is transformed to a non-linear equation and the frequency response is different. CCM is assumed here.

$$L = 100\ \mu\text{H} \tag{4.8}$$

The role of the output capacitance is to minimize voltage overshoot and ripple at the output. Real capacitors have a certain amount of parasitic equivalent series resistance (ESR) and inductance (ESL). The latter can be ignored at frequencies below around 500 kHz[12], but the ESR has a real effect, and thus the output ripple (noise) is mainly determined by the total output capacitance and the total ESR. The relationship between output capacitance C and total peak-to-peak ripple voltage ΔV is given in [5] as

$$\Delta V = \frac{\Delta I_L T_{sw}}{8C} \tag{4.9}$$

but it does not take the ESR into account.

The ratio of total peak-to-peak ripple voltage ΔV to output voltage V_{out} is given in [5] by

$$\frac{\Delta V}{V_{out}} = \frac{\pi^2}{2}\left(\frac{f_0}{F_{sw}}\right)^2 (1 - D) \tag{4.10}$$

where $f_0 = 1/2\pi\sqrt{LC}$ is the LC cutoff frequency, F_{sw} is the switching frequency and D the dutycycle. Since we already have decided on the output inductor value, we can solve Equation 4.10 for a maximum desired ripple voltage of, say, $\Delta V = V_{out} \cdot 3\%$. Doing so reveals the converter needs output capacitance of[2]

$$C_{out} = 59\ \mu\text{F} \tag{4.11}$$

The drop incurred to the ESR is

$$\Delta V_{ESR} = \Delta I_L \cdot R_{ESR} = \left(\frac{V_{in} - V_{out}}{L}\right) D T_{sw} R_{ESR} \tag{4.12}$$

[2]Considering how a large battery will load down the output line, the value is probably not critical—but we'll go by the book for now.

The chosen solid tantalum capacitors are low-ESR types, and by putting two or three in parallel the resistance is further reduced. Their collective equivalent resistance is only around 40 mΩ, and solving Equation 4.12 for V_{in} = 5.5V, V_{out} = 3V, T_{sw} = 1/127500 kHz and L = 100 μH, reveals a noise contribution from the ESR of only 5 mV.

The theoretical LC component values and other parameters related to the operation of the buck converter is summarized in Table 4.1. The actual values used on the PCB will use these values as starting points.

Table 4.1: Theoretical buck converter parameters.

Parameter	**Value**
V_{in} (max/min)	6V/4V
V_{out} (max/min)	3.6V/2.7V
Dutycycle (min/max)	45%/90%
PWM Frequency	127 kHz
Switch/SR resistance, $R_{DS(on)}$	6 mΩ
Input capacitance	57 μF
Output Inductor	100 μH
Output capacitance	59 μF
Output ripple[3]	$3\% \cdot V_{out}$
Output ESR ripple	5 mV

4.4 Digital Control Block

The digital control block is the brain, ears and eyes of the circuit and ultimately sets the dutycycle on the buck converter depending on the sampled sensor inputs. In this section the control scheme is described, and also an example of the ADC interfacing process is given.

4.4.1 System State Machine

The main system control loop is implemented as a finite state machine (FSM) that runs on a microcontroller. Currently there are three possible system states, including two different regulation modes:

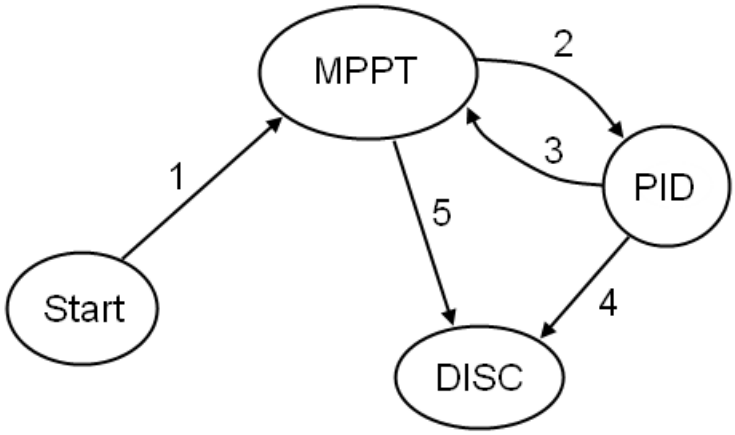


Figure 4.9: Main system loop state machine and transitions. The system will at any time be in one of the four states depending on the conditions of events.

MPPT State:
This is the main regulation mode and the implementation of the Perturb & Observe algorithm described in Section 3.4.2. By tracking the MPP of the solar panel, battery charge current is maximized. Seen from a battery perspective, this mode takes care of the Constant Current (CC) charging phase of the battery.

PID State
This is the secondary regulation mode which regulates the battery voltage at 3.6V (or any voltage desired) with a discrete PID controller that can be found in Appendix C. It takes care of the Constant Voltage (CV) phase of charging, where the end-of-charge criterion is currently set at an arbitrary minimum current. Hysteresis is built into the system to avoid oscillation between the PID and MPPT state.

DISC state
This state disconnects the solar panels from the battery bus, and leaves the system in a battery-discharge only condition. There isn't any magic going on here; the state simply discontinues PWM generation and activates internal pull-downs on the microcontroller PWM port, leaving both the high-side and SR switch in an open state. There should be an option to put the EPS system in a low power mode in this state or as a separate state, but that has not been implemented yet.

4.4.2 Event Transition Conditions

1. Power ON
2. $V_{bat} >= V_{bat(MAX)}$: Leave constant current mode and regulate battery voltage.
3. $V_{bat} < (V_{bat(MAX)}$ - hysteresis): Battery is below its float voltage, go back to constant current mode.
4. An error flag is set or the charge current $I_{bat} < I_{bat(REF)}$ has dropped to 10% (or any limit desired) of the max charge current and thus the battery is per definition fully charged.
5. An error flag is set, disconnect panels until error is resolved.

The error flags can be set to trigger on parameters such as battery over-temperature, low solar panel voltage, or any other test that is implemented.

4.5 Large Signal Analysis

While the digital control algorithm has two regulation modes, the system has four operating modes: MPPT charging, PID charging, MPPT discharging, and battery only discharging.

Since the battery is a "stiff" voltage source, system stability is guaranteed in the battery discharge only mode. When the system peak-power tracks, source line 1 must be a constant power source line as in Figure 4.10. Source line 2

is a stiff voltage source and represents the discharging battery. The battery provides an appropriate amount of current to compensate for the discrepancy between source line 1 and the load line, and thus only one equilibrium point exists.

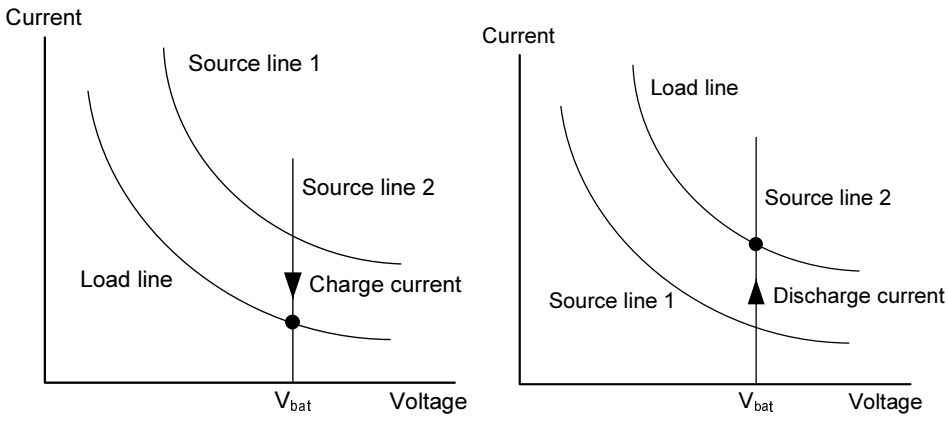


Figure 4.10: MPPT charge (left) and discharge (right) mode. When the load is heavier than the solar panels can handle alone, the battery sources the difference by discharging. When the load draw is less than what the solar panels can handle, what is left goes to battery charging.

During the MPPT charging mode the battery acts as a varying voltage current sink, and any extra power from the solar array will flow into the battery to compensate for the discrepancy between source line 1 and the load line, as shown to the left in Figure 4.10.

The change from MPPT charge to MPPT discharge mode does not cause any stability problem since the converter always tracks the MPP of the solar array[14]—the MPPT control algorithm doesn't care if the battery is charging or discharging. This is also in agreement with the tests that have been done in Chapter 6.

During the last operating mode, when the battery is charged with constant voltage, the battery is a constant power load line at any instant, as seen from the solar array[14]. Since we're moving off the MPP in this mode the solar array voltage is floating, and will settle at the only stable operating point towards the open-circuit condition, as discussed in Section 2.4.6. Thus, in this operation mode, the solar array voltage will vary with load demand.

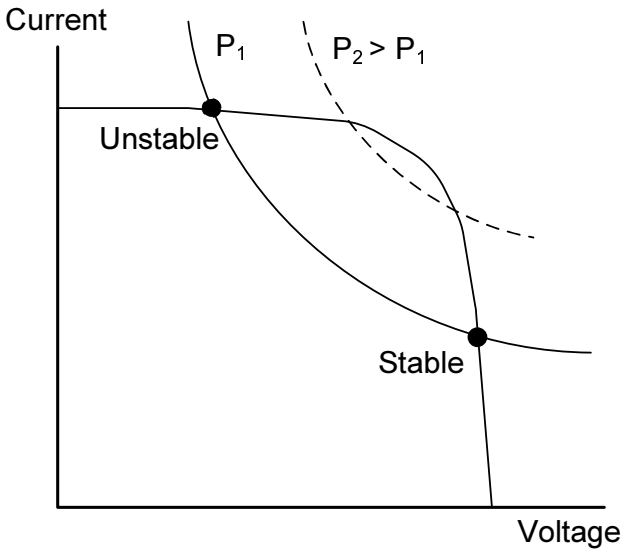


Figure 4.11: During the PID (constant voltage) mode, the battery, as seen from the solar array, is a constant power load line at any instant while the solar array voltage is unregulated and floating. Only one stable equilibrium point exists however, which is that towards the open-circuit voltage.

When the load increases to the point where the two operating points become one, the system switches to MPPT mode.

4.5.1 A Note on Small-signal Loop Response

Apart from deriving the transfer function for the output LC filter, no attempt has been made to determine the cross-over frequency, open-loop gain and phase margin for the system.

The *analog* control loop equivalent of this system has a well-documented set of equations that can be used to determine the three mentioned criterions for a stable system. However, for the digital control loop with multiple modes, the situation is not as clear. It can be said, though, that the effective lag of the digital-control loop is the combination of two effects: processing delay and update interval. The former includes the analog-to-digital conversion time and the MPPT/PID control-algorithm calculations, and the latter is the dutycycle update interval of the respective regulation states.

4.6 Sensor to ADC - Analog Interface Design

The sensors and ADC are the previously mentioned eyes and ears of the circuit, and deserve a little extra attention due to their importance. Four measurement points, shown as circles in Figure 4.3, are currently sampled and used in the control scheme. They are:

1. Input (solar panel) voltage
2. Inductor output current
3. Output (battery) voltage
4. Battery temperature[4]

In order to utilize the entire dynamic range of the ADC, the sensor output voltage span and the ADC input voltage span must be matched[4]. This is in general a two-step process involving level-shifting and amplification with op-amps as the prime candidate for the job.

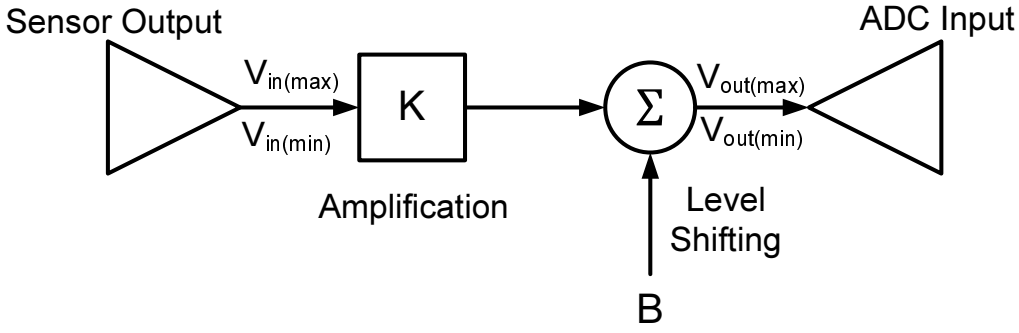


Figure 4.12: Transducer interface design (TID) block diagram.

In the first stage in Figure 4.12, the sensor's output voltage is scaled (amplified) by a constant K, mapping the output range of the sensor to the input range of the ADC. Once this is done the signal is shifted by a bias B, completing the interface circuit. The general equations for this process are linear and can be stated as,

[4]A dummy termistor located on the PCB is used here for now.

$$\begin{aligned} V_{out(max)} &= K \times V_{in(max)} + B \\ V_{out(min)} &= K \times V_{in(min)} + B \end{aligned} \tag{4.13}$$

where $V_{in(max)}$ and $V_{in(min)}$ represents the maximum and minimum output voltages from the input transducer corresponding to the max/min physical variables that the sensor measures. Similarly, $V_{out(max)}$ and $V_{out(min)}$ represent the inherent voltage range of the ADC. Since we will be using the multiplexed ATXmega ADC for all the sensed signals in the system, the output voltages needed for all the interfaces will be the same. They are given by the maximum allowed voltage reference which is given in the ATXmega datasheet as

$$V_{out(max)} = V_{ref} \leq V_{CC} - 0.6V = 2V \tag{4.14}$$

The ADC is setup with differential sampling with the negative input grounded so

$$V_{out(min)} = 0V \tag{4.15}$$

We'll apply this to the battery voltage sensor as an example, and let the rest be, as it is just a matter of rinse and repeat for the other sensor inputs.

4.6.1 Example: Sensing Battery Voltage

To provide some headroom, the battery will be assumed to operated from a low minimum of $V_{IN(min)} = 2.7V$ to a high maximum of $V_{IN(max)} = 3.7V$. The ADC input voltage range is still $V_{OUT(min)} = 0V$ to $V_{OUT(max)} = 2V$.

Using the equations from 4.13

$$\begin{aligned} 0V &= 2.7V \times K + B \\ 2V &= 3.7V \times K + B \end{aligned}$$

The first equation yields

$$K = \frac{-B}{2.7V}$$

and the second equation yields

$$2V = 3.7V \times \frac{-B}{2.7V} + B$$

and thus

$$\begin{aligned} K &= 2 \\ B &= -5.4V \end{aligned}$$

Using the general equation of an op-amp, we finally get

$$V_{OUT} = K \cdot V_{IN} + B = 2V_{IN} - 5.4 \tag{4.16}$$

which can then used to calculate the resistor values in the op-amp circuit. Referring to sheet 8 in the schematics (in Appendix), the resistor values can be found from

$$V_{OUT} = \left(\frac{R_{810} + R_{808}}{R_{808}}\right) \cdot \ V_{IN} - V_{REF}\left(\frac{R_{810}}{R_{808}}\right)$$

4.7 Telemetry and the TWI Bus

Telemetry is handled by Atmel's version of the I^2C communication bus; the Two Wire Interface (TWI) which is detailed in Appendix D. The EPS system has been configured as a slave, and as such it will respond to a request from a master device. To avoid wasting resources on polling, slave address recognition and data-complete interrupts are enabled. This ensures that the EPS system will continue its duties, even in the case of a faulty TWI master.

4.7.1 Slave Reaction to Address Packet

The slave Address/Stop Interrupt Flag is set when a start condition succeeded by a valid address packet is detected. When this happens, the SCL line is forced low, giving the slave time to respond or handle any data as needed.

Once a `START–ADDRESS–ACK` sequence occurred, the combination of the read-/write bit and bus condition give rise to one of the four cases the slave must react to. These are summarized below:

Case 1: *Address packet accepted - Direction bit set.*
Read/write bit is high, indicating a master read operation. Clock stretching is performed by forcing the SCL line low. If an acknowledgement is sent by the slave, its hardware will set the Data Interrupt Flag indicating that data is needed for transmit. If a negative acknowledgement is sent however, the slave will wait for a new start condition and address match.

Case 2: *Address packet accepted - Direction bit cleared.*
Read/write is low, indicating a master write operation. The clock line is forced low and if an acknowledgement is sent by the slave, the slave will wait for data to be received. After this, more data, repeated start or stop may be received. A negative acknowledgment will force the slave to wait for a new start condition and address match.

Case 3: *Collision.*
When the slave is unable to send a positive or negative acknowledgment, the Collision Flag is set. This disables the data and (negative) acknowledge output from the slave logic, and also releases the clock hold. A start or repeated start condition will be accepted.

Case 4: *Stop condition received.*
This case is analog to cases 1 and 2, except that when a stop condition is received, the slave Address/Stop Flag is set and not the Address Match Flag as in case 1 and 2.

4.7.2 Receiving Data

The TWI module is currently not configured to accept any attempts to be written *to*, i.e., there is no data *command* functionality at present. However, the required drivers are in place should the need arise in the future.

4.7.3 Transmitting Data

When the system acknowledges a master's attempt to read telemetry data, all sensor-data is refreshed, converted to 16 bit values corresponding to the measured values in mV, mA or degrees Celsius where appropriate. The final step before shipping them off to the TWI data buffers, is splitting the integers into high- and low byte unsigned characters to match the TWI 8-bit data field. For example, if the solar panel voltage is 5.5V, the third and fourth telemetry bytes will be 0b00010101 and 0b01111100. The makeup and order of the eight telemetry bytss are summarized in Table 4.2.

Table 4.2: Telemetry data sent on master request.

Byte	**Data**
1	Solar panel current sensor 8 MSB
2	Solar panel current sensor 8 LSB
3	Solar panel voltage sensor 8 MSB
4	Solar panel voltage sensor 8 LSB
5	Battery voltage sensor 8 MSB
6	Battery voltage sensor 8 LSB
7	Battery temperature 8 MSB
8	Battery temperature 8 LSB

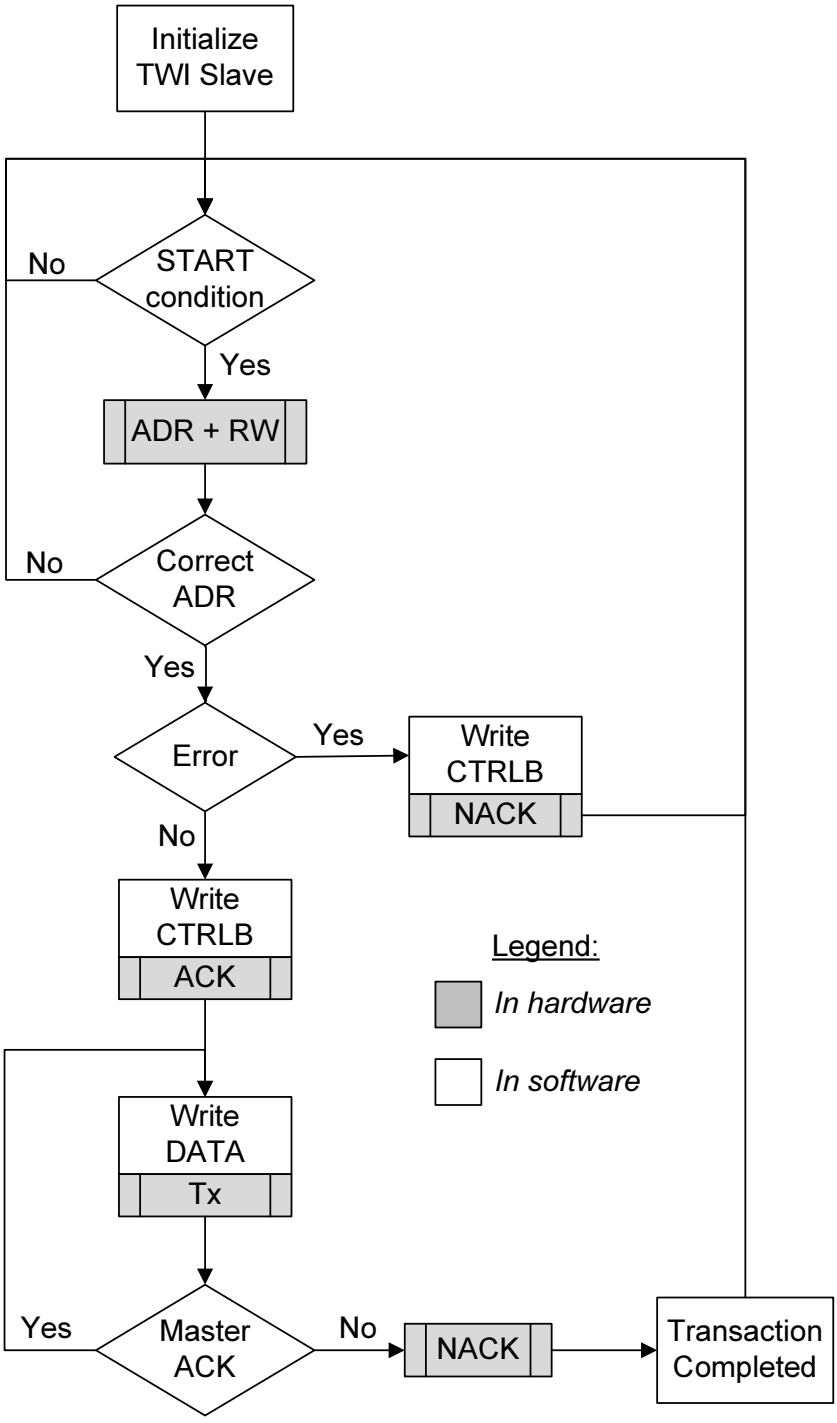


Figure 4.13: Flow diagram of the implemented slave algorithm.

4.8 Powering the Microcontroller

The EPS reliance on the microcontroller for most of its tasks, raises a critical question: How does the EPS provide power for itself?

There are two possible power sources available to the microcontroller; the solar cells (V_{SA}) and/or the battery (V_{BAT}). The problem of choosing one over the other can be traced back to a desire to operate the EPS continuously throughout the orbit, and the need to be able to recover from a flat[5] battery. Since the solar array does not provide power during the eclipse, and the microcontroller cannot run from a flat battery, the two requirements means that neither the solar panels nor the battery can sustain system power on their own.

To overcome this problem, the scheme in Figure 4.14 was implemented.

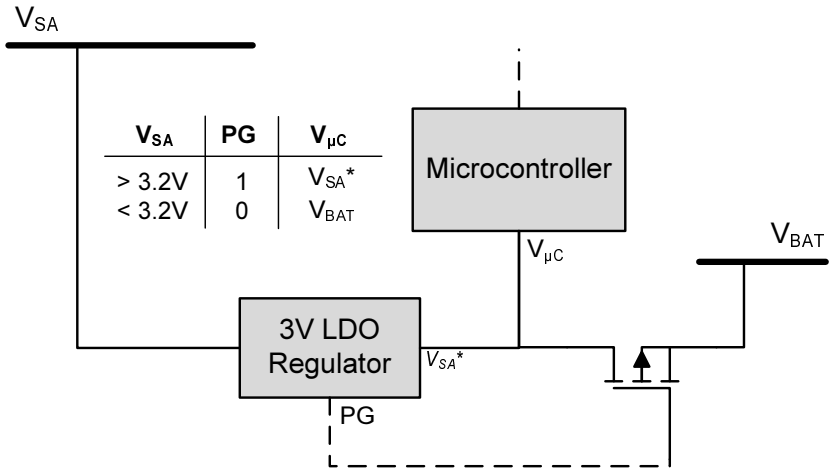


Figure 4.14: Scheme for powering the system for the duration of an orbit.

A low dropout voltage regulator[6] is used to step down the solar array voltage to a usable $V_{SA}* = 3.0$V. As long as the LDO input is higher than about 3.2V, the regulator output will be stable at $V_{SA}* = 3.0V$ and the PG signal, being active high, keeps the PMOS turned off. If the array voltage drops

[5]Defined here as below 2.7V where the uC no longer guarantees its current operating clock frequency.

[6]TPS76630 from Texas Instruments

below 3.2V, the active high `Power Good` signal pulls the gate of the p-type MOSFET[7] to ground—well below the threshold voltage ($V_{GS(th)} = -1.5V$) required to turn the FET on—at which point the MCU power is taken from the battery.

Since the battery voltage range is within the rated input voltage range of the microcontroller, the battery voltage is used directly without further regulation. This keeps the loss during the eclipse sleep-mode to the negligible conduction loss in the MOSFET, while the LDO provides a stable power source for the MCU during the period it matters the most.

A potential drawback with this approach is the switching back and forth in the case of a free tumbling satellite. Testing should be done to evaluate this effect, and a solution where the LDO's active low enable signal is used to only use the solar array power in the case of battery failure, should be considered.

The PG signal could also trigger an interrupt that disconnects the solar array from the bus, and forces the microcontroller into a low-power sleep state where it would stay for the duration of the battery-only power mode. This idea has not yet been implemented though.

In general, the power dissipation in the ATxmega is proportional to the square of the device's supply voltage, so to minimize power consumption, the lowest possible supply voltage should be used. Also, the overall power consumption of the satellite might benefit if the guidelines in Atmel application note AVR1010: "Minimzing the power consumption of XMEGA devices" was followed.

[7]FDN306P from Fairchild Semiconductor

4.9 Fault Protection and Monitoring Unit

Each subsystem that is connected to the power bus via the backpanel, is buffered from the main bus by a autonomous fault protection unit that also monitors the subsystem's current consumption.

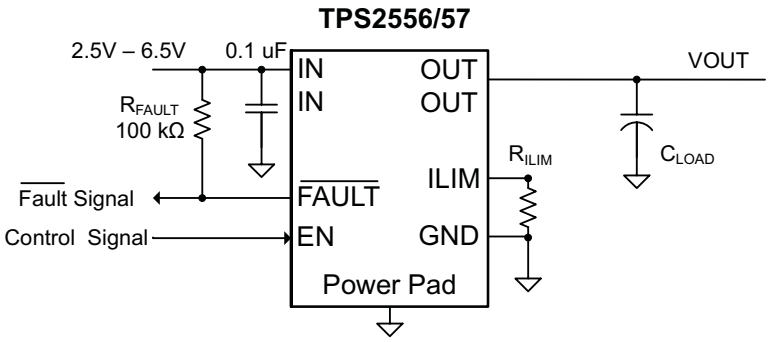


Figure 4.15: Overcurrent protection with TPS2556.

The chosen protection device is TPS2556 from Texas Instruments which is based on a 22mΩ high-side N-channel mosfet that is driven with an internal charge-pump. With a response time (t_{IOS}) of 3.5μs, it offers a programmable current-limit (I_{OS}) threshold between 500mA and 5A. The current-limit is programmable via an external resistor, R_{ILM}, given by

$$I_{OS(max)} = \frac{99038V}{{R_{LIM}}^{0.947}} \tag{4.17}$$

where the current is given in units of amperes if R_{LIM} is given in units of kΩ.

When a over-current or over-temperature condition is detected the $\overline{\texttt{FAULT}}$ logic output is asserted low until the condition is resolved, and the device automatically shuts off the troubled subsystem.

The `ENABLE` signal controls the device supply current and is compatible with TTL and CMOS levels, allowing for a microcontroller to manually turn individual subsystems ON or OFF as necessary. The device also has a built-in

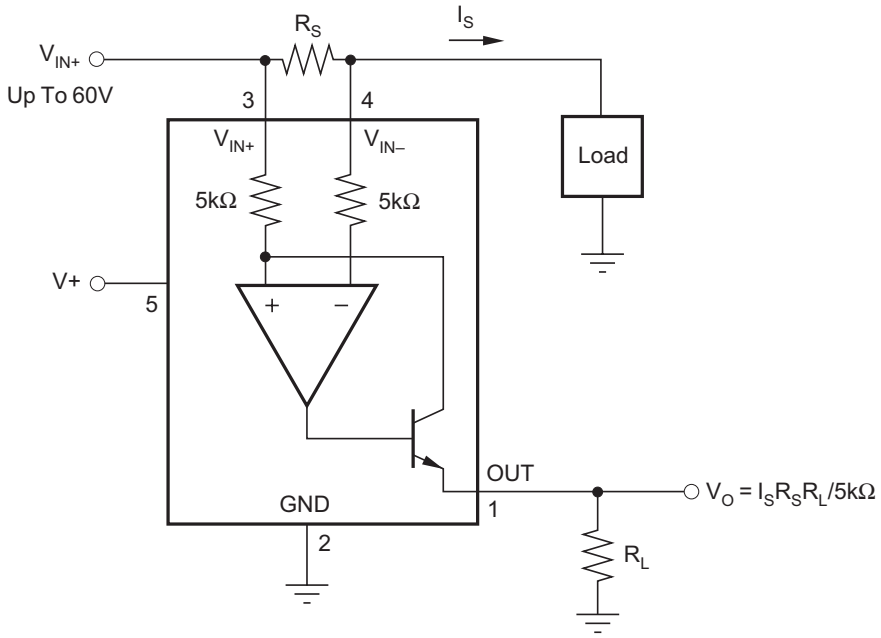


Figure 4.16: Current monitoring with INA138.

undervoltage lockout with hysteresis (35mV) which disables the switch until the input voltage reaches the turn-on threshold (2.35V).

To monitor the current consumption of each subsystem another device from TI is used. The INA138 is a unipolar high-side current monitor with high gain, allowing for a small sense resistor and low loss. It operates from 2.5V to 36V which is well within the expected system voltage range.

A 25mΩ sense resistor was chosen in this prototype, which, together with the 2A upper limit set by TPS2556, and a maximum ATxmega ADC voltage of 2V, means that we need R_L to be

$$R_L = \frac{V_o \cdot 5\text{k}\Omega}{R_S \cdot I_S} = \frac{2\text{V} \cdot 5\text{k}\Omega}{25\text{m}\Omega \cdot 2\text{A}} = 200\text{k}\Omega$$

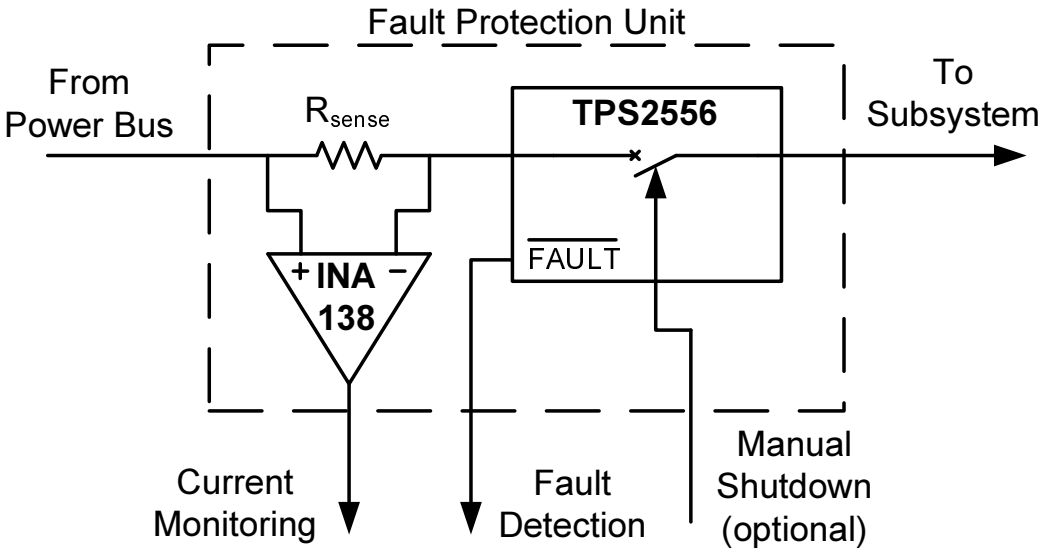


Figure 4.17: Automatic and/or manual overcurrent protection with TPS2556 combined with the current monitoring capabilities of INA138 constitute the fault protection unit.

This unit will be implemented next to each subsystem's connector on the backpanel.

Chapter 5

PCB Realization

A total of three printed circuit boards (PCBs) were designed and realized. Only the next two paragraphs will be devoted to the two first prototypes, before moving on to the main focus of this section: the PCB realization of the system described in Chapter 4. All work with the PCBs, from net list to schematics through wiring board design and finally post-processing, was done on the computer assisted design (CAD) suite Zuken Cadstar v12.1.

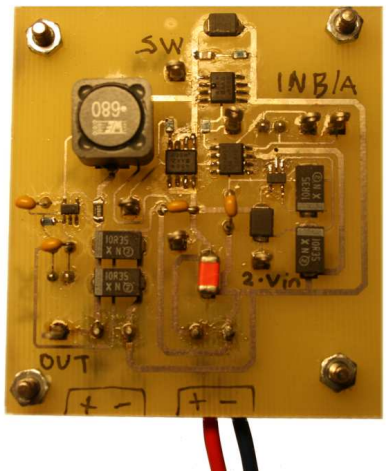


Figure 5.1: The first prototype was a two-layer PCB with a stand-alone buck converter with a driver and some supporting electronics.

The first prototype, shown in Figure 5.1 was realized on a two-layer board which basically consisted of a synchronous buck converter, a driver circuit, snubber network, a current sensor and some testpoints. The PWM signal was generated on a STK600/ATXmega development board from Atmel. This setup met the wall around the time work on the control algorithm began and the need arose to sample the various signals.

The second prototype, shown in Figure 5.2, was realized on a four-layer board. All of the working parts of the first board was transferred to the second prototype, which also included on-board digital control and ADC interfacing for the first time. In addition, typical 'bells and whistles' functionality was added for debugging.

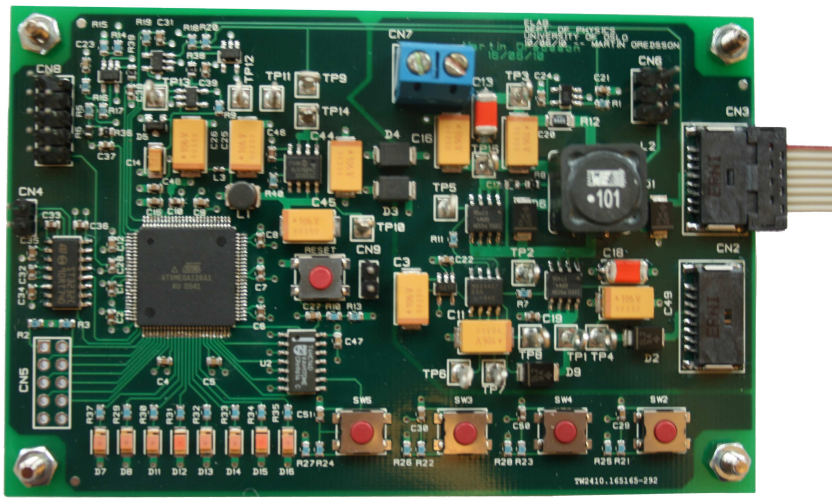


Figure 5.2: The second prototype was a four-layer PCB with on-board digital control, sampling and LEDS and switches for debugging, in addition to the buck converter on the right side of the board.

Finally, the system described in Chapter 4, was realized on the four-layer board shown in Figure 5.3. Apart from a few tweaks here and there, this board is electronically similar to the second prototype which did most of the tasks it was intended to do. The big change though was the introduction of a, second, redundant regulation system on a board that was smaller than the second (single-regulator) prototype. The remainder of this chapter concludes the work done on the final EPS prototype.

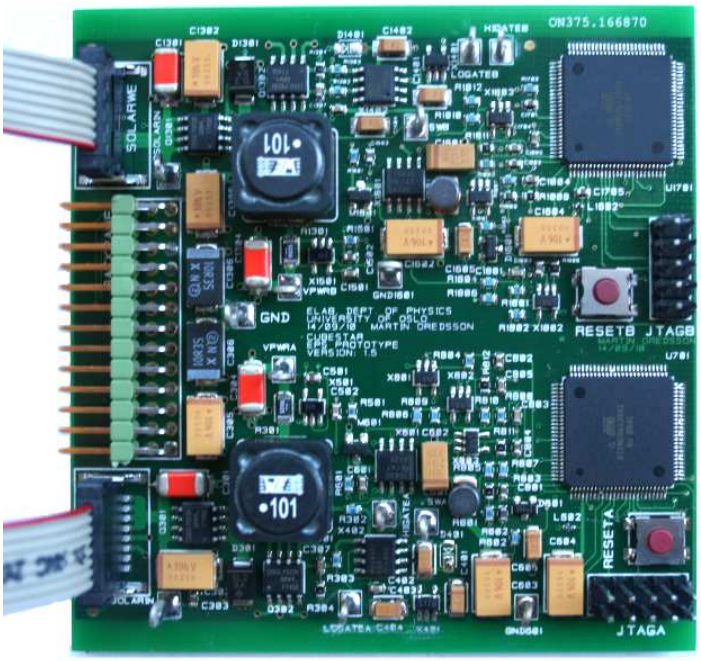


Figure 5.3: The third and last prototype is a four-layer PCB created with the CubeSTAR module. It contains two redundant regulation blocks with JTAG programmable digital parts, RS232 driver for debugging, I^2C interface, and connectors for the solar panels and backpanel/battery connector.

5.1 Physical Dimensions

Electronic design rules such as physical board dimensions, min/max trace width, and component-to-component/pads/etc distances, are all contained within pre-made templates made available to the CubeSTAR students by the Electronic Laboratory (ELAB) at the Univeristy of Oslo.

The two available templates are the *module* and the *backpanel* templates. However, neither the templates nor the design rules contained within them are set in stone, and evolve—at the mercy of ELAB—with the student's designs and needs. Their current status is shown in Figure 5.5. Using the module template means staying within the physical size of 80x75 mm with a maximum building height of 25 mm. The backpanel template is only shown

here for the sake of completion, and will not be discussed further.

The board itself is a standard FR4 type, where the dielectric between the copper planes is typically a woven fiberglass cloth that is reinforced with a flame resistant[1] epoxy resin.

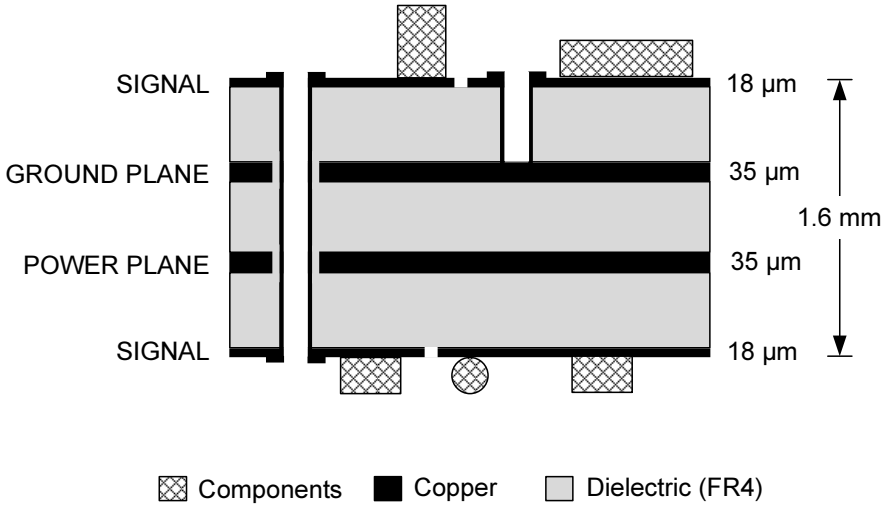


Figure 5.4: Four layer PCB stackup showing the inter-plane plated through-hole vias connections (not to scale).

Total board thickness is 1.6 mm with 18 μm (0.5 Oz) Cu foils for the signal layers, and 35 μm (1 Oz) foils for the Ground and Power layers. Wherever possible, 0603 SMD components were chosen for the passives, and plated-through hole vias with dimensions 1.0/0.5 mm for inter-plane connections. Where space was tight the via size was reduced to 0.6/0.3 mm, with the two numbers representing via pad and hole diameter, respectively.

[1] **F**lame **R**esistant, hence the abbreviation: FR4

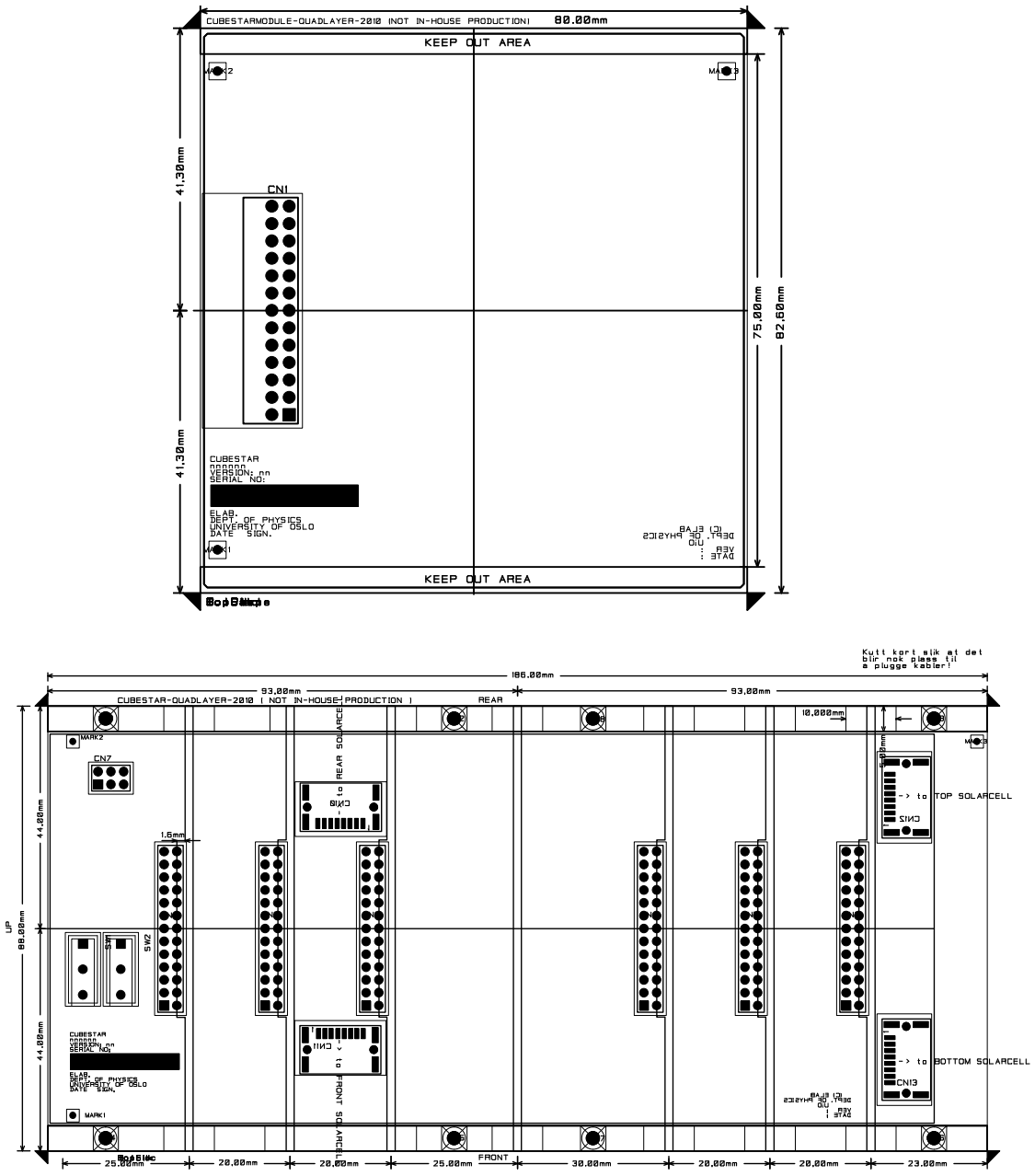


Figure 5.5: The current status of the CubeSTAR module (top) and back-panel (bottom) templates. Both were created and are maintained by the Electronics Laboratory (ELAB) at UiO

5.2 Electromagnetic Compatibility

Choosing a switching buck converter as the power regulator means that we are choosing an inherently noisy power system. The Cubesat formfactor, as with the shrinking size of electronics in general, means that the subsystems circuits will operate in close proximity to each other. The task then—not only for this subsystem, but for all the subsystem designers—is to make sure that the circuits affect each other adversely as little as possible.

The definition of electromagnetic compatibility given in [25] is,

> *Electromagnetic compatibility (EMC) is the ability of an electronic system to (1) function properly in its intended electromagnetic environment and (2) not be a source of pollution to that electromagnetic environment.*

The first part of the definition is related to the *susceptibility* of a system, which is the dual of immunity; the capability of a device to respond to noise. The second part is related to the system's level of *emission*, or interference-causing potential.

This section deals with some of the sources of EMC problems and what has been done to minimize their effect in the PCB design and layout.

5.2.1 Conductor Parasitics

Being far from the ideal straight lines known from the schematics, real life conductors always have a certain amount of resistive and reactive behavior. Since these conductors inter-connect every part of the circuit, a quick look at how they (mis-)behave may be a good idea. The geometry used in this section is shown in Figure 5.6, where the distance d also will be used to represent the height of a trace above a ground plane.

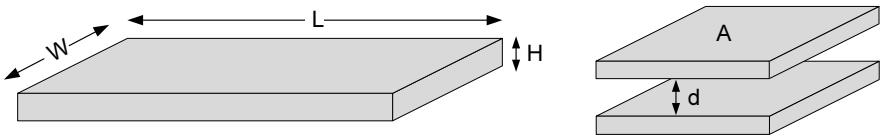


Figure 5.6: Trace inductance and plane capacitance geometries.

Resistance

The trace resistance in Ohms per meter for a material with a given resistivity ρ in nΩ·m is

$$\text{Resistance } (\Omega/\text{m}) = \frac{\rho}{W \cdot H}$$

where W and H are the trace width and height in units of meters, as illustrated in Figure 5.6. For example, a 5 cm long copper ($\rho = 16.8$ nΩ·m) trace, with W = 1.0 mm and H = 18 μm, has a resistance of 47 mΩ. Resistance versus trace width is illustrated in Figure 5.7.

Inductance

Strip inductance is another parasitic to consider. Even at relatively low frequencies, a conductor usually has more inductive reactance than resistance[25]. The trace loop inductance, valid as long as d > W, is

$$\text{Inductance } (\text{nH/cm}) = 1.996 \times \ln\left[\frac{5.98 \cdot \text{d}}{0.8W + H}\right]$$

where W is the trace width and H is the trace thickness at a distance d above the ground plane. Any units can be used for d, W, and H, as long as they are consistent. For example, a 5 cm long trace with a cross section of 0.25 mm x 18 μm, running 0.5 mm above its return ground plane, has an inductance of roughly 25 nH. The effect of trace width on inductance is illustrated in Figure 5.8.

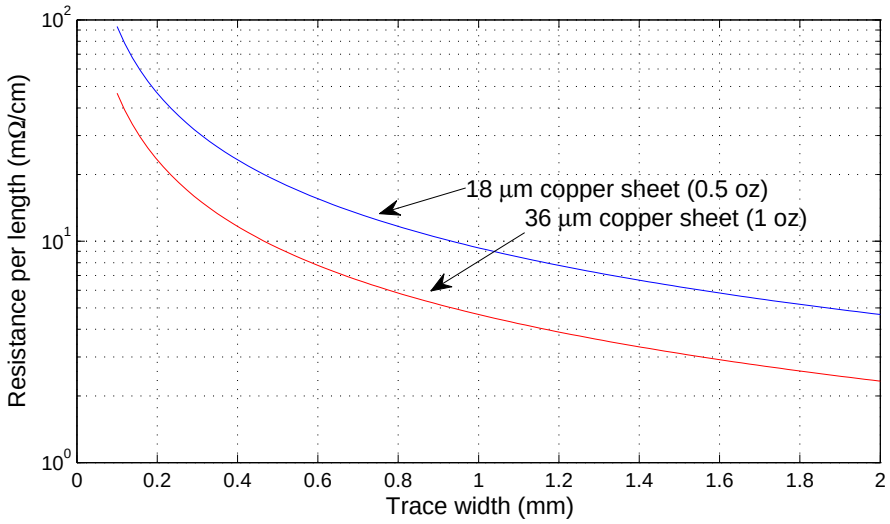


Figure 5.7: The resistance per length versus trace width for the two sheet thicknesses used on the printed circuit board.

Capacitance

The capacitance in pF for the parallel-plate capacitor shown to the right in Figure 5.6, is

$$\text{Capacitance} = \epsilon_0 \epsilon_r \frac{A}{d}$$

where $\epsilon_0 = 8.85 \times 10^{-12}$ F/m is the vacuum permittivity, $\epsilon_r \approx 4.5$ is the relative dielectric constant of the FR-4 board material, A is the plate area in cm^2, and d is the distance between the plates. For example, two 10 cm^2 plates 0.5 mm apart has a total capacitance of only 800 pF—not of much use as decoupling capacitance in power management.

In fact, for frequencies below 500 MHz, the simplest way to improve the EMC performance of a four-layer board is to place the signal layers as closely as possible to the current-return planes, thus minimizing the current loop areas at the expense of this small and insignificant inter-plane capacitance[25]. This was not, however, implemented on the current prototype where equidistant

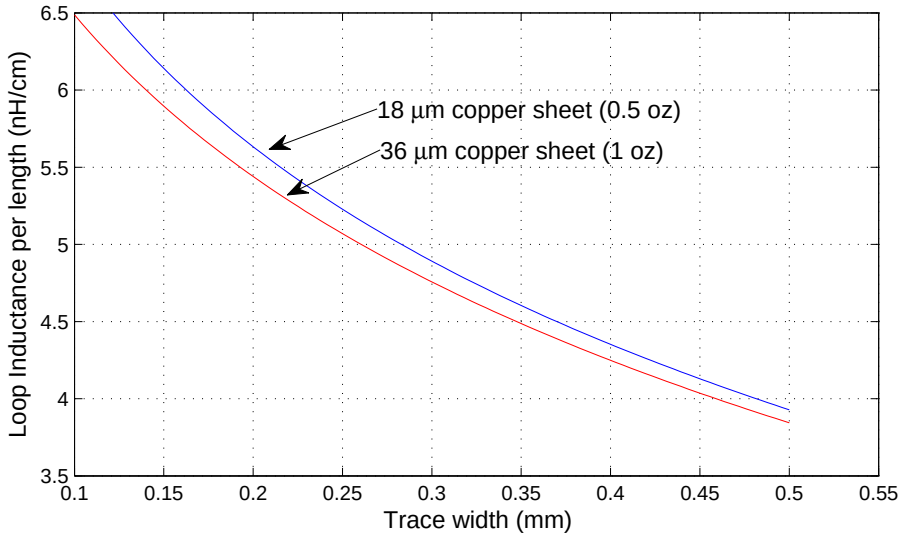


Figure 5.8: Loop inductance versus trace width for rectangular PCB traces located 0.5 mm above a ground plane for the two different sheet thicknesses used in the PCB.

layers are used.

5.2.2 Noise and Interference

The definition of noise given in [25] is,

> *Noise is any electrical signal present in a circuit other than the desired signal.*

Two things are immediately clear with that definition. First of all; noise wins. There's simply no way a real-life signal will look like the simulated ideal signal. But there's still hope; by reducing the undesirable effect or *interference* of noise until the noise magnitude is at an acceptable level, the problem is solved. Second, whether or not a signal is classified as noise depends on *where* it is. A desired signal in one part of the system can become noise if coupled to another part of the system.

The noise path is often illustrated as shown in Figure 5.9.

Figure 5.9: Typical noise path. All three blocks must be successfully identified to analyze a noise problem.

The source of noise can be grouped into three categories: 1) intrinsic noise such as thermal and shot noise; 2) man-made noise such as digital electronics and switching; 3) noise from natural disturbances such as sunspot activity and lightning. Some of the techniques used to here to reduce noise are generic and can be used on all kinds of noise, only noise reduction of the the second category has been actively pursued here.

The coupling channel may be either a solid conductor which is run through a noisy environment, or by means of electric and magnetic field coupling, while the receptor in Figure 5.9 is the affected system.

5.2.3 Current Return Path and Ground Noise

A lot can be said about electrical currents, but one thing is universally true; it can only flow in loops. Michael Faraday discovered that when the magnetic flux enclosed by a loop of wire changes with time, a current is produced in that loop, indicating that an *electromotive force* (emf) is induced around the loop[22]. This wonderful result can be stated mathematically in what is know as Faraday's Law:

$$\oint_C \boldsymbol{E} \cdot d\boldsymbol{l} = -\frac{d}{dt} \int_S \boldsymbol{B} \cdot d\boldsymbol{S} \tag{5.1}$$

where S is the surface bounded by the closed path C while $\boldsymbol{E}$ and $\boldsymbol{B}$ are the electric and magnetic field vectors. The minus sign on the right-hand

side of Equation 5.1 ensures that Lenz's Law is satisfied: the sense of the induced emf is such that any current it produces tends to oppose the change in the magnetic flux producing it. Solving Faraday's Law for a given situation reveals that the induced emf is proportional to 1) loop area, 2) current strength, and 3) frequency.

From a PCB perspective, the above result means that if the current drawn by a device must take a long de-tour around the board before returning to the device's Vcc pin, a large current loop will be created which will be susceptible to electromagnetic noise from surrounding sources, or, as in the case of high current loops, create noise itself. Of the listed three, only loop area is usually under the designer's control at the production stage of development, and by keeping it as small as possible, the risk of noise problems can be greatly reduced.

By adding a separate ground plane as in Figure 5.4, the loop area is reduced to the area traced by distance between the signal and ground planes in addition to the track length. But "ground" is not a magic black hole where current disappears. The impedance of any conductor is complex and can be written as

$$Z_g = R_g + j\omega L_g \tag{5.2}$$

From Equation 5.2 it is clear how the ohmic resistance R_g dominates the total impedance at low frequencies ω, while the inductance L_g dominates at higher frequencies. Also, any conductor carrying a current will, per Ohm's Law, have an voltage drop associated with it. This is true also for ground planes. The voltage drop (noise), which may cause interference in the ground system, is given by

$$V_g = I_g Z_g \tag{5.3}$$

where I_g is the ground current and Z_g is the ground impedance defined in Equation 5.2. In fact, Equation 5.3 and 5.2 provide good reasons for the decision to use the term *current return path* in place of *ground plane* in the section title because it demonstrates that, assuming a ground current, no two physically separated points in a ground plane will ever be at the same potential. Further, since the impedance is frequency dependent, the term

return current moves the focus more towards the actual path taken by the ground current.

We then have two options if we want to minimize the ground noise voltage; minimize Z_g or decrease I_g by forcing the ground current through a different path. The two different scenarios in Figure 5.10 illustrates how frequency affects the current return path.

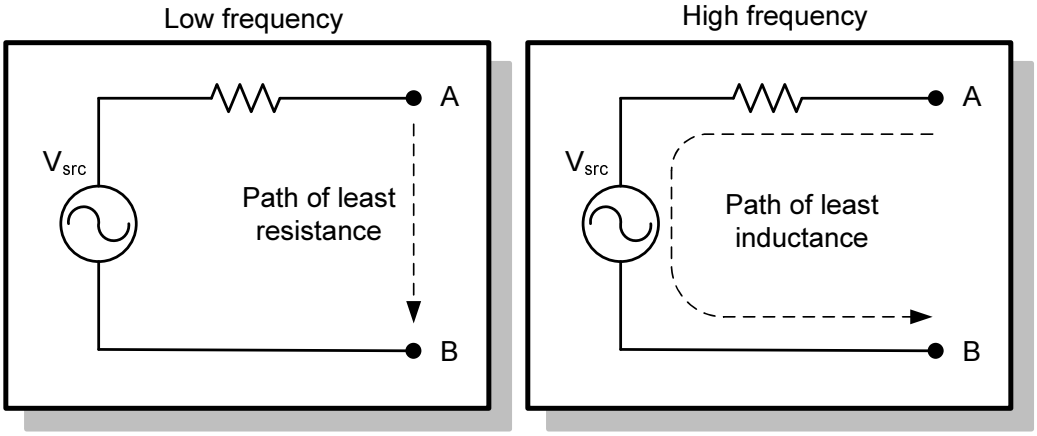


Figure 5.10: The path taken by the return current in the ground plane is dependent on frequency. At higher frequencies the return current seeks the path of least inductance, which is traced out by the path that minimizes the loop area.

In the low-frequency case, the current takes the path of least resistance, which is the direct path from A to B in the ground plane. In the high-frequency case, the current takes the path of least inductance (or loop area) which is the return path in the ground plane directly under the signal trace. The distinction between low- and high frequency is typically a few hundred kilohertz[25], and from the above discussion we can conclude that high-frequency signals in fact behave as desired as far as reducing loop areas is concerned—that is, as long as we don't interrupt the signal path. On the other hand, the low-frequency signals can create large and undesired current loops with improper layout.

5.3 PCB Layout Noise Reduction Techniques

Noise and EMI issues with switch-mode DC-DC converters are often a result of bad board layout and component placement[2]. Cutting corners here may cause errant switching, excessive voltage ringing and even circuit latch-up. So, not only does the layout dictate the EMC performance; getting it right can be crucial to get correct functionality. Some of the techniques that were used to tame the circuit are discussed below.

5.3.1 Decoupling

When a logic gate in an IC switches, a current transient, dI, rushes through the ground and power traces as seen to the left in Figure 5.11. From the familiar expression for voltage across a inductor

$$V_L = L \cdot \frac{dI}{dt}$$

it is clear the current spike will produce a corresponding voltage spike across the trace inductances—an *unwanted* voltage which, per the definition in Section 5.2.2, is regarded as noise on the line.

Regardless of the aforementioned measures taken against noise, PCB trace distance will introduce an impedance ($R_{p,g}$ and $L_{p.g}$ in Figure 5.11) which will create switching noise from the transient currents. Decoupling capacitors are used to maintain a low dynamic impedance from the individual IC supply voltage to ground. The capacitor stores energy in the form of charges and helps minimize the local supply voltage droop when a fast current pulse is taken from it. Placing the decoupling cap close to the circuit it is decoupling, and making L_{p2} as small as possible, is crucial—in fact, the inductance of a long trace between the cap and IC pin can form a high-Q tuned LC circuit which might generate ringing effects that are worse than the situation with no cap at all.

The exact capacitance value is not critical, but it can be calculated by considering the transient current demand in relation to the acceptable power rail

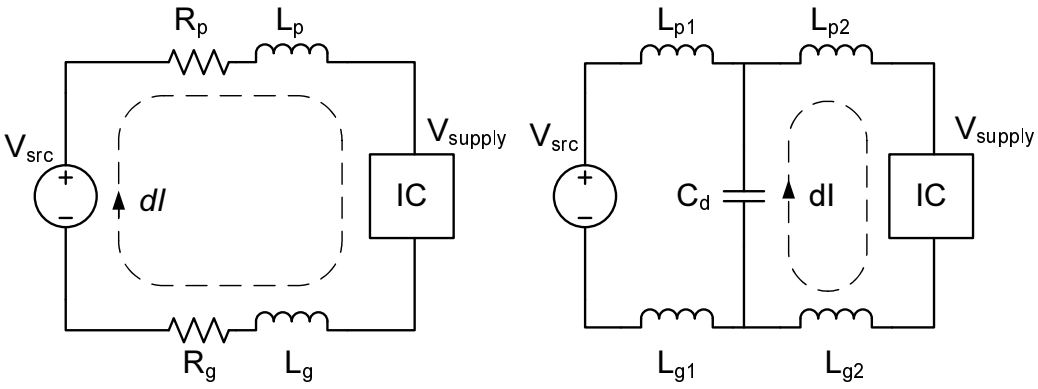


Figure 5.11: Transient power-supply current with and without a decoupling capacitor.

voltage droop. For example, if a chip sinks 10ns × 50mA current pulses and the acceptable voltage droop is 0.2V, then

$$I_C = C \cdot \frac{dV}{dt} \Rightarrow C_{min} = \frac{I_C \cdot \Delta t}{\Delta V} = 2.5 \text{ nF}$$

Normal values for decoupling capacitors are in the range from 10 nF to 100 nF. For low-frequency decoupling, tantalum electrolytics of 1-2 μF in areas where several devices may turn on simultaneously and draw current are recommended. Additionally, a single large capacitor of 10-47 μF at the power entry of the board is recommended to cope with frequency components in the kHz range.

5.3.2 Reducing the Loop Areas

Currents up to 1A, averaging around half that value, enter the board during the constant current battery charge stage. Therefore, the current loop areas for the on- and off states must be identified and minimized. This is shown in Figure 5.12, where the the low-side mosfet supplies the load with current while the high-side switch is off.

The loop created by the input capacitors and the two series MOSFETs is kept as small as possible. By doing so, any noise generated by the pulsating

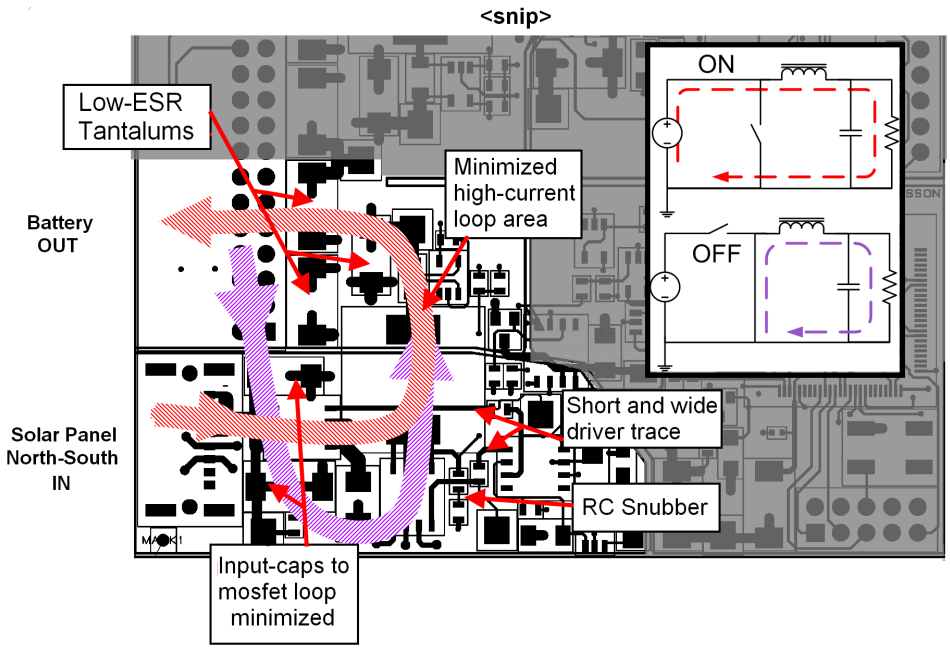


Figure 5.12: A cutout from the PCB illustrating some of the measures taken to minimize circuit noise. The situation shows the high current loops while charging the battery.

current in the input capacitors ESR and ESL is minimized, giving the high side MOSFET the full input voltage to work from.

The loop formed by the low-side MOSFET, the inductor and output capacitor is also minimized, as this helps reduce ringing on the switch (SW) node where the power switches and output inductor meet.

Since the digital part of the PCB is separated with "slots" in the ground plane, it becomes paramount not to cross the gaps. In Figure 5.13 a portion of the traces on the bottom signal layer can be seen. By forcing the traces on the bottom layer directly above their ground return current path, the loop areas are kept to a minimum.

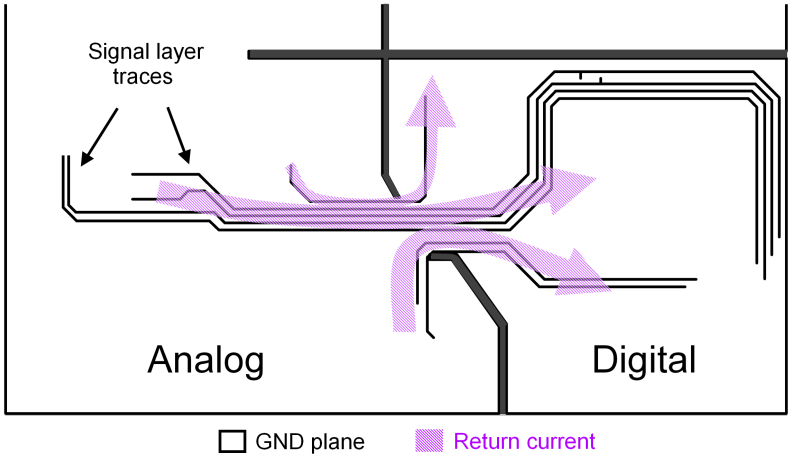


Figure 5.13: The ground plane is separated into digital and analog sections. By not crossing the gaps, the return current is forced to return underneath the signal traces thus minimizing the loop areas.

5.3.3 Implementation of a Passive RC Snubber

Unfortunately, there is a flip-side to the effort of trying to make the converter as efficient as possible. By keeping the parasitic resistances as low as possible, the decay time of the ringing increases, as there is little damping left in the circuit. Ringing on the SW node is almost inevitable due to the resonant

tank formed by the (lumped) parasitic capacitance C_p and inductance L_p at the node, as shown in Figure 5.14. The main culprits and cause of ringing are the drain-to-source capacitance of the SR switch, package inductance in the high-side switch and PCB traces and any capacitance seen looking into the output inductor.

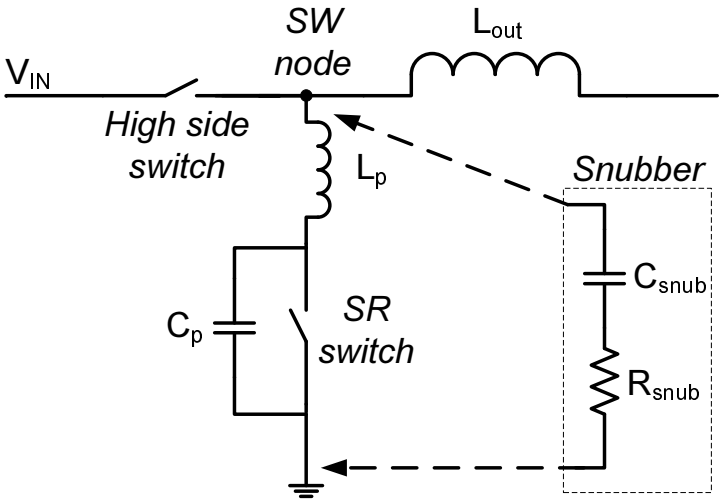


Figure 5.14: A simplified parasitic model for the switch node.

Since C_p is charged through L_p, the stored inductor energy has to go somewhere once the SW node reaches V_{in}. This energy shows up as ringing, where the tank's oscillating frequency is

$$f_0 = \frac{1}{2\pi\sqrt{L_p \cdot C_p}} \tag{5.4}$$

The snubber's basic function is to absorb energy from the node's parasitic reactances, which in turn means a reduction in voltage- and current spikes, noise, and power dissipation in the SR switch. The energy is not saved, however—dissipation is merely shifted from the SR switch to the snubber resistor, thus providing less stressful operating conditions for the switch. It's also worth noting that, for a given switching frequency f_{sw}, the dissipated energy in the snubber network

$$P_{snub} = \frac{1}{2} \cdot C_{snub} V_{in}^2 \cdot f_{sw} \tag{5.5}$$

is independent of the snubber resistor value. In fact, the only influence available over power dissipation at this point is the choice of C_{snub}, so care must be taken when choosing its value.

The snubber capacitor is charged to V_{in} when the high-side switch turns on, and discharged through R_{snub} when the same switch turns off. This does not mean the resistor can be chosen arbitrarily large however. One important design constraint is that the chosen resistor value should be such that the snubber RC time constant times three [6], is less than the minimum on-time or maximum off-time of the power switches, i.e.

$$R_{snub} \cdot C_{snub} < TD_{min} \tag{5.6}$$

where T is the switching period and D_{min} is the minimum dutycycle. This allows the snubber capacitor to fully charge and discharge during each portion of the switching period.

From an EMI/RFI perspective, the energy that flows back and forth between the C_p's electric field and L_p's magnetic field does not generally pose a problem[16]. However, the nature of low-voltage converters require the use of low gate threshold MOSFETs which are susceptible to parasitic turn-on if the ringing dv/dt becomes large. Also, the low cost and ease of implementation is in clear favor of using a RC network.

Parasitic Turn-on

The nature of low-voltage converters require the use of low gate threshold MOSFETs. A large dV/dt on the switch node represents a potential failure mode as these spikes may be capacitively coupled from the drain to the gate, and thus inadvertently turning on the SR during its off state. This can be understood by considering how a rising voltage on the switch node causes currents to flow in the parasitic capacitances and inductances in Figure 5.15.

As Figure 5.16 shows, a rise on the SW voltage also raises the SR drain voltage. This rate of change in voltage is related to the current flowing into C_{dg} and C_{ds} by

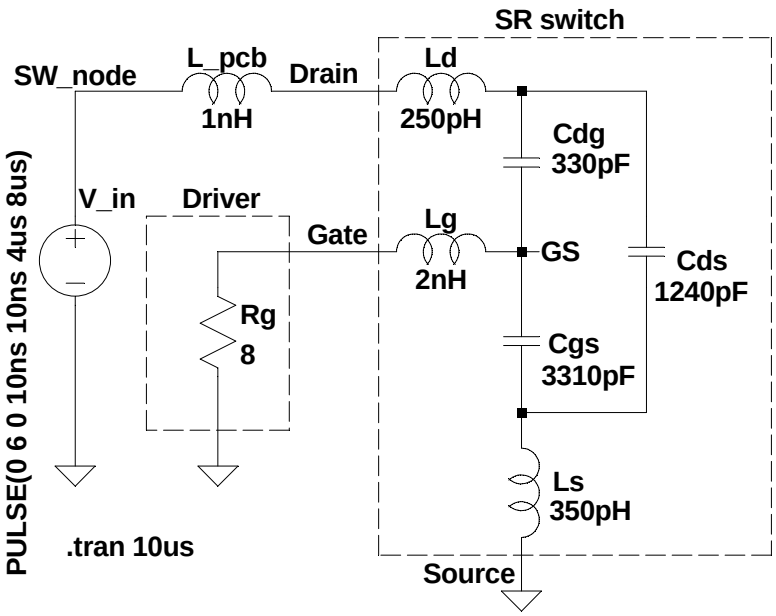


Figure 5.15: A parasitic SPICE model of the SR MOSFET can be used to predict if parasitic turn-on will be a problem.

$$i(t) = C \cdot \frac{dv(t)}{dt} \tag{5.7}$$

where C_{gs} should be added to C_{dg} in order to represent the total branch capacitance, C, as seen from the drain node. This capacitance is three times greater than that of C_{ds}, so most of the current will be geared towards charging the the gate capacitance. The current into C_{dg} must either be shunted to ground through the driver, or go into C_{gs}. The latter option causes the voltage on C_{gs} to rise, and if left unattended this voltage rise may above the gate threshold voltage and momentarily turn on the MOSFET. As mentioned in Section 4.3, a scenario where both switches are turned on simultaneously would short the input (i.e. the solar panels) to ground, and must be avoided at all costs. A SPICE simulation model is presented in [16] which can be used to predict this event *a priori*.

For this simulation the MOSFET used was IRF7456 (i.e., a different MOSFET from the one used to calculate the snubber network), but the model is generic and can be used for any similar situation. The manufacturers normally characterize their MOSFETs indirectly in their datasheets by C_{ISS},

C_{RSS} and C_{OSS}. Being functions of V_{DS} their values are estimated from plots in the datasheet at the maximum voltage that occurs across the device. The maximum voltage seen at the SW node is

$$V_{SW(max)} = V_{in(max)} - V_{out(min)} = 6V - 2.7V = 3.3V$$

From the IRF7456 datasheet we have

$$\begin{aligned} C_{DG} &= C_{RSS} = 330\text{pF} \\ C_{GS} &= C_{ISS} - C_{RSS} = 3640\text{pF} - 330\text{pF} = 3310\text{pF} \\ C_{DS} &= C_{OSS} - C_{RSS} = 1570\text{pF} - 330\text{pF} = 1240\text{pF} \end{aligned}$$

The parasitic inductances used are estimations from [16] where the typical SO-8 package leads each contribute around 2 nH of inductance. With four source leads, three drain leads and one gate lead, in addition to a small pcb trace inductance, the model in Figure 5.15 was used to produce the plot in Figure 5.16.

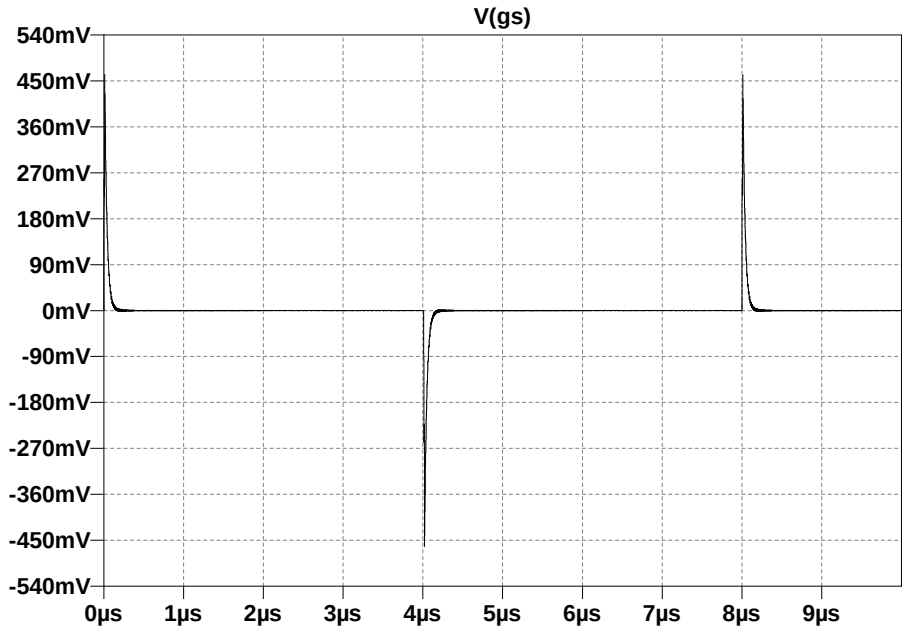


Figure 5.16: The SPICE simulation shows that the dv/dt bump has an amplitude of around 450 mV.

Simulation shows that shoot-through should not pose a problem under the given conditions. The spike—peaking at around 0.450 V—is below IRF7456's gate threshold voltage $V_{th} = 0.6$V. This is comparable to the real-life situation shown in Figure 5.17 which was captured before the snubber was implemented. The dv/dt bump can clearly be seen when the high side switch closes. In fact, due to the short duration of the voltage spike, a threshold crossing would probably not be a problem anyway due to the MOSFET's inherent turn-on delay.

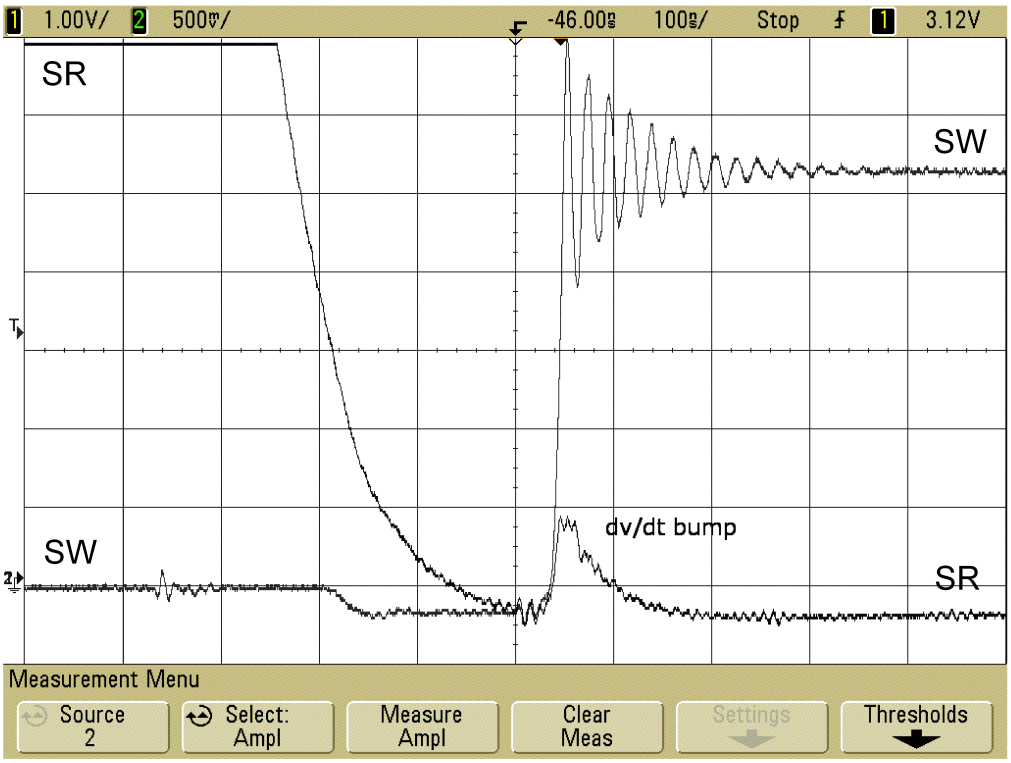


Figure 5.17: Capacitive coupling of large dv/dt ringing on the SW node may cause the SR gate voltage to rise above its threshold voltage while it is in the "off" state.

To counter the parasitic turn on effect, the following measures are taken:

1. low driver pull-down (sinking) impedance

2. reduced rate of voltage rise on the SW node by using a snubber

3. keep the impedance from gate to source of the SR as low as possible

4. close attention to drive circuit layout: wide short track for low impedance, GND track (plane) directly beneath it for low inductance and minimized loop area in the circuit

The third point must be weighed up against the need to minimize the excessive inductive energy buildup by limiting the switching time on the main switch. This is done by adding some resistance between the driver and the gate, keeping in mind that this increases switching loss.

Choosing Component Values

We know the tank's resonant frequency from measurement in Figure 5.18, and want to damp the ringing. Now what? With Equation 5.4 in mind, it would help if one of the parasitics was known. Of the two, C_p is the most straight-forward to find, because the switch node capacitance is dominated by SR's output capacitance $C_{oss} = C_{ds} + C_{gd}$, which can be found from the capacitance-voltage plots in the mosfet datasheet. Using C_{oss} as a starting point, the snubber capacitance will generally be two to four times this value. The final choice of C_{snub} should[6] be such that the frequency of the ringing to be damped is halved—at which point, the circuit capacitance is four times the original value. In other words, when C_{snub} halves the ringing frequency, its value is three times that of C_p.

In Figure 5.19, a 4.7nF cap has been added across the SR switch, and the frequency is now around 7.5MHz. Since the resonant frequency of an LC circuit is inversely proportional to the square root of the LC product, we can conclude from this that the node's parasitic capacitance is a third of 4.7nF, or

$$C_p \approx 1500\text{pF} \tag{5.8}$$

which is in agreement with the MOSFET's given datasheet[2] value for output capacitance.

[2]The mosfet used in this setup was Si4864DY from Vishay

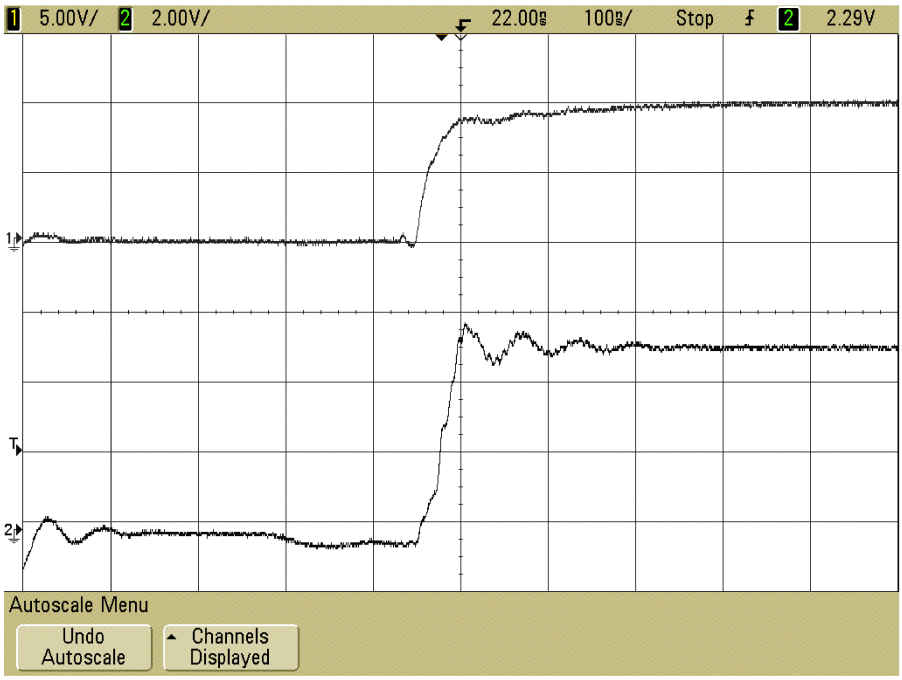


Figure 5.18: Without the snubber, the switch-node (bottom curve) rings with a period of T = 65 ns. The top curve is the high-side switch.

While C_p trivially ended up being equal to the output capacitance of the SR switch, the parasitic inductance is a little bit more coy, as it depends on PCB trace geometry and the mosfet component packaging (SO-8 in this case).

In the general case, L_p can be determined experimentally by measuring the ringing period T_1, before adding a parallel capacitor C_{test} of known value and noting the change in ringing period. The parasitic inductance will then be given by

$$L_p = (T_2^2 - T_1^2) \cdot \frac{1}{4\pi^2 \cdot C_{test}}$$

But with the prior knowledge of C_p as we now have, we can simply solve Equation 5.4 for L_p.

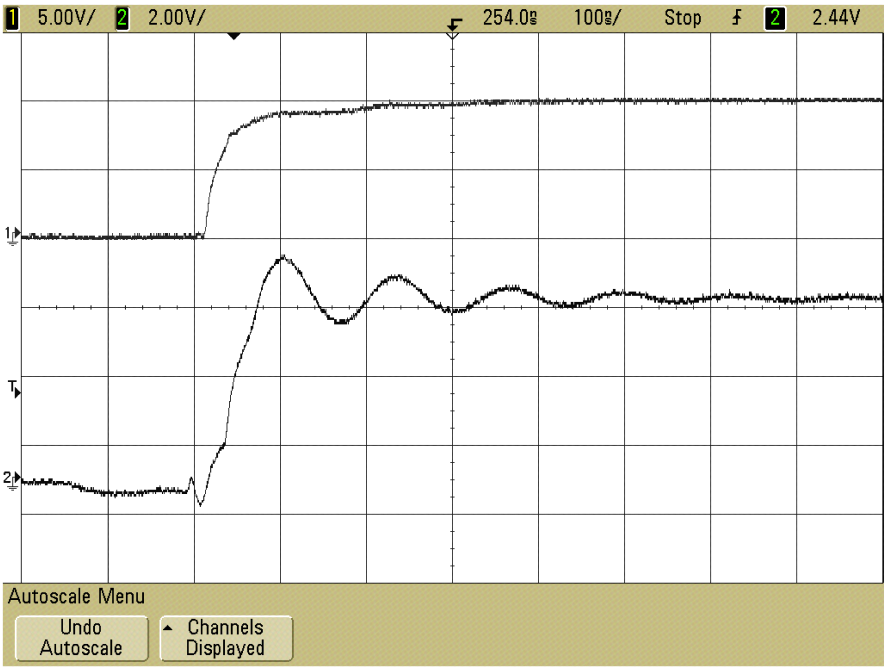


Figure 5.19: By adding a lone 4.7nF capacitor ($R_{snub} = 0$) across the SR switch, the ringing period is increased to T = 130 ns.

$$L_p = \frac{1}{(2\pi \cdot f_0)^2 \cdot C_p} \approx \frac{1}{(2\pi \cdot 15\text{MHz})^2 \cdot 1500\text{pF}} \approx 75\text{nH} \tag{5.9}$$

The final step is to choose the snubber resistor such that it equals the characteristic impedance of the resonant tank, allowing for damping near $Q = 1$.

$$Q = \frac{1}{R_{snub}\sqrt{L_p/C_p}} = 1$$

or

$$R_{snub} = \sqrt{\frac{L_p}{C_p}} = \sqrt{\frac{75\text{nH}}{1.5\text{nF}}} = 7.1\Omega \tag{5.10}$$

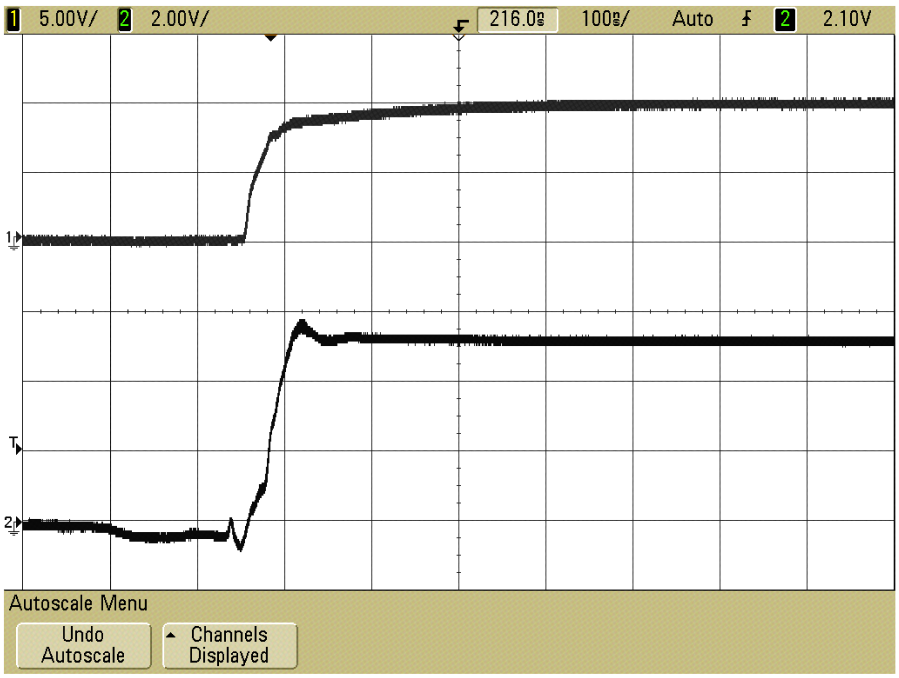


Figure 5.20: By placing a RC snubber across the SR switch, very little ringing is left on the switch-node waveform when the high-side switch (top curve) closes.

The resistor value was rounded up to the nearest standard value and the effect of the implemented snubber can be seen in Figure 5.20, where very little ringing is left on the waveform.

Chapter 6

Test Results

All of the tests related to system performance were done on the second prototype PCB.

6.1 Dark Current-Voltage Measurements on UTJ Solar Cell

This test was done to measure the *dark current* of the solar cells, which is the second term of the output current from Equation 2.3, repeated here for convenience:

$$I = I_{sc} - I_0 \left[\exp\left(\frac{qV_D}{nk_BT} \right) - 1 \right] - \frac{V_D}{R_{SH}}$$

The procedure used here involved covering the solar cells to eliminate light-generated current and then using a power supply to force electrical current through the cell from the positive contact to the negative. The current and voltage was measured as the power supply voltage was increased from zero to a predetermined limit. The limit was set to a maximum current of 20mA, to avoid damaging the cells. The voltage across the cell was measured along with the drop over the resistor.

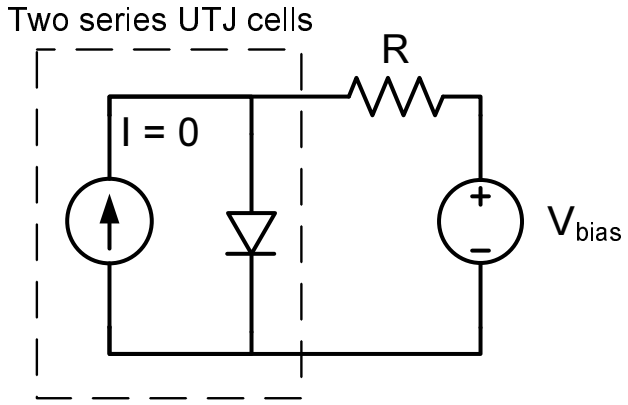


Figure 6.1: Test setup of the two series connected cells in complete darkness. A lab supply was varied and the voltage across the diode was measured. The voltage across the resistor (R = 100 Ω) was used to calculate the current.

The resulting direction for current flow is opposite to that of the photo-generated current (i.e. we're in the fourth quadrant of the IV plot), but the electrical configuration still means that the cell's p-n junction is forward biased as during normal operation. The test was repeated three times in different temperatures: 23 °C, 3 °C and -20 °C. Two series connected cells were used.

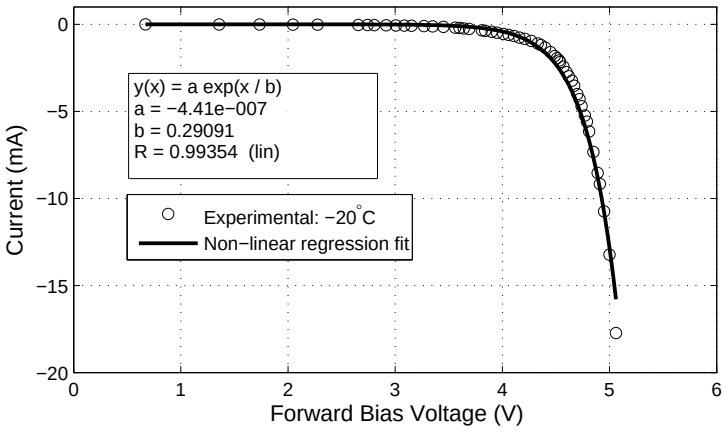


Figure 6.2: Dark current measurement in -20°C.

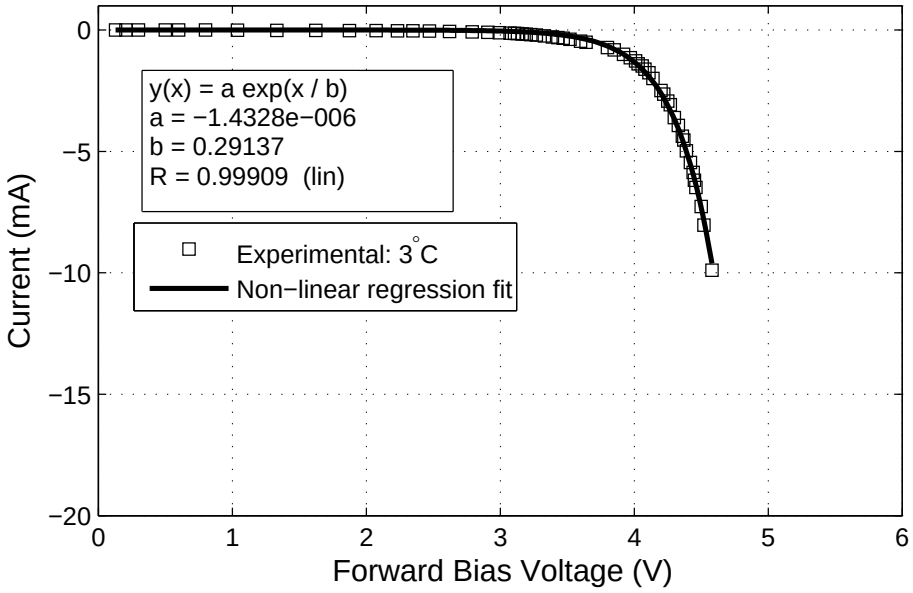


Figure 6.3: Dark current measurement in 3°C.

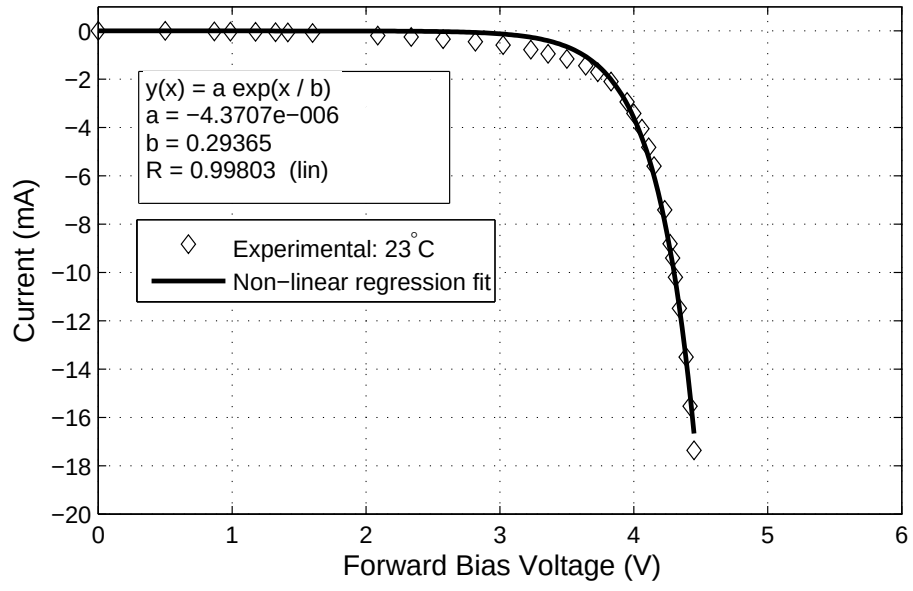


Figure 6.4: Dark current measurement in 23°C.

Regression Analysis

The plot in Figure 6.5 was made by plotting the non-linear fits returned from the Matlab extension package Ezfit.

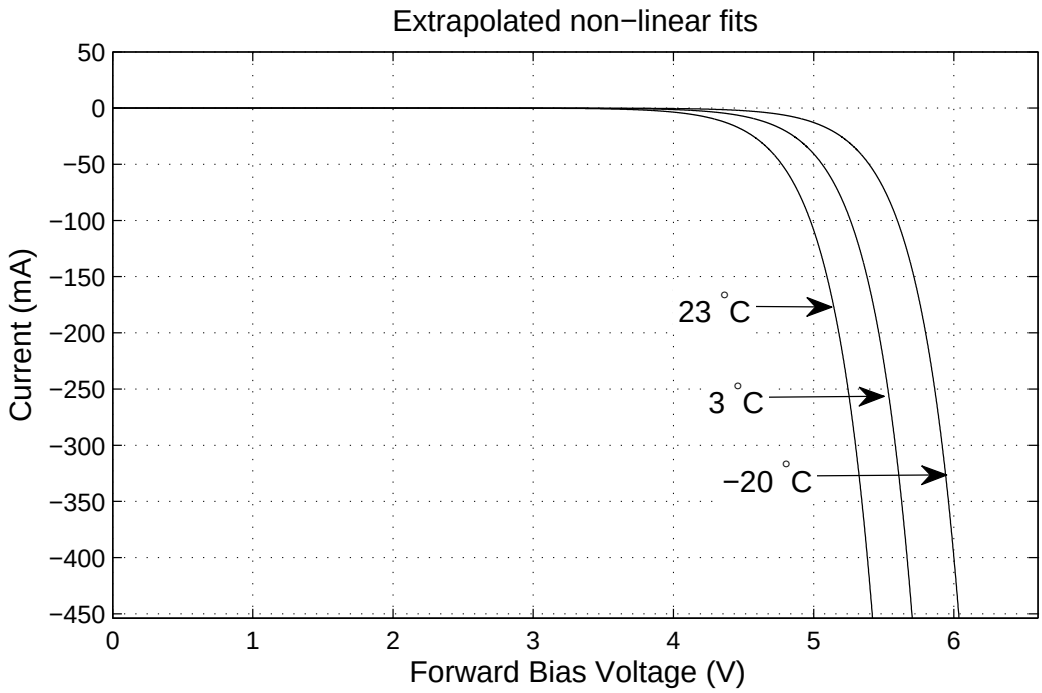


Figure 6.5: Extrapolation of the non-linear regression fits.

Finally, in Figure 6.6 a comparison was made between the extrapolated values from the dark current measurements and the total output characteristics of two series connected cells. The latter values were taken from the datasheet which states temperature coefficient of -5.9 mV/°C.

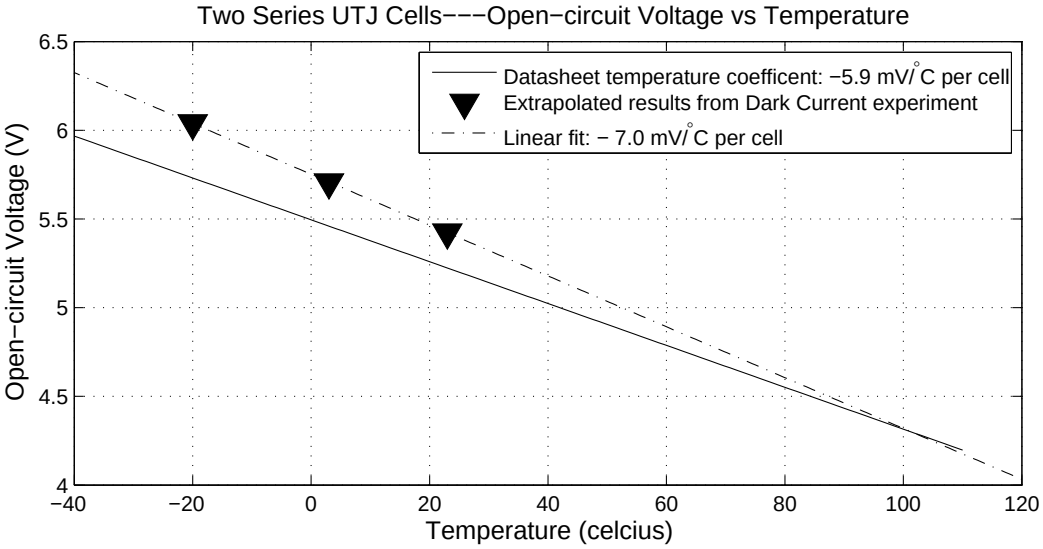


Figure 6.6: Comparison of the predicted datasheet temperature coefficient for V_{oc} and the results from the dark-current extrapolation with the Ezfit Matlab add-on package.

6.2 UTJ Solar Cell Angular Response

This test was done to determine the angular response of two series connected solar cells (see Section 2.5). The cells were mounted on a flat plastic assemble. The location was the roof of the Chemistry building at UiO , 17th December 2009 at noon. Temperature was -7°C.

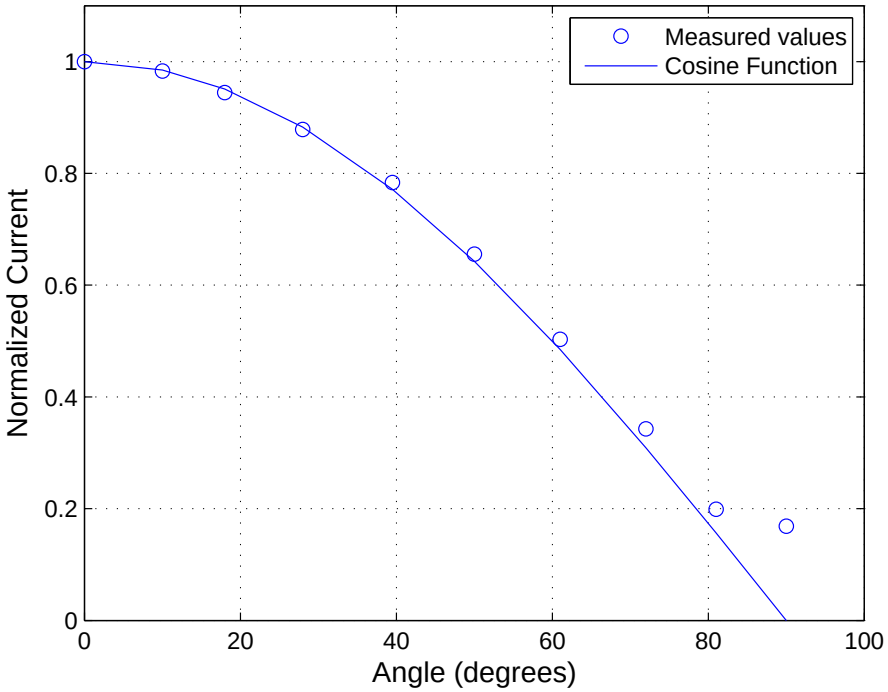


Figure 6.7: Angular response of two series connect UTJ solar cells.

Contrary to the theoretical prediction (i.e. the Kelly cosine) the experiment showed a slight *positive* deviation from the cosine function. However, the experiment was plagued by reflections from various directions, which has not been controlled for.

6.3 Open Loop Dutycycle Stepping

In this test the solar simulator circuit was setup with an open-circuit voltage of around 5.3V, and a short-circuit current of 0.67A, as shown in Figure 6.8.

The feedback loop on the prototype PCB was opened (in software) and the dutycycle was manually stepped with switch buttons on the PCB, resulting in the output in Figure 6.9. The dutycycle in the plots are 8-bit register values that can be converted to percent of full dutycycle by multiplying by 255 and dividing by 100.

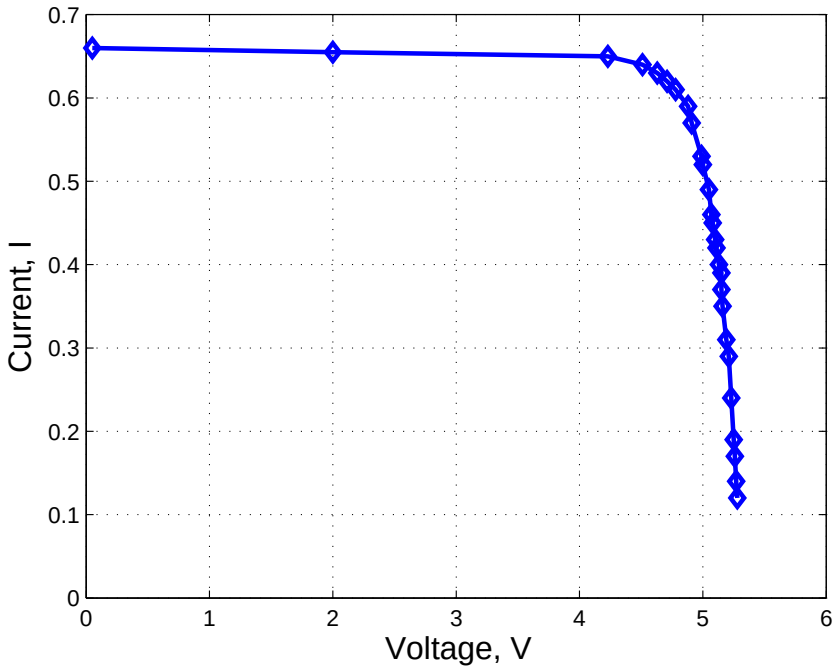


Figure 6.8: Solar simulator IV curve on which the MPPT testing was done.

It can be seen that, for this combination of open-circuit voltage and short-circuit current, there exists an optimal ratio of input and output voltage (i.e., the dutycycle) that results in the maximum output power from the solar panel.

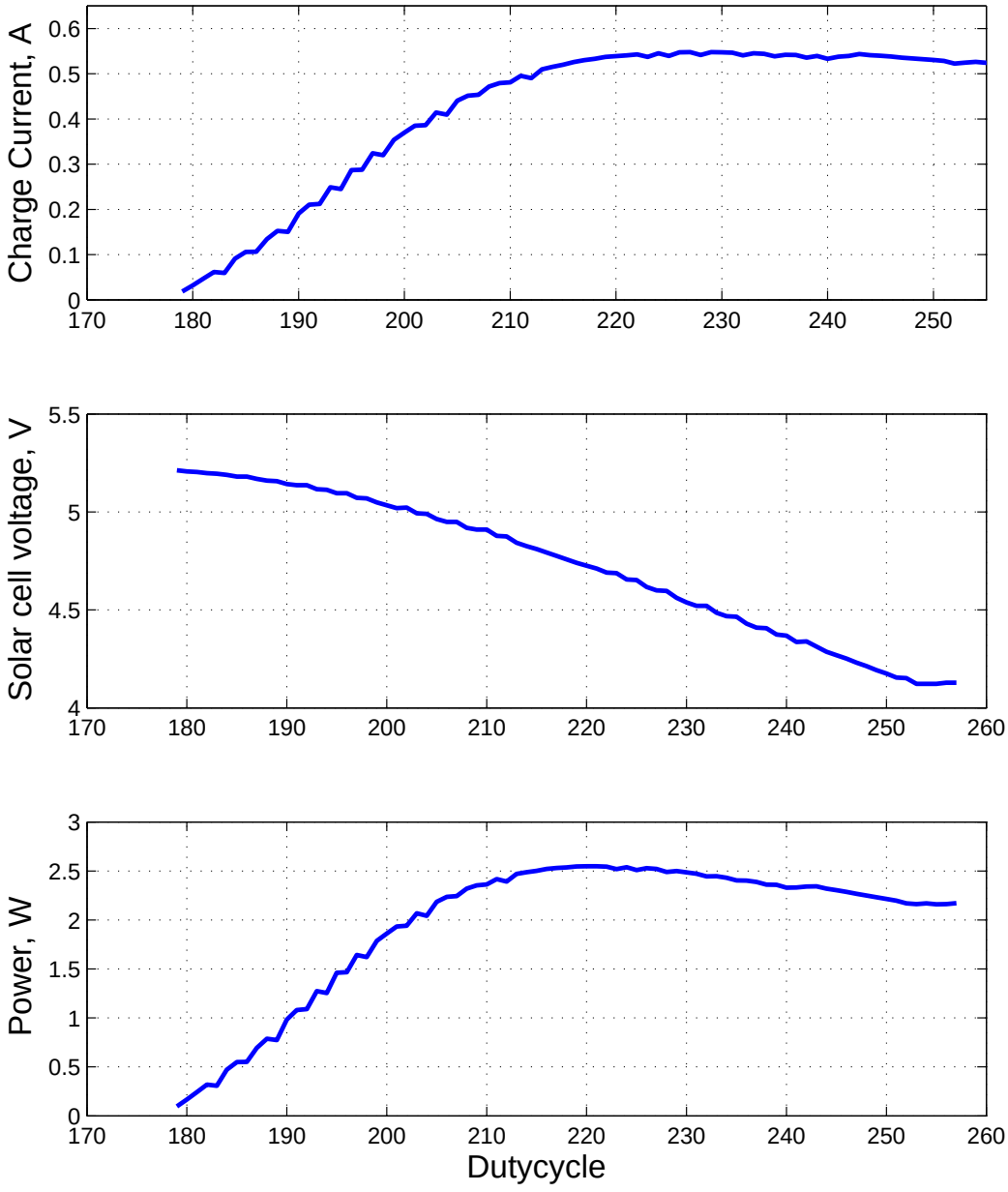


Figure 6.9: The effect of dutycycle on current, voltage and power when the solar simulator was set up with the I-V curve in Figure 6.8.

6.4 P&O Characteristic Oscillation

This test was done to illustrate the oscillating characteristic of the Perturb and Observe MPPT algorithm. By holding down a press button on the prototype board, the dutycycle was held at a low value. When the button was released the system resumed normal operation and seeked out the MPP as shown in Figure 6.10. Per definition the algorithm will oscillate around the MPP.

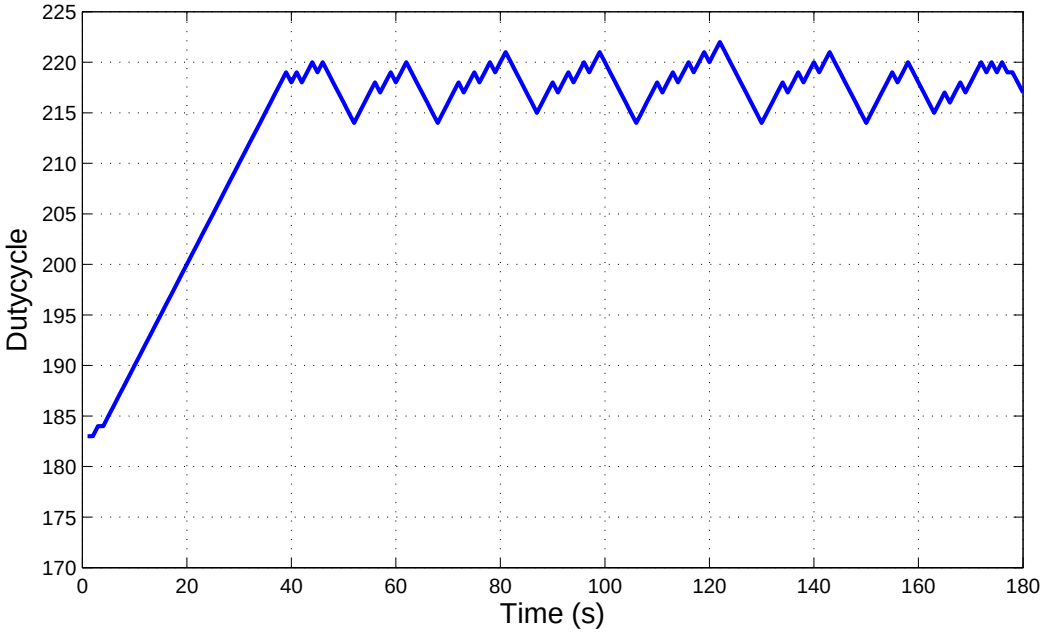


Figure 6.10: The control algorithm seeks out the maximum power point and the characteristically oscillates around the MPP.

6.5 P&O Response Time

In this test the open-circuit voltage of the Solar Simulator circuit was varied during the charging of a battery. This was done to simulate varying orbital temperature and to evaluate the algorithm's ability to track different power

levels. The "temperature" was varied by adjusting trimpot B on the solar simulator circuit. The solar simulator circuit output voltage and inductor current was sampled and the product used as the y-axis value.

As can be seen in Figure 6.11, the response of the MPPT to abrupt changes in V_{mp} results in undershooting. However, sudden changes (i.e. the time it takes to turn a trimpot a couple of turns) in temperature are not expected in orbit, except for perhaps the exit from eclipse where the system is thrown back into action.

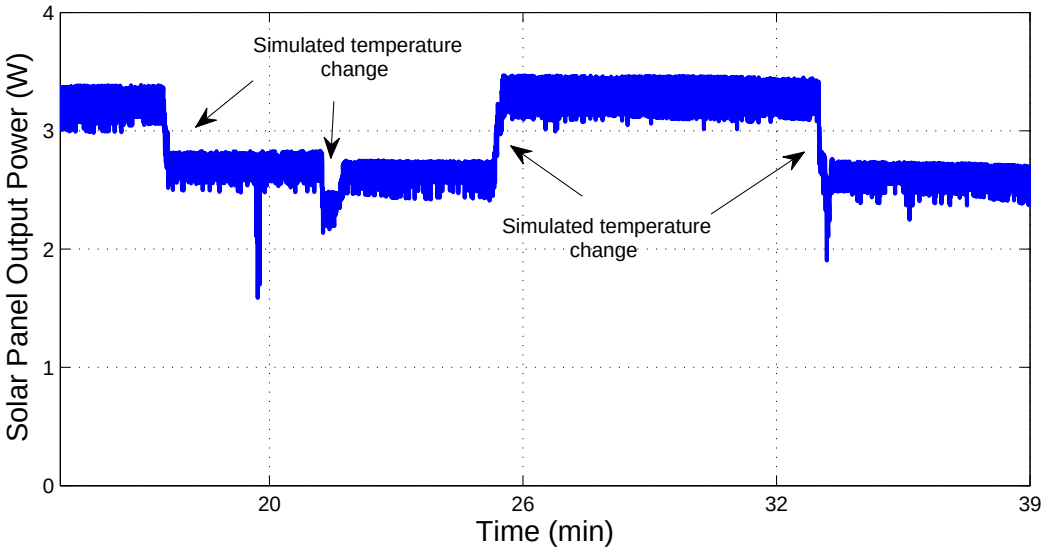


Figure 6.11: System/MPPT response to variations in open-circuit voltage and hence V_{mp}.

6.6 MPPT to PID State Transition

A single $LiFePO_4$ cell that had been discharged to 2.9V with a 6 Ω resistor in room temperature was used in this test. It was charged as shown in Figure 6.12, where the MPPT state takes care of the Constant Current phase while the PID state takes care of the Constant Voltage phase.

The storage cell was then immediately discharged again with the 6 Ω resistor

to 3.24V this time. The PID float voltage was set to 3.6V and recharging resumed. The full charge-discharge-charge cycle with two different float voltages (set in software) can be seen in Figure 6.13

In the first charging the float voltage was set to 3.47V and in the second charging the float voltage was set to 3.62V. A note on the battery voltage curve: the sampling time of one sample per 5 seconds does not catch the spikes that can be seen on a scope. The spikes (not shown) have a duration of about 10ns and an amplitude of around 400mV and occur on every rising SW node flank, i.e., the noise frequency is—not surprisingly— equal to the switching frequency. It's unclear yet whether or not these spikes present a problem, and no attempt has been made to remove them. However it is possible that part of this effect could be traced back to a poorly damped snubber network on the prototype during this test. Another source of uncertainty here is the ground strap of the scope probe; earlier tests with different ground strap lengths showed doubling and even tripled effect of overshoot ringing due to the added ground strap inductance.

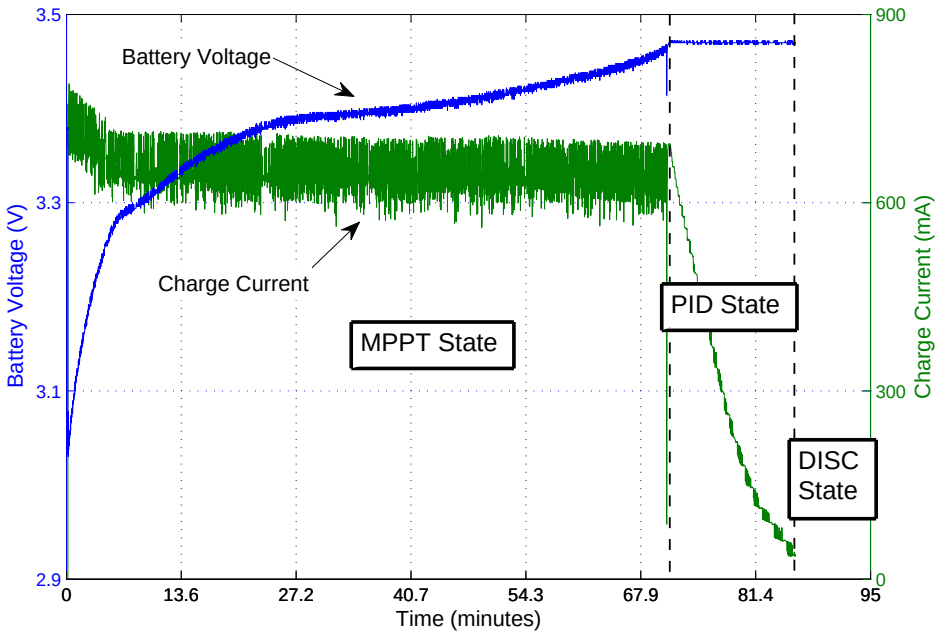


Figure 6.12: A closer look at the first charge of the charge-discharge-charge cycling performed. The system is in MPPT mode during the CC phase and switches to PID mode to achieve CV.

The complete charge-discharge-charge cycle can be seen in Figure 6.13.

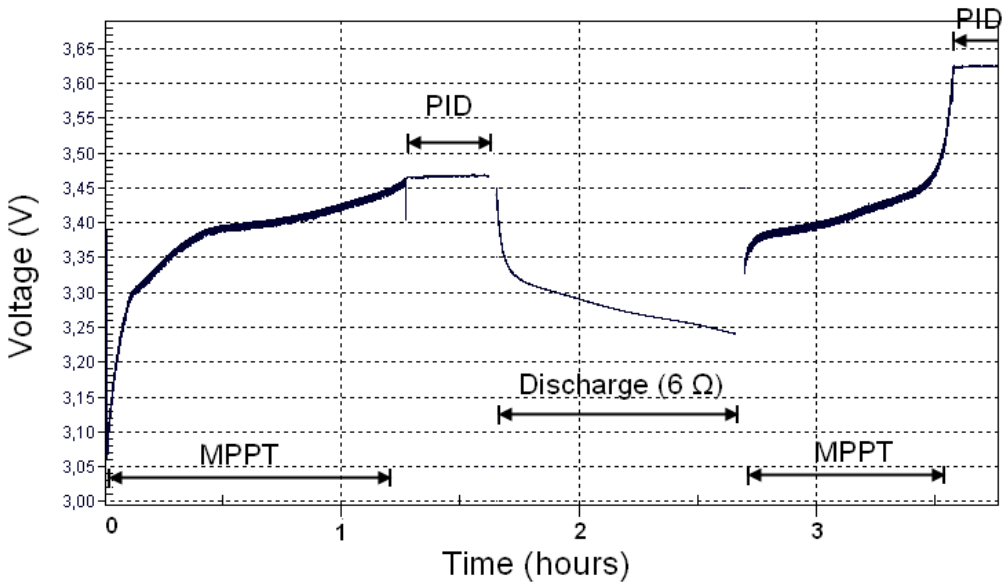


Figure 6.13: The first charge cycle was done with a float voltage (adjustable in software) of 3.47V and the second charge cycle with 3.62V.

6.7 Operation Mode Transitions

To see how the system reacted to a partial charge/discharge while sourcing current for a dummy load potentiometer, the setup in Figure 6.14 was used. One of the terminals of the pot meter was secured while the other was manually put in contact with the regulator output.

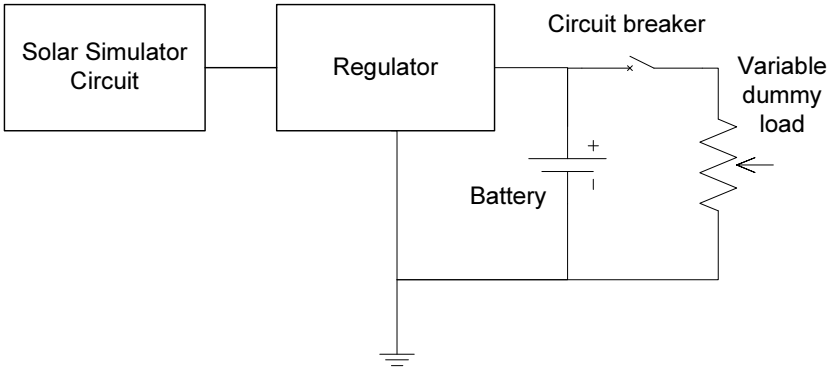


Figure 6.14: Test setup to test the large signal response.

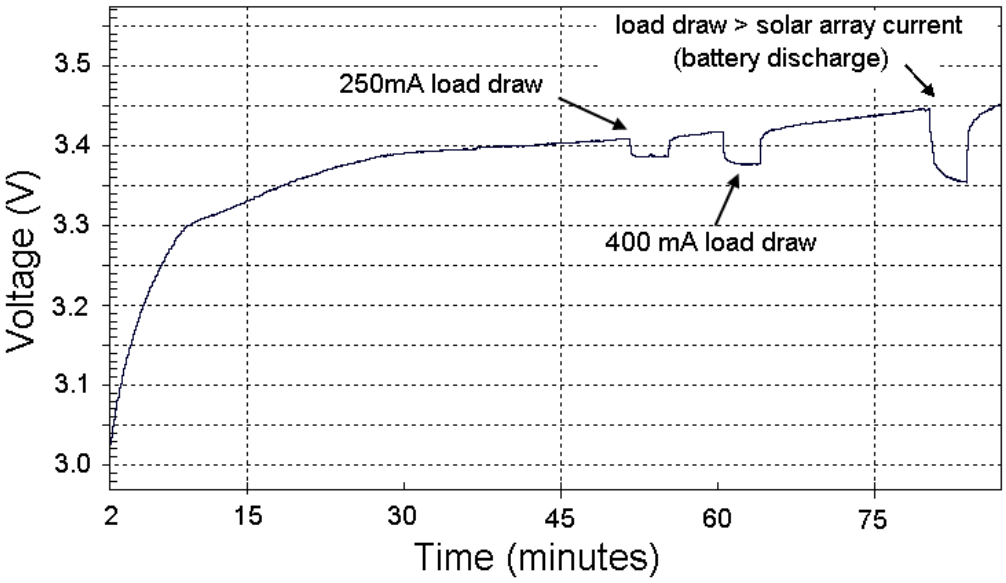


Figure 6.15: MPPT mode. The battery voltage when a secondary load draws current while the battery is charging. The three periods of dummy load draw were done with increasing load; the first two are examples of MPPT charge mode, and the third is MPPT discharge mode where the dummy load draws more current than the solar array can supply.

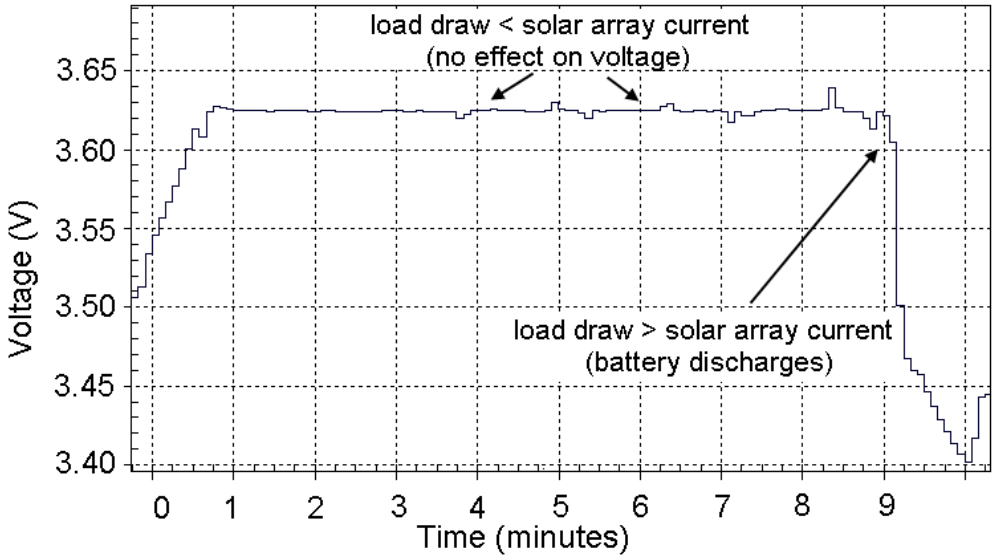


Figure 6.16: PID mode. Similar dummy draws as in MPPT mode were done. The two first that were within the solar array capability did not affect the 3.6V regulated battery voltage. The third attempt with a draw larger than what the panels could handle, resulted in the battery discharging.

6.8 Over-discharge Abuse Tolerance

In this test the battery was discharged to 0.9V—well below the recommended discharge cutoff at 2.0V. There was no sign of the battery struggling, other than the quickly falling terminal voltage. Once the discharge was stopped the battery was left with floating terminals. The next charge cycle was performed

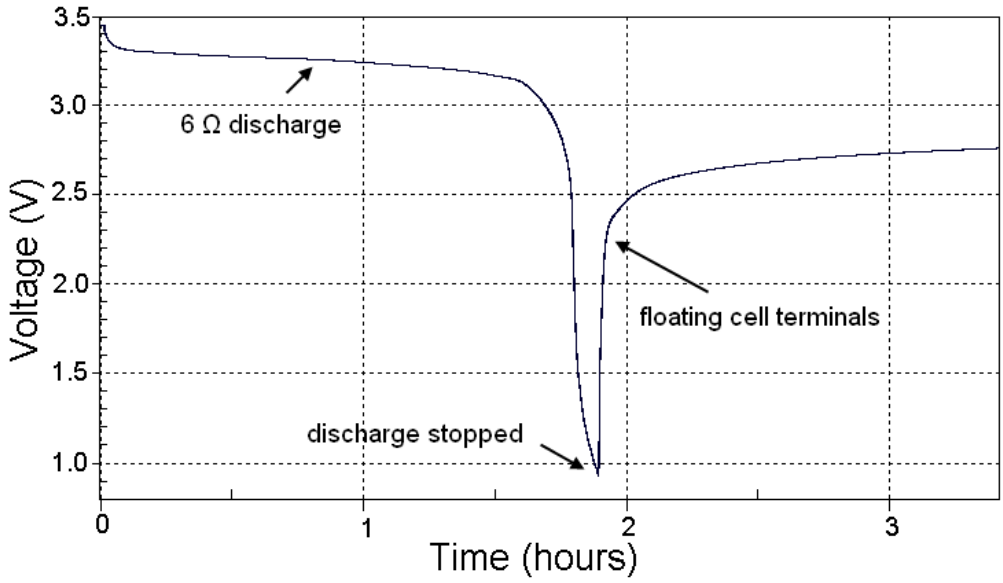


Figure 6.17: Over-discharging the $LiFePO_4$ cell to 0.9V. When the load is removed, the battery voltage approaches 2.7V.

with the same charge current as the other tests—i.e., no pre-qualification with low current charging was performed. No noticeable effect on the next charge curve could be seen, and the battery acted as normal. Moreover, the next discharge curve was nearly identical to the "standard" discharge curve. Thus, from this test, the $LiFePO_4$ cell from A123 Systems seems to be able to withstand a single over-discharge event without any detrimental effect on capacity or discharge voltage levels. However, any long term effects of such abuse are unknown.

6.9 $LiFePO_4$ Discharge in -17°C

In this test a single $LiFePO_4$ cell was charged to 3.6V in room temperature and then discharged through a 10 Ω resistor in a kitchen freezer holding a temperature of -17°C.

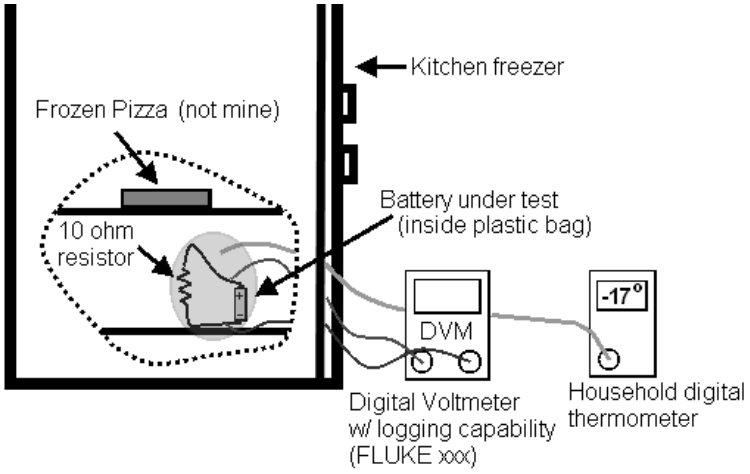


Figure 6.18: Setup for the -17 degree $LiFePO_4$ discharge test.

The cell was then recharged again in room temperature to test if any permanent damage was done. The room temperature discharge curves before and after the cold case were nearly identical and are therefor represented by a single curve. Both the shape and level of the "cold" discharge curve was affected.

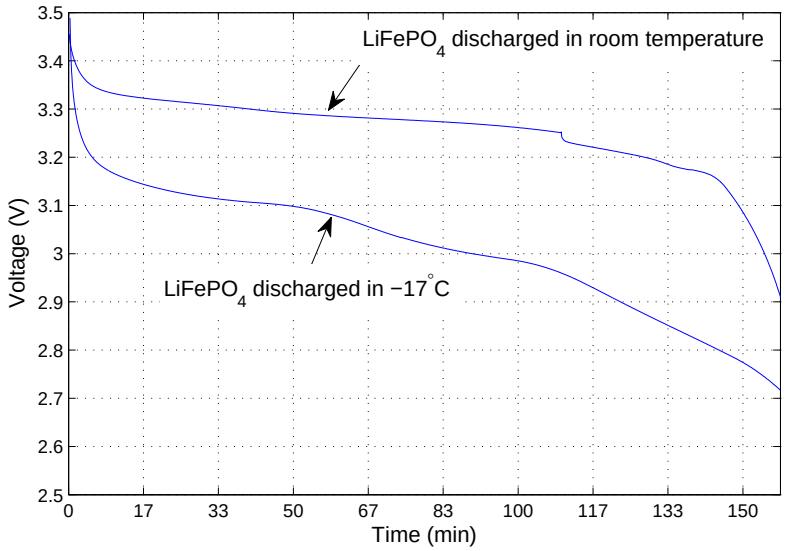


Figure 6.19: $LiFePO_4$ discharge characteristic in cold temperature versus room temperature.

6.10 $LiFePO_4$ vs Lithium Ion Polymer (LiPo)

In Figure 6.20 the $LiFePO_4$ discharge curve can be contrasted against a conventional 3.7V lithium ion polymer cell[1]. Both were discharged through a 10 Ω resistor. The LiPo cell was charged with a dedicated Li-ion charger IC chip, while the $LiFePO_4$ cell was charged manually, with a simple lab supply. Any effect this might have on the "flatness" of the curves should favor the LiPo cell.

[1]The battery is a 1100 mAh VARTA Easypack, which was tested in the early stages of development.

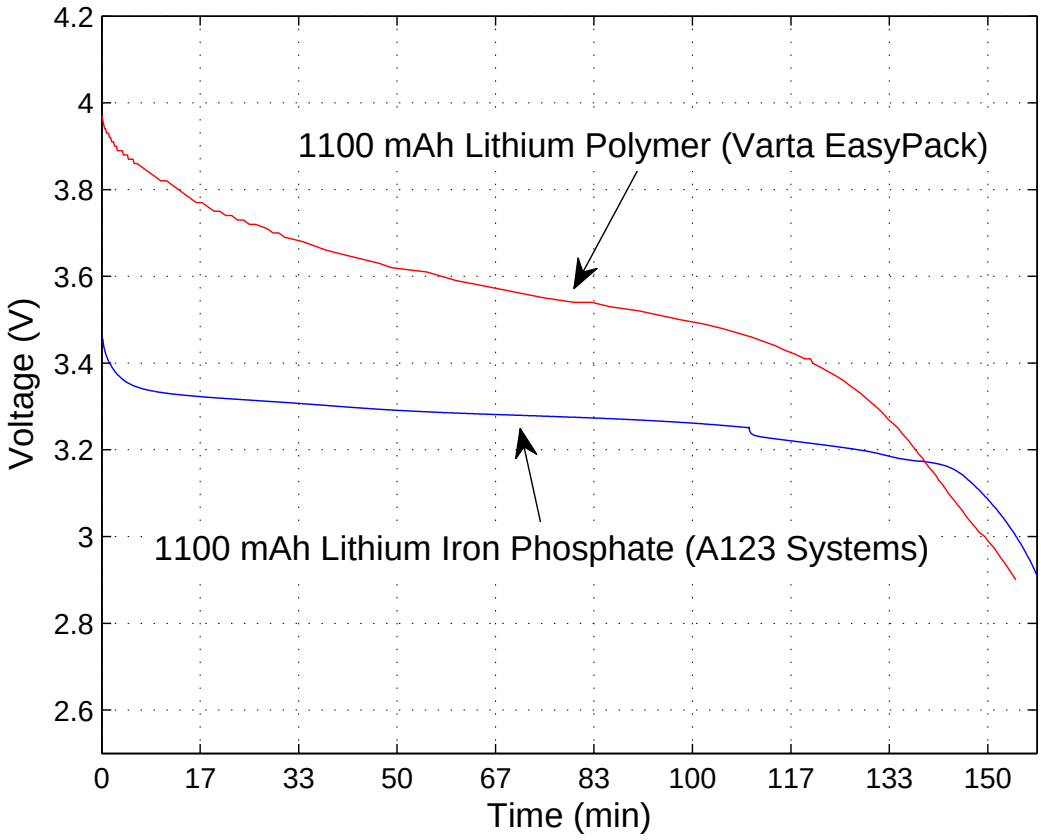


Figure 6.20: Comparison of the "flatness" of the discharge curves of a Lithium Ion Polymer cell and a $LiFePO_4$ cell.

Chapter 7

Conclusions

This thesis has described an electrical power system prototype that is designed to supply a free-trumbling CubeSTAR satellite with a continuous orbital average power of around 2W, based on an assumed EPS efficiency of around 80%. A single regulator has been shown to be sufficient to track the maximum power point of the solar panels and charge the battery.

By implementing the control loop in software, the control system can evolve with the CubeSTAR project as the need for new functionality arises. For example, to speed up the digital control processing, a current-only measuring scheme may be used for the maximum power point tracker[24].

The tests show that by using the proposed 4.4 Ah battery pack and a shallow depth of discharge, an unregulated 3.0V system should be within reach. However, the test also show that the $LiFePO_4$ cells react poorly to cold temperature, and a reliable battery heater should be considered. The NASA report found in [17] states an optimum temperature between 20 and 40°C for most lithium ion batteries, and this is probably a good guideline for lithium iron phosphate cells too.

The flat discharge curve of the $LiFePO_4$ cells are promising, as is their apparent abuse tolerance. But with no flight-history (as far as I know) and without the proven record in space that for example Lithium ion polymer has, the question remains: can these batteries fly? Vacuum testing should be a minimum requirement before launching Lithium Iron Phosphate into

space.

The system fits comfortably within the specified module PCB area. Apart from decoupling and a serial driver for debugging, only the top side was used. A consequence of trying to save precious PCB real-estate on this prototype, each side's protection diode must be placed external from this prototype together with the joint that connects two opposite faced solar panels.

Bibliography

[1] Atmel application note aVR221: Discrete pid controller.

[2] Maxim application note 3645: Correct board layout lowers EMI of switchmode converters.

[3] Spectrolab 28.3% Ultra Triple Junction (UTJ) Solar Cells (datasheet).

[4] Steven F Barret. *Embedded Systems Design with the Atmel AVR Microcontroller*. Morgan & Claypool, New Jersey, 2010.

[5] Christophe Basso. *Switch-Mode Power Supplies: SPICE Simulations and Practical Designs*. McGraw-Hill Professional, New York, 2008.

[6] Todd C. Philip. Snubber Circuits: Theory, Design, and Applications. Technical report, 1994.

[7] Luis Castaner and Santiago Silvestre. *Modelling Photovoltaic Systems using PSpice*. John Wiley & Sons, Ltd., Chichester, 2002.

[8] Charles D. Brown. *Elements of Spacecraft Design*. American Institute of Aeronautics and Astronautics, Inc, Reston, VA, 2002.

[9] Photochemical Dynamics Group. The Basic Physics and Design of III-V Multijunction Solar Cells. Technical report, Ecole Polytechnique Federale de Lausanne, http://photochemistry.epfl.ch/EDEY/III-V_physics.pdf.

[10] Bo H. Cho, Jae R. Lee, and Fred C. Y. Lee. Large-Signal Stability Analysis of Spacecraft Power Processing Systems. *IEEE Transactions on Power Electronics*, 5, No. 1:110–116, 1990.

[11] Chihchiang Hua and Jongrong Lin. A Modified Tracking Algorithm for Maximum Power Point Tracking of Solar Array. *Energy Conversion and Managment*, 45:911–925, 2003.

[12] Abraham I. Pressman, Keith Billings, and Taylor Morey. *Switching Power Supply Design, Third Edition.* McGraw Hill, New York, 2009.

[13] Edward J. Simburger, Daniel Rumsey, David Hinkley, Simon Liu, and Peter Carian. Distributed Power System for Microsatellites.

[14] Zhenhua Jiang and Roger A. Dougal. Multiobjective MPPT/Carging Controller for Standalone PV Power Systems under Different Insolation and Load Conditions. *Industry Applications Conference, 39th IAS Annual Meeting*, 2:1154–1160, 2004.

[15] Vincent L. Pisacane. *Fundamentals of Space Systems, Second Edition.* Oxford University Press, New York, 2005.

[16] Brian Lynch and Kurt Hesse. Under the Hood of Low-Voltage DC/DC Converters. Technical report, 2003.

[17] Barbara McKissock, Patricia Loyselle, and Elisa Vogel. Guidelines on Lithium-ion Battery Use in Space Applications. Technical report, Glenn Research Center, Cleveland Ohio, 2009.

[18] D. O'Sullivan. Space Power Electronics – Design Drivers. *ESA Journal*, 18:1–23, 1994.

[19] Mukund R. Patel. *Spacecraft Power Systems.* CRC Press, Boca Raton, 2005.

[20] Paul R. Sharps. Growth and Development of GaInAs for Use in High-efficiency Solar Cells. *Research Triangle Institute, Annual Subcontravt Report*, page 10, 1992.

[21] Charles R. Sullivan and Matthew J. Powers. A High-Efficiency Maximum Power Point Tracker for Photovoltaic Arrays in a Solar-Powered Race Vehicle. *Power Electronics Specialists Conference, 24th Annual IEEE*, 24:574–580, 1993.

[22] Nannapaneni Narayana Rao. *Elements of Engineering Electromagnetics, Sixth Edition.* Pearson Prentice Hall, New York, 2004.

[23] V. Salas, E. Olias, A. Barrado, and A. Lazaro. Review of the Maximum Power Point Tracking Algorithms for Stand-alone Photovoltaic Systems. *Solar Energy Materials & Solar Cells*, 90, 2006.

[24] V. Salas, E. Olias, A. Lazaro, and A. Barrado. New Algorithm using only One Variable Measurement Applied to a Maximum Power Point Tracker. *Solar Energy Materials & Solar Cells*, 87:675–684, 2004.

[25] Henry W. Ott. *Electromagnetic Compability Engineering*. John Wiley & Sons, Inc, New Jersey, 2009.

Appendix A

Solar Panel Simulation Circuit

A.1 Motivation

The reliance on sunshine—a commodity lacking in most labs—becomes a challenge when working with a system that uses solar cells as its power source. This is especially true for work on the part of the regulation system that deals with the maximum power point tracking on the solar cells. The non-linear output characteristics of a solar cell means it is inherently an unstable (both in current and voltage) power source—a behavior that can not be replicated with the typical lab power supply.

While xenon-lamp based solar simulators are available, their cost, size and heat make them impractical for the purposes described above. To overcome these issues, a power supply was built that simulates the behavior of a single side's panel (i.e., two parallel strings of two cells in series).

A.2 Simulation Goals

As discussed in Chapter 2, the CubeSTAR satellite will be mounted with Spectrolab's 28.3% Ultra Triple Junction solar cells. The cell properties that are relevant for this section are summarized in Table A.1, and the resulting

I-V curve[1] for a single cell is shown in Figure A.1. Due to the negligible difference between the characteristic I-V curve of the circuit's chosen silicon diode (IN4148) and that of the Ge-GaAs-InP based solar cell, the only remaining parameters of interest are the short-circuit current and open-circuit voltage (i.e., we know the shape but need curve's end points).

Table A.1: Spectrolab UTJ Characteristics

Current density (J_{sc})	17.05mA/cm^2
Open-circuit voltage (V_{oc})	2.665V
Cell area[2] (A)	26.62 cm^2

By assuming a high-quality solar cell (i.e., low series resistance R_s, and high shunt resistance R_{sh}), the short-circuit current is equal to the photo-generated current I_L. Thus, for a single cell

$$I_{sc} \approx I_L = \text{Area} \times J_{sc} = 453\text{mA} \tag{A.1}$$

and with an open-circuit voltage of 2.665V coupled with the fact that the curve is of exponential nature, we have the information needed to replicate the behavior in the simulator circuit.

[1]The plot is from Spectrolab's datasheet, to be found here:
http://www.spectrolab.com/solarcells.htm

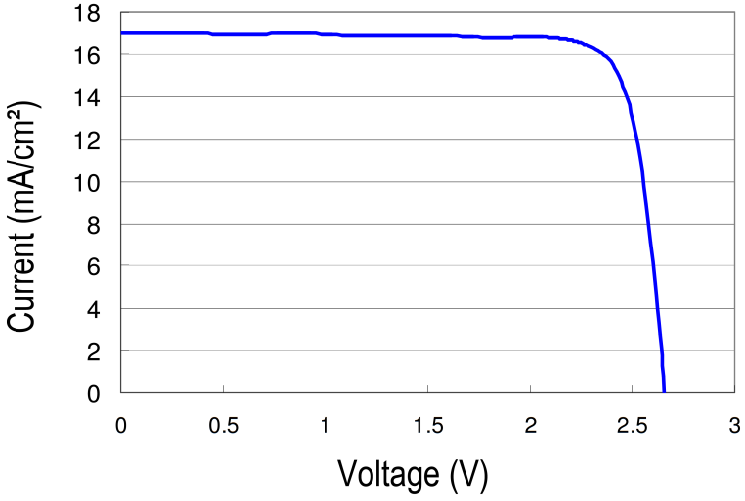


Figure A.1: I-V characteristics for a single 28.3% Ultra Triple Junction (UTJ) solar cell.

A.3 SPICE Circuit Implementation

The solar cell simulation circuit consists of a constant-current generator that acts as a reference to a pair of op-amps that, in turn, boosts the generator current while maintaining the desired I-V curve.

In the SPICE circuit diagram shown in Figure A.3, U_1 operates as a reference generator whose current passes through the diode D_1, before being sensed over the low-side shunt R_{sh1}. The diode D_2 and op-amp U_2 perform temperature correction, while the op-amp U_3 and bipolar transistor Q_1 make up the last output amplifier stage.

The solar cell's irradiance-dependent short-circuit current, I_L, is simulated by adjusting the R_b resistor in Figure A.3. To simulate the open-circuit voltage of the solar cell, which in reality is a function of temperature and the number of series cells, the R_a resistor can be adjusted in order to move the open-circuit voltage point.

It's assumed that all four solar cells that make up the four panels on each face are exposed to the same insolation and temperature. The series-parallel

panel configuration means that the short-circuit current and open-circuit voltage from Table A.1 both can be multiplied by two. By choosing a 50Ω trimpot for R_b and a 2kΩ trimpot in series with a 3.3kΩ resistor for R_a, the circuit can be used to simulate the entire range of expected insolation and the temperature-dependent voltage range from about 4V to 6V.

In the SPICE plot in Figure A.2, $R_b = 36\Omega$ and $R_a = 4.3\text{k}\Omega$ were arbitrarily chosen with the resulting I-V and P-V curves shown.

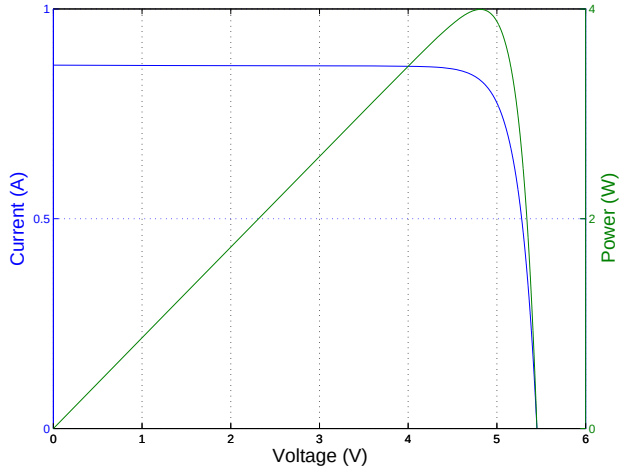


Figure A.2: Simulated current-voltage and power-voltage behavior of the SPICE circuit in in Figure A.3. The short-circuit and open-voltage values can be adjusted within a suitable range to take into account varying insolation and temperature.

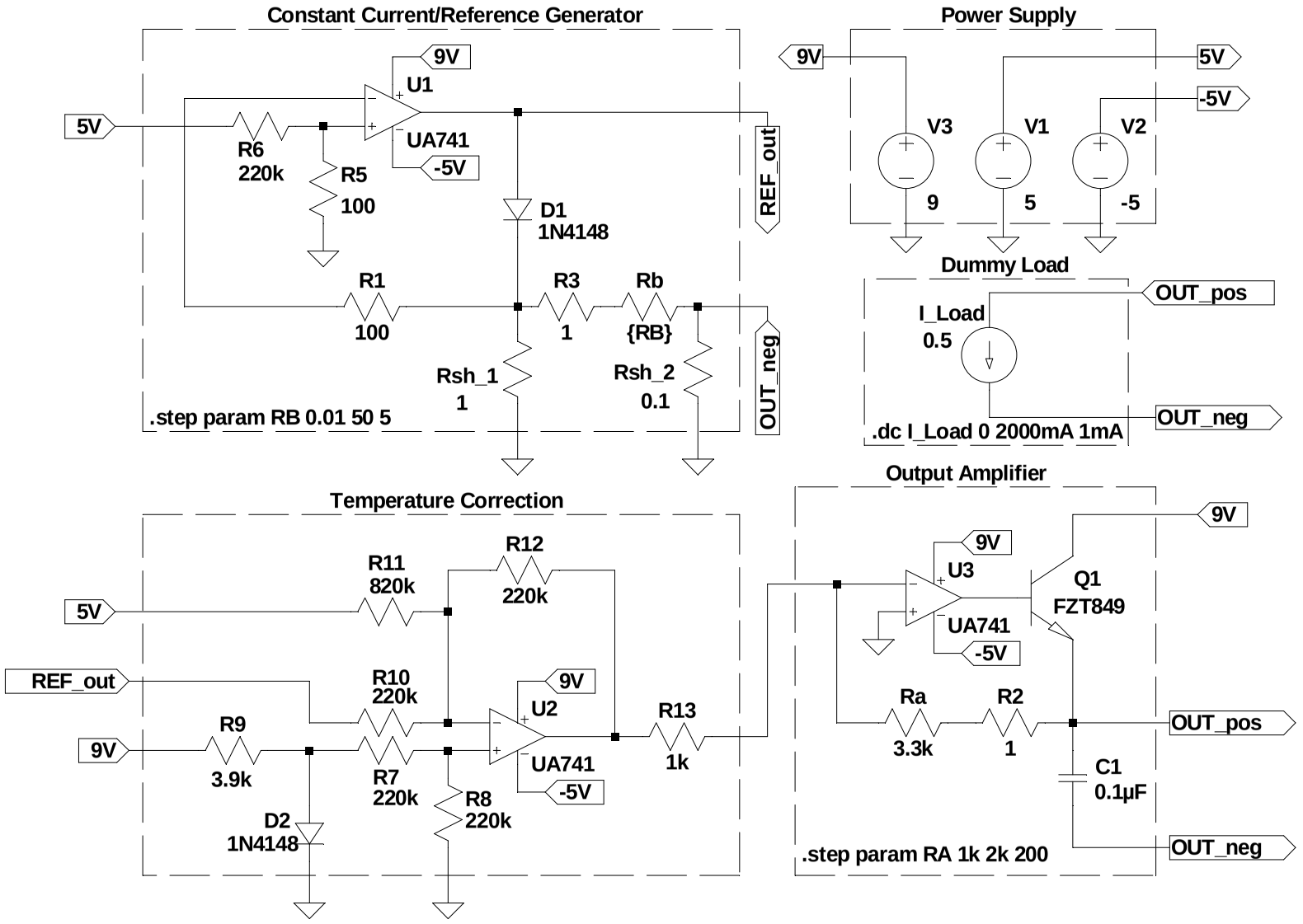


Figure A.3: Circuit diagram of the solar cell simulation model.

A.4 PCB Realization

The SPICE circuit in Figure A.3 was realized on a two-layer PCB, illustrated in Figure A.4 and Figure A.5. Some minor rework (not shown) was needed before the circuit worked as intended, and a heatsink was also added to prevent the transistor from overheating.

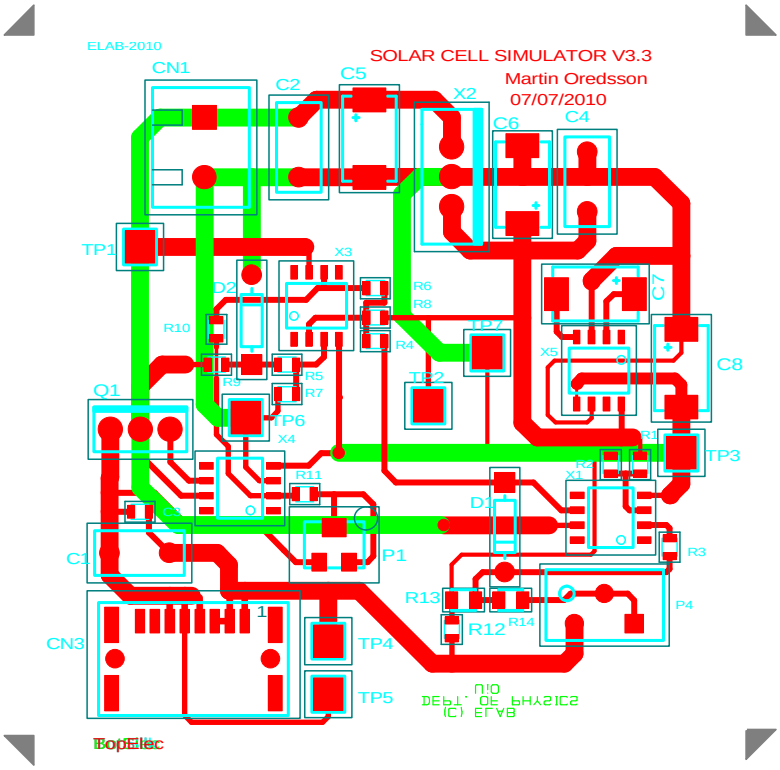


Figure A.4: Two layer circuit layout. *Note: the ground connection for the shunt resistor R12 should be re-routed with a shorter track.*

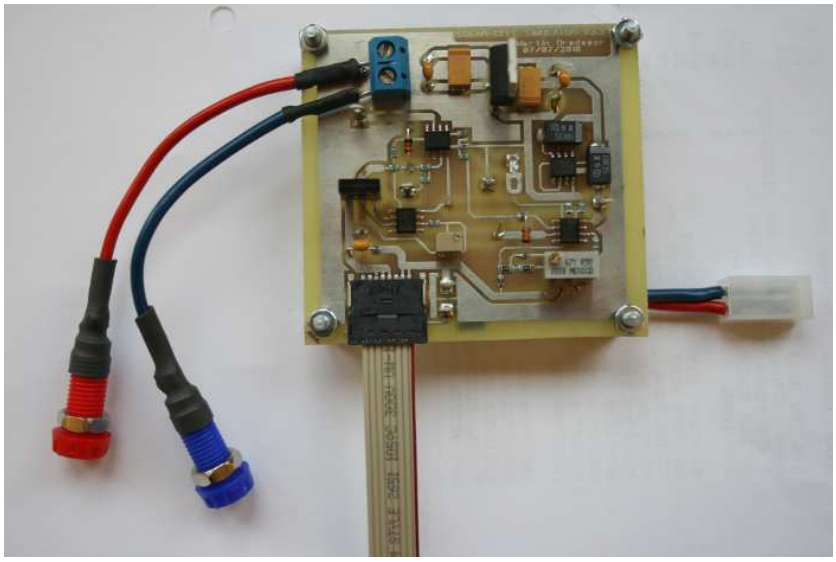


Figure A.5: The finished mounted PCB with a 8-pin male output connector from ERNI.

A.5 Results

A 50Ω potentiometer was used as a varying load to generate the I-V curves in Figure A.6. These curves correspond well to the SPICE simulation and the I-V curve from the UTJ datasheet, enabling us to use this circuit as the power source while working with the EPS system.

Table A.2: Solar cell simulation circuit parameters

Short-circuit current (I_{sc})	$0.15 - 0.86$A
Open-circuit voltage (V_{oc})	$4 - 6$V

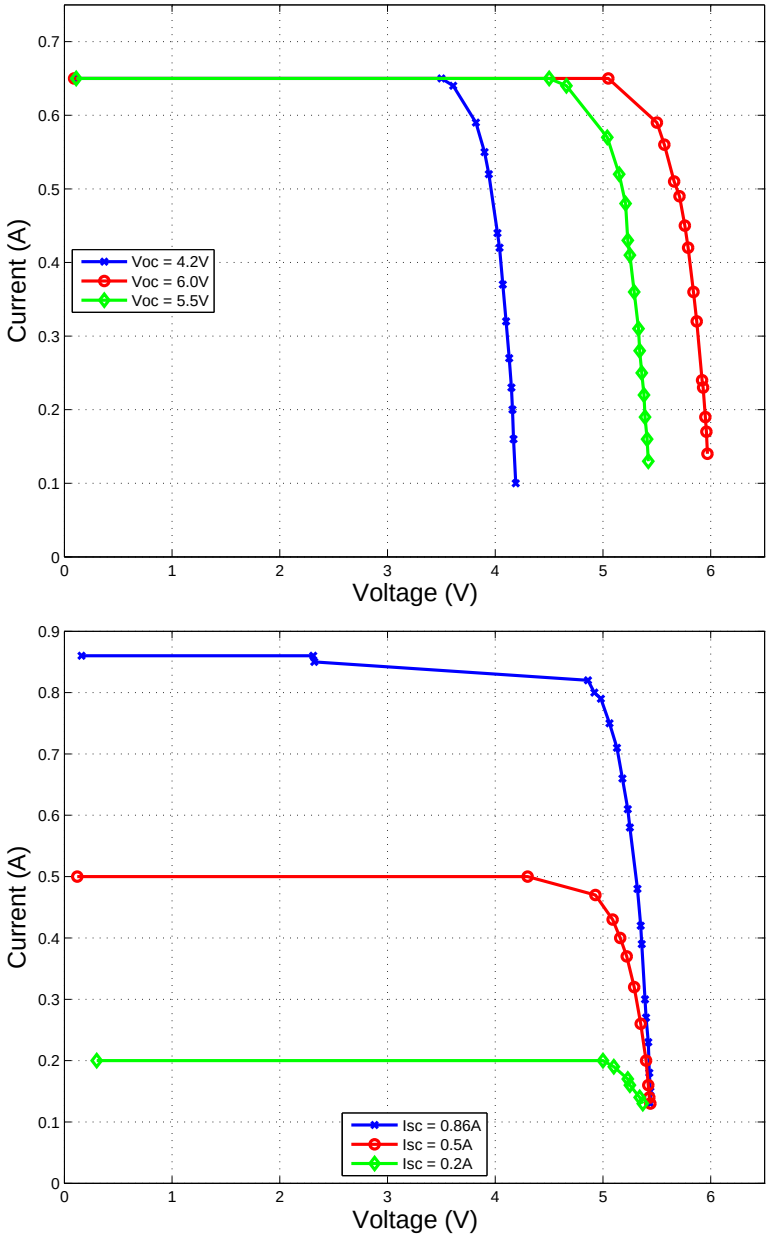


Figure A.6: The open-circuit voltage (top) can be adjusted from 4V to 6V to take into account the expected temperature range experienced by the real solar cells. The short-circuit current (bottom) can be adjusted from 0.15A to 0.86A to take into account the expected insolation range experienced by the real solar cells. *Note: These curves do not really flatter the circuit. With more measurement points the output truly is exponential. See e.g. Figure 6.8.*

Appendix B

Triple Junction Solar Cell SPICE Model

B.1 Creating a Subcircuit in SPICE

To achieve a modular simulation design we can create *subcircuits* within the SPICE environment. One of the benefits of using subcircuits is that we can encapsulate the complete model[1] of our solar cell within a single re-usable block of code which then can be associated with an appropriate circuit symbol that can be included as any other component in a spice circuit diagram.

The first two lines of the program defines a subcircuit named `utj` using the `.subckt` keyword, with nodes 100, 101, and 102:

```
.subckt utj 100 101 102
+params:area=1,j0a=1,j0b=1,j0c=1,jsc=1,rs=1,rsh=1
```

The listed parameters are cell area, diode saturation currents `j0x` for each junction semiconductor material, short-circuit current density `jsc`, series re-

[1]This approach also leaves the door open to later include input parameters such as sun irradiance, temperature variations and load effects.

sistance `rs`, and shunt resistance `rsh`. All are loaded with dummy values at this stage which are replaced by real values once the circuit is excited.

Next, the cell's real property as a irradiance-dependent current source is modeled with a voltage-dependent current source, or a *G-device* in SPICE terms. The G-device's input voltage at node 102 (in volts) will be used to represent the irradiance in W/m^2, and is described as follows:

```
Gxxx n+ n- value={expression}
```

where the current towards `n+` is determined by the expression between the curly brackets. The expression that will be used to represent the short circuit current is

$$I_{sc} = \frac{J_{sc}A}{1000}G \tag{B.1}$$

so, by naming it `Girrad`, line 3 becomes

```
Girrad 100 101 value={(jsc*area/1000)*v(102)/1366}
```

In the next six lines, the three pn-junctions of the solar cell are represented by diode definitions and their models. The general form of the diode definition in SPICE is

```
d[name] [anode] [cathode] [modelname]
.model ([modelname] d [parmtr1=x] [parmtr2=y] . . .)
```

but we only need the saturation current here, and if we take the Germanium junction as an example, the code becomes

```
DGe 103 104 Ge
.model Ge D(IS={j0a*area})
```

where the diode's reverse bias saturation current is set to be proportional to the cell area as it should. Finally, the shunt- and series resistances are specified as `rsh` and `rs`, before the subcircuit `ends` statement it called.

The complete code for the subcircuit is

```
1  .subckt utj 100 103 102 params: area=1, jsc=1, rs=1,
       rsh=1
2  + j0a=1, j0b=1, j0c=1
3  girrad 100 101 value={(jsc*area/1000)*v(102)/1366}
4  d1 101 104 gainp
5  .model gainp d(is={j0a*area})
6  d2 104 105 gaas
7  .model gaas d(is={j0b*area})
8  d3 105 100 ge
9  .model ge d(is={j0c*area})
10 rsh 101 100 {rsh}
11 rs 101 103 {rs}
12 .ends utj
```

Listing B.1: Using a SPICE sub-circuit to model a triple-junction solar cell.

In the next section, we'll throw the subcircuit into an excitation circuit and create some plots.

B.2 Excitation Circuit

The subcircuit in Figure B.1 is excited with two voltage sources; `Vbias` and `Virrad`, where the latter represents the irradiance.

The characteristic I-V curve is then easily obtained by sweeping the bias source over the operating range of the cell while Virrad source is set to 1366 W/m^2. Comparing the curve against Spectrolab's own datasheet plot in Figure A.1, the saturation current within the subcircuit can be adjusted until a satisfactory match is obtained.

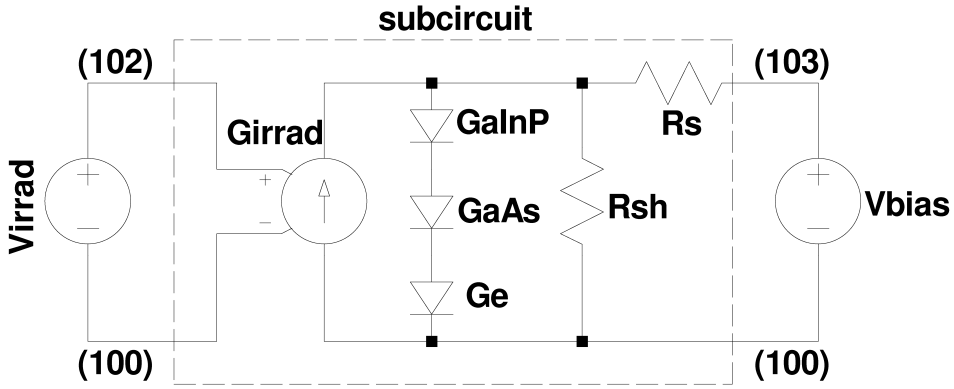


Figure B.1: SPICE circuit used to generate the plots in this section.

By parameterizing the value of our irradiance source with the following lines

```
Virrad param 32 0 dc {ir}
.step param ir list 0 500 1000 1366
```

we can plot four I-V curves for four different values of insolation to illustrate the current-irradiance relationship.

Although there are other factors that come into play regarding the temperature dependence of the solar cell, one of the dominant factors, and one which we have at arms length in our model, is the saturation current I_0. By parameterizing it with

```
.step param I0 list 9.5e-25 9.5e-20 9.5e-15 9.5e-10
```

and combining the result with the fact that Spectolab's UTJ cells have a negative temperature coefficient of -5.9 mV/°C, the information can be presented graphically as in Figure 2.8.

The complete code for the excitation circuit is:

```
.include UTJ_subckt.lib
X1 0 31 32 utj params: area=26.62 rs={RS} rsh=300
+ jsc=17.05 j0a=9.5e-20 j0b=1e-16 j0c=1e-14
Vbias 31 0 dc 0
Virrad 32 0 dc 1366
RS = 50e-3
*.param RS=0.001
*.step param RS list 0.0001 0.001 0.01 0.1 1
.plot dc i(vbias)
.dc vbias 0 3 0.01
.probe
.end
```

Listing B.2: Exciting the sub-circuit.

Appendix C

Discrete PID Controller

A PID controller[1] is used to stabilize the voltage during the constant voltage charging phase[1]. By measuring the *process* value (battery voltage) and comparing this value to the reference (the preset battery float voltage), the error can be calculated and used to determine the new process input. This input will then try to adjust the measured process back to the desired setpoint.

Furthermore, the PID controller is capable of manipulating the process input based on the history and rate of change of the signal. This can be used to our advantage in the attempt to tame the battery voltage once it approaches the float voltage. The region between 3.5 and 3.6V is characterized by a large voltage gradient and once the battery approaches 3.6V and is saturated with charge, a continued attempt to put charge in the battery is answered with a quick increase in voltage. By implementing a PID controller, the voltage overshoot is prevented and a more accurate and stable control is achieved.

C.1 Proportional Term

The Proportional term returns a system control input that is proportional to the current error value. Except in the cases of zero control input or a system process value equal to the desired reference, using only a P term gives a

[1] Although it's implemented as a full PID controller, the derivative gain is currently set to a very low value, so it's perhaps closer to a PI-controller. Se note about tuning.

stationary error. A high valued P term results in a large change in the output for a given change in the error, and/but setting the it too large will result in a unstable system. Setting it low results in a less responsive/sensitive controller, which might be unable to react quickly enough to variations in the input.

The proportional term output is given by

$$P_{out} = K_p \cdot e(t) \tag{C.1}$$

where K_p is the proportional (or gain) tuning parameter, e is the difference (error) between the measured and desired values at the present time, t.

C.2 Integral Term

The Integral term adds the sum of the previous errors to the system control input, and as such is proportional to both the magnitude and duration of the error. By summing up the error over time and multiplying with the integral gain K_i, the I-term returns the scaled accumulated offset that should have been corrected previously.

The integral term output is given by

$$I_{out} = K_i \int_0^t e(\tau)\, dt \tag{C.2}$$

where K_i is the integral tuning parameter, τ is dummy integration variable, and e(t) the time dependent error as before.

By adding the I term to the P term, the stationary error from the P only case, is removed. However, since the I term is also responding to accumulated errors from the past, overshooting the setpoint is a typical side-effect. These issues should be dealt with by tuning the loop with one of the plethora of tuning methods available.

C.3 Derivative Term

The derivative term deals with the rate of change of the process error, and is calculated by determining the slope of the error over time. This rate is then multiplied by the derivative gain K_d:

$$D_{out} = K_d \frac{de(t)}{dt} \tag{C.3}$$

The end effect of the D term is to slow down the rate of change of the controller output, which might be used to counter the overshoot effect from the I term.

C.4 Implementation

The three terms are then added together and returned as the controller output u(t)

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)\, dt + K_d \frac{de(t)}{dt} \tag{C.4}$$

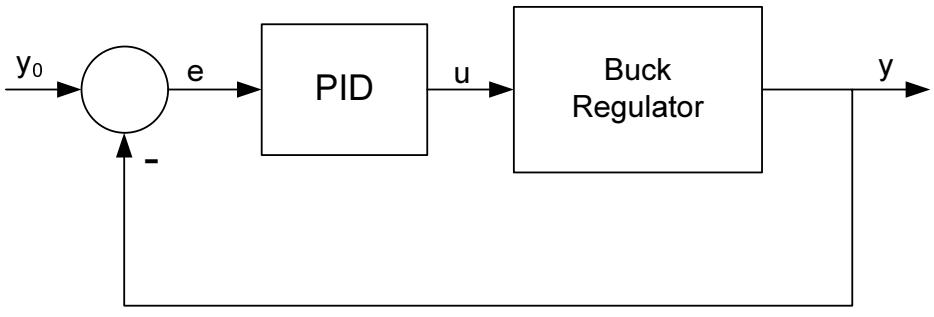


Figure C.1: When the system is in the CV state the PID regulator controls the buck converter dutycycle.

In the digital world, integrals become sums and derivatives become slopes, so

$$\frac{de(t)}{dt} \rightarrow \frac{e(t_k) - e(t_{k-1})}{\Delta t} \tag{C.5}$$

$$\int_0^{t_k} e(\tau)\, d\tau \rightarrow \sum_{i=1}^{k} e(t_i)\Delta t \tag{C.6}$$

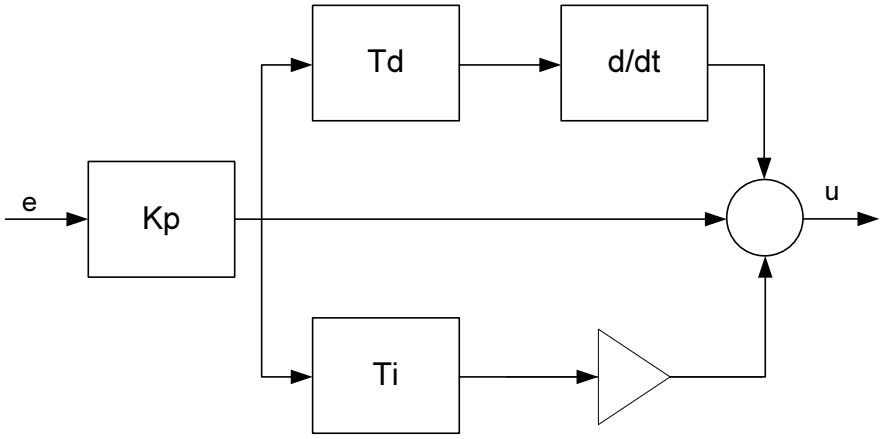


Figure C.2: PID regulator block diagram.

The discrete controller is thus described by

$$u(n) = K_p \cdot e(n) + K_i \cdot \sum_{k=0}^{n} e(k) + K_d \cdot [e(n) - e(n-1)] \tag{C.7}$$

where $K_i = \frac{K_p T}{T_i}$ and $K_d = \frac{K_p T_d}{T}$

C.4.1 A Note on Loop Tuning

A plethora of methods exists in the literature, but with limited time to invest in the interesting topic of PID tuning, the PID regulator was tuned by trial

and error until a satisfactory result was achieved. The P, I, and D terms are in any case readily available and easy to change in the PID.h header file.

Appendix D

Two Wire Interface (TWI)

This appendix provides a quick breakdown of the TWI bus and the necessary background information for the described telemetry implementation in Section 4.7.

D.1 Electrical Characteristics

Atmel's I^2C compatible TWI bus operates at 100 kbp/s and consists of two active lines; Serial Data (SDA) and Serial Clock (SCL). Both are bidirectional open-collector lines that require pull-up resistors. Any device connected to the TWI bus has a unique address and acts as either a master or slave, with the former being the initiator of data a transaction.

D.2 Start and Stop Conditions

On a idle bus, both SDA and SCL are high. To initiate a transaction, a device must first pull SDA low before pulling SCL low, as in Figure D.1. This is called a `START` condition (S). By initiating a transaction with the start condition, a device automatically becomes the master, and all other connected devices are considered slaves until a `STOP` condition (P) is issued.

The stop condition, being the start condition's dual, is initiated by first releasing the SCL line, followed by the release of the SDA line.

Further, a master can prevent other masters from starting their own transactions by sending a new start condition before stopping the current one. This is known as a `REPEATED START`, and can be used by the master to address another slave without first generating a stop condition.

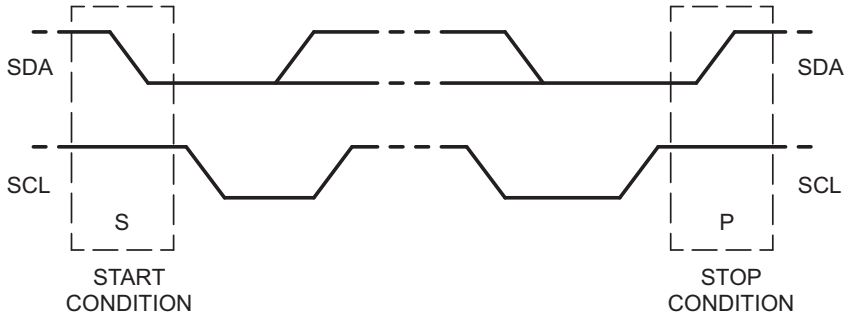


Figure D.1: Start and stop conditions.

D.3 Address

After the start condition, the next seven bits contain the `ADDRESS` (A) of the slave followed by a `R/`$\overline{\texttt{W}}$ bit which specifies the direction of the transaction. When a slave recognizes its address it will `ACK`nowledge by pulling SDA low in the next clock cycle. Meanwhile, all other slaves should keep the TWI lines released (hence the need for pull-ups), and wait for the next `START`–`ADDRESS`–`R/`$\overline{\texttt{W}}$ sequence.

Figure D.2: The first byte after the start condition.

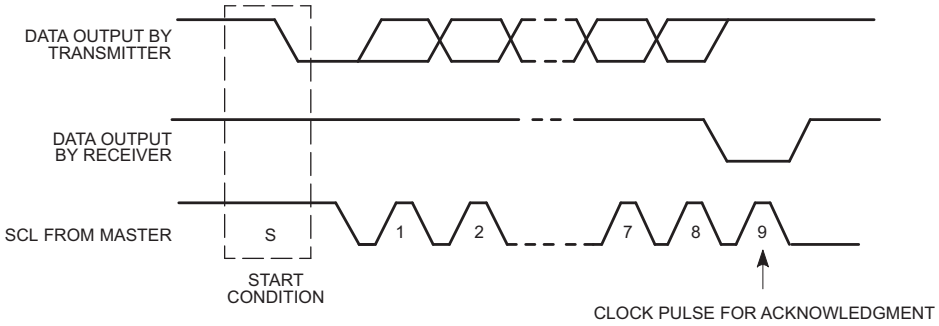


Figure D.3: Acknowledge on the TWI bus.

D.4 Data Transfer

In the bottom part of Figure D.4, the read/write bit is low, indicating a master write transaction. The data transfer will proceed once the slave has acknowledged its address, and the transaction will be on-going for a arbitrary number of `DATA` packets as long as no stop condition is initiated.

The master read transaction, shown in the top part of Figure D.4, is initiated by setting the read/write bit high. When the slave acknowledges the address, the master can start receiving data from the slave. Again, there is no inherent limitation to the number of data packets that can be transferred—the transaction is terminated when the masters sends a `NACK` followed by a stop condition.

A third transfer modus is possible by combining the write and read transactions into a combined transaction. By sending a `REPEATED START`–`ADDRESS`–`R/`$\overline{\texttt{W}}$, the master can at any time change the direction of the data transfer within a transaction.

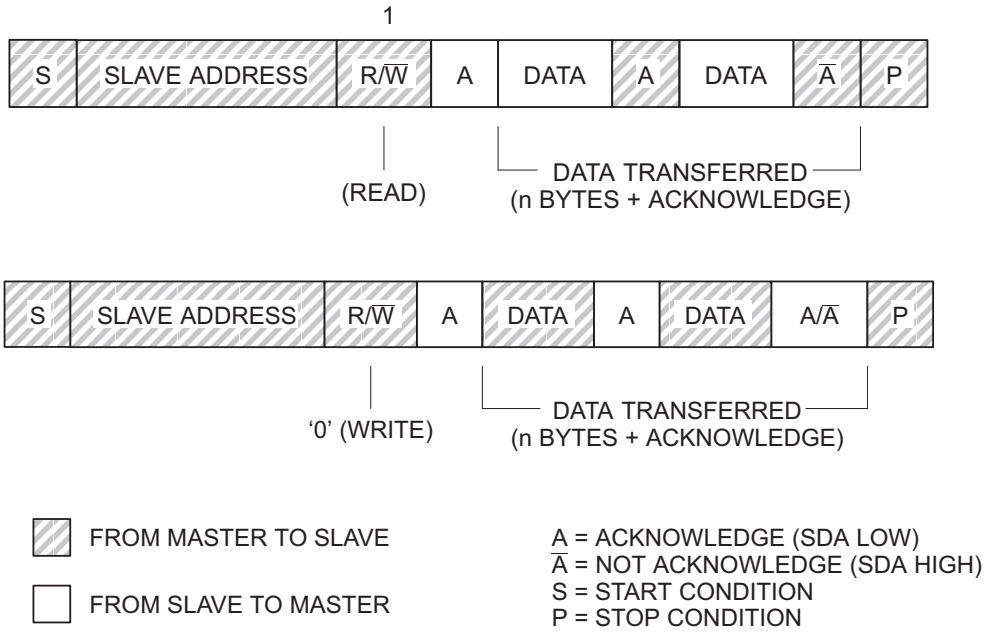


Figure D.4: Master read (top) and write (bottom) transaction.

D.5 Clock Stretching

Although the clock is always controlled by the master, any device on the bus can hold the clock low at any time, thus providing a slave a way to hold back a transaction in case it needs more time to process data. When this happens, the master cannot continue clocking, and the slave is said to perform *clock stretching*.

Appendix E

Production Files

The following sections contain the parts list, schematics and Gerber files that were created with the computer assisted software suite Zuken Cadstar V12.1.

E.1 Parts List

Parts List

CADSTAR Design Editor Version 12.1

Design: C:_workdir\Cadstar_wrk\V1.5\V1.5d\EPS_Prototype_v1.5d.scm

Design Title:
EPS CubeSTAR Prototype

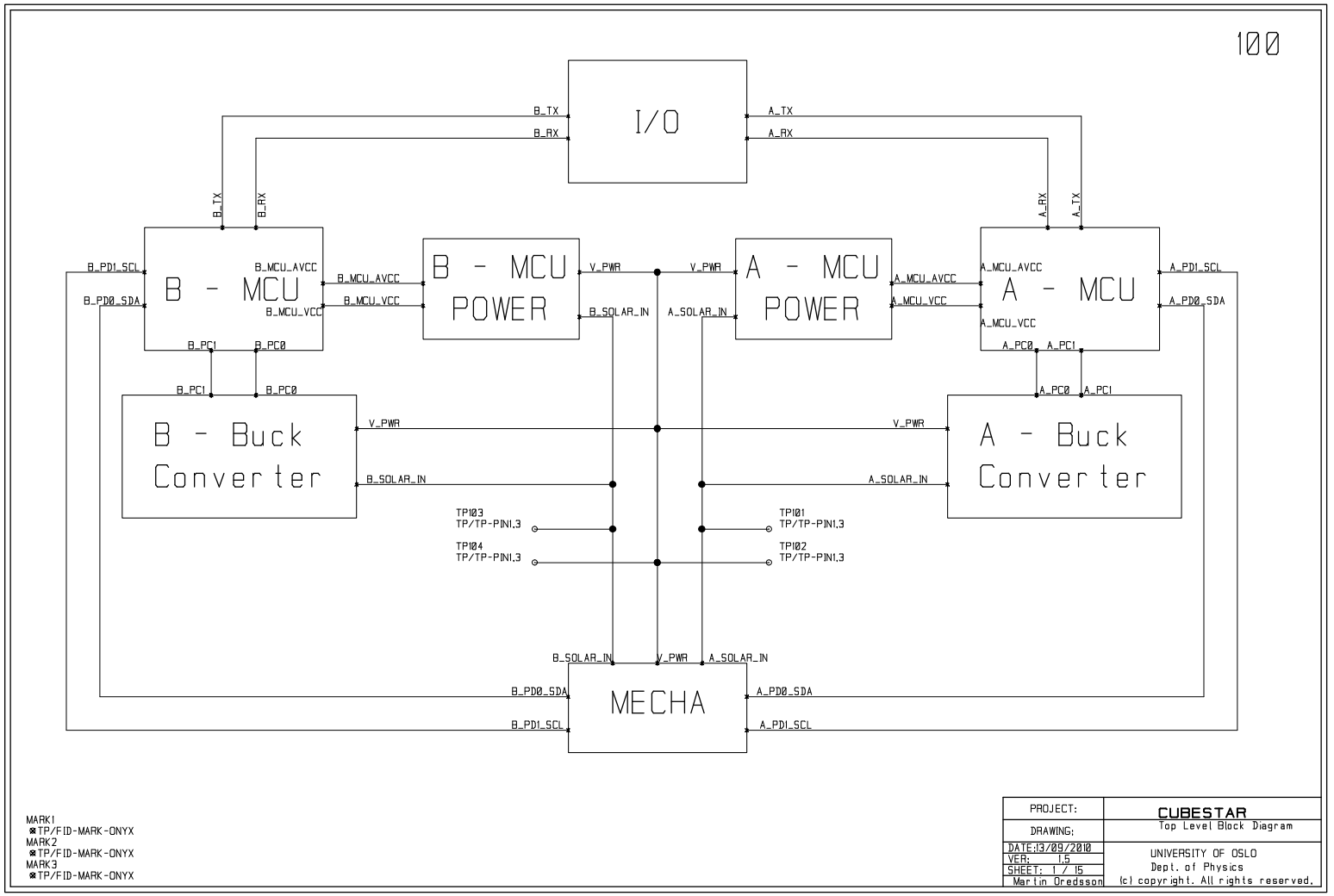
Date: 16. september 2010
Time: 12:11

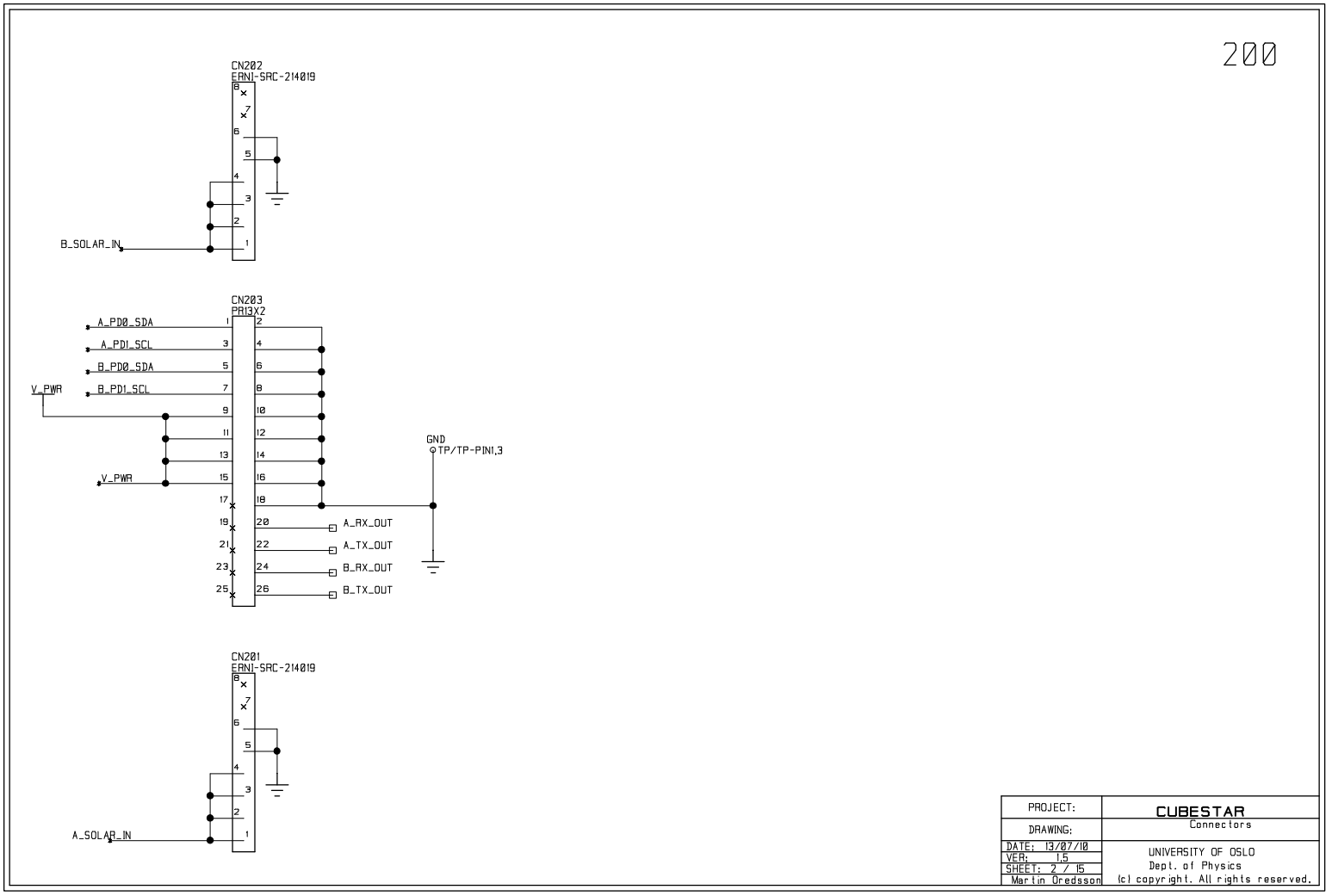
Part Name/Number	Description	Qty.	Comps.
X-XX-XXX-XX	ATMEL AVR MICROCONTROLLER	2	U701 U1701
X-XX-XXX-XX	BURR BROWN CURRENT SHUNT MONIT	2	X501 X1501
E-65-759-63	10% 16V 0603 X7R	49	C303 C403 C501-502 C601 C701-713 C801-802 C804-805 C901-905 C1303 C1403 C1501-1502 C1603 C1701-1713 C1802-1805
E-65-758-49	10% 50V 0603 X7R	2	C803 C1801
E-65-757-81	10% 50V 0603 X7R	2	C307 C1307
RS-248-007	ERNI 1.27mm SRC SINGLE ROW MIN	2	CN201-202
E-43-714-31	13X2 TYCO PINROW ANGELED	1	CN203
E-43-704-33	5X2 SCOTT ELEC. PINROW	2	CN701 CN1701
F1713895	SMD SCHOTTKY DIODE 20V/2A	2	D401 D1401
F-9550780	SMD VERY LOW DROP SCHOTTKY DIO	2	D301 D1301
F1635948	WURTH CHOKE, SMD, 100UH	2	L301 L1301
E-58-830-04	KOA SMD-W COIL	2	L601 L1601
F-1422325	MOSFET DRIVER, DUAL, 1.5A	2	X402 X1402
F-XX-XXX-XX	+/-5% 0402 LQG15H-series	2	L602 L1602
F-1663786	MICROPOWER OPAMP	6	X801-803 X1801-1803
F-1197392	CHARGE PUMP VOLTAGE DOUBLER	2	X401 X1401
RS-661-6468	LDO, 3.0V	2	X601 X1601
E-60-440-02	RESISTOR KOA 0603 1% 0.1W	8	R304 R803 R811-812

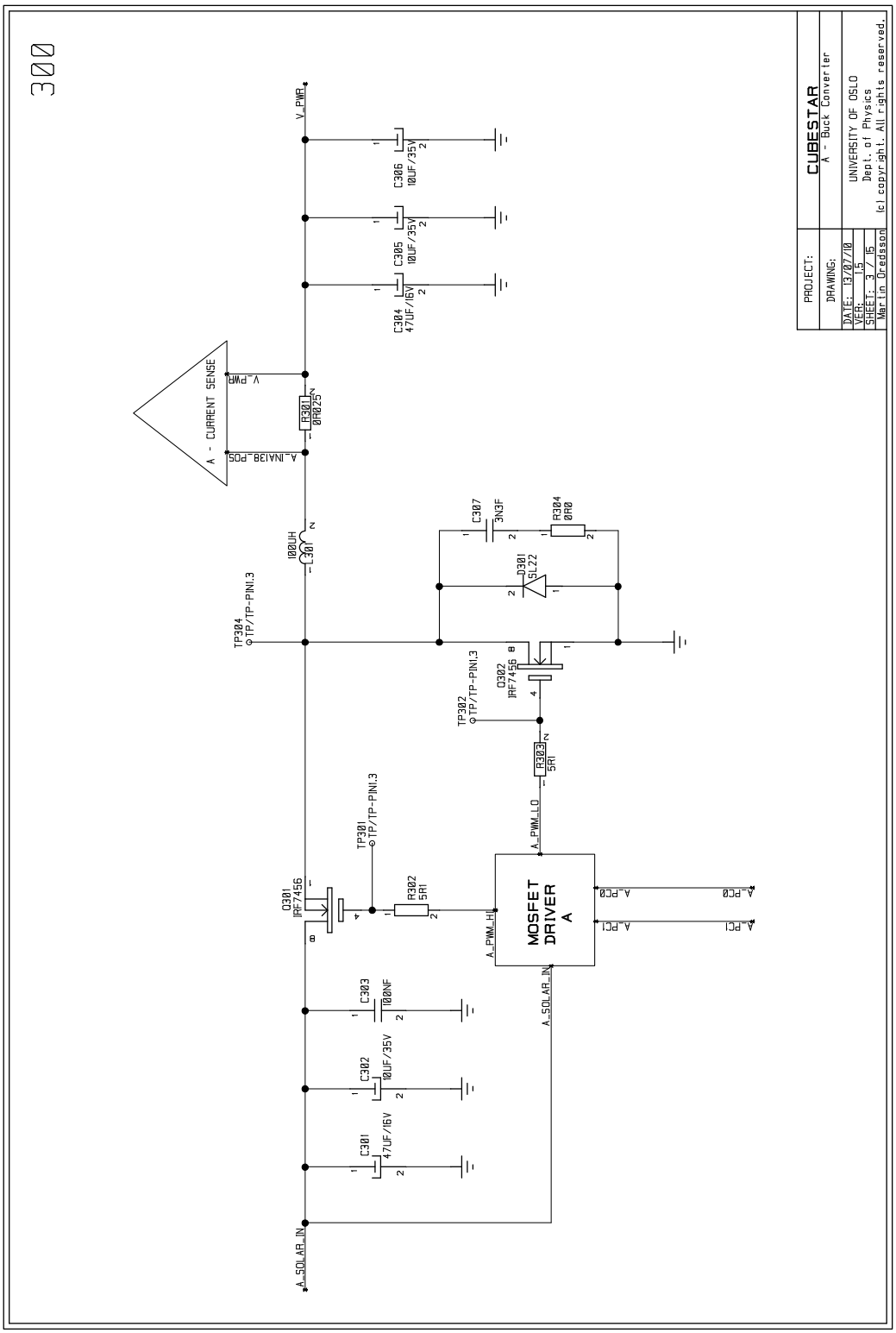
			R1304
			R1805
			R1808
			R1811
F-1703806	CURRENT SENSE RESISTOR 1206 1%	2	R301
			R1301
E-60-452-64	RESISTOR KOA 0603 1% 0.1W	4	R801-802
			R1807
			R1809
E-60-450-25	RESISTOR KOA 0603 1% 0.1W	6	R601
			R703
			R804
			R1601
			R1703
			R1803
E-60-445-80	RESISTOR KOA 0603 1% 0.1W	2	R602
			R1602
E-60-448-46	RESISTOR KOA 0603 1% 0.1W	4	R701-702
			R1701-1702
E-60-451-08	RESISTOR KOA 0603 1% 0.1W	2	R807
			R1812
E-60-451-24	RESISTOR KOA 0603 1% 0.1W	2	R806
			R1804
E-60-451-40	RESISTOR KOA 0603 1% 0.1W	2	R809
			R1806
E-60-451-57	RESISTOR KOA 0603 1% 0.1W	2	R808
			R1801
E-60-454-05	RESISTOR KOA 0603 1% 0.1W	2	R501
			R1501
E-60-451-73	RESISTOR KOA 0603 1% 0.1W	2	R805
			R1810
E-60-438-71	RESISTOR KOA 0603 1% 0.1W	4	R302-303
			R1302-1303
E-60-452-56	RESISTOR KOA 0603 1% 0.1W	2	R810
			R1802
F-1053844	VOLT REF, 2.048V, 0.5%. uPOW	2	D601
			D1601
E-73-217-48	DUAL RS-232 TX/RX 3.0V - 5.5V	1	IC901
E-35-790-18	ALPS-SMD PUSH BUTTON	2	SW701
			SW1701
F-1658504	SOLID TANTAL CAP	4	C401-402
			C1401-1402
F-1754123	TANTAL ELECTROLYTIC CAP	10	C302
			C305-306
			C603-604
			C1302
			C1305-1306
			C1602
			C1604
F-1754003	SOLID TANTAL CAP	4	C301
			C304
			C1301
			C1304
E-67-732-46	TANTAL ELECTROLYTIC CAP	4	C404
			C602
			C1404
			C1601
E-67-736-34	TANTAL ELECTROLYTIC CAP	2	C605
			C1605
F1471047	P-CHANNEL MOSFET 12V/2.6A 0.5W	2	M601
			M1601
F-9102639	N-CHANNEL MOSFET 20V	4	Q301-302
			Q1301-1302

End of report

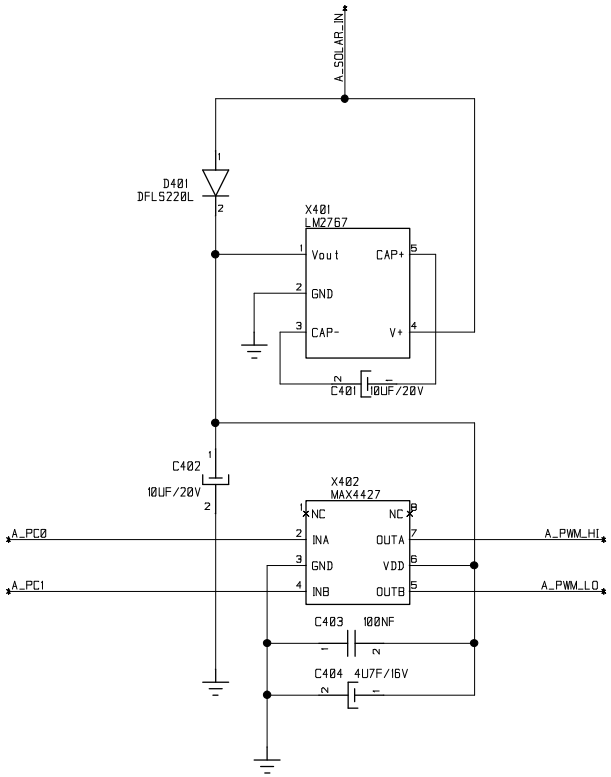
E.2 Schematics


100
I/O
B_TX
B_RX
A_TX
A_RX
B_TX
B_RX
A_RX
A_TX
B_PD1_SCL
B_PD0_SDA
B - MCU
B_MCU_AVCC
B_MCU_VCC
B_MCU_AVCC
B_MCU_VCC
B - MCU POWER
V_PWR
B_SOLAR_IN
V_PWR
A_SOLAR_IN
A - MCU POWER
A_MCU_AVCC
A_MCU_VCC
A_MCU_AVCC
A_MCU_VCC
A - MCU
A_PD1_SCL
A_PD0_SDA
B_PC1
B_PC0
A_PC0
A_PC1
B_PC1
B_PC0
A_PC0
A_PC1
B - Buck Converter
V_PWR
B_SOLAR_IN
V_PWR
A_SOLAR_IN
A - Buck Converter
TP103
TP/TP-P1N1.3
TP104
TP/TP-P1N1.3
TP101
TP/TP-P1N1.3
TP102
TP/TP-P1N1.3
B_SOLAR_IN
V_PWR
A_SOLAR_IN
B_PD0_SDA
B_PD1_SCL
MECHA
A_PD0_SDA
A_PD1_SCL
MARK1
TP/FID-MARK-ONYX
MARK2
TP/FID-MARK-ONYX
MARK3
TP/FID-MARK-ONYX
PROJECT:
CUBESTAR
DRAWING:
Top Level Block Diagram
DATE:13/09/2010
VER: 1.5
SHEET: 1 / 15
Martin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.


200
CN202
ERNI-SRC-214019
B_SOLAR_IN
CN203
PRI3XZ
A_PD0_SDA
A_PD1_SCL
B_PD0_SDA
B_PD1_SCL
V_PWR
V_PWR
GND
TP/TP-PIN1,3
A_RX_OUT
A_TX_OUT
B_RX_OUT
B_TX_OUT
CN201
ERNI-SRC-214019
A_SOLAR_IN
PROJECT:
CUBESTAR
DRAWING:
Connectors
DATE: 13/07/10
VER: 1.5
SHEET: 2 / 15
Martin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.

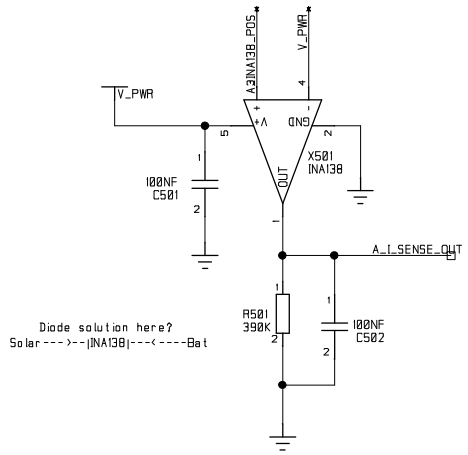

300
PROJECT:
CUBESTAR
A - Buck Converter
DRAWING:
DATE: 13/07/10
VER: 1.5
SHEET: 3 / 15
UNIVERSITY OF OSLO
Dept. of Physics
Marlin Oredsson
(c) copyright. All rights reserved.
V_PWR
C306 10UF/35V
C305 10UF/35V
C304 47UF/16V
A - CURRENT SENSE
V_PWR
A_INA138_POS
R301 0R025
100UH
L301
C307 3N3F
R304 0R0
D301 SL22
TP304 TP/TP-PIN1.3
C302 IRF7456
TP302 TP/TP-PIN1.3
R303 5R1
A_PWM_LO
TP301 TP/TP-PIN1.3
C301 IRF7456
R302 5R1
MOSFET DRIVER A
A_PWM_HI
A_PC0
A_PC1
A_SOLAR_IN
C303 100NF
C302 10UF/35V
C301 47UF/16V
A_SOLAR_IN

400

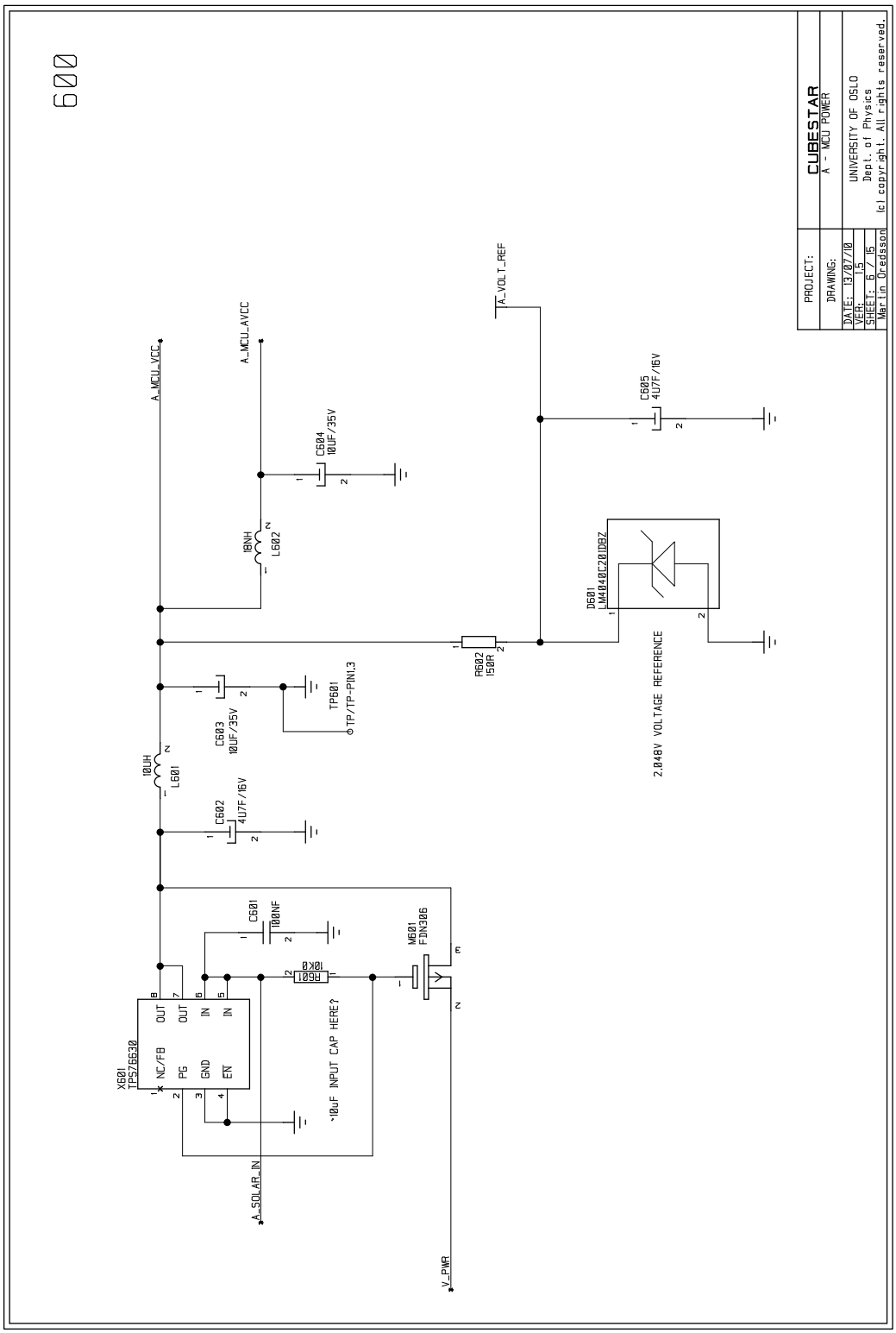


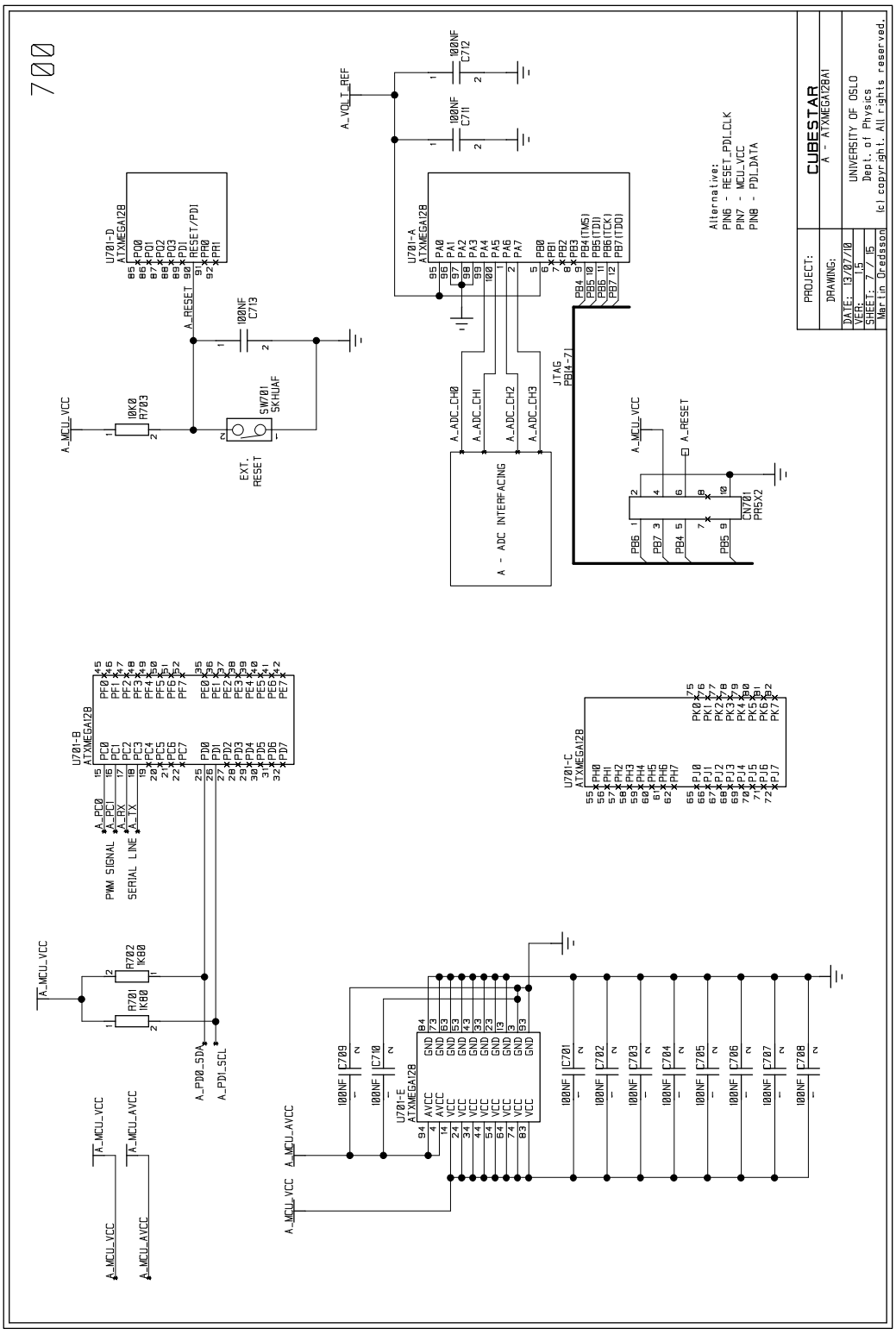
PROJECT:	CUBESTAR
DRAWING:	A - MOSFET DRIVER
DATE: 13/07/10	UNIVERSITY OF OSLO
VER: 1.5	Dept. of Physics
SHEET: 4 / 15	
Martin Oredsson	

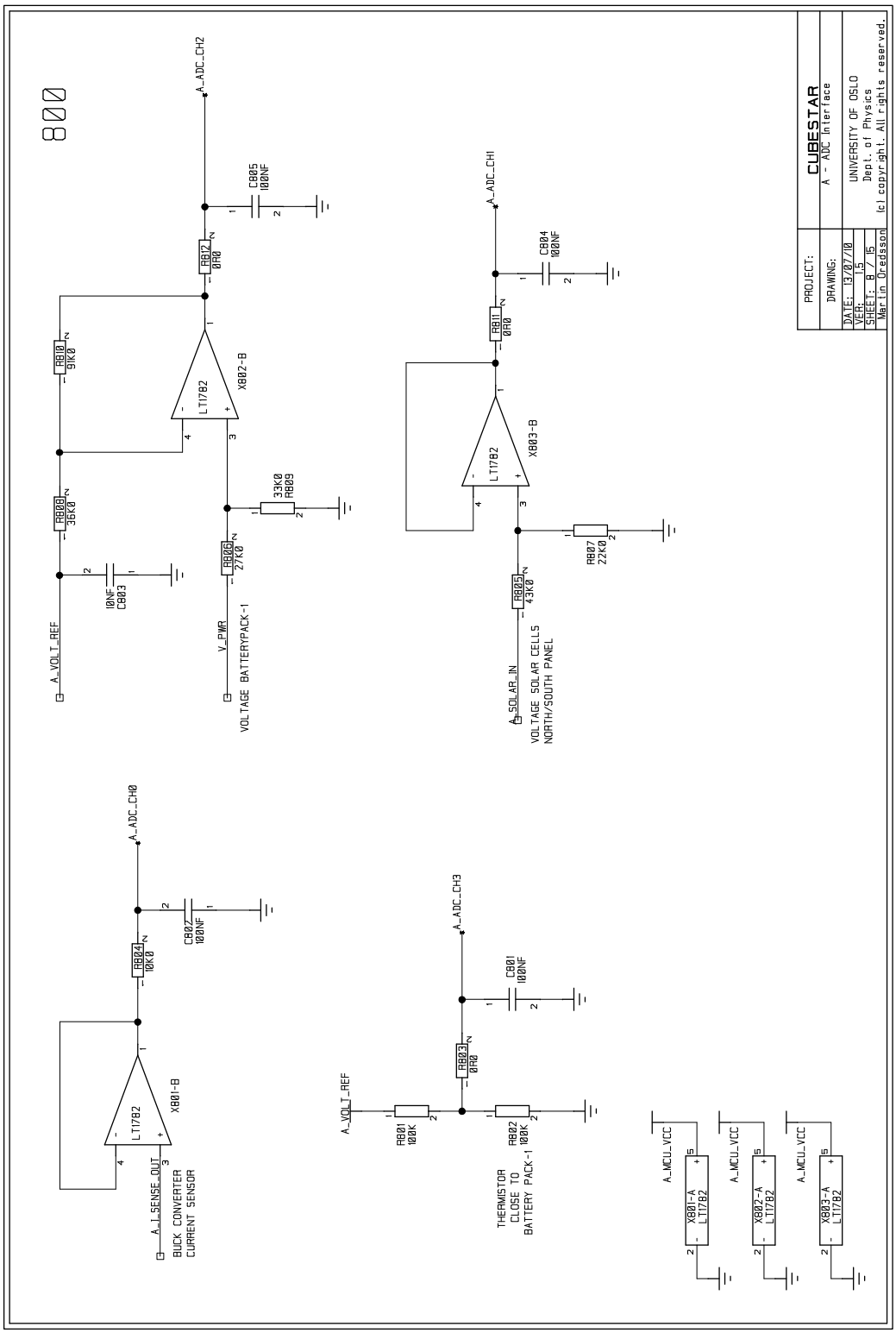
500



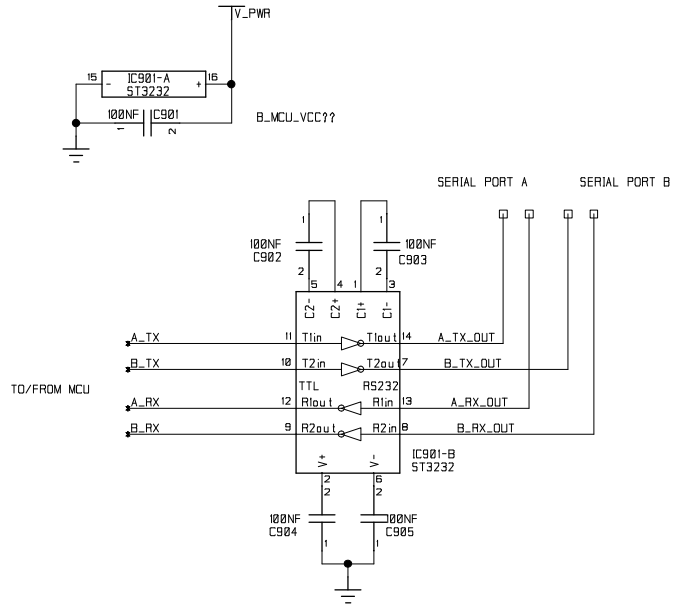
PROJECT:	CUBESTAR
DRAWING:	A - Current Sensor
DATE: 13/07/10	UNIVERSITY OF OSLO
VER: 1.5	Dept. of Physics
SHEET: 5 / 15	
Martin Oredsson	


600
PROJECT: CUBESTAR
DRAWING: A - MCU POWER
DATE: 13/07/10
VER: 1.5
SHEET: 6 / 15
Marlin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.
A_MCU_VCC
A_MCUL_AVCC
A_VOLT_REF
C604 10UF/35V
C605 4U7F/16V
10NH L602
D601 LM4040C20IDBZ
R602 150R
C603 10UF/35V
TP601 TP/TP-PIN1.3
10UH L601
C602 4U7F/16V
2.048V VOLTAGE REFERENCE
C601 100NF
R601 10K0
M601 FDN306
X601 TPS76630
NC/FB
PG
GND
EN
OUT
OUT
IN
IN
10UF INPUT CAP HERE?
A_SOLAR_IN
V_PWR

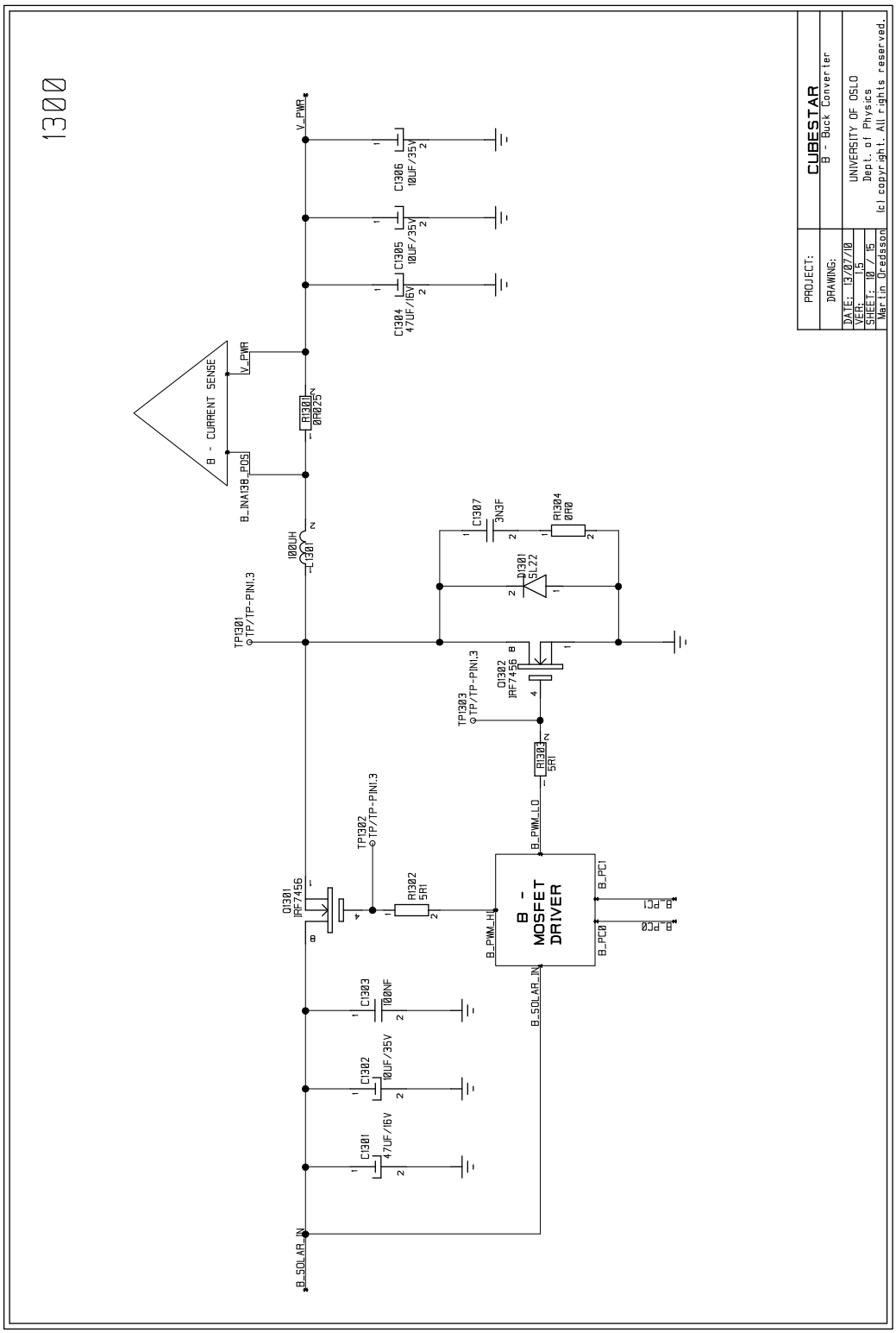

700
PROJECT:
CUBESTAR
A - ATXMEGA128A1
DRAWING;
DATE: 13/07/10
VER: 1.5
SHEET: 7 / 15
Marlin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.
A_MCU_VCC
A_MCU_AVCC
A_VCC_I_REF
A_RESET
A_PD0_SDA
A_PD1_SCL
PWM SIGNAL
SERIAL LINE
EXT. RESET
JTAG PB[4-7]
A - ADC INTERFACING
Alternative:
PIN6 - RESET_PDI_CLK
PIN7 - MCU_VCC
PIN8 - PDI_DATA
U701-A ATXMEGA128
U701-B ATXMEGA128
U701-C ATXMEGA128
U701-D ATXMEGA128
U701-E ATXMEGA128
CN701 PH5X2
SW701 SKHUAF


A_ADC_CH2
A_ADC_CH1
A_ADC_CH0
A_ADC_CH3
A_VOLT_REF
V_PWR
A_SOLAR_IN
A_I_SENSE_OUT
A_MCU_VCC
LT1782
X802-B
X803-B
X801-B
X801-A
X802-A
X803-A
VOLTAGE BATTERYPACK-1
VOLTAGE SOLAR CELLS NORTH/SOUTH PANEL
BUCK CONVERTER CURRENT SENSOR
THERMISTOR CLOSE TO BATTERY PACK-1
PROJECT: CUBESTAR
DRAWING: A - ADC Interface
DATE: 13/07/10
VER: 1.5
SHEET: 8 / 15
UNIVERSITY OF OSLO
Dept. of Physics
Marlin Ordesson
(c) copyright. All rights reserved.

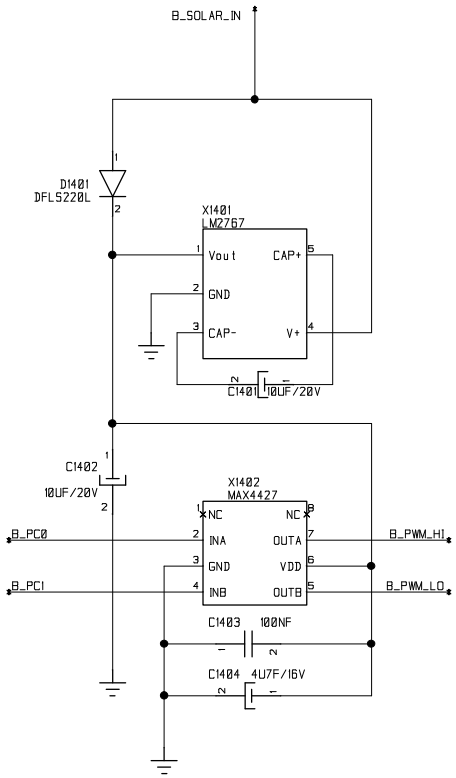
900



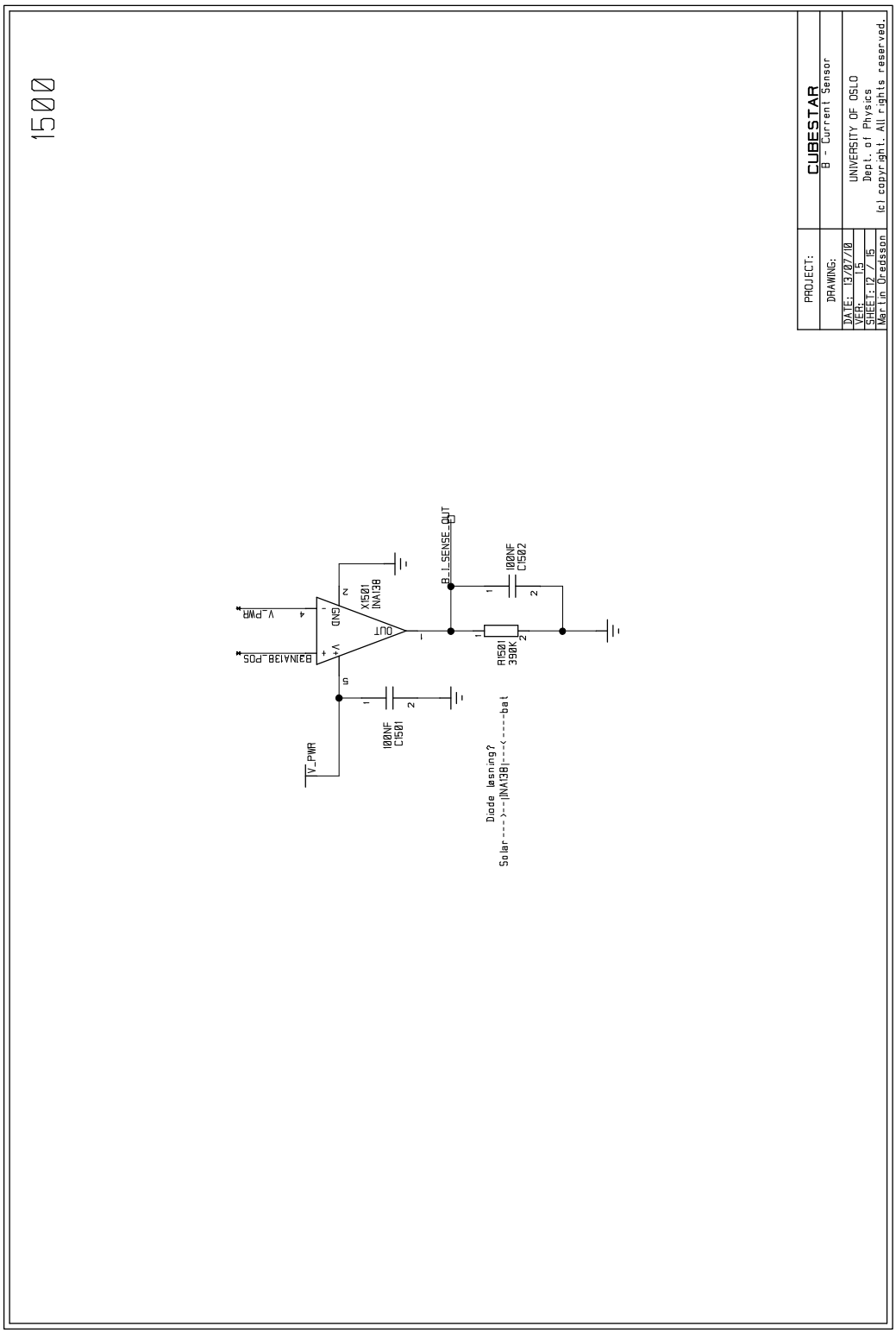
PROJECT:	CUBESTAR
DRAWING:	A+B Serial Driver (debugging)
DATE: 13/07/10	UNIVERSITY OF OSLO
VER: 1.5	Dept. of Physics
SHEET: 9 / 15	
Martin Oredsson	

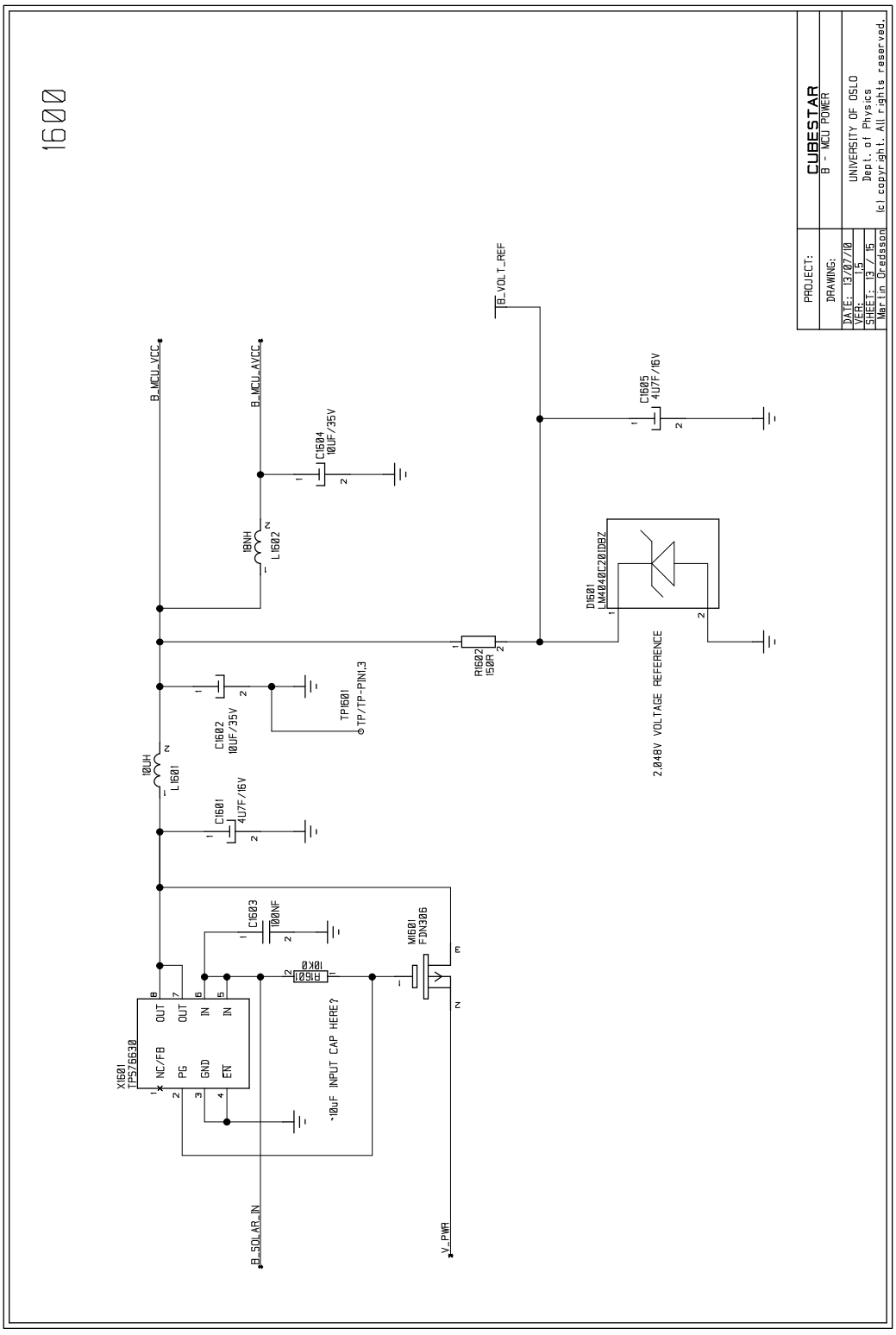

1300
V_PWR
C1306 10UF/35V
C1305 10UF/35V
C1304 47UF/16V
B - CURRENT SENSE
V_PWR
B_INA138_POS
R1301 0R025
100UH
L1301
C1307 3N3F
R1304 0R0
D1301 SL22
TP1301 TP/TP-PIN1.3
Q1302 IRF7456
TP1303 TP/TP-PIN1.3
R1303 5R1
B_PWM_LO
TP1302 TP/TP-PIN1.3
Q1301 IRF7456
R1302 5R1
B_PWM_HI
B - MOSFET DRIVER
B_PC1
B_PC0
C1303 100NF
C1302 10UF/35V
C1301 47UF/16V
B_SOLAR_IN
PROJECT: CUBESTAR
DRAWING: B - Buck Converter
DATE: 13/07/10
VER: 1.5
SHEET: 10 / 15
Marlin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.

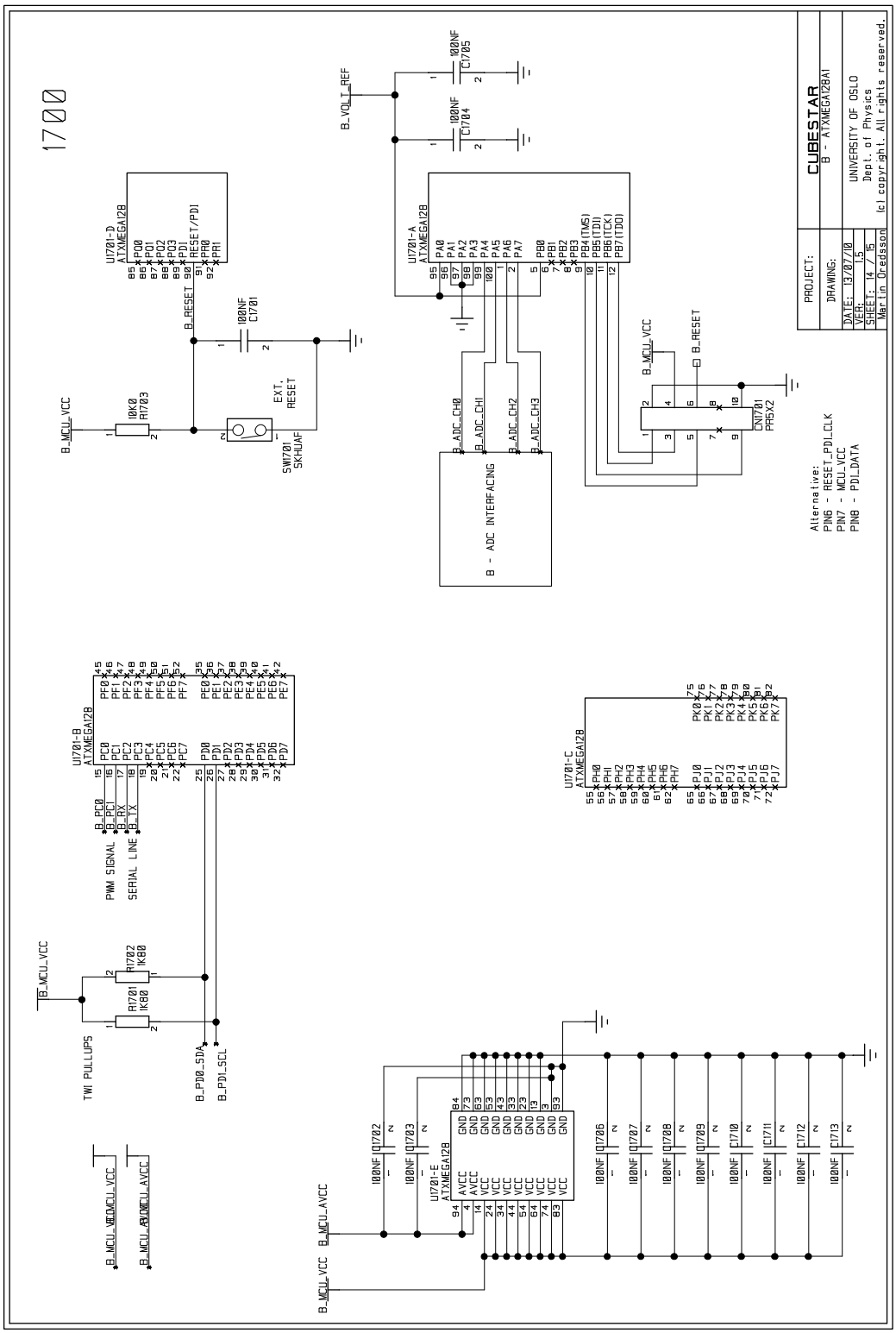
1400

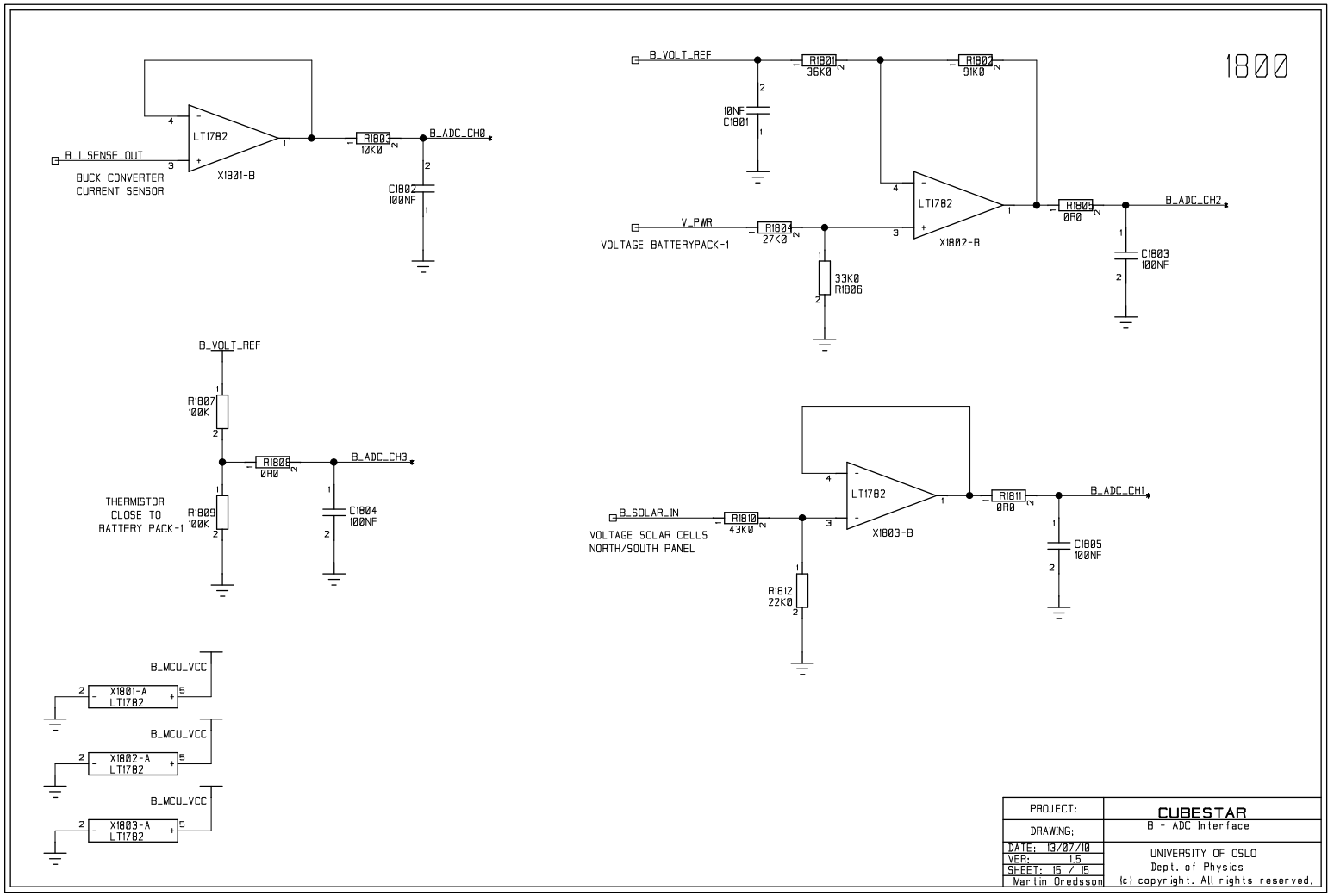


PROJECT:	CUBESTAR
DRAWING:	B - MOSFET DRIVER
DATE: 13/07/10	UNIVERSITY OF OSLO
VER: 1.5	Dept. of Physics
SHEET: 11 / 15	
Martin Oredsson	


1500
PROJECT:
CUBESTAR
DRAWING:
B - Current Sensor
DATE: 13/07/10
VER: 1.5
SHEET: 12 / 15
Martin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.
B31NA138_POS
V_PWR
GND
OUT
V+
X1501
INA138
B_I_SENSE_OUT
100NF
C1502
R1501
330K
100NF
C1501
V_PWR
Diode losning?
Solar---->--|INA138|---<---bat


1600
PROJECT: CUBESTAR
DRAWING: B - MCU POWER
DATE: 13/07/10
VER: 1.5
SHEET: 13 / 15
Marlin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.
B_MCU_VCC
B_MCU_AVCC
B_VOLT_REF
B_SOLAR_IN
V_PWR
C1604 10UF/35V
C1605 4U7F/16V
C1602 10UF/35V
C1601 4U7F/16V
C1603 100NF
L1602 18NH
L1601 10UH
R1602 150R
R1601 10K0
D1601 LM4040C2010BIZ
M1601 FDN306
X1601 TPS76630
TP1601 TP/TP-PIN1.3
2.048V VOLTAGE REFERENCE
10UF INPUT CAP HERE?
NC/FB
PG
GND
EN
OUT
OUT
IN
IN


1700
B_MCU_VCC
B_VOLT_REF
B_RESET
B_ADC_CH0
B_ADC_CH1
B_ADC_CH2
B_ADC_CH3
B_PD0_SDA
B_PD1_SCL
B_MCU_AVCC
TWI PULLUPS
PWM SIGNAL
SERIAL LINE
EXT. RESET
B - ADC INTERFACING
Alternative:
PIN6 - RESET_PDI_CLK
PIN7 - MCU_VCC
PIN8 - PDI_DATA
CUBESTAR
B - ATXMEGA128A1
UNIVERSITY OF OSLO
Dept. of Physics
Marlin Ordegsson
(c) copyright. All rights reserved.


1800
B_I_SENSE_OUT
BUCK CONVERTER
CURRENT SENSOR
LT1782
X1801-B
R1803
10K0
B_ADC_CH0
C1802
100NF
B_VOLT_REF
R1801
36K0
R1802
91K0
10NF
C1801
V_PWR
VOLTAGE BATTERYPACK-1
R1804
27K0
33K0
R1806
X1802-B
R1805
0R0
B_ADC_CH2
C1803
100NF
B_VOLT_REF
R1807
100K
R1808
0R0
B_ADC_CH3
THERMISTOR
CLOSE TO
BATTERY PACK-1
R1809
100K
C1804
100NF
B_SOLAR_IN
VOLTAGE SOLAR CELLS
NORTH/SOUTH PANEL
R1810
43K0
X1803-B
R1811
0R0
B_ADC_CH1
C1805
100NF
R1812
22K0
B_MCU_VCC
X1801-A
LT1782
X1802-A
LT1782
X1803-A
LT1782
PROJECT:
CUBESTAR
DRAWING:
B - ADC Interface
DATE: 13/07/10
VER: 1.5
SHEET: 15 / 15
Martin Oredsson
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.

E.3 Gerber Files

The result of post-processing is a collection of files in the Gerber file format. The Gerber files listed below completely defines the PCB, and is used by the board house's manufacturing machines. Each layer has its own Gerber file, which is briefly described below.

Layer 1: TopSilk Layer

This layer contains all visible text on the top side of the finished PCB, including text for component names, connectors and testpoints.

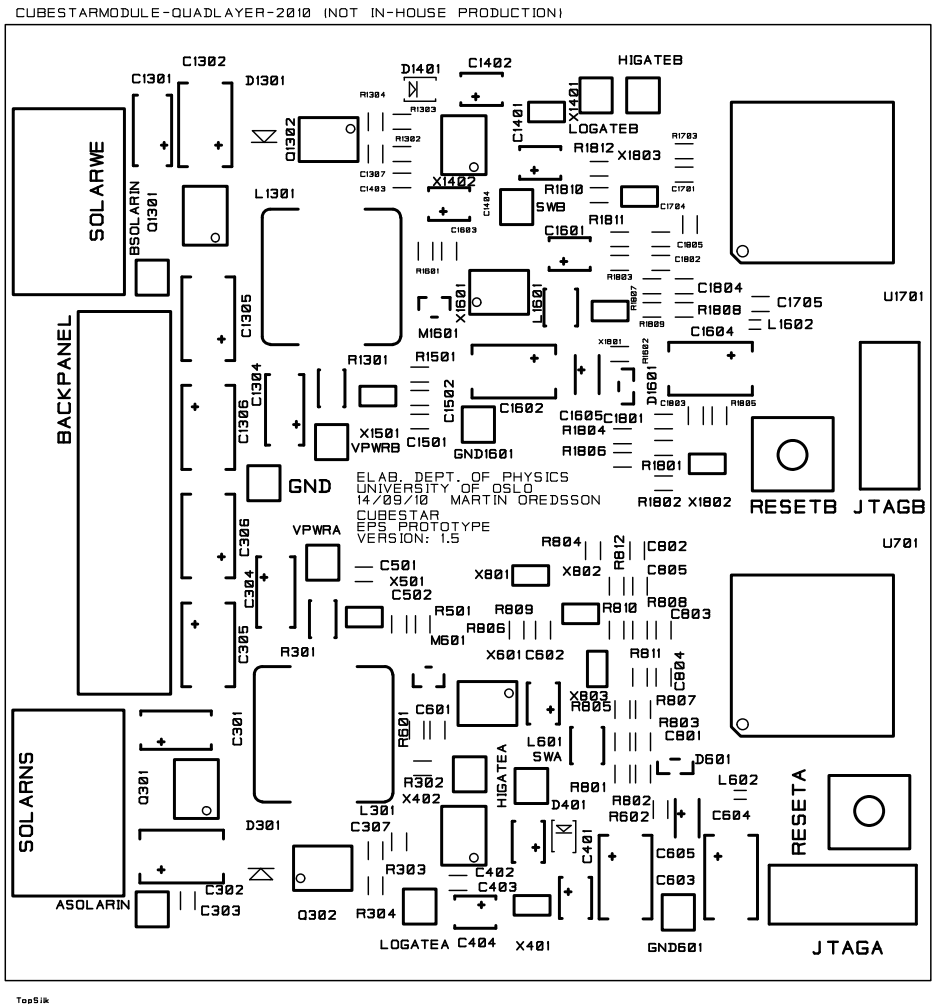


Figure E.1: TopSilk layer includes component names, outlines and other visible text.

Layer 2: TopStop Layer

This layer defines the solder resist mask, giving the finished PCB its characteristic green color. The solder mask greatly simplifies the soldering process because it treats solder paste during reflow much in the same way that oil treats water. Thus by having a solder resist between pads, the risk for solder "bridges" (a short circuit) between component pins is greatly reduced. The pads on this layer are oversized by 8 mils to provide a buffer zone around the pad.

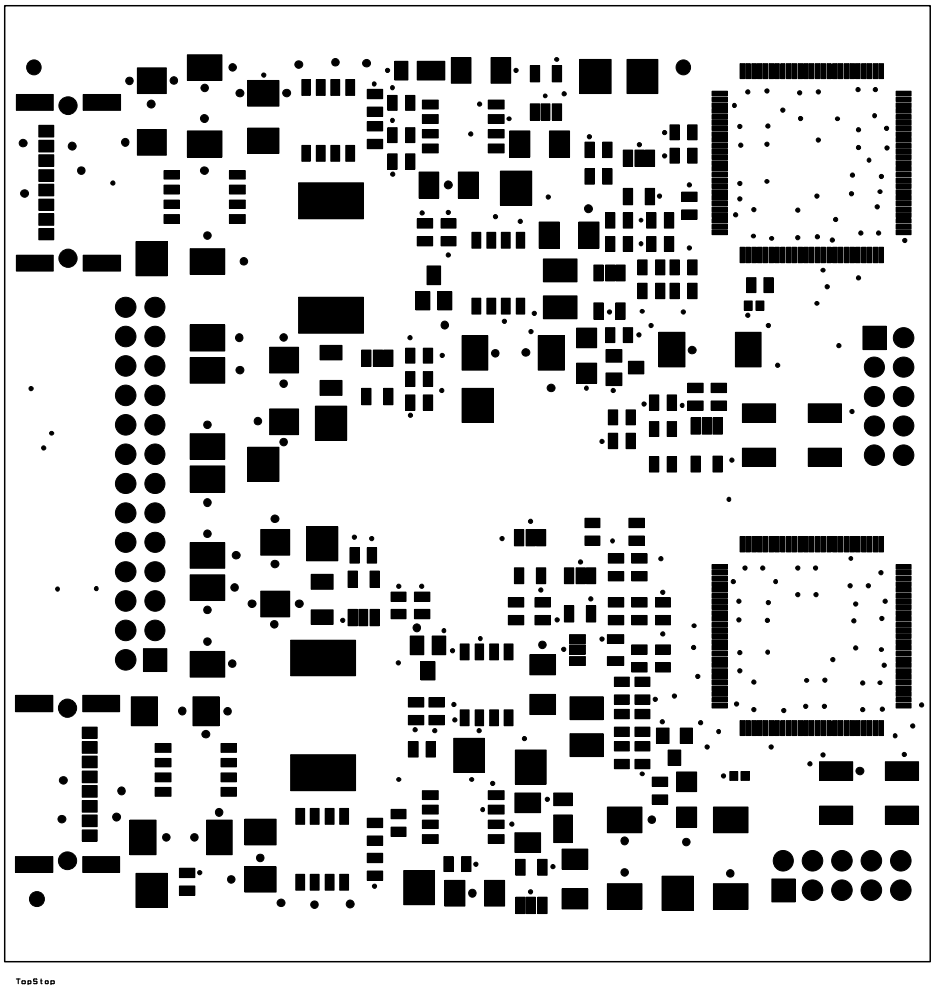


Figure E.2: TopStop layer defines the solder resist masks.

Layer 3: TopPaste Layer

This layer defines the mask for the soldering stencil, if such a stencil is used. The stencil is a aluminium sheet with openings where the pads are. Once in place the solder may be applied and once the stencil is removed, only the pads are left covered with solder. The pads on this layer should be undersized by 10% to avoid smearing.

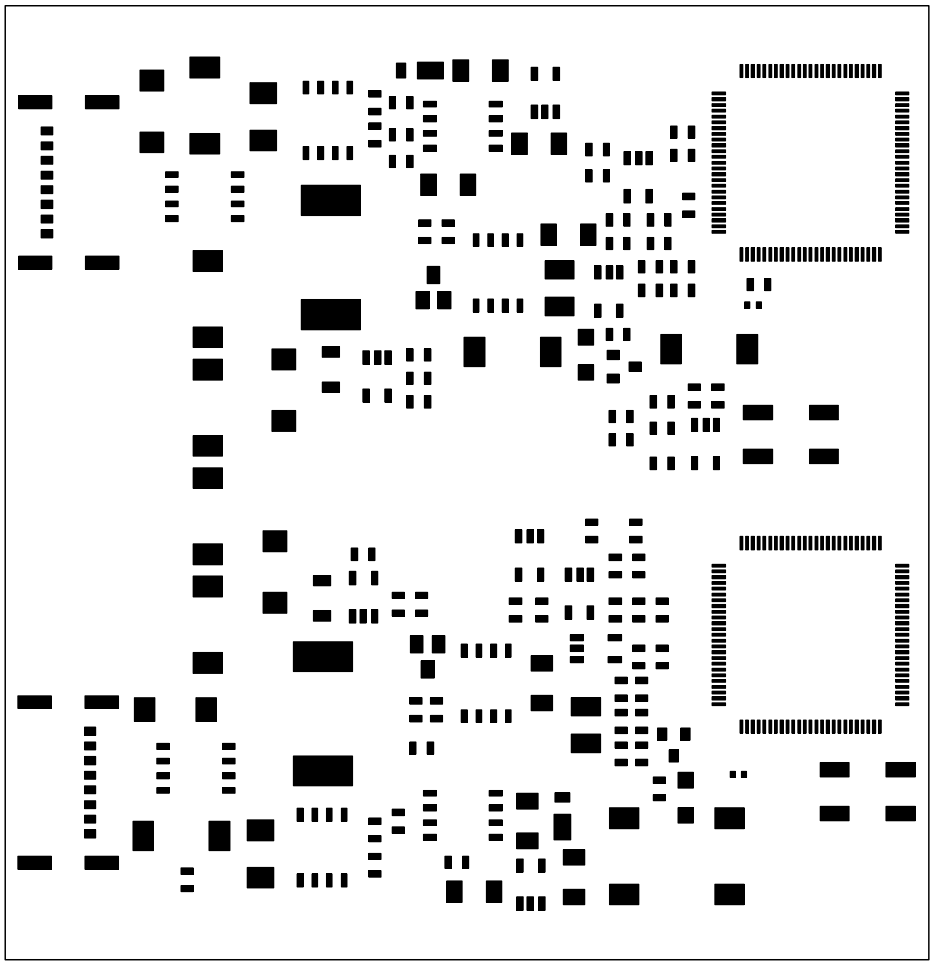


Figure E.3: TopPaste layer defines the openings in the soldering paste stencil.

Layer 4: TopElec Layer

This is the top copper layer, and it carries most of the signal traces on the board. Among these are the high-current input from the solar cells, the switch node, and the mosfet drive signals which are all routed uninterrupted on this layer.

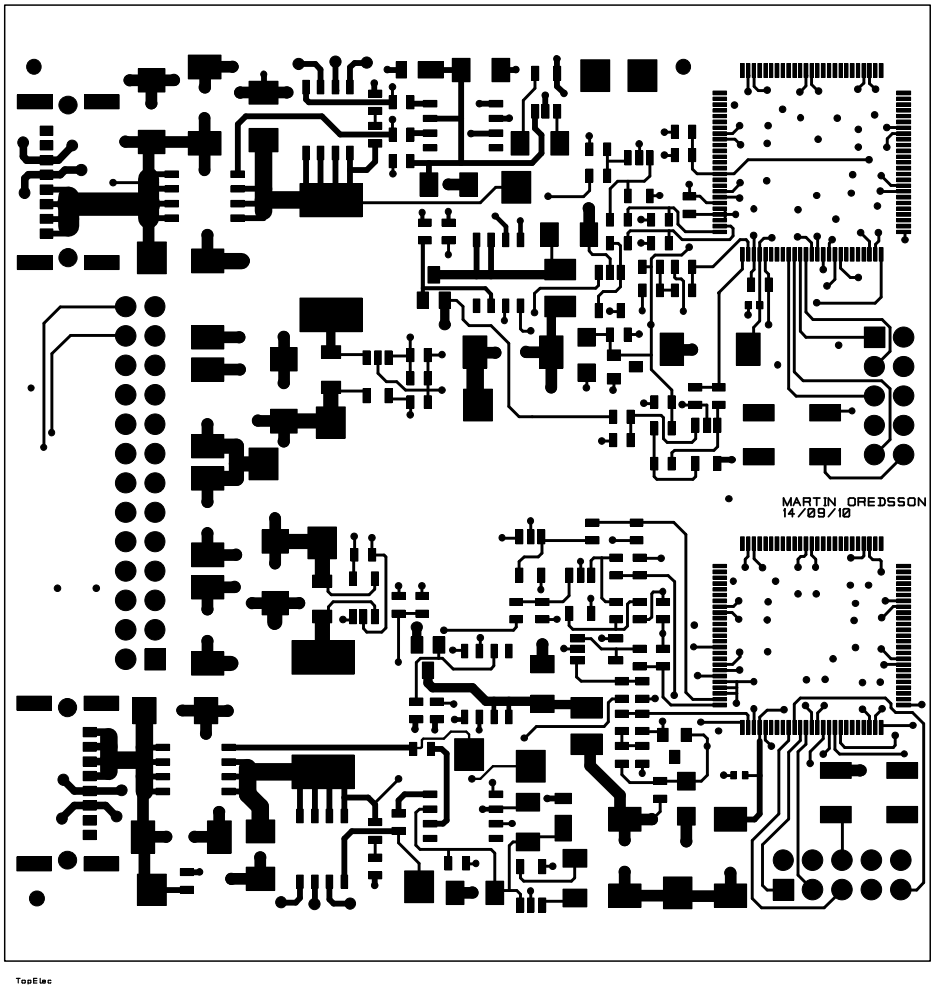


Figure E.4: TopElec layer is the first copper layer and defines the copper tracks and pads on the layer.

Layer 5: Ground Layer

The second, and arguably the most important, copper layer—mistakes made here are punished hard by mother nature. The ground plane is separated into areas for analog and digital to avoid the issues discussed in Section 5.3

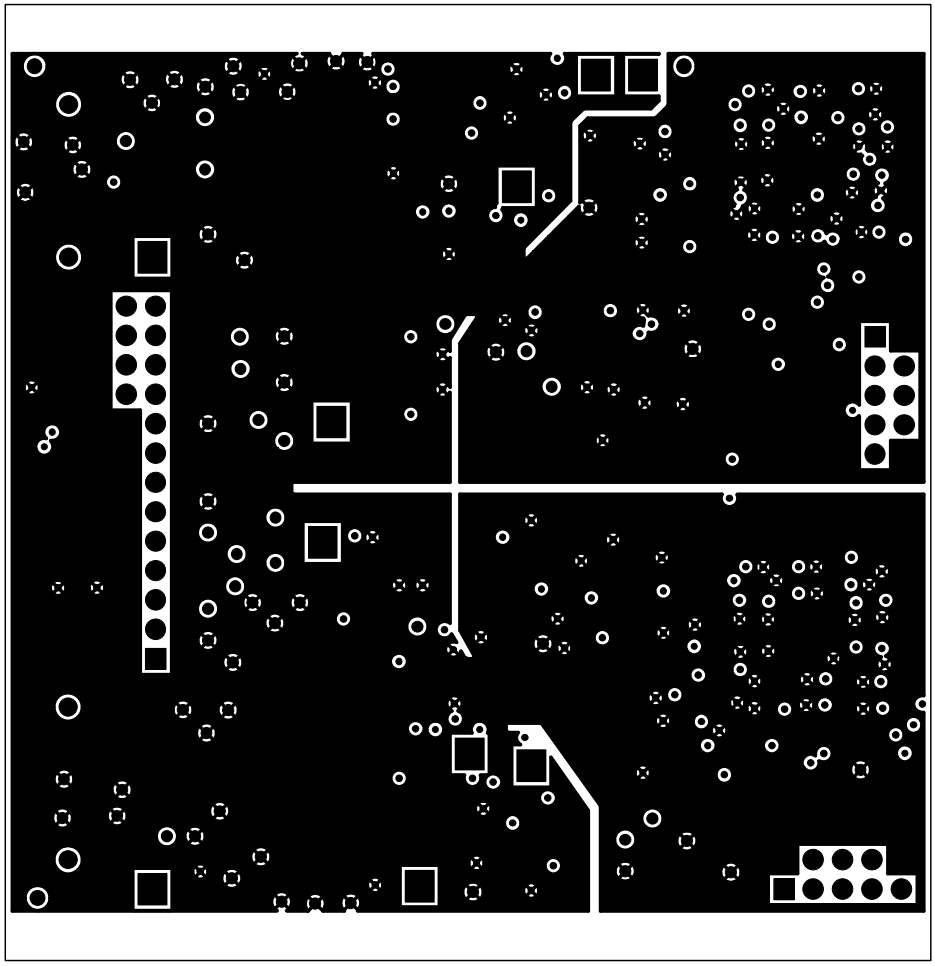


Figure E.5: Ground layer is the second copper layer. It is segregated into analog, digital and battery parts.

Layer 6: Power Layer

The third copper layer is the power plane, which is split into solar array power, digital power and battery power areas. The latter is the battery "bus" from which the remaining satellite subsystems are powered.

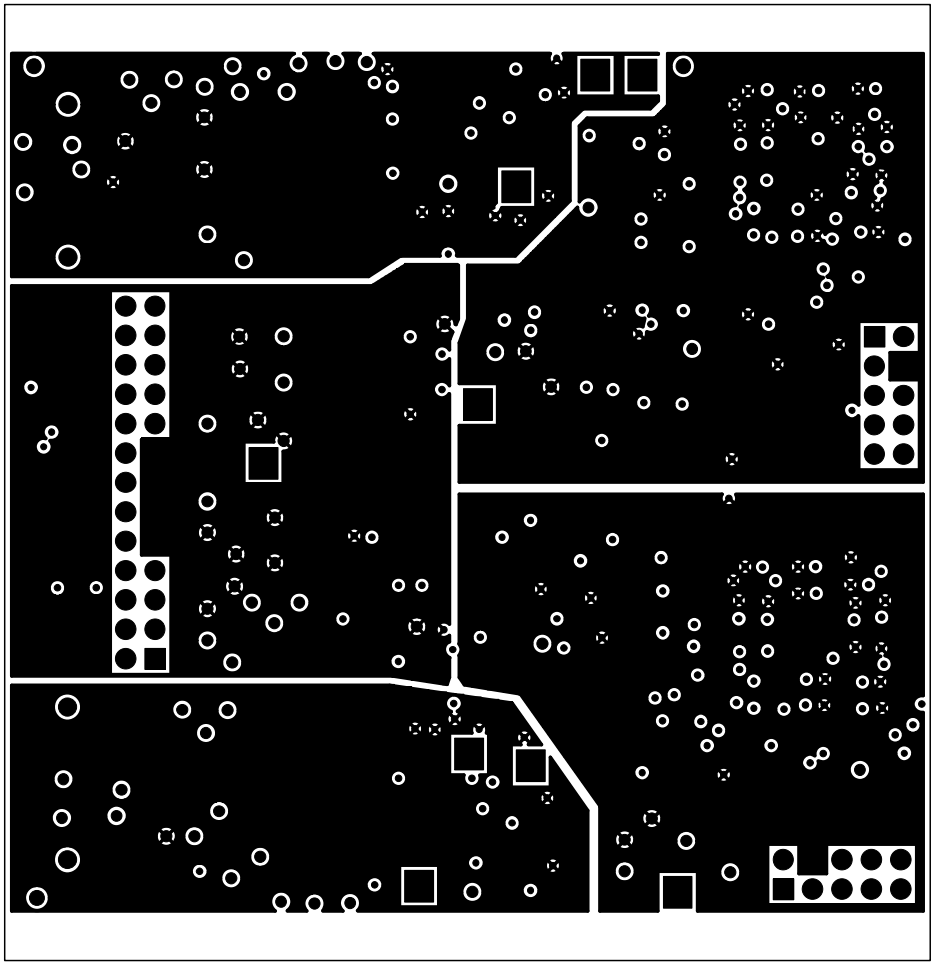


Figure E.6: Power layer is the third copper layer. It is split into power "islands" for two opposite solar array side, digital, and battery.

Layer 7: BotElec Layer

The fourth and last copper layer is the bottom signal layer. It is mainly used for decoupling and routing digital traces. It also holds the RS-232 driver used for debugging.

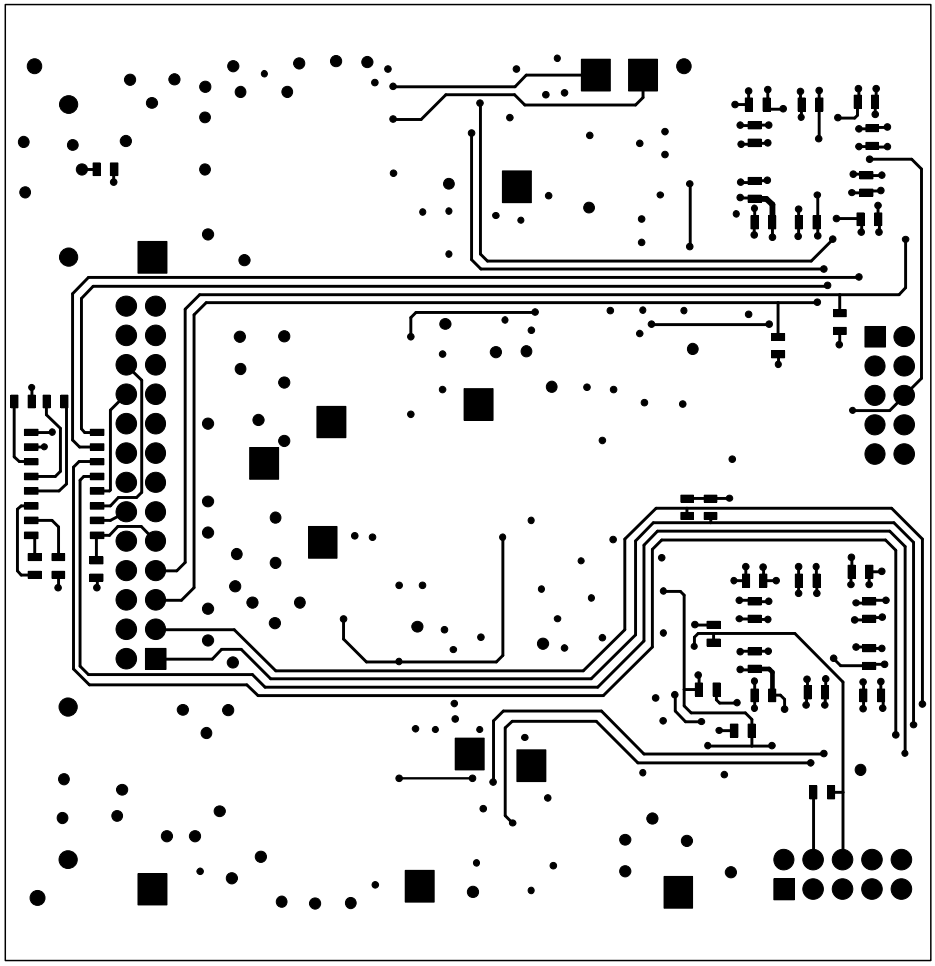


Figure E.7: BotElec layer is the fourth and last copper layer and defines the copper tracks and pads on the layer.

Layer 8: BotStop Layer

This is the solder resist mask for the bottom layer, see TopStop.

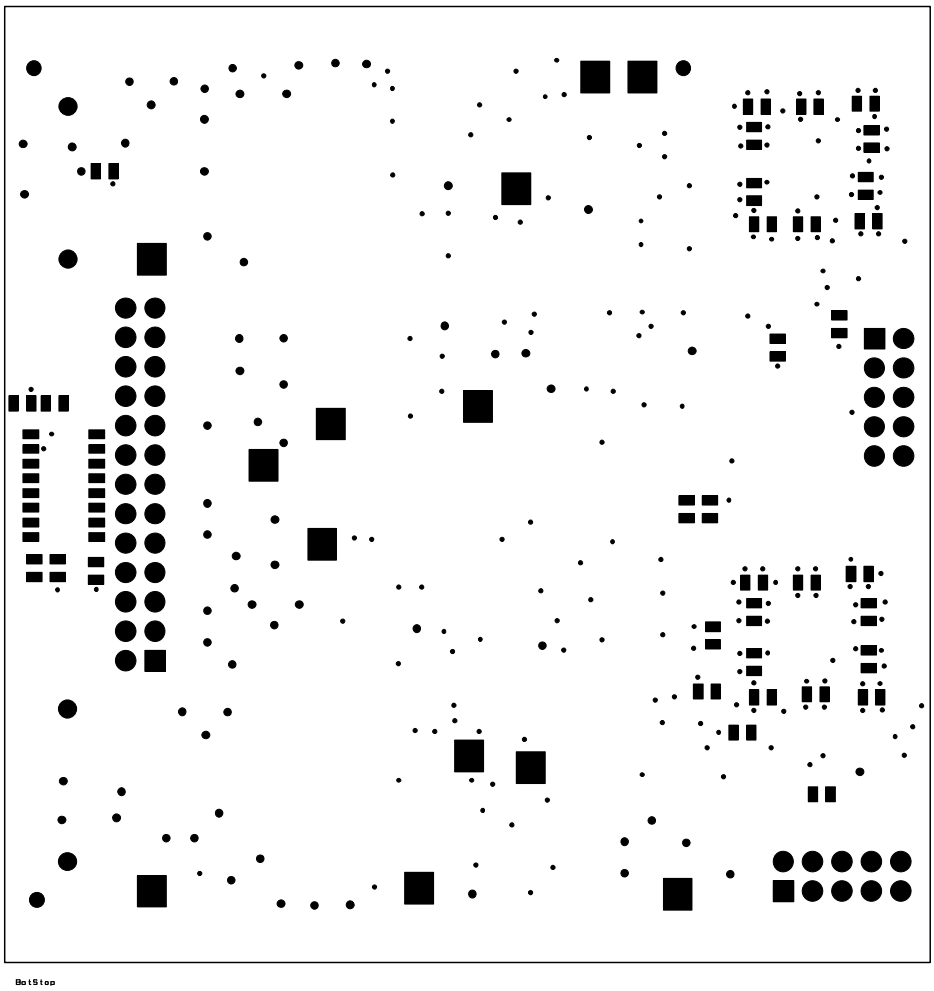


Figure E.8: BotStop layer defines the solder resist mask for the bottom layer.

Layer 9: Drill File

The NC (numeric control) drill file was created with Exellon's control codes, and defines the drill holes on the board. The dimensions of vias, testpoints, and any other through holes are defined here.

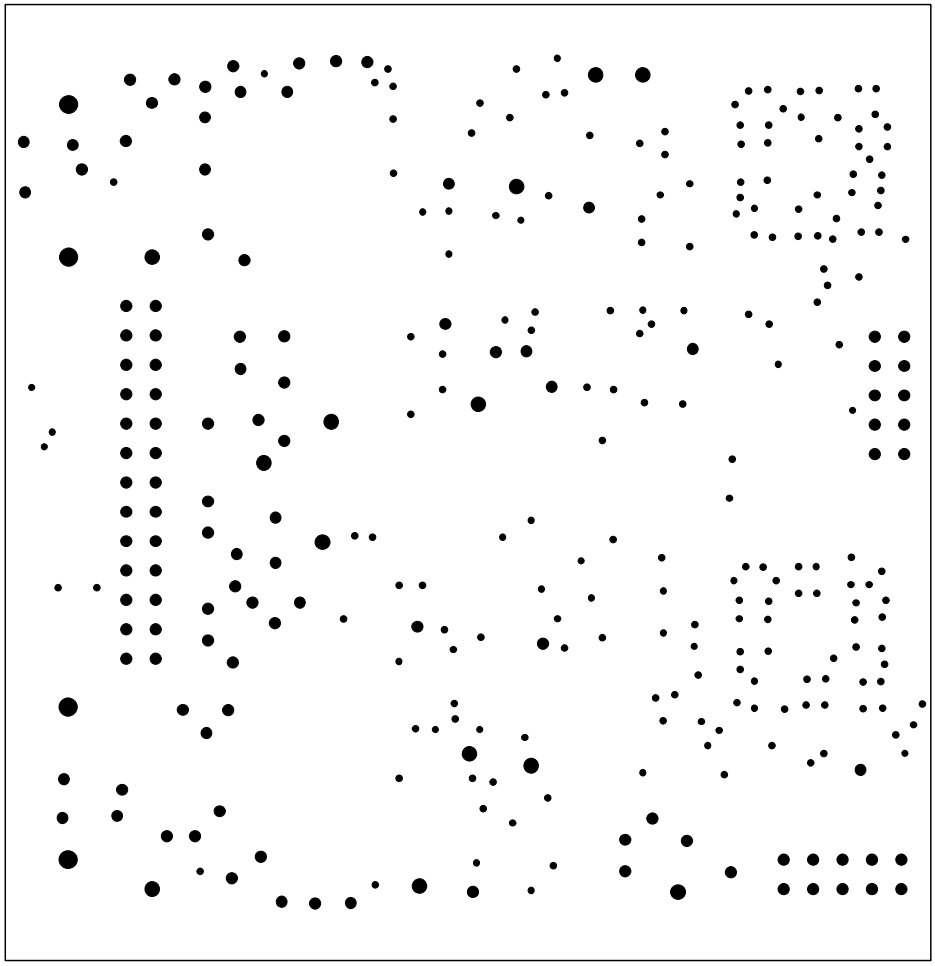


Figure E.9: Drill layer is not really a layer, but in any case, it defines the drill holes for all the plated vias and testpoints used.

Appendix F

C Code for ATXmega128A1

Listing F.1: main.c

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <stdio.h>
4  #include <stdbool.h>
5  #include <stdint.h>
6  #include "avr_compiler.h"
7  #include "adc.h"
8  #include "battery.h"
9  #include "clock.h"
10 #include "fsm.h"
11 #include "lifespecs.h"
12 #include "mppt.h"
13 #include "pid.h"
14 #include "pwm.h"
15 #include "serial.h"
16 #include "twi_slave.h"
17 #include "main.h"
18
19
20 //*****************************************************
21 // Variable definitions.
22 //*****************************************************
23 #define NUM_BYTES        8
24 gFlags_t gFlags = {0, 0};
25 SolarPanels_t SolarPanelNS;
26 TWI_Slave_t twiSlave;
27 uint8_t telemetry[NUMBER_OF_TELEMETRY_POINTS * 2];
28
29 //*****************************************************
```

```
30 // Setup printf() for debugging purposes.
31 //*************************************************
32 FILE mystdout = FDEV_SETUP_STREAM(uart_putchar,
33                 NULL,
34                 _FDEV_SETUP_WRITE);
35
36 int main(void)
37 {
38     // Connect the standard output
39     // to the address of mystdout
40     stdout = &mystdout;
41
42     //*************************************************
43     // Setup the GPIO ports for their respective duties
       .
44     //*************************************************
45     // PWM output on PORTC
46     PORTC.DIRSET = 0xFF;
47     // Switches input on PORTF
48     PORTF.DIRSET = 0x00;
49
50     //*************************************************
51     // Initialize System Peripherals
52     //*************************************************
53     // Sets the system- and peripheral clocks.
54     CLK_init();
55
56     // Enable high-level interrupts
57     PMIC_SetVectorLocationToApplication();
58     PMIC.CTRL |= PMIC_HILVLEN_bm | PMIC_LOLVLEN_bm;
59
60     // Enable interrupts globally
61     sei();
62
63     // Enables and starts PWM on PORTC[0:1].
64     PWM_Start();
65     // Sets up the serial comm. link used for debugging
       .
66     USART_init();
67     // Initialize the adc converter channels.
68     ADC_init();
69     // Initialize PID regulator
70     pid_Init(   K_P * SCALING_FACTOR,
71                 K_I * SCALING_FACTOR,
72                 K_D * SCALING_FACTOR,
73                 &pidData);
74
75     // Initialize TWI slave
76     TWI_SlaveInitializeDriver(  &twiSlave,
```

```
77                                  &TWID,
78                                  TWID_SlaveProcessData);
79      TWI_SlaveInitializeModule(  &twiSlave,
80                                  SLAVE_ADDRESS,
81                                  TWI_SLAVE_INTLVL_HI_gc)
                                        ;
82
83      typedef struct {
84          unsigned char NextState;
85          unsigned char Error;
86      } NormalParameters_t;
87
88      NormalParameters_t NormalParameters;
89      unsigned char CurrentState = ST_MPPT;
90      uint16_t SAvoltage, SAcurrent, BatVoltage;
91      char i = 0;
92
93      unsigned char NormalMode(unsigned char inp)
94      {
95          unsigned char NextState;
96          SAvoltage = SolarArrayNSVoltage();
97          SAcurrent = SolarArrayNSCurrent();
98          BatVoltage = BatteryVoltage();
99
100         switch (CurrentState) {
101
102             case ST_MPPT:
103                 MPPT(SAcurrent, SAvoltage);
104                 NextState = ST_MPPT;
105                 if(BattData.Voltage >= BAT_VOLTAGE_MAX)
                        {
106                     NextState = ST_CVOLTAGE;
107                     NormalParameters.NextState =
                            NextState;
108
109                 } else if (BattData.Voltage <
                        BAT_VOLTAGE_MAX) {
110                     NextState = ST_MPPT;
111                     NormalParameters.NextState =
                            NextState;
112                 }
113             break;
114
115             case ST_CVOLTAGE:
116                 ST_CVoltage();
117                 if(     BattData.Voltage <=
118                         (BAT_VOLTAGE_MAX -
                            BAT_VOLTAGE_HYST)) {
119                     NextState = ST_MPPT;
```

```
120                     NormalParameters.NextState =
                           NextState;
121                 } else {
122                     NextState = ST_CVOLTAGE;
123                     NormalParameters.NextState =
                           NextState;
124                 }
125             break;
126
127             case ST_DISCONNECT:
128         ST_Disconnect();
129         if(     BattData.Voltage <=
130                         (BAT_VOLTAGE_MAX -
                            BAT_VOLTAGE_HYST)) {
131                     PWM_Start();
132           NextState = ST_MPPT;
133                     NormalParameters.NextState =
                           NextState;
134                 } else {
135                     NextState = ST_DISCONNECT;
136                     NormalParameters.NextState =
                           NextState;
137                 }
138       break;
139
140             case ST_POWERSAVE:
141                 break;
142
143             case ST_SOFTSTART:
144                 break;
145
146             default:
147             break;
148         }
149         return(NextState);
150     }
151
152     while(1){
153         CurrentState = NormalMode(1);
154
155         // Usart Debugging info
156         i++;
157         if (i==50)
158             {
159                 printf("%d %u %u %u %u\n",
160                         NormalParameters.NextState,
161                         SolarPanelNS.Current,
162                         SAvoltage,
163                         BattData.Voltage,
```

```
164                         DutyCycle);
165                 i = 0;
166             }
167     }
168
169   SetDutyCycle(185);
170     return 1;
171 }
172
173 void UpdateTWI(void) {
174   for(uint8_t i = 0; i<8; i++) {
175     twiSlave.sendData[i] = GetTelemetry(i);
176   }
177 }
178
179 void TWID_SlaveProcessData(void)
180 {
181   uint8_t bufIndex = twiSlave.bytesReceived;
182   twiSlave.sendData[bufIndex] =
183             (~twiSlave.receivedData[bufIndex]);
184 }
185
186 /* TWI slave Address or Stop Interrupt vector.
187 *
188 *  TWID Slave Interrupt vector.
189 */
190 ISR(TWID_TWIS_vect)
191 {
192   UpdateTWI();
193     TWI_SlaveInterruptHandler(&twiSlave);
194 }
195
196 /* Timer/Counter Overflow Channel A Interrupt vector.
197 *
198 *  Updates the dutycycle buffer once per period.
199 */
200 ISR(TCC0_OVF_vect)
201 {
202   TCC0.CCABUF = DutyCycle;
203 }
204
205 /* Timer/Counter Compare or Capture Channel A
206 *  Interrupt vector offset.
207 *  Used by the PID regulator to set the update flag.
208 */
209 ISR(TCC1_CCA_vect)
210 {
211     static uint16_t i = 0;
212     if(i < TIME_INTERVAL)
```

```
213         i++;
214     else {
215         gFlags.pidTimer = TRUE;
216         i = 0;
217     }
218 }
219
220 /*  Move interrupt vector table to application area.
221  *
222  *  This function moves the interrupt vector table
223  *  to application area. The function writes the
224  *  correct signature to the Configuration Change
225  *  Protection register before writing the CTRL
226  *  register. Interrupts are automatically ignored
227  *  during the change enable period.
228  */
229 void PMIC_SetVectorLocationToApplication(void)
230 {
231     uint8_t temp = PMIC.CTRL & ~PMIC_IVSEL_bm;
232     CCP = CCP_IOREG_gc;
233     PMIC.CTRL = temp;
234 }
235
236 uint16_t SolarArrayNSVoltage(void)
237 {
238     uint16_t SAVoltage = ReadADC(&ADCA.CH1, offset_ch1,
            5);
239     SolarPanelNS.Voltage = SAVoltage;
240     return(SAVoltage);
241 }
242
243 uint16_t SolarArrayNSCurrent(void)
244 {
245     uint16_t SACurrent = ReadADC(&ADCA.CH0, offset_ch0,
            5);
246     if(SACurrent < 0)
247         SACurrent = 0;
248     if(SACurrent > 60000)
249         SACurrent = 0;
250     SolarPanelNS.Current = SACurrent;
251     return(SACurrent);
252 }
```

Listing F.2: main.h

```
1 #ifndef MAIN_H
2 #define MAIN_H
3
```

```
4  // Minimum solar panel current below which the panels
       are disconnected.
5  #define SOLARPANEL_CURRENT_MIN  0
6  // Minimum solar panel voltage
7  #define SOLARPANEL_VOLTAGE_MIN  0
8  #define TELEMETRY_BYTES   4
9  #define TWIS_STATUS_READY   0
10
11
12 //*************************************************
13 // State machine states
14 //*************************************************
15 // Initialization state
16 #define ST_INIT       (10)
17 // Start-up state
18 #define ST_START      (20)
19 // Prequalification state for over-discharged batteries
20 #define ST_PREQUAL      (30)
21 // Prequalification control state
22 #define ST_PREQUAL_CTRL   (40)
23 // Maximum power point tracking state
24 #define ST_MPPT       (50)
25 // Maximum power point tracking control state
26 #define ST_MPPT_CTRL    (60)
27 // Constant voltage state with PID regulation
28 #define ST_CVOLTAGE     (70)
29 // Constant voltage control state
30 #define ST_CVOLTAGE_CTRL  (80)
31 // Sleep state
32 #define ST_SLEEP      (90)
33 // Disconnect state
34 #define ST_DISCONNECT   (100)
35 // Discharge state (same as disconnect?)
36 #define ST_DISCHARGE    (110)
37 // Error state that deals with any error that may arise
38 #define ST_ERROR      (120)
39
40
41 //*************************************************
42 // Solar Panels struct declarations
43 //*************************************************
44 /* Holds status and various data for the solar panels.
45  *
46  * These data are updated by sensors (TODO)
47  */
48 typedef struct
49 {
50   uint16_t Voltage;
51   uint16_t Current;
```

```
52   uint16_t Temperature;
53   uint16_t CutoffVoltage;
54 } SolarPanels_t;
55
56
57 //*****************************************************
58 // Flags for status information
59 //*****************************************************
60 /* Holds flags.
61  *
62  * These data are updated by
63  */
64 typedef struct {
65     uint8_t pidTimer:1;
66   uint8_t pwmTimer:1;
67 } gFlags_t;
68
69 //*****************************************************
70 // Global variables
71 //*****************************************************
72 extern unsigned char CurrentState;
73 extern int8_t   offset_ch0, offset_ch1,
74                 offset_ch2, offset_ch3;
75 extern gFlags_t gFlags;
76 extern SolarPanels_t SolarPanelNS;
77
78 //*****************************************************
79 // Function prototypes
80 //*****************************************************
81 int main (void);
82 void PMIC_SetVectorLocationToApplication(void);
83 uint16_t SolarArrayNSCurrent(void);
84 uint16_t SolarArrayNSVoltage(void);
85 void UpdateTWI(void);
86
87 #endif //MAIN_H
```

Listing F.3: adc.c

```
1  #include <avr/io.h>
2  #include <stdio.h>
3  #include <inttypes.h>
4  #include <stddef.h>
5  #include "avr_compiler.h"
6  #include "adc.h"
7
8  /* Reads the selected ADC channel.
9   *
10  * Takes a pointer to an ADC channel, offset, and
```

```
11  * the number of averaged samples as argument,
12  * and starts a conversion on the given channel.
13  * An average over [samples] samples is then
14  * calculated before the conversion
15  * is terminated and the answer is returned.
16  */
17 uint16_t ReadADC(   ADC_CH_t *adc_ch,
18                     uint8_t offset,
19                     uint8_t samples) {
20
21   uint32_t dummy = 0;
22   uint16_t result = 0;
23
24   for(uint8_t i = 0; i<samples; i++) {
25     ADC_Ch_Conversion_Start(adc_ch);
26     do {
27     } while (!ADC_Ch_Conversion_Complete(adc_ch));
28           if(i != 0) {
29        dummy += (uint32_t)ADC_ResultCh_GetWord_Signed(
              adc_ch, offset);
30     }
31   }
32   result = (uint16_t)(dummy/(samples-1));
33   return result;
34 }
35
36 /* Helper function for the ReadADC function.
37  *
38  * Clears Interrupt flag by setting IF high, stores
39  * contents of the channels RES into answer before
40  * returning the answer.
41  */
42 int16_t ADC_ResultCh_GetWord_Signed(ADC_CH_t * adc_ch,
      int8_t signedOffset)
43 {
44     int16_t answer;
45
46   // Clear interrupt flag
47   adc_ch->INTFLAGS = ADC_CH_CHIF_bm;
48
49   // Return result register contents
50   answer = adc_ch->RES - signedOffset;
51   return answer;
52 }
53
54
55 /* Calculates the offset on a given ADC channel.
56  *
57  * This function does one or several measurements to
```

```
58  * determine the offset of the ADC. The ADC must be
59  * configured and enabled before this function is run.
60  * It only returns the low byte of the 12-bit
61  * conversion, because the offset should never
62  * be more than +-8 LSB off.
63  *
64  */
65 int8_t ADC_Offset_Get_Signed(ADC_t * adc, ADC_CH_t *ch,
        bool oversampling)
66 {
67     if (oversampling)
68     {
69     int16_t offset=0;
70     for (int i=0; i<4; i++)
71     {
72       ADC_Ch_Conversion_Start(ch);
73
74       do {
75       } while (!ADC_Ch_Conversion_Complete(ch));
76
77       // Returns ch.RES and adds it to offset
78       offset += ADC_ResultCh_GetWord_Signed(ch, 0x00);
79     }
80     // Takes the average offset value and throws the
           excess high-order bits
81     return ((int8_t)(offset/4));
82     }
83     else //** Do one conversion to find offset:
84     {
85       int8_t offset=0;
86
87       // Starts a conversion on channel "ch"
88       ADC_Ch_Conversion_Start(ch);
89
90       do{
91       }while(!ADC_Ch_Conversion_Complete(ch));
92       offset = (uint8_t)ADC_ResultCh_GetWord_Signed(ch,
             0x00);
93
94       return offset;
95     }
96 }
97
98 /* Initializes the system ADC.
99  *
100 * Sets up the ADc, and so on. RTFM.
101 *
102 */
103 void ADC_init(void)
```

```
104 {
105   PORTA.DIRSET = 0x00;
106
107   // Clear RESOLUTION and CONVMODE Mode
108   // Set 8bit ADC resolution
109   // Enable signed conversion mode
110   ADCA.CTRLB =
111         ((ADCA.CTRLB & ~(ADC_RESOLUTION_gm |
              ADC_CONMODE_bm))
112     | ADC_RESOLUTION_12BIT_gc
113     | (ADC_ConvMode_Signed ? ADC_CONMODE_bm : 0));
114
115   // Clear PRESCALER bits
116   // Set ADC_clk to Peripheral_clk/256
117   ADCA.PRESCALER =
118         (ADCA.PRESCALER & ~(ADC_PRESCALER_gm))
119     | ADC_PRESCALER_DIV64_gc;
120
121   // Clear REFSEL bits
122   // Set Reference to Internal Vcc/1.6
123   ADCA.REFCTRL =
124         (ADCA.REFCTRL & ~(ADC_REFSEL_gm))
125     | ADC_REFSEL_AREFA_gc;
126
127   // Clear Ch0 Input Mode and Gain factor
128   // Enable single-ended or differential input signal
129   // Set input amplification to 1x
130   ADCA.CH0.CTRL =
131         (ADCA.CH0.CTRL & ~(ADC_CH_INPUTMODE_gm
132         | ADC_CH_GAINFAC_gm))
133     | ADC_CH_INPUTMODE_DIFF_gc
134     | ADC_CH_GAIN_1X_gc;
135
136   // Clear Ch1 Input Mode and Gain factor
137   // Enable single-ended or differential input signal
138   // Set input amplification to 1x
139   ADCA.CH1.CTRL =
140         (ADCA.CH1.CTRL & ~(ADC_CH_INPUTMODE_gm
141         | ADC_CH_GAINFAC_gm))
142     | ADC_CH_INPUTMODE_DIFF_gc
143     | ADC_CH_GAIN_1X_gc;
144
145   // Clear Ch1 Input Mode and Gain factor
146   // Enable single-ended or differential input signal
147   // Set input amplification to 1x
148   ADCA.CH2.CTRL =
149         (ADCA.CH2.CTRL & ~(ADC_CH_INPUTMODE_gm |
              ADC_CH_GAINFAC_gm))
150     | ADC_CH_INPUTMODE_DIFF_gc
```

```
151     | ADC_CH_GAIN_1X_gc;
152
153   // Clear Ch1 Input Mode and Gain factor
154   // Enable single-ended or differential input signal
155   // Set input amplification to 1x
156   ADCA.CH3.CTRL =
157         (ADCA.CH3.CTRL & ~(ADC_CH_INPUTMODE_gm |
              ADC_CH_GAINFAC_gm))
158     | ADC_CH_INPUTMODE_DIFF_gc
159     | ADC_CH_GAIN_1X_gc;
160
161   // Set ADC4 pin as both positive and negative analog
        input
162   ADCA.CH0.MUXCTRL =
163         (uint8_t) ADC_CH_MUXPOS_PIN4_gc
164     | ADC_CH_MUXNEG_PIN4_gc;
165
166   // Set ADC5 pin as both positive and negative analog
        input
167   ADCA.CH1.MUXCTRL =
168         (uint8_t) ADC_CH_MUXPOS_PIN5_gc
169     | ADC_CH_MUXNEG_PIN5_gc;
170
171   // Set ADC6 pin as both positive and negative analog
        input
172   ADCA.CH2.MUXCTRL =
173         (uint8_t) ADC_CH_MUXPOS_PIN6_gc
174     | ADC_CH_MUXNEG_PIN6_gc;
175
176   // Set ADC7 pin as both positive and negative analog
        input
177   ADCA.CH3.MUXCTRL =
178         (uint8_t) ADC_CH_MUXPOS_PIN7_gc
179     | ADC_CH_MUXNEG_PIN7_gc;
180
181   ADCA.CALL = ReadCalibrationByte(
182         offsetof(NVM_PROD_SIGNATURES_t, ADCACAL0) );
183   ADCA.CALH = ReadCalibrationByte(
184         offsetof(NVM_PROD_SIGNATURES_t, ADCACAL1) );
185
186   // Enable the ADC.
187   ADC_Enable(&ADCA);
188
189   // Wait until common mode voltage is stable.
190   ADC_Wait_8MHz(&ADCA);
191
192   // Returns the offset between the two differential
        inputs.
193   offset_ch0 =
```

```
194         ADC_Offset_Get_Signed(&ADCA, &ADCA.CH0, true);
195   offset_ch1 =
196         ADC_Offset_Get_Signed(&ADCA, &ADCA.CH1, true);
197   offset_ch2 =
198         ADC_Offset_Get_Signed(&ADCA, &ADCA.CH2, true);
199   offset_ch3 =
200         ADC_Offset_Get_Signed(&ADCA, &ADCA.CH3, true);
201
202   // Disable the ADC
203   ADCA.CTRLA = ADCA.CTRLA & (~ADC_ENABLE_bm);
204
205   // Set ADC5 pin as positive analog input.
206   // Set ADC4 pin as negative analog input.
207   ADCA.CH0.MUXCTRL =
208         (uint8_t) ADC_CH_MUXPOS_PIN4_gc
209         | ADC_CH_MUXNEG_PIN2_gc;
210   ADCA.CH1.MUXCTRL =
211         (uint8_t) ADC_CH_MUXPOS_PIN5_gc
212         | ADC_CH_MUXNEG_PIN2_gc;
213   ADCA.CH2.MUXCTRL =
214         (uint8_t) ADC_CH_MUXPOS_PIN6_gc
215         | ADC_CH_MUXNEG_PIN2_gc;
216   ADCA.CH3.MUXCTRL =
217         (uint8_t) ADC_CH_MUXPOS_PIN7_gc
218         | ADC_CH_MUXNEG_PIN2_gc;
219
220   // Clear the SWEEP bits.
221   // Set channel 0 as active channel for sweep.
222   ADCA.EVCTRL = (ADCA.EVCTRL & ~(ADC_SWEEP_gm))
223         | ADC_SWEEP_0123_gc;
224
225   // Enable the ADC.
226   ADC_Enable(&ADCA);
227
228   // Wait until common mode voltage is stable.
229   ADC_Wait_8MHz(&ADCA);
230
231   // Enable free running mode.
232   ADCA.CTRLB |= ADC_FREERUN_bm;
233
234 }
235
236
237 /* Function: ADC_Wait_8Mhz()
238  *
239  * 1) Store current PRESCALER Value
240  * 2) Set new PRESCALER value to Sys_clk/4
241  * 3) Delay 4 cycles
242  * 4) Restore old PRESCALER value
```

```
243  * 5) Return
244  *
245  * NOTE: Default clk is 2MHz and therefore
246  * below the maximum frequency to use this
247  * function.
248  *
249  */
250 void ADC_Wait_8MHz(ADC_t * adc)
251 {
252   /* Store old prescaler value. */
253   uint8_t prescaler_val = adc->PRESCALER;
254
255   /* Set prescaler value to minimum value. */
256   adc->PRESCALER = ADC_PRESCALER_DIV4_gc;
257
258   /* Wait 4*COMMON_MODE_CYCLES for common mode to
        settle. */
259   _delay_us(4*COMMON_MODE_CYCLES);
260
261   /* Set prescaler to old value*/
262   adc->PRESCALER = prescaler_val;
263 }
264
265 /* Reads the calibration bytes from factory.
266  *
267  *  Bla bla bla.
268  */
269 uint8_t ReadCalibrationByte(uint8_t index)
270 {
271   uint8_t result;
272
273   /* Load the NVM Command register
274     *  to read the calibration row.
275     */
276   NVM_CMD = NVM_CMD_READ_CALIB_ROW_gc;
277   result = pgm_read_byte(index);
278
279   /* Clean up NVM Command register. */
280   NVM_CMD = NVM_CMD_NO_OPERATION_gc;
281   return(result);
282 }
```

Listing F.4: adc.h

```
1 #ifndef ADC_H
2 #define ADC_H
3
4 #include <stdbool.h>
5
```

```
6  //************************************************
7  // Definitions for the Analog-to-Digital peripheral.
8  //************************************************
9  // Signed conversion mode flag.
10 #define ADC_ConvMode_Signed     true
11 // Unsigned conversion mode flag.
12 #define ADC_ConvMode_Unsigned     false
13 // Number of common mode cycles.
14 #define COMMON_MODE_CYCLES      16
15 // Enable selected ADC.
16 #define ADC_Enable(_adc) ((_adc)->CTRLA |=
       ADC_ENABLE_bm)
17 // Disable selected ADC.
18 #define ADC_Disable(_adc) ((_adc)->CTRLA = (_adc)->
       CTRLA & (~ADC_ENABLE_bm))
19 // Starts a single conversion on a channel.
20 #define ADC_Ch_Conversion_Start(_adc_ch) ((_adc_ch)->
       CTRL |= ADC_CH_START_bm)
21 // True or false flag which sets upon conversion
       complete on a given channel.
22 #define ADC_Ch_Conversion_Complete(_adc_ch) \
23     (((_adc_ch)->INTFLAGS & ADC_CH_CHIF_bm) != 0x00)
24 // Selects the analog input pins on a channel. See
       ADC_CH_MUXPOS/NEG_enum types.
25 #define ADC_Ch_InputMux_Config(_adc_ch, _posInput,
       _negInput) \
26     ((_adc_ch)->MUXCTRL = (uint8_t)(_posInput |
          _negInput))
27
28
29 //************************************************
30 // Global variables
31 //************************************************
32 extern int8_t   offset_ch0, offset_ch1, offset_ch2,
       offset_ch3;
33 int8_t offset_ch0, offset_ch1, offset_ch2, offset_ch3;
34
35 //************************************************
36 // Function prototypes
37 //************************************************
38 uint16_t  ReadADC(ADC_CH_t *adc_ch, uint8_t offset,
       uint8_t samples);
39 int16_t   ADC_ResultCh_GetWord_Signed(ADC_CH_t * adc_ch
       , int8_t signedOffset);
40 int8_t    ADC_Offset_Get_Signed(ADC_t * adc, ADC_CH_t *
       ch, bool oversampling);
41 void    ADC_init(void);
42 void    ADC_Wait_8MHz(ADC_t * adc);
43 uint8_t   ReadCalibrationByte(uint8_t index);
```

```
44
45 #endif //ADC_H
```

Listing F.5: clock.c

```
1  #include <inttypes.h>
2  #include <avr/io.h>
3  #include <util/delay.h>
4  #include "avr_compiler.h"
5
6  void CCPWrite(volatile uint8_t * address, uint8_t value
       );
7
8  void CLK_init(void)
9  {
10     /* Enable 32 MHz Internal RC Oscillator */
11     OSC.CTRL |= OSC_RC32MEN_bm;
12
13     /*
14     * Select Prescaler Division Factors, where:
15     * "A" selects the clock frequency of clk_PER4
16     * relative to clk_SYS. "B" sets the clock frequency
17     * of clk_PER2 relative to clk_PER4, and "C" sets
18     * the clock frequency of clk_PER and clk_CPU
19     * relative to clk_PER2.
20   *
21   * With internal 32MHz osc, A = 1, B = C = 2
22     * the system is running with:
23   *
24   * ---------------------------------------
25   *  Clk_sys = 32 MHZ
26   *  Clk_per4 = Clk_sys / 1 = 32 MHZ
27   *  Clk_per2 = Clk_per4 / 2 = 16 MHz
28   *  Clk_per = Clk_per2 / 2 = 8 MHz
29   * ---------------------------------------
30     *
31   */
32     uint8_t PSconfig =
33         (uint8_t) CLK_PSADIV_1_gc
34         | CLK_PSBCDIV_2_2_gc;
35     CCPWrite(&CLK.PSCTRL, PSconfig);
36
37
38     /*
39     * Wait until 32MHz Internal RC Oscillator
40     * is stable and ready to be used as Sysclk,
41     * and then enable it.
42     */
43     do {} while ((OSC.STATUS & OSC_RC32MRDY_bm) == 0);
```

```
44      uint8_t clkCtrl =
45          (CLK.CTRL & ~CLK_SCLKSEL_gm)
46          | CLK_SCLKSEL_RC32M_gc;
47      CCPWrite(&CLK.CTRL, clkCtrl);
48      clkCtrl = (CLK.CTRL & CLK_SCLKSEL_RC32M_gc);
49
50  }
51
52  /*  CCP write helper function written in assembly.
53   *
54   *  This function is written in assembly because
55   *  of the time critical operation of writing to
56   *  the registers.
57   *
58   *  address: A pointer to the address to write to.
59   *  value:   The value to put in to the register.
60   */
61  void CCPWrite(volatile uint8_t * address, uint8_t value
        )
62  {
63  #ifdef __ICCAVR__
64
65      // Store global interrupt setting in scratch
66      // register and disable interrupts.
67          asm("in  R1, 0x3F \n"
68          "cli"
69          );
70
71      // Move destination address pointer
72      // to Z pointer registers.
73      asm("movw r30, r16");
74  #ifdef RAMPZ
75      asm("ldi  R16, 0 \n"
76              "out  0x3B, R16"
77          );
78
79  #endif
80      asm("ldi  r16,  0xD8 \n"
81          "out  0x34, r16  \n"
82  #if (__MEMORY_MODEL__ == 1)
83          "st     Z,  r17  \n");
84  #elif (__MEMORY_MODEL__ == 2)
85          "st     Z,  r18  \n");
86  #else /* (__MEMORY_MODEL__ == 3) || (__MEMORY_MODEL__
        == 5) */
87          "st     Z,  r19  \n");
88  #endif /* __MEMORY_MODEL__ */
89
```

```
90      // Restore global interrupt setting from scratch
            register.
91          asm("out  0x3F, R1");
92
93  #elif defined __GNUC__
94      AVR_ENTER_CRITICAL_REGION( );
95      volatile uint8_t * tmpAddr = address;
96  #ifdef RAMPZ
97      RAMPZ = 0;
98  #endif
99      asm volatile(
100         "movw r30,  %0"       "\n\t"
101         "ldi  r16,  %2"       "\n\t"
102         "out   %3, r16"       "\n\t"
103         "st     Z,  %1"       "\n\t"
104         :
105         : "r" (tmpAddr), "r" (value), "M" (CCP_IOREG_gc
                ), "i" (&CCP)
106         : "r16", "r30", "r31"
107         );
108
109     AVR_LEAVE_CRITICAL_REGION( );
110 #endif
111 }
```

Listing F.6: clock.h

```
1  #ifndef CLOCK_H
2  #define CLOCK_H
3
4  //************************************************
5  // Function prototypes
6  //************************************************
7  void CLK_init(void);
8  void CCPWrite(volatile uint8_t * address, uint8_t value
       );
9
10 /* This macro will protect the following code
11  * from interrupts.
12  */
13 #define AVR_ENTER_CRITICAL_REGION( ) uint8_t volatile
       saved_sreg = SREG; \
14                                          cli();
15
16 /* This macro must always be used in conjunction
17  * with AVR_ENTER_CRITICAL_REGION
18  * so the interrupts are enabled again.
19  */
20 #define AVR_LEAVE_CRITICAL_REGION( ) SREG = saved_sreg;
```

```
21 #endif //CLOCK_H
```

Listing F.7: battery.c

```
1  #include <avr/io.h>
2  #include "adc.h"
3  #include "battery.h"
4  #include "lifespecs.h"
5
6  Batteries_t BattData;
7
8  //*********************************************
9  // Functions
10 //*********************************************
11 /* Refreshes battery status information
12  * TODO
13  */
14 unsigned char BatteryStatusRefresh(void)
15 {
16   // Assume the worst..
17   unsigned char success = FALSE;
18
19   BattData.Charged = FALSE;
20   BattData.Low = TRUE;
21   BattData.Temperature = 0;
22   BattData.Capacity = 0;
23   BattData.MaxCurrent = 0;
24   BattData.MaxTime = 0;
25   BattData.MinCurrent = 0;
26
27   uint16_t Temperature = BatteryTemp();
28
29   if(Temperature >= BAT_TEMPERATURE_MAX || Temperature
       <= BAT_TEMPERATURE_MIN) {
30     //BattData.Temperature = Temperature;
31     //Flag battery temperature
32   }
33
34   uint16_t BatVoltage = BatteryVoltage();
35
36   // Is the battery voltage above minimum safe cell
       voltage?
37   if (BatVoltage >= BAT_VOLTAGE_MIN) {
38     BattData.Low = FALSE;
39   }
40
41   // Is the battery charged?
42   if (BatVoltage >= BAT_VOLTAGE_LOW) {
43     BattData.Charged = TRUE;
```

```
44    }
45
46    if ((!BattData.Low)) {
47      success = TRUE;
48    } else {
49      BattData.Low = FALSE;  // (This is just a
          technicality..)
50      success = FALSE;
51    }
52
53    return(success); //when is it false...? eh?
54  }
55
56  uint16_t BatteryVoltage(void)
57  {
58    uint16_t BatVoltage = ReadADC(&ADCA.CH2, offset_ch2,
        5);
59    BattData.Voltage = BatVoltage;
60    return(BatVoltage);
61  }
62
63  uint16_t BatteryTemp(void)
64  {
65    uint16_t BatTemperature = ReadADC(&ADCA.CH3,
        offset_ch3, 5);
66    BattData.Temperature = BatTemperature;
67    return(BatTemperature);
68  }
```

Listing F.8: battery.h

```
1  #ifndef BATTERY_H
2  #define BATTERY_H
3  #define TRUE  1
4  #define FALSE 0
5
6  //***********************************************
7  // Battery struct declarations
8  //***********************************************
9  /* Holds status and various data for the battery
10  *
11  * These data are updated by
12  * BatteryStatusRefresh() and
13  * BatteryDataRefresh().
14  *
15  * TODO
16  */
17 typedef struct
18 {
```

```
19      // Battery fully charged. (TRUE/FALSE)
20      unsigned char Charged   : 1;
21      // Battery low voltage. (TRUE/FALSE)
22      unsigned char Low       : 1;
23      // Battery temperature, in centigrade.
24      uint16_t Temperature;
25      uint16_t Voltage;
26      // Capacity, in mAh.
27      unsigned int  Capacity;
28      // Charge current, in mA.
29      unsigned int  MaxCurrent;
30      // Charge cut-off time, in minutes.
31      unsigned int  MaxTime;
32      // Cut-off current, in mA.
33      unsigned int  MinCurrent;
34  } Batteries_t;
35
36  //************************************************
37  // Global variable(s)
38  //************************************************
39  extern Batteries_t BattData;
40
41  //************************************************
42  // Function prototypes
43  //************************************************
44  unsigned char BatteryCheck(void);
45  unsigned char BatteryStatusRefresh(void);
46  uint16_t BatteryVoltage(void);
47  uint16_t BatteryTemp(void);
48
49  #endif //BATTERY_H
```

Listing F.9: fsm.c

```
1   #include <avr/io.h>
2   #include <stdint.h>
3   #include <util/delay.h>
4   #include <stdio.h>
5   #include "fsm.h"
6   #include "pwm.h"
7   #include "main.h"
8   #include "pid.h"
9
10  void ST_CVoltage(void) {
11      // Run PID calculations once every PID timer
            timeout
12      if(gFlags.pidTimer) {
13          RefValue = GetReference();
14          MeasuredValue = GetMeasurement();
```

```
15          InputValue = pid_Controller(RefValue,
                MeasuredValue, &pidData);
16
17          Set_Input(InputValue);
18          gFlags.pidTimer = FALSE;
19      }
20    _delay_ms(1);
21  }
22
23  void ST_Disconnect(void) {
24    PWM_Stop();
25  }
```

Listing F.10: fsm.h

```
1  #ifndef FSM_H
2  #define FSM_H
3  //*****************************************************
4  // Global variables
5  //*****************************************************
6  extern int16_t RefValue, MeasuredValue, InputValue;
7
8  //*****************************************************
9  // Function prototypes
10 //*****************************************************
11 void ST_CVoltage(void);
12 void ST_Disconnect(void);
13
14 #endif //FSM_H
```

Listing F.11: mppt.c

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <inttypes.h>
4  #include <stdio.h>
5  #include "mppt.h"
6  #include "pwm.h"
7
8  /* MAXIMUM POWER POINT TRACKING
9  * ------------------------------
10 * Tracks the maximum power point of the solar panel
11 * according to the Perturb and Observe Method.
12 *
13 * Returns status information.
14 */
15
16 //*****************************************************
17 // MPPT Algorithm nr1: "Perturb and Observe"
```

```
18 //*****************************************************
19 /*  Track the maximum power point on the solar panel.
20  *
21  *  This function tracks the maximum power point (MPP)
22  *  on the solar panel according to the Perturb and
23  *  Observe Method. TODO: A flag is set when the
        battery
24  *  has reached its maximum voltage level, and a new
25  *  state is returned to the main loop.
26  */
27 unsigned char MPPT(uint16_t current, uint16_t voltage)
       {
28   uint32_t Power = (uint32_t)voltage * (uint32_t)
         current;
29     //check for old = new case here
30     if (Power > OldPower) {
31         if (voltage > OldVoltage) {
32             PWM_DecrementDutyCycle(1);
33         } else if (voltage < OldVoltage) {
34             PWM_IncrementDutyCycle(1);
35         }
36     } else if (Power < OldPower) {
37         if (voltage > OldVoltage) {
38             PWM_IncrementDutyCycle(1);
39         } else if (voltage < OldVoltage) {
40             PWM_DecrementDutyCycle(1);
41         }
42     }
43     OldPower = Power;
44     OldVoltage = voltage;
45
46   _delay_ms(1); //debugging
47     return(1);//ChargeParameters.NextState);
48 }
```

Listing F.12: mppt.h

```
1  #ifndef MPPT_H
2  #define MPPT_H
3
4  uint32_t OldPower;
5  uint16_t OldVoltage;
6
7  //*****************************************************
8  // Global variable(s)
9  //*****************************************************
10 extern uint16_t DutyCycle;
11
12 //*****************************************************
```

```
13 // Function prototypes
14 //*****************************************************
15 unsigned char MPPT(uint16_t current, uint16_t voltage);
16 #endif
```

Listing F.13: pid.c

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <inttypes.h>
4  #include <stdbool.h>
5  #include <stdlib.h>
6  #include <stdint.h>
7  #include "adc.h"
8  #include "battery.h"
9  #include "lifespecs.h"
10 #include "pwm.h"
11 #include "pid.h"
12
13 pidData_t pidData;
14
15 /*
16  * Initialization of PID controller parameters.
17  */
18 void pid_Init(int16_t p_factor, int16_t i_factor,
       int16_t d_factor, pidData_t *pid)
19 {
20   // Setup Timer/Counter1 for PID.
21   TC1_init();
22
23   // Start values
24     pid->sumError = 0;
25     pid->lastProcessValue = 0;
26
27     // Tuning constants
28     pid->P_Factor = p_factor;
29     pid->I_Factor = i_factor;
30     pid->D_Factor = d_factor;
31
32     // Limits to avoid overflow
33     pid->maxError = MAX_INT / (pid->P_Factor + 1);
34     pid->maxSumError = MAX_I_TERM / (pid->I_Factor + 1)
         ;
35 }
36
37
38 /* Controller Algorithm
39 *
40 *  This function calculates the PID error and returns
```

```
41 *  the PID. Takes as input: desired (setPoint) value,
42 *  the measured (processValue) value and PID status (
       pid_st).
43 */
44 int16_t pid_Controller( int16_t setPoint,
45                         int16_t processValue,
46                         pidData_t *pid_st)
47 {
48     int16_t error, p_term, d_term;
49     int32_t i_term, ret, temp;
50
51     error = setPoint - processValue;
52
53     // Calculate Pterm and limit error overflow
54     if(error > pid_st->maxError) {
55         p_term = MAX_INT;
56     } else if(error < -pid_st->maxError) {
57         p_term = -MAX_INT;
58     } else {
59         p_term = pid_st->P_Factor * error;
60     }
61
62     // Calculate Iterm and limit integral runaway
63     temp = pid_st->sumError + error;
64     if(temp > pid_st->maxSumError) {
65         i_term = MAX_I_TERM;
66         pid_st->sumError = pid_st->maxSumError;
67     } else if(temp < -pid_st->maxSumError) {
68         i_term = -MAX_I_TERM;
69         pid_st->sumError = -pid_st->maxSumError;
70     } else {
71         pid_st->sumError = temp;
72         i_term = pid_st->I_Factor * pid_st->sumError;
73     }
74
75     // Calculate Dterm
76     d_term = pid_st->
77         D_Factor * (pid_st->lastProcessValue -
               processValue);
78     pid_st->lastProcessValue = processValue;
79
80     ret = (p_term + i_term + d_term) / SCALING_FACTOR;
81     if(ret > MAX_INT) {
82         ret = MAX_INT;
83     } else if(ret < -MAX_INT) {
84         ret = -MAX_INT;
85     }
86     return((int16_t)ret);
87 }
```

```
88
89 /*
90  * Resets the integrator
91  */
92 void pid_Reset_Integrator(pidData_t *pid_st)
93 {
94     pid_st->sumError = 0;
95 }
96
97 int16_t GetReference(void)
98 {
99     // Returns CELL_VOLTAGE_MAX.
100     return CELL_VOLTAGE_MAX;
101 }
102
103 int16_t GetMeasurement(void)
104 {
105   return(BatteryVoltage());
106     //return(ReadADC(&ADCA.CH2, offset_ch2, 11));
107 }
108
109 /* Sets the PID output as input to the system.
110 *
111 *  This function takes the output value from
112 *  the PID controller and level shifts it into
113 *  a positive range, normalizes the output
114 *  before setting the the new dutycycle.
115 */
116 void Set_Input(int16_t inputValue)
117 {
118   uint8_t NewDutyCycle;
119     // Level shift the PID output value to a positive
        range.
120     shifted_inputValue = inputValue + abs(PID_ERROR_MIN
        );
121     // Normalize input range.
122     normalized_inputValue = ((shifted_inputValue/
        PID_SHIFTED_ERROR_MAX));
123     // Calculate the new dutycycle
124     NewDutyCycle = DUTYCYCLE_MIN + (
        DUTYCYCLE_VALID_RANGE * normalized_inputValue);
125     // Set the new dutycycle
126     SetDutyCycle(NewDutyCycle);
127 }
128
129 void TC1_init(void)
130 {
131   /* Set the clock source to
132         Clk_per/1024 = 8MHz/1024 = 7812,5 Hz
```

```
133         = 1 / 7812,5 s = 128us clock tick
134     */
135   TCC1.CTRLA =
136         (TCC1.CTRLA & ~TC1_CLKSEL_gm)
137         | TC_CLKSEL_DIV8_gc;//64_gc;
138
139   /* Set the Timer Waveform Generation Mode
140     *  to normal, i.e. no waveform, and enable
141     *  CC on channel A.
142     */
143   TCC1.CTRLB =   (TCC1.CTRLB & ~TC1_WGMODE_gm)
144                     | TC_WGMODE_NORMAL_gc
145                     | TC1_CCAEN_bm;
146
147   /* Enable interrupts on CC channel A
148     *  with HI priority
149     */
150   TCC1.INTCTRLB = (uint8_t) TC_CCAINTLVL_HI_gc;
151
152   /* Set the period for Timer1 */
153   TCC1.PER = 0x0FF0;
154
155   /*
156   * 16 bit Compare or Capture register.
157     * CCxH + CCxL = CCA. The reg is updated from
158     * CCaBUF. If CNT equals CCA (or CCx),
159   * the comparator signals a match that sets
160     * the CC channel's flag at the next timer
161     * clock cycle and the event and optional
162   * interrupt is generated.
163   */
164   TCC1.CCA = 0x0FF0;
165 }
```

Listing F.14: pid.h

```
1  #ifndef PID_H
2  #define PID_H
3
4  #include "stdint.h"
5
6  int16_t RefValue, MeasuredValue, InputValue,
       shifted_inputValue;
7  float normalized_inputValue;
8
9  // Limits for the error
10 #define PID_ERROR_MAX           225.0
11 #define PID_ERROR_MIN           -73.0
12 #define PID_SHIFTED_ERROR_MAX
```

```
13          (PID_ERROR_MAX + abs(PID_ERROR_MIN))
14  #define PID_SHIFTED_ERROR_MIN
15          (PID_ERROR_MIN + abs(PID_ERROR_MIN))
16
17  // Maximum values used to avoid sign/overflow problems
18  #define MAX_INT          INT16_MAX
19  #define MAX_LONG         INT32_MAX
20  #define MAX_I_TERM         (MAX_LONG / 2)
21
22  // Boolean values
23  #define FALSE          0
24  #define TRUE           1
25
26  //*************************************
27  // Definitions for the PID gains which
28  // should be modified after tuning.
29  //*************************************
30  // Proportional gain
31  #define K_P            0.35
32  // Integral gain
33  #define K_I            0.26
34  // Derivative gain
35  #define K_D            0.005
36
37  //*************************************
38  // Definition for the desired PID sample
39  // time interval. For a n-bit counter:
40  // TIME_INTERVAL = (desired interval
41  // [sec]) * (frequency [Hz]) / (2^n-1)
42  //*************************************
43  //! PID sampling time interval
44  #define TIME_INTERVAL        20
45
46  //*************************************
47  // Definition for the scaling factor
48  //*************************************
49  //! Scaling factor.
50  #define SCALING_FACTOR       128
51
52  //*************************************
53  // PID Status declarations
54  //*************************************
55  /* Holds the setpoints and data used by
56   * the PID control algorithm.
57   */
58  typedef struct {
59    int16_t lastProcessValue;
60    int32_t sumError;
61    int16_t P_Factor;
```

```
62    int16_t I_Factor;
63    int16_t D_Factor;
64    int16_t maxError;
65    int32_t maxSumError;
66  } pidData_t;
67
68  extern pidData_t pidData;
69  //struct PID_DATA pidData;
70  //pidData_t pidData; ??
71  /*
72  struct PID_DATA {
73    int16_t lastProcessValue;
74    int32_t sumError;
75    int16_t P_Factor;
76    int16_t I_Factor;
77    int16_t D_Factor;
78    int16_t maxError;
79    int32_t maxSumError;
80  };
81  typedef struct PID_DATA pidData_t;*/
82
83
84  //************************************
85  // Function prototypes
86  //************************************
87  // Returns the reference voltage
88  // (in ADC counts) for the PID algorithm.
89  int16_t GetReference(void);
90  // Returns the battery voltage from
91  // ADC channel 2.
92  int16_t GetMeasurement(void);
93  // Sets the dutycycle to the new input.
94  void Set_Input(int16_t inputValue);
95  // Initialize the the PID routine.
96  void pid_Init(int16_t p_factor,
97                int16_t i_factor,
98                int16_t d_factor,
99                pidData_t *pid);
100 // The actual PID algorithm. Returns the
101 // new inputvalue for the converter.
102 int16_t pid_Controller(int16_t setPoint,
103         int16_t processValue,
104         pidData_t *pid);
105 // Resets the integrator to avoid integral run-away.
106 void pid_Reset_Integrator(pidData_t *pid_st);
107 // Timer/Counter for PID
108 void TC1_init(void);
109
110 #endif
```

Listing F.15: pwm.c

```
1  #include <avr/io.h>
2  #include <stdio.h>
3  #include "pwm.h"
4
5  //*********************************************
6  // Starts the PWM output to the high- and
7  // low side switches.
8  //*********************************************
9  /* Starts PWM output with
10  * dead-time insertion on PORTC.
11  */
12 void PWM_Start(void)
13 {
14   // Enable output on PC0.
15     PORTC.DIRSET = (PIN0_bm | PIN1_bm);
16     // Set 16bit period.
17     TCC0.PER = (PWM_period & HiResMask);
18     // Enable Single Slope PWM and Enable Compare
          Capture Channel A (CCA).
19     TCC0.CTRLB = (TCC0.CTRLB & ~TC0_WGMODE_gm) |
          TC_WGMODE_SS_gc | TC0_CCAEN_bm;
20   // Selects starts a clock source. Must use DIV1 if Hi
       -Res enabled.
21     TCC0.CTRLA = (TCC0.CTRLA & ~TC0_CLKSEL_gm) |
          TC_CLKSEL_DIV1_gc;
22
23   // Enable interrupts on CC channel A with HI priority
       .
24   TCC0.INTCTRLA = (uint8_t) TC_CCAINTLVL_LO_gc;
25
26   // Enable Dead-Time Generator for CCA and mirror Low-
       side signal on PC1.
27     AWEXC.CTRL |= AWEX_DTICCAEN_bm;
28     // Enables Output Override on pin 0 and 1.
29     AWEXC.OUTOVEN = (PIN0_bm | PIN1_bm);
30     // Writes number of peripheral clock cycles of Dead
          -Time to Both Sides.
31     // AWEXC.DTBOTH = 2;
32   // Sets number of peripheral clock cycles of Dead-
       Time to Low side.
33   AWEXC.DTLS = 3;
34   // Writes number of peripheral clock cycles of Dead-
       Time to High Sides.
35   AWEXC.DTHS = 3;
36   // Enable High Resolution on Timer/Counter 0.
37     HIRESC.CTRL |= HIRES_HREN_TC0_gc;
38   // Set default dutycycle. TCC0.CCABUF =
       DUTYCYCLE_DEFAULT;
```

```
39    DutyCycle = SetDutyCycle(DUTYCYCLE_DEFAULT);
40    TCC0.CCABUF = DutyCycle;
41  }
42
43  //***********************************************
44  // Stops the PWM output to the
45  // high- and low side switches.
46  //***********************************************
47  /*! Stops PWM output, and activates the internal
48   *  pull-downs on the NMOS gates, leaving the the
49   *  switches in an open state.
50   */
51  void PWM_Stop(void)
52  {
53    // Turn off Timer/Counter (no clock selected)
54    TCC0.CTRLA = (TCC0.CTRLA & ~TC0_CLKSEL_gm) |
        TC_CLKSEL_OFF_gc;
55    // Set waveform generation to normal mode,
56      // i.e, NO waveform generation.
57    TCC0.CTRLB = 0;
58    // Turn off Advanced Waveform Generation module.
59    AWEXC.CTRL = 0;
60    // Turn off Hi Resolution module.
61    HIRESC.CTRL = 0;
62    // Set port C to input.
63    PORTC.DIR = 0;
64    // Enable Totempole w/ pull-down on PIN0
65    PORTC.PIN0CTRL = PORT_OPC_PULLDOWN_gc;
66    // Enable Totempole w/ pull-down on PIN1
67    PORTC.PIN1CTRL = PORT_OPC_PULLDOWN_gc;
68
69    //SetErrorFlag(PWM_STOPPED);
70  }
71
72  /* Decrements the PWM duty cycle, if not already at min
       .
73   *
74   * TRUE Success, duty cycle could be incremented.
75   * FALSE Failure, duty cycle already at maximum.
76   */
77  unsigned char PWM_DecrementDutyCycle(uint8_t StepSize){
78    if (DutyCycle > DUTYCYCLE_MIN)  {
79      DutyCycle -= StepSize;
80      return(TRUE);
81    } else {
82      return(FALSE);
83    }
84  }
85
```

```
86 /*  Increments the PWM duty cycle, if not already at
       max.
87  *
88  *  TRUE Success, duty cycle could be incremented.
89  *  FALSE Failure, duty cycle already at maximum.
90  */
91 unsigned char PWM_IncrementDutyCycle(uint8_t StepSize){
92   if (DutyCycle < DUTYCYCLE_MAX) {
93     DutyCycle += StepSize;
94     return(TRUE);
95   } else {
96     return(FALSE);
97   }
98 }
99
100 /* Sets the PWM dutycycle within the defined
101  * MAX and MIN values.
102  *
103  * TRUE Success, duty cycle could be incremented.
104  * FALSE Failure, duty cycle already at maximum.
105  */
106 uint8_t SetDutyCycle(uint8_t NewDutyCycle) {
107     // Check if new dutycycle is ok
108     // before setting the new value.
109     if(NewDutyCycle > DUTYCYCLE_MAX) {
110         DutyCycle = DUTYCYCLE_MAX;
111     //OutOfBounds = TRUE;
112     } else if(NewDutyCycle < DUTYCYCLE_MIN) {
113         DutyCycle = DUTYCYCLE_MIN;
114     } else DutyCycle = NewDutyCycle;
115
116     // Return the dutycycle value
117     return(DutyCycle);
118 }
```

Listing F.16: pwm.h

```
1  #ifndef PWM_H
2  #define PWM_H
3
4  //*********************************************
5  // Definitions for Pulse-Width-Modulation module.
6  //*********************************************
7  // The period used for the PWM clock (Timer0).
8  #define PWM_period        0x00FF
9  // High resoultion mask used to make sure the
10 // two LSBs in TCC0.PER are zero.
11 #define HiResMask         0xFFFC
12 // Maximum PWM dutycycle.
```

```
13 #define DUTYCYCLE_MAX     236
14 // Minimum PWM dutycycle.
15 #define DUTYCYCLE_MIN     180
16 // Valid range of the PWM dutycycle based
17 // on min/max limits.
18 #define DUTYCYCLE_VALID_RANGE
19        (DUTYCYCLE_MAX - DUTYCYCLE_MIN)
20 // A "safe" default PWM dutycycle for debugging etc.
21 #define DUTYCYCLE_DEFAULT   210
22
23 #define TRUE          1
24 #define FALSE         0
25
26 //*************************************************
27 // Global variables
28 //*************************************************
29 extern uint16_t DutyCycle;
30
31 uint16_t DutyCycle;
32 //*************************************************
33 // Function prototypes
34 //*************************************************
35 void PWM_Start(void);
36 void PWM_Stop(void);
37 unsigned char PWM_DecrementDutyCycle
38                 (uint8_t StepSize);
39 unsigned char PWM_IncrementDutyCycle
40                 (uint8_t StepSize);
41 uint8_t SetDutyCycle
42                 (uint8_t NewDutyCycle);
43
44 #endif //PWM_H
```

Listing F.17: serial.c

```
1  #include <avr/io.h>
2  #include <stdio.h>
3  #include "serial.h"
4
5  //*****************************************************
6  // Setup the USART on PORTC. See p241 in the datasheet.
7  //*****************************************************
8  void USART_init(void)
9  {
10     // Set PORTC[6:7] as output for serial
           communication.
11   PORTC.OUTSET = PIN6_bm;
12   PORTC.DIRSET = PIN6_bm;
13   /*  Set BSCALE = 0 and BSEL[10:0] = 51 (or 12).
```

```
14     *  This gives a baud rate of 9600 (or 38600) with a
           f_per = 8Mhz,
15     *  as per the equations on p. 238 in the datasheet.
16     */
17    USARTC0.BAUDCTRLB = (USARTC0.BAUDCTRLB & (~
          USART_BSCALE_gm | ~USART_BSEL_gm)) ;
18      USARTC0.BAUDCTRLA = 51;
19    // Disable interrupts
20      USARTC0.CTRLA = (USART_RXCINTLVL_OFF_gc |
            USART_TXCINTLVL_OFF_gc);
21    // Asynchronous mode, No parity, 8 bit, 1 Stop bit (
          default, i.e. USART_SBMODE_bm not set)
22      USARTC0.CTRLC = (USART_CMODE_ASYNCHRONOUS_gc |
            USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc);
23    // Enable transmitter
24    USARTC0.CTRLB = USART_TXEN_bm;
25  }
26
27  /* Macro for writing to the USART on PORTC.
28  *
29  *  This macro is used for debugging purposes only.
30  */
31  int uart_putchar(char c, FILE *stream)
32  {
33      if (c == '\n')
34          uart_putchar('\r', stream);
35
36    while (!(USARTC0.STATUS & USART_DREIF_bm));
37
38      USARTC0.DATA = c;
39      return 0;
40  }
```

Listing F.18: serial.h

```
1  #ifndef SERIAL_H
2  #define SERIAL_H
3
4  //*****************************************************
5  // Function prototypes
6  //*****************************************************
7  void USART_init(void);
8  int uart_putchar(char c, FILE *stream);
9
10 #endif //SERIAL_H
```

Listing F.19: twislave.c

```
1  #include <avr/io.h>
```

```
2  #include <stdio.h>
3  #include <util/delay.h>
4  #include "battery.h"
5  #include "main.h"
6  #include "twi_slave.h"
7
8  /*  Initalizes TWI slave driver structure.
9   *
10  *  Initialize the instance of the TWI Slave
11  *  and set the appropriate values.
12  *
13  *  twi:
14  *      The TWI_Slave_t struct instance.
15  *  module:
16  *      Pointer to the TWI module.
17  *  processDataFunction:
18  *      Pointer to the function that
19  *      handles incoming data.
20  */
21 void TWI_SlaveInitializeDriver(
22                 TWI_Slave_t *twi,
23                 TWI_t *module,
24                 void (*processDataFunction) (void))
25 {
26     twi->interface      = module;
27     twi->Process_Data   = processDataFunction;
28     twi->bytesReceived  = 0;
29     twi->bytesSent      = 0;
30     twi->status         = TWIS_STATUS_READY;
31     twi->result         = TWIS_RESULT_UNKNOWN;
32     twi->abort          = false;
33 }
34
35
36 /*  Initialize the TWI module.
37  *
38  *  Enables interrupts on address recognition
39  *  and data available. Remember to enable interrupts
40  *  globally from the main application.
41  *
42  *  twi:
43  *      The TWI_Slave_t struct instance.
44  *  address:
45  *      Slave address for this module.
46  *  intLevel:
47  *      Interrupt level for the TWI slave ISR.
48  */
49 void TWI_SlaveInitializeModule(
50                                 TWI_Slave_t *twi,
```

```
51                                  uint8_t address,
52                                  TWI_SLAVE_INTLVL_t
                                       intLevel)
53 {
54   twi->interface->SLAVE.CTRLA = intLevel |
55                                  TWI_SLAVE_DIEN_bm |
56                                  TWI_SLAVE_APIEN_bm |
57                                  TWI_SLAVE_ENABLE_bm;
58
59   twi->interface->MASTER.CTRLA &= 0;
60   twi->interface->SLAVE.ADDR = (address<<1);
61 }
62
63
64 /*  Common TWI slave interrupt service routine.
65  *
66  *  Handles all TWI transactions and responses to
67  *  address match, data reception, data transmission,
68  *  bus error and data collision.
69  *
70  *   twi:
71  *      The TWI_Slave_t struct instance.
72  */
73 void TWI_SlaveInterruptHandler(TWI_Slave_t *twi)
74 {
75   uint8_t currentStatus = twi->interface->SLAVE.STATUS;
76
77   /* If bus error. */
78   if (currentStatus & TWI_SLAVE_BUSERR_bm) {
79     twi->bytesReceived = 0;
80     twi->bytesSent = 0;
81     twi->result = TWIS_RESULT_BUS_ERROR;
82     twi->status = TWIS_STATUS_READY;
83   }
84
85   /* If transmit collision. */
86   else if (currentStatus & TWI_SLAVE_COLL_bm) {
87     twi->bytesReceived = 0;
88     twi->bytesSent = 0;
89     twi->result = TWIS_RESULT_TRANSMIT_COLLISION;
90     twi->status = TWIS_STATUS_READY;
91   }
92
93   /* If address match. */
94   else if ((currentStatus & TWI_SLAVE_APIF_bm) &&
95            (currentStatus & TWI_SLAVE_AP_bm)) {
96     TWI_SlaveAddressMatchHandler(twi);
97   }
98
```

```
99    /* If stop (only enabled through slave read
         transaction). */
100   else if (currentStatus & TWI_SLAVE_APIF_bm) {
101     TWI_SlaveStopHandler(twi);
102   }
103
104   /* If data interrupt. */
105   else if (currentStatus & TWI_SLAVE_DIF_bm) {
106     TWI_SlaveDataHandler(twi);
107   }
108
109   /* If unexpected state. */
110   else {
111     TWI_SlaveTransactionFinished(
112                     twi, TWIS_RESULT_FAIL);
113   }
114 }
115
116 /*  TWI address match interrupt handler.
117  *
118  *  Prepares TWI module for transaction when
119  *  an address match occurs.
120  *
121  *  twi:
122  *      The TWI_Slave_t struct instance.
123  */
124 void TWI_SlaveAddressMatchHandler(TWI_Slave_t *twi)
125 {
126   /* If application signalling need to abort (error
         occurred). */
127   if (twi->abort) {
128     twi->interface->SLAVE.CTRLB =
129                     TWI_SLAVE_CMD_COMPTRANS_gc;
130     TWI_SlaveTransactionFinished(
131                     twi, TWIS_RESULT_ABORTED);
132     twi->abort = false;
133   } else {
134     twi->status = TWIS_STATUS_BUSY;
135     twi->result = TWIS_RESULT_UNKNOWN;
136
137     /* Disable stop interrupt. */
138     uint8_t currentCtrlA =
139                     twi->interface->SLAVE.CTRLA;
140     twi->interface->SLAVE.CTRLA =
141                     currentCtrlA & ~TWI_SLAVE_PIEN_bm;
142
143     twi->bytesReceived = 0;
144     twi->bytesSent = 0;
145
```

```
146     /* Send ACK, wait for data interrupt. */
147     twi->interface->SLAVE.CTRLB =
148                     TWI_SLAVE_CMD_RESPONSE_gc;
149   }
150 }
151
152
153 /*   TWI stop condition interrupt handler.
154  *
155  *   twi:
156  *      The TWI_Slave_t struct instance.
157  */
158 void TWI_SlaveStopHandler(TWI_Slave_t *twi)
159 {
160   /* Disable stop interrupt. */
161   uint8_t currentCtrlA =
162             twi->interface->SLAVE.CTRLA;
163   twi->interface->SLAVE.CTRLA =
164             currentCtrlA & ~TWI_SLAVE_PIEN_bm;
165
166   /* Clear APIF, according to flowchart don't ACK or
        NACK */
167   uint8_t currentStatus =
168             twi->interface->SLAVE.STATUS;
169   twi->interface->SLAVE.STATUS =
170             currentStatus | TWI_SLAVE_APIF_bm;
171
172   TWI_SlaveTransactionFinished(twi, TWIS_RESULT_OK);
173 }
174
175 /*  TWI data interrupt handler.
176  *
177  *  Calls the appropriate slave read or write handler.
178  *
179  *  twi:
180  *      The TWI_Slave_t struct instance.
181  */
182 void TWI_SlaveDataHandler(TWI_Slave_t *twi)
183 {
184   if (twi->interface->SLAVE.STATUS & TWI_SLAVE_DIR_bm)
        {
185     TWI_SlaveWriteHandler(twi);
186   } else {
187     TWI_SlaveReadHandler(twi);
188   }
189 }
190
191 /*   TWI slave read interrupt handler.
192  *
```

```
193  *  Handles TWI slave read transactions and responses.
194  *
195  *   twi:
196  *      The TWI_Slave_t struct instance.
197  */
198 void TWI_SlaveReadHandler(TWI_Slave_t *twi)
199 {
200   /* Enable stop interrupt. */
201   uint8_t currentCtrlA = twi->interface->SLAVE.CTRLA;
202   twi->interface->SLAVE.CTRLA =
203             currentCtrlA | TWI_SLAVE_PIEN_bm;
204
205   /* If free space in buffer. */
206   if (twi->bytesReceived < TWIS_RECEIVE_BUFFER_SIZE) {
207     /* Fetch data */
208     uint8_t data = twi->interface->SLAVE.DATA;
209     twi->receivedData[twi->bytesReceived] = data;
210
211     /* Process data. */
212     twi->Process_Data();
213
214     twi->bytesReceived++;
215
216     /* If application signalling need to abort (error
          occurred),
217      * complete transaction and wait for next START.
          Otherwise
218      * send ACK and wait for data interrupt.
219      */
220     if (twi->abort) {
221       twi->interface->SLAVE.CTRLB =
222                 TWI_SLAVE_CMD_COMPTRANS_gc;
223       TWI_SlaveTransactionFinished(
224                 twi, TWIS_RESULT_ABORTED);
225       twi->abort = false;
226     } else {
227       twi->interface->SLAVE.CTRLB =
228                 TWI_SLAVE_CMD_RESPONSE_gc;
229     }
230   }
231   /* If buffer overflow, send NACK and wait for next
        START. Set
232    * result buffer overflow.
233    */
234   else {
235     twi->interface->SLAVE.CTRLB =
236                 TWI_SLAVE_ACKACT_bm
237                 | TWI_SLAVE_CMD_COMPTRANS_gc;
238     TWI_SlaveTransactionFinished(
```

```
239                 twi, TWIS_RESULT_BUFFER_OVERFLOW);
240   }
241 }
242
243
244 /*  TWI slave write interrupt handler.
245  *
246  *  Handles TWI slave write transactions and responses.
247  *
248  *  twi:
249  *      The TWI_Slave_t struct instance.
250  */
251 void TWI_SlaveWriteHandler(TWI_Slave_t *twi)
252 {
253   /* If NACK, slave write transaction finished. */
254   if ((twi->bytesSent > 0) && (twi->interface->SLAVE.
      STATUS &
255                                TWI_SLAVE_RXACK_bm)) {
256     twi->interface->SLAVE.CTRLB =
         TWI_SLAVE_CMD_COMPTRANS_gc;
257     TWI_SlaveTransactionFinished(twi, TWIS_RESULT_OK);
258   }
259   /* If ACK, master expects more data. */
260   else {
261     if (twi->bytesSent < TWIS_SEND_BUFFER_SIZE) {
262       uint8_t data = twi->sendData[twi->bytesSent];
263       twi->interface->SLAVE.DATA = data;
264       twi->bytesSent++;
265
266       /* Send data, wait for data interrupt. */
267       twi->interface->SLAVE.CTRLB =
            TWI_SLAVE_CMD_RESPONSE_gc;
268     }
269     /* If buffer overflow. */
270     else {
271       twi->interface->SLAVE.CTRLB =
            TWI_SLAVE_CMD_COMPTRANS_gc;
272       TWI_SlaveTransactionFinished(twi,
            TWIS_RESULT_BUFFER_OVERFLOW);
273     }
274   }
275 }
276
277 /*  TWI transaction finished function.
278  *
279  *  Prepares module for new transaction.
280  *
281  *  twi:
282  *      The TWI_Slave_t struct instance.
```

```
283  *  result:
284  *      The result of the transaction.
285  */
286 void TWI_SlaveTransactionFinished(
287         TWI_Slave_t *twi, uint8_t result)
288 {
289   twi->result = result;
290   twi->status = TWIS_STATUS_READY;
291 }
292
293 /*  Telemetry-grabbing function.
294  *
295  *  Prepares and fetches sensor-data
296  *  for transmission via TWI.
297  *
298  *  OBS! Perhaps this method should read the
299  *  struct members of each sensor, in order to
300  *  reduce the process time spent in the ISR.
301  *
302  *  data:
303  *      The high- or low byte of a sensor.
304  *  info:
305  *      The return value converted to mV.
306  */
307 uint8_t GetTelemetry(char data) {
308     uint16_t BV_temp, BT_temp, SAV_temp, SAC_temp;
309     uint8_t info;
310
311     switch(data) {
312         case 0://BAT_VOLTAGE_HI_BYTE:
313             BV_temp =
314                 (BatteryVoltage())*ADC_TO_MILLIVOLTS;
315             info = ((BV_temp) >> 8);
316         break;
317
318         case 1://BAT_VOLTAGE_LO_BYTE:
319             info =
320                 (BatteryVoltage())*ADC_TO_MILLIVOLTS;
321         break;
322
323         case 2://BAT_TEMP_HI_BYTE:
324             BT_temp =
325                 (BatteryTemp())*ADC_TO_MILLIVOLTS;
326             info = ((BT_temp) >> 8);
327         break;
328
329         case 3://BAT_TEMP_LO_BYTE:
330             info =
331                 (BatteryTemp())*ADC_TO_MILLIVOLTS;
```

```
332         break;
333
334         case 4://SA_VOLTAGE_HI_BYTE:
335             SAV_temp =
336                 (SolarArrayNSVoltage())*
                       ADC_TO_MILLIVOLTS;
337             info =  ((SAV_temp) >> 8);
338         break;
339
340         case 5://SA_VOLTAGE_LO_BYTE:
341             info =
342                 (SolarArrayNSVoltage())*
                       ADC_TO_MILLIVOLTS;
343         break;
344
345         case 6://SA_CURRENT_HI_BYTE:
346             SAC_temp =
347                 (SolarArrayNSCurrent())*
                       ADC_TO_MILLIVOLTS;
348             info = ((SAC_temp) >> 8);
349         break;
350
351         case 7://SA_CURRENT_LO_BYTE:
352             info =
353                 (SolarArrayNSCurrent())*
                       ADC_TO_MILLIVOLTS;
354         break;
355
356         default:
357             info = 0;
358         break;
359     }
360     return(info);
361 }
```

Listing F.20: twislave.h

```
1  #ifndef TWI_DRIVER_H
2  #define TWI_DRIVER_H
3
4  #include "avr_compiler.h"
5
6  /* Buffer size defines. */
7  #define TWIS_RECEIVE_BUFFER_SIZE        8
8  #define TWIS_SEND_BUFFER_SIZE           8
9  #define TWIS_STATUS_READY               0
10
11 //! Slave address.
12 #define SLAVE_ADDRESS    0x49
```

```
13
14 /* Transaction status defines.*/
15 #define TWIS_STATUS_READY                0
16 #define TWIS_STATUS_BUSY                 1
17
18 #define ADC_TO_MILLIVOLTS               1.820
19
20 #define BAT_VOLTAGE_HI_BYTE             0x80
21 #define BAT_VOLTAGE_LO_BYTE             0x40
22 #define BAT_TEMP_HI_BYTE                0x20
23 #define BAT_TEMP_LO_BYTE                0x10
24 #define SA_VOLTAGE_HI_BYTE              0x08
25 #define SA_VOLTAGE_LO_BYTE              0x04
26 #define SA_CURRENT_HI_BYTE              0x02
27 #define SA_CURRENT_LO_BYTE              0x01
28
29 //************************************************
30 // TWI struct/enum declarations
31 //************************************************
32 /* Transaction result enumeration */
33 typedef enum TWIS_RESULT_enum {
34   TWIS_RESULT_UNKNOWN            = (0x00<<0),
35   TWIS_RESULT_OK                 = (0x01<<0),
36   TWIS_RESULT_BUFFER_OVERFLOW    = (0x02<<0),
37   TWIS_RESULT_TRANSMIT_COLLISION = (0x03<<0),
38   TWIS_RESULT_BUS_ERROR          = (0x04<<0),
39   TWIS_RESULT_FAIL               = (0x05<<0),
40   TWIS_RESULT_ABORTED            = (0x06<<0),
41 } TWIS_RESULT_t;
42
43 typedef struct TWI_Slave {
44     // Pointer to what interface to use
45     TWI_t *interface;
46     // Pointer to process data function
47     void (*Process_Data) (void);
48     // Read data
49     register8_t receivedData[TWIS_RECEIVE_BUFFER_SIZE];
50     // Data to write
51     register8_t sendData[TWIS_SEND_BUFFER_SIZE];
52   // Number of bytes received
53     register8_t bytesReceived;
54     // Number of bytes sent
55     register8_t bytesSent;
56   // Status of transaction
57     register8_t status;
58     // Result of transaction
59     register8_t result;
60   // Strobe to abort
61     bool abort;
```

```
62 } TWI_Slave_t;
63
64 //*************************************************
65 // Function prototypes
66 //*************************************************
67 void TWI_SlaveInitializeDriver(
68                 TWI_Slave_t *twi,
69                 TWI_t *module,
70                 void (*processDataFunction) (void));
71
72 void TWI_SlaveInitializeModule(
73                 TWI_Slave_t *twi,
74                 uint8_t address,
75                 TWI_SLAVE_INTLVL_t intLevel);
76
77 void TWI_SlaveInterruptHandler(TWI_Slave_t *twi);
78 void TWI_SlaveAddressMatchHandler(TWI_Slave_t *twi);
79 void TWI_SlaveStopHandler(TWI_Slave_t *twi);
80 void TWI_SlaveDataHandler(TWI_Slave_t *twi);
81 void TWI_SlaveReadHandler(TWI_Slave_t *twi);
82 void TWI_SlaveWriteHandler(TWI_Slave_t *twi);
83 void TWI_SlaveTransactionFinished(
84             TWI_Slave_t *twi, uint8_t result);
85 void TWID_SlaveProcessData(void);
86 uint8_t GetTelemetry(char data);
87
88 #endif /* TWI_DRIVER_H */
```