FIELDCRAFT DIGITAL

BMS Firmware Diagnostic Guide

STM32L476 Flash & RAM Analysis, Looptime Mapping, and Production Test Architecture

D I A G N O S E . O P T I M I Z E . D E P L O Y.

Edition v1.0 | 2026 | $19.00

Table of Contents

Chapter 1: Introduction

1.1 Purpose and Scope

This guide provides a comprehensive diagnostic reference for Battery Management System (BMS) firmware engineers working with the STM32L476RETx microcontroller. The document covers memory analysis, real-time performance profiling, production testing, and field diagnostic procedures essential for robust BMS firmware development and validation.

BMS firmware operates in safety-critical environments where failures can result in catastrophic battery pack damage, thermal runaway, or safety hazards. This guide equips engineers with systematic diagnostic approaches to identify, isolate, and resolve firmware-level issues across the complete product lifecycle — from initial development through production testing and field deployment.

Diagnostic Scope

The diagnostic procedures described in this guide address firmware-level analysis only. Hardware fault diagnosis (PCB shorts, component failures, connector issues) requires separate troubleshooting procedures not covered herein.

1.2 Target Hardware Platform

The primary target platform for this guide is the STM32L476RETx microcontroller in the LQFP64 package, based on the ARM Cortex-M4F core running at 80 MHz with a hardware FPU. The firmware reference implementation is the Marvel 3W BMS, which uses a dual-bank bootloader design supporting over-the-air (OTA) firmware upgrades and rollback between primary and secondary application images.

Parameter	Value	Diagnostic Relevance
Core	ARM Cortex-M4F @ 80 MHz	WCET baseline, looptime calculation
Flash Memory	512 KiB	Bootloader + dual app images + shared config
SRAM	96 KiB mapped (device up to 128 KiB)	Stack sizing, ThreadX objects
FPU	FPv4-SP-D16, hard ABI	MATLAB floating-point algorithm validation
CAN	1x CAN1 (PA11/PA12)	Message timing, bus loading
SPI	SPI1 (PA4-PA7)	CMU / CCM communication
I2C	I2C1 (PB6/PB7)	EEPROM and ancillary devices
GPIO	Up to 64 pins (LQFP64)	Pin mapping, signal routing

1.3 Software Stack Overview

The reference BMS firmware stack is built upon Microsoft Azure RTOS ThreadX 6.1.10 as the real-time operating system kernel, with the STM32Cube L4 HAL providing driver abstraction. Application layers implement cell voltage monitoring via ADBMS6815 CMUs, pack current/voltage measurement via the LTC2949 CCM, passive cell balancing, contactor control, SOC/SOH estimation, and CAN-based communication following UDS/ISO-TP conventions.

Layer	Component	Version	Function
RTOS	Azure RTOS ThreadX	6.1.10	Task scheduling, IPC, timing
HAL	STM32L4xx HAL Driver	CubeL4	Driver abstraction
Comm	CAN / ISO-TP	Custom	Diagnostic and normal communication
App	BMS Core	Custom	Safety, SOC, balancing, charging
Algorithms	MATLAB/Simulink Embedded Coder	Generated	Protection, state machine, SOC
Tools	OpenOCD + ST-Link	0.11.0	Flash and debug

Chapter 2: STM32L476 Flash Memory Analysis

2.1 Flash Memory Map

The STM32L476RETx provides 512 KiB of on-chip Flash memory. The Marvel 3W firmware partitions this memory into a custom bootloader, two equal application slots, and a shared common-data region used to store the active-app marker and image header. Understanding this memory map is critical for linker script configuration, bootloader implementation, and OTA update capability.

Region	Start Address	Size	Purpose
Bootloader	0x0800_0000	50 KiB	Startup, OTA dispatcher
Application Primary	0x0800_C800	200 KiB	Primary firmware image
Application Secondary	0x0804_0000	200 KiB	Secondary firmware image
Common Data	0x0807_2A60	54 KiB	Shared image header / active-app marker

2.2 Sector Breakdown and Allocation

The STM32L4 series program flash is organized into 2 KiB pages. When designing the flash allocation, page boundaries must be respected to avoid erase operations affecting adjacent regions. The bootloader region occupies the first 50 KiB of flash and contains the vector table, startup code, and partition-switching logic. The bootloader never changes during normal operation.

Each application image is 200 KiB and is checksummed using CRC-32. The bootloader verifies the image CRC on every startup before jumping to the selected partition. If verification fails, the bootloader falls back to the other partition.

Module	Code (KiB)	RO Data (KiB)	Total (KiB)	Budget %
ThreadX Kernel	~20	~2	~22	~5.7
STM32L4 HAL	~45	~9	~54	~14.1
SPI/I2C/UART Drivers	~10	~2	~12	~3.1
CAN Stack + ISO-TP	~16	~3	~19	~4.9
BMS Safety Logic	~28	~5	~33	~8.6
MATLAB Algorithms	~70	~18	~88	~22.9
Cell Balancing	~12	~3	~15	~3.9
Bootloader	~24	~4	~28	~7.3
Application + Utils	~80	~20	~100	~26.0
Reserved	--	--	~13	~3.4

Module	Code (KiB)	RO Data (KiB)	Total (KiB)	Budget %
TOTAL	~305	~66	~371	~96.6

2.3 Flash Wear Leveling

Flash memory has a limited erase/program cycle count, typically 100,000 cycles per page for the STM32L4. The EEPROM and common-data regions, which store frequently-updated parameters such as SOC counters, DTC records, and configuration data, require wear-leveling algorithms to distribute writes across multiple pages and extend overall flash lifetime.

```
/* Image header structure used by bootloader */
typedef struct __attribute__((packed))
{
    uint16_t image_magic;       /* 0x7777 (IMAGE_MAGIC) */
    uint16_t image_hdr_version; /* 1 */
    uint32_t crc;
    uint32_t data_size;
    uint8_t  image_type;
    uint8_t  version_major;
    uint8_t  version_minor;
    uint8_t  version_patch;
    uint32_t active_app;        /* 1 or 2 */
    uint32_t reserved;
    char     git_sha[8];
} image_hdr_t;
```

Design Note

At 100,000 erase cycles per page and one record write every 10 seconds, a wear-leveling scheme spanning 16 pages provides approximately 30 years of flash endurance — well beyond the typical 15-year vehicle service life.

2.4 ECC Handling

The STM32L4 flash controller includes single-error-correction, double-error-detection (SECDED) ECC hardware operating on 64-bit flash words. Single-bit errors are automatically corrected; double-bit errors should be treated as critical faults.

BMS firmware must handle ECC events as potential indicators of flash degradation or radiation-induced soft errors. The recommended handling strategy involves logging single-bit corrections and declaring a critical fault on double-bit detection, forcing the BMS to a safe state.

2.5 Flash Programming Best Practices

Reliable flash programming in a BMS context requires adherence to several key practices. Flash writes must never occur from code executing in the same flash bank being programmed. The application uses a dedicated bootloader for programming, and binaries must be patched before flashing.

Practice	Implementation	Rationale
Patch headers	Run patch_image_header.py on app binaries	Fills CRC and data_size
Atomic updates	Write header only after payload verified	Prevents corrupted records
Dual-bank OTA	Maintain two complete application images	Rollback on corrupt update
Verify after write	Bootloader computes CRC32 over image	Detects partial/corrupt flash
Boot-loop guard	Swap partition after BOOTLOOP_RETRY_COUNT	Recover from bad image

Chapter 3: RAM Analysis & Optimization

3.1 RAM Memory Map

The STM32L476RETx provides up to 128 KiB of SRAM; the Marvel 3W firmware maps 96 KiB starting at 0x2000_0000. The first 256 bytes are reserved as shared RAM for bootloader/application flags. The remainder is used for runtime data, ThreadX object stacks, and global/static variables.

Region	Size	Location	Content
Shared RAM	256 B	0x2000_0000	Boot flags, partition-switch marker
Runtime RAM	96 KiB − 256 B	0x2000_0100	ThreadX stacks, .data, .bss
Task Stacks	~28 KiB	Runtime RAM	ThreadX thread stacks
ThreadX Objects	~12 KiB	Runtime RAM	Queues, mutexes, event flags, timers
.data / .bss	~40 KiB	Runtime RAM	Global/static variables, buffers
Stack Top	--	0x2001_8000	Main stack limit symbol _estack

3.2 Stack Allocation Per Task

Proper stack sizing is critical for BMS firmware safety. An undetected stack overflow can corrupt adjacent memory regions, leading to unpredictable behavior in safety-critical tasks. ThreadX thread stacks are allocated from byte pools at startup.

Task	Priority	Stack (bytes)	Trigger	Utilization Estimate
BatteryData	P1	8192	100 ms periodic	Medium
contactorControl	P1	4096	100 ms periodic	Medium
firmware_upgrade_thread	P0	8196	100 ms periodic	Low
coulombCounting	P3	4096	100 ms periodic	Medium
chargingAlgorithm	P3	4096	100 ms periodic	Medium
canTxHandler	P4	2048	Queue event	Low-Medium
canRxHandler	P4	2048	Queue event	Low-Medium
iwdgResetTaskHandler	P3	1024	1000 ms periodic	Low

3.3 Heap Usage Monitoring

ThreadX uses byte pools for stacks and queues; the firmware avoids dynamic heap allocation after initialization. Runtime heap monitoring should track byte-pool availability and thread stack

high-water marks using ThreadX APIs such as tx_thread_stack_analyze.

```
/* ThreadX RAM usage monitor - STM32L476 */
#include "tx_api.h"
#include <stdio.h>

#define RAM_MONITOR_INTERVAL_MS     10000

typedef struct {
    ULONG heapFree;
    ULONG heapMin;
    ULONG totalStackAllocated;
    ULONG totalStackFree;
    UINT  taskCount;
} RamStats_t;

void BMS_PrintRamStats(void) {
    TX_THREAD *p;
    ULONG stack_used, stack_size;
    ULONG heapFree = tx_byte_pool_available(&byte_pool_0);

    printf("\n=== BMS RAM Usage Report ===\n");
    printf("Heap Free: %lu bytes\n\n", heapFree);

    p = _tx_thread_created_ptr;
    while(p != TX_NULL) {
        tx_thread_stack_analyze(p, &stack_used, &stack_size);
        printf(" Task: %-20s | Stack Free: %4lu bytes\n",
               p->tx_thread_name,
               stack_size - stack_used);
        p = p->tx_thread_created_next;
        if(p == _tx_thread_created_ptr) break;
    }
}
```

Implementation Note

The RAM monitor task should run at the lowest priority level to avoid interfering with safety-critical operations. Report generation via UART adds latency; for production builds, consider logging to a circular buffer in RAM for deferred retrieval via diagnostic services.

3.4 Memory Fragmentation Analysis

Memory fragmentation occurs when allocated and free blocks become interleaved, preventing the allocation of large contiguous regions. In long-running BMS firmware, fragmentation can eventually cause allocation failures. Because ThreadX byte pools use fixed-size partitions, fragmentation risk is greatly reduced compared to a generic heap.

Pool Name	Block Size	Block Count	Total (KiB)	Usage
CAN_TxQueue	sizeof(CAN message)	200	~3.2	Transmit message buffers
CAN_RxQueue	sizeof(CAN message)	200	~3.2	Receive message buffers
Thread stacks	varies	8	~28	Thread private stacks
ISO-TP buffers	4096	2	~8	OTA transfer blocks

3.5 RAM Overflow Detection and Prevention

RAM overflow detection requires both compile-time and runtime mechanisms. At compile time, the linker script defines symbols for stack top and heap boundaries. At runtime, ThreadX provides stack underflow/overflow detection and the stack high-water API.

```
/* Linker-provided symbols (from LD script) */
extern uint32_t _sstack[];
extern uint32_t _estack[];
extern uint32_t _sheap[];
extern uint32_t _eheap[];

/* ThreadX stack overflow notification */
VOID tx_thread_stack_error_handler(TX_THREAD *thread_ptr) {
    DTC_SetHardFault(DTC_STACK_OVERFLOW_DETECTED);
    BMS_EnterSafeState(SAFE_STATE_REASON_MEMORY_FAULT);
}

/* Runtime byte-pool allocation check */
void* BMS_HeapMalloc(size_t size) {
    void *p = NULL;
    UINT status = tx_byte_allocate(&byte_pool_0, &p, size, TX_NO_WAIT);
    if(status != TX_SUCCESS) {
        DTC_SetHardFault(DTC_HEAP_EXHAUSTED);
        BMS_EnterSafeState(SAFE_STATE_REASON_MEMORY_FAULT);
    }
    return p;
}
```

Chapter 4: Looptime Mapping

4.1 Real-Time Task Scheduling

The BMS firmware operates as a multi-rate real-time system where different functions execute at distinct periodicities based on their safety criticality and response requirements. ThreadX tasks are created with fixed priorities and use periodic sleep or queue waits for timing.

```
/* ThreadX task creation snippet */
tx_thread_create(&collectBatteryData, "BatteryData",
                 entry_collectBatteryData, 0,
                 pointer, sizeStack_collectBatteryData,
                 priority_P1, priority_P1,
                 TX_NO_TIME_SLICE, TX_AUTO_START);

tx_thread_create(&contactorControl, "contactorControl",
                 entry_contactorControl, 0,
                 pointer, sizeStack_contactorControl,
                 priority_P1, priority_P1,
                 TX_NO_TIME_SLICE, TX_AUTO_START);
```

4.2 Task Timing Table

Task Name	Period (ms)	Priority	Stack (bytes)	WCET (µs) est.	Deadline (ms)
BatteryData	100	P1	8192	< 5000	100
contactorControl	100	P1	4096	< 3000	100
firmware_upgrade_thread	100	P0	8196	< 2000	100
coulombCounting	100	P3	4096	< 2000	100
chargingAlgorithm	100	P3	4096	< 2000	100
canTxHandler	event	P4	2048	< 1000	10
canRxHandler	event	P4	2048	< 1000	10
iwdgResetTaskHandler	1000	P3	1024	< 500	1000

The firmware upgrade thread runs at the highest priority (P0) to ensure OTA data is processed promptly. Data collection and contactor control share P1 because both are safety-critical and run at the same 100 ms cadence. CAN handlers run at P4 (lowest) and are event-driven via queues.

4.3 CMU & CCM Sampling Timing

Cell voltages and temperatures are read from ADBMS6815 cell-monitoring units over SPI1. Pack current, pack voltage, and auxiliary measurements are read from the LTC2949 current/coulomb-counter monitor on the same SPI bus. The 100 ms BatteryData task coordinates these reads.

Signal	Source	Channels	Update Rate	Notes
Cell voltages	ADBMS6815	Up to 144 (12 CMUs x 12)	100 ms	PEC10 error checking
Cell temperatures	ADBMS6815 AUX	Up to 108 (12 x 9)	100 ms	NTC or thermistor
Pack current	LTC2949	1	100 ms	PEC15 error checking
Pack voltage	LTC2949	1	100 ms	HV divider
CCM temperature	LTC2949	1	100 ms	Monitor IC temp

4.4 CAN Message Periodicity

CAN bus communication follows a periodic transmission schedule. The CAN1 peripheral on PA11/PA12 operates at 250 kbps with hardware mailboxes and interrupt-driven reception into a 200-message receive queue.

Message	CAN ID	DLC	Period (ms)	Priority
BMS_Status	0x110	8	100	High
BMS_Batt_Param_2	0x109	8	100	High
CMU_Temperature_Info	0x112	8	100	Medium
CMU_CellVoltages_1	0x113	8	100	Medium
CMU_CellVoltages_2	0x116	8	100	Medium
Dynamic_Limits	0x12A	8	100	Medium
Charger_Request	0x107 / 0x1806E5F4	8	100	High

4.5 Worst-Case Execution Time Analysis

WCET analysis determines the maximum time a task can execute, accounting for all possible code paths, cache effects, and interrupt interference. For BMS safety certification, WCET should be measured using the ARM DWT cycle counter or ThreadX execution profiling.

```
/* WCET measurement using DWT cycle counter */
#define DWT_CYCCNT (*(volatile uint32_t *)0xE0001004)

typedef struct {
    uint32_t wcetUs;
    uint32_t bcetUs;
    uint32_t avgUs;
    uint32_t jitterUs;
    uint32_t violationCount;
} TimingProfile_t;

void BMS_MeasureWCET(TaskID_t task, uint32_t startCycles) {
    uint32_t elapsed = (DWT_CYCCNT - startCycles) / 80; /* 80 MHz */
    TimingProfile_t *t = &taskTiming[task];
    if(elapsed > t->wcetUs) t->wcetUs = elapsed;
    if(elapsed < t->bcetUs) t->bcetUs = elapsed;
    t->avgUs += (elapsed - t->avgUs) >> 4;
}
```

Chapter 5: ThreadX & Runtime Trace Setup

5.1 ThreadX Kernel Configuration

Microsoft Azure RTOS ThreadX is the real-time kernel used in the Marvel 3W firmware. The kernel is configured at compile time through tx_user.h and linked into the application. Key configuration parameters include tick rate, maximum priorities, and timer thread stack size.

Parameter	Typical Value	Purpose
TX_TIMER_TICKS_PER_SECOND	1000	1 ms tick resolution
TX_MAX_PRIORITIES	32	Priority levels available
TX_MINIMUM_STACK	512	Smallest valid thread stack
TX_TIMER_THREAD_STACK_SIZE	1024	Timer thread stack

5.2 TraceX Buffer Setup

TraceX is the Microsoft tool for visualizing ThreadX execution. It records events to a circular buffer in target RAM that can be downloaded via debug interface. A buffer size of 8-16 KB is recommended for BMS profiling.

```
/* TraceX configuration for STM32L476 */
#define TX_TRACE_BUFFER_SIZE        16384

UCHAR trace_buffer[TX_TRACE_BUFFER_SIZE];

void BMS_TraceX_Init(void) {
    tx_trace_enable(trace_buffer, TX_TRACE_BUFFER_SIZE, 32);
}
```

5.3 Event Logging

TraceX captures context switches, mutex/semaphore operations, queue access, interrupt entries/exits, and user-defined events. User events can mark specific BMS operations for timing analysis.

```
/* User event IDs for BMS-specific logging */
#define TRACE_EVT_CMU_START       100
#define TRACE_EVT_CMU_COMPLETE    101
#define TRACE_EVT_CAN_TX          102
#define TRACE_EVT_CAN_RX          103
#define TRACE_EVT_BALANCE_START   104
#define TRACE_EVT_FAULT_SET       105
#define TRACE_EVT_CONTACTOR_CMD   106

/* Usage example in data collection task */
BMS_TRACE_EVENT(TRACE_EVT_CMU_START);
adBms6815_read_cell_voltages(...);
BMS_TRACE_EVENT(TRACE_EVT_CMU_COMPLETE);
```

5.4 Timeline Analysis

The TraceX timeline view displays all recorded events on a horizontal time axis, enabling visual identification of timing anomalies, priority inversions, and excessive interrupt latency.

Metric	Target	Analysis Method
Task period jitter	< 5% of period	Measure delta between successive starts
Context switch time	< 20 µs	Time between thread exit and next entry
Interrupt latency	< 2 µs	Time from signal to ISR entry
ISR duration	< 50 µs	Total time in interrupt handler
CPU load	< 15%	Idle thread run-time percentage

5.5 Profiling a BMS Session

Follow this procedure to capture and analyze a representative BMS firmware session: build with trace enabled, initialize TraceX after kernel entry, start the host application, exercise all code paths, and save the recording for offline analysis.

Performance Impact

TraceX adds a small per-event overhead and consumes RAM for the event buffer. At typical event rates the CPU overhead is less than 1%. Disable trace in production builds to save RAM and CPU cycles.

Chapter 6: MIL vs PIL Testing Architecture

6.1 Model-in-the-Loop Overview

Model-in-the-Loop (MIL) testing validates the BMS control algorithm in a pure simulation environment before any hardware or embedded code exists. The Marvel 3W firmware uses MATLAB/Simulink Embedded Coder to generate core algorithms for protection, contactor logic, cell balancing, SOC estimation, charging, and the high-level state machine.

Module	Generated Files	Purpose
CellBalancing	CellBalancing.c, WriteMosfetsData.c	Passive cell balancing
Contactors	Contactors.c, Contactors_data.c	Contactor state machine
CoulombCounting	SOCEstimation.c, SOC_ReadFromFlash.c, ...	SOC/SOH estimation
DataPipeline	DataPipeline.c, DynamicCurrentLimits_perParallelCell.c	Sensor processing, i^2t
Protection	Protection.c	OV/UV/OCC/OCD/OT/UT logic
ChargingAlgorithm	ChargingAlgorithm.c	CC/CV charging control
StateMachine	HighLevelStatemachine.c	BMS high-level state machine

Important Agent Note

The files in Library/MATLAB/ are auto-generated from Simulink models and must never be hand-edited. If an algorithm needs changing, update the Simulink model and regenerate the C code.

6.2 Processor-in-the-Loop

Processor-in-the-Loop (PIL) testing executes compiled production code on the actual STM32L476 target while the plant model continues to run on the host PC. The host and target communicate via UART or SWO, exchanging sensor inputs and actuator outputs each simulation step. PIL validates code generation correctness, fixed-point behavior, compiler optimization effects, and timing budget.

6.3 Test Vector Generation

Test vectors are generated from MIL simulation recordings and replayed during PIL and HIL testing. Each vector contains the complete state of all BMS inputs at a given timestep, along with the expected outputs computed by the golden model.

```
/* Test vector structure */
typedef struct {
    uint32_t time_ms;
    int16_t  cellVoltages_mV[MAX_CELLS];
    int16_t  packCurrent_A;
    int8_t   temperatures_C[MAX_TEMPS];
    uint8_t  expectedSOC;
    uint8_t  expectedBalance;
    uint16_t expectedFault;
} TestRecord_t;

bool BMS_RunTestVector(const TestRecord_t *rec) {
    BMS_SetCellVoltages(rec->cellVoltages_mV);
    BMS_SetCurrent(rec->packCurrent_A);
    BMS_SetTemperatures(rec->temperatures_C);
    BMS_RunCycle(rec->time_ms);
    return (BMS_GetSOC() == rec->expectedSOC) &&
           (BMS_GetFaults() == rec->expectedFault);
}
```

6.4 Automated Test Reporting

The repository includes a Unity-based unit-test harness under Tests/. Integrate unit tests, static analysis, and PIL regression into CI/CD. When adding new test files, update Tests/CMakeLists.txt.

Chapter 7: Diagnostic Trouble Codes

7.1 UDS DTC Format

Diagnostic Trouble Codes in automotive applications follow ISO 15031-6, encoded as 3-byte values. The BMS firmware uses a project-wide bmsStatus_t enum and peripheral-level error codes for runtime fault reporting.

7.2 BMS-Specific DTC Definitions

The firmware defines system-level status returns in bmsConfiguration.h and peripheral error codes in bmsErrorCodes.h. Protection flags are generated by the MATLAB protection algorithm and include over-voltage, under-voltage, over-current, over-temperature, thermal runaway, and short-circuit flags.

Category / Flag	Name	Detection Condition	Safe State
OV	Over-Voltage	Any cell > CELL_MAX_VOLTAGE	Open contactors
UV	Under-Voltage	Any cell < CELL_MIN_VOLTAGE	Open contactors
OCC	Over-Current Charge	Charge current exceeds threshold	Limit/disable charge
OCD	Over-Current Discharge	Discharge current exceeds threshold	Limit/disable discharge
OTC	Over-Temp Charge	Cell > 55 °C (LFP)	Limit/disable charge
UTD	Under-Temp Discharge	Cell < 0 °C (LFP)	Limit/disable discharge
SC	Short Circuit	LTC2949 short-circuit detect	Open contactors
ThermalRunaway	Thermal Runaway	Rapid temp rise / gradient	Open contactors

Common status returns include BMS_OK (0), BMS_ERROR, BMS_BUSY, BMS_TIMEOUT, and peripheral-specific errors such as BMS_CMU_PEC10_ERROR, BMS_CCM_PEC15_ERROR, BMS_EEPROM_WRITE_FAILED, and BMS_FLASH_CRC_NOT_EQUAL.

7.3 DTC Storage Strategy

DTC records must persist across power cycles. The firmware stores SOC history, critical flags, and common data in an external I2C EEPROM (4 Kbit, address 0xA0). Wear leveling is implemented at the application level by rotating records across EEPROM sections.

```
/* EEPROM section layout */
#define SOC_SECTION              0x00
#define CRITICAL_FLAGS           0x32
#define COMMON_DATA              0x64
#define MERLYN_HANDLER_SECTION   0x96

/* DTC-style status byte */
#define DTC_STATUS_TEST_FAILED              0x01
#define DTC_STATUS_CONFIRMED                0x08
#define DTC_STATUS_WARNING_INDICATOR        0x80
```

Chapter 8: Production Test Procedures

8.1 ICT Flash Programming

In-Circuit Test (ICT) flash programming loads the initial firmware and calibration data into the STM32L476 during PCB assembly. The programming fixture uses SWD via ST-Link or the built-in UART bootloader to write the flash image before functional testing begins.

```
/* OpenOCD programming sequence (Linux) */
openocd -f ./openocd.cfg \
        -c "program Build/marvel3-bootLoader.bin exit 0x08000000"
python patch_image_header.py Build/marvel3-appPrimary.bin
python patch_image_header.py Build/marvel3-appSecondary.bin
openocd -f ./openocd.cfg \
        -c "program Build/marvel3-appPrimary.bin exit 0x0800C800"
openocd -f ./openocd.cfg \
        -c "program Build/marvel3-appSecondary.bin reset exit 0x08040000"
```

Always run patch_image_header.py before flashing application binaries; otherwise the bootloader will reject them because of invalid CRC or data_size fields.

8.2 EOL Functional Test Sequence

End-of-Line (EOL) testing validates the complete BMS functionality after assembly. The firmware includes functionalTests.c with manually triggered test modes that inject synthetic values into the data pipeline.

Step	Test	Method	Pass Criteria	Timeout (ms)
1	Power-on self test	Internal	All tests pass	500
2	Cell voltage path	Inject CMU data	Error < 5 mV	200
3	Current sensor	Inject CCM data	Error < 0.5% FS	200
4	Temperature sensor	Inject AUX data	Error < 1 °C	200
5	CAN communication	Send/receive test frames	0 errors	200
6	Contactor drive	Command on/off	Correct feedback	500
7	Fault injection	OV/UV/OT/OC tests	Detected < 100 ms	500
8	EEPROM write/verify	Write/read pattern	100% match	500

8.3 HiPot and Insulation Testing

HiPot testing applies a high voltage between the HV+ bus and chassis ground to verify insulation integrity. The BMS must survive this test without damage and report correct isolation resistance afterward. Disconnect or protect sensitive measurement inputs during HiPot.

8.4 Calibration Procedures

Production calibration compensates for component tolerances. Key parameters include cell voltage gain, current sensor offset/gain, and temperature sensor curves. Calibration coefficients are stored in EEPROM.

Parameter	Pre-Cal Accuracy	Post-Cal Accuracy	Reference
Cell voltage	±10 mV	±1 mV	6.5-digit DMM
Pack current	±2% FS	±0.3% FS	Calibrated shunt
Temperature	±3 °C	±0.5 °C	RTD probe

Chapter 9: Field Diagnostics

9.1 CAN Diagnostic Session

Field diagnostics of the BMS use CAN-based communication. The firmware receives configuration and test commands via standard CAN IDs and supports production/diagnostic modes through the CAN RX handler.

RX ID	Purpose
0x103	Flash configuration data
0x602	Vehicle state / charger type
0x105	EEPROM reset / test commands
0x106	Test command
0x6FA / 0x6FB	CCM data
0x1FE	Stark vehicle state

9.2 Live Parameter Reading

Live parameters are broadcast periodically on the CAN bus. Key signals include pack voltage, pack current, max/min cell voltage, average cell temperature, SOC, SOH, and dynamic current limits. These can be captured with any CAN analyzer or datalogger.

9.3 Fault Retrieval

Faults are indicated through the BMS status message, protection flags, and EEPROM-stored critical flags. Retrieve fault history by reading the CRITICAL_FLAGS section from EEPROM over the diagnostic interface.

9.4 Data Transfer for Log Download

For detailed event logs, the firmware can expose flash or EEPROM contents over ISO-TP on CAN. The OTA module (Library/Utility/can-tp-module-threadx/) implements multi-frame transport suitable for log download as well as firmware upload.

9.5 Remote Diagnostics

Remote diagnostics are possible by forwarding CAN frames from a vehicle gateway. Ensure diagnostic sessions requiring safety-critical writes (e.g., partition switch) are authenticated and rate-limited.

Chapter 10: Reference Tables

10.1 STM32L476 Pin Mapping

Pin	Function / Label
PA4	SPI1_NSS (software)
PA5	SPI1_SCK
PA6	SPI1_MISO
PA7	SPI1_MOSI
PA8	CAN1_EN
PA9 / PA10	USART1_TX / RX
PA11 / PA12	CAN1_RX / TX
PB6 / PB7	I2C1_SCL / SDA
PB10 / PB11	LPUART1_RX / TX
PC0-PC3	ADC1_IN1-IN4
PC6-PC9	TIM3_CH1-CH4 (PWM)
PC13	WakeUp input
PC14 / PC15	LED1 / LED2

10.2 CAN Message Matrix

Message	ID (hex)	Period	Description
BMS_Status	0x110	100 ms	Balancing limit, flags, pack current
BMS_Batt_Param_2	0x109	100 ms	SOC, SOH, BMS state, pack voltage
CMU_Temperature_Info	0x112	100 ms	CMU temperatures + capacity
CMU1_CellVoltages_1-4	0x113-0x115	100 ms	CMU1 cell voltages
CMU2_CellVoltages_1-4	0x116-0x118	100 ms	CMU2 cell voltages
Dynamic_Limits	0x12A	100 ms	Dynamic limits, firmware version
Charger_Request	0x107 / 0x1806E5F4	100 ms	Charger request

10.3 ThreadX Config Parameters

Parameter	Value / Notes
RTOS	Microsoft Azure RTOS ThreadX 6.1.10
Tick rate	1 ms (typical)
Max priorities	32

Parameter	Value / Notes
Time slicing	Disabled (TX_NO_TIME_SLICE)
IPC	Queues, mutexes, event flags, timers

10.4 MISRA-C Compliance Checklist

Item	Status / Guidance
Rule 1 — Language	C99 (gnu99) with compiler warnings enabled
Rule 8 — Types	Use fixed-width types (stdint); MATLAB uses uint32_T
Rule 13 — Pointers	Validate pointers before dereference
Rule 15 — Switch	All enum values handled in switch statements
Rule 17 — Functions	Single exit point preferred; status returns checked

10.5 Glossary

Term	Definition
ADBMS6815	Analog Devices 12-cell monitor / balancer IC
CCM	Current and Coulomb-counter Monitor (LTC2949)
CMU	Cell Monitoring Unit (ADBMS6815 + cells)
ISO-TP	ISO 15765-2 transport protocol over CAN
LTC2949	ADI high-voltage battery pack monitor / coulomb counter
OTA	Over-the-Air firmware update
PIL	Processor-in-the-Loop testing
SOC/SOH	State-of-Charge / State-of-Health
ThreadX	Azure RTOS real-time operating system
UDS	Unified Diagnostic Services (ISO 14229)