Pack Manager Program System Design Document

Latest Revision: 26 March 2014

Prepared by: Naing Htet

Abstract

This document describes the design of the software system used for Pack Manager Program (PacMan Program). This would also serve to update the rest of the team and the advisors as to how the software was designed and perform its functions.

Table of Contents

Introduction

Scope

The PacMan Program has 3 main functions. They are:-

1) Poll the data from Battery Management System (BMS)
2) Relay data gathered from BMS to Central System Control and Data Acquisition (SCADA)
3) Oversee charging the battery pack using cell balancing algorithm

The PacMan Program is designed to be as self-sufficient as possible requiring little to no human input. It also considers the safety of the battery pack and display and log relevant information for the user.

The PacMan Program is run on TS 8160-4200, a single boarded computer. The computer is called the PacMan board and runs Debian Linux kernel version 2.6.36.2. The program is developed in C programming language and uses gcc compiler.

Overview

The document has been structured to first include a System Overview which shows the system setup of the PacMan board and the overall function of the system. This is followed by a section on the System Architecture – a detailed overview of the design and architecture of the software and its subsystems. A detailed description of the functions supported with state transition diagrams and data flow diagrams have also been included. The rest of the sections involve more information on the API and the interface.

Reference Materials

Please refer to the following documents.
TS 8160-4200 wiki page
http://wiki.embeddedarm.com/wiki/TS-8160-4200

Software Maintenance Plan
TODO

Software Maintenance Manual
TODO

Use case Design Document
TODO

Cell Balancing Algorithm Memo
TODO

System Overview

System Setup

The PacMan board is connected to the BMS of each cell of a battery pack using I2C communication. The PacMan program uses the board's I2C adapter to communicate and poll the data from BMS. The data from BMS is used for charging the battery pack and cell balancing algorithm. The PacMan can also return the data if the data is being requested by Central SCADA. The PacMan is connected to Central SCADA through serial communication (RS232). When charging the battery pack, the PacMan can open the charging relay and communicate with the charging power supply using serial communication to control to recognize when and how to charge the battery pack. For safety, the program also interacts with a watchdog and safety loop relay. More information on each connected module can be found later in the document. The connections of the PacMan are shown in the figure below.

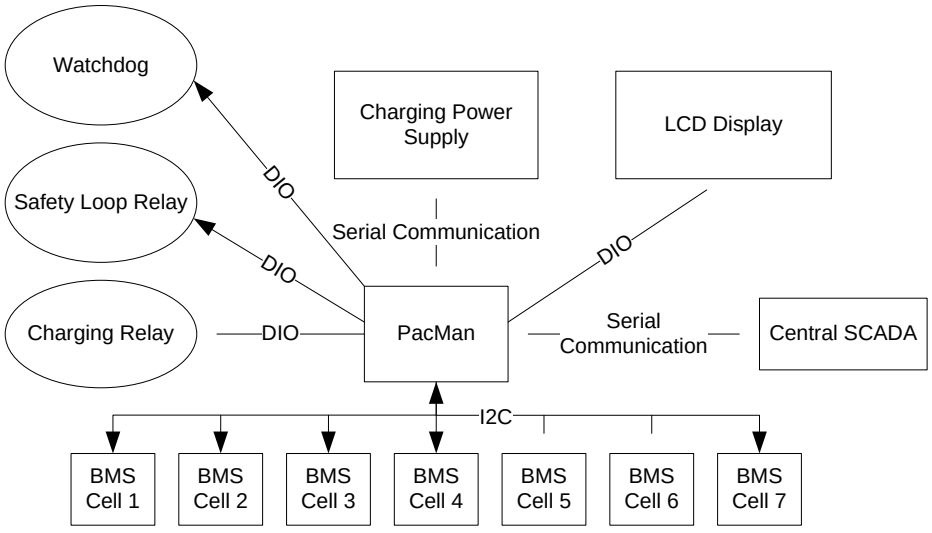

Figure 1. PacMan System Interfaces

System Functions

The use cases for the software are:-

1 Boot-up and configuration

1.1 - Auto-bootup PM program
1.2 - Auto-configure system parameters, alarms, shut down rules and safety rules from config file
1.3 - Set alarms, shut down rules and configuration parameters
1.4 - Check for safety and follow safety rules

2 Active State

2.1 - Auto-detects devices
2.2 - Poll data from BMS boards through I2C
2.3 - Relay data to the Central SCADA
2.4 - Display battery pack information via LCD display
2.5 - Log events, faults and exceptions

3. Charging State

3.1 - Enter charging state automatically
3.2 - Charge the battery pack by balancing individual cell charge levels
3.3 - Display charging information via LCD display
3.4 - Log charging information

The details of each use case can be found in Use Case Design Document listed in Reference Materials. In summary, the functions of the software are to check for the safety of the battery pack, to smart-charge the battery pack and to reply to data requests from Central SCADA. They are done with as little human input as possible and the system uses its connected modules to achieve its functions.

System Architecture

System Startup Procedure

One of the requirements of the system is for the program to start up when the power is supplied to the board. To accomplish this, I configured the system to boot directly to Debian on SD card automatically. It will then loads up the kernel and login as root. A script will be scheduled to run on startup and this script will run the PacMan Program. The program should run as a foreground process until the board is reset.

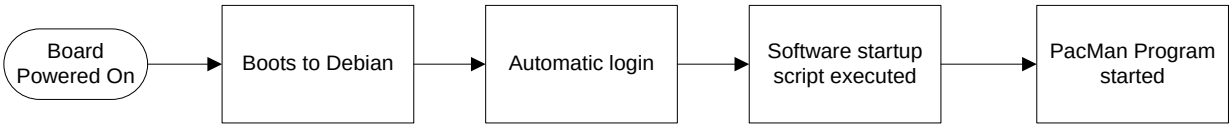

Figure 2 System Startup Flow Chart

Software Architecture Overview

The key goals of the design are to:-

1) be as self-sufficient as possible with little human interaction
2) recover automatically from errors if certain conditions are fulfilled
3) perform multiple functions as the same time.

For the above goals, the program is designed to be object-oriented and multi-threaded. This design choice allows the program to break down into different modules that are capable of working independently but yet collaborate together to achieve the goals.

There are five primary modules of the system. They are in the form of threads running concurrently. They are:-

1) Main Controller – in charge of setting up the program and creating and managing other threads
2) Safety Checker – in charge of checking if the battery pack is in safe conditions
3) Charger – in charge of charging the battery pack and the cell balancing algorithm
4) Data Relay – in charge of relaying data to the Central SCADA
5) BMS Data Poller – in charge of communicating with BMS and polling data

Two other interfaces are also available – LCD display and logger. All of the threads can use the LCD display and logger to log or display relevant information at any time.

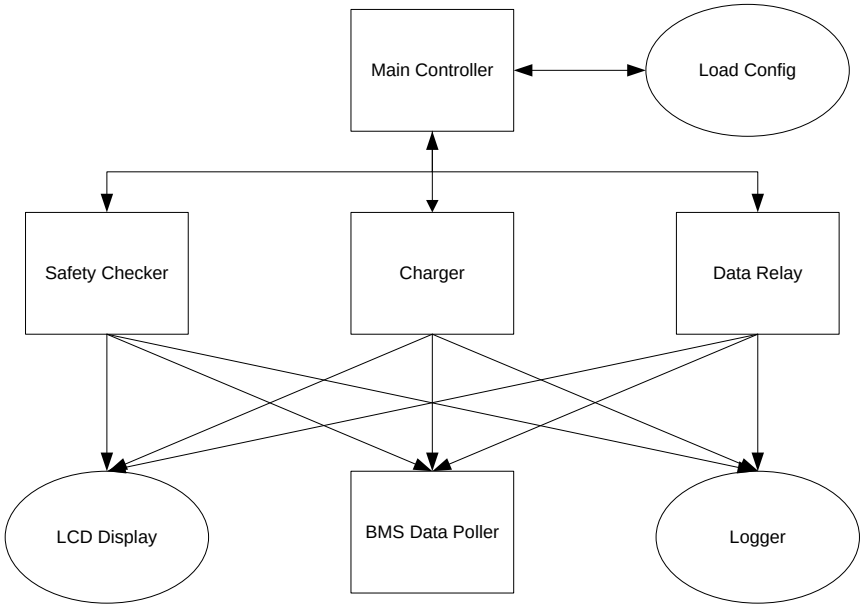

Figure 3. Software Architecture Overview

Main Controller

The purpose of the main controller is to setup the program and start all the other threads. It is the parent thread and also in charge of handling errors among the threads.

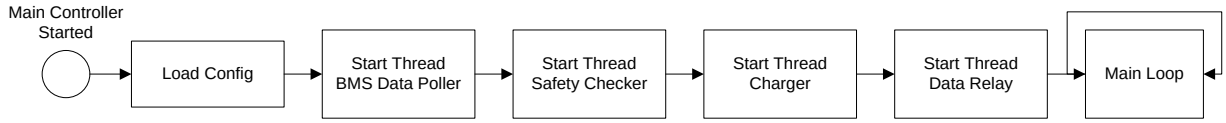

Figure 4. Main Controller Flow Chart

As soon as the program is started, the main controller is run. The first thing that the main controller should do is load the config file. If the loading was successful, it sets the parameters obtained from the config file. Using the parameters, it will start BMS data poller, safety checker, charger and data relay threads. It will then go into the main loop. Main loop handles the errors raised by other threads and quit the other threads if necessary. More information on the main loop will be available in Error Handling section.

Load Config

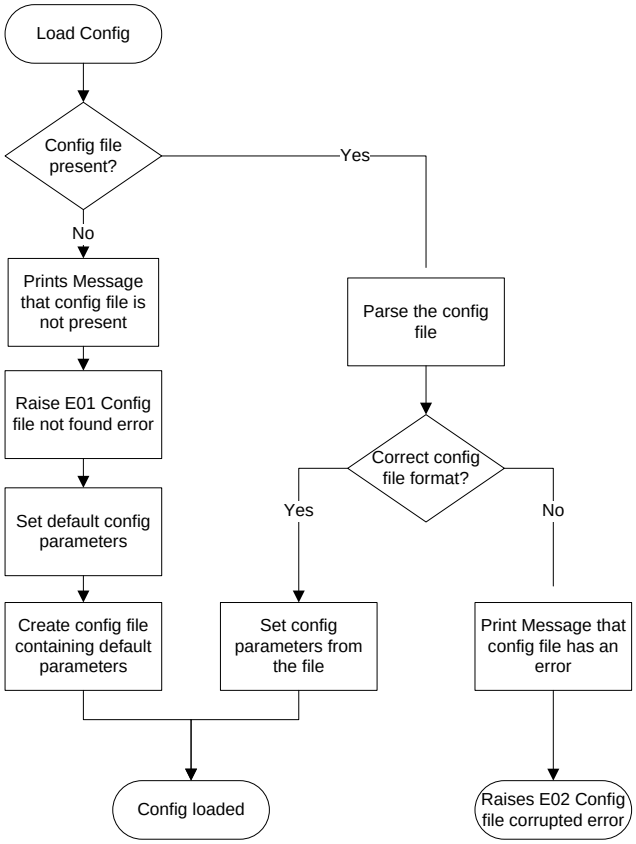

Figure 5. Load Config Flow Chart

As shown in the above flow chart, it first checks if the configuration file exists. If it doesn't, it sets to default parameters. If the file exists but is not in the correct format or syntax, it will raise an error. This error will block the program until the configuration file is fixed and the board is reset. More information on these errors can be found in Error Handling section.

It is important to note that Load Config is not a thread but a block of code executed in the main thread or the main controller. Configuration parameters are needed before the main functions of the program started. The configuration parameters include the addresses of BMS the data poller should connect to, shutdown and safety rules and other system parameters.

BMS Data Poller

The flow diagram for BMS data poller is available below. If the configuration parameters are started successfully, the main controller creates the BMS data poller thread. The first thing the data poller should do is check if all I2C devices are connected. If they are not, E03 error flag is set. This flag will pause the Safety Check thread and Charger thread as the threads cannot function if the I2C BMS devices cannot be communicated and thus, data cannot be acquired from them. After the error flag is set, the data poller will wait for the address to be reconnected by repeatedly check if the addresses are connected. Once they are connected, it will resume its main function.

The main function of the data poller is to receive data polling request from other threads and return data to them. The important thing about this function is that this is a blocking call and only one call can occur at a time. For example, when Safety Checker thread asks the BMS data poller to poll voltage values, Safety Checker will have to wait for the response from the data poller. While it is waiting for the data, if Charger thread asks the BMS data poller to poll current values, the thread will have to wait till Safety Checker is finished. This is due to the fact that in I2C communication, read requests can be overwritten. If two requests are sent one after another and the data poller tries to read the response from BMS, the response will be only for the second request. This will be achieved through the usage of mutexes.

The most common error with the BMS I2C is that devices addresses can be lost. This can happen when the cables get disconnected or one of the I2C device 's address gets changed accidentally. When this happens, this will raise the E03 error flag again. If an error other than a device address not found error is received, it will retry the request three times. If the retries fail, the data poller will raise E04 error and force the user to fix the error. More information on these errors can again be found in Error Handling section.

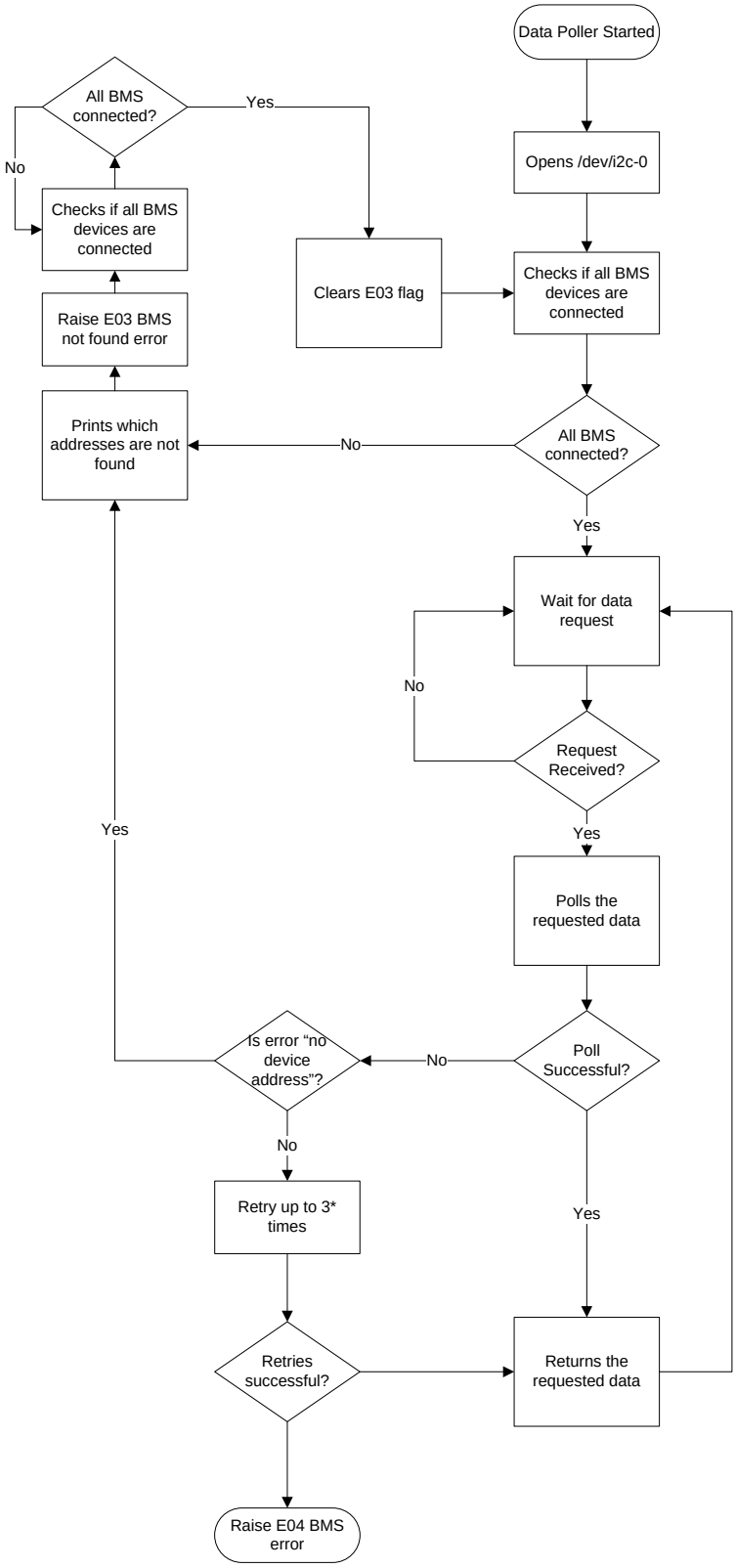

Figure 6. BMS Data Poller Flow Chart

Safety Checker

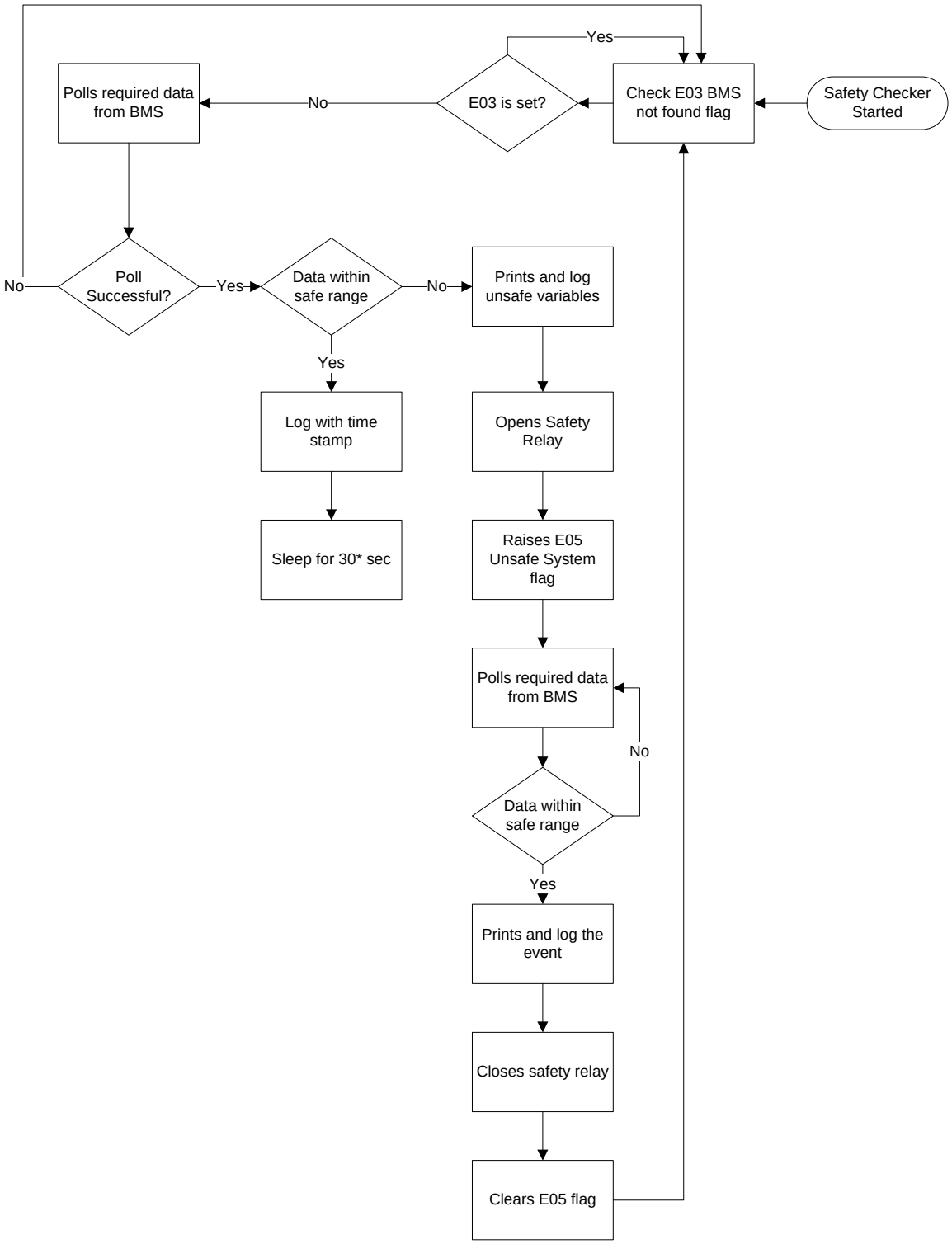

Figure 7. Safety Checker Flow Chart

As shown in the flow chart above, the main function of the Safety Checker is to repeatedly check if the system values are within safe range. The system values include voltage, current and temperature. Safe range is defined by the config file. The config file will include what the low voltage threshold and high voltage threshold are and etc. In the case that one of the value is not within predetermined safe range, it will raise E05 flag. E05 flag will only pause the Charger thread, not allowing the battery pack to be charged or stop the charging. The safety relay will be opened and then it will wait till the values are back to normal. After the values are back to normal, it will clear E05 flag and closes the safety relay and normal operations will resume.

Charger

The Charger thread is one of the most important threads. The Charger thread is what makes the smart-charging possible. Smart-charging allows the user to plug in the charger and the program will charge the battery pack when necessary and stops charging without any user interaction. The charging will initiate when the state of charge is below set threshold. The threshold will be set by the config file. When charging initiates, the error flags will be checked. If no errors exist, it will then start the Charging power supply connected to the PacMan. If the power supply doesn't respond or is not connected, it will then raise an error allowing the user to know that the battery is below threshold and the power supply is not connected. When the user connects the power supply, it will start to charge the battery pack. First, the charge relays will be closed. Then, it will go into the charging main loop. The charging main loop consists of cell balancing algorithm and safety checks. Cell balancing algorithm uses cell bypasses to make sure that the cells are charged uniformly. While it is being charged to 100%, the program will repeatedly check for safety values and see if the power supply is still connected. In the case that the power supply got disconnected or the system gets into an unsafe situation, it will try to exit the main loop. Power supply will be disabled and charge relays will be opened to get to the state the PacMan and the power supply were before the charging initiated. Then, it will wait for the issues to be resolved. If the battery pack was charged to 100% without any error, it will print and log that charging completed and wait till SOC drops below threshold again.

Detailed information of cell balancing algorithm and how State of Charge (SOC) is calculated is available in cell balancing algorithm memo found in Reference Materials.

Flow chart of the charger thread can be found below.

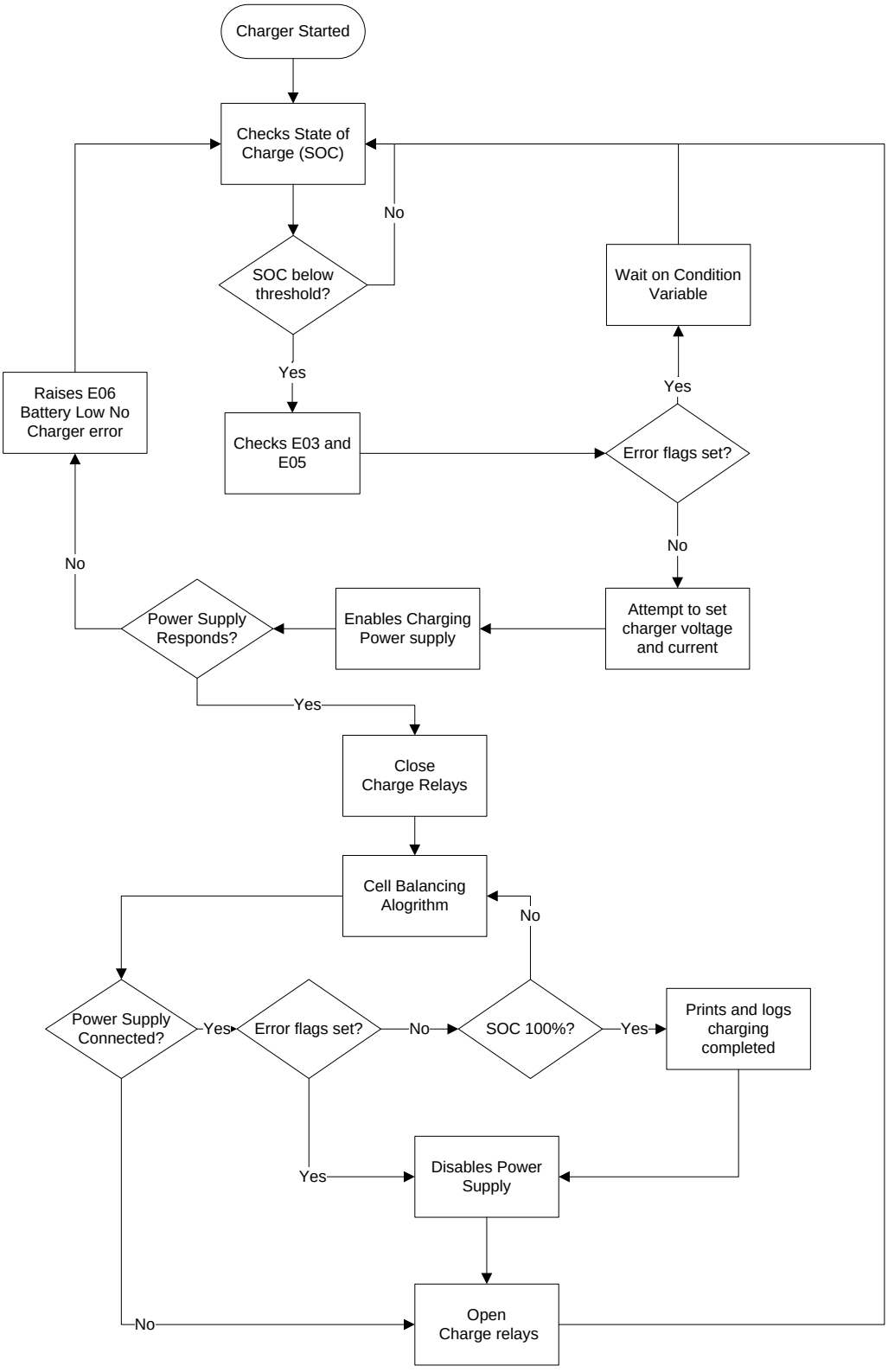

Figure 8. Charger Flow Chart

Data Relay

The purpose of the data relay thread is to provide data from BMS and the PacMan to the Central SCADA. It mainly uses I2C to communicate with the BMS board. One unique characteristic of this data relay is that it is not affected by most of the error flags. If the BMS is not available or cannot be connected, it will instead return an error message to the Central SCADA. Thus, Data Relay also provides information about the state of the PacMan program.

More information on the Data Relay Communication Protocol can be found later in this document.

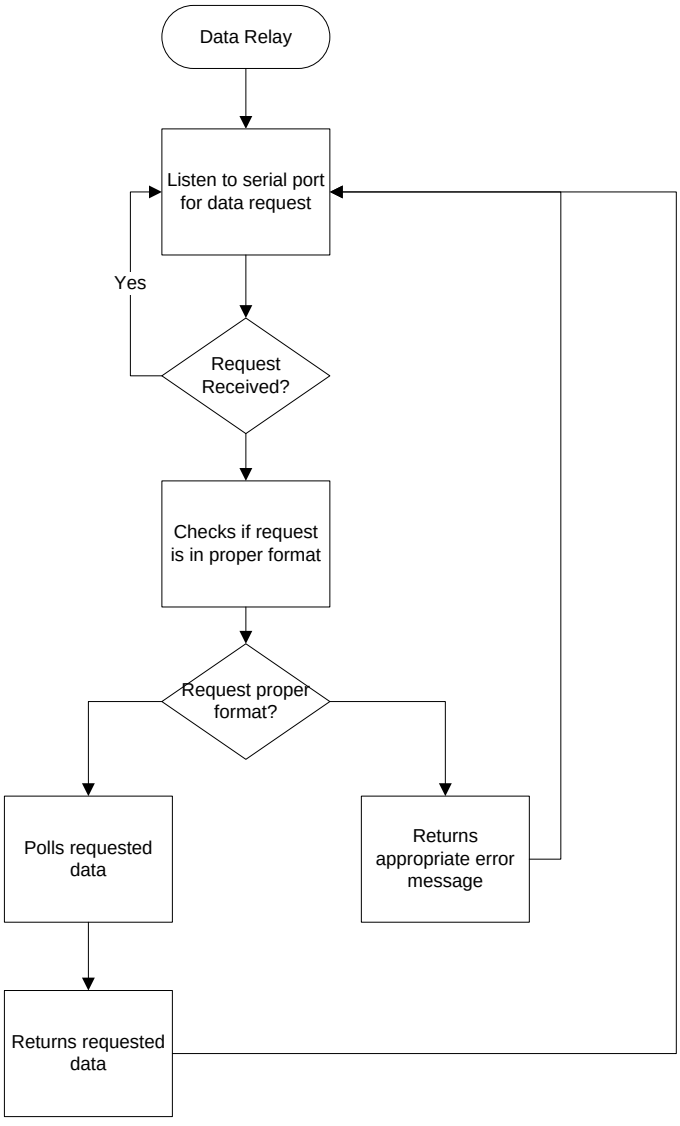

Figure 9. Data Relay Flow Chart

Error Handling

This section describes the error states the program can potentially get into.

Error Code	BMS	Safety	Charger	DataRelay
E01 : Config file not found				
E02 : Config file corrupted	✓	✓	✓	✓
E03 : BMS not found		✓	✓	
E04 : BMS error	✓	✓	✓	✓
E05 : Unsafe System			✓	
E06 : Battery Low and no charger				
E07 : Unexpected error	✓	✓	✓	✓

The above table lists the errors and the threads affected by the errors. For example, E05 only pauses the Charger thread. There are three errors that affect the whole program and they are E02, E04 and E07. When these errors occur, the main thread stops all the threads and informs the user. The main thread also handles pausing and resuming other threads if other errors occur. Using mutexes and condition variables, the program can be made to pause the threads and resume when the errors have been handled. The flow chart for this is shown below.

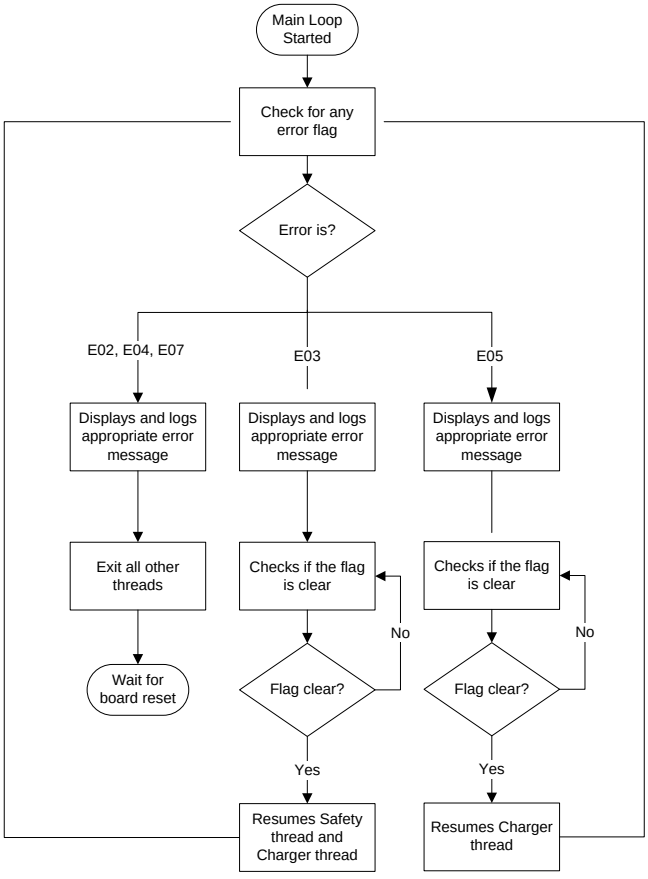

Figure 10. Main Loop Flow Chart

Below are short descriptions of the errors and how to solve them.

E01 : Config file not found
Configuration file is not found by the program and default configuration parameters will be used instead. If this is not intended, the user should make sure that the config file is in proper location.

E02 : Config file corrupted
The syntax or the config file is incorrect and the program will require a manual restart. This is because a simple mistake in configuration file can make the system unsafe. For example- setting the temperature high threshold to 100'C. The program will refuse to function until the error is fixed and the program is restarted.

E03 : BMS not found
If BMS addresses cannot be found, the user should check which BMS are missing and connect or restart them if necessary. While this error occurs, there will be no safety check and charging will not be allowed.

E04 : BMS error
This happens when the communication with BMS gives an unexpected error. When this happens, the user should check the log file to see why this error occurred. Ideally, this error should never be raised but if it were raised, this error needs user inspection and manual restart.

E05 : Unsafe system
When one of the system parameters of the battery pack is not within safe range, this error will occur. The board will attempt to solve this error by opening the safety relay and letting the values get back to normal. During this error, charging will not be allowed or interrupted.

E06 : Battery low and no charger
When the state of charge of the battery pack is low, the program will attempt to charge the pack by itself. If it cannot find the power supply or the charger, it will raise this error. This error will only stop the program from charging the battery pack without the power supply. All the other functions will resume normally. The user should connect the power supply to not let the battery pack completely run out of power.

E07 : Unexpected error
As much as I wish that the program is perfect with no error possible, we have to account for the fact that something terrible could happen and an unexpected error occurs within the program. If such error occurs, it will be logged and displayed and all the operations will cease as it is dangerous to solve an error that is not expected. If this error occurs, the user should try to solve the issue, reset the board and not curse the programmer who let the error happen.

Data Relay Communication Protocol

This describes the Communication Interface Protocol used for communication between the Pack Manager(PM) board and Central SCADA. Central SCADA connects to the PM board using serial communication.

Data format
Serial data format is 8 bit, one start bit and one stop bit with no parity bit.

End of Message
The end of message is the End Of Transmission Character (ASCII 4 or Ctrl+D in RealTerm).

Acknowledge
The transmitter of the message should receive an “OK” message from the recipient of the message. If an error is detected, the recipient will return an error message instead.

Protocol
The messages will be transmitted and received in ASCII for human readability. Central SCADA will act as the master and the PM board, the slave. This means that the PM board should only transmits message if the request from Central SCADA is addressed to it.

1) The first part of the message will be the pack number. For Central SCADA, this will be the pack number that the message is addressed to and for the PM board, this will be its own pack number.

2) The second part of the message will be the command. In an acknowledgement message, this will be either “OK” or one of the error messages.

3) The third part of the message is the argument of the command. This may be omitted if the command does not require argument. For the response message from PM board, this will be the response to the command. If there is more than one response, all the responses will be listed with ‘Spaces’ between them.

4) The parts of the message will be separated by Space characters (ASCII 32).

An example message -

CENTRAL SCADA PM BOARD

1 V? 1
(Pack number + space + command + space + argument)
This is a command to pack 1 asking for voltage of cell number 1.

1 OK
(Pack number + ACK)
This is an ACK to the command.

1 ??
(Pack number + response)
The response is the voltage of cell number 1. Please note that while the returned response is 'double', it will be displayed in ASCII and thus, not human readable.

1 OK
(Pack number + ACK)
This is an ACK by the Central SCADA to the response by PM Board.

Command List

Command	Description
V? n	Gets the cell voltage of 'n' cell. If 'n' is omitted, all cell voltages will be returned in the order of increasing cell numbers.
T? n	Gets the cell temperature of 'n' cell. If 'n' is omitted, all cell temperatures will be returned in the order of increasing cell numbers.
XT? n	Gets the temperature from external sensor 'n'. If 'n' is omitted all external sensor readings will be returned in order of increasing sensor numbers
C?	Gets the current in the discharge path of the battery pack
BPSS? n	Gets the bypass resistor switch state of 'n' cell. If 'n' is omitted, all bypass resistor switch states will be returned in the order of increasing cell numbers.
ADDR?	Gets the PM board address.

CELLCNT?	List the addresses of I2C devices connected to it.
TEST?	Returns ‘42’. (Test Command)
BPST? n	Gets the bypass time in minutes of ‘n’ cell. If ‘n’ is omitted, all times will be returned in the order of increasing cell numbers.
SAFETY?	Gets the current state of the safety loop relay on the pack manager
SOC?	Gets the current state of charge of the battery pack
Test Commands	
TESTMODE n	Turns test mode on/off. 0 - Test mode off 1 - Test mode on
TWD n	Turns the watchdog timer’s input on/off. This command is only available in test mode. If ‘n’ is omitted, WD input will be turned on by default 0 - Watchdog input off 1 - Watchdog input on
TOB n	Fakes an out-of-bounds sensor reading for test purposes. This command is only available in test mode. If ‘n’ is omitted, defaults to 0 (use normal readings) 0 - Use normal sensor readings 1 - Emulate out-of-bound sensor reading
TLVT n	Turns the low voltage threshold alarm on/off. This command is only available in test mode. If ‘n’ is omitted, defaults to 1 (Low voltage threshold on) 0 - Low Voltage threshold alarm off 1- Low Voltage threshold alarm on

Note : More commands will be added as necessary.

Error Message List

Error	Description
EBADFRMT	The format of the message is wrong or unknown. Usually happens when the message has missing spaces.
EBADCMD	The command is illegal or unknown.
EBADARG	The argument is in a bad format or missing.
ENOCELL	The specified cell is not connected or found. Checks with CELLCNT? command.
EERROR	This should not happen. This error message is returned when an unexpected error occurs within the PM board. This is the default error message if the none of the errors fits in the above categories. Checks the log file of PM board for more information.

Note: Central SCADA should always return “OK” even if the response from the PM board is different from what is expected.

Example Error Message:

CENTRAL SCADA PM BOARD

1 V? abcd

EBADARG