Hardware and Software Framework for an Open Battery Management System in Safety-Critical Applications

M. Akdere, M. Giegerich, M. Wenger, R. Schwarz, S. Koffel, T. Fühner, S. Waldhör, J. Wachtler
V.R.H. Lorentz, M. März
Fraunhofer IISB, Erlangen, Germany
Vincent.Lorentz@iisb.fraunhofer.de

Abstract – **Lithium ion batteries are a common choice for many use cases, ranging from medical devices to automotive and airborne applications. Despite their widespread application, lithium ion batteries still remain an expensive, yet sensitive component within these systems. In order to maintain the operability of the battery system over its designated service life an appropriate battery management system (BMS) is required. The development of such a BMS is a challenging task, as various technological, environmental and application-specific aspects have to be considered. Especially safe and reliable operation of the battery system is an important and critical issue in this context. Besides these safety critical aspects, the BMS also includes extensive non safety related components and functions. Therefore, in order to fulfill safety-critical requirements, it is mandatory to keep the respective hardware and software components isolated. Redundancy, partitioning and the implementation of diagnostic functions at several software layers and different hardware partitions are the mechanisms for ensuring the integrity of the system. For performance and economical reasons, these techniques have to be tailored to the application. Based on a real-time operation system, a flexible and extensible strategy for a software framework with minimal code size, lean interfaces and few dependencies is introduced. The use of a dedicated *BMS-Engine* with a partitioned database enables the implementation of a stringent safety concept, which is discussed and demonstrated to be feasible.**

I. INTRODUCTION

A battery management system (BMS) serves a multitude of purposes for which it is comprised of various functions and sub-modules. While some of these objectives such as extending the lifetime of the battery pack or the improvement of the overall system performance are not related to safety [1], many others are directly targeting the safe operation of battery cells and must be classified as safety-relevant. Moreover, the determination of cell parameters, the prediction of the battery behavior as well as the implementation of the according algorithms on an embedded system still pose quite a challenge. However, the required accuracy, rate and reliability of measured and estimated signals have to comply with the safety goals. Furthermore, the development of battery chemistries is rapidly proceeding. Consequently, not only the cell characterization and parameter identification are still in the focus of research, also their evaluation methods for the battery state are subject to constant change. On the hardware side, the available integrated electric circuits for cell monitoring and balancing are also proceeding and especially advances in the data communication interface require adaptions on BMS hardware and software. Consequently, the development of battery management systems goes with a continuous change of requirements or, at least, reappraisal of safety-relevant aspects.

All of these issues hinder the predetermination of generic battery management system architectures and currently a change in the application requirements often causes considerable effort for system adaption or even redesign. Proprietary commercial-of-the-shelf BMS hardware and software sytems are mostly complex and large systems that often can not be adapted to meet the individual requirements. For developing, testing, evaluation and benchmarking of battery system components, an easy-to-use, adaptable and extensible platform facilitates the revelation of critical issues in a timely manner and so helps to reduce costs.

The Fraunhofer IISB has developed a free, open and flexible research and development platform, foxBMS, for battery management systems. The general features of foxBMS are introduced in [2]. It includes all hardware and software components that are required for mobile and stationary applications using modern rechargeable electrochemical energy storage systems (e.g., lithium-ion batteries, redox-flow batteries, super capacitors). The foxBMS comprises two microcontrollers and duplicated peripheral hardware for the adaption of redundant components. The hardware and software architecture simplifies the development of BMS components in a live environment. This paper focuses on the safety aspects and features provided by foxBMS. It demonstrates the flexibility and extensibility of the platform in adapting error handling and safety-critical components to different hardware and software concepts.

II. SAFETY-RELATED SYSTEM DESIGN

A. Safety Features and Goals of the BMS

Keeping the battery cells in the safe operating area (SOA) is a safety goal of the BMS. Absolute maximum limits for the specific battery chemistry are usually stated in the datasheet by the battery manufacturer. To prevent the use of the battery

beyond these limits, the BMS will rigorously open the contactors in the case of harsh limit violations. This measure has to be regarded as the last line of defense to enforce the SOA. In normal operation an unexpected load drop to the attached power electronics is avoided by an additional software functionality called “state of function” (SOF), which provides a metric for the current capability of the battery pack. For example, to prevent an accelerated degeneration of the battery materials (aging) through operation near the limits given by their electrochemical properties the SOF is reduced. Especially fast charging at low temperatures leads to accelerated aging [3]. To attenuate these conditions, the lowered current capability, i.e. the state of function (SOF), is calculated and communicated to other control units. The operator or the supervisory control unit is encouraged to respect these suggested limits, but violation of the current ranges is not immediately resulting in an opening of the contactors, unless it is configured to do so. The measurement of cell voltages, cell temperatures and the battery current, as well as the detection and handling of SOA-violations within a given time are safety-critical tasks. In an airborne application, where in extreme cases the full availability of the system must be ensured, the operating area has to be widened at the expense of batteries health state. A typical automotive application would rather prefer to enter a limp-home mode with reduced system performance. Instead, a stationary application would use more sensitive settings for the safe operating area, as the long-term availability of the battery packs is of major importance. While the SOA definitely is a safety goal of the battery system, derating functions (SOF) have to be evaluated for possible relevance to safety of the overall system. Safety measures for failure cases have to be implemented in all related tasks (i.e. measurement tasks, estimation algorithms and regulation tasks). However, system dynamics, noisy measurements and presence of uncertainties in the model-based algorithms prevent a reliable diagnosis. Thus, additional measures for cross checking and verification are implemented to increase the reliability and robustness of the battery state monitoring. For example, an imprecise cell voltage measurement can be supplemented by a coulomb counter to estimate the SOC. Another example is the use of software diversity based on different kinds of cell models for state estimations, like electrical-circuit models, physics-based statespace models or electrochemical models as described in [1,4].

B. General Safety Methods

Like in any safety-critical embedded system, the monitoring of critical signals is the key issue to complying with system requirements and to prevent potential hazards. All involved components for data acquisition, data processing and system control have to be designed accordingly. Redundancy, partitioning and failure detection are among the main techniques to improve availability, robustness and reliability of the system. However, the development and certification of safety-critical applications is very expensive. To reduce the costs, safety-critical components are encapsulated, and safety measures can hence be restricted to the respective parts. Nevertheless, freedom from interference have to be ensured between the replicated or partitioned components. An example is the use of multi-core controllers. Safety-critical software is replicated and partitioned on multiple cores. This concept improves the system integrity, but still cannot provide the required isolation at the point of shared resources in the microcontroller (e.g., memory access, clock system and power supply). When the safety-related system requirements impose a design with strict separation of the computational system, downsizing the main controller to a single core and outsourcing of integrated features pose a possible solution. However, this measure is also directly related to costs and system complexity. Moreover, safety-critical functions and diagnostics have to be implemented in different abstraction layers and hardware partitions.

C. Safety-Related Methods of foxBMS

While the open system architecture of foxBMS is readily applicable to a wide range of BMS applications, its flexibility and extensibility allow for a straightforward integration of different safety strategies, for example, to investigate their behavior or impact on the system performance. This is supported by inter-controller communication and synchronization functions. Furthermore, the framework supports the integration of newly developed, potentially immature software modules and algorithms. As part of this framework, safety-critical functions and diagnostics are implemented at different layers and hardware partitions (Fig. 1). In addition, defined interfaces allow easy integration of application-specific extensions and relocation of functions to other partitions. This allows the integration of diagnostic functions and the described methods of redundancy and partitioning of hardware and software components in different ways.

III. HARDWARE ARCHITECTURE

The hardware topology of a BMS for lithium-ion batteries comprises the following components [5]:

- cell monitoring (e.g., temperature, voltage monitoring)
- passive or active cell balancing
- current measurement
- contactor and interlock control and monitoring
- isolation monitoring
- communication interfaces to peripherals and the environment, e.g. user or a superior control unit

A. Peripheral Hardware Components

The hardware of foxBMS consists of the BMS-Master, which is the main control and computation unit, and the

BMS-Slaves mounted on each battery module to monitor the battery cell voltages and temperatures.

As shown in Fig. 1 and Fig. 2, the BMS-Master follows a redundant topology. It uses two independent microcontroller units (MCU) based on an ARM Cortex-M4F core. Both MCUs are supplied independently and have their own physical interface to the battery monitoring. The BMS-Slaves, each of which is based on two state-of-the-art LTC6804-1 monitoring ICs, read each battery cell redundantly. This monitoring topology allows the

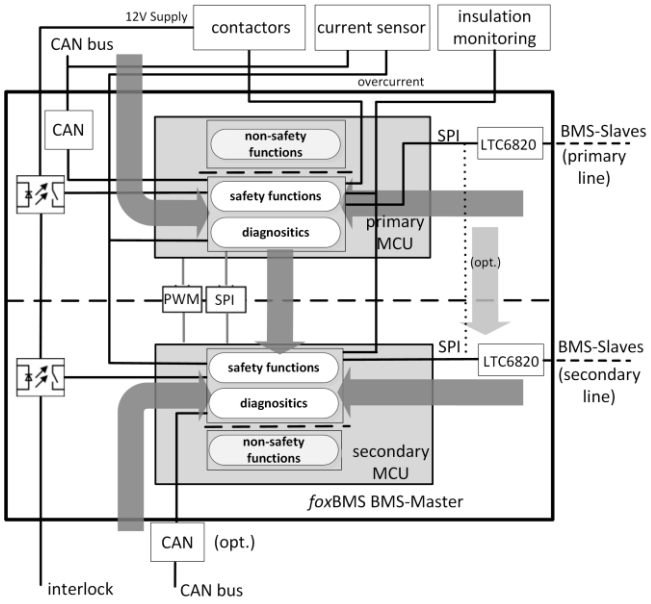

Fig. 1: foxBMS hardware topology and software partitions

primary MCU to perform data acquisition and further computations, whereas the secondary MCU is able to check for the safe operation area of the battery cells independently.

In the case of any violation of the safe operating area, each MCU is able to control and read the feedback of the common interlock line. Opening the interlock line is designed to immediately open the main contactors of the battery system to prevent the battery from any damage. In addition, an isolated interface between the two MCU exists for diagnostics and data distribution. The acquired data of primary and secondary cell monitoring lines from the BMS-Slaves are provided to both the primary and secondary MCU respectively. In case of a device failure in one of the monitoring lines, both MCUs are able to process the acquired data of one monitoring line. Alternatively or additionally, the second MCU can operate the associated monitoring ICs in a different operation mode. While the digital filters of the primary monitoring ICs are configured for higher noise reduction and lower data acquisition rates of all cell voltages, e.g. 50 Samples/s, the secondary MCU can measure the voltage of a specific cell in fast mode up to a rate of 2 kSample/s. This allows the verification and online tracking of cell parameters. For example, if the frequency spectrum of the current contains adequate components, the drift of the internal cell resistance can be detected to estimate the state of health (SOH).

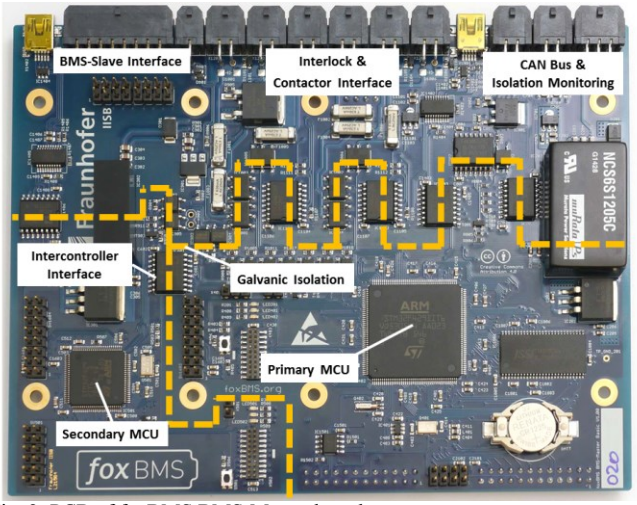

Fig. 2: PCB of foxBMS BMS-Master board

B. *Cell Monitoring*

Every battery cell is monitored by a BMS-Slave unit on the battery module level. The circuit of the BMS-Slaves is designed for battery modules consisting of 12 cells. The BMS-Slaves use a dual monitoring IC concept, where the voltage of every cell is read by two ICs redundantly. In addition, temperature sensors, installed inside the battery module, are read by these monitoring ICs. To increase safety, each IC is connected to its own temperature sensors. The proper operation of the monitoring ICs and the communication interfaces are verified by built-in diagnostics and cyclic redundancy checks (CRC). Optionally a single monitoring concept can be applied to both MCUs by using separate SPI buses (Fig. 1).

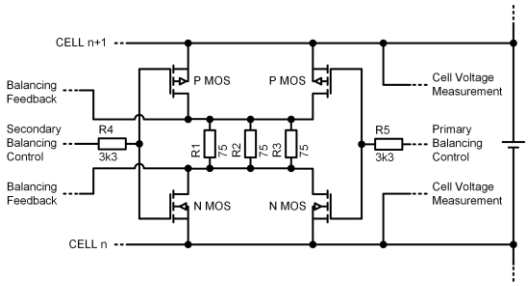

Fig. 3: Dual MCU controlled balancing circuit in the BMS-Slaves

C. *Balancing Control*

To prevent the balancing circuit from deep discharging any battery cell in the case of a single failure of a balancing MOSFET, an XOR control topology is implemented (Fig. 3). To enable cell balancing of a specific battery cell, the control pins of the balancing circuit must not be equal. This enables any of the BMS-Master MCU to stop the balancing process in

the event of a device failure (e.g., if the balancing MOSFET is shorted) or a dropout of the primary MCU. Therefore, this control technique is also suitable for fail operational systems.

D. *Dual Microcontroller System*

The STM32F429 microcontroller supports several advantageous features for a safety-related system design [6]. Listing the important ones:

- one window watchdog and one independently clocked watchdog
- memory protection unit (MPU)
- hardfault exception (detection of systematic software faults, random and transient hardware faults)
- port locking mechanism
- hardware based CRC unit
- 2 priority-based DMA controller
- DMA-based SPI communication with automatic CRC error checking

E. *Watchdog Concept and Synchronization*

Each MCU integrates a window watchdog and an independent watchdog. The window watchdog is clocked by the main clock of the MCU and can be adjusted with high time resolution. It primarily addresses the detection of software runtime faults. Failure modes of the clock system are detected by the independent watchdog, which is clocked by a dedicated clock with less accuracy. However, an external watchdog is mandatory for further reducing common failure modes (e.g., common power supply). For this purpose, both MCUs are connected by an isolated SPI-bus for data exchange and an isolated pulse-width modulation (PWM) signal for system synchronization.

IV. SOFTWARE ARCHITECTURE

A. *foxBMS Software*

The layered model and the core components of the foxBMS software are described in detail in [2]. In this paper a short summary is given.

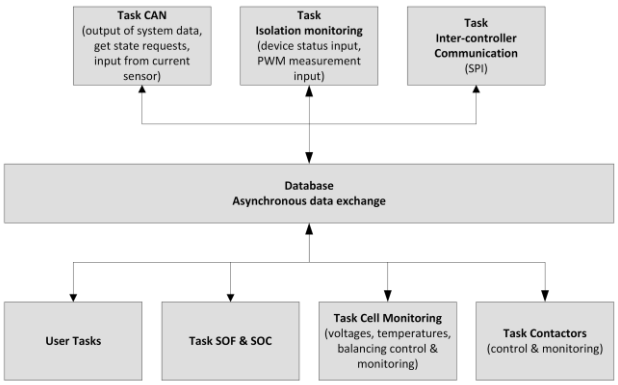

Fig. 4: data-provider and data consumer tasks and database

The real-time operating system is one of the core software components. In foxBMS, FreeRTOS is used, which offers scheduling with tasks (preemptive multitasking), queues and synchronization techniques like mutual exclusions (mutexes) and semaphores. The hardware abstraction layer on the lowest level implements interfaces to hardware peripherals, e.g., CAN bus, SPI bus and digital inputs and outputs. A second layer of abstraction relates to external circuitry like integrated circuits for battery monitoring and isolation monitoring devices. The respective sensor data is stored in a database that handles access to these data with queueing and locking mechanisms. The application-specific functionalities, like SOC and balancing algorithms, battery lifetime predictions and similar routines are located in a third top-level layer. This layer is subdivided in the *BMS-Engine*, i.e. the safety-relevant components, and user-specific components. The *BMS-Engine* includes all safety-relevant modules, a centralized database and those components that are to be invariant and unaffected by application-specific implementations. The benefit of a centralized database is the reduced complexity of the interfaces, since all components store and load their relevant data in this component (Fig. 4). Furthermore, the temporal interference between data producers and data consumers is minimized with the help of the queuing and locking mechanisms accompanied with the double buffer storage. This also facilitates an independent diagnostic functionality described later in this paper.

B. *Diagnostics and Safety-Related Software Strategy*

Like for the hardware, redundancy and partitioning are best practice for safety-critical software. There, the goal is to increase system availability, reliability and robustness. Software modules can also be replicated (diversity or homogeneous redundancy) and partitioned on both MCUs when input and output signals are interfaced (Fig. 6). For instance, replicas of battery state algorithms can be computed on both MCUs and compared at the same rate as both are equally performant. Due to replication of peripheral hardware and microcontrollers, the verification of software drivers is provided for the cell monitoring, current sensor and the interlock monitoring. The overall system consistency and timing constraints are verified by both controllers. The primary MCU serves as a test platform and is meant to operate as the actual battery management with diagnostics. The secondary MCU cross-checks the limits of the safe operating area of the battery cells. It also executes diagnostic functions that require the highest grade of isolation and integrates redundant software modules (cell monitoring, balancing control and interlock states). The centralized database in the *BMS-Engine* provides an asynchronous interface for data exchange. It supports three different levels of safety-related software modules and diagnostic services with different isolation characteristics.

- ***BMS Diagnostics:***
  High-priority task in the context of operating system

on the primary MCU, e.g. SOA diagnostics (optional implementation on the second MCU for redundancy purposes)

- ***System Diagnostics:***
  System monitoring and diagnostic functions integrated in the primary MCU with a higher grade of isolation to the context of the operating system (optional implementation on the second MCU for redundancy purposes)
- ***System Consistency Check:***
  Overall system monitoring and diagnostic functions integrated in both MCUs

C. *Diagnostics and Safety-Related Software Architecture*

The foxBMS software architecture adopts the described safety mechanisms in combination with the specific features of the real-time operating system FreeRTOS and the architecture of the ARM Cortex M4 controller to establish a flexible and extensible safe system with different topologies. The key aspects are:

- Temporal partitioning with real-time interrupts (RT-interrupts)
- Spatial partitioning of tasks
- Memory and peripheral access restrictions of FreeRTOS and tasks [7,8]

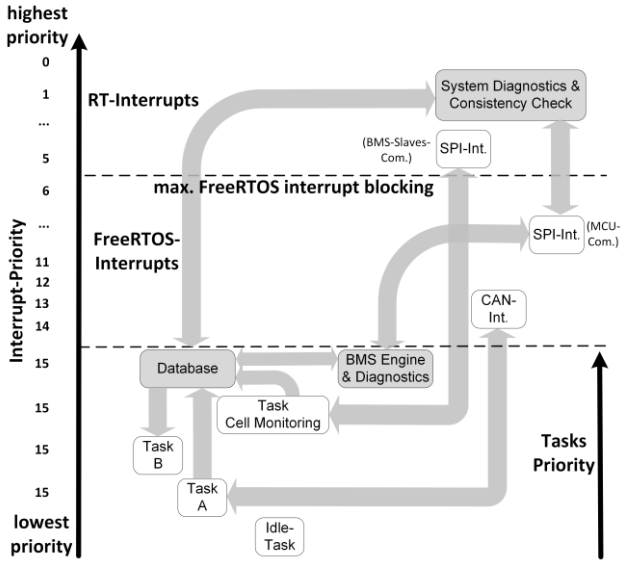

Fig. 5: Dataflow and priority assignment of tasks, BMS Engine with database and RT-Interrupts

One property of FreeRTOS is that global interrupts are never disabled. Synchronization methods increase the interrupt threshold to a certain adjustable level. Interrupts above this level (real-time interrupts) can bypass FreeRTOS which reduces timing dependencies. This allows for the integration of software modules with temporal isolation.

Another responsive feature of FreeRTOS with enabled memory protection unit (MPU) mode is that it provides spatial partitioning of the kernel and tasks. STM32F429 supports two privilege levels (privileged and non-privileged mode) and eight MPU regions with eight sub-regions each. The FreeRTOS kernel permanently occupies the first four region registers and the remaining four registers are dedicated to the task to set up their own stack usage, access modes and restrictions to other memory regions and peripherals [9]. When different regions in the MPU configuration overlap, a higher region register overwrites the access rights of a lower one. This means, that by configuration the access rights of tasks can be restricted or extended above the barriers of FreeRTOS kernel. Additionally, in the context of a RT-interrupts, software modules can be implemented which are isolated from the FreeRTOS kernel.

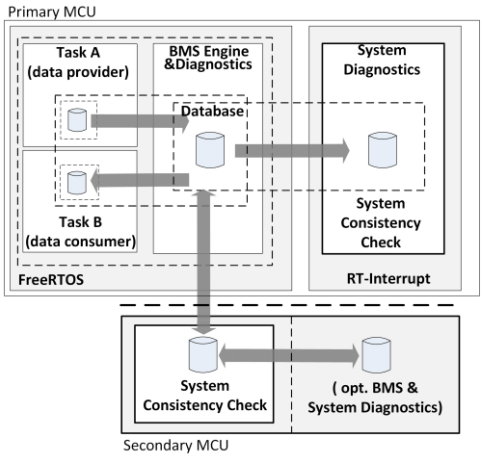

Fig. 6: Data isolation of tasks, database and real-time interrupts

D. *Implementation Details of BMS-Engine and Database*

As shown in Fig. 5, the database task in the *BMS-Engine* is assigned to a priority greater than the highest application task priority and is periodically called at the highest frequency (e.g., 1 ms). Its main purpose is to wait for a transfer request (message queue) from a task and stay in blocked state in the meantime. Tasks are representing data consumers and data providers with appropriate access to lower level software drivers. After receiving a request, the database enters the running state, verifies the requesting task, checks the data consistency and transfers the data. Tasks accessing the database are instantly blocked after filling the message queue with the required data (task-ID and signature, pointer to data-buffer) and continue after transfer of data is completed. This blocking procedure is the same for data providers and consumers. To assign the priority of the database task above all consumer and provider tasks has the advantage that the database task is never suspended by them. The transfer of data is very efficient as there are no timing delays, which would be the case if a task were to send and wait for a reply [7]. In addition, the data is copied directly (pass by reference) from the provider task to the database task or from the

database task to the consumer task without a buffering in the message queue as the tasks accessing the database are blocked and unable to corrupt the data in the meantime. Furthermore, the number of system calls and the required context switching times are reduced. As an example, acquisition of all cell voltages of 14 serially connected battery modules with 12 cells per module at a rate of 50 Samples/s, results in a transmission rate of 17.5 kByte/s (350 Bytes per transfer). To store the data in the database, 50 data transfers and 100 task switches per second between provider and database task are required. The same applies to all further read accesses (e.g., from communication or diagnostic tasks). Every data transfer takes about 60 µs.

The database task is prepared for interfacing custom functions in a separate partition (Fig. 6). This facilitates the implementation of additional diagnostic and monitoring features above all application task priorities within the context of FreeRTOS. At this point, the first level of diagnostics (*BMS-Diagnostics*) is implemented. The MPU configuration of application tasks prohibits accessing the memory area of the database. In contrast, the MPU of the database task is configured with read and write access rights to private data of application tasks. Furthermore, database and FreeRTOS are restricted to access the data region of a separate software module which is triggered by a real-time interrupt. In this context the second level of diagnostics (*System Diagnostics*) is integrated with additional access rights. Due to double-buffering of the data-blocks the subroutine of a real-time interrupt is able to gain access to consistent data at any time. *System Diagnostics*, *System Consistency Checks* and software modules (and replicas) in this context are protected against faulty application tasks and maintain a higher level of isolation. All required input and output signals are interfaced by the centralized database.

E. System Consistency Check

The watchdog system of a single MCU is not able to detect all possible failure modes on its own especially due to the common power supply. To overcome this problem, the watchdog of each MCU serves as an external watchdog of the other MCU. This functionality is implemented on both MCUs as a part of the *System Consistency Check* in the third diagnostics level. Moreover, the *BMS-Engine* transfers data blocks via an SPI interface between both MCUs. Safety-critical data blocks replicate the following information:

- Measured values of the cell voltages, cell temperatures, battery current, balancing control and monitoring
- Estimated values of battery states(SOC, SOF)
- BMS diagnostics (SOA-violations)
- System monitoring (task states and notifications with timestamps, interrupt counter, etc.)

According to the adapted peripheral hardware (for redundancy and diversity purpose) and the required safety measures, this allows the distribution and replication of diagnostic functions and software modules on both microcontrollers.

V. CONCLUSION

Redundancy, partitioning and the implementation of diagnostic functions is suggested as a technique to improve the safety integrity of a BMS. To address this, we have introduced various solutions in adapting error handling and safety-critical components to different hardware and software concepts of a BMS. Based on the flexible and extensible hardware architecture, and a centralized database, different kind of designs for redundancy and partitioning are feasible. As the hardware design and all software components are open-source and freely available, investigations in different safety strategies, the comparision of different model-based algorithms and their effects on both the system performance and the safety goals can easily be evaluated.

ACKNOWLEDGMENT

The research leading to these results has received funding as part of the Energie Campus Nürnberg (“EnCN”) which is financed by the State of Bavaria as part of the program Bavaria on the move, and the SEEDs project which is funded by the Bavarian State Ministry of Economic Affairs, Infrastructure, Transport and Technology in the framework of the Bavarian initiative for research and development of technology in the energy sector.

REFERENCES

[1] Gregory L. Plett, “Battery Management Systems, Volume 1, Battery Modeling”, 2015

[2] M. Giegerich, M. Akdere, C. Freund, T. Fühner, J.L. Grosch, S. Koffel, R. Schwarz, S. Waldhör, M. Wenger, V.R.H. Lorentz, M. März, “Open, Flexible and Extensible Battery Management System for Lithium-Ion Batteries in Mobile and Stationary Applications”, IEEE IES ISIE, June 2016

[3] B. Lunz, Z. Yan, J. B. Gerschler, D. Sauer, “Influence of plug-in hybrid electric vehicle charging strategies on charging and battery degradation costs”, Energy Policy, Volume 46, July 2012

[4] Waldhoer, S. “Development and Implementation of an Electrochemical Battery Model,” Master’s Thesis, University of Erlangen-Nuremberg, 2015

[5] M. Brandl, H. Gall, M. Wenger, V. Lorentz, M. Giegerich, F. Baronti, G. Fantechi, L. Fanucci, R. Roncella, R. Saletti, S. Saponara, A. Thaler, M. Cifrain, W. Prochazka, “Batteries and battery management systems for electric vehicles", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012 , vol., no., pp.971,976, 12-16, March 2012

[6] “Managing memory protection unit (MPU) in STM32 MCUs”, Application Node AN4838, http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00272912.pdf

[7] F. Bruns, D. Kuschnerus, A. Showk, A. Bilgic, “An Extensible Partitioning Framework for Safety-Critical Systems”, ERTS2 2012 Embedded Realtime Software and Systems, February 2012 (http://web1.see.asso.fr/erts2012/...)

[8] B. Shah, B. Krishnamurthy, “Implementation of MPU for a Safe FreeRTOS Frame-work”, International Journal on Recent and Innovation Trends in Computing and Communication, IJRITCC, May 2015

[9] “FreeRTOS homepage.” [Online]. Available: http://www.freertos.org/