# A brief record of upgrading a Battery Management System

1st Cosmin-Ionuț Năstase
*dept. Electronică și Calculatoare*
*Universitatea Transilvania din Brașov*
Brașov, România
cosmin-ionut.nastase@student.unitbv.ro

***Abstract*—In today's evolution of alternative energy , objects such as electric vehicles requires a lot of batteries for normal usage, like an conventional fuel powered vehicle. So like in the classical version of the cars, where we need to have fuel management systems in order to operate safely and reliable , in the EVs their batteries needs to be managed correctly in order to achieve maximum reliability from them.So for correct managing of the battery pack is absolutely essential to use a so-called Battery Management System that can monitor each vital parameter for each battery in order to balance load to each other, in order to prevent premature failure. This project focuses on building a part of a battery management system, the sensing part of the BMS , developing firmware using FreeRTOS(real-time operating system) and testing the actual software using an AVR development board and sensors.**

***Index Terms*—AVR, BMS, battery pack, RTOS ,temperature**

## I. Introduction

By its simplistic design, a battery proves to be a very reliable piece of equipment in a system when it is used correctly. Every manufacturer gives to the end-users a datasheet of its battery product, where we can find the nominal parameters of that battery and a what-so-called SoC(State-of-Charge) versus SoD(State-of-Discharge) graph , that indicates the optimal functioning curve of that battery for its nominal capacity. Both SoC and SoD graph are equal necessary in a reliable system , because a manufacturer guarantees a number of working hours or Charge-Discharge cycles for that battery product without any of its internal parameters to be affected.

Of course, SoC or SoD mismatch can cause in the most cases premature failure of that battery, but that is not the only cause that can lead to battery failure. Because we assume that a battery pack for a EV or Smart Grid is made of "x" cells configured each in parallel/series configuration with its neighbour, each cell internal impedance can differ , so the self-discharge rate of each cell is not equal to the other ones, also the charging characteristic is different. In a nutshell, the charging process in this case can be affected by stopping the charge too early because one or more of that battery cells reached maximum capacity and the system considered that pack fully charged. In reality , some of those batteries are not completely charged , so SoC curve could not be achieved. The same principle is applied also when the cells get discharged because of the normal operation, the less charged cells get a more profound discharge than the other, thus this fact can irreversible damage them. This is what is called by the manufacturers improper operation of their batteries. In this case they can not guarantee a very long lifespan of their products.

### A. Importance of a BMS in renewable energy systems or EVs

Because the importance of SoC and SoD of a battery for the correct operation of a battery pack, it is crucial to implement a system which figures the control and energy management for each cell.For that it is necessary to implement algorithms which follow evolution in time of the base parameters for each cell (cell voltage , charge current , discharge current and cell temperature) [1] Using this base parameters a BMS needs to be capable to maintain a correct SoC curve in normal operation. With this main capability comes other secondary calculations which needs to get some attention from the system, like SoH(State-of-Health), estimated lifespan, capability to signal faults in the system from which the end-user relies on easy diagnosis.

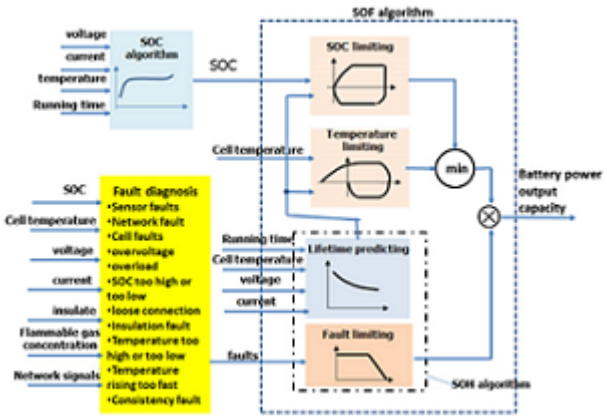


Fig. 1. BMS parametric estimation block [2]

### B. Objectives of a BMS

So the main usage of a BMS in renewable energy systems is to maintain a maximum lifespan of each battery cell by monitoring the base parameters for them. Thus it is necessary

that the system needs to adapt the battery pack's characteristics in real time to achieve maximum reliability for a long operating time. And because main talking highlights the usage of batteries in EVs, those systems needs to manage energy from each cell of the battery pretty much wisely to obtain maximum working range between the charges. As I mentioned before, the algorithm behind this system needs to be implemented in close connection with the base parameters of the manufacturers datasheet for the batteries used in the pack. Starting with that parameters, the system needs to be flexible enough to adapt that cells to optimum working conditions. Thus is needed to measure each cell parameter in real time so that system "knows" if that cell is healthy or can lead to damage the battery-pack because of the parameter mismatch, or SoC/SoD incorrectly chosen. [3]

To set a correct global SoC characteristic a BMS "learn" the charging and discharging behavior of the cell pack by making successive charge-discharge events, in which the system compares the base parameters with the measured ones and makes adjustments to achieve a better SoC/SoD curve. Of course we cannot create an ideal situation because of the each cell internal impedance. As we presumably charge that pack at the optimum parameters, the BMS reads terminal voltages of each cell or pair of cell according to the chosen topology. But the terminal voltage always differ to OCV(open-circuit voltage), because in this case the internal impedance influences the measurement by drawing an internal residual current in every cell.

SoC Calculation is based on the terminal voltage of the cells, so the settling time of each cell in part is very important because of a correct SoC. With a correct SoC now calculated we can approach cell balancing, which is a crucial task of an any BMS. The main approaches for this task is bottom , mid or top-balancing. Mid or top-balancing are the most used methods to balance cell packs because most of EVs manufacturers use them for maintain the battery at a 50 percent SoC state, because in that case the battery can store more energy from a future charging or from external events, like energy recovery from braking.

## II. Objective of this project

The aim of this project was to upgrade an existing battery management system , based on a proper design using a Master-Slave topology to which they were used Low-Power PIC16 microcontrollers to implement the BMS algorithm for SoC calculation and charging capability respectively , and also to implement a Master circuit, which role is to gather data from every BMS circuit and compare them with some existent data to see if the cells are used correctly or the BMSs needs some fine tuning. In the original design, those BMSs come with a LIN(k-Line) communication between them and master which was made using the UART peripheral of the microcontroller used, of which messages were translated to k-Line using a dedicate transciever ASIC.

In the part of the project that i've managed to made, i wanted to see if a possible architecture change can achieve more capabilities to make BMS more flexible in eventual changes like different chemistry cells or a powerful power supply etc.

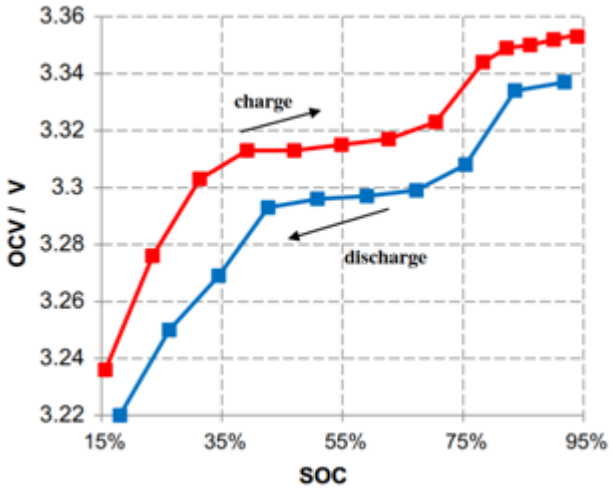


Fig. 2. SoC versus SoD curves for a LiFePO4 cell [2]

The actual part of the project treats the problem of reading the temperature per cell pack using an one-wire temperature/humidity sensor from the Asair AM family, sensor data were acquired by an ATMega128 microcontroller which is used to run a a loop program for controlling those parameters, and then decode and print them to an LCD matrix display. As for external communication, the system is able to send parameters to a terminal through an RS232 connection.

Despite the other project communication, i want to extend this transmission to CAN, which is more disruptive-proof than the k-Line connection, which is good also but for shorter transmission distances.

### A. Making use of FreeRTOS

The original project was built as a finite-state machine, but without an dedicated operating system.

With this project i try to integrate a real-time operating system to manage the actual functions that needs to be executed in order to make a working system.

In a classical manner, every function declared in a code is executed at a certain time in the program (basically it is executed when it is called from the loop) or at a special time(when an interrupt happens and that function is called from the interrupt itself). In a nutshell, they are 2 types of functions depending by execution time of that function itself:

1.Non-time critical functions - which are running in the background of the program,they have low priority in execution

2.Critical time functions - functions with the highest priority in the program chain. They need to be executed at a certain time in a certain time for the system to work properly.

With the classic approach , the time critical issue is treaten with interrupts, because they can be triggered externally, by an event for example, or internally , by repeatedly triggering a flag from an internal counter for example.

In the other approach, we can use an operating system as a middleware between hardware and actual software in order to treat with execution time of each function in the chain. Actually, every known real-time operating system for some of families of micro-controllers on the actual market or development boards uses a handful of methods to make the ”important” functions work at the perfect timing. Compared to the classical approach , a RTOS treats with every function declared in the program via tasks. A task is able to set to a function a certain grade of priority, which in the case of FreeRTOS starts from the IDLE priority,background tick, and can be incrementally raised to a higher level depending of the priority we need to have for a certain function.

### *B. Implementation of FreeRTOS in the project*

Because before this project I haven't experienced any program with FreeRTOS it was pretty much a learning experience, and for that the actual project is presented from a conceptual-wise vision.

## III. Materials used and implementation

As a hardware for developing this project I used some pretty common parts to start interacting with FreeRTOS, parts that I can reuse in the future developing of the final project.

Software-wise , as I have mentioned before the main focus will be on implementing FreeRTOS in the actual program to see how complex is to implement a complete device firmware using this middleware. Of course, the whole program will be written using C and some assembler commands(such as ”nop” command).

The main focus is to interface correctly the LCD Matrix and AMS/DHT sensor to ATMega128, by developing dedicated libraries for each one , and also outputting serial data ,in this case, to computer to see actual continous working of the implemented system.

### *A. System architecture*

The following block diagram summarizes the main components of the system and the way that they interact with each other.(Fig.3)

As it will be seen in the figure, the microcontroller performs tasks that treats every component of the system itself at a desired time. So the system will gradually perform read-write operations for sensor to achieve the desired data from it.

The chosen sensor (Asair AMS2302) is a combo between a temperature sensor implemented with a NTC thermistor and a pelicular humidity sensing cell. An ASIC collects voltage information from those two elements, an then codes them in a 40 bit header with information collected from elements.

This header is formatted like:

- 16bits-Humidity Info
- 16bits-Temperature Info
- 8bits-Checksum

It is clear that every 16 bits of data transmitted by the sensor contains info for humidity or temperature. But the checksum is also important for the system because if the sum of bits of the data does not correspond to the checksum, that header is ignored and the system waits for new data from the sensor at the next request.

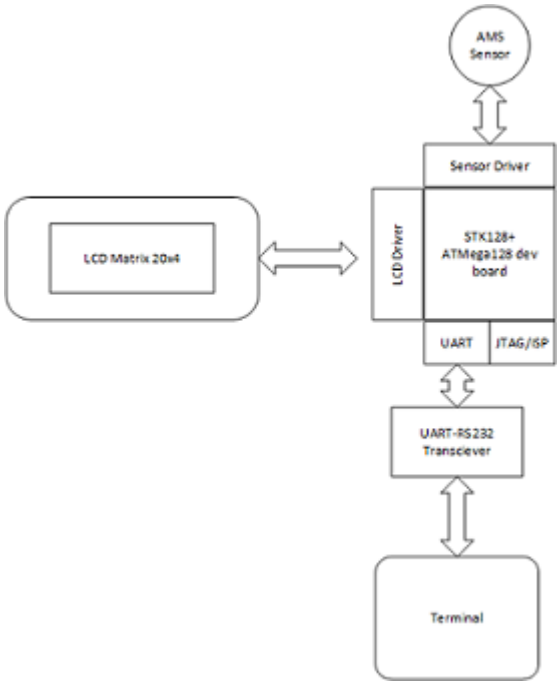


Fig. 3. Block diagram of the implemented system

The data acquired from the sensor's elements is precise compared to a dedicated thermometer(the difference between the sensor and the thermometer is like 0.1 degrees Celsius in the most cases).

But the major flaw of this family of sensors is acquisition time. The time between 2 requests made by the microcontroller is recommended to be like 2 seconds or more in order to reduce possible misreadings from the ASIC.

When the header sent by the sensor reaches the microcontroller, the driver written for this element checks if the data contained in the header is good by making the sum of information bits and comparing it with the checksum. And if data is good then a specific function splits the header in humidity/temperature parameters, which will be inherited in LCD printing task.

The main operation of the system is split between 2 main tasks, one which treats with verifying if sensor is ready to send data , which triggers a binary semaphore that stays active until the data is gathered. When data is achieved the semaphore releases and the main task continues to update the LCD with the current parameters.

When data from sensor gets updated on the LCD , a message is sent to RS232 to report the status of the system , which is decoded by the terminal, using the specific baud rate for the transmission. Further implementations can support receiving messages from the terminal and decoding them properly.

As it can be seen in Fig.4, the main software loop task is described using a flowchart in which main processes of each

task are overviewed.

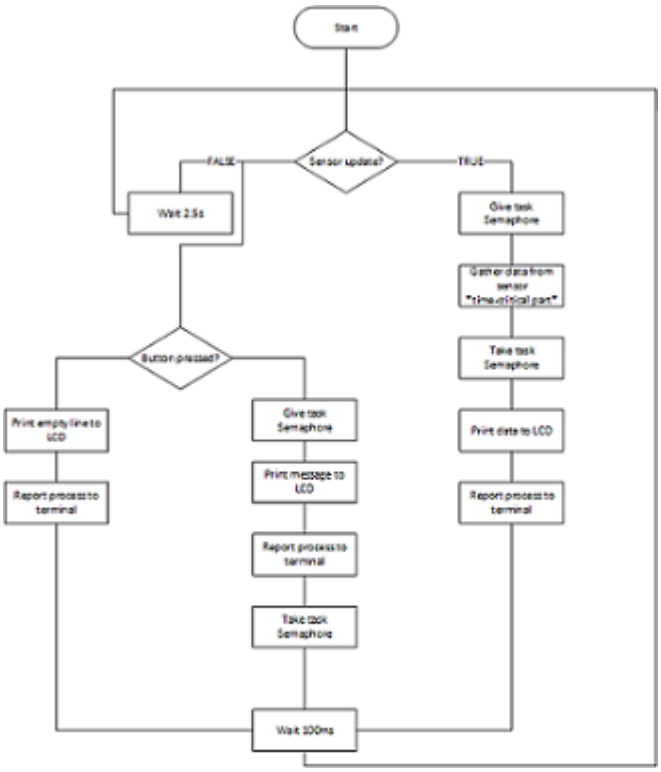


Fig. 4. Flowchart diagram of the implemented system

## IV. Testing the algorithm and drivers

For this part the testing started gradually, so after a driver written , it was tested both in a classic approach and after with FreeRTOS. This was a pretty good exercise to see if the functions that were developed specific to work with RTOS can also be used without any middleware.

First driver developed, after making some basic functions to get used with the RTOS commands was the LCD driver. This was made with an extensive functionality , to have a pretty good base for future upgrades to the system. Te main capability of the driver is that the parameters are written directly in RAM and after tat the LCD receives command to read data directly from RAM. Parallel adressing is used to fulfil this feature.

The second developed for this project was the sensor interfacing driver. Because this family of sensors has one-wire adressing mode, it is necessary for the microcontroller to send requests to the sensor's ASIC at a certain time using time-determined impulses to trigger data acquisition. After the request ended, sensor's chip send the header of data that i have mentioned before. The main task of the driver is to check if the data received from the sensor is correct using checksum verification , and if is good the header gets split in temperature information and humidity information. Because the request header is strictly determined in time , this driver needs to operate time-critical.

The third driver developed is an UART driver , which is able to send or receive messages like strings or bytes. In a further implementation of kLine or CAN communication the UART driver needs to be capable to send bytes to the transcievers to make the configuration message. In this part of the project, this driver is used to send messages to a terminal via RS232 communication to see if the tasks generated are working properly.

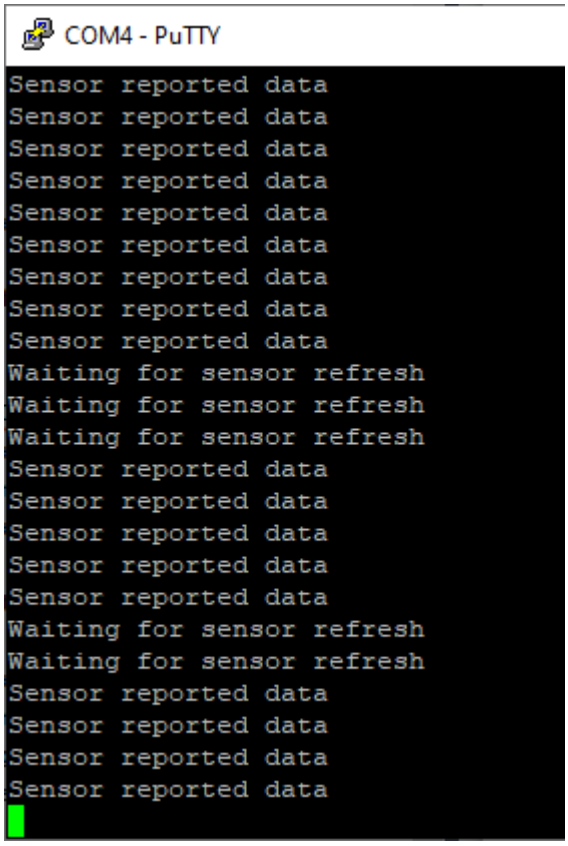


Fig. 5. Task reporting via RS232 in PuTTY terminal

As for further testing , after putting the drivers in the same place and creating the necessary tasks for handling correct data from sensor and outputting it through LCD Driver according to the flowchart from Fig.4 I started to make successive tests which concluded some timing issues in the sensor driver. After the timing issue was corrected , i've made a comparison of the data outputted by the sensor with an actual thermometer to see if the temperature is reported correctly. The test was a success , because as it will be seen in Fig.6 , the data displayed on the LCD Matrix by the system is equal to the temperature readen from the thermometer.

## V. Conclusions and further modifications

With this project i can say that i have learned some new features regarding to firmware development. FreeRTOS is a pretty powerful middleware to treat the timing problems of different parts of the code written, but is a very very vast domain to learn. With this actual operating sistem I saw the principal limitation in the future upgrade of the original BMS design. The AVR microcontroller chosen for this application is


Fig. 6. Comparison between a calibrated thermometer and data from the sensor

yet powerful for the actual needs of the system, but the major flaw in the whole story is te power drawn of it. It is known that in mobile domain like automotive it is needed to design a system with very low power consumption, and in this case this implies migrating to a series of microcontroller with low power, like STM32Lxx series. But as for learning purposes and driver development that can be easily migrated to other platforms , the AVR8 series was a good choice.

Another conclusion is that digital temperature sensors can be successfully used in a BMS application, because they can give a pretty good measurement for this application. It is known that operating temperature is a vital parameter in a correct usage of batteries. Because battery cells give the best efficency in the area of 25-40 degree Celsius it is crucial that the cell pack temperature to be monitored constantly and maintained in that interval. Also humidity measurement is a good indicator if a cell is posibly faulty(cell leakage can happen to every type of battery) or the battery pack lost its external seal, sign that the batteries can be damaged in the future.

## REFERENCES

[1] K. Cheng, "Battery management system(bms) and soc development for electrical vehicles," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 1, pp. 76–88, 2011.

[2] S. F. Tie and C. W. Tan, "A review of energy sources and energy management system in electric vehicles," *Renewable and Sustainable Energy Reviews*, vol. 20, pp. 82 – 102, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032112006910

[3] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, "A review on the key issues for lithium-ion battery management in electric vehicles," *Journal of Power Sources*, vol. 226, pp. 272 – 288, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378775312016163