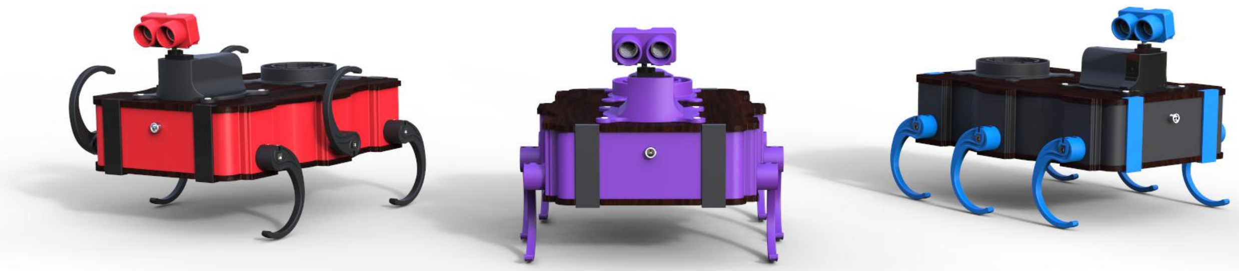
Development of Power System Control for the DeciZebro

Niels Fakkel (4483774)
Lennart de Jong (4440501)

Supervisors:
Chris Verhoeven (EWI, TuDelft)
Daniel Booms (The Zebro Project, TuDelft)

June 18, 2018



ABSTRACT

The development of swarm robotics strives for resilience, a key factor in this is energy supply. Without energy a robot swarm cannot operate, so a self sustaining power management system is of great importance. In this thesis, the design of the power management system control hardware and firmware for the new DeciZebro is discussed. The power flow between the battery and peripherals of the robot will be controlled with safety as a top priority.
The firmware of the power system reflects the thinking and being of a real animal in code. When the power system wants to recharge it will tell the brain of the robot to get back to a charging station. In case of an error, a pain signal in the form of an interrupt is send through the nerves of the robot.
The firmware and hardware design is done, and several parts of the system have been tested and verified. The prototype PCB is completed, however testing has not yet taken place at the current time.

Contents

1 Introduction

The Zebro project is a swarm robotics research group at the TU Delft. The idea was inspired by swarm behaviour exhibited in nature by animals. Each individual in the swarm follows the same set of simple rules and by doing that the swarm exhibits emergent behaviour. The swarm becomes something greater than the sum of its parts and is then able to accomplish more complex tasks. One of the most important requirements for the swarm to operate is resilience. Each member of the swarm should be expendable. To expand upon this principle the Zebro robot itself is also made to be resilient. This is done through modularity of each component; every module in the Zebro can be replaced through a basic connector interface and module can be independently developed as long as they fulfil the interface standards of the Zebro. This means the robots can be deployed on areas humans would not be able to survive or a difficult to access, such as disaster areas or even the moon.

One of the key factors in the resilience of the swarm is its energy supply. If the swarm is unable to acquire enough energy it will collapse as the Zebro's will cease operation. In this regard the Zebro can be seen as an electric animal; much like in nature when an animal is unable to supply itself it dies. Previous design of the Zebro power management system lacked this sustainability and consisted of a laptop charger interface and a battery. This allowed the a single DeciZebro to be operational for a few hours before human intervention is required to connect the laptop charger.

Different Zebro robots exist in different sizes and the goal of our group is to design a new power management system for the DeciZebro that allows for it to operate autonomously and thus remove the need for human intervention. This will allow the swarm in theory to remain operational indefinitely. To accomplish this the charging capabilities have to be expanded by adding a solar charging module as well as an inductive charging module. The project is commissioned by the Zebro team of TU delft, led by Mattijs Otten and supervised by dr.ir. Chris Verhoeven and Daniël Booms. The project has been divided as:

- The peripheral part manages the three input sources.
- The power electronics part develops the battery management system as well as the output converters.
- the control electronics develops both the necessary hardware and firmware to manage the system and serve as communication interface.

1.1. Problem definition

The topic of this thesis is the control electronics part of the power management system. The control electronics have to monitor the state of the system at any given moment and make decisions that ensure the safe operation of the Zebro. It has to function as the interface between the power management system and the main Zebro controller. Our final product consists of the following deliverables:

Power management system firmware Due to the modular nature of the Zebro the power management system has to make decisions autonomously based on its own inputs. Its power flow must be managed by the firmware. The firmware should be able to communicate with the main Zebro controller according to the ZebroBus protocol. The firmware for the microcontroller is written in C.

Control system hardware In order to achieve the desired operation of the system, the hardware implementation has to be provided. The hardware implementation is fully integrated into the power management system's PCB and will be designed in KiCad.

1.2. STATE OF THE ART

Power management systems of this scale are common in CubeSats. Their Electrical Power System (EPS) makes use of solar panels as input and a battery for storage. They are often controlled by a dedicated microcontroller unit (MCU) that regulates the power flow and monitors the state of the system using sensors. [1]

The CubeSTAR EPS uses the ATxmega128A1U as dedicated MCU that communicates via inter-integrated circuit (I^2C) bus with the other subsystems. For its internal communication it uses a second I^2C bus to read out temperature, voltage and current of the system as well as State of Charge of the battery. The MCU does not handle the maximum power point tracking (MPPT) algorithm, instead it uses a dedicated IC to achieve this. It uses a buck-converter to power the internal sensor system at 3.3V. Current limiting switches are used to enable and disable the different outputs to subsystems as well as close off the input from the solar panels. [2]

The Nanosatellite CubeStar uses a finite state machine as main loop for the power system control. The system has two charging states for the constant current and constant voltage part of Li-ion charging, it has a separate disconnect state in case of emergency. The internal system is powered with the use of a LDO. The MCU handles the MPPT algorithm for the solar panels through pulse width modulation (PWM) of a synchronous buck converter. [3]

Architectures of other robot power supplies can be considered as well. In [4] a power management system structure of a service robot is proposed, it can select either an adapter, a wireless power receiver or a battery pack as input. The power is then converted to the different outputs using a buck-boost converter for the motors and a buck converter and LDO for the control systems. The power system however lacks a dedicated control unit.

While most power systems are larger than the one developed for DeciZebro, much of the same principles apply. This paper describes the development and implementation of a DC microgrid control system with the use of a finite state machine. A two layer state machine is designed, the first layer to handle major states such as emergencies or shutdown, the second layer is used to nuance the system and handle the different scenarios. [5]

1.3. THESIS LAYOUT

The thesis is structured as follows:

- Chapter 2 describes the Programme of Requirements divided into functional and system requirements.
- Chapter 3 shows an overview the top-level design of the system together with the hardware interfaces.
- Chapter 4 discusses the hardware design choices made for the control electronics.
- Chapter 5 describes the firmware design for the power management system.

- Chapter 6 describes the testing and verification of different components of the system.
- Chapter 7 discusses the results of the testing performed on the various components
- Chapter 8 provides a conclusion on the current system and recommendations.

2 Programme of Requirements

The project has a number of requirements set by the proposer and with further talks these have been refined in order to draw the Programme of Requirements (PoR). The main goal of the project is to design a power management and battery management module for the new DeciZebro line. The design should be implemented on a PCB allowing it to be integrated directly on the module slot of the robot. The power system should support solar charging and inductive charging aside from a laptop charger. This subgroup is responsible for the control of the system, so mainly firmware and the hardware needed for this. The system has been divided into a power management system and a battery management system. This PoR will list the power system's functional requirements as well as the system requirements.

2.1. Functional Requirements

[FR-1] The entire system has to fit within the specified dimensions of the robot

[FR-2] Mechanical and software interfaces have to be well defined and comply with the current Zebro standards

[FR-3] Support 230VAC charging via laptop charging interface

[FR-4] Support inductive charging via the QI charging standard, featuring receiving coil(s) that are embedded at the bottom of Zebro

[FR-5] Support charging batteries via a PV module at the top of Zebro

[FR-6] Support directly supplying Zebro modules via the PV cell as well as via the battery

[FR-7] Utilize a damage protection system to avoid damage to any part of Zebro

[FR-8] The power supply should be able to deliver 16V - 1A continuously, with a peak of 3A, fused without voltage regulation

[FR-9] The power supply should be able to deliver 5V - 1A and 3.3V - 1A, both fused with voltage regulation

[FR-10] There should be an overcurrent protection module

[FR-11] The module must communicate with the DeciZebro main controller according to ZebroBus protocol [6]

[FR-12] The cost of the complete system must be below 100 EUR when mass produced

2.2. SYSTEM REQUIREMENTS

From the general functional requirements specific system requirements were distilled. Some requirements were added after the initial design through discussion with the other subgroups in order to ensure successful integration of the system:

[SR-1] Relevant system parameters must be monitored

[SR-2] Detect overvoltage, overcurrent and undervoltage faults and recover from these faults if possible

[SR-3] The battery management system is to be controlled and read out by the microcontroller

[SR-4] Temperature must be monitored, at T_{high} a warning signal must be given, at T_{max} the Zebro must shut down

[SR-5] The battery must be able to charge while the On/Off switch is Off

[SR-6] Expose system parameters to the main ZebroBus

[SR-7] Error count must be logged with the most recent error code being stored

[SR-8] In case of a critical error or shutdown an interrupt must be generated on the ZebroBus

[SR-9] The system must have a debug facility

[SR-10] Implement a maximum power point tracking (MPPT) algorithm for the Solar converter

[SR-11] Control the digital potentiometer of the laptop/QI converter

3
System design

While the project consists of three subgroups the complete system is still full of dependencies. This chapter will give an overview of the complete system as well as discuss the external interfaces.

3.1. System overview

Power enters the system from one or two of the three power sources. The laptop charger and QI input are connected to a power path selector after which one of the two is connected to the DC-DC converter. This fulfils requirements **[FR-3]**, **[FR-4]** and **[FR-5]**. The converter reference voltage can be adjusted to match the voltage of the powerbus. The powerbus voltage is determined by the batteries and is between 12V while fully discharged and 16.8V while fully charged. The solar panel is connected to a separate DC-DC converter that tracks the maximum power point (MPP) of the panel. This setup allows for both the solar panel and either the laptop charger or QI to deliver power simultaneously. Each input can be enabled or disabled from the centralised control unit. The reasoning for this is to ensure that in case of a faulty input it can be locally disabled. As protective measure, two fuses have been placed at the input, both before the respective DC-DC converter. This acts as failsafe for when an internal component fails such as a capacitor that turns into a short circuit.

The power then either flows to the battery management system or the outputs depending the load connected at the output. The external battery management system detects the load on the batteries and autonomously determines whether or not the batteries should charge or discharge, this information is then communicated to the MCU. Directly connected to the powerbus is the internal power supply of the control system. The On/Off switch is connected in series with the powerbus, ensuring the outputs are safe while still allowing the batteries to be charged.

The output converters set the voltage to the required levels and each of the three can be independently switched off to allow for more resilience in the system. The output voltages and currents are also monitored and the 16V bus features overcurrent protection using an analog comparator as required by **[FR-10]**. The BMS connection as well as every input and output features a TVS diode to protect the system against electrostatic discharge. The outputs all have non-resettable fuses as hardware protection to comply with requirement **[FR-7]**.

A block diagram of the complete system is shown in Figure 3.1. The hardware design will be discussed more in depth in Chapter 4.

3.1.1. External interfaces

The final system is implemented on a PCB and must conform to requirement **[FR-1]**. To comply with **[FR-2]** the external hardware interfaces have to be well defined and while some of the connectors and interfaces are predefined, others have been selected. This paragraph will give an overview of the con-

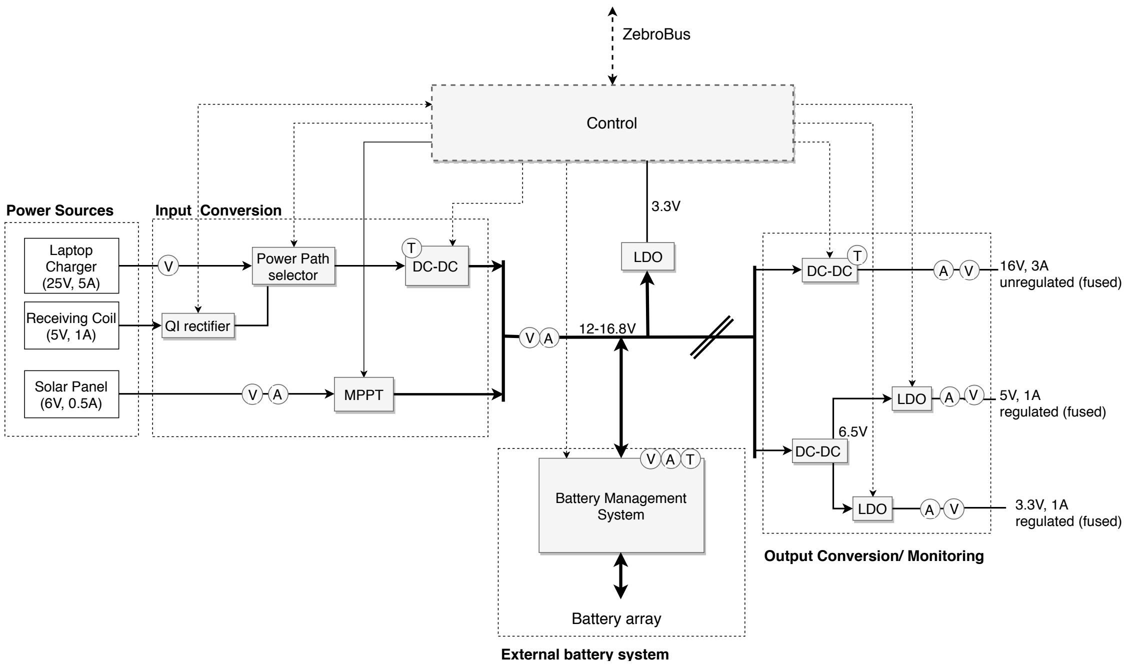


Figure 3.1: Overview of the complete power management system. The V, A and T represent voltage, current and temperature monitoring respectively.

nector interfaces of the power management system. Figure 3.2 shows the relevant interfaces for the control system.

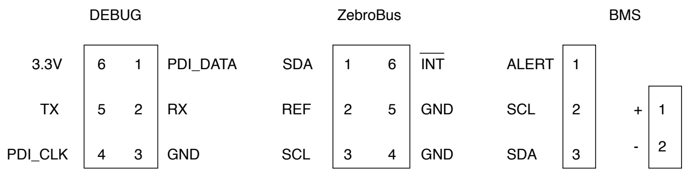


Figure 3.2: Debug, ZebroBus and BMS hardware interfaces

ZebroBus The connector interface of the ZebroBus has been specified in the design document listed on the Zebro Github repository. It is a 6 pin 2.54mm connector that connects the ZebroBus Dataline, Clockline, Bus supply voltage as well as the Interrupt pin and Ground.

Debug interface The debug interface consists of 2x3 2.54mm pin header. It is a Programming and Debug interface and can also support UART.

BMS interface The battery management system interface consists of a 3 pin 2.56mm header for data signals and a two pin 3.96mm JST connector for the power lines.

On/off switch The on/off switch has been specified in the requirements and is a DPST switch. The PCB connector used for this will be the 4 pin Molex KK 396 rated at 7A. The switch is mounted on the frame of the DeciZebro and as such a wire needs to run from the PCB to the switch and back. In order to reduce the electromagnetic interference caused by these wires they will be twisted. [7]

Solar input The solar panel itself will be mounted on the allocated space on top of the DeciZebro, the panel will then be connected to the PCB using a 2 pin Molex KK 254. Each connector pin is rated at 4A.

QI input The receiving coil will be mounted in the allocated space inside the DeciZebro, the coil will then be connected to the PCB using a 2 pin Molex KK 254.

Laptop charger input The laptop charger connector has been specified and will be connected to PCB using a 2 pin Phoenix Screw terminal.

Output All three outputs use the standard 4 pin Molex KK 254 according to the DeciZebro interface standard. The 16v output fans out over 4 of these connectors while the other two fan out over 2.

LEDs To help with debugging and gives visual confirmation of the system, LED's are positioned at the inputs and outputs for the system indicating their status.

4 HARDWARE DESIGN

Hardware is an essential part of the design as physical change has to be possible to control the system. In this chapter the requirements for the hardware are listed followed by the design and choices made during this.

4.1. REQUIREMENTS

The requirements for the hardware of the control system are not only determined by the Programme of Requirements but also established through communication with the other subgroups. The main requirements for the hardware design are:

- Supply power to the internal system from the powerbus
- Allow for communication to exist with the Zebro main controller
- Ability to monitor the voltage, temperature and current in the system

4.2. DESIGN

In order to fulfil the hardware requirements, the design is split into different components. These components consist of:

- The internal power supply
- Temperature sensing
- Current/Voltage sensing
- Reverse polarity protection
- The main control unit

The schematics were made in KiCad to allow for easier design of the PCB. The complete schematics are listed in Appendix B.

4.2.1. INTERNAL POWER SUPPLY

Just like the outputs, the control electronics require power. These components however do not operate at the voltage level present in the system and because of to this a solution has to be found. To find the correct location for the internal power supply, the requirements have to be considered. When the switch is off the outputs of the system should be safe but still allow for charging of the batteries [SR-5]. To ensure this the internal power supply has to be placed between the batteries and the switch.

There are different ways to reduce a voltage in a DC system, common methods include a buck converter used as in [2] or a low-dropout regulator (LDO). A switching converter has the advantage of having a higher efficiency and therefore also dissipates less heat, it does however require more components and therefore space which is limited. It also introduces switching noise to the internal power system which is undesirable. LDO's do not have a switching component and thus do not cause switching noise, they also also benefit from being smaller in size and are less complex. Their topology differs from other voltage regulators as they use open drain or open collector topology as opposed to an emitter follower one. This allows the LDO to bring the transistor into saturation allowing the voltage drop to reach the saturation voltage of the transistor. [8] However the voltage drop is still achieved by dissipating power and thus generating heat.

The power use by the internal system is quite limited and because of that the LM3480-3.3 LDO from Texas Instruments was selected. It converts the bus voltage to 3.3V and can deliver a maximum current of 100mA. While the LDO itself does not cause switching noise or transients the power bus itself does contain them due to the input converters and because of that additional filtering is desired. It was therefore decided to place an electromagnetic interference (EMI) filter at the input of the LDO in order to suppress these transients as shown in Figure 4.1. The output of the LDO also has two decoupling capacitors.

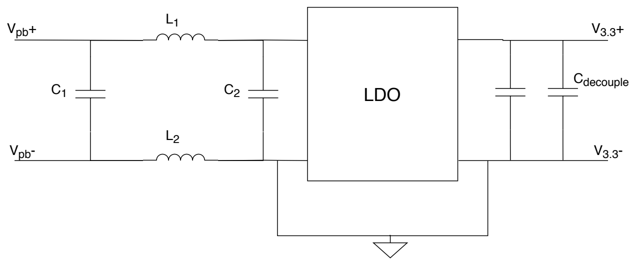


Figure 4.1: The conversion from the power bus to the internal supply using a LDO preceded by an EMI filter.

EMI FILTER

The input Cúk converter produces unwanted electromagnetic interference (EMI) due to its switching nature. This unwanted EMI is conducted through the power lines to the powerbus and to prevent it from disturbing the internal power supply it is filtered. The switching frequency of the laptop charger/QI converter is 500 kHz and in order to filter it the cut-off frequency of the EMI filter must be sufficiently low. The transfer function of the EMI filter shown in Figure 4.2 can be calculated as:

$$\frac{V_O}{V_I} = \frac{X_c}{\sqrt{X_L^2 + X_c^2}} \tag{4.1}$$

Solving for the cut-off frequency gives:

$$f_0 = \frac{1}{2\pi\sqrt{L_1 + L_2 + C_2}} \tag{4.2}$$

The values chosen for the converter were $L_1 = L_2 = 270nH$ and $C_1 = C_2 = 4.7uF$. This results in a cut-off frequency of 100 kHz which is sufficient, opting for a lower cut-off frequency would require larger components.

4.2.2. TEMPERATURE MONITORING

The temperature inside the Zebro is an important parameter. There are strict limits for the Li-ion battery operating range and because of that it is crucial to monitor the system in order to detect if temperature thresholds are reached.[9] Different choices exist to measure the temperature, the most common choices are a NTC thermistor or a silicon bandgap temperature sensor.

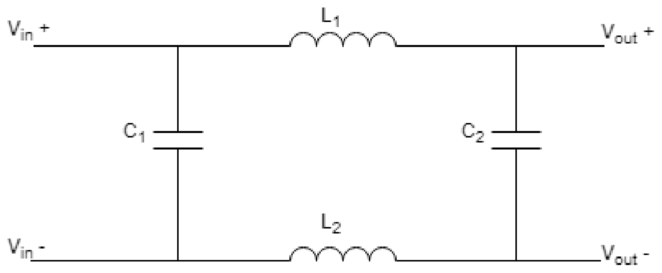


Figure 4.2: Differential mode EMI filter

A negative temperature coefficient thermistor has, as a first order approximation, a temperature that depends linearly on the resistance. Using a voltage division and an ADC the temperature can be computed. This would require either an external ADC or the microcontroller to perform the conversion. Another downside is that the NTC cannot be turned off within the circuit and thus consumes power. The silicon bandgap temperature sensor is based on the fact that the forward voltage of a silicon diode, such as the base-emitter junction of a bipolar junction transistor depends on the temperature. Using a Brokaw bandgap reference the base-emitter voltage can be measured and from there the temperature can be derived. [10] The benefit of the bandgap sensor is that it is already integrated in commercial and industrial digital sensors that are available on the market and will therefore not require additional hardware to function.

Table 4.1 shows the considered temperature sensors. The LM75B digital temperature sensor was selected based on its lower operating current and price compared to the other sensors. Its accuracy over a broad range is acceptable as the temperature thresholds are not precisely defined. [11]

Table 4.1: Trade-off table for temperature sensor selection.

	NCT75	LM75B	SI7060	TMP175
Accuracy	+/- 1 C	+/- 2 C	+/- 1 C	+/- 2 C
Price	€0,50	€0,47	€0,37	€2,18
Resolution	12 bit	11 bit	7 bit	12 bit
Operating current	800 uA	100 uA	500 nA	50 uA
Range (at specified accuracy)	0 - +70 C	-25 - +100 C	0 - +70 C	-40 - +125 C

The sensors are positioned near the input Cúk converter and the output Buck-boost converter to monitor the temperature at locations where the largest power conversions take place. The sensors will be connected via the Two-Wire interface used as internal communication. The complete application schematic is shown in Figure 4.3. The battery management system has the possibility of connecting an external thermistor able to be placed directly on the battery pack.

4.2.3. Current and Voltage monitoring

Monitoring the voltage and current within a system has numerous advantages as it provides valuable information about the state of the system as well as the power draw of different output modules. It will also help pick up unwanted behaviour that is present in the system.

There are several ways to measure current based on different physical principles: directly by applying Ohm's law or indirectly based on Faraday's law of induction. The direct approach involves connecting a resistor in series with the load and measuring the voltage drop over the resistor due to the current. This type of measurement is invasive as it is electrically connected to the system that is being monitored, it is however quite accurate even at lower currents. [12]

The indirect way of current sensing is realized through the hall-effect. The size of a magnetic field is

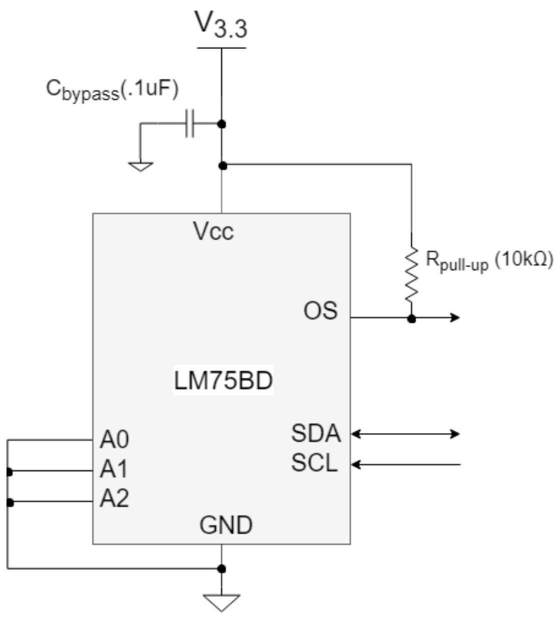


Figure 4.3: Application of the temperature sensor.

proportional to current and as such allows for the current to be indirectly measured by looking at the magnetic field strength. However this method requires a relatively high current to be present in order to achieve the same level of accuracy as shunt measurements. [12] On top of this, hall-effect sensors are more expensive than their shunt counterpart and as such it was decided to use the shunt measurement technique. To accomplish this the INA226 from Texas Instruments was selected.

Two setups exist for shunt sensing, high side and low side, illustrated in Figure 4.4. Low side sensing places the shunt resistor between ground and the load resulting in a low common-mode voltage. This method puts less requirements on the amplifier used but does bring disadvantages. As the current through the load and shunt changes so does the the voltage across the shunt, this results in a ground potential that can differ from that of the source affecting other measurements. It can also allow for short circuits on the load to remain undetected. On the other hand placing the shunt at the high side brings a high common-mode voltage that the amplifier must be able to handle, however it does remove the ground disturbance and is able to detect these short circuits. It was therefore decided to use the high side measurement.

The INA226 is an integrated current and voltage monitor with I^{2}C- and SMBus-compatible interface. It can handle a common-mode voltage of up to 36V and operates at a supply voltage between 2.7V and 5.5V. The internal ADC is 12 bits and has a maximum range of 81.92mV. The sensor can average over a number of measurements which allows for random peaks and other disturbances to be suppressed. The input range of the ADC sets the requirement for selecting the shunt resistor.

$$R_{shunt} = \frac{81.92mV}{I_{shunt}} \tag{4.3}$$

The different points in the system handle different currents and as such so will the shunt resistor. For the power bus the maximum current is set at 5A, this results in a shunt value of 16mΩ. There is however more to the choice of the shunt resistor. Power dissipation also has to be taken into account; a larger shunt resistance offers a higher resolution due to the larger voltage drop but at the cost of more power dissipated across it. More power also means that a larger shunt resistor is required. It was therefore chosen to use a value 5mΩ resulting in a maximum dissipation of $5^2 * 5m = 125mW$. The other shunt values were calculated the same way and are listed in Table 4.2.

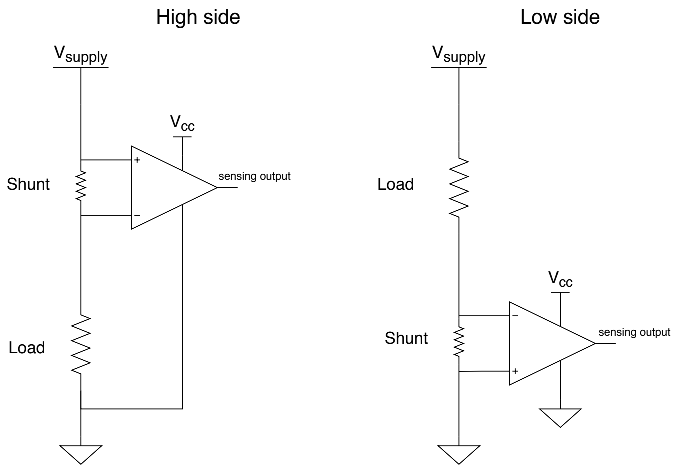


Figure 4.4: High side and low side shunt measurement setups.

	R_{shunt} [Ω]
PV input	100m
Power bus	5m
5v output	50m
3.3v output	50m

Table 4.2: Shunt resistor values

The differential measurement is very susceptible to noise due to low voltages across shunt. The sensor offers internal filtering by averaging over a number of measurements. This helps remove peaks and takes the burden off the MCU. However it is also recommended to use a differential low pass filter when transients are present at the sampling rate frequency (500kHz) or close to sampling rate harmonics. Since the switching frequency of the input Cúk converter matches the sampling rate it was decided to use a differential input filter to suppress the transients. The filter is shown in Figure 4.5.

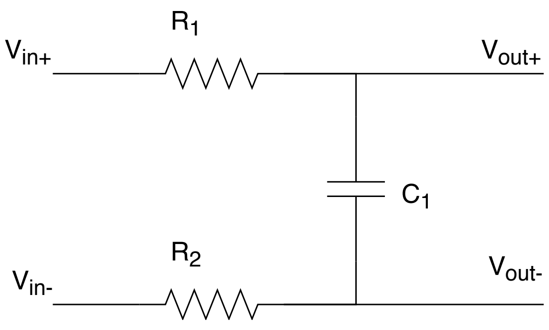


Figure 4.5: Input filter for the shunt measurement

The transfer function can be calculated from the topology as:

$$\frac{V_o}{V_i} = \frac{X_c}{\sqrt{R^2 + X_c^2}} \tag{4.4}$$

This results in a cut-off frequency of:

$$f_0 = \frac{1}{2\pi(R_1 + R_2 + C_1)} \tag{4.5}$$

A low value resistor was recommended by the manufacturer and therefore a 10Ω was used, the capacitor value was chosen at 100nF resulting a cut-off frequency of:

$$f_0 = \frac{1}{2\pi(10+10+100*10^{-9})} = 79.6kHz \tag{4.6}$$

It was decided not to use the differential low-pass filter at the PV input sensor and the 5- and 3.3V output sensors as no transients around the vulnerable frequency range are present; the solar panel provides no switching noise and the output LDO's act as a buffer between the switching buck-converter and the sensors. The complete topology is shown in Figure 4.6.

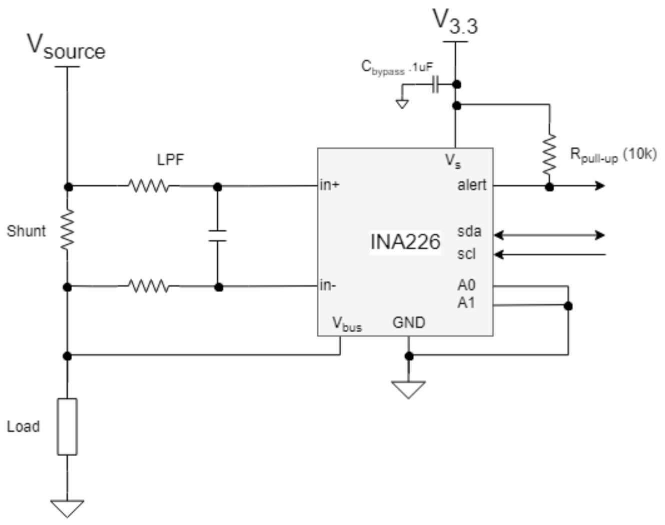


Figure 4.6: The INA226 applied topology.

Main bus overcurrent protection The INA226 provides telemetry and features an alert output for when an overcurrent/voltage condition is measured. However it only processes the measurements after the ADC conversion and as such has latency. In order to allow for faster response time in the case of an overcurrent event on the main bus an analog comparator is used. It is applied in the same manner as the INA226 as a high side shunt current sensor and has a differential low pass filter at the input to suppress the noise caused by the buck-boost converter. The comparator output is directly connected to the enable signal of the buck-boost and thus allows for immediate shutdown during an overcurrent event. Controlling the output is done by using the enable signal of the INA300 which is connected to the MCU. When enabled the output alert will be default high thus enabling the 16V output. The connection is shown in Figure 4.7.

Analog voltage measurement A basic method of measuring the voltage at a point in the system is by using a voltage division to reduce voltage to a valid range for an ADC. This method lacks the accuracy provided by the sensor but does allow for checking ranges. It is used at the laptop charger input to detect the voltage for the acceptable range of 5-22V for the Cúk converter set by the peripheral subgroup. Three more analog voltage measurements are done at the output to detect if a fuse has been blown, for in that case the voltage measured will drop to zero.

4.2.4. REVERSE POLARITY PROTECTION

The system needs to be foolproof and therefore also account for user error in connecting the system to external sources. Measure has to be taken to prevent reverse polarity. In order to achieve this a blocking diode is used in systems. [2] However the voltage drop across a silicon diode is typically around 0.7V and when placed on the powerbus will dissipate 1W or more continuously. Luckily an alternative solution exists in MOSFETs. A P-channel MOSFET's drain-source resistance while conducting is extremely low,

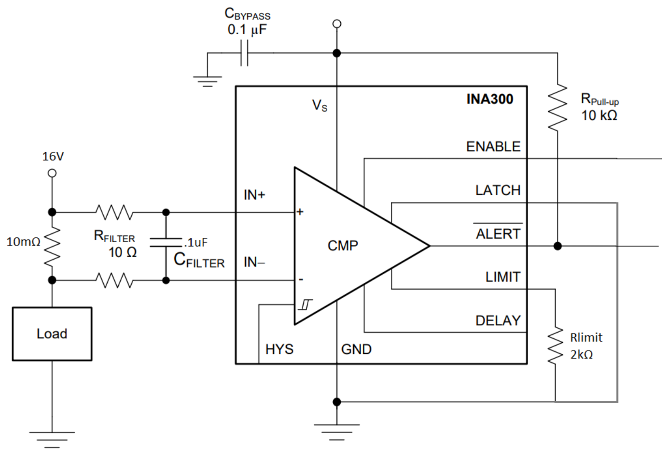


Figure 4.7: Application of the INA300 as output comparator

in the order of mΩs, resulting in a minimal voltage drop The gate of the MOSFET is connected directly to the negative voltage line as shown in Figure 4.8.

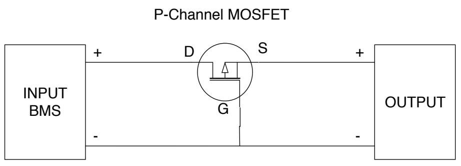


Figure 4.8: Reverse polarity MOSFET.

When the input and output are connected correctly V_D will be 12-16.8V and V_G as well as V_S remain 0V. Current then has to flow through the body diode in order to raise V_S so that to the point where V_{GS} drops below the threshold voltage of -3V at which point the MOSFET turns on and the voltage between V_D and V_S becomes near equal. When reverse biased the body diode does not conduct and therefore V_{GS} remains positive resulting it never turning on.

4.3. MAIN CONTROL UNIT

The ATxmega32A4U-AU was selected as MCU for the power management system. There are a number of reasons for this. The MCU has to be able to actively communicate on two busses at once, the ZebroBus and the internal bus. This requirement alone limits the number of available microcontrollers. It also needs to have enough I/O pins to handle all the control signals. It is low power with a supply voltage that is the same as that of the sensors: 3.3V. It is cheap and has heritage within the Zebro project. The AU version was selected over the MH version as the MH version is a ball grid array chip which greatly increases the difficulty of PCB routing while saving minimal surface area.

It is an 8-bit microcontroller that can operate at a clock frequency of 32MHz but can be scaled down. It has 34 programmable I/O pins, an internal ADC as well as a PWM generator. [13]

4.3.1. BACKUP POWER SUPPLY

The ZebroBus protocol dictates that in the case of an emergency the Zebro interrupt should be raised to give a warning to the main controller of the robot. This should allow for critical data to be saved and appropriate action taken. To accomplish this a back-up capacitor is placed near the MCU power supply. This will provide power in the case of sudden disconnection of the battery management system or failure of the internal power supply. In order to prevent the capacitor current from supplying the internal 3.3V a schottky diode is used. A schottky diode offers a lower forward voltage drop compared to a silicon diode and offers a faster switching time. An equivalent circuit of the backup power is shown in figure 4.9.

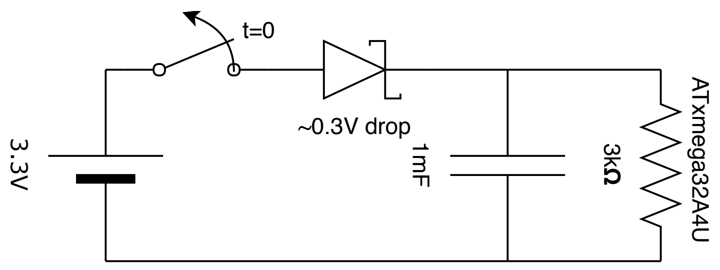


Figure 4.9: Backup power equivalent circuit

A back on the envelope calculation has been done to determine how long the capacitor can supply backup power. In normal conditions the switch is closed, in this case there will be approximately 3V over the capacitor and microcontroller, as the schottky diode has a voltage drop. The microcontroller is represented as a 3k resistor, because it consumes about 1mA at 3V 2MHz according to [14]. The capacitor is fully charged.

Once the switch opens at t=0, there is still 3V over the capacitor. However there is no current coming from the source anymore, so all current will be drawn by the microcontroller. So now there is a resistor in series with the capacitor, which means the standard discharge formula can be used:

$$V_c(t) = V_0 \cdot e^{-\frac{t}{RC}} \tag{4.7}$$

The result is plotted in figure 4.10. The microcontroller's minimum operating voltage at 2MHz is 1.6V [14]. So doing a raw calculation this means it has 1.886 seconds of backup power to store the critical data.

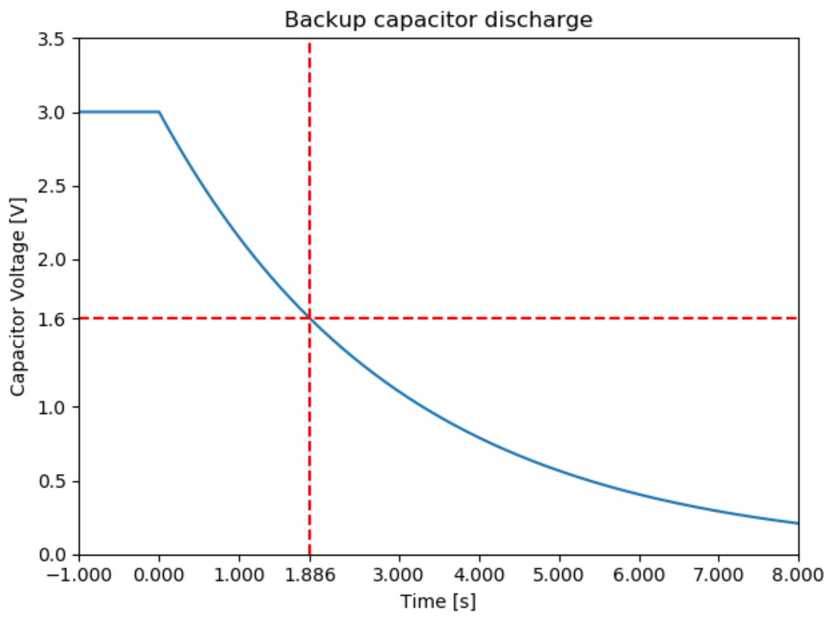


Figure 4.10: Backup power capacitor discharge prediction

4.3.2. ANALOG SUPPLY FILTER

The clock switching of MCU causes switching noise in the digital supply voltage. This noise can transfer to the analog supply rail and influence the ADC measurements. Since the noise is present at a relatively high frequency, a few to tens of MHz, a ferrite bead is used as a filter instead of an inductor. A ferrite bead is preferred since it offers a superior real impedance at high impedances compared to an inductor.

4.3.3. ANALOG INPUT PROTECTION

The pins of the MCU can handle a maximum voltage of $V_{cc} + 0.3$ and to make sure this will always be the case a Zener diode is placed in parallel to the measurement point and ground. The Zener diode has a low forward voltage and when exceeded it will conduct, thus regulating the voltage at the specified forward voltage, in this case selected to be 3.3V.

4.3.4. ZEBROBUS

For the ZebroBus interface a Buffer and interrupt had to be implemented both are shown in figure 4.11.

Buffer The requirements state the the module's internal state is not allowed to interfere with the ZebroBus [FR-12]. To accomplish this a buffer is implemented that separates the supply voltages of the module's bus connection and the external bus.

Interrupt The ZebroBus protocol also states a special interrupt signal that is default high and should be grounded in case of emergency. This is implemented with the use of a simple NMOS. The pullup supply for this interrupt line is external.

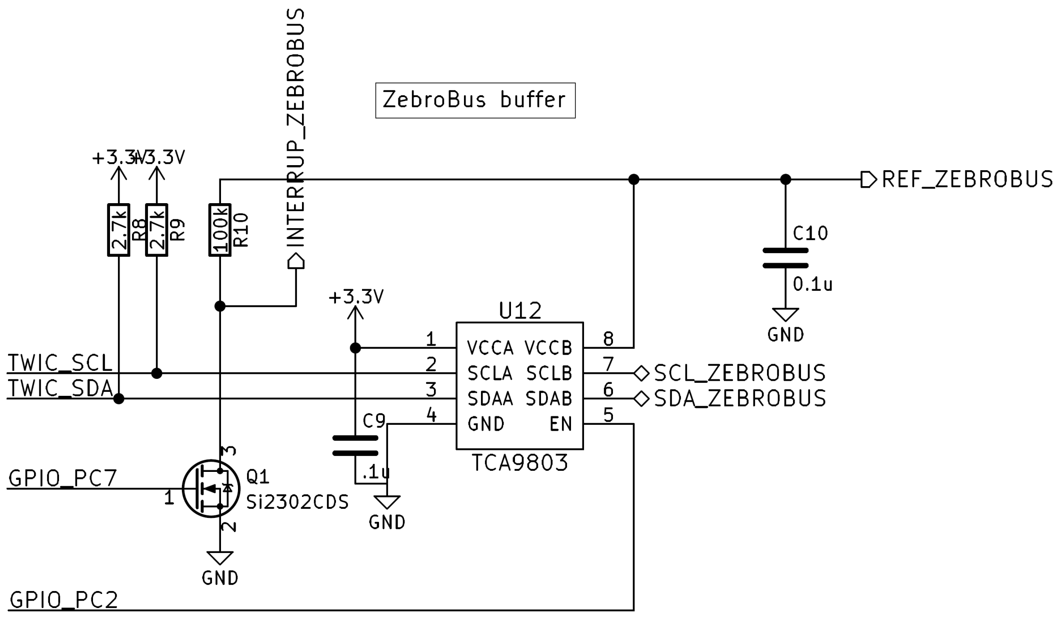


Figure 4.11: ZebroBus Buffer and Interrupt

5 FIRMWARE DESIGN

5.1. REQUIREMENTS

The firmware ties the whole system together and because of that has to fulfil an array of tasks.

- Communicate with the main ZebroBus controller (PoR **[FR-12]**)
- Initialise the complete module on startup
- Monitor the system parameters (PoR **[SR-1]**)
- Generate the PWM signal for MPPT of the solar panel
- Recover from errors autonomously (PoR **[SR-2]**)
- Determine the correct power scheme
- Implement a MPPT algorithm for the Solar converter
- Control the digital potentiometer of the laptop and QI charger

5.2. TOP-LEVEL FIRMWARE

The top-level firmware routine is shown in Figure 5.1. During startup the MCU initialises itself as well as its subsystems. As it enters the main loop it proceeds to perform a routine sensor check in order to monitor the state of the system. The watchdog timer is reset and the power scheme is then determined with the help of the sensor data. The power scheme is implemented as a Finite State Machine (FSM), see figure 5.3. After the current state of the power system is determined the system will update its virtual registers and increment the loop counter to provide fresh data for the ZebroBus. Interrupts are enabled except during execution of crucial commands. The interrupts can originate from the ZebroBus, the internal bus and from within the MCU itself.

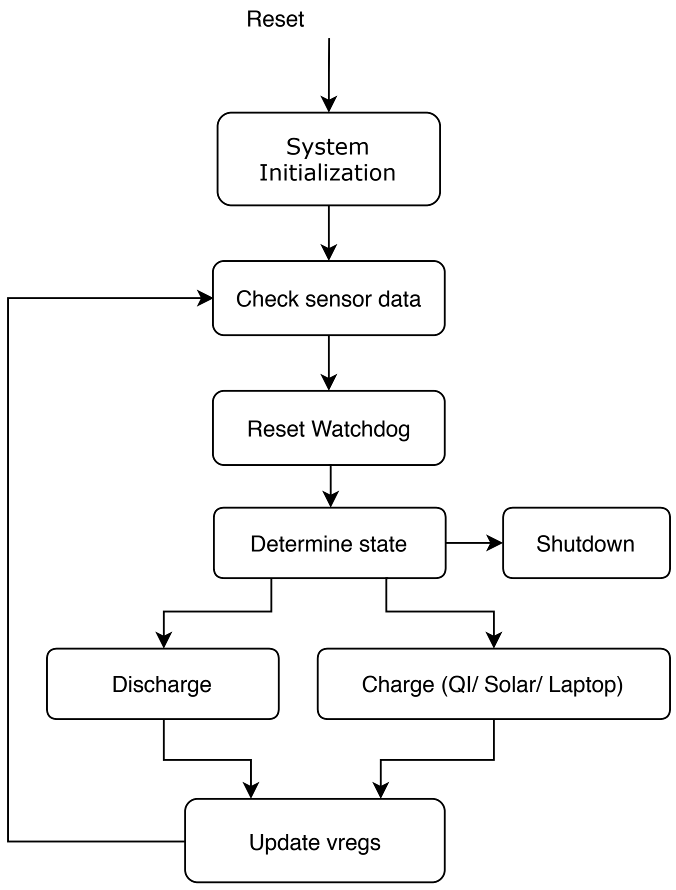


Figure 5.1: Flowchart of the top-level firmware architecture

5.2.1. SYSTEM INITIALISATION

During startup the MCU will first configure the clock followed by initialising the various internal functions such as the virtual registers. The FSM is initialised and set to the start state with no inputs or outputs enabled. This will ensure that minimum power flows through the system before its parameters are known. After this it will proceed to initialise the I/O pins of the system and configure the sensors.

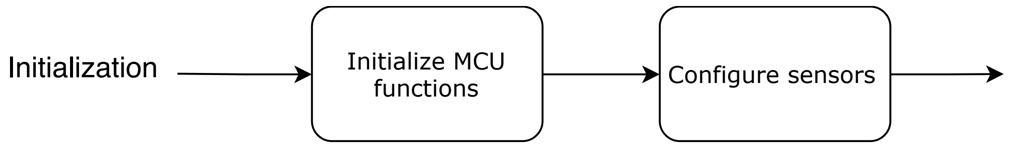


Figure 5.2: Initialisation procedure of the system.

5.2.2. RETRIEVING SENSOR DATA

Most of the monitoring in system is done via the internal data bus which will be described in-depth in Section 5.4. The sensor data also includes analog signals originating from the laptop charger input and the three output fuses. These signals are converted by the internal ADC of the microcontroller before being stored in the virtual registers. The implementation is described in 5.3.3.

Sensor Data Routine All sensor data coming into the microcontroller has to be checked, for both determining the next FSM state and doing error handling. A check function for each sensor has been written in which the data is compared to the predetermined values. Based on the comparison an error could be reported, see table 5.5. After that the value of the sensor data is written into the right virtual register, table 5.3.

For example in the case of the temperature sensors. If the temperature of the input is larger than 45^oC then the OTI should trigger, because this is the maximum charging temperature. The output over-temperature error is triggered once the output temperature is above 60^oC. This comparison is implemented into the check function. The function includes a switch statement, which based on what address is read out determines what data to compare and store in the right virtual registers, see the code implementation below:

```c
void Check_And_Write_Temperature_Registers(uint8_t adr, uint8_t high_byte, uint8_t low_byte)
{
  switch(adr) {
    case ADDRESS_TEMP_SENS_INPUT :
    if (low_byte > 0x2d)//If the input temperature is higher than 45 degrees Celsius trigger OTI
    {
      error_report(ERROR_OTI);
    }
    vregs_write(VREGS_INPUT_TEMP_LOW, low_byte);
    vregs_write(VREGS_INPUT_TEMP_HIGH, high_byte);
    break;
    case ADDRESS_TEMP_SENS_OUTPUT :
    if (low_byte > 0x41) //If the output temperature is higher than 60 degrees Celsius trigger OTO
    {
      error_report(ERROR_OTO);
    }
    vregs_write(VREGS_OUTPUT_TEMP_LOW, low_byte);
    vregs_write(VREGS_OUTPUT_TEMP_HIGH, high_byte);
    break;
    default :
    error_report(ERROR_UNKNOWN);
  }
}
```

5.2.3. DETERMINING POWER SCHEME

While the firmware runs in a loop, the input and output settings of the system will be determined by a Finite State Machine (FSM), this is done due to the limited amount of possible scenarios. A FSM is more efficient and reliable which is prefered in the case of controlling lithium-ion batteries[15]. Figure 5.3 shows an overview. After initialisation the system is in the start state after which it checks for the different transition conditions and selects the next state. The system is divided into three main operational states defined by the state of the connected power supplies, the batteries will either be charging or discharging depending on it.

In the start state all outputs and inputs are disabled, this is done to ensure safe startup by preventing power flow under unknown conditions. In every operational state the solar panel module will be enabled, it will always assist the current state of the system, it either helps with charging or it lessens the discharging rate of the batteries as stated in **[FR-6]**.

The charge state enables either the QI module or the laptop charger depending on the power path selector at the input. The outputs are enabled as well to allow the Zebro to stay operational while charging. This choice was made with the idea of the swarm in mind: when the Zebro has positioned itself on a charging pad and another Zebro that requires urgent power arrives at the station it can still move out of the way. Unique to this state is that the digital potentiometer is controlled to adjust the reference voltage of the laptop/QI input DC-DC converter based on the battery voltage. Solar charging on the other hand will disable the 16V output or it would never be able to charge. The solar panel delivers a tiny amount power, so it functions as a last resort for when the batteries are at a critically low voltage and no other charging method is present. To allow for more efficient charging the Zebro should ideally be put in a sleep state to minimize ambient power usage. The discharge state is the default operating state when all systems are nominal and the Zebro is out on the field: only the solar panel is supporting the batteries and all outputs are enabled. The shutdown state is entered when the temperature surpasses the maximum allowed value. It could also enter shutdown state in case a critical error occured

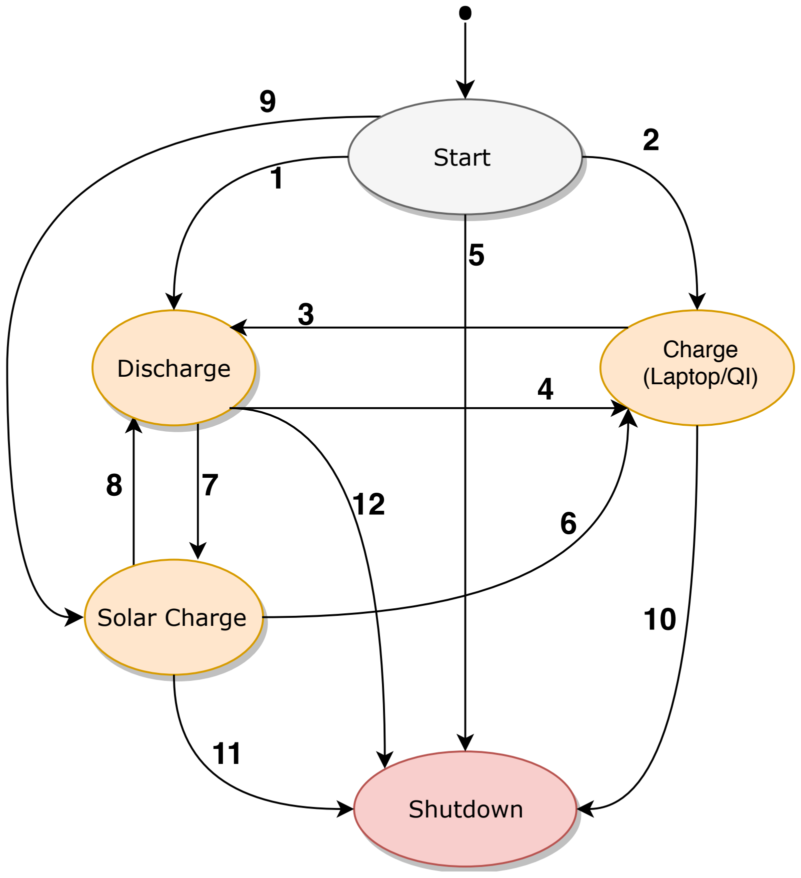


Figure 5.3: Top-level state machine for the system power scheme.

in the system concluding the operation. Termination is achieved by disabling the discharge NMOS on the BMS through its host controller. An overview of the I/O of the system is shown in Table 5.1.

The transition conditions from the starting state are defined as:

Transition	Transition condition	Transition description
1	$(\sim(V_{laptop} \vert V_{QI})\&(V_{bat} > V_{critical})\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	nominal operation, no charger present
2	$(((V_{laptop} \vert V_{QI}) == 1)\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	charger is connected
3	$(\ (V_{laptop} \vert V_{QI})\&(V_{bat} > V_{critical})\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	charger is disconnected
4	$((V_{laptop} \vert V_{QI} == 1)\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	charger is detected
5	$((T_{bat} > T_{max}) \vert (E_{crit})$	shutdown state
6	$((V_{laptop} \vert V_{QI}) == 1)\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	charger is detected
7	$((V_{bat} < V_{critical})\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	last resort solar charging
8	$((V_{bat} < V_{restore})\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	Acceptable SoC reached
9	$((V_{bat} < V_{critical})\&(T_{bat} < T_{max}))\& \sim(E_{crit})$	last resort solar charging
10	$(T_{bat} > T_{max}) \vert (E_{crit})$	shutdown state
11	$(T_{bat} > T_{max}) \vert (E_{crit})$	shutdown state
12	$(T_{bat} > T_{max}) \vert (E_{crit}r)$	shutdown state

Table 5.1: Transition conditions for the the state machine

Aside from critical errors, faults can occur in the system. These faults might be an overcurrent in a specific bus or overvoltage. To account for this every output has a health flag, good will result in the system turning on when set, bad will prevent the output from being enabled functioning as an authority above the state machine. The implementation is discussed further in section 5.5.

State	Laptopcharger	Solar charger	16V out	5V out	3.3V out
Start	disabled	disabled	disabled	disabled	disabled
Laptop/QI charging	enabled	enabled	enabled	enabled	enabled
Solar Charging	disabled	enabled	disabled	enabled	enabled
Discharge	disabled	enabled	enabled	enabled	enabled
Shutdown	disabled	disabled	disabled	disabled	disabled

Table 5.2: Overview of the power scheme for every state.

5.3. FUNCTIONS

5.3.1. WATCHDOG TIMER

Fault recovery is an important part of the system and a requirement **[SR-2]**. The firmware allows for the system to recover from hardware errors, however the firmware itself is also prone to malfunctioning due to the code or external causes. The DeciZebro swarm will operate autonomously and when a fatal error occurs the system must be able to recover or the Zebro will fail. In embedded software this error recovery is achieved through an external hardware timer also known as a watchdog timer. It runs independently from the CPU and will therefore not fail in case of an error within the code. To prevent the watchdog from resetting the MCU has to clear the timer before it expires. When the timer does reach its threshold it will log the event and reset the system, shown in Figure 5.4. The log serves as a counter to check if the system keeps failing. The log is also directly written to the loop counter virtual register, (0x12) in Table 5.3.

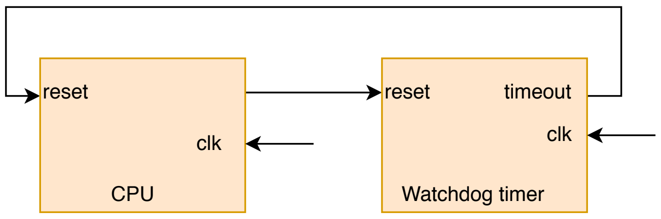


Figure 5.4: Watchdog configuration.

5.3.2. MPPT IMPLEMENTATION

The MPPT is realized together with the peripheral subgroup. An overview of the system is shown in Figure 5.5. The idea is to maximize power transfer by load matching using a DC-DC converter, in this case a Cúk converter. Different MPPT techniques exist such as Perturb and Observe (P&O), Incremental conductance and fuzzy logic design[16]. As the Zebro swarm moves around terrain the irradiance can vary, this combined with the simplicity and high efficiency is why the P&O algorithm was selected[17]. The algorithm measures the power coming from the solar panel and checks if it has increased compared to the previous step, if so it continues incrementing in that direction, if not it takes a step back. The algorithm climbs along the P-V curve and is therefore also known as a hill climbing algorithm. Once the maximum power point is reached the climbing doesn't stop and the algorithm oscillates around it[18].

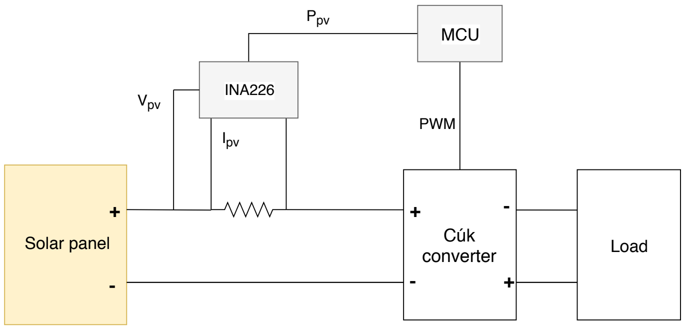


Figure 5.5: Overview of the solar power converter with MPPT.

Figure 5.5 shows the setup used to implement the MPPT, the INA226 will provide the current and voltage measurement. Moving along the P-V is done by changing the duty cycle of the PWM signal of the converter. The initial step is defined as right however this is arbitrary and serves only as initializaton. The implementation is quite straightforward. Memory is kept of the previous step direction and the power measurement is refreshed every loop by the sensor. A waveform generator is set on the compare match of the counter register and the compare value register. When matched the output will become 0. Using a peripheral clock of 2MHz a 12-bit resolution PWM can be generated at 8kHz, however this frequency can be made higher by using the 128MHz PLL oscillator.

```
 if (((power > PowerOld) & (LastStep == RIGHT)) | ((power < PowerOld) & (LastStep == LEFT)))
 {
   PWM_IncrementDutyCycle(1);
   TCC0.CCDBUF = DutyCycle;
   LastStep = RIGHT;
 }
 else
 {
   PWM_DecrementDutyCycle(1);
   TCC0.CCDBUF = DutyCycle;
   LastStep = LEFT;
 }
PowerOld = power;
```

5.3.3. ADC IMPLEMENTATION

The MCU has a built in 12-bit ADC with 4 conversion channels it allows for internal comparison and can generate an interrupt when a user defined threshold is reached. The maximum internal reference voltage is $V_{cc}/1.6$ resulting in a measurement range of roughly 0V to 2V. The selected values for the voltage division at the laptop charger result in a measured value of 2V at 25V and 0V at 0V resulting in a resolution of approximately 6mV/LSB, which is more than sufficient to detect whether or not it is valid. For the each of the fuses the voltage division results in a measurement of 2V when the specific output is enabled. The ADC channels measuring the fuse signals are only enabled when the relevant outputs are enabled and will generate an interrupt when the value drops below comparison threshold, chosen at 0.5V.

5.4. COMMUNICATION

In the power system there are multiple ways in which the micro controller is able to communicate. Figure 5.6 shows a hierarchical overview of the communication. Within the power system there is a Internal Bus, this bus is used to read out the sensors and battery system data. Outside there is the communication with the rest of the Zebro Robot, using the so called ZebroBus protocol [6], as defined in requirement **[FR-11]**. The main controller is on top as it controls all the systems in the Zebro, the power system controller is one layer below with the sensors and batterysystem on the internal bus. The power management system thus needs to communicate with two different busses, both will be discussed in this section.

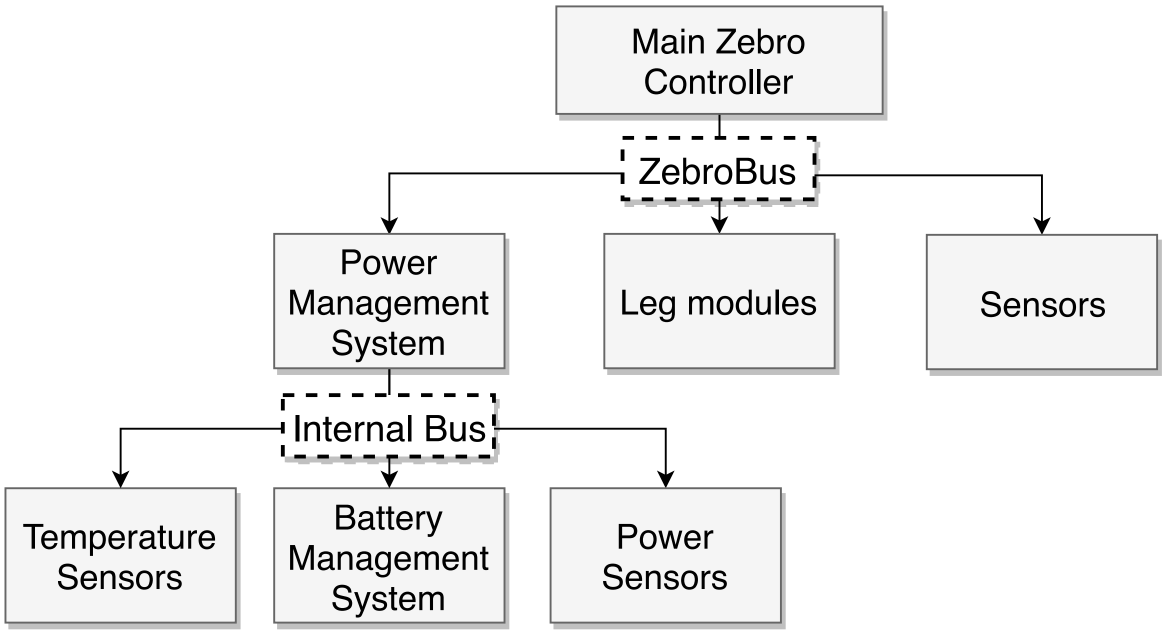


Figure 5.6: Bus Hierarchy Zebro

5.4.1. ZEBROBUS, NERVOUS SYSTEM

The definition of a nervous system is: An animal's or person's nervous system consists of its brain and all the nerves in its body that together make movement and feeling possible by sending messages around the body. For a robot a digital nervous system has to be made, in this case a digital communication protocol. The protocol is named ZebroBus and defined so that every part is connected to the brain the same way. Zebro should be aware of how long it can survive and when it got hurt. Critical situations in which fuses blow up or batteries run dry need to be communicated with the brain/main-controller of the robot. ZebroBus is a protocol based on I^2C, where one Master will send out the data it wants to receive or set, in this case the main controller. Also there are multiple Slaves which can be read to or written from. The power system will be a slave, so the slave version of the ZebroBus protocol will be applied. The most important information about the power system should be available for the ZebroBus Master to read. The information is readily updated and stored in Virtual Registers. These virtual registers are continuously reachable by the Master. Another addition to the ZebroBus communication is an Interrupt pin, this pin can be set high from any slave and is only used in extreme situations. One example is the situation in which the internal temperature of the robot is too high, creating a very big risk of the batteries exploding. Then, the Interrupt pin will be set high so the main controller knows it has to shut down before the power is cut off.

ZebroBus Address According to the ZebroBus protocol only the addresses 0x10 to 0x6F can be used for Slave devices. Multiple of these addresses have already been chosen. Therefore the power system address will be *0x60*.

Virtual Register The ZebroBus uses the same convention as normal I^2C protocols, where data is transferred by first transmitting the register address of the data the masters wants to read or write on the slave, followed by the actual data. All ZebroBus devices should have a set of Virtual Registers that comprise the following 6 rules defined[6].

1. The vregs are byte addressed
2. The vregs have 256 fields

3. When reading or writing more than one byte in a single I^{2}C transaction, the read or write register address should auto increment
4. After receiving the STOP bit, the read or write register address should be reset to the address that was last received.
5. The register address is shared for read and writes. i.e. the I^{2}C sequence
   START 0x31 0x42 0x43 STOP START READ_BYTE STOP
   should do the following:
   (a) Write 0x42 to register address 0x31
   (b) Write 0x43 to register address 0x32
   (c) Read the value in register address 0x31

Virtual Register Fields For as defined in the ZebroBus protocol rules 1 and 2 there are 256 vreg fields each containing one byte of data. The fields consist of 4 kinds: Shared, Class specific, device specific and reserved for further use.

1. Field addresses 0x00 - 0x1F represent the Shared fieds, and are identical for all ZebroBus modules.
2. Field addresses 0x20 - 0x3F represent the Class specific fields, and are equal for all the same classes of modules (e.g. leg modules, sensors, power management systems, ...). In this case all measured sensor values such as power sensors and temperature. These are in word format so e.g. 0x20 will be Temp lsb byte and 0x21 Temp msb byte.
3. Field addresses 0x40 - 0xEF represent the Device specific fields and differ for each module. In this case the battery registers, the power system reads in from the battery system, this could differ per battery management system.
4. Field address 0x60 is the Status register of the power system. Each of the 8 bits defines the current situation of the power system, as follows:

Bit 7	Solar input: On[1],Off[0]
Bit 6	Laptop input: On[1],Off[0]
Bit 5	QI input: On[1],Off[0]
Bit 4	3.3V output: On[1],Off[0]
Bit 3	5V output: On[1],Off[0]
Bit 2	16V output: On[1],Off[0]
Bit 1,0	The FSM state: Start[11], Charge[01],Discharge[10],Shutdown[00]

Each of the fields has been defined for the power management system in the `virtual_register.h` file, An overview of each field address and the implementation is given in table 5.3.

5.4.2. INTERNAL BUS, SENSORY

The Internal bus could be seen as the sensory of the robot. All the physical senses of the robot are brought to the nervous system through the internal bus. All sensors in the power system are connected to the Internal bus. The microcontroller is the gateway in between the sensors and the main controller. Each sensor value is read in and stored in the corresponding virtual register. The microcontroller is therefore the Master on the Internal bus.
All digital sensors bus and register addresses are defined in the `global_register.h` file, an overview of these is given in Table 5.4.

Address Dec	Hex	Size Bytes	Dir	Name	Description
0	0x00	1	r	ZebroBus version	Highest of the ZebroBus protocol supported
1	0x01	1	r	class id	Class this module is part of
2	0x02	1	r	product id	Product ID of this module
3	0x03	1	r	product version	Product version of this module
4	0x04	1	r/w	serial id	Serial number of this module
5	0x05	1	r	software version	Software version of this module
6	0x06	1	r	zebrobus address	Primary ZebroBus slave address of this module
7	0x07		r	reserved	
...	...	...	...	...	Reserved for future use
16	0x0F		r	reserved	
17	0x10	1	r	quick status	One byte module status message
18	0x11	1	r/w	sync counter	Time synchronisation counter
19	0x12	1	r	loop counter	Counts the times we go through the main control loop
20	0x13	1	r/w	error counter	Counts the number of errors that occurred
21	0x14	1	r	last error	Error code of the last error
22	0x15		r	reserved	
...	...	...	...	...	Reserved for future use
31	0x1F		r	reserved	
32	0x20	2	r	Solar Current	The solar cell input current measurement
34	0x22	2	r	Qi Current	The QI input current measurement
36	0x24	2	r	Powerbus Current	The Powerbus current measurement
38	0x26	2	r	3V3 Current	The 3.3V output current measurement
40	0x28	2	r	5V Current	The 5V input current measurement
42	0x2a	2	r	Solar Voltage	The solar cell input voltage measurement
44	0x2c	2	r	Qi Voltage	The QI input voltage measurement
46	0x2e	2	r	Powerbus Voltage	The Powerbus voltage measurement
48	0x30	2	r	3V3 Voltage	The 3.3V output voltage measurement
50	0x32	2	r	5V Voltage	The 5V input voltage measurement
52	0x34	2	r	3V3 Fuse Voltage	ADC 3.3V fuse voltage measurement
54	0x36	2	r	5V Fuse Voltage	ADC 5V fuse voltage measurement
56	0x38	2	r	16V Fuse Voltage	ADC 16V fuse voltage measurement
58	0x3a	2	r	Laptop input Voltage	ADC laptop input voltage measurement
60	0x3c	2	r	Input Temp	Temperature measurement close to input converters
62	0x3e	2	r	Output Temp	Temperature measurement close to ouput converters
64	0x40	2	r	Battery Status	Status of the battery 11.23[19]
66	0x42	2	r	Time To Empty	11.7 Time for battery to discharge [19]
68	0x44	2	r	Time To Full	Time for the battery to charge 11.6[19]
70	0x46	2	r	Battery Temp	Temperature of the battery 11.9[19]
72	0x48	2	r	Battery Voltage	Voltage over the battery 11.10[19]
74	0x4a	2	r	Battery Current	Current through battery 11.11[19]
76	0x4c	2	r	State Of Charge	State of Charge of the battery 11.14[19]
78	0x4e	2	r	State Of Health	State of Health of the battery 11.41[19]
80	0x50	2	r	Number Of Cycles	Number of Cycles of the battery 11.24[19]
96	0x60	1	r	Status	This register reflects the current state of the system

Table 5.3: Virtual register addresses

Address		Description
Bin	Hex	
0b1001000	0x48	Temperature Sensor at Input
0b1001001	0x49	Temperature Sensor at Output
0b1000010	0x42	Current Sensor at Solar Input
0b1000000	0x40	Current Sensor at the power bus
0b1000001	0x41	Current sensor at 5V Output
0b1000011	0x43	Current Sensor at 3.3V Output
0b0100101	0x25	Digital potentiometer
0b0101111	0x2F	QI receiver
0b0001011	0x0b	Battery management system

Table 5.4: Internal bus addresses

Temperature Sensor The temperature sensor as chosen in section 4.2.2 is a LM75BD. This is an I^2C-bus based device. Register 0x00 hexadecimal is the temperature register. The 11 most significant bits of the two bytes define the temperature value in 2's complement notation. Each time the temperature sensor is read in by the TWI the value will be checked and reported

5.4.3. TWO WIRE INTERFACE (TWI)

The microcontroller chosen is an Atmel ATxmega32A4U, this microcontroller has been specifically chosen because it has 2 TWI ports. This choice was made because TWI is compatible with both SMBus[20] and I^2C[21] busses.

SMBus and I^2C Multiple different devices are connected to the bus, so it is very important that all are compatible. For example the Battery Management System works solely on the SMBus protocol, and the ZebroBus is only based on I^2C. Therefore knowing how to deal with the differences is essential. SMBus and I^2C are almost identical. However there are some differences that have to be taken into account. The most important difference between both protocols is Timeout. In Timeout a slave device resets the interface when the Clock goes down for longer that the timeout time. For SMBus this timeout time is defined as 35ms, thus it has a minimum clock speed of 10kHz. The I^2C protocol in contrast has no timeout implemented, so both master and slave can hold the clock low for as long as it wants to process data. To make sure both are compatible, the microcontroller has to wait for a certain timeout to give the I^2C devices more processing time, but also detect a bus error once a SMBus device resets the interface. Secondly, SMBus is limited to the clock speed of 100kHz, while I^2C permits speeds up to 400kHz. Complete compatibility can only be ensured below 100kHz, because then the protocols work at the same speed. So the clock rate of 100kHz has to be used. Other minor differences are some hardware specifications listed in table 5.5. Although there is quite a difference in level specifications and pullup resistor recommendations, this is often not a big issue. The reason is that bus devices often always have a output voltage swing close to the supply voltage, which complies with both protocols.

Protocol	I^2C	SMBus
Timeout	None	25 to 35 ms
Frequency	100kHz, 400kHz(or 2Mhz)	10 to 100kHz
V_{HIGH}	$0.7 \cdot V_{DD}, 3V$Fixed	2.1V
V_{LOW}	$0.3 \cdot V_{DD}, 1.5V$Fixed	0.8V
I_{max}	3mA	$350\mu A$
$R_{pullup}3V$	>1k	>8.5k
$R_{pullup}5V$	>1.6k	>14k

Table 5.5: Hardware specification differences between I^2C[21] and SMBus[20]

Bus topology In general TWI provides a simple two-wire communication bus consisting of a serial clock line (SCL) and a serial data line (SDA). Both lines are open-collector and pulled by a pull-up resistor and form a Bus together where multiple devices can be connected. A device connected can be either a Master or a Slave, where the master controls all bus communication. The topologies of the ZebroBus and Internal Bus are shown in figures 5.7 and 5.8. The buffer in the ZebroBus topology has to convert the 5V pull-up voltage of the main controller to the 3.3V of the microcontroller on the power system 4.3.4.

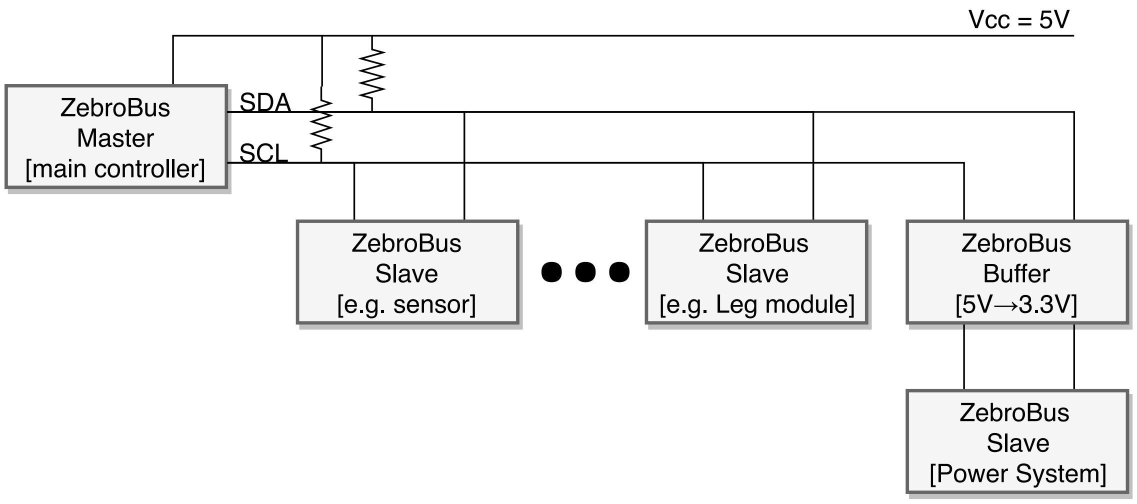

Vcc = 5V
SDA
SCL
ZebroBus Master [main controller]
ZebroBus Slave [e.g. sensor]
ZebroBus Slave [e.g. Leg module]
ZebroBus Buffer [5V→3.3V]
ZebroBus Slave [Power System]

Figure 5.7: ZebroBus Two Wire Bus topology

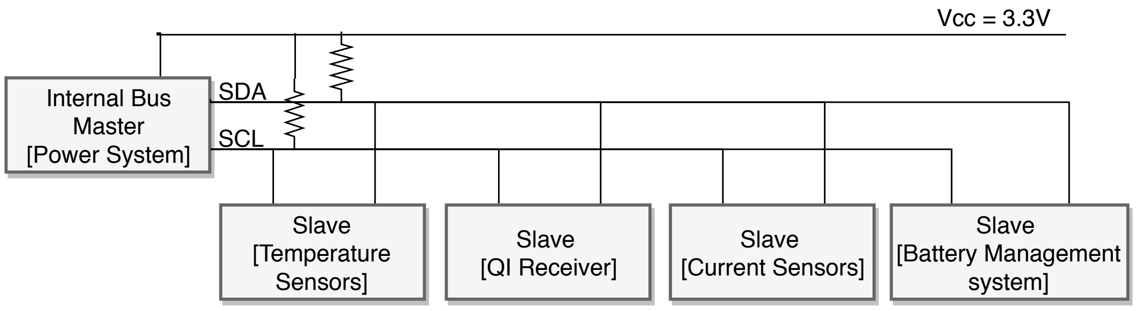

Vcc = 3.3V
SDA
SCL
Internal Bus Master [Power System]
Slave [Temperature Sensors]
Slave [QI Receiver]
Slave [Current Sensors]
Slave [Battery Management system]

Figure 5.8: Internal Bus Two Wire Bus topology

TWI transaction A transaction is a complete transfer from Start to Stop, with possibly a repeated start in between. A typical two wire transaction is shown in 5.9. First of all the Master starts by sending a Start bit. Then it sends the 7 bit Address of the Slave it wants to reach, followed by a Read/Write bit. If the Slave gives back an Acknowledge, the Master or Slave can send data over the bus. This repeats until a Notacknowledge NACK is send by either of the two, followed by a Stop bit of the Master which will end the transaction.

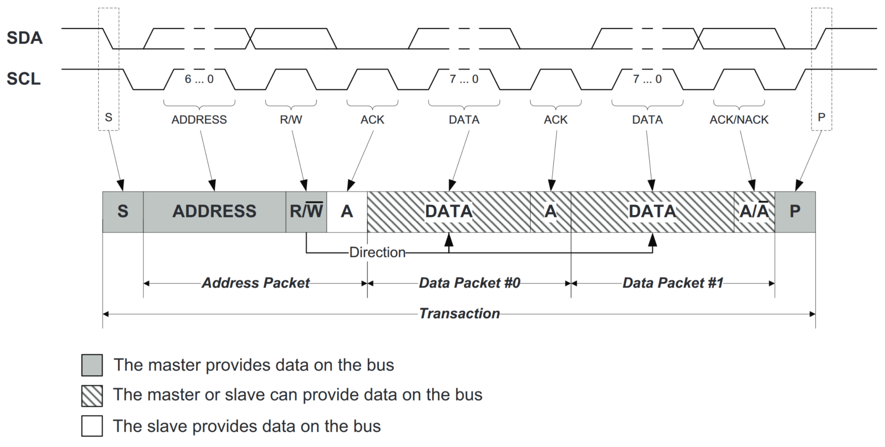


Figure 5.9: TWI Transaction[14]

If the Master wants to read out a value from the Slave it should put the R/W bit high and wait for the acknowledge. After that the master can receive the data from the slave, which can be either one byte or multiples as shown in figure 5.10. After the last byte the Master has to send a NACK followed by a stop byte to finish the read transaction.

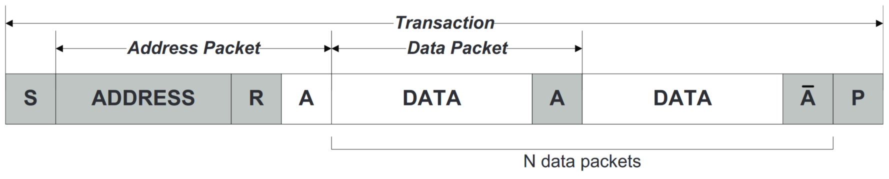


Figure 5.10: TWI Read[14]

In case the Master wants to write a value to the Slave it should put the R/W bit low and wait for the acknowledge. Then the master can keep sending data bytes, until it gets a NACK from the slave. Sending a last stop byte will stop the transaction and the bus will be open again.

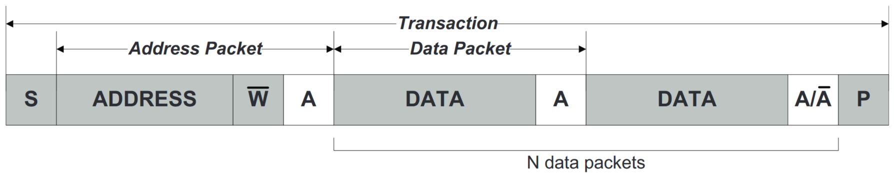


Figure 5.11: TWI Write[14]

In the case of the power system the sensors often have many more than one byte to read out. So instead of returning a single byte after a write command, the Master first has to specify which register address it wants to read from. This can be done by first doing a write transaction with the specific register address as data. Immediately after that a repeated start [Sr] can be send, followed by a read transaction. This way the bus is not interrupted in between writing the address and reading the data. Figure 5.12 shows such a transaction.

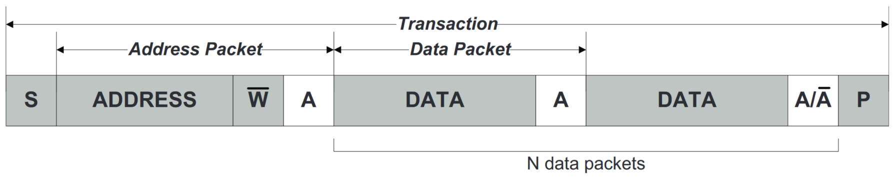


Figure 5.12: TWI Read Slave Register

TWI MASTER IMPLEMENTATION

The TWI Master implementation is used in the Internal Bus to set and read out all sensors within the power system. The Master is responsible for starting and stopping transactions between the different Slaves on the power system. The built-in TWI driver of Atmel is used to implement the TWI master. This choice has been made, because other selfconstructed techniques as bit-banging are often less efficient and are more fault sensitive. Bit banging, means only one process can be done at the time, while the built-in TWI driver uses interrupts, which allows parallelism and makes the MCU usage way more efficient. [22]

In order to implement the TWI master driver it first has to be included and initialised. The Internal Bus is connected to the SDA and SCL of port C, so the master will be initialized to &TWIC. Then, because the sensors are not the most critical the interrupt level is set to low. The Baudrate and Clockspeed have to be defined and set. The BAUDRATE is set to 100kHz, as that complies with both SMBus and I^{2}C. See the code below:

```c
#include "twi_master_driver.h"
#include "sensor.h"
TWI_Master_t  twiMaster;

#define BAUDRATE  100000
#define CPU_SPEED 2000000
#define TWI_BAUDSETTING TWI_BAUD(CPU_SPEED, BAUDRATE)

void sensor_init()
{
  TWI_MasterInit(&twiMaster, &TWIC, TWI_MASTER_INTLVL_LO_gc, TWI_BAUDSETTING);
}
```

Each sensor has its own bus address and register addresses which have to be read out and stored into the virtual registers of the microcontroller. So a function for each sort of devices has been made. Below is the code of the master readout of the current sensors. For each current sensor address, each voltage and current register will be read out. The TWI master write and read function will be called. This function will first write the register value to the slave address and then read two bytes, just as in figure 5.12. After waiting for the transaction to be done the read data can be checked and written to the corresponding virtual registers. Within the Check and write registers function the data is checked comparing it to the limits to raise error flags. At last the virtual registers are updated.

```c
void Read_Current_Registers(void)
{
  uint8_t current_addresses[4] = {ADDRESS_CURRENT_SENS_BUS,
                                  ADDRESS_CURRENT_SENS_5V,
                                  ADDRESS_CURRENT_SENS_3V3,
                                  ADDRESS_CURRENT_SENS_SOLAR};
  uint8_t current_register[2] = {CUR_BUS_VOLTAGE_REG,
                                 CUR_CURRENT_REG};

  for (uint8_t AdrPos = 0; AdrPos < 4; AdrPos++)
  {
    for (uint8_t RegPos = 0; RegPos < 2; RegPos++)
    {
      TWI_MasterWriteRead(&twiMaster,
                          current_addresses[AdrPos],
                          &current_register[RegPos],
                          1,
                             2);
      while(twiMaster.status != TWIM_STATUS_READY) {}
      Check_And_Write_Current_Registers(current_register[RegPos],
                                        current_addresses[AdrPos],
                                        twiMaster.readData[0],
                                        twiMaster.readData[1]);
    }
  }
  //update vregs
  vregs_writeout();
}
```

TWI SLAVE IMPLEMENTATION

The TWI Slave will communicate with the ZebroBus[6]. The ZebroBus is connected through the buffer(4.3.4) to port E on the microcontroller. The slave register needs to be initialised and set to port E, also the slave process data function is defined to decide over what is send over the bus. The slave module is initialised to a new address, which will be 0x60 based on the ZebroBus protocol [6]. The interrupt level is set to low and low level interrupts are enabled in line 8 and 9. Note also the interrupt handle needs to be declared as in line 14-16.

```
1  TWI_Slave_t   twiSlave;
2
3  void zebrobus_init()
4  {
5    /* Initialize TWI slave. */
6    TWI_SlaveInitializeDriver(&twiSlave, &TWIE, TWIE_SlaveProcessData);
7    TWI_SlaveInitializeModule(&twiSlave,0x60,TWI_SLAVE_INTLVL_LO_gc);
8    PMIC.CTRL |= PMIC_LOLVLEN_bm;
9    sei();
10 }
11
12 /*! TWIC Slave Interrupt vector. */
13 ISR(TWIE_TWIS_vect)
14 {
15   TWI_SlaveInterruptHandler(&twiSlave);
16 }
```

Once there is an interrupt from the slave driver the data processing function is called. This function is the junction between the ZebroBus and the virtual registers. The first received byte from the ZebroBus is the register Index of the virtual register(line 3). The amount of bytes received determines the size of the buffer, and thus how many bytes should be written(line 4). As stated in the ZebroBus protocol[6] the slave will always give the value of the first register requested(line 5). During prototyping, however 2 bytes could be read in right after each other so the next byte in the register is stored in the following send data(line 6). A possibility for the ZebroBus to read in the complete virtual register is not implemented in the ZebroBus protocol, but discussed in recommendations(8)(line 7-11). When the slave receives more than one byte, each next byte has to be written into the virtual registers, just as described in rule 6 of the ZebroBus protocol[6](line 13-17).

```
1  void TWIE_SlaveProcessData(void)
2  {
3    uint8_t bufIndex = twiSlave.receivedData[0];
4    uint8_t bufSize = twiSlave.bytesReceived;
5    twiSlave.sendData[0] = vregs_read_buffer(bufIndex); //send the read data from register
6    twiSlave.sendData[1] = vregs_read_buffer(bufIndex+1);//send the read data from register+1 if word
       is requested
7    /*Suggestion for ZebroBus protocol to read all registers*/
8    //for (uint8_t i=0;(bufIndex+i)<VREGS_FILE_TOTAL_SIZE;i++)
9    //{
10   //  twiSlave.sendData[i] = rw_reg[bufIndex+i];
11   //}
12   for (uint8_t recvIndex = 0; recvIndex<bufSize; bufIndex++)
13   {
14     recvIndex++;
15     vregs_write(bufIndex, twiSlave.receivedData[recvIndex]);
```

```
    }
}
```

5.5. ERROR HANDLING, PAIN

The Zebro robot can get hurt, just like a normal animal it reflexes and sends a signal to the brain it feels pain. For example if you touch a hot soldering iron, your first reflex is to get your hand away and secondly you feel the pain. This reflex skips the brain and retracts the hand immediately. The overcurrent protection symbolizes this reflex, while the interrupt received by the MCU represents the sting that was measured by the sensors. Errors in the system generate interrupts. Each error can trigger an ISR to respond appropriately. An example of error handling is shown in Figure 5.13.

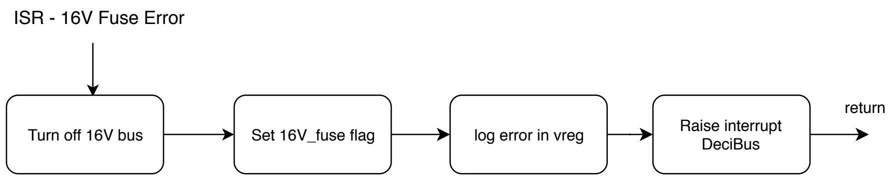


Figure 5.13: Example of error handling within the system.

The flag set will prevent the FSM from turning the output on again, the fuse flag cannot be cleared by the main Zebro controller to restore functionality as represents a physical connection being broken. The overcurrent and voltage flags on the other hand do allow for this. The error will be reported, its value stored in a register and the error count is increased**[SR-7]**. Critical errors mainly concern the batteries as they have to be safe and secured in any scenario and when faults occur.

Error numbers The error handling has been implemented by having a separate array with the size of all errors. All addresses in the error array have been defined as a array address. The error addresses are defined in `error.h`, of which an overview is shown in Table 5.6. These error addresses are also the Error numbers which the main controller can read out in case an error was reported.

Error Number	Error Name	Description	Source
0	NONE	no error	
1	OCA	Over-Current Alarm	Battery Management System
2	TCA	Terminate Charge Alarm	Battery Management System
3	OTA	Over Temperature Alarm	Battery Management System
4	TDA	Terminate Discharge Alarm	Battery Management System
5	RCA	Remaining Capacity Alarm	Battery Management System
6	RTA	Remaining Time Alarm	Battery Management System
7	BUSY	Battery Busy	Battery Management System
8	RESERVEDCMD	Reserved command	Battery Management System
9	UNSUPPORTEDCMD	Unsupported command	Battery Management System
10	ACCESSDENIED	Access denied	Battery Management System
11	UNDEROVERFLOW	Underflow or Overflow	Battery Management System
12	BADSIZE	Bad Size	Battery Management System
13	OCBUS	Over-current at Power Bus	Current Sensor
14	OCSOLAR	Over-current at Solar input	Current Sensor
15	OC3V3	Over-current at 3.3V output	Current Sensor
16	OC5V	Over-current at 5V output	Current Sensor
17	OC16V	Over-current at 16V output	Current Sensor
18	OCQI	Over-current at QI input	QI Receiver
19	FUSE3V3	Fuse blown at 3.3V output	ADC
20	FUSE5V	Fuse blown at 5V output	ADC
21	FUSE16V	Fuse blown at 16V output	ADC
22	OTI	Over-Temperature at input	Temperature Sensor
23	OTO	Over-Temperature at output	Temperature Sensor
24	UTI	Under-Temperature at input	Temperature Sensor
25	UTO	Under-Temperature at output	Temperature Sensor
26	UNKNOWN		
...	...	Reserved for future use	
255	ERROR UNKNOWN		

Table 5.6: Error numbers and array addresses

Error reporting Once a sensor reads in a value that is troublesome it will report the error. This is done by a check after each read in from any of the sensor values. If the value is above a certain threshold, e.g. temperature at the input above 45^oC then it will call the error report function. In the error report function interrupts are temporarily turned off, so that the error can be reported safely. If the value in the error register was already reported before, then it will increase the error counter by 1(line 7-9). Then it will set the value of the register corresponding to the error code given to 1, so the register logs the specific error(line 10). The error number is also stored in the last error variable. Lastly both the error counter and last error virtual registers are updated(line 14-15) and interrupts are turned on again(line 17).

```
1  void error_report(uint8_t error)
2  {
3    /* BEGIN critical section */
4    cli();
5
6    /* check what errors where already reported */
7    if(errors_array[error - 1] == 0){
8      error_counter++;
9    }
10   errors_array[error - 1] = 1;
11
12   last_error = error;
13
14   vregs_write(VREGS_ERROR_COUNTER, error_counter);
```

```
  vregs_write(VREGS_LAST_ERROR, last_error);

  sei();
  /* END critical section */
}
```

ZebroBus Interrupt In case of a critical error or shutdown, meaning the power will go off, the main controller needs to be aware of what happened. Therefore in case of a critical error the interrupt pin is pulled low **[SR-8]**, by setting the interrupt mosfet 4.3.4.

6
PROTOTYPING

Testing the control for a power system without having the system yet is difficult, as debugging is not possible. Therefore a partial prototype of the control part of the system has been made to simulate and debug the code. For this the DeciZebro Leg module PCB was used and slightly adapted, see Figure 6.1. It uses the ATxmega32A4 microcontroller. Several parameters and functions have been tested on the prototype board which will save time debugging the main PCB:

1. **Power consumption**
2. **Communication, internal and external**
3. **Temperature sensor**
4. **Current sensor**
5. **Voltage division ADC measurement**
6. **MPPT implementation**

However not all functions are available on the Leg module and there are many hardware differences, so when the prototype PCB arrives many tests have to be done to verify successful integration of the system. This test plan includes:

1. **Execution Time** With a time measurement of the total loop, the energy consumption can be calculated
2. **MPPT integration** A test of the combination of reading in the data from the current sensor and setting the eventual pwm, to verify the power increases
3. **Error handling** Different errors should be artificially set, to verify the errors are well handled and reported
4. **FSM** The working of the FSM has to be verified, to go to the correct states while changing the inputs
5. **Sensor calibration** On the new PCB, all sensors have to be recalibrated and checked on noise behaviour
6. **Runtime** In order to see if the watchdog timer works, and if the power system is able to work for a longer time. A long runtime test should be executed.
7. **Efficiency** The efficiency of the power system should be measured, to see how much energy is lost between input and output, and in what way this could be improved

Condition	Current	Voltage	Power
Standby	0.47mA	4.944 V	2.32 mW
Sleep idle	0.57mA	4.933 V	2.81 mW
Empty loop	87.3mA	4.945 V	432 mW
0% Dutycycle	91.2 mA	4.935 V	450 mW
50% Dutycycle	90.8 mA	4.932 V	448 mW
100% Dutycycle	91.0 mA	4.932 V	449 mW
100% Dutycycle + 2 LEDS	103.9 mA	4.899 V	509 mW

Table 6.1: Power consumption

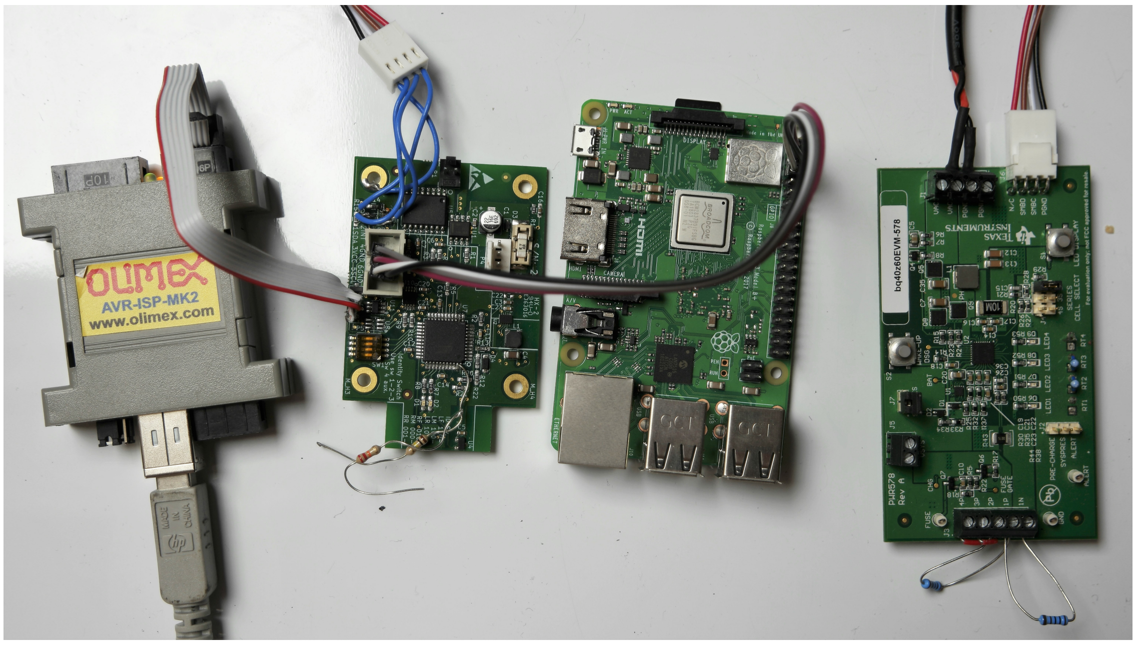


Figure 6.1: Communication test setup

6.1. POWER CONSUMPTION

The leg test module is powered by an AVR-ISP-MK2 usb interface. This USB connection could be intercepted with a special board, hence all power drained from the usb port could be measured. Different conditions have been tested in order to get a view on what a PCB could consume. The results are shown in table 6.1. Note the usb voltage is 5V, so the conversion to 3.3V costs some power already.

6.2. COMMUNICATION

The communication of both the Internal bus and ZebroBus have been tested using the leg module as microcontroller. The communication test setup is shown in Figure 6.1

ZebroBus The leg module is designed to also be able to communicate with ZebroBus, therefore it already had a port and buffer available for prototyping. To test the ZebroBus communication the I^2C pins of a Raspberry Pi have been connected to the leg module. Using the Pi I^2C interface it was possible to both read and write the virtual registers of the microcontroller. So the ZebroBus is verified to work properly.

Internal bus Attaching devices to the internal bus of the leg module was more difficult, as the device did not have output pins for connecting another bus. Therefore separate wires have been handsoldered onto the leg module to intercept the internal bus. With this bus all slave devices have been tested and verified to work, including:

- **Battery Management System:** Using a test battery module bq40z60EVM-578 it has been verified the SMBus slave of the battery mangagement system was write-/readable
- **Temperature Sensor:** Using a breakout board with a LM75B it has been verified the temperature could be read out via I^2C
- **Current Sensor:** Using a breakout board with an INA226 the current sensor communication was verified

6.3. TEMPERATURE SENSOR

The temperature sensor(LM75B) measures the temperature on the device and has to be very accurate in case of a battery system. If the temperature is too high during charging this can severely damage the batteries. During prototyping this has been taken very serious and a complete reference measurement has been done in a range of 10 to 65 ^{o}C. A calibrated kitchen ntc temperature sensor(GTTMBBQ02) has been used as reference. The result is shown in figure 6.2.

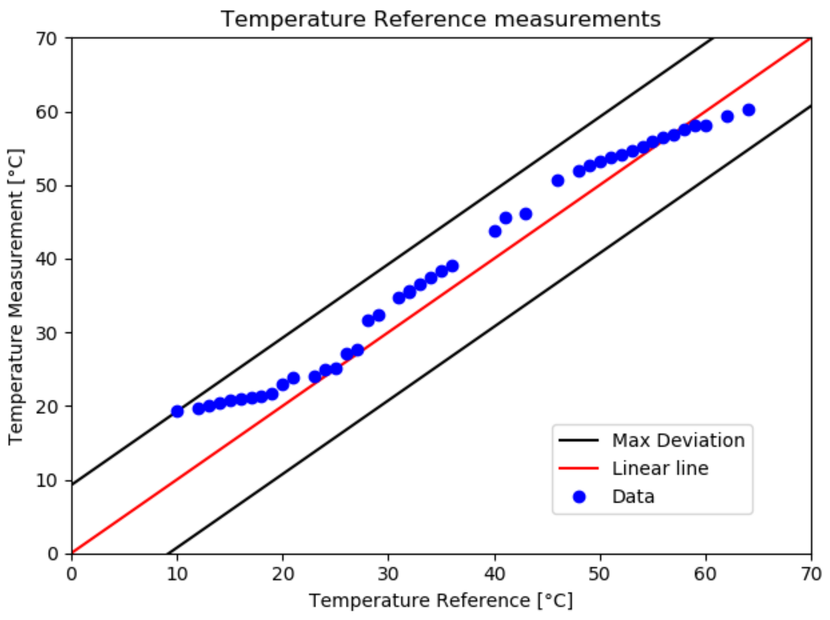


Figure 6.2: Temperature reference measurement

The value of the temperature sensor is defined as the 11 most significant bits and has a resolution of 0.125^{o}C[11]. So initially the hexadecimal value was shifted to the right by 5 and then multiplied by 0.125. In the ideal case the measured temperature would be equal to the reference temperature, this would result in a linear line. However the data measured is off this line for many times. With a small python script **C** statistical calculations have been done to get a view on the accuracy.

The maximum deviation is 9.25^{o}C and standard deviation is 2.014^{o}C. Also the measured temperature looks to be too high at low temperatures, this could be because the reference temperature sensor was in between the cooler and the LM75B.

6.4. CURRENT AND VOLTAGE SENSOR

The INA226 measures only the bus voltage and the differential voltage across the shunt resistor. Both of these measurements are automatically stored with a resolution of 1.25mV/bit and 2.5uV/bit respectively. However the current and power measurement have to be calibrated by the user as they are dependent on the value of the shunt resistor and the expected current.

The calibration values can be calculated form the following equations [23]:

$$CAL = \frac{0.00512}{Current\_LSB * R_{SHUNT}} \tag{6.1}$$

Where

$$Current_{LSB} = \frac{MaximumExpectedCurrent}{2^{15}} \tag{6.2}$$

Filling in the equations with the expected values of the system the resolution will be the highest. However by sacrificing a bit of resolution more convenient conversion value can be selected. Table 6.2 shows the calibration values for each of the current sensors.

Table 6.2: Calibration values for each current sensor.

	Expected maximum current	Current_LSB	Selected_LSB	CAL
Solar input	.5 A	15 uA/bit	25 uA/bit	2048
Bus	5A	153 uA/bit	250 uA/bit	4096
5V output	1A	31 uA/bit	50 uA/bit	2048
3.3V output	1A	31 uA/bit	50 uA/bit	2048

The current can now be calculated as:

$$Current = \frac{ShuntVoltage * CalibrationRegister}{2048} \tag{6.3}$$

The power can then be computed from the current value and the measured Bus voltage:

$$Power = \frac{Current * BusVoltage}{20000} \tag{6.4}$$

A measurement of the bus voltage was done with the INA226 and a multimeter as reference. In a range from 6 to 25 V, multiple measurements are done and stored. The result is plotted in figure 6.3. A linear fit through the data gave the following formula:

$$V_{Bus} = 0.001246 \cdot \texttt{Measured data} + 0.04878 \tag{6.5}$$

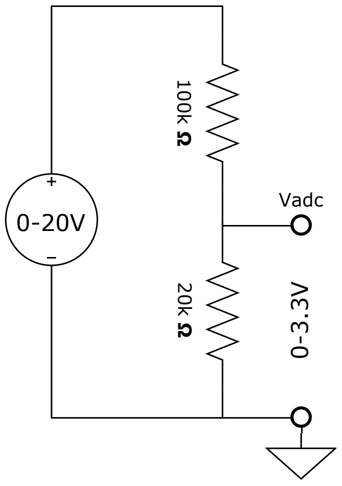


Figure 6.4: ADC test setup

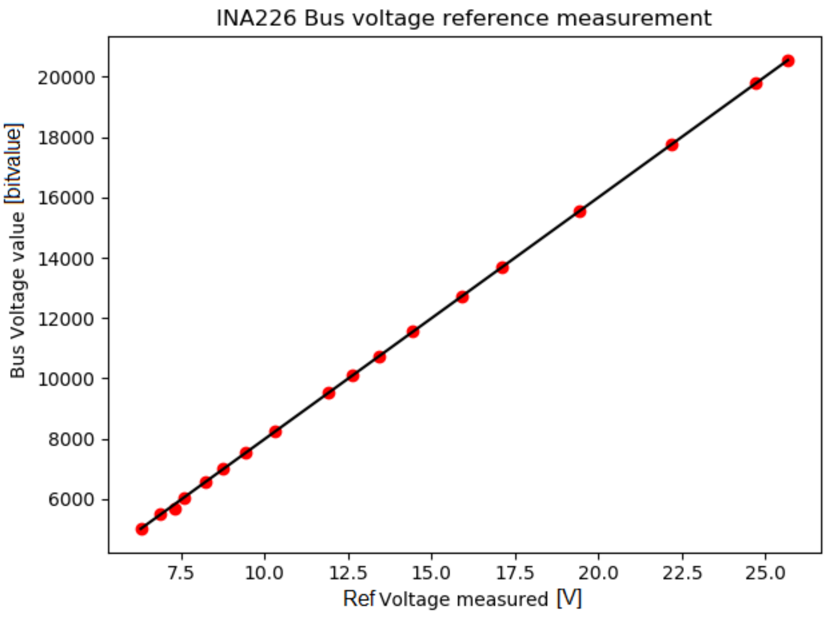


Figure 6.3: INA226 bus voltage reference measurement

6.5. ADC

The microcontroller has to use multiple of its analog to digital converters in order to read relevant data from the power system. Examples of uses are in the fuses and at the input. To get a view on what the ADC reads in and what values it gives back this had to be tested preliminary. A small test setup has been made with a voltage division connected to a DC power supply, as shown in figure 6.4. The power supply can be set to 20V, which means the ADC voltage ranges from 0 to 3.3V.

In order to test the ADC, different voltages have been applied. In steps of 0.1V, the voltage was measured and compared to the value read out by the ADC. However as discussed in chapter 5.3.3 it can only measure from 0.2 to about 2V, above that it gives an over-/underflow. The results are shown in figure 6.5.

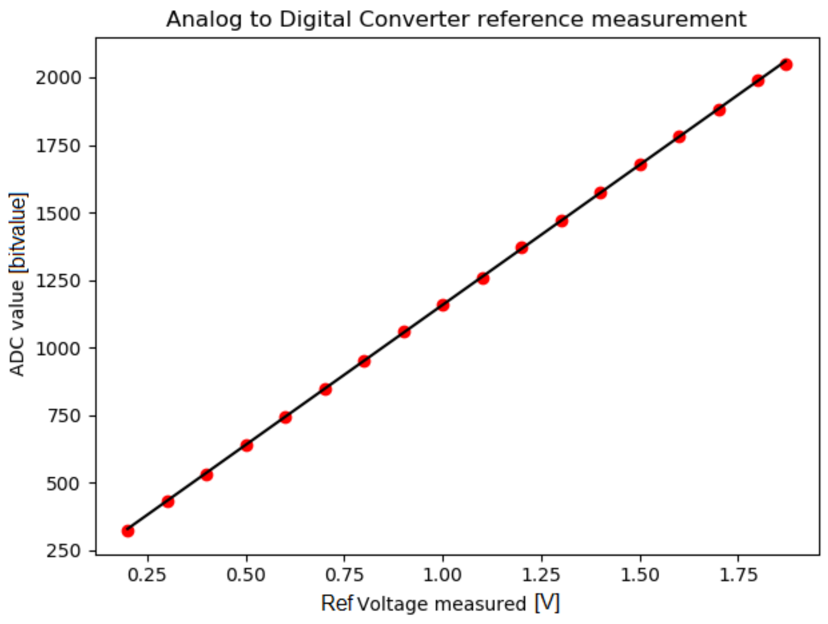


Figure 6.5: ADC reference measurement

With this measurement information the conversion can be done accurately. The following formula was extracted from the linear fit of the graph:

$$V_{ADC} = 0.0009653 \cdot \texttt{Measured Data} - 0.1182 \tag{6.6}$$

At approximately the value of 1.87V the ADC gives an overflow value and cannot become any higher. Also below the value of 0.2V the ADC value is unreliable, thus an underflow value will be generated.

6.6. MPPT

Optimizing the harvesting of energy from the solar panel can be done with a Maximum PowerPoint Tracking. The converter is designed by the peripheral group. The way to control this converter to get the most energy out of the solar panel was discussed in section 5.3.2. This chapter will explain how the solar panel was tested to prove the panel has a maximum power point.

Pulse Width modulation The Cúk converter of the Solar panel has to be controlled by the Microcontroller, so that the MPPT algorithm can be applied. The Dutycyle of the pwm signal driving the Mosfet of the Cúk converted determines the amount of power which is extracted from the solar panel. A function was written to set the dutycycle of the pulse width modulation. An integer of 0 sets the dutycycle to 0% and 256 sets it to 100%. The maximum peak on the solar panel you get with a dutycycle of 80%, after some trial and error the integer value was set to 200. The pulse is shown in 6.6.

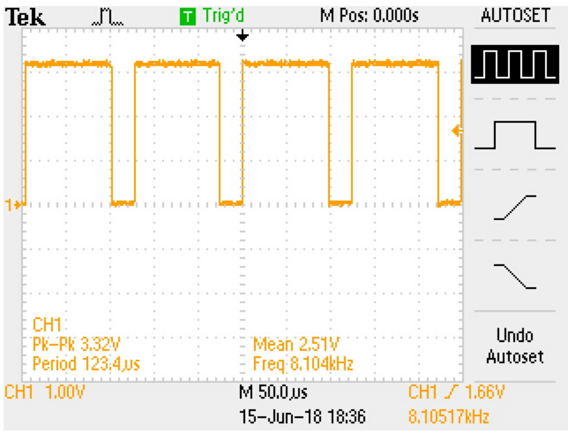


Figure 6.6: PWM signal on the mosfet, compared to the output voltage

A test program has been written to find out how the dutycycle influences the solar power. The test-program sets the dutycycle from 0 to 100% in a loop. Both the voltage on the PWM signal and the voltage over the load are measured. The result is shown in figure 6.7.

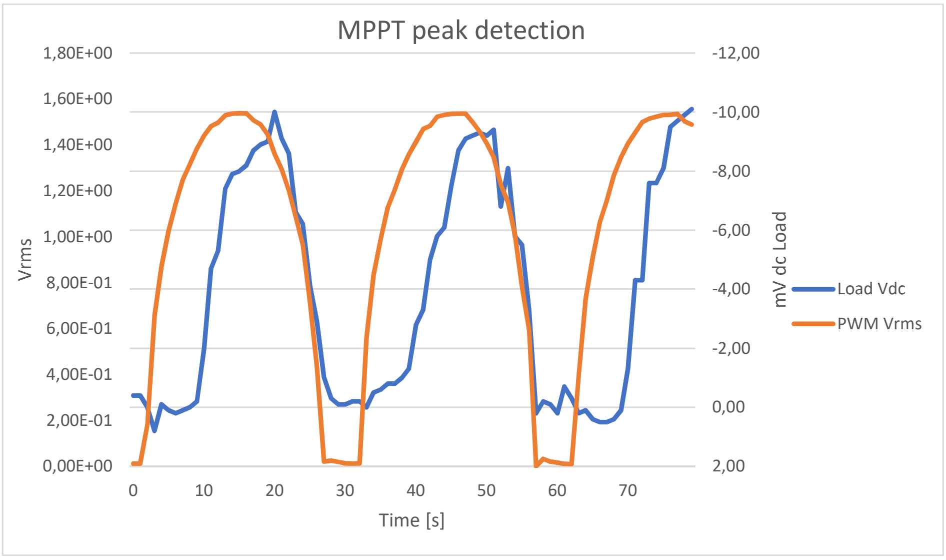


Figure 6.7: Dutycycle MPPT test

Figure 6.7 shows one loop in which it is clear the rms voltage of the pwm signal increases and decreases fluently as the dutycycle goes from 0 to 1 00%. The voltage over the load is negative as the Cúk converter inverts the voltage. So the maximum powerpoint is the point where the voltage over the load is most negative. The graph above shows one clear peak at a dutycycle of approximately 70%.
Multiple measurements have been done to get a view on the consistency of the Maximum PowerPoint peaks. Approximately 40 minutes of data has been collected and the peak was detected with a peak detection algorithm in Python, see appendix **C**. The script detects the peaks in the MPPT and compares it to the PWM dutycycle. A big data analysis has been done with all dutycycle calculations. The statistical data is shown in a histogram in figure 6.8. The mean of the distribution is 68 and the standard deviation 7. This means the MPP dutycycle is quite consistent.

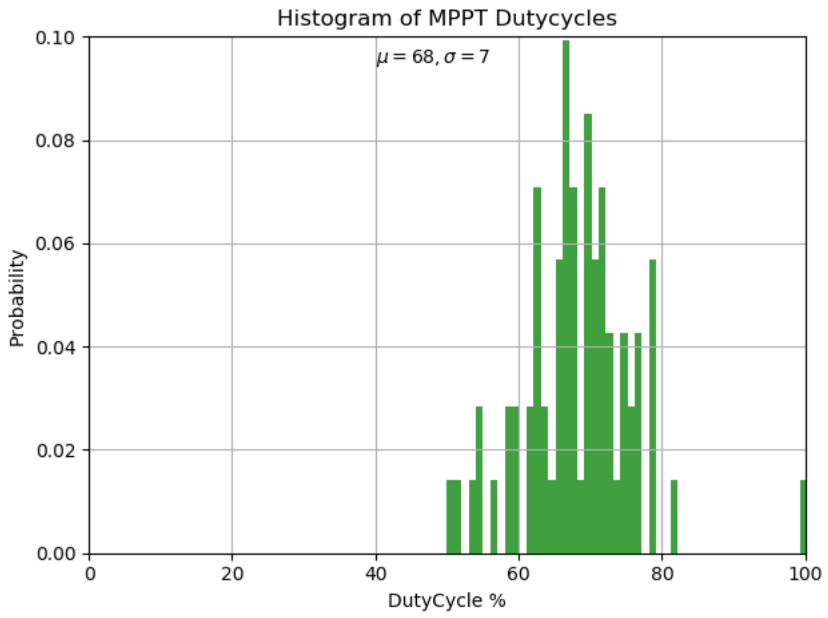


Figure 6.8: Histogram Maximum Power Point dutycycle

7
Discussion

7.1. Power consumption

The power consumption measurements show a large deviation from the expected value listed in the datasheet of the MCU, The sleep and standby modes to however show very little power consumption. The maximum current draw appears to be slightly over 100mA, the specified limit of the selected LDO, the two debugging LEDs account for roughly 15mA and as such should be used with care. The results indicate that the power draw of the prototype PCB's internal system might also be at the specified limit and as such special notice of this will be taken during testing. The increased current consumption will also lower the expected duration of the backup power supply, if a constant current consumption of 90mA is assumed the backup time will be around 21ms. One thing to keep in mind with this test was that AVR-ISP-MK2 also consumes power and that the current draw by the MCU was not measured directly at the supply pins as this was impossible. This will be verified in the prototype PCB, in the design the current drawn by the internal power supply can be measured by measuring the current flowing through the EMI filter.

7.2. Communication

Communication with the ZebroBus, the battery management system and the current/voltage sensor has been verified. The BMS used for communication testing is not identical to the designed prototype, however the interface is identical and as such it can be seen as representative. Communication with the temperature and current sensors is verified. Communication with the digital potentiometer and the QI controller has not yet been tested as it is a ball grid array IC, however since the internal bus has been verified for the other sensors, we are confident that communication will be successful on the prototype PCB.

7.3. Temperature sensor

The measurement results with the temperature sensor show a standard deviation of 2^oC. The test conditions were however far from ideal. Especially at the higher and lower temperature range the test results show increasingly large deviations. The measurement does seem to indicate a slight offset from a linear relation, cause of this could be internal however the test setup is also a possible reason. In the prototype PCB it must therefore be verified if the temperature measurement does indeed have offset and if so, be accounted for.

7.4. Current and Voltage sensor

The bus voltage measurement appears to be quite accurate and agree with the specification with a offset of $\pm 7.5mV$. The shunt current measurement was not yet performed, however this test will not be representative as the shunt measurement will be performed on the prototype PCB and can be subject to interference from various sources. The impact of this on the measurement will have to be verified on the prototype.

7.5. ADC

The voltage division setup connected to the test MCU shows that the upper and lower boundaries of the reference voltage are slightly below the specified boundaries by the datasheet. The selected voltage division will still deliver results within the boundary. The maximum measured input voltage will be $1.87V * 108k/8k \approx 25V$. The lower boundary being slightly limited will not impact the system. By connecting the setup to multiple pins of the MCU at once, the multichannel conversion was also verified.

7.6. MPPT

The resulting load voltage coming from the Cúk converter is below 10mV, which is almost nothing. The reason for this is not proven, however we suspect the microcontroller PWM cannot deliver enough current to fully open the MOSFET in the Cúk. At the time of writing this thesis this has been discussed with the input group and a possible solution such as a gate driver or different MOSFET has not been tested as of yet. The PWM signal frequency is quite low at 8.1kHz in the current configuration resulting a significant voltage ripple if operating correctly. The MCU allows for the PWM to run on the 128MHz pll oscillator so by using it the frequency can be improved while maintaining resolution. This has yet to be tested however.

The Maximum Powerpoint could be clearly visualised with the measurements done. However the application of the Maximum Powerpoint Tracking technique has not been tested yet. Seperately reading in the sensor and incrementing the PWM signal have been verified, but the combination was not be tested as the sensor and the PWM could not be connected at the same time. So this MPPT integration test also still has to be done when the main PCB arrives.

8 Conclusion and recommendations

The control hardware for the power management system is designed and the proposed design takes into account the requirements stated in the PoR. The design offers resilience against errors caused by internal and external faults by localizing the shut down of the outputs and inputs. Components have been selected by keeping in mind the size restrictions set and the price of the system. The firmware design is implemented with the use of a finite state machine and ISR's to allow errors to be cleared by the main Zebro controller providing more flexibility during operation. Safety is a high priority in the design and several hardware and software measures have been implemented. For hardware, non resettable fuses were used in series with every incoming and outgoing power line to ensure that during an extended overcurrent event the fuse will blow and break the connection. TVS diodes were used to prevent ESD from harming the system and a watchdog timer servers as a software backup in case it malfunctions.

At the time of writing the thesis the system has been designed and several components have been tested and verified. Further testing of the system is being undertaken and the prototype PCB has been ordered, however it has not yet arrived at this time. Complete system integration and debugging will therefore be done in the following weeks.

Recommendations The system was developed in a short amount of time and because of that not all aspects have been thoroughly examined to provide optimal performance. The focus consisted mainly on getting it all working. The firmware design can be optimized for power usage by making more appropriate use of interrupts, the ADC conversion can also be stored by skipping the CPU and using direct memory access.

A recommendation for the ZebroBus protocol regards its the safety and resilience, it is a open I^{2}C bus without any check of the data, such as a Cyclic Redundancy Check. In case of a space environment such as the moon radiation could easily manipulate the data. A bit flip would result in a wrong read in the main controller of the ZebroBus and have considerable consequences.

A

MCU SIGNAL OVERVIEW

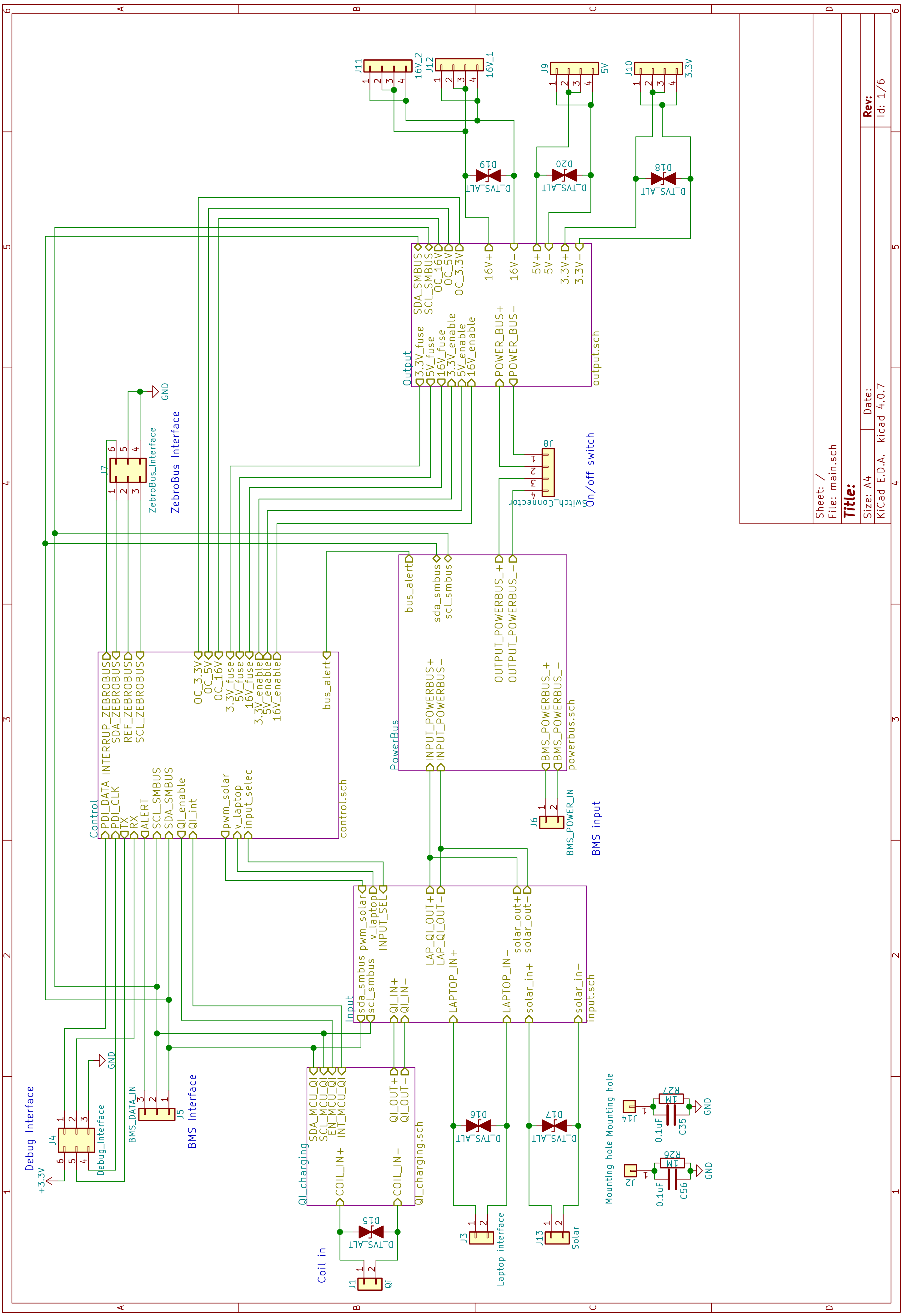
Table A.1: List of signal lines connected to the MCU.

Pin	Dir.	Type	Name	Description
PA1	IN	A	3.3V_fuse	3.3V output fuse alert
PA2	IN	A	5V_fuse	5V output fuse alert
PA3	IN	A	16V_fuse	16V output fuse alert
PA4	IN	A	v_laptop	input voltage laptop connector
PA5	IN	D	OC_3.3V	3.3V sensor overcurrent/voltage alert
PA6	IN	D	OC_5V	5V sensor overcurrent/voltage alert
PA7	IN	D	OC_16V	16V shutdown alert due to overcurrent.
PB0	IN	D	bus_alert	Powerbus overcurrent/voltage alert
PB1	IN	D	QI_int	Overcurrent/voltage/temperature alert QI charger
PB2	IN	D	input_select	QI or laptop charger input selection
PC0	BI	D	SDAA	Data line for the ZebroBus
PC1	IN	D	SCLA	Clock line for the ZebroBus
PC2	OUT	D	buffer_enable	Enable signal for the ZebroBus buffer
PC3	OUT	D	pwm_solar	PWM control signal for the MPPT Solar converter
PC7	OUT	D	Interrupt	Interrupt line for ZebroBus
PD0	OUT	D	3.3V_enable	3.3V output enable signal
PD1	OUT	D	5V_enable	5V output enable signal
PD2	OUT	D	16V_enable	16V output enable signal
PD3	IN	D	ALERT	BMS alert
PD5	OUT	D	QI_enable	QI charger enable
PD6	IN	D	RX	UART communication
PD7	OUT	D	TX	UART communication
PE0	BI	D	SDA_SMBUS	Internal TWI
PE1	OUT	D	SCL_SMBUS	Internal TWI
PE2	IN	D	OT_output	Overtemperature alert output
PE3	IN	D	OT_input	Overtemperature alert input
PR0	OUT	D	led_0	Debugging led
PR1	OUT	D	led_1	Debugging led
$\overline{RESET}/PDI$	IN	D	PDI_CLK	Clock signal of debug interface
PDI	IN	D	PDI_DATA	Data signal of debug interface

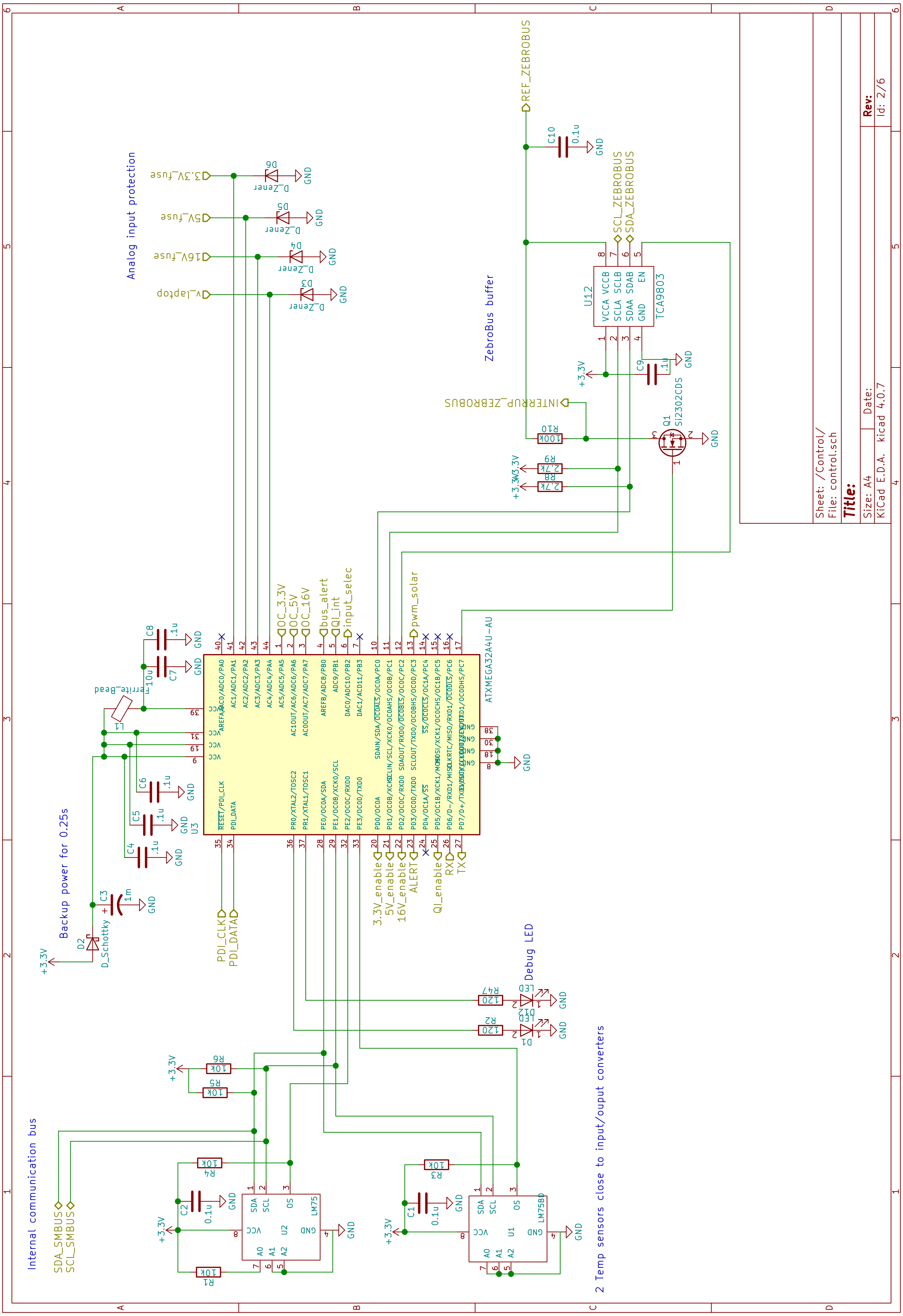
B

KiCad schematics

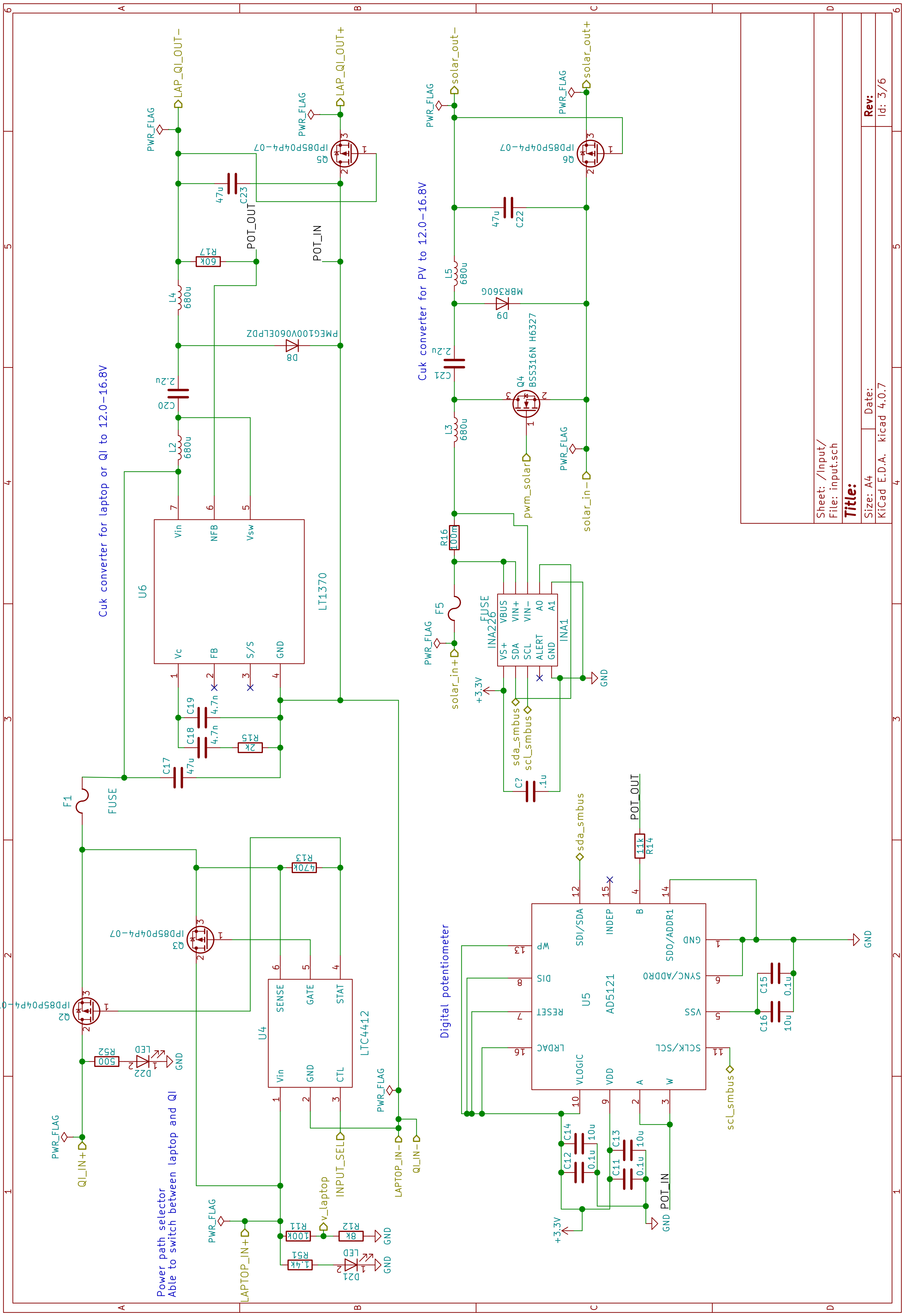
The following pages contain all kicad schematics, rendered to pdf format.

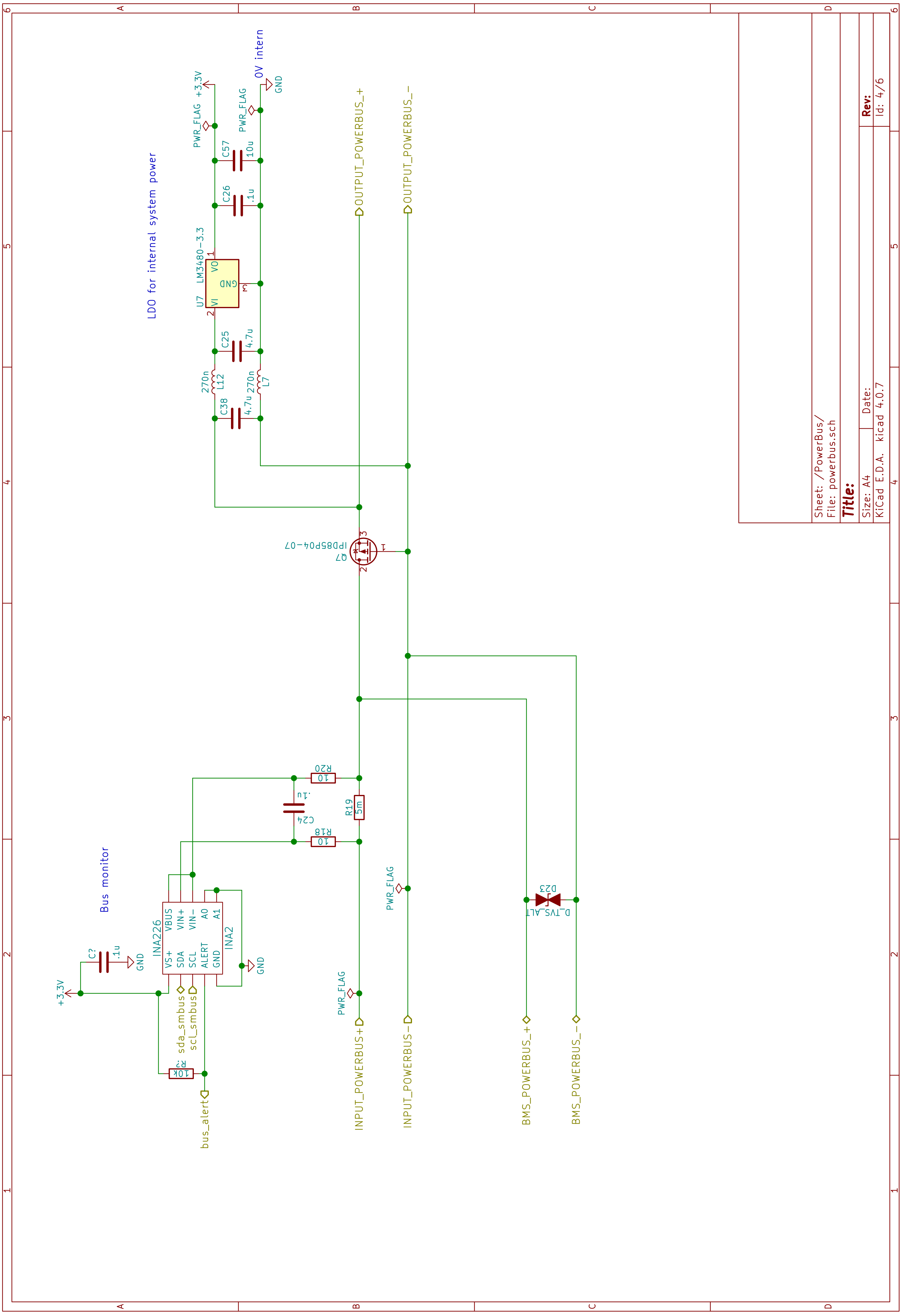


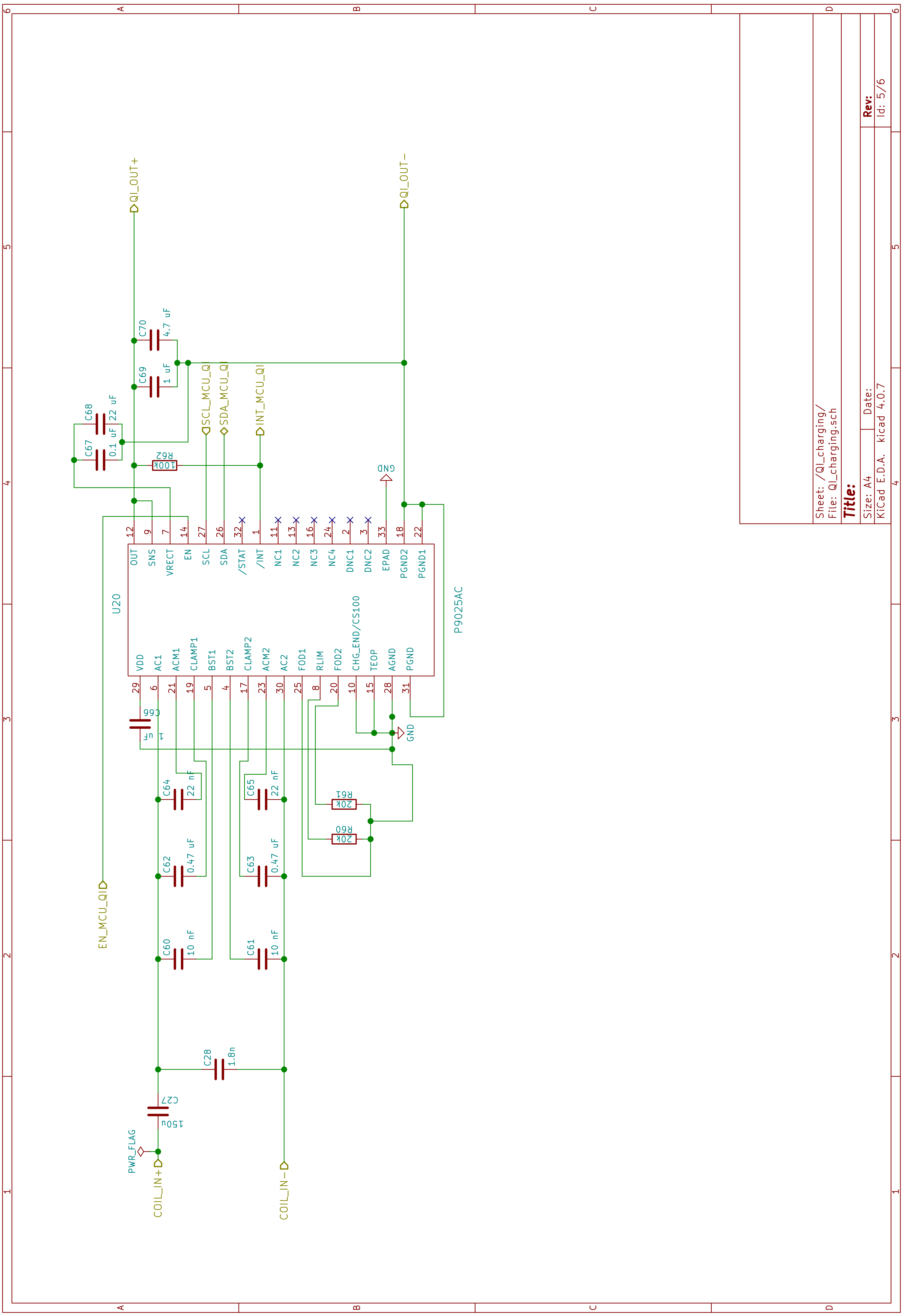
Output
SDA_SMBUS
SCL_SMBUS
OC_16V
OC_5V
OC_3.3V
16V+
16V-
5V+
5V-
3.3V+
3.3V-
3.3V_fuse
5V_fuse
16V_fuse
3.3V_enable
5V_enable
16V_enable
POWER_BUS+
POWER_BUS-
output.sch
J11
16V_2
J12
16V_1
J9
5V
J10
3.3V
D19
D20
D18
D_TVS_ALT
J8
Switch_Connector
On/off switch
J7
ZebroBus_Interface
ZebroBus Interface
GND
Control
PDI_DATA
PDI_CLK
TX
RX
ALERT
SCL_SMBUS
SDA_SMBUS
QI_enable
QI_int
pwm_solar
v_laptop
input_selec
INTERRUP_ZEBROBUS
SDA_ZEBROBUS
REF_ZEBROBUS
SCL_ZEBROBUS
OC_3.3V
OC_5V
OC_16V
3.3V_fuse
5V_fuse
16V_fuse
3.3V_enable
5V_enable
16V_enable
bus_alert
control.sch
PowerBus
bus_alert
sda_smbus
scl_smbus
OUTPUT_POWERBUS_+
OUTPUT_POWERBUS_-
INPUT_POWERBUS+
INPUT_POWERBUS-
BMS_POWERBUS_+
BMS_POWERBUS_-
powerbus.sch
J6
BMS_POWER_IN
BMS input
Input
sda_smbus
scl_smbus
pwm_solar
v_laptop
INPUT_SEL
QI_IN+
QI_IN-
LAP_QI_OUT+
LAP_QI_OUT-
LAPTOP_IN+
LAPTOP_IN-
solar_out+
solar_out-
solar_in+
solar_in-
input.sch
QI_charging
SDA_MCU_QI
SCL_MCU_QI
EN_MCU_QI
INT_MCU_QI
QI_OUT+
QI_OUT-
COIL_IN+
COIL_IN-
QI_charging.sch
D15
D16
D17
J1
Qi
Coil in
J3
Laptop interface
J13
Solar
J4
Debug_Interface
Debug Interface
+3.3V
J5
BMS_DATA_IN
BMS Interface
J14
J2
Mounting hole
Mounting hole
R27
1M
C35
0.1uF
R26
1M
C56
0.1uF
Sheet: /
File: main.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.7
Id: 1/6

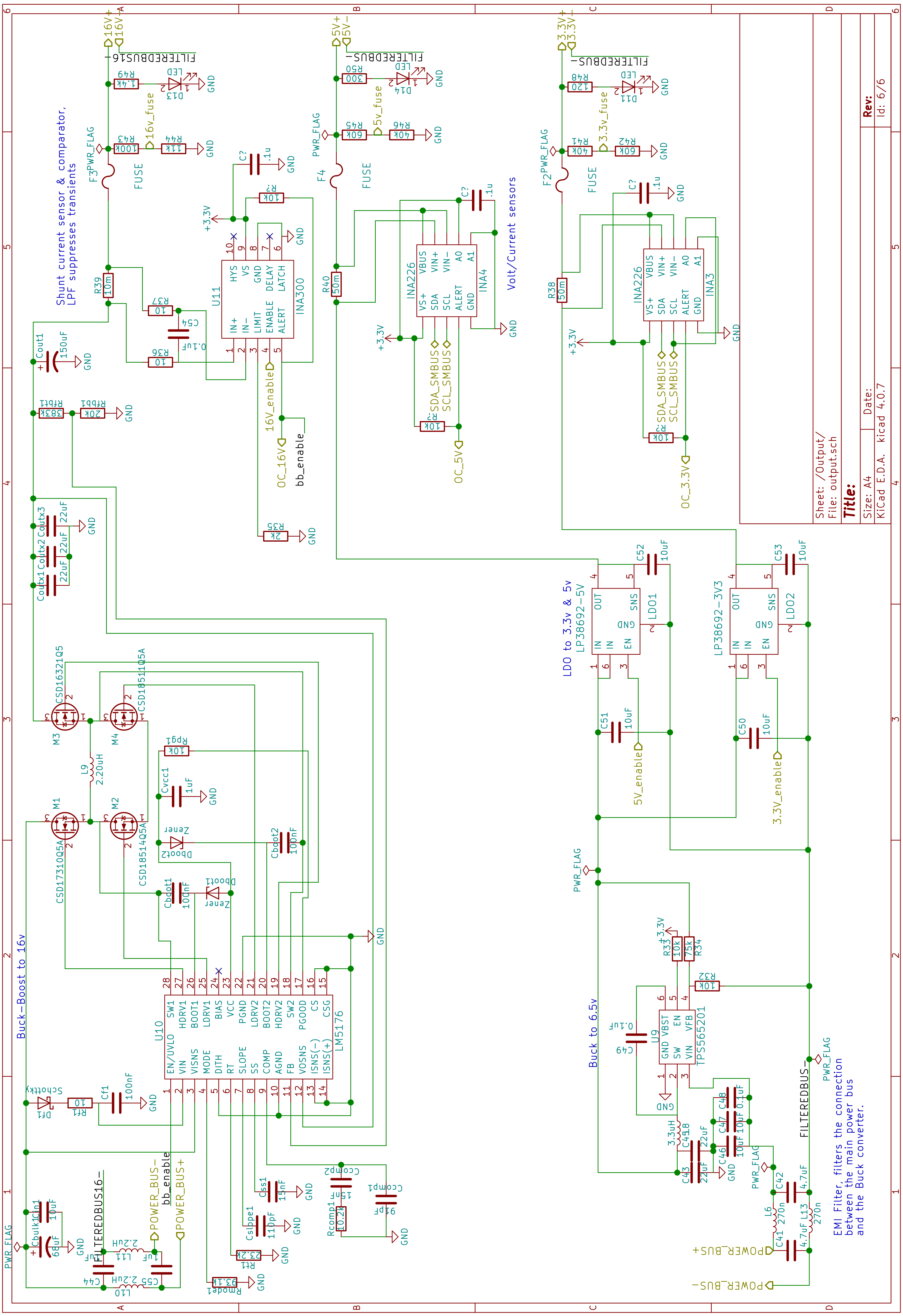


Analog input protection
3.3V_fuse
5V_fuse
16V_fuse
v_laptop
D3 D4 D5 D6 D_Zener
GND
ZebroBus buffer
REF_ZEBROBUS
C10 0.1u
SCL_ZEBROBUS
SDA_ZEBROBUS
U12 TCA9803
VCCA VCCB SCLA SCLB SDAA SDAB GND EN
C9 .1u
+3.3V
INTERRUP_ZEBROBUS
R10 100k
Q1 Si2302CDS
R9 2.7k
R8 2.7k
OC_3.3V
OC_5V
OC_16V
bus_alert
QI_int
input_selec
pwm_solar
U3 ATXMEGA32A4U-AU
C8 .1u
C7 10u
L1 Ferrite_Bead
C6 .1u
C5 .1u
C4 .1u
Backup power for 0.25s
C3 1m
D2 D_Schottky
PDI_CLK
PDI_DATA
3.3V_enable
5V_enable
16V_enable
ALERT
QI_enable
RX
TX
Debug LED
R47 120
D12 LED
R2 120
D1 LED
R6 10k
R5 10k
Internal communication bus
SDA_SMBUS
SCL_SMBUS
R4 10k
C2 0.1u
U2 LM75
R1 10k
R3 10k
C1 0.1u
U1 LM75BD
2 Temp sensors close to input/ouput converters
Sheet: /Control/
File: control.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.7
Id: 2/6


Cuk converter for laptop or QI to 12.0-16.8V
Cuk converter for PV to 12.0-16.8V
Power path selector
Able to switch between laptop and QI
Digital potentiometer
U6
LT1370
Vin
NFB
Vsw
Vc
FB
S/S
GND
U4
LTC4412
SENSE
GATE
STAT
CTL
U5
AD5121
SDI/SDA
INDEP
SDO/ADDR1
WP
DIS
RESET
LRDAC
VLOGIC
VDD
SYNC/ADDR0
VSS
SCLK/SCL
INA1
INA226
VBUS
VIN+
VIN-
A0
A1
VS+
SDA
SCL
ALERT
Q2 IPD85P04P4-07
Q3 IPD85P04P4-07
Q5 IPD85P04P4-07
Q6 IPD85P04P4-07
Q4 BSS316N H6327
D8 PMEG100V060ELPDZ
D9 MBR360G
D21 LED
D22 LED
L2 680u
L3 680u
L4 680u
L5 680u
C11 0.1u
C12 0.1u
C13 10u
C14 10u
C15 0.1u
C16 10u
C17 47u
C18 4.7n
C19 4.7n
C20 2.2u
C21 2.2u
C22 47u
C23 47u
C? .1u
R11 100K
R12 8K
R13 470K
R14 11k
R15 2K
R16 100m
R17 60K
R51 1.4K
R52 500
F1 FUSE
F5 FUSE
PWR_FLAG
QI_IN+
QI_IN-
LAPTOP_IN+
LAPTOP_IN-
LAP_QI_OUT+
LAP_QI_OUT-
INPUT_SEL
v_laptop
POT_IN
POT_OUT
solar_in+
solar_in-
solar_out+
solar_out-
pwm_solar
sda_smbus
scl_smbus
+3.3V
GND
Sheet: /Input/
File: input.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad 4.0.7
Rev:
Id: 3/6


LDO for internal system power
0V intern
GND
PWR_FLAG
+3.3V
C57
10u
C26
.1u
U7
LM3480-3.3
VI
VO
GND
C25
4.7u
270n
L12
C38
L7
OUTPUT_POWERBUS_+
OUTPUT_POWERBUS_-
Q7
IPD85P04-07
R20
10
C24
.1u
R19
5m
R18
10
Bus monitor
INA226
INA2
VS+
SDA
SCL
ALERT
VBUS
VIN+
VIN-
A0
A1
C?
R?
10K
sda_smbus
scl_smbus
bus_alert
INPUT_POWERBUS+
INPUT_POWERBUS-
BMS_POWERBUS_+
BMS_POWERBUS_-
D23
D_TVS_ALT
Sheet: /PowerBus/
File: powerbus.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.7
Id: 4/6


Sheet: /QI_charging/
File: QI_charging.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.7
Id: 5/6
U20
P9025AC
OUT
SNS
VRECT
EN
SCL
SDA
/STAT
/INT
NC1
NC2
NC3
NC4
DNC1
DNC2
EPAD
PGND2
PGND1
VDD
AC1
ACM1
CLAMP1
BST1
BST2
CLAMP2
ACM2
AC2
FOD1
RLIM
FOD2
CHG_END/CS100
TEOP
AGND
PGND
QI_OUT+
QI_OUT-
SCL_MCU_QI
SDA_MCU_QI
INT_MCU_QI
EN_MCU_QI
GND
GND
C70 4.7 uF
C69 1 uF
C68 22 uF
C67 0.1 uF
R62 100k
C66 1 uF
C64 22 nF
C65 22 nF
C62 0.47 uF
C63 0.47 uF
C60 10 nF
C61 10 nF
R61 20k
R60 20k
C28 1.8n
C27 150u
PWR_FLAG
COIL_IN+
COIL_IN-
A
B
C
D
1
2
3
4
5
6


Sheet: /Output/
File: output.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.7
Id: 6/6
Shunt current sensor & comparator,
LPF suppresses transients
Volt/Current sensors
LDO to 3.3v & 5v
LP38692-5V
LP38692-3V3
Buck-Boost to 16v
Buck to 6.5v
EMI Filter, filters the connection
between the main power bus
and the Buck converter.
FILTEREDBUS16-
FILTEREDBUS-
16V+
16V-
5V+
5V-
3.3V+
3.3V-
PWR_FLAG
FUSE
F3
F4
F2
16v_fuse
5v_fuse
3.3v_fuse
LED
D13
D14
D11
R49 1.4K
R43 100K
R44 11K
R50 300
R45 60K
R46 40K
R48 120
R41 40K
R42 60K
R39 10m
R40 50m
R38 50m
R37 10
R36 10
C54 0.1uF
R? 10K
C? .1u
U11
INA300
HYS VS GND DELAY LATCH
IN+ IN- LIMIT ENABLE ALERT
INA226
INA4
INA3
VBUS VIN+ VIN- A0 A1
VS+ SDA SCL ALERT GND
SDA_SMBUS
SCL_SMBUS
16V_enable
bb_enable
OC_16V
OC_5V
OC_3.3V
R35 2k
+3.3V
GND
Cout1 150uF
Rfbt1 383K
Rfbb1 20K
Coutx1 22uF
Coutx2 22uF
Coutx3 22uF
M3 CSD16321Q5
M4 CSD18511Q5A
M1 CSD17310Q5A
M2 CSD18514Q5A
L9 2.20uH
Rpg1 10K
Cvcc1 1uF
Zener
Dboot2
Cbdot2 100nF
Cbdot1 100nF
Dboot1
U10
LM5176
EN/UVLO VIN VISNS MODE DITH RT SLOPE SS COMP AGND FB VOSNS ISNS(-) ISNS(+)
SW1 HDRV1 BOOT1 LDRV1 BIAS VCC PGND LDRV2 BOOT2 HDRV2 SW2 PGOOD CS CSG
Schottky
Df1
Rf1 10
Cf1 100nF
Cin1 10uF
Cbulk1 68uF
POWER_BUS-
POWER_BUS+
L11 2.2uH
L10 2.2uH
C44 1uF
C55 1uF
Rmode1 93.1K
Rt1 23.2K
Cslope1 110pF
Css1 15nF
Ccomp2 15n
Rcomp1 10.2K
Ccomp1 91pF
C52 10uF
C53 10uF
C51 10uF
C50 10uF
LDO1
LDO2
IN IN EN GND OUT SNS
5V_enable
3.3V_enable
R33 10K
R34 75K
R32 10K
C49 0.1uF
U9
TPS565201
GND SW VIN VBST EN VFB
L8 3.3uH
C45 22uF
C43 22uF
C46 10uF
C47 10uF
C48 0.1uF
L6 270n
L13 270n
C41 4.7uF
C42 4.7uF

C
CODE

The code has been developed in the private Zebro Github repository, so the files discussed in this thesis are copied to an open Github. See: `https://github.com/nfakkel/DeciZebroPowerSystemControl`
This includes:

- Schematics - kicad project
- Microcontroller Firmware - C Code
- Statistical Analysis - Python Code

Bibliography

[1] S. Notani and S. Bhattacharya, *Flexible electrical power system controller design and battery integration for 1u to 12u cubesats,* in *2011 IEEE Energy Conversion Congress and Exposition* (2011) pp. 3633–3640.

[2] O. Knut, *Design and Implementation of the Electrical Power System for the CubeSTAR Satellite*, Master's thesis, University of Oslo (2013).

[3] M. Oredsson, *Electrical Power System for the CubeSTAR Nanosatellite*, Master's thesis, University of Oslo (2010).

[4] C. Dehong, Y. Xingang, and C. Jia, *A program about power management system for service robot,* in *2017 29th Chinese Control And Decision Conference (CCDC)* (2017) pp. 4912–4915.

[5] M. Saleh, Y. Esa, and A. Mohamed, *Centralized control for dc microgrid using finite state machine,* in *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)* (2017) pp. 1–5.

[6] P. D. Vaere and D. Booms, *ZebroBus Standard Specification*, 1st ed. (2017).

[7] B. Laumeister, *Use a twist (and other popular wires) to reduce emi/rfi,* (2012).

[8] J. Williams, *High efficiency linear regulators,* (1989), [Accessed June 11, 2018].

[9] B. University, *Bu-410: Charging at high and low temperatures,* .

[10] A. P. Brokaw, *A simple three-terminal ic bandgap reference,* IEEE Journal of Solid-State Circuits **9**, 388 (1974).

[11] *LM75B Digital temperature sensor and thermal watchdog*, NXP (2015), rev. 6.1.

[12] S. Ziegler, R. C. Woodward, H. H. C. Iu, and L. J. Borle, *Current sensing techniques: A review,* IEEE Sensors Journal **9**, 354 (2009).

[13] *Datasheet ATxmega32A4U*, Atmel (2014).

[14] *XMEGA AU MANUAL*, Atmel.

[15] Y. Sheng, C. Xikun, S. Dong, and L. Qinhuang, *Design of lithium battery pack fault diagnosis system based on two levels state machine,* in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)* (2013) pp. 1929–1932.

[16] O. Singh and S. K. Gupta, *A review on recent mppt techniques for photovoltaic system,* in *2018 IEEMA Engineer Infinite Conference (eTechNxT)* (2018) pp. 1–6.

[17] D. P. Hohm and M. E. Ropp, *Comparative study of maximum power point tracking algorithms,* Progress in Photovoltaics: Research and Applications **11**, 47 (2003).

[18] J. J. Nedumgatt, K. B. Jayakrishnan, S. Umashankar, D. Vijayakumar, and D. P. Kothari, *Perturb and observe mppt algorithm for solar pv systems-modeling and simulation,* in *2011 Annual IEEE India Conference* (2011) pp. 1–6.

[19] *bq40z60 Programmable Battery Pack Manager* (2017).

[20] *System management bus specification,* .

[21] NXP, *I2c-bus specification and user manual,* .

[22] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 5th ed. (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2013) Chap. 4.

[23] *INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor with I2C Compatible Interface*, Texas Instruments.