CONSULTANCY PROJECT REPORT

IOT BASED REMOTE MONITORING AND APP DEVELOPMENT FOR BATTERY MANAGEMENTSYSTEM

- DATA ACQUISITION FROM BMS TO CLOUD DATABASE

Submitted by

GIRISH R (2020505005)

HARISH REDDY A (2020505007)

Guided by

DR. N. PAPPA (Professor, Dept. of IE)
DR. S. SUTHA (Professor, Dept. of IE)
DR. S. MEYAPPAN (Asst. Professor, Dept. of IE)

Sponsored by

AMPTON ENERGY
CHENNAI

MADRAS INSTITUTE OF TECHNOLOGY CAMPUS ANNA UNIVERSITY: CHENNAI-600 044

BONAFIDE CERTIFICATE

Certified that the consultancy project report titled "**IOT BASED REMOTE MONITORING AND APP DEVELOPMENT FOR BATTERY MANAGEMENT SYSTEM"** is the bonafide work of

GIRISH R	**2020505005**
HARISH REDDY A	**2020505007**

who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here does not form part of any project on the basis of which degree or award was conferred on an earlier occasion on this or anyother candidate.

SIGNATURE	**SIGNATURE**
Dr. N. PAPPA	**Dr. S. MEYYAPPAN**
PROJECT HEAD SUPERVISOR	**SUPERVISOR**
Professor,	Assistant Professor,
Dept. of Instrumentation Engg.	Dept. of Instrumentation Engg.
MIT Campus, Anna University	MIT Campus, Anna University
Chromepet, Chennai - 600044	Chromepet, Chennai-600044

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **Dr. J. PRAKASH**, Dean, Madras Institute of Technology, for having given consent to carry out the project work at Madras Institute of Technology Campus, Anna University.

We are grateful to **Dr. S. SRINIVASAN**, Professor and Head of the Department of Instrumentation Engineering, Madras Institute of Technology Campus, Anna University for being a constant support.

We express our gratitude to our project head supervisor, **Dr. N. PAPPA**, Professor, Department of Instrumentation Engineering, and **Dr. S. SUTHA,** Professor, Department of Instrumentation Engineering, for providing a chance to take on the consultancy project.

We extend our sincere gratitude to our project guide, **Dr. S. MEYYAPPAN**, Assistant Professor, Department of Instrumentation Engineering, guiding us through every phase of the project. We are grateful for his thoroughness, tolerance and ability to impart his knowledge to us. Apart from adding his own input, he has encouraged us to think on our own and give form to our thoughts.

GIRISH R	**2020505005**
HARISH REDDY A	**2020505007**

ABSTRACT

Battery Management System is an electronic system that manages a battery pack by performing various functions such as monitoring, optimizing performance and protection. The project revolves around deriving a strategy and designing a system with the given BMS to perform data acquisition and remote monitoring through Web and Mobile Application. The Hardware part of the problem statement focuses mainly on understanding the given Daly's Smart BMS in terms of working and communication with host. The work involves selection of suitable hardware and database for data acquisition and data storage respectively. The system is developed performs the following functions namely data acquisition of all parameters from BMS, data validation (Error check) and pushing it into a real-time database.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Overview

The problem statement aims at acquiring various parameters of the Smart BMS - Daly's Smart BMS LifePO4 19S 60V 30A and providing IoT enablement for the purpose of remote monitoring. It involves selection of suitable microcontrollers for data acquisition and also for IoT purposes, selection of databases for data storage and understanding the communication and data format used by the given BMS. Raspberry Pi Model 3B+ is used for communication with BMS and also acts as a gateway. Google Firebase is chosen as a data storage platform to store BMS data. All the parameters from the BMS are stored in the real-time database with time-stamps. An option is also provided to vary the sampling rate of the data according to requirement. By default, the sampling rate is 1 sample/minute. The system is designed in such a way that all the parameters from the BMS are updated in the real-time database every minute with time-stamp generated.

1.2 Working flow diagram of system

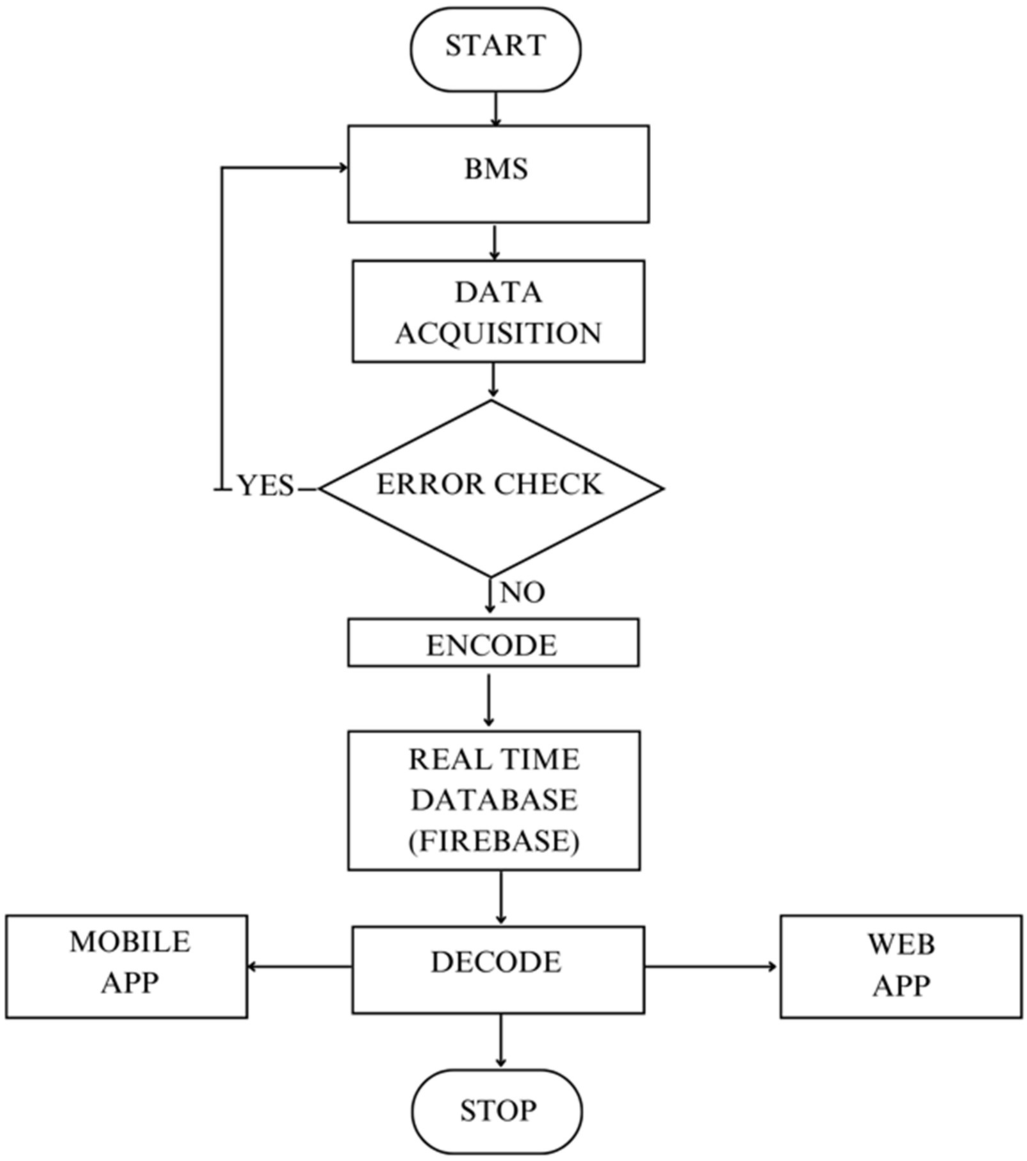


Fig 1.1 Working flow diagram of the system

CHAPTER 2

HARDWARE DESCRIPTION

2.1 BMS hardware connections diagram

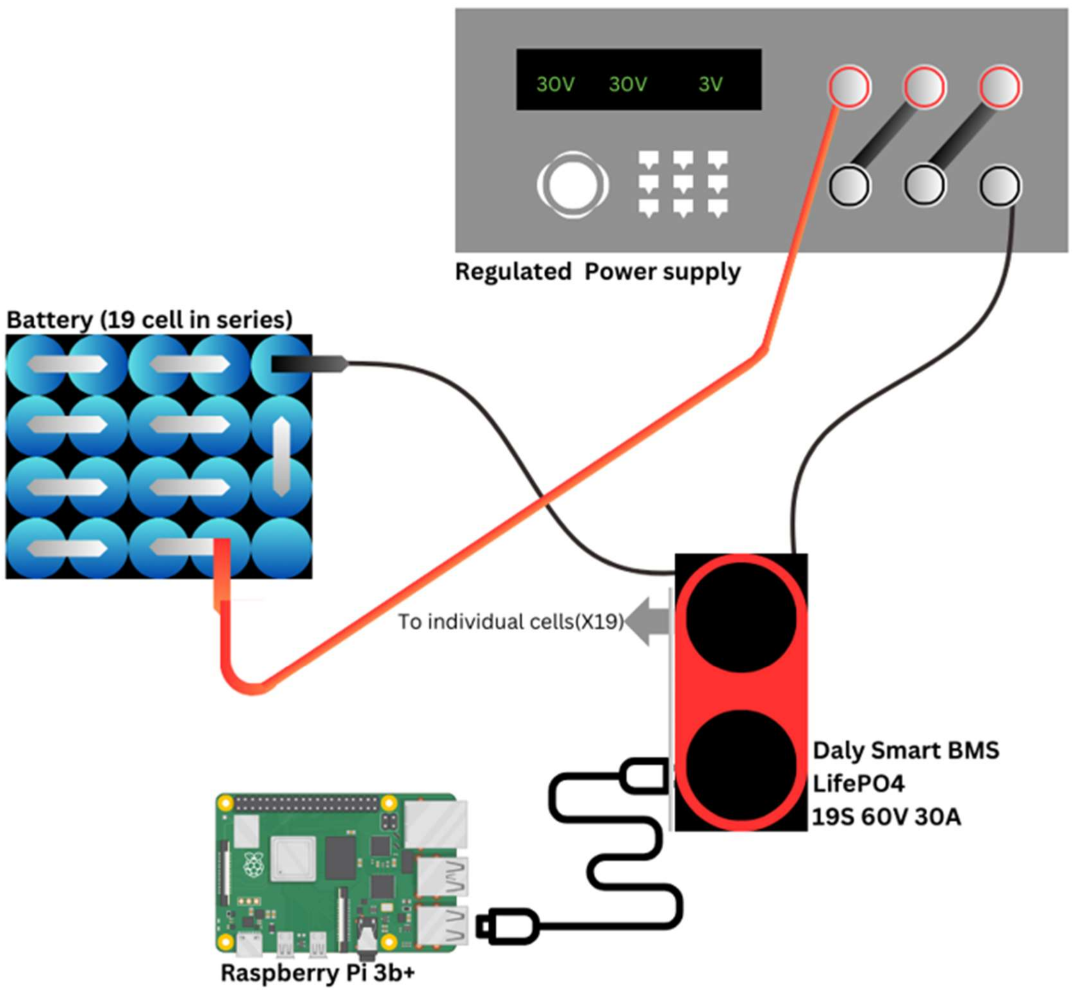


Fig 2.1 Hardware connection details

2.2 BMS Setup

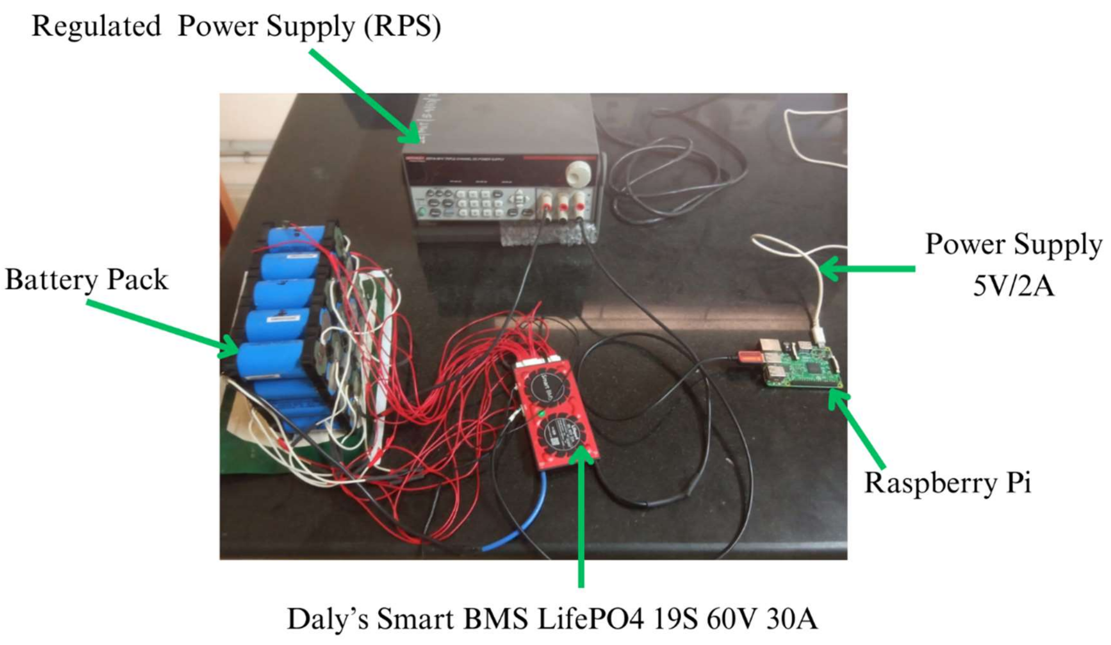

Regulated Power Supply (RPS)
Battery Pack
Power Supply 5V/2A
Raspberry Pi
Daly's Smart BMS LifePO4 19S 60V 30A

Fig 2.2 Real-time setup details

2.3. BMS setup description

With reference to the Fig 2.1, the hardware consists of 4 components - Regulated Power Supply (RPS), Battery Pack (19 Cells), Daly's Smart BMS LifePO4 19S 60V 30A and Raspberry Pi 3B+. The components RPS, Battery Pack and Daly's Smart BMS LifePO4 19S 60V 30A are connected in series. The terminals of RPS are connected in series in order to produce 63-65V to charge the battery pack of 60V.

The way of series connection is as follows:

The battery pack consists of 19 cells connected in series, the positive terminal of the battery is connected to the positive terminal of power supply (RPS) and the negative terminal is connected to the P-terminal of Daly's Smart BMS. The B-terminal of BMS is connected to the negative terminal of the power supply (RPS).

The Raspberry Pi is connected to the Daly's Smart BMS via UART (Universal Asynchronous Transmitter Receiver) to USB cable (which is provided by Daly's BMS originally for their windows app - PC Master). The Raspberry Pi communicates with Daly's Smart BMS by UART communication (One of the types of serial communication).

The description of abbreviation '19S 60V 30A' in the Daly's Smart BMS LifePO4 19S 60V 30A is

- 19S - Given BMS can measure the parameters of a battery pack containing 19 cells.
- 60V - Given BMS can measure the voltages of the battery pack, whose total voltage is lesser or equal to 60V.
- 30V - Given BMS can measure the voltages of the battery pack, whose total current is lesser or equal to 30A.

The BMS contains required temperature and current sensing circuitry along with a microprocessor which serves dual purpose, first one is storage of battery parameters in its registers (0x90 to 0x98 - In Daly's Smart BMS LifePO4 19S 60V 30A) and second is enablement of serial communication of Host/Master devices with microprocessor in it). The procedure for data acquisition from the registers of BMS is explained in detail in Chapter 4.

CHAPTER 3

HARDWARES, SOFTWARES AND LIBRARIES

Table 3.1 Table showing hardware requirement list

3.1 HARDWARE LIST	
S. No.	**Name**
1.	Raspberry Pi Model 3B+
2.	Power Adapter 5V/2.5A with USB B-type(Micro USB)
3.	HDMI Cables (Required for Programming)
4.	Regulated Power Supply for Battery Pack

Table 3.2 Table showing software requirement list

3.2 SOFTWARE LIST	
S. No.	**Name**
1.	Raspberry Pi Imager
2.	Raspberry Pi OS Version - 1.7.5
3.	Python version 3.11
4.	VNC Viewer

Table 3.3 Table showing libraries requirement list

3.3 LIBRARIES LIST		
S. No.	**Name**	**Purpose**
1.	pyrebase	Google Firebase functions
2.	serial	Pyserial API - Serial communication functions
3.	time	Time stamp generation and delay
4.	datetime	Time stamp generation

CHAPTER 4

DATA ACQUISITION FROM BMS

4.1 Communication format and message frame structure

Daly Smart BMS LifePO4 19S 60V 30A uses Serial Communication UART with a request-response model of communication.
The communication format used is 9600, N,8,1. It means it operates at a 9600 baud rate, no parity bit, 8 data bits and 1 stop bit for data transmission.
For data acquisition, a request message frame (format specified below) is sent with the required address to BMS and the BMS responds to it. The request frame format (From Raspberry Pi) shown in Fig 4.1. The response frame format (From BMS) shown in Fig 4.2.

Start Flag	PC address	Data ID	Data length	Data Content	Checksum (1 Byte)
0xA5(Fixed)	0x40(UPPER-ADD)	Refer to Section3	8 Bytes(fixed)		

Fig 4.1 Request Frame format used in UART communication

Start Flag	PC address	Data ID	Data length	Data Content	Checksum (1 Byte)
0xA5(Fixed)	0x01(UPPER-ADD)	Refer to Section3	8 Bytes(fixed)		

Fig 4.2 Response Frame format used in UART communication

In general, both request and response message frames are of 13 bytes in which request frame has the address of Host/Raspberry Pi with no data content (All 0s) and response frame has the address of BMS/Slave with data content filled with values.

The Checksum byte for the message frames depends upon the content in the frame (more explained in detail in)

4.2 Drivers used and compatibility

Daly Smart BMS LifePO4 19S 60V 30A UART cable uses CH341 driver for UART communication with the Windows Application PC Master (Originally created by manufacturer). The driver belongs to the family of CH34x drivers.
CH34x are the chips that are commonly used for USB to Serial Conversion. Since BMS uses CH340 chips for serial communication, it is required to install a CH340 driver in the host from which the request messages are sent and response messages are received.
The function of CH340 is shown in Fig 4.3.

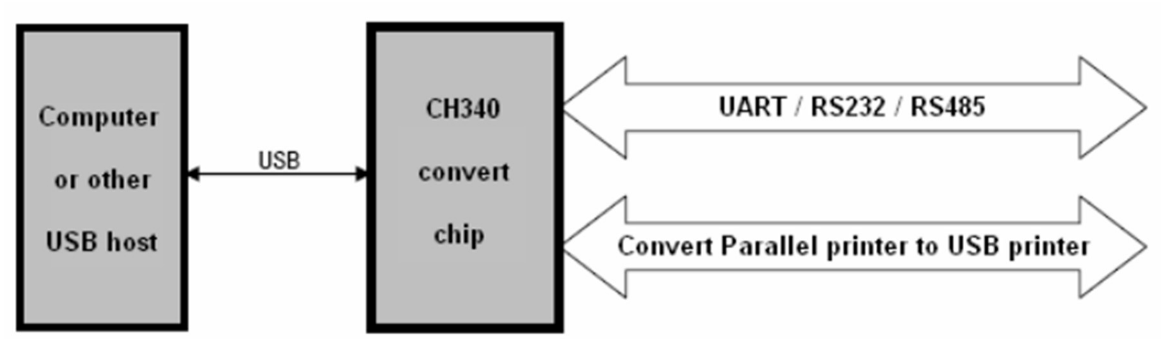


Fig 4.3 Picture depicting the working of CH340 chip and driver

The function of CH341 is shown in Fig 4.4.

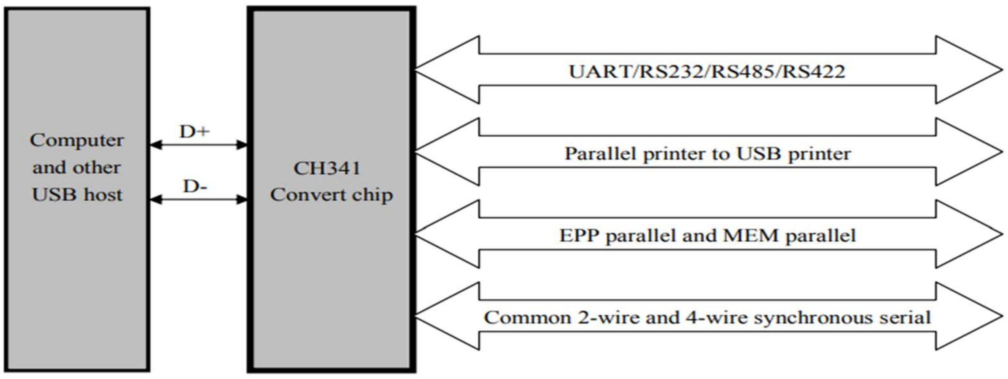

Computer
and other
USB host
D+
D-
CH341
Convert chip
UART/RS232/RS485/RS422
Parallel printer to USB printer
EPP parallel and MEM parallel
Common 2-wire and 4-wire synchronous serial

Fig 4.4 Picture depicting the working of CH341 chip and driver

In the case of the CH340 driver used by Daly Smart BMS LifePO4 19S 60V 30A, the specific driver is installed by default in the USB Ports of Raspberry Pi Model 3B+. So, the data is acquired by connecting the USB cable of BMS to the USB port of Raspberry Pi.

4.3 Register access in BMS

In Daly Smart BMS LifePO4 19S 60V 30A has a total of 9 data registers from addresses 0x90 to 0x98. The data message descriptions of each register are given in Table 4.1

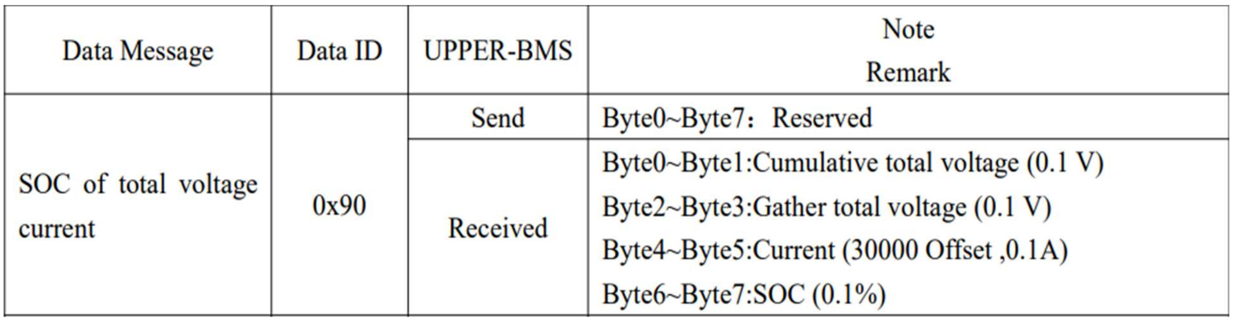
Table 4.1: Table showing different data messages in BMS

Register Address	Data message
0x90	SOC of total voltage current
0x91	Maximum & Minimum voltage
0x92	Maximum & Minimum temperature
0x93	Charge & discharge MOS status
0x94	Status information 1
0x95	Cell voltage 1~48
0x96	Cell temperature 1~16
0x97	Cell balance State 1~48
0x98	Battery failure status

The parameters in each data register location are unique and its decoding logic is unique for each and every address. The information about how to get the parameters from each data message is mentioned in the communication protocol datasheet which is attached with this document in Appendix (Chapter 7).

4.3.1 Data decoding example for single frame data message (all register addresses except 0x95 and 0x96)

The description of data message for register address 0x90 is shown in Fig 4.5.



Data Message	Data ID	UPPER-BMS	Note Remark
SOC of total voltage current	0x90	Send	Byte0~Byte7：Reserved
		Received	Byte0~Byte1:Cumulative total voltage (0.1 V) Byte2~Byte3:Gather total voltage (0.1 V) Byte4~Byte5:Current (30000 Offset ,0.1A) Byte6~Byte7:SOC (0.1%)

Fig 4.5 Data message description of register 0x90

For example, if the message data of 0x90 is a5 01 90 08 02 89 00 00 75 30 01 e6 55. With the help of remarks in Received row after ‘08’ byte, the following next 8 bytes are data bytes (For any frame)

- For Byte 0 ~ Byte 1: 02 89 conversions into decimal gives 649, there is a unit given in the brackets of Cumulative total voltage 0.1V, so multiplying 649 with 0.1V results in 64.9V

  Cumulative total voltage = 64.9V.
- For Byte 2 ~ Byte 3: 00 00 => 0 (in decimal) , 0*0.1V = 0V

  Gather total voltage = 0V.
- For Byte 4 ~ Byte 5: 75 30 => 30000(in decimal)

  (30000-30000) *0.1A = 0A (30000 is subtracted from the resulted

value because 30000 is offset mentioned in the Received row)
Current: 0A

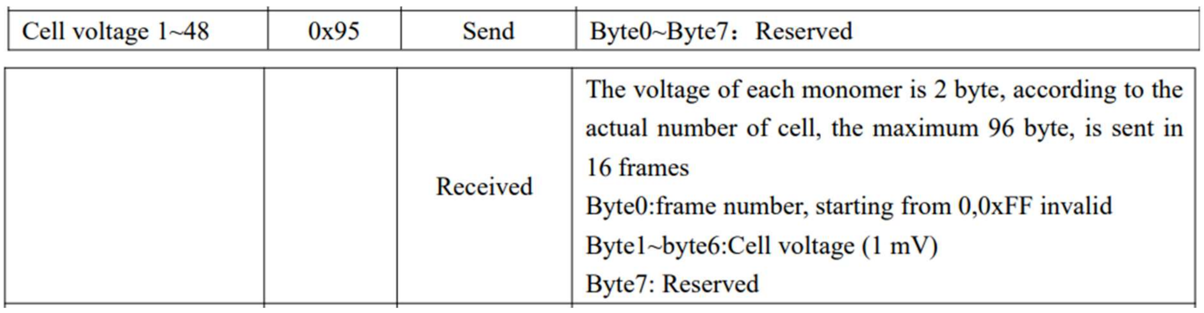
- For Byte 6 ~ Byte 7: 01 e6 => 486(in decimal)

  SOC = 486 * 0.1 % = 48.6% (Refer to description of SOC in the datasheet)

Similarly, by seeing the remarks of all received data messages, the parameters mentioned in the datasheet can be interpreted. (Note: follow the conversion given in brackets near to each parameter multiply them (in some cases - subtraction with offset) with the decimal value to get the required parameter value with units)
The above example process is followed for all other register addresses (Except 0x95 and 0x96) with respective data message descriptions/remarks.

4.3.2 Data decoding strategy for multi-frame data message (register addresses 0x95 and 0x96)

According to the datasheet description of 0X95 Register Address (shown in Fig 4.6) (In the Communications content information in datasheet),
For the Daly Smart BMS LifePO4 19S 60V 30A, there are 19 cells. In each frame 3 cell voltages are sent with cell voltage 2 bytes each. (Every Frame starts with 0xa5 byte).
First Byte represents the Frame Number (If 48 cells are present then, the number of frames is 48/3 = 16, 3 cell voltages in each frame), the next 6 bytes (Byte 1 ~ Byte 6) of each frame corresponds to 3 cell voltages and the last byte Byte 7 is the reserved byte.



Cell voltage 1~48	0x95	Send	Byte0~Byte7：Reserved
		Received	The voltage of each monomer is 2 byte, according to the actual number of cell, the maximum 96 byte, is sent in 16 frames Byte0:frame number, starting from 0,0xFF invalid Byte1~byte6:Cell voltage (1 mV) Byte7: Reserved

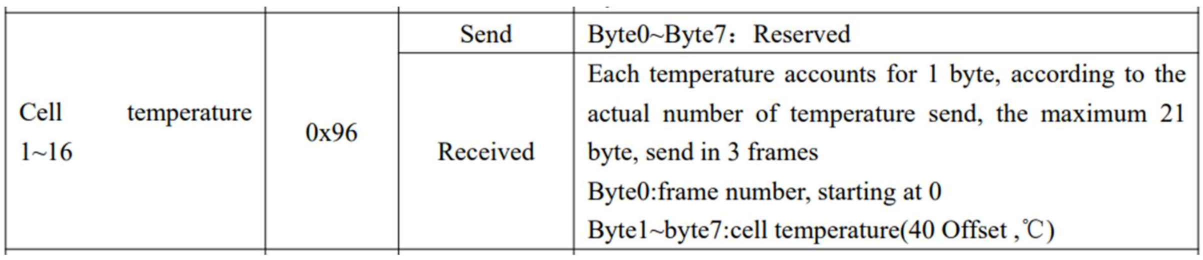
Fig 4.6 Data message description of register 0x95

According to the datasheet description of 0X96 Register Address (shown in Fig 4.7) (In the Communications content information in datasheet – attached in Appendix Chapter 7).

For the Daly Smart BMS LifePO4 19S 60V 30A, there are 19 cells. In each frame 7 cell temperatures are sent with cell temperature 1 byte each. (Every Frame starts with 0xa5 byte).

First Byte represents the Frame Number (Maximum 3 frames value - 0x00 to 0x02), the next 7 bytes (Byte 1 ~ Byte 7) of each frame corresponds to 7 cell temperatures.

The value of temperature (in decimal) should be subtracted with 40 Deg C to obtain the cell temperature value in Deg C.



Cell temperature 1~16	0x96	Send	Byte0~Byte7：Reserved
		Received	Each temperature accounts for 1 byte, according to the actual number of temperature send, the maximum 21 byte, send in 3 frames Byte0:frame number, starting at 0 Byte1~byte7:cell temperature(40 Offset ,℃)

Fig 4.7 Data message description of register 0x96

4.4 Error checking through checksum (all frames)

Last byte of every frame (response as well as request frames) is allocated to Checksum. Checksum is an error detection strategy used to find whether there is any modification of data during transmission from sender to receiver.

Example of how error is checked is given below for a response message from 0x90 address,
a5 01 90 08 02 89 00 00 75 30 01 e6 55
55 - Last Byte of message is checksum value sent by the BMS, upon receiving the following operations are performed

- Sum of all the bytes from Byte 0 to Byte 12 leaving checksum byte
- Taking the Lowermost 2 digits of the sum expressed in Hex format, taking it as calculated checksum.
- Comparing the calculated checksum value with Original Checksum byte in the frame, if equal then no error is present if not equal, error is present and the frame is received again from BMS.

Performing the above operations on the given example frame,
Sum = a5+01+90+08+02+89+00+00+75+30+01+e6 = 355(in Hex)
Lowermost 2 digits of Sum (in Hex) = 55.
The calculated checksum is equal to the checksum byte present in the message, so the data has no error.

4.5 OVERALL DATA ACQUISITION FLOW DIAGRAM FOR ANY ADDRESS (0X90 TO 0X98)

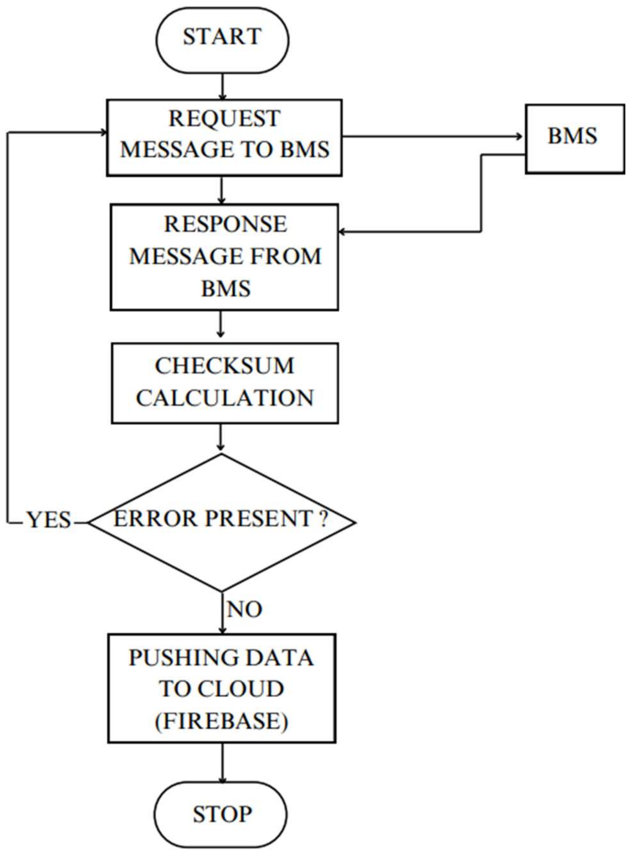


Fig 4.8 Flow Diagram depicting data acquisition procedure for each data message

CHAPTER 5
DATABASE

5.1 Google Firebase

Google firebase was opted to host the real-time database. Google Firebase's Realtime Database offers a powerful advantage in its seamless synchronization and real-time updates across devices and platforms. It provides developers with a scalable and flexible NoSQL database hosted in the cloud, eliminating the need for manual synchronization and ensuring that all connected clients receive immediate updates. This real-time capability facilitates dynamic and collaborative applications such as chat apps, collaborative editing platforms, and multiplayer games. Additionally, Firebase's robust security rules and offline support further enhance its reliability, making it a preferred choice for developers aiming to build responsive and engaging real-time applications with minimal effort and maximum scalability.

5.2 Pyrebase: To access firebase from Raspberry pi

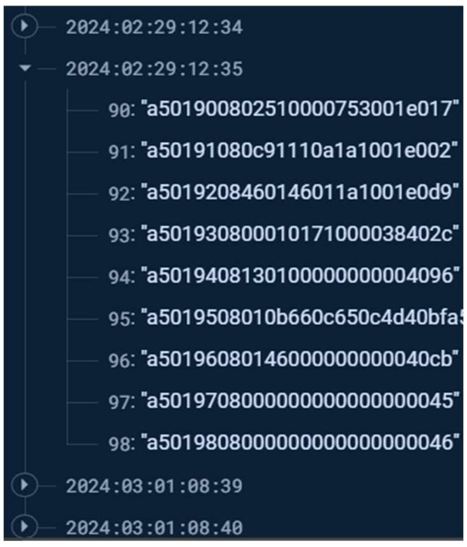
Pyrebase is a valuable library for Python developers seeking to leverage the capabilities of Firebase within their projects. It effectively serves as a wrapper for the Firebase API, providing a user-friendly interface that simplifies interaction and streamlines development workflows.

By offering functionalities for managing user authentication, database operations, and file storage, Pyrebase empowers developers to seamlessly integrate these essential Firebase services into their Python applications. This not only reduces the complexity of directly interacting with the underlying API but also enhances development efficiency by providing a concise and intuitive set of tools.

It is important to acknowledge, however, that while Pyrebase offers significant advantages, potential limitations exist. These include the possibility of limited official documentation and concerns regarding long-term maintenance of the library. Considering these factors alongside the specific needs of your project is crucial when determining the suitability of Pyrebase for your development endeavors.

5.3 Data format in Realtime Database

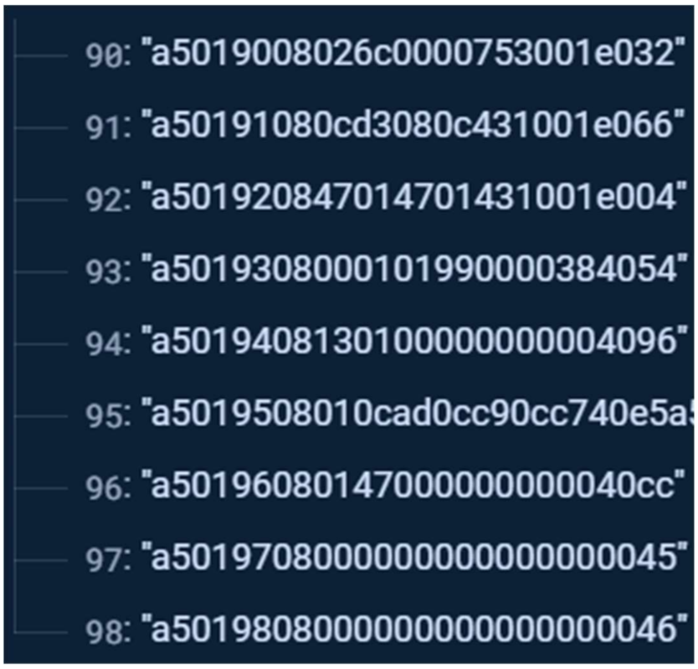
The data acquired by the raspberry pi is sent to store in such a Realtime database in either of the two formats–Historical or Overwritten. In the historical format, data is listed under the timestamp when it was read by the pi, since the sampling rate is 1 sample/sec, the timestamp is as shown in Fig 5.1.



```
2024:02:29:12:34
2024:02:29:12:35
    90: "a50190080251000075300 1e017"
    91: "a50191080c91110a1a1001e002"
    92: "a5019208460146011a1001e0d9"
    93: "a501930800010171000038402c"
    94: "a501940813010000000004096"
    95: "a5019508010b660c650c4d40bfa
    96: "a50196080146000000000040cb"
    97: "a50197080000000000000000045"
    98: "a50198080000000000000000046"
2024:03:01:08:39
2024:03:01:08:40
```

Fig 5.1: Historical data format in firebase Realtime database.

In overwritten format, the new data overwrites the previously recorded data. This format also does not have any timestamping as the only data that is present in the database is from the previous sample. The screenshot from the said database is shown in the Fig 5.2.



```
90: "a5019008026c0000753001e032"
91: "a50191080cd3080c431001e066"
92: "a50192084701470143100 1e004"
93: "a501930800010199000038405 4"
94: "a5019408130100000000004096"
95: "a5019508010cad0cc90cc740e5a
96: "a50196080147000000000040cc"
97: "a501970800000000000000000045"
98: "a501980800000000000000000046"
```

Fig. 5.2: Overwriting data format in firebase Realtime database.

CHAPTER 6
USER MANUAL

6.1 Setting up Data Acquisition station

As given in Table 4.1, Raspberry Pi 3B+ must be used to setup the data acquisition from BMS.

1. To install OS and BOOT up raspberry pi follow the instruction given in the following link: https://www.raspberrypi.com/documentation/computers/getting-started.html

2. After the initial boot up of raspberry pi, a RealVNC viewer can be used to connect raspberry pi to PC via internet. To set this up follow the instruction given in the following link: https://www.pitunnel.com/doc/access-vnc-remote-desktop-raspberry-pi-over-internet

3. Now, the necessary hardware connection must be done either following the same connection given in Fig 2.1 and Fig 2.2 or anyone apt for Daly Lifepo4 BMS 19S 60V 30A.

4. To start the data acquisition, ensure that BMS is not in sleep state, if it is found to be in sleep state, charge the setup slightly above the rated voltage, i.e., 61V to 63V.

   After ensuring the BMS is wake, download the code files from the links given below.

Code for Historical data Acquisition:

https://drive.google.com/file/d/14EAsn9RUeE5Tanlvp_7-_wF33xHzvPrM/view?usp=sharing

Code for Overwriting Data Acquisition:

https://drive.google.com/file/d/1MbvQG_6utY5WIiHHEhzQ8cm78o36m8Fq/view?usp=sharing

5. To change the database config, edit the “firebaseConfig”. This information about the database can be taken from the Realtime database which will be discussed in the next section.

6. After any changes, save the text file as .py file in the raspberry pi. Then run the program to start uploading the acquired data.

6.2 To setup firebase Realtime database

Follow the instruction given in the link to start a Realtime database in google firebase:

https://firebase.flutter.dev/docs/database/start/

To get the database url and other important information of the Realtime Firebase database, follow the instruction given in the link below:

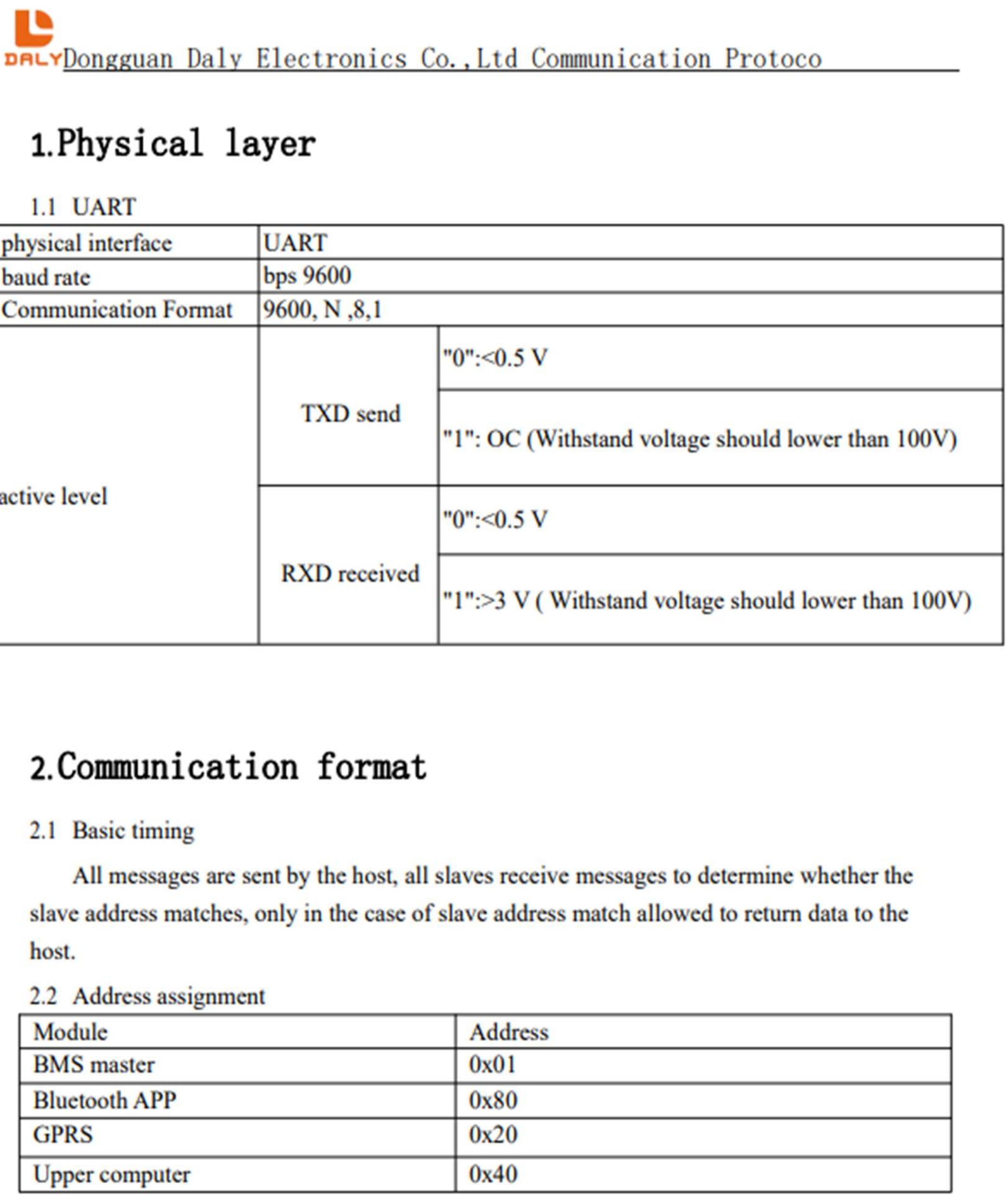
https://www.appypie.com/faqs/how-can-i-get-api-key-auth-domain-database-url-and-storage-bucket-from-my-firebase-account

Once the firebase config is obtained paste it in the program and run it. Thus, the station has been set and started to run.

CHAPTER 7

APPENDIX

COMMUNICATION DATASHEET OF DALY'S SMART BMS LIFEPO4 19S 60V 30A



Dongguan Daly Electronics Co.,Ltd Communication Protoco

1.Physical layer

1.1 UART

1. physical interface	UART	
2. baud rate	bps 9600	
3. Communication Format	9600, N ,8,1	
4. active level	TXD send	"0":<0.5 V
		"1": OC (Withstand voltage should lower than 100V)
	RXD received	"0":<0.5 V
		"1":>3 V (Withstand voltage should lower than 100V)

2.Communication format

2.1 Basic timing

All messages are sent by the host, all slaves receive messages to determine whether the slave address matches, only in the case of slave address match allowed to return data to the host.

2.2 Address assignment

Module	Address
BMS master	0x01
Bluetooth APP	0x80
GPRS	0x20
Upper computer	0x40

2.3 UART Communication Format

2.3.1 PC send

Start Flag	PC address	Data ID	Data length	Data Content	Checksum (1 Byte)
0xA5(Fixed)	0x40(UPPER-ADD)	Refer to Section3	8 Bytes(fixed)		

2.3.2 The slave responds to the host command

Start Flag	PC address	Data ID	Data length	Data Content	Checksum (1 Byte)
0xA5(Fixed)	0x01(UPPER-ADD)	Refer to Section3	8 Bytes(fixed)		

Note:

1. For each data,there is a fixed data length,can not read two data at a time.
2. The test is the sum of all previous data(only low byte).

3.Communications content information

Data Message	Data ID	UPPER-BMS	Note Remark
SOC of total voltage current	0x90	Send	Byte0~Byte7：Reserved
		Received	Byte0~Byte1:Cumulative total voltage (0.1 V) Byte2~Byte3:Gather total voltage (0.1 V) Byte4~Byte5:Current (30000 Offset ,0.1A) Byte6~Byte7:SOC (0.1%)
Maximum & Minimum voltage	0x91	Send	Byte0~Byte7：Reserved
		Received	Byte0~Byte1:Maximum cell voltage value (mV) Byte2:No of cell with Maximum voltage Byte3~byte4: Minimum cell voltage value (mV) Byte5:No of cell with Minimum voltage
Maximum & Minimum temperature	0x92	Send	Byte0~Byte7：Reserved
		Received	Byte0: Maximum temperature value (40 Offset ,°C) Byte1: Maximum temperature cell No Byte2: Minimum temperature value (40 Offset ,°C) Byte3: Minimum temperature cell No
Charge & discharge MOS status	0x93	Send	Byte0~Byte7：Reserved
		Received	Byte0:State (0 stationary 1 charge 2 discharge) Byte1:Charge MOS state Byte2:Discharge MOS status Byte3:BMS life (0~255 cycles) Byte4~Byte7:Remain capacity (mAH)
Status information 1	0x94	Send	Byte0~Byte7：Reserved
		Received	Byte0:No of battery string Byte1: No of Temperature Byte2: Charger status (0 disconnect 1 access) Byte3: Load status (0 disconnect 1 access) Byte4: Bit 0:DI1state Bit 1:DI2state Bit 2:DI3state Bit 3:DI4state Bit 4:DO1state Bit 5:DO2state Bit 6:DO3state Bit 7:DO4state Byte 5~Byte 7：Reserved
Cell voltage 1~48	0x95	Send	Byte0~Byte7：Reserved

		Received	The voltage of each monomer is 2 byte, according to the actual number of cell, the maximum 96 byte, is sent in 16 frames Byte0:frame number, starting from 0,0xFF invalid Byte1~byte6:Cell voltage (1 mV) Byte7: Reserved
Cell temperature 1~16	0x96	Send	Byte0~Byte7：Reserved
		Received	Each temperature accounts for 1 byte, according to the actual number of temperature send, the maximum 21 byte, send in 3 frames Byte0:frame number, starting at 0 Byte1~byte7:cell temperature(40 Offset ,℃)
Cell balance State 1~48	0x97	Send	Byte0~Byte7：Reserved
		Received	0： Closed 1： Open Bit0: Cell 1 balance state ... Bit47:Cell 48 balance state Bit48~Bit63：reserved
Battery failure status	0x98	Send	Byte0~Byte7：Reserved
		Received	0->No error 1->Error Byte 0 Bit 0: Cell volt high level 1 Bit 1: Cell volt high level 2 Bit 2: Cell volt low level 1 Bit 3: Cell volt low level 2 Bit 4: Sum volt high level 1 Bit 5: Sum volt high level 2 Bit 6: Sum volt low level 1 Bit 7: Sum volt low level 2 Byte 1 Bit 0: Chg temp high level 1 Bit 1: Chg temp high level 2 Bit 2: Chg temp low level 1 Bit 3: Chg temp low level 2 Bit 4: Dischg temp high level 1 Bit 5: Dischg temp high level 2 Bit 6: Dischg temp low level 1 Bit 7: Dischg temp low level 2 Byte 2 Bit 0: Chg overcurrent level 1 Bit 1: Chg overcurrent level 2 Bit 2: Dischg overcurrent level 1 Bit 3: Dischg overcurrent level 2 Bit 4: SOC high level 1

			Bit 5: SOC high level 2 Bit 6: SOC Low level 1 Bit 7: SOC Low level 2 Byte 3 Bit 0: Diff volt level 1 Bit 1: Diff volt level 2 Bit 2: Diff temp level 1 Bit 3: Diff temp level 2 Bit 4~Bit7:Reserved Byte 4 Bit 0: Chg MOS temp high alarm Bit 1: Dischg MOS temp high alarm Bit 2: Chg MOS temp sensor err Bit 3: Dischg MOS temp sensor err Bit 4: Chg MOS adhesion err Bit 5: Dischg MOS adhesion err Bit 6: Chg MOS open circuit err Bit 7: Discrg MOS open circuit err Byte 5 Bit 0: AFE collect chip err Bit 1: Voltage collect dropped Bit 2: Cell temp sensor err Bit 3: EEPROM err Bit 4: RTC err Bit 5: Precharge failure Bit 6: Communication failure Bit 7: Internal communication failure Byte6 Bit 0: Current module fault Bit 1: Sum voltage detect fault Bit 2: Short circuit protect fault Bit 3: Low volt forbidden chg fault Bit4-Bit7：Reserved Byte7: Fault code