

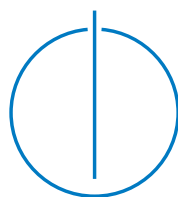


DEPARTMENT OF INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

**Scalable and Modular Architecture for
Self-organizing Power Systems on Small
Spacecrafts**

Florian Mauracher





DEPARTMENT OF INFORMATICS

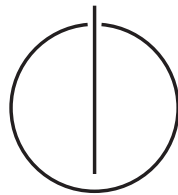
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

**Scalable and Modular Architecture for
Self-organizing Power Systems on Small
Spacecrafts**

**Skalierbare und modulare Architektur
für selbstorganisierende
Stromversorgungssystem auf kleinen
Raumfahrzeugen**

Author: Florian Mauracher
Submission Date: 15. February 2019
Supervisor: Prof. Dr. Martin Schulz
Advisors: M.Sc. Dai Yang
M.Sc. Florian Schummer
M.Sc. Sebastian Rückerl



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Garching, 15. February 2019

Florian Mauracher

Abstract

This thesis describes the architecture and coordination algorithms of a modular power system for small spacecrafts. The system is currently being developed at the Chair of Astronautics (LRT) of the Technical University of Munich (TUM) in cooperation with the University of New South Wales (UNSW) Canberra.

Based on the principles of distributed systems the power system consists of multiple boards. While each board is able to operate independently, multiple boards can be joined together to extend the capabilities of a single board system in regards to power generation, power storage and power delivery as well as increasing reliability and fault tolerance.

Contents

1	Introduction	1
1.1	EPS for CubeSats	2
1.2	UNSW Mission	3
1.3	Goals of this Thesis	5
2	EMPS System	6
2.1	System Architecture	6
2.2	Requirements	7
2.2.1	UNSW M2 Mission	7
2.2.2	Basic EPS	8
2.2.3	Distributed EPS	8
3	Related Work	10
3.1	Coordination Algorithms	10
3.1.1	Leader Election	10
3.1.2	Distributed Shared Memory	11
3.2	Summary	12
4	Software System Design	13
4.1	Algorithm Selection	13
4.1.1	Shared State	13
4.2	Interfaces	13
4.2.1	Configuration	13
4.2.2	Telemetry	14
4.3	External Interface	15
4.4	Coordination	16
4.4.1	Shared State Update	16
4.4.2	Switch Selection Algorithm	17
4.5	Error Case Analysis	18
5	Evaluation	21
5.1	Functional Testing	21
5.1.1	Single Board Test Cases	21
5.2	Multi-Board Test Setup	23
5.2.1	Multi-Board Test Cases	23
5.2.2	Error Cases	25
5.3	Spacecraft Integration Testing	25
5.4	Power Consumption	27
5.5	Summary	28
6	Conclusion and Outlook	29

Bibliography	32
A Appendix	34
A.1 Data Structures	34

1 Introduction

A CubeSat is a miniaturized satellite with fairly restricted dimensions and mass that still provides similar functionalities and capabilities as bigger spacecrafts [1]. The Munich Orbital Verification Experiment II (MOVE-II) satellite is a CubeSat that was developed and built by students at the Technical University of Munich (TUM) in a cooperation between the Chair of Astronautics (LRT) and the Scientific Workgroup for Rocketry and Spaceflight (WARR) [2]. It was launched on 03.12.2018 on board of a Falcon 9 rocket from Vandenberg in California and is currently being operated by students. The assembled engineering model of the satellite is visible in Figure 1.1 while Figure 1.2 shows a top view of the solar cells.

Enabled by the miniaturization of electronic components these CubeSats reduce the entry barrier to space for private and public organizations. Nowadays a one Unit (1U) CubeSat with dimensions of $10 \times 10 \times 10$ cm and a mass of 1.33 kg is able to contain a miniaturized version of all essential systems that are required for its survival and operation in an orbit around the earth.

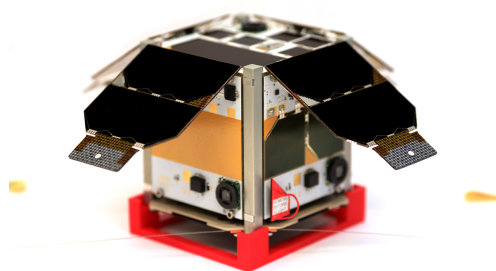


Figure 1.1: *Assembled engineering model of the 1U MOVE-II CubeSat*

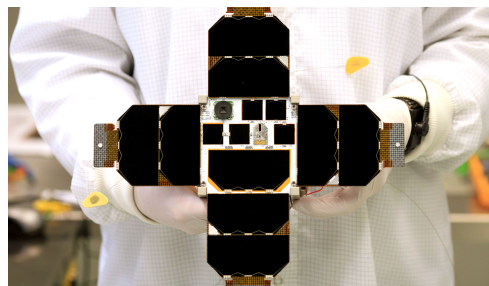


Figure 1.2: *Top view of the engineering model of the MOVE-II CubeSat*

The *CubeSat Design Specification* [3] standardizes the dimensions, weight and required tests for a CubeSat to be launched into space. This standardized platform for small satellites reduces the coordination effort with the launch provider and in turn lowers the launch costs for these satellites. With lower development and launch costs the number of small spacecrafts has increased significantly in recent years [4].

One of the main challenges in building these small spacecraft is the high degree of integration between all systems. In order to increase component reuse in small spacecrafts the different tasks are often organized in subsystems. Nevertheless to achieve a high performance density, subsystems are usually specifically adapted to every mission, which leads to high failure rates [5][6].

One approach to resolve adaptability and reliability issues at the same time is the use of subsystems that are modular and scalable. A subsystem that is designed as a modular and scalable system from ground up can be easily adapted to the individual mission requirements without significant adjustments and thus rely on the flight heritage of previous missions using the same modular system.

1.1 EPS for CubeSats

An Electrical Power System (EPS) is an essential component of basically all spacecrafts. It provides means to harvest energy through solar cells, temporarily store the energy and provide power to other systems of the spacecraft. Additionally it provides telemetry data of the current system state and sensor measurements while handling incoming control commands. These basic EPS inputs and outputs are visible in Figure 1.3.

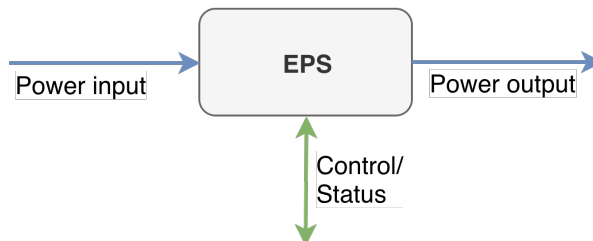


Figure 1.3: *EPS basic interfaces*

Generally EPS feature several key components internally as visualized in Figure 1.4. A Battery charge regulator (BCR) controls the incoming energy from the solar cells to optimizes their efficiency through Maximum power point tracking (MPPT) and handles the charging of the battery. Several DC-to-DC converters (DCDCs) provide different output voltages for other subsystems which are connected through multiple switches in order to individually switch each subsystem. A microcontroller is responsible for controlling the input and output electronics while several voltage, current and temperature sensors allow monitoring of the system's environment. Often the system features mechanisms to react to changes in the environment. An example for this would be the battery temperature rising above a defined threshold causing the system to disable non essential subsystems in order to reduce the load on the battery. An external digital communications interface exposes the system's telemetry and accepts commands to configure and change the state of the system. This interface is commonly connected to the satellite's On-board Data Handling (OBDH) system and thus the term OBDH will be used for the rest of this thesis when referring to the external system controlling the EPS.

Most of the CubeSat EPS in use today use a centralized design with a single board [7][8][9]. While a centralized design decreases complexity it has several downsides. Often systems need to be adjusted to fit the specific mission requirements decreasing reusability where each change could potentially introduce new errors in a flight proven system. More critically a centralized design acts as a single point of failure.

In the MOVE-II mission the EPS is one of two subsystems that were bought from a commercial CubeSat component supplier. All other subsystems were designed and built by students of the TUM [2].

From the experience gathered during the MOVE-II project a new project was devised to develop a scalable and modular EPS. By following the principles of distributed computing, fault tolerance can be increased and improved flexibility to mission requirements can be provided.

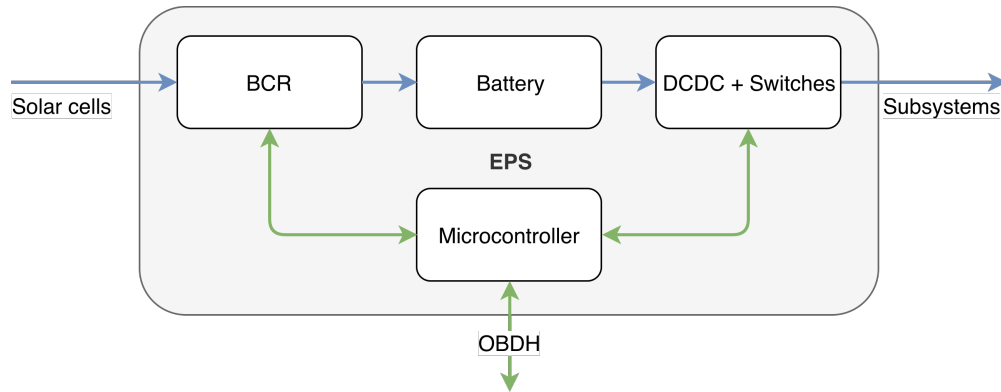


Figure 1.4: EPS internal components

1.2 UNSW Mission

Based on a cooperation between the University of New South Wales (UNSW) Canberra Space and the LRT at TUM since 2017 a common interest in the development of a EPS for small spacecrafts was identified.

The UNSW offered to provide a launch opportunity as part of their upcoming UNSW M2 mission for a EPS developed by TUM in cooperation with the TUM spin-off Orbital Oracle Technologies GmbH (orora.tech). The UNSW M2 mission consists of two six Unit (6U) CubeSats Each measuring $30 \times 20 \times 10$ cm joined together on launch and separated at some later point in the mission as visible in Figure 1.5 and Figure 1.6.

While the M2 satellite relies on a commercial EPS for its primary mission, a single board of the EPS developed by TUM will be part of the experimental payload on both spacecrafts.

The objectives of this cooperation and the resulting Extendable modular power supply (EMPS) project can be summarized by the following goals derived from the original collaboration agreement [10]:

- Flight qualify an Electrical Power System, for future CubeSat missions of the TUM and associated parties (orora.tech, UNSW).
- Build a demonstration board to fly on the two M2 Satellites with all necessary functionality to qualify the system in space to prove thermal and radiation resistance of the system over time as well as resilience to the launch environment.
- Qualify the architecture with multiple boards of the system on ground with further functional and vacuum testing.
- Educate students and build and demonstrate capabilities in spacecraft design at the Technical University of Munich.
- Foster and deepen the cooperation between the TUM's Chair of Astronautics and UNSW Canberra Space, with the goal to exchange expertise, make use of synergies and enable more demanding missions for both universities.

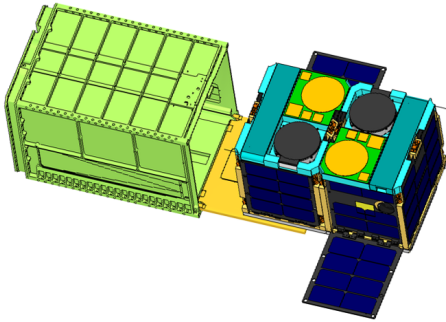


Figure 1.5: UNSW M2 satellite with CubeSat dispenser[10]

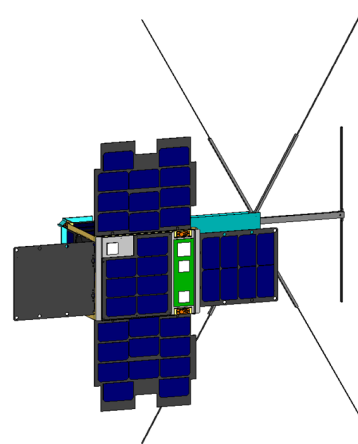


Figure 1.6: UNSW M2 satellite in deployed state[10]

The EMPS team consists of 5 people of which one is responsible for the electronics design and manufacturing [11] while two software engineers develop the software for the embedded system along with the necessary infrastructure for debugging, testing and analysis. Development of the software is structured in the following manner: Both software developers collaborate on the implementation of the hardware abstraction layer for the embedded system. The single-board interface and coordination of the interface with UNSW is specifically assigned to one developer [12], while the author of this thesis is responsible for the high level software algorithms for the distributed system. The original concept for a distributed EPS integrating electronics, battery and a solar cell in a single component was developed by another team member in his masters thesis [13].

As common with spaceflight projects Figure 1.7 depicts the mission patch designed for the EMPS project.

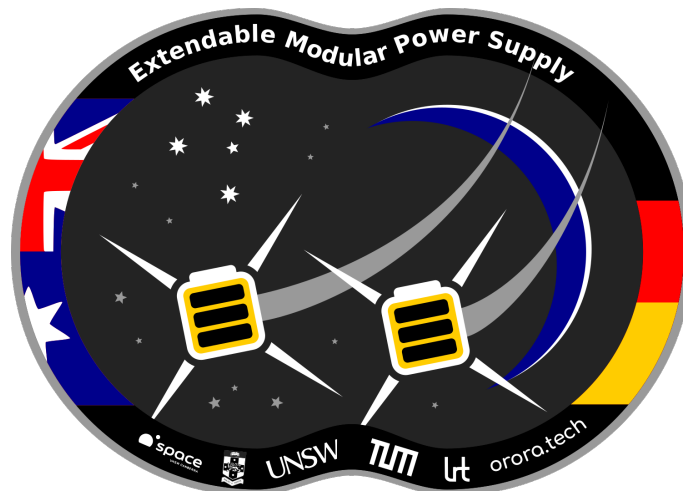


Figure 1.7: Mission patch of the EMPS project

1.3 Goals of this Thesis

This thesis describes the development of the software system for an EPS, specifically focusing on the distributed concepts enabling a scalable, modular and robust system.

The background and environment of the surrounding project is described in Chapter 2 along with the requirements imposed on the designed system.

This involves the analysis and selection of a suitable communication protocol and the design and implementation of a coordination and decision algorithm that fulfills the requirements for redundant mission operations in Chapter 3 and Chapter 4. An analysis of possible system states and error cases is performed in Section 4.5 to ensure the system always behaves in an expected manner.

While the development of the underlying embedded system along with the coordination and adjustments for incremental hardware revisions are not a focus of this thesis, they are necessary for the overall EMPS project. Additionally they provide the foundation to develop the described coordination algorithm and validate its design and implementation through functional testing.

Chapter 5 describes this functional testing of the requirements and error cases to evaluate the system along with a description of the test and development environment.

Finally Chapter 6 summarizes the results and provides an outlook on possible future applications of the system.

2 EMPS System

This chapter describes the hardware architecture and environment of the EMPS system which serves as a foundation for this thesis.

2.1 System Architecture

In contrast to conventional centralized EPSs the EMPS board features a modular design with multiple identical boards. While each board operates independently, multiple boards can be joined together to extend the capabilities of a single board system in regards to power generation, power storage, power delivery, the number of switches as well as increasing reliability and fault tolerance.

Each board features a Texas Instruments (TI) MSP430 microcontroller with Ferroelectric RAM (FRAM) memory as a high reliability embedded processor with low power consumption [14]. In contrast to flash storage common in most microcontrollers the TI MSP430 series with FRAM memory provides improved radiation tolerance and Single event upset (SEU) tolerance [15][16] and has flight heritage from multiple CubeSat missions [17][4].

The selected MSP430FR5989-EP microcontroller features a 16-Bit Reduced instruction set computer (RISC) architecture with a 16 MHz frequency, 2 kB of main memory and 128 kB of FRAM for code and data storage. In contrast to increasingly common ARM-based microcontrollers with >100 MHz and multiple megabytes of flash memory, the resource limitations of this microcontroller need to be considered at all times during the development of the base system and the high level logic.

A common Controller Area Network (CAN) bus connects all EMPS boards and is also accessible for external systems to provide the telemetry, configuration and control interface.

Compared to logical point to point connections between individual boards a shared CAN bus simplifies the communication architecture as its message based design features an identifier in all messages that can be leveraged for topic based communication and multicast messaging. Each message identifier has a fixed priority and serves as a hardware collision avoidance mechanism ensuring real time delivery of critical messages.

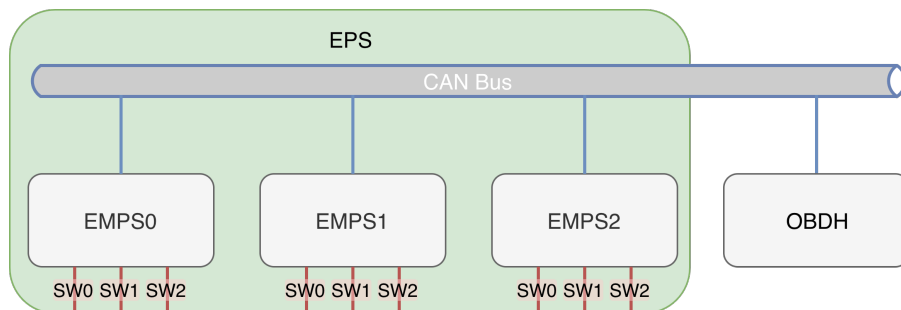


Figure 2.1: *EMPS multi-board setup with shared CAN bus*

Figure 2.1 shows an example of a EMPS multi-board setup with three boards connected

through a common CAN bus together with a OBDH. Every EMPS board features multiple switches to supply subsystems with power. Subsystems can be connected to switches of different EMPS boards when additional redundancy is desired for critical components.

While every EMPS board in this example features three switches that can be connected to subsystems the actual EMPS hardware features 6 switches. Two switch provide the unregulated battery voltage while two other switches provide 3.3V and 5V respectively, regulated through DCDC converters.

One additional DCDC converter provides a regulated voltage to the integrated digital circuitry, specifically the MSP430 microcontroller and the temperature, voltage and current sensors.

The *InterpanelCharge* and *InterpanelDischarge* connections are connected in a multi-board setup to distribute energy across the batteries of all boards.

Finally an external inhibit pin along with a physical switch electrically disconnects the battery during launch until deployment as required by the CubeSat Design Specification [3].

The described high level hardware architecture is visualized in Figure 2.1.

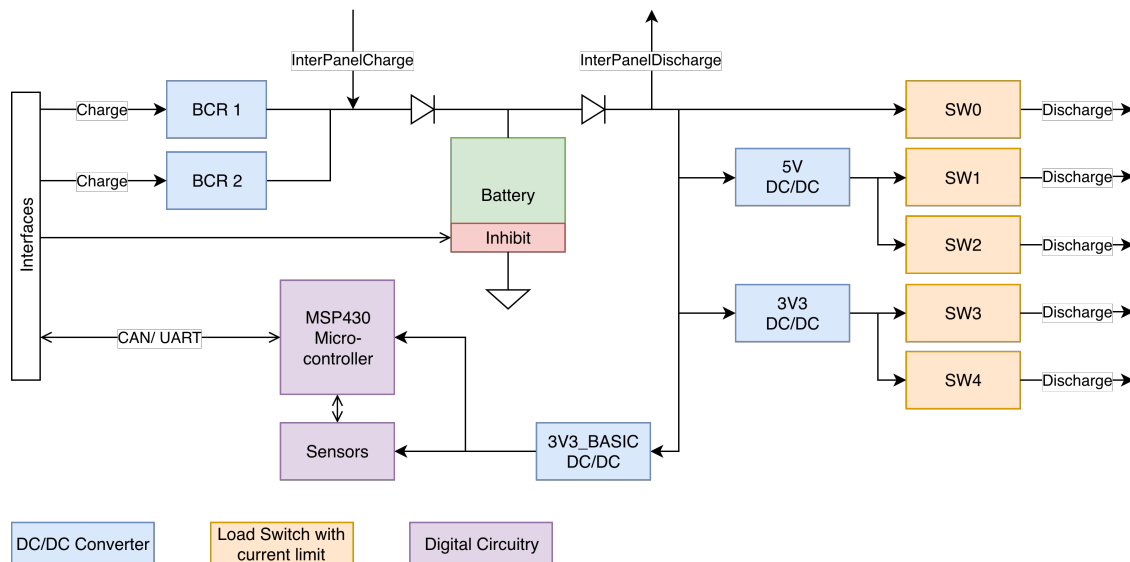


Figure 2.2: EMPS hardware architecture

2.2 Requirements

2.2.1 UNSW M2 Mission

For the UNSW mission an extensive list of formal requirements describing mechanical, electrical and interface properties was defined at the start of the cooperation with multiple adjustments during the projects duration, due to changes in the project scope and potential issues.

The UNSW mission deviates in certain aspects from the original EMPS project as it only consists of a single board and the external communication interface changed from

CAN to Universal Asynchronous Receiver/Transmitter (UART) due to time constraints on UNSW side.

None of the UNSW requirements are particularly relevant for the focus of this thesis and thus only a basic set of EPS requirements shall be listed in the following section.

2.2.2 Basic EPS

In contrast to conventional space mission small satellites often have lower requirements on availability and reliability [18]. Nevertheless a certain set of necessary minimal requirements exists to ensure the utility of the system as a small satellite EPS.

All requirements listed in the following use the requirement levels in accordance with RFC2119 [19]. The term *shall* refers to a mandatory requirement, while the term *should* refers to a recommended but not mandatory requirement.

R.1 Power input

The EPS shall be able to handle the maximum occurring input power in orbit.

R.2 Power storage

The EPS shall be able to store sufficient electrical power.

R.3 Power supply

The EPS shall be able to provide sufficient electrical power for all supplied subsystems.

R.4 Power channels

The EPS shall support separate power channels for connected subsystems where each channel can be switched independently.

R.5 Power monitoring

The EPS shall monitor individual power flows to all subsystems.

R.6 Overcurrent detection

The EPS shall keep each subsystem within its respective electrical operational limits.

R.7 Overcurrent shutdown

The EPS shall be able to shut down selective power channels in case of overload.

R.8 Telemetry and control

The EPS shall have a data exchange and command interface with OBDH.

R.9 Power flow telemetry

The EPS shall provide knowledge about the power flows to the OBDH.

R.10 Electrical inhibit

The EPS shall be able to separate power from the system for the time between final integration and ejection of the satellite from the launcher.

2.2.3 Distributed EPS

For distributed operation of the EMPS the following requirements were identified:

RD.1 Transparent external interface

To the OBDH controlling the EMPS the distributed system shall handle access to the system in a transparent manner. Regardless of the number of interconnected EMPS boards the external interface should remain consistent and the system can be addressed as a whole instead of communicating with each board individually.

RD.2 Single supply

Due to electrical limitations and to enable accurate current measurements only a single switch should be active during normal operation for subsystems connected to multiple EMPS boards.

RD.3 Failover

In case of a failure on one EMPS board subsystems connected to multiple boards shall be supplied by one of the other connected EMPS boards without external intervention. A short interruption in the supply during the failover is acceptable causing all other EMPS boards to act as hot standby systems.

RD.4 Number of boards

The restrictions in available volume and mass in small spacecrafts result in a practical upper bound for the number of simultaneously connected boards. Thus the system shall be able to handle up to 16 boards connected together.

3 Related Work

In order to select appropriate mechanisms for coordination within the system presented in Chapter 2 and to fulfill the requirements defined in Section 2.2 existing concepts and solutions in the area of distributed computing were analyzed.

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system [20].

With advances in processor technology and improvements in network bandwidth distributed systems have become increasingly common in recent years and are a common approach to improve the reliability of critical systems. Nevertheless distributed systems remain a difficult area of research partly due to the multifaceted nature of these systems. Many different approaches and algorithms for communication and coordination within these systems exist each with their own advantages and disadvantages.

3.1 Coordination Algorithms

A critical component for every distributed system is the coordination between the individual nodes of the system [21]. This coordination is required to maintain the correctness of applications in distributed environments.

Coordination algorithms address various requirements in distributed systems. One of these conditions is mutual exclusion. Distributed mutual exclusion means that only one process accesses a resource at any given time to safeguard multiple processes against parallel access [22]. This requires creation of a critical section or region represented as a code where to enforce mutual exclusion. The central assumption in distributed mutual exclusion is that message delivery occurs reliably and without fail within a single critical region. In operating systems, mutual exclusion involves semaphores and atomic actions. In distributed systems, a mutual algorithm must meet three primary requirements or conditions: safety, liveness, and ordering. Safety means that only one process can execute in the critical region at any time. Liveness condition means freedom granted for requests to enter and exit the critical region. Lastly, ordering means systematic access for enhanced coordination.

3.1.1 Leader Election

One common approach to ensure consistency within a distributed system is the election of a single leader or coordinator [23]. A leader election algorithm describes the required steps to ensure that a system of nodes is able to transition from a state where nodes are unaware of or unable to communicate with a single coordinator to a state where all nodes agree on a single leader. After a leader election algorithm has been run, each node throughout the system recognizes a particular, unique node as the leader.

This elected coordinator can then act as central authority and ensures a consistent state within the system as all changes to the state need to be confirmed by this leader. These algorithms are based on the assumption that each process knows the election values of all the processes, but without knowledge of the processes being executed.

While many leader election algorithms exist, the Bully and Ring algorithms present two common solutions to this problem [24].

In the Bully Algorithm a process initiates election of a new coordinator when it notices no response in the existing coordinator or upon receiving an election message.

The Ring Algorithm views processes as a ring and assumes that processes know their successors. In the initial step, each process is assigned a non-participant status, and then the process that assumes the highest identifier in the logical ring becomes the leader.

Additional variations of these algorithms exist to simplify message passing or reduce bandwidth consumption by minimizing the number of message passing to the elected leader [23].

Conclusion

While these algorithms offer solutions to the problem of electing a leader in a distributed system, they introduce significant complexity and communication overhead. Even after a central leader is elected every change in the system requires multiple messages between the nodes and the leader.

3.1.2 Distributed Shared Memory

One approach to enable coordination between multiple processes within a system is the use of shared memory for coordination. The shared memory provides a consistent state across all nodes which enables decisions based on the memory contents.

Distributed Shared Memory (DSM) applies the shared memory approach for coordination to distributed systems. An abstraction layer ensures that each node has its own copy of the shared memory and can act solely based on the state contained in this memory [25].

Figure 3.1 shows a distributed system consisting of three nodes where a *Mapping Manager* act as the abstraction layer that synchronizes a portion of the nodes own memory with the other nodes. The complexity and communication overhead of the synchronization algorithm increases with the strictness of memory consistency requirements imposed by the application accessing the memory.

Conclusion

The distributed shared memory approach provides the convenience of shared memory programming in a distributed architecture [26].

Its primary downside is the significant overhead associated with the synchronization of shared memory across multiple systems [20]. Every change in the shared memory must be tracked and all nodes need to be notified.

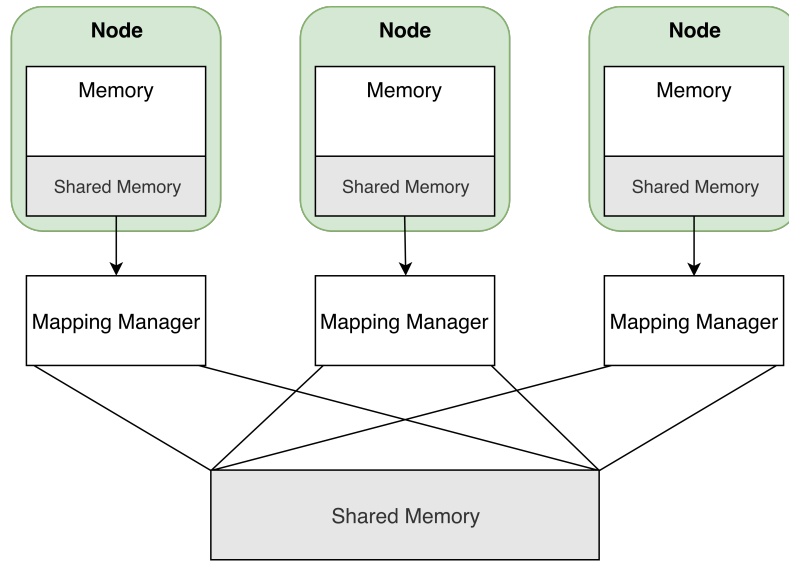


Figure 3.1: *Distributed shared memory*

3.2 Summary

While none of these approaches provides an optimal solution to coordination in distributed systems, they provide a foundation for a coordination algorithm adapted to the specific requirements of the project which will be described in the following chapter.

4 Software System Design

4.1 Algorithm Selection

As described in Chapter 3, many coordination algorithms in distributed systems rely on complex and communication intensive message passing techniques to ensure a consistent system state.

With the relaxed consistency requirements from Section 2.2 in combination with the properties of the target system a more lightweight solution can be constructed to coordinate between the EMPS boards.

4.1.1 Shared State

By leveraging the communication properties of the common CAN bus along with additional information sources, a consistent shared state can be maintained between the boards where each board holds a copy of this state in its main memory. This shared state contains all relevant information so every board can act independently solely relying on the information contained in this data while still fulfilling the requirements from Section 2.2.

This system approximates the DSM model described in Section 3.1.2 but applies a certain set of restrictions that facilitate the synchronization between nodes.

Most importantly every board only writes to a small section of its shared memory/ state (the *BoardTelemetryShared* as described in Section 4.2.2) and treats the rest of its shared memory as read-only that is only updated by the *Mapping Manager* when updates from other boards are received.

Additionally instead of synchronizing every change to the board's writable section of the shared state, the boards state is only shared periodically with the other boards.

Finally the CAN bus in combination with an abstraction on top of the CAN bus driver provides a convenient interface to exchange multicast messages between the boards [12]. This further reduces the number of messages as the boards only need to send a single message to update their own state in all other boards.

4.2 Interfaces

When initially flashing the firmware every board is assigned a unique identifier (*BoardID*) which allows identification of state updates from different boards. After a board receives a new state update message it saves the state of this *BoardID* in it's own shared state data structure.

4.2.1 Configuration

In addition to a unique identifier and the shared state the boards store a configuration data structure that contains information on which subsystems are connected to the switches on all boards (*BoardConfig*) and the expected properties of each subsystem (*SubsystemConfig*).

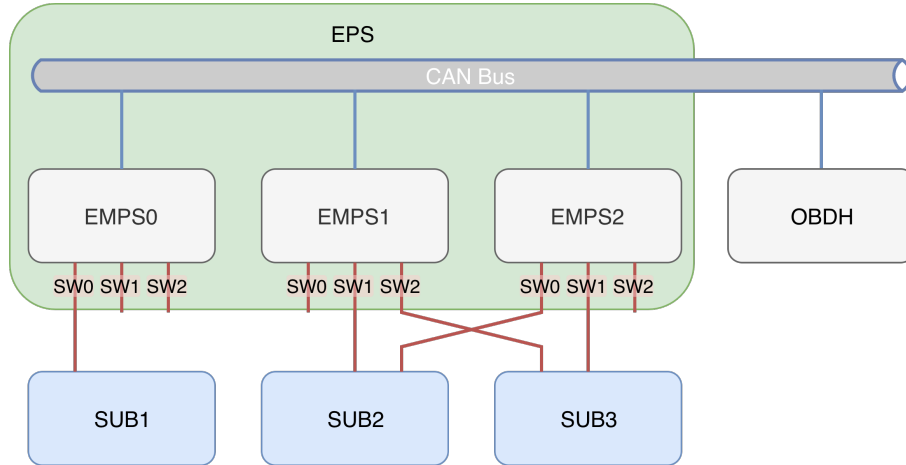


Figure 4.1: EMPS multi-board setup with connected subsystems

Figure 4.1 shows a possible setup with three EMPS boards and three subsystems. Subsystem 1 (*SUB1*) is only connected to *SW0* of *EMPS0* and thus relies on *EMPS0* for power supply. In contrast *SUB2* and *SUB3* are connected to switches on two different EMPS boards each and can thus be supplied from either one of those boards. The resulting mapping is shown in Table 4.1 and assigns every board and switch to either a subsystem or as not connected (NC). The configuration additionally contains a bitmask to mark individual sensors each board as untrusted in case a component should fail.

The second part of the configuration contains information about every subsystem. A Boolean for the *initial state* determines whether a switch for the subsystem should be enabled by default and allows critical subsystems to be enabled without an external command. A *timeout* value controls if the system should be powered constantly when enabled (value 0) or only for a certain period of time (value in seconds). If desired a maximum current can be configured that, when exceeded, sets an error flag in the subsystems telemetry and disables the switch.

The memory layout of the resulting board and subsystem configuration data structures is visualized in Table A.2 and Table A.3.

Board	Switch	Subsystem
EMPS0	SW0	SUB1
EMPS0	SW1	NC
EMPS0	SW2	NC
EMPS1	SW0	NC
EMPS1	SW1	SUB2
EMPS1	SW2	SUB3
EMPS2	SW0	SUB2
EMPS2	SW1	SUB3
EMPS2	SW2	NC

Table 4.1: Mapping from Board and switch to subsystem

4.2.2 Telemetry

The telemetry data structure contains all runtime state and sensor data that is collected by the system.

It is split into two parts, the *BoardTelemetry* (individual per board) and *GlobalTelemetry* (shared state). The *BoardTelemetry* contains the *BoardTelemetryShared* with the local system state and in a second part the sensor measurements and relevant debug data.

The *BoardTelemetryShared* is shared across boards as described in Section 4.1.1. The datastructure is visualized in Table A.1 and includes the following elements with relevant information for the other boards:

Version Protocol and firmware version information

BoardID The unique identifier of the board

Board count Number of active boards

Revision counter Counter to indicate changes in data structure

Uptime Time since the last reset

Board state Bitmask indicating the current board state

Switch Info An array containing the current state of all local switches and their assigned subsystem. The state contains information whether the switch is enabled or disabled as well as fault states. These fault states include a previously detected overcurrent on the switch and the presence of a voltage above a threshold while the switch is disabled.

Temperature The maximum temperature aggregated from all local temperature sensors

Configuration checksum A checksum of the *Configuration* structure

Telemetry checksum A checksum of this data structure except the checksum itself

Finally the *GlobalTelemetry* contains the aggregated shared state in the form of an array of *BoardTelemetryShared* structures indexed by the *BoardID*.

4.3 External Interface

The external interface provides the OBDH with measures to retrieve telemetry from the system, set configuration values and to enable or disable individual subsystems. The external interface is accessible through the shared CAN bus causing all EMPS boards to receive the command from the OBDH. When a command is received the execution of the EMPS microcontrollers is interrupted and the command is processed. Afterwards a single board sends back a response over the CAN bus.

It provides the following commands:

GetBoardTelemetry(*BoardID*, offset)

The board with the matching *BoardID* responds to the command with the content of the *BoardTelemetry* structure at the specified offset.

GetGlobalTelemetry(offset)

The board with the lowest available *BoardID* responds with the content of the *BoardTelemetry* structure at the specified offset.

GetSystemConfig(offset)

The board with the lowest available *BoardID* responds with the content of the *SystemConfig* structure at the specified offset.

SetSystemConfig(offset, value)

The SetSystemConfig command is processed by all boards and updates the value of the *SystemConfig* structure at the specified offset.

GetBoardConfig(offset)

The board with the lowest available *BoardID* responds with the content of the *SystemConfig* structure at the specified offset.

SetBoardConfig(offset, value)

The SetBoardConfig command is processed by all boards and updates the value of the *BoardConfig* structure at the specified offset.

SwitchSubsystem(SubsystemID, state)

The SwitchSubsystem command is processed by all boards where every board updates the requested state for the selected subsystem. Afterwards the switch selection algorithm as described in Section 4.4.2 is initiated.

4.4 Coordination

While all boards in the EMPS system are connected through the CAN bus the microcontroller on all boards run independently.

After startup the hardware and connected sensors are initialized. Afterwards the board continuously performs the processing loop visible in Figure 4.2 while simultaneously saving incoming state messages from other boards to the *GlobalTelemetry* data structure.

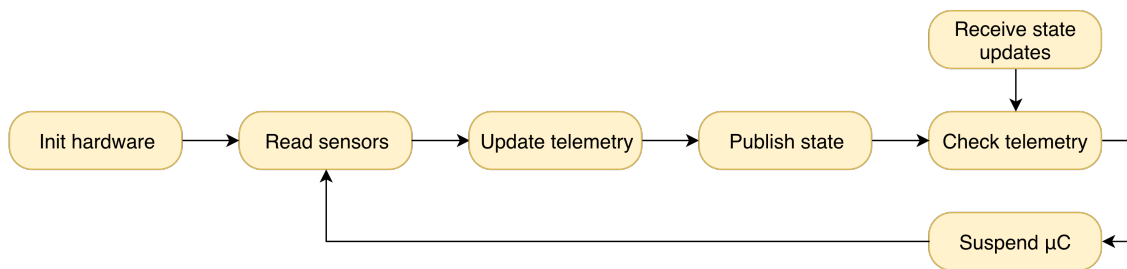


Figure 4.2: EMPS processing loop

Each iteration of the processing loop starts with reading the measurements from all connected temperature, voltage and current sensors. These measurements are then stored in the boards local telemetry along with information about the current system state like uptime and switch state information. Subsequently the current system state is shared with the other boards over the CAN bus. The contents of this state (the *BoardTelemetryShared*) are described in Section 4.2.2.

4.4.1 Shared State Update

In the next step the system iterates over the *GlobalTelemetry* structure to processes all received *BoardTelemetryShared* updates according to the following steps:

- If a board hasn't updated its state in the last 10 iterations it is ignored.
- Update the number of available boards according to the number of valid entries in the *GlobalTelemetry* structure.
- Compute the status (enabled or disabled) of all subsystem from switch information of all available boards.

- Compare the requested state with the current state and increase a subsystem configuration mismatch counter by 1.
- If the mismatch counter of a subsystem reaches 10 initiate the switch selection algorithm as described in Section 4.4.2.

Finally the microcontroller is suspended until 200ms have passed since start of the processing loop to ensure a consistent processing loop frequency of 5 Hz is reached.

4.4.2 Switch Selection Algorithm

The switch selection algorithm is executed in response to an external *SwitchSubsystem* command or when a mismatch between the expected and current state of a subsystem is detected.

It's input consists of the *SubsystemID*, the requested state of the subsystem, the *BoardConfiguration* structure and the *GlobalTelemetry*. While the algorithm runs on all boards, the following set of conditions ensures that only a single board activates a switch for the subsystem.

The board is selected if,

- it has a matching subsystem switch,
- it has no fault bit set for the board and switch state,
- and has the lowest *BoardID* among the boards that fulfill the above criteria.

These criteria ensure that all other boards terminate the execution of the algorithm before changing the state of their local switches.

The algorithm starts by iterating over all boards starting with *BoardID* zero. It checks to the *BoardConfiguration* if the board with the current *BoardID* provides a switch for the specified *SubsystemID* and increments the *BoardID* if this is not the case. In the next step the fault bits for the board and the relevant switch are checked and the *BoardID* is incremented if either bit is set. If both conditions are fulfilled the board compares the current *BoardID* with it's own *BoardID* and aborts the algorithm if they don't match.

The final remaining board then activates its switch for the subsystem, updates its own state accordingly and publishes its updated *BoardTelemetryShared*.

For disabling a subsystem the algorithm can be simplified and the steps above are not required. Every board with a switch associated to the subsystem can simply disable the switch independent of its previous state.

4.5 Error Case Analysis

To verify the reliable operation of the system described in this chapter a list of possible error cases along with their mitigation and outcome shall be checked in the following.

4.5.1 Sensor failure

Description Due to the number of discrete sensors on every board there exists a risk of failure of individual sensor during the expected mission duration.

Mitigation While the sensors provide additional safety, none of them are critical for the operation. To selectively disable individual sensors the *BoardConfiguration* contains a bitmask where the individual bits map to sensors. Once a bit is set the related sensors is marked as untrusted and it's sensor values are no longer used.

Outcome With the failed sensor disabled the system can no longer fulfill R.9 *Power flow telemetry* for the channel associated with the sensor. Nevertheless all other sensors still provide telemetry and the overall system operation and safety is unaffected as R.6 *Overcurrent Detection* and R.7 *Overcurrent Shutdown* can still be handled by the switch itself.

4.5.2 Subsystem overcurrent

Description Due to transient or persistent electrical issues in a subsystem or the EMPS board an overcurrent condition might occur where the subsystem consumes more than its allocated power. This could impact the overall system performance or in the case of a short circuit potentially damage the subsystem or the EMPS board.

Mitigation On the EMPS board the switch associated with the affected subsystem provides multiple ways to indicate faults. On one hand sensors are used to monitor the state of switches, subsystems and the battery. In case a current measurement exceeds the configured maximum current for this subsystem the switch will be disabled by the microcontroller.

Additionally the switch itself provides a hardware overcurrent shutdown that disables the switch above a maximum current that the EMPS is capable to provide. This hardware shutdown acts as an additional safety with a current limit above the configurable software limit.

Outcome Once a overcurrent condition occurs the switch is disabled and the switch state indicates a fault.

4.5.3 Switch for a subsystem indicates a fault

Description Once a fault on a switch occurs the switch is disabled and the switch state indicates a fault. One example for this is a subsystem overcurrent condition as described in Section 4.5.2. The information about this switch fault is then propagated to the other subsystems as part of the shared state.

Mitigation The resulting mismatch in expected subsystem state and actual subsystem state causes the switch selection algorithm to run as described in Section 4.4.

Outcome If the subsystem is connected to multiple switches on different EMPS boards another board will enable the switch corresponding to the subsystem. If the subsystem is only connected to a single EMPS board the subsystem will stay disabled.

4.5.4 All Switches for a subsystem indicate a fault

Description When a switch for a subsystem indicates a fault the switch is disabled on the EMPS board and the switch selection is run as described in Section 4.5.2. When the corresponding switch on the next board also indicates a fault the process is repeated until all switches for that subsystem indicate a fault and the subsystem stays disabled.

Mitigation The subsystem can be enabled again by the *SwitchSubsystem* command from the OBDH which in turn clears the fault state of all switches of that subsystem. To prevent critical systems from being permanently disabled, subsystems can have their *initial state* set to enabled as described in Section 4.2.1. This causes the EMPS boards to automatically clear the switch fault once all boards indicate an error and restart the whole switch selection process from the beginning.

Outcome Repeated attempts to enable the subsystem ensure the system can recover from transient electrical issues.

If the electrical issue on the subsystem is persistent the subsystem will repeatedly enter the fault state. While this outcome is not optimal, it is preferred to stopping all attempts to power a subsystem critical for operation of the satellite.

4.5.5 Board failure

Description In case of a complete board failure the board will no longer provide state updates to other boards.

Mitigation An internal timeout on the EMPS board keeps track of the time of the last state update reception from every other EMPS board. If a board fails to provide periodic state updates and the time since the last update exceeds a threshold its board state is ignored by all other boards as described in Section 4.4.

This results in a mismatch of the state of subsystems that were previously powered by the failed board. Consequently the switch selection algorithm from Section 4.4.2 runs.

Outcome All subsystems that are connected to multiple boards and were previously powered by the failed board will now be powered by another board. Subsystems that are only connected to the failed board can no longer be powered.

4.5.6 Transmission error in state update

Description The state update from another board is received with transmission errors.

Mitigation While individual messages on the CAN bus already contain a checksum ensuring the integrity of those messages, each CAN message is limited to 8 bytes of payload data. To ensure the integrity of the 40 bytes of shared state update an additional checksum over the complete state update (excluding the checksum) is calculated

before transmission and verified after reception. Packets with invalid checksum are discarded on reception and treated if no update was received.

Outcome Transmission errors are detected and all decision based on the shared state are performed with a sufficient delay that the next state updates are received in time. Thus occasional transmission errors in state updates have no negative effect on the system.

4.5.7 Split system

Description In a split system communication is no longer possible between a subsection of all nodes/ EMPS boards. An example for this would be a three board configuration where two boards are still able to communicate with each other and the OBDH while the third board is unable to receive/ send messages over the CAN bus.

Mitigation With the boards unable to share their state updates with each other a consistent shared state across boards is no longer possible.

Nevertheless no subsystems will be powered by multiple boards at the same time as the presence of voltage on a disabled switch is indicated in the switch state as described in Section 4.2.2.

Outcome Subsystems connected to switches of the boards separated from the OBDH can no longer be enabled or disabled. Subsystems with an enabled *initial state* will stay enabled and subsystems only connected to the boards with a disabled *initial state* can no longer be enabled.

Again while this is not a perfect outcome the overall system is not severely impacted as the enabled *initial state* is mainly intended for critical subsystems that should be permanently enabled and subsystems with a disabled *initial state* can still be controlled as long as they are connected to an additional EMPS board.

4.5.8 CAN bus failure

Description In case of a complete failure of the CAN bus no internal communication between the boards as well as external communication with the OBDH is possible.

Mitigation While no communication between the boards is possible the boards still act independently and enable all subsystems with an enabled *initial state*.

Outcome Without the CAN bus the OBDH can no longer control the state of the subsystems. A failure of the CAN bus results in the worst possible outcome and currently presents a single point of failure. Nevertheless critical subsystems with an enabled *initial state* will still be supplied with power by the EMPS boards and for these subsystems the EMPS essentially behaves like a power supply that can't be controlled.

5 Evaluation

The following chapter details the steps that were performed in order to evaluate the functionality and performance of the system described in the previous chapters. While the fulfillment of the basic EPS requirements Section 2.2.2 is verified the main focus lies on the distributed operation of the system and the associated requirements from Section 2.2.3.

5.1 Functional Testing

To validate the nominal operation of the system all requirements and error cases, a set of functional tests was performed. Each test case starts with a brief test description followed by the outcome of the test and a reference to the requirements or error cases this test case verifies. While some requirements were not specifically covered in this thesis they are essential parts of the EMPS software and hardware and test cases for these requirements are thus included.

5.1.1 Single Board Test Cases

T.1 Power input

Test description

To test if the system is capable of charging the batteries an external power source was connected to the BCR input pins. The battery voltage and input current were measured before and some time after connecting the external power source.

Outcome

The input current was measured at the expected level after connecting the external power source and the battery voltage increased between the measurements.

Reference

R.1 *Power input*

T.2 Power storage and supply

Test description

To test if the system is capable of storing and supplying a sufficient amount of electrical power the batteries were fully charged. Subsequently the battery was discharged over a load resistance connected to a regulated voltage switch until a minimum battery voltage level was reached.

Outcome

Integrating the measured constant discharge current over the duration of the discharge cycle and multiplying it with the voltage resulted in a sufficient battery capacity in line with the specified capacity of the battery when factoring in DCDC conversion inefficiencies.

Reference

R.2 *Power storage* and R.3 *Power supply*

T.3 Telemetry and control

Test description

To test if the communication interface an external OBDH was connected to the CAN bus. Afterwards the commands specified in Section 4.3 were sent from the OBDH over the CAN bus.

Outcome

Commands to receive telemetry resulted in the expected telemetry data being sent by the EMPS board over the CAN bus. Commands to control the state of a subsystem resulted in a change of the subsystems state as confirmed by a voltage measurement on the associated switch as well as a change in the subsystems telemetry.

Reference

R.8 *Telemetry and control*

T.4 Power channels

Test description

To test if the system is capable of providing power over all available switches a load resistance is connected to all switches sequentially. After enabling the subsystem associated with the connected switch the voltage and current for this switch was retrieved over the telemetry interface.

Outcome

The system was able to supply all switches with power and provided voltage and current telemetry for every switch.

Reference

R.4 *Power channels*, R.5 *Power monitoring* and R.9 *Power flow telemetry*

T.5 Overcurrent detection and shutdown

Test description

To test if the system is capable of detecting an overcurrent condition and subsequently disables the associated switch, a variable load resistance was connected to all switches sequentially. For every switch the value of the variable resistance was gradually reduced until the output current exceeded the maximum output current. This test was repeated twice, once with a maximum output current configured in software and a second time with no maximum output current configured in software to test the hardware current limit.

Outcome

For all performed tests the switch was disabled automatically once the maximum output current was exceeded. Additionally the subsystem telemetry associated with the switch indicated an overcurrent condition as expected.

Reference

R.6 *Overcurrent detection* and R.7 *Overcurrent shutdown*

T.6 Electrical inhibit

Test description

To test if the system can be completely disabled in spite of the built in batteries, the mechanical inhibit switches were activated.

Outcome

As soon as the mechanical inhibit switches were activated the voltage supplied to the microcontroller dropped and the microcontroller stopped running.

Reference

R.10 *Electrical inhibit*

5.2 Multi-Board Test Setup

The basic multi-board test setup consists of two EMPS boards and is visible in Figure 5.1. On every EMPS board two subsystems are connected to the output switches. *Subsystem 2* is connected to *Switch 2* of every board while *Subsystem 4* is connected to *Switch 4*. While the mapping between switch and subsystem can be configured in the *BoardConfig* as described in Section 4.2.1 the default configuration maps every switch to the subsystem with the same number and thus no configuration is required for this setup.

All EMPS boards are connected to the shared CAN bus along with a Raspberry Pi acting as OBDH to send commands over the external command interface.

For tests with more than two EMPS boards this test setup can easily be extended by connecting an additional board to the shared CAN bus while additionally connecting the two present subsystems with the corresponding switches of the new board.

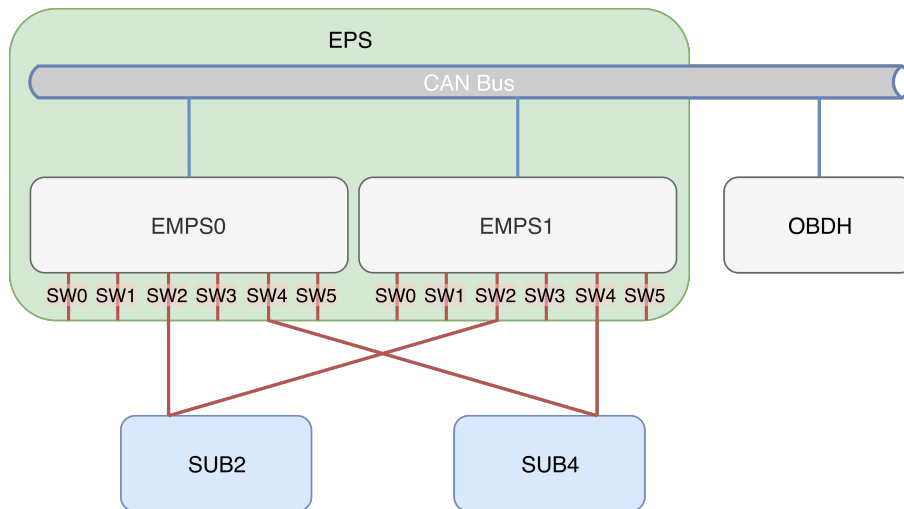


Figure 5.1: Test configuration with two EMPS boards

5.2.1 Multi-Board Test Cases

TD.1 Transparent external interface

Test description

To test the external interface in the presence of multiple boards, the system was configured as described in Section 5.2.

Afterwards the commands specified in 4.3 were sent from the OBDH over the CAN bus as previously tested for a single board in T.3.

Outcome

Commands that specifically addressed a single board by its *BoardID* resulted in a response from that board as described in Section 4.3. All other commands resulted in a response from the board with the lowest available *BoardID*. For the OBDH controlling the EMPS system with multiple boards the external interface behaves identical to a single board system.

Reference

RD.1 *Transparent external interface*

TD.2 Enable subsystem**Test description**

To test if subsystems can be enabled in a multi-board setup, the command to enable *Subsystem 2* was sent from the OBDH. Afterwards the telemetry of all boards was retrieved to ensure only a single switch for the subsystem was activated across all boards.

Outcome

As expected this resulted in *Switch 2* on the board with the lowest *BoardID* being enabled as indicated by the telemetry.

Reference

RD.2 *Single supply*

TD.3 Disable subsystem**Test description**

Following the activation of *Subsystem 2* in TD.2, the command to disable *Subsystem 2* was sent from the OBDH. Afterwards the telemetry of all boards was retrieved to ensure the subsystem was disabled across all boards.

Outcome

As expected *Subsystem 2* was no longer supplied by any board after being disabled and all associated switches were disabled as indicated by the telemetry.

Reference

RD.2 *Single supply*

TD.4 Failover**Test description**

This test checks if a subsystem continues to be supplied by the EMPS after the board originally supplying the subsystem fails. After enabling *Subsystem 2* as described in Item TD.2 the board with the lowest *BoardID* was disconnected from the CAN bus as well as from the subsystems.

Outcome

After disconnecting the board *Subsystem 2* was without power for a few seconds until *Switch 2* on the remaining board was activated.

Reference

RD.3 *Failover*

TD.5 Number of boards**Test description**

A test with the maximum number of theoretically supported boards was not possible due to limited amount of available EMPS boards. During testing up to six EMPS boards were connected to a shared CAN bus for extended periods of time as visible in Figure 5.2.

To test the behavior of the system with more than six boards additional six microcontrollers with CAN capability were connected. These microcontroller continuously updated and shared their own shared state but did not respond to commands over the external command interface.

Outcome

The system performed as expected in the configuration with six EMPS boards as well as in the configuration with six EMPS boards and six additional microcontrollers.

Reference

RD.4 *Number of boards*

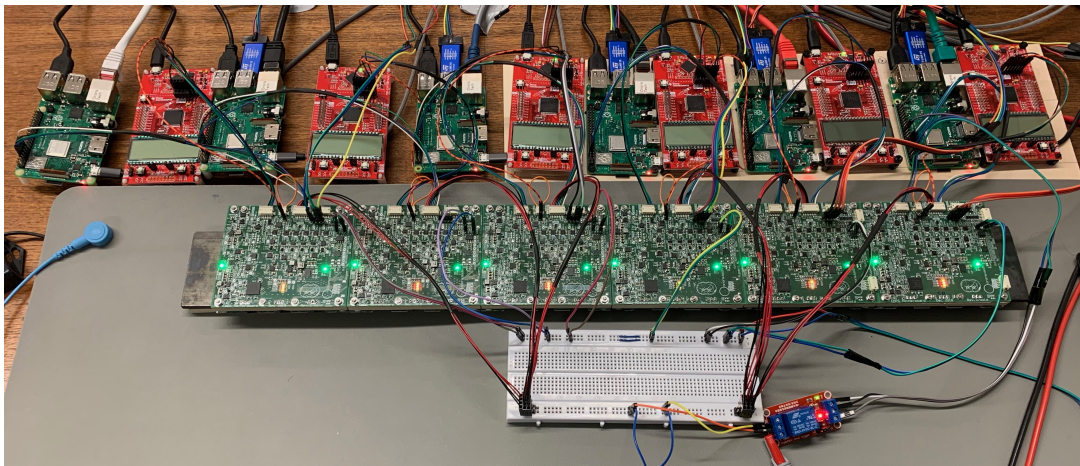


Figure 5.2: *Test setup with six EMPS boards connected to a shared CAN bus*

5.2.2 Error Cases

Test for the error cases described in Section 4.5 were performed according the error case description and the outcome matched the expected outcome.

5.3 Spacecraft Integration Testing

To facilitate the development and testing of the EMPS system an extensive support infrastructure was created during the project.

Every EMPS board is continuously connected to a dedicated Raspberry Pi and a MSP430 programmer as visible in Figure 5.3. This Raspberry Pi is connected to the external interface of the EMPS board and is thus able to act as OBDH sending commands to the EMPS

board and retrieve its telemetry. Additionally the Raspberry Pi can control the BCR input of the EMPS board to enable or disable charging. A MSP430 programmer attached to the Raspberry Pi allows flashing new firmware on the microcontroller. A connection to the system can be established remotely through the Ethernet connection of the Raspberry Pi.

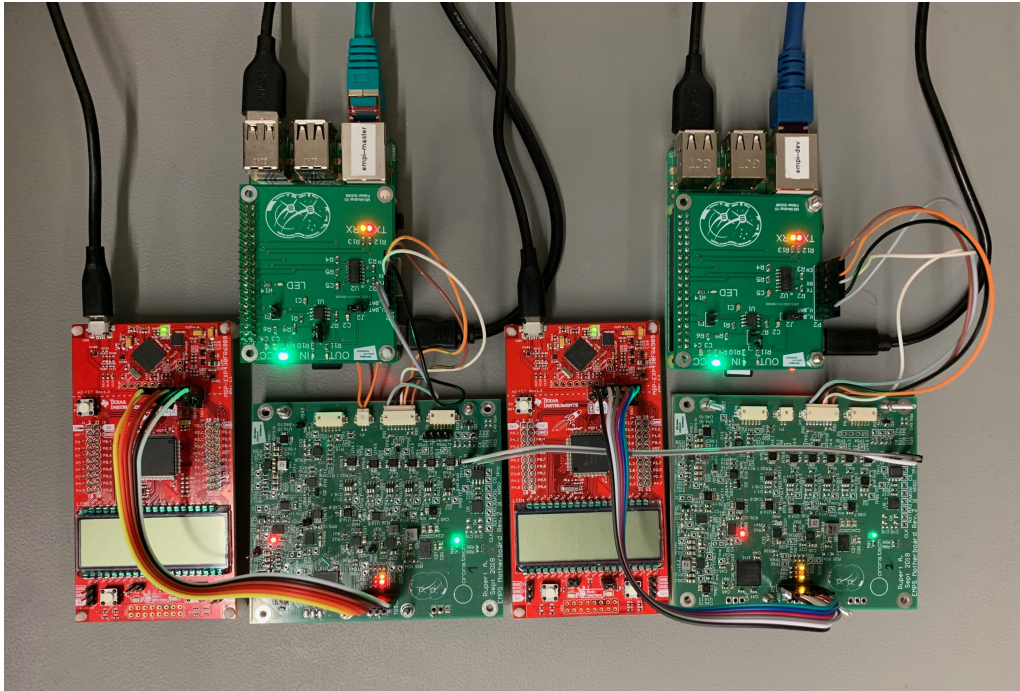


Figure 5.3: *Picture of the development environment with two EMPS boards*

A Continuous Integration and Continuous Delivery (CI/CD) pipeline on the Gitlab server hosting the source code repository is initiated every time a new commit is pushed and is visible in Figure 5.4.

In the first phase of the build process the firmware is compiled by a docker container running on a Gitlab-runner. The resulting firmware images are saved and archived as build artifacts on the Gitlab server. In the next phase of the CI/CD pipeline the Gitlab-runner connects to the Raspberry Pi in order to flash the new firmware on the EMPS board. Finally in the test phase of the CI/CD pipeline a test suite is executed on the Raspberry Pi. This test suite communicates with the EMPS board through the external command interface, temporarily enables subsystems and checks telemetry values to ensure the new firmware behaves as expected. A similar pipeline is used to run unit test on the microcontroller.

To keep track of the state of all boards a web based visualization based on Grafana provides access to the current and historic telemetry of all EMPS boards. In this setup a Python script running on the Raspberry Pi continuously retrieves the telemetry of a single EMPS board over the external command interface. The script then parses the data and publishes the parsed telemetry over MQTT. Telegraf subscribes to MQTT topic of the parsed telemetry and stores the data in the time series database InfluxDB. Finally the Grafana frontend is used to visualize the data stored in the database.

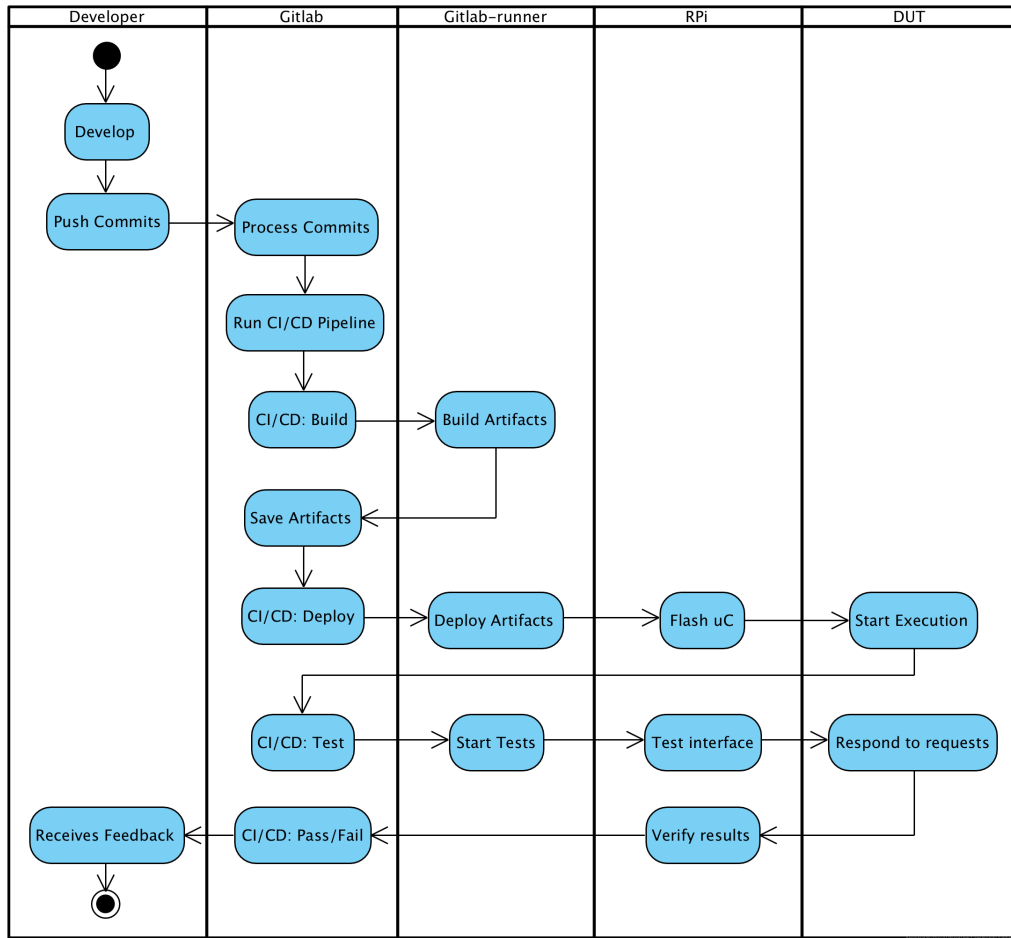


Figure 5.4: *CI/CD hardware integration*

The whole setup provides a convenient way to develop for an embedded target platform, as no physical hardware access is required to test new software revisions. Unit and Integration tests enable a stable and consistent system behavior across firmware revisions while the data visualization is essential for keeping track of the system performance over time.

5.4 Power Consumption

By measuring the power consumption of a single EMPS board with a variable number of other EMPS boards connected to the system, the impact of the number of boards on the overall power consumption can be estimated.

Figure 5.5 shows two graphs with the number of connected EMPS boards in the upper graph and the power consumption of a single board in the lower graph. No significant correlation between the number of connected EMPS boards and the power consumption is visible.

Sending messages over the CAN bus consumes energy from the board sending the message,

but due to the design of the coordination algorithm the number of state updates a single boards sends is unaffected by the number of boards. Although the board's microcontroller needs to process more status updates from other boards and thus spends less time in a low power sleep mode this additional processing proves to be insignificant compared to the overall power consumption of the system.

In conclusion the power consumption of the EMPS system scales linearly with the number of boards as a single board's current consumption remains constant independent of the number of connected boards.

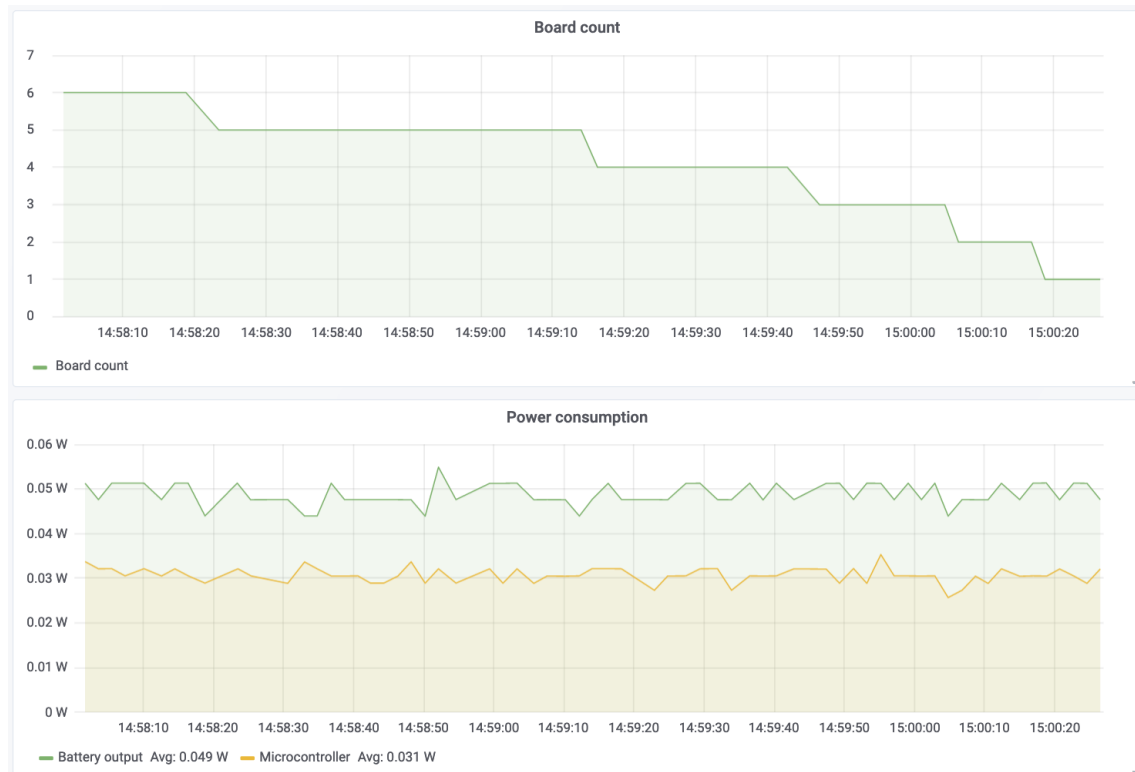


Figure 5.5: *Power consumption measurements*

5.5 Summary

Functional testing of the EMPS system shows that all requirements are fulfilled and the system behaves as expected in an extensive set of defined error cases.

During development a set of unit and integration tests as described in Section 5.3 ensure that every feature is continuously tested.

Finally a suite of environmental test were performed in the Thermal Vacuum Chamber (TVAC) of the LRT to test the system closer to its expected operation environment in the vacuum of space.

6 Conclusion and Outlook

At the time of writing this thesis the next and final hardware revision for the UNSW mission is currently in production. After assembly and testing this hardware will be shipped to Australia at the end of February. Finally the UNSW M2 mission is scheduled to launch in August 2019 and will verify the EMPS hardware as well as software.

The full multi-board version of the EMPS project is currently planned to be used in the EVE-1 mission, the first satellite developed by orora.tech. The project timeline schedules the completion of the first satellite at the end of 2019. The hardware design will be adjusted, integrating the solar panels in the smartpanel architecture and most UNSW mission specific test circuits will be removed.

The software and communication protocols developed in this thesis serve as a solid foundation for all future satellite missions and applications of the EMPS.

List of Figures

1.1	Assembled engineering model of the 1U MOVE-II CubeSat	1
1.2	Top view of the engineering model of the MOVE-II CubeSat	1
1.3	EPS basic interfaces	2
1.4	EPS internal components	3
1.5	UNSW M2 satellite with CubeSat dispenser[10]	4
1.6	UNSW M2 satellite in deployed state[10]	4
1.7	Mission patch of the EMPS project	4
2.1	EMPS multi-board setup with shared CAN bus	6
2.2	EMPS hardware architecture	7
3.1	Distributed shared memory	12
4.1	EMPS multi-board setup with connected subsystems	14
4.2	EMPS processing loop	16
5.1	Test configuration with two EMPS boards	23
5.2	Test setup with six EMPS boards connected to a shared CAN bus	25
5.3	Picture of the development environment with two EMPS boards	26
5.4	CI/CD hardware integration	27
5.5	Power consumption measurements	28

List of Tables

4.1	Mapping from Board and switch to subsystem	14
A.1	Shared state of a single board	35
A.2	Board configuration data structure for 3 boards	36
A.3	Subsystem configuration data structure for 3 boards	36

Acronyms

1U one Unit

6U six Unit

BCR Battery charge regulator

CAN Controller Area Network

CI/CD Continuous Integration and Continuous Delivery

DCDC DC-to-DC converter

DSM Distributed Shared Memory

EMPS Extendable modular power supply

EPS Electrical Power System

FRAM Ferroelectric RAM

LRT Chair of Astronautics

MOVE-II Munich Orbital Verification Experiment II

MPPT Maximum power point tracking

OBDH On-board Data Handling

orora.tech Orbital Oracle Technologies GmbH

RISC Reduced instruction set computer

SEU Single event upset

TI Texas Instruments

TUM Technical University of Munich

TVAC Thermal Vacuum Chamber

UART Universal Asynchronous Receiver/Transmitter

UNSW University of New South Wales

WARR Scientific Workgroup for Rocketry and Spaceflight

Bibliography

- [1] J. Straub, “CubeSats: A Low-Cost, Very High-Return Space Technology”, en, *Proceedings of the AIAA Reinventing Space Conference*, p. 6, 2012.
- [2] M. Langer, F. Schummer, N. Appel, T. Gruebler, K. Janzer, J. Kiesbye, L. Krempel, A. Lill, S. Rueckerl, and M. Weisgerber, “MOVE-II THE MUNICH ORBITAL VERIFICATION EXPERIMENT II”, en, p. 19, 2017.
- [3] M. Arash, “CubeSat Design Specification”, en, *CalPoly SLO*, p. 42, 2014.
- [4] R. Shimmin, “Small Spacecraft Technology State of the Art”, en, *NASA Ames Research Center, Mission Design Division*, p. 173, 2015.
- [5] M. Langer and J. Bouwmeester, “Reliability of CubeSats - Statistical Data, Developers’ Beliefs and the Way Forward”, en, p. 12, 2016.
- [6] M. Swartwout, “The First One Hundred CubeSats: A Statistical Look”, en, p. 21, 2013.
- [7] C. Clark, “Huge Power Demand...Itsy-Bitsy Satellite: Solving the CubeSat Power Paradox”, en, p. 8, 2010.
- [8] M. Pajusalu, E. Ilbis, T. Ilves, M. Veske, J. Kalde, H. Lillmaa, R. Rantsus, M. Pelakauskas, A. Leitu, K. Voormansik, V. Allik, S. Lätt, J. Envall, and M. Noorma, “Design and pre-flight testing of the electrical power system for the ESTCube-1 nanosatellite”, en, *Proceedings of the Estonian Academy of Sciences*, vol. 63, no. 2S, p. 232, 2014, ISSN: 1736-6046. DOI: 10.3176/proc.2014.2S.04.
- [9] T. Aburouk, S. Kim, H. Masui, and M. Cho, “Design, Fabrication, and Testing of an Electrical Double-Layer Capacitor-Based 1U CubeSat Electrical Power System”, en, p. 17, 2018.
- [10] Cooperation agreement between UNSW and TUM, 2018.
- [11] R. Amann, “Development and Test of a Controllable Battery Charge Regulator for CubeSat Applications”, unpublished, Master’s thesis, TUM.
- [12] F. Schöttl, “Software development for a modular power supply for cubesats”, unpublished, IDP Report, TUM.
- [13] T. Grüber, “Highly Integrated Smart Satellite Panels for Commercial Space Applications”, en, p. 63, 2017.
- [14] T. Instruments, “Msp430fr5989-ep mixed-signal microcontroller datasheet”, 2017.
- [15] R. Netzer, K. Avery, W. Kemp, A. Vera, B. Zufelt, and D. Alexander, “Total Ionizing Dose Effects on Commercial Electronics for Cube Sats in Low Earth Orbits”, en, in *2014 IEEE Radiation Effects Data Workshop (REDW)*, Paris, France: IEEE, Jul. 2014, pp. 1–7. DOI: 10.1109/REDW.2014.7004607.

- [16] S. M. Guertin, M. Amrbar, and S. Vartanian, “Radiation Test Results for Common CubeSat Microcontrollers and Microprocessors”, en, in *2015 IEEE Radiation Effects Data Workshop (REDW)*, Boston, MA, USA: IEEE, Jul. 2015, pp. 1–9. DOI: 10.1109/REDW.2015.7336730.
- [17] I. Kronhaus, “Design of the UWE-4 Picosatellite Orbit Control System using Vacuum-Arc-Thrusters”, in *International Electric Propulsion Conference*, 2013.
- [18] L. J. Paxton, ““Faster, better, and cheaper” at NASA: Lessons learned in managing and accepting risk”, en, *Acta Astronautica*, vol. 61, no. 10, pp. 954–963, Nov. 2007. DOI: 10.1016/j.actaastro.2006.10.014.
- [19] S. O. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, RFC 2119, 1997. DOI: 10.17487/RFC2119.
- [20] M. van Steen and A. S. Tanenbaum, *Distributed systems*, en, Third edition (Version 3.01 (2017)). The Netherlands: Published by Maarten van Steen, 2017, OCLC: 1006750554, ISBN: 978-1-5430-5738-6.
- [21] M. Raynal, “A Look at Basics of Distributed Computing”, en, in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Nara, Japan: IEEE, Jun. 2016, pp. 1–11, ISBN: 978-1-5090-1483-5. DOI: 10.1109/ICDCS.2016.109.
- [22] R. Singh, J. Malviya, and K. Jha, “A Survey of Mutual Exclusion Algorithms in Distributed Computing”, en, p. 4, 2013.
- [23] M. EffatParvar, N. Yazdani, M. EffatParvar, A. Dadlani, and A. Khonsari, “Improved algorithms for leader election in distributed systems”, en, in *2010 2nd International Conference on Computer Engineering and Technology*, Chengdu, China: IEEE, 2010, ISBN: 978-1-4244-6347-3. DOI: 10.1109/ICCET.2010.5485357.
- [24] S. Balhara and K. Khanna, “Leader Election Algorithms in Distributed Systems”, en, *International Journal of Computer Science and Mobile Computing*, p. 6, 2014.
- [25] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, Honghui Lu, R. Rajamony, Weimin Yu, and W. Zwaenepoel, “TreadMarks: Shared memory computing on networks of workstations”, en, *Computer*, vol. 29, no. 2, pp. 18–28, 1996, ISSN: 00189162. DOI: 10.1109/2.485843.
- [26] B. Nitzberg and V. Lo, “Distributed shared memory: A survey of issues and algorithms”, en, *Computer*, vol. 24, no. 8, pp. 52–60, Aug. 1991, ISSN: 0018-9162. DOI: 10.1109/2.84877.

A Appendix

A.1 Data Structures

Addr	Addr + 0	Addr + 1	Addr + 2	Addr + 3
0x00	protocol_major	protocol_minor	firmware_major	firmware_minor
0x04	board_id	board_count	revision_counter[0]	revision_counter[1]
0x08	uptime[0]	uptime[1]	uptime[2]	uptime[3]
0x0C	state.mode	state.error	state.flags	state.flags
0x10	switches_0.subsystem_id	switches_0.state	switches_1.subsystem_id	switches_1.state
0x14	switches_2.subsystem_id	switches_2.state	switches_3.subsystem_id	switches_3.state
0x18	switches_4.subsystem_id	switches_4.state	switches_5.subsystem_id	switches_5.state
0x1C	switches_6.subsystem_id	switches_6.state	switches_7.subsystem_id	switches_7.state
0x20	temperature[0]	temperature[1]	-	-
0x24	config_checksum[0]	config_checksum[1]	telemetry_checksum[0]	telemetry_checksum[1]

Table A.1: Shared state of a single board

Addr	Addr + 0	Addr + 1	Addr + 2	Addr + 3
0x00	boards_0.switches_0.subsystem_id	boards_0.switches_1.subsystem_id	boards_0.switches_2.subsystem_id	boards_0.switches_3.subsystem_id
0x04	boards_0.switches_4.subsystem_id	boards_0.switches_5.subsystem_id	boards_0.switches_6.subsystem_id	boards_0.switches_7.subsystem_id
0x08	boards_0.disabled_sensors[0]	boards_0.disabled_sensors[1]	-	-
0x0C	boards_1.switches_0.subsystem_id	boards_1.switches_1.subsystem_id	boards_1.switches_2.subsystem_id	boards_1.switches_3.subsystem_id
0x10	boards_1.switches_4.subsystem_id	boards_1.switches_5.subsystem_id	boards_1.switches_6.subsystem_id	boards_1.switches_7.subsystem_id
0x14	boards_1.disabled_sensors[0]	boards_1.disabled_sensors[1]	-	-
0x18	boards_2.switches_0.subsystem_id	boards_2.switches_1.subsystem_id	boards_2.switches_2.subsystem_id	boards_2.switches_3.subsystem_id
0x1C	boards_2.switches_4.subsystem_id	boards_2.switches_5.subsystem_id	boards_2.switches_6.subsystem_id	boards_2.switches_7.subsystem_id
0x20	boards_2.disabled_sensors[0]	boards_2.disabled_sensors[1]	-	-

Table A.2: Board configuration data structure for 3 boards

Addr	Addr + 0	Addr + 1	Addr + 2	Addr + 3
0x00	subsystems_0.initial_state	-	-	-
0x04	subsystems_0.timeout[0]	subsystems_0.timeout[1]	subsystems_0.maximum_current[0]	subsystems_0.maximum_current[1]
0x08	subsystems_1.initial_state	-	-	-
0x0C	subsystems_1.timeout[0]	subsystems_1.timeout[1]	subsystems_1.maximum_current[0]	subsystems_1.maximum_current[1]
0x10	subsystems_2.initial_state	-	-	-

Table A.3: Subsystem configuration data structure for 3 boards