Embedded System Robustness Testing Report

Repository: inquis_gen_3_0
Analysis Date: July 17, 2025
Prepared by: GPT-4 Embedded Systems Analysis

Executive Summary

This report details a concise and targeted series of high-yield robustness tests designed explicitly to induce errors, crashes, or unexpected behaviors in the Inquis Gen 3.0 dual-MCU embedded medical device. Based on extensive analysis of the architecture, memory safety, and outstanding development tasks, the following test recommendations prioritize critical vulnerabilities, particularly memory corruption, state synchronization issues, and communication resilience.

Targeted Robustness Testing

Piston Control State Machine Tests

Priority	Test Scenario	Targeted Failure Conditions	States Impacted
CRITICAL	Abrupt Lid Removal	Lid detection interrupts during piston motion	`ACTIVE_MOTION_BACKWARD`, `ACTIVE_MOTION_FORWARDS`
HIGH	Rapid CO2 Depletion Simulation	State transitions triggered by piston position	`OUT_OF_CO2`, `CONFIRMING_STOP`
HIGH	Vacuum Timeout & Button Press	Vacuum stack timeout boundary conditions	`VACUUM_HOLDING_AT_BACK`, `AWAITING_ASPIRATION`
HIGH	Forced Invalid Position	Invalid sensor inputs to exceed position thresholds	`CONFIRMING_STOP`, `SYSTEM_ERROR`

Expected Impact: Expose synchronization faults and error recovery robustness in critical operational scenarios.

LED & Light State Machine Tests

Priority	Test Scenario	Targeted Failure Conditions	States Impacted
HIGH	Rapid Handle Connect/Disconnect	Communication synchronization during LED connection	`START_HANDLE_CONNECT` - `AWAIT_HANDLE_CONFIG`
HIGH	Repeated Clot & Clog Cycles	Rapid impedance changes causing state oscillation	`CLOT`, `CLOGGED`, `MIRROR_HANDLE`
MEDIUM	Flashing Pattern Stress Test	LED state flashing logic race conditions	`CONNECTING`, `CLOGGED`, `CLOT`

Expected Impact: Identify race conditions and synchronization vulnerabilities between LED indicators and underlying state logic.

Audio State Machine Tests

Priority	Test Scenario	Targeted Failure Conditions	States Impacted
HIGH	Continuous Clot Sound Triggers	Audio playback buffer overflow or corruption	`CLOT_SOUND_START`, `CLOT_SOUND`
MEDIUM	Bleed Path Sound Timeout Test	Oscillation at sound trigger threshold conditions	`BLEED_PATH_SOUND_START`, `BLEED_PATH_SOUND`

Expected Impact: Stress-test audio buffer management and state-driven sound playback logic under frequent state transitions.

Handle Impedance & Pressure Sampling Tests

Priority	Test Scenario	Targeted Failure Conditions	States Impacted
CRITICAL	Impedance Array Overflow	`memmove` operations causing impedance buffer overflow	`MONITOR_THRESHOLDS`, `ASPIRATION_EVAL_ANALYSIS`
HIGH	Wall Latch Condition Oscillation	Continuous threshold boundary impedance/pressure	`WALL_LATCH`, `WALL_LATCH_DELAY`
HIGH	Pressure Drop Stress Test	Rapid oscillating pressure changes	`START_ASPIRATION`, `ASPIRATION_EVAL_DATA_COLLECTION`

Expected Impact: Highlight array boundary vulnerabilities and robustness of impedance-pressure analysis logic.

Inter-MCU Communication & FIFO Buffer Tests

Priority	Test Scenario	Targeted Failure Conditions	Systems Impacted
CRITICAL	FIFO Overflow Exploit	Packet generation exceeding defined packet size	FIFO communication, CMS, Handle
HIGH	Communication Timeout Injection	Randomized communication drops	UART communication, CMS, Handle
HIGH	UART Communication Overload	Induced UART overruns with excessive traffic	UART communication, CMS, Handle

Expected Impact: Validate error handling and recovery mechanisms for critical inter-MCU communications.

Memory Safety & Resource Management Tests

Priority	Test Scenario	Targeted Failure Conditions	Modules Impacted
MEDIUM	Configuration Load Memory Exhaustion	Malformed/excessive configuration file sizes	Configuration loader, bbstr
MEDIUM	Continuous Reconfiguration Cycles	Frequent configuration changes uncovering leaks	bbstr, memory management utilities
MEDIUM	Long-Term Sample Counting Overflow	Integer overflow due to continuous sampling	Sampling system, han_sample

Expected Impact: Assess the device's stability and error management in scenarios of resource exhaustion and integer overflow.

Architectural Robustness & Abstraction Layer Tests

Priority	Test Scenario	Targeted Failure Conditions	Modules Impacted
HIGH	Direct Peripheral Manipulation	Abstraction violation through direct state variable access	`lights.c`, abstraction barriers
MEDIUM	Subsystem Build Flag Misuse	Improperly defined subsystem compilation flags	Build system, `comm.c`

Expected Impact: Identify and resolve architectural vulnerabilities that could lead to systemic stability and maintainability issues.

Recommendations for Implementation

- **Implement Critical Tests Immediately:** Prioritize tests related to FIFO buffer overflow, impedance array overflow, and abrupt lid removal due to their severe risk to device stability and patient safety.
- **Automate Tests Where Feasible:** Develop automated test harnesses to systematically apply the outlined stress tests, ensuring consistent coverage and reproducibility.

- **Monitor and Document Failure Modes:** Clearly document any observed failures, including state transitions, memory conditions, and recovery actions, to facilitate targeted remediation efforts.

Interrupt Vulnerability Testing Tasks

Critical System Stress Testing Requirements

Based on the comprehensive interrupt analysis, the following testing procedures are essential to demonstrate and fix race conditions, buffer overflows, and synchronization vulnerabilities that could interfere with medical device functionality during critical procedures.

High-Priority Interrupt Vulnerability Tests

`common/comm.c` - FIFO Buffer Race Condition Testing

Issue Type: Critical Race Condition
Status: Requires immediate stress testing
Location: `fifo.c:112-133, comm.c:292-372`

Test Scenario	Implementation	Expected Failure Mode	Priority
Rapid UART Interrupt Storm	Generate continuous UART idle interrupts during FIFO operations	FIFO pointer corruption, packet loss	CRITICAL
Concurrent FIFO Access	Trigger interrupts during main loop FIFO read operations	Memory corruption, invalid packet reads	CRITICAL
DMA Boundary Overflow	Send malformed frames with excessive `n_packets` values	Buffer overflow in interrupt context	CRITICAL

Test Implementation:

```
// Test rapid interrupt generation during FIFO operations
void test_fifo_race_condition() {
    for (int i = 0; i < 10000; i++) {
        // Main loop FIFO access
        fifo_read_get_ptr(&recv_fifo);

        // Trigger UART interrupt during FIFO operation
        HAL_UARTEx_RxEventCallback(&huart2, sizeof(test_frame));

        // Verify FIFO integrity
        assert(fifo_is_valid(&recv_fifo));
    }
}

// Test DMA buffer overflow with malformed packets
void test_dma_buffer_overflow() {
    Frame malformed_frame;
    malformed_frame.n_packets = 0xFFFF; // Excessive packet count
    malformed_frame.crc = calculate_crc(&malformed_frame);

    // Send malformed frame to trigger overflow
    memcpy(_recv_buffer, &malformed_frame, sizeof(malformed_frame));
    HAL_UARTEx_RxEventCallback(&huart2, sizeof(malformed_frame));

    // Check for memory corruption
    check_stack_canary();
}
```

`handle/inquis/han_state.c` - Handle State Machine Race Condition

Issue Type: Critical State Corruption
Status: Requires immediate testing
Location: `han_state.c:89-99, han_state.c:422-463`

Test Scenario	Implementation	Expected Failure Mode	Priority

Test Scenario	Implementation	Expected Failure Mode	Priority
State Transition Interrupt	Trigger comm_reply_callback during state transitions	Inconsistent state transmission to CMS	CRITICAL
Multi-word State Corruption	Interrupt during clot detection state updates	False clot detection or missed clots	CRITICAL
Button Press Race	Trigger communication interrupt during button handling	Missed aspirate button presses	HIGH

Test Implementation:

```
// Test state variable corruption during interrupt
void test_handle_state_race() {
    // Start state machine transition
    han_state_transition(HAN_STATE_ASPIRATION_EVAL_DATA_COLLECTION);

    // Trigger comm_reply_callback interrupt during transition
    comm_reply_callback();

    // Verify state consistency
    assert(_comm_reply_curr_state == han_state_get_current());
    assert(_comm_reply_imp_state_val == han_imp_state_get_current());
}

// Test clot detection state corruption
void test_clot_detection_race() {
    // Simulate clot detection in progress
    han_state_set_clot_detected(true);

    // Trigger interrupt during clot state update
    comm_reply_callback();

    // Verify clot state transmitted correctly
    assert(_comm_reply_last_aspiration_with_new_clot_seen_ms > 0);
}
```

Interrupt Priority Inversion Testing

Issue Type: Critical Timing Violation
Status: Requires priority testing
Location: `stm32l4xx_hal_msp.c:276-278`

Test Scenario	Implementation	Expected Failure Mode	Priority
Communication Interrupt Blocking	Generate continuous UART interrupts	Timer interrupts blocked, timing violations	HIGH
State Machine Timing Violations	Block critical interrupts during state transitions	Missed state transition deadlines	HIGH
Button Response Degradation	High interrupt load during button polling	Missed button presses, poor responsiveness	MEDIUM

Test Implementation:

```
// Test interrupt priority inversion
void test_interrupt_priority_inversion() {
    uint32_t start_time = HAL_GetTick();

    // Generate continuous UART interrupts
    while (HAL_GetTick() - start_time < 1000) {
        trigger_uart_interrupt();

        // Check if timer interrupts are being blocked
        if (timer_interrupt_count < expected_timer_count) {
            // Priority inversion detected
            assert(false);
        }
    }
}
```

Medium-Priority Interrupt Vulnerability Tests

Communication Protocol Timing Tests

Issue Type: Timing Vulnerability
Status: Requires stress testing
Location: `comm.c:358-360, comm.c:275-280`

Test Scenario	Implementation	Expected Failure Mode	Priority
Reply Timeout Under Load	Generate high interrupt load during reply timing	Communication timeouts, Handle isolation	MEDIUM
Timer Interrupt Jitter	Measure timer interrupt consistency under load	Variable reply timing, protocol violations	MEDIUM

System Integration Stress Tests

Full System Interrupt Storm Testing

Test Purpose: Verify system stability under maximum interrupt load **Implementation:**

```
void test_full_system_interrupt_storm() {
    // Enable all interrupt sources simultaneously
    enable_uart_interrupts();
    enable_timer_interrupts();
    enable_dma_interrupts();
    enable_gpio_interrupts();

    // Generate maximum interrupt load
    for (int duration = 0; duration < 60000; duration++) { // 60 seconds
        trigger_all_interrupts();

        // Monitor for system failures
        check_communication_integrity();
        check_state_machine_consistency();
        check_memory_corruption();
        check_timing_violations();

        delay_ms(1);
    }
}
```

Medical Procedure Simulation Under Stress

Test Purpose: Verify device reliability during critical medical procedures **Implementation:**

```
void test_medical_procedure_under_stress() {
    // Simulate aspiration procedure
    start_aspiration_procedure();

    // Generate interrupt stress during critical phases
    while (aspiration_in_progress()) {
        // Phase 1: Clot detection
        if (in_clot_detection_phase()) {
            stress_test_impedance_interrupts();
        }

        // Phase 2: Wall latch detection
        if (in_wall_latch_phase()) {
            stress_test_pressure_interrupts();
        }

        // Phase 3: Aspiration control
        if (in_aspiration_control_phase()) {
            stress_test_piston_interrupts();
        }

        // Verify medical procedure continues correctly
        assert(procedure_still_valid());
    }
}
```

Advanced Vulnerability Testing Procedures

Configuration and Input Validation Vulnerabilities

Priority	Test Scenario	Targeted Failure Conditions	Implementation Method
CRITICAL	Malformed Configuration File Attack	Integer overflow, memory exhaustion during config parsing	Load config files with extreme values, negative numbers, oversized strings
HIGH	Configuration Structure Type Confusion	Memory corruption via type assumptions in `config.c:326`	Modify config with non-integer fields, test structure boundary violations
HIGH	Default Config Boundary Testing	Array bounds violations in config loading	Test configs with values exceeding defined ranges (pressure thresholds, timing values)
MEDIUM	SD Card Configuration Corruption	Filesystem corruption during config write/read cycles	Interrupt power during config saves, test with corrupted FAT32 structures

Test Implementation:

```
// Test extreme configuration values
void test_config_boundary_violations() {
    // Test integer overflow scenarios
    config.cms_Ls = INT_MAX;
    config.handle_K = -1;
    config.vacuum_threshold = 0xFFFFFFFF;

    // Test string fields with excessive lengths
    strcpy(config.device_name, "A_VERY_LONG_STRING_THAT_EXCEEDS_BUFFER_SIZE_...");

    // Verify system behavior with invalid configs
    test_system_startup_with_invalid_config();
}

// Test configuration memory exhaustion
void test_config_memory_exhaustion() {
    // Create config file with 10,000+ lines
    generate_massive_config_file();

    // Monitor memory usage during parsing
    monitor_heap_usage_during_config_load();

    // Verify graceful degradation vs system crash
}
```

State Machine Transition Boundary Testing

Priority	Test Scenario	Targeted Failure Conditions	FSM States Targeted
CRITICAL	Invalid State Injection	Array bounds access in state name lookups	All CMS and Handle states
CRITICAL	State Transition Race Conditions	Inconsistent state updates during interrupts	Handle states 3002, 5002-5003
HIGH	State Machine Deadlock Testing	Circular state dependencies, stuck states	Piston vacuum states 71-72
HIGH	State Overflow Attack	State enumeration values beyond array bounds	LED states, audio states
MEDIUM	State Timing Boundary Violations	Missed state transitions due to timing edge cases	All timed state transitions

Test Implementation:

```
// Test state enumeration boundary violations
void test_state_boundary_attacks() {
    // Force invalid state values into state machines
    cms_state.curr_piston_state = 999;  // Beyond valid range
    cms_state.curr_led_state = -1;      // Negative state

    // Trigger state name lookup to cause array bounds violation
    log_state_transition();  // Should crash or corrupt memory
}

// Test state machine deadlock scenarios
void test_state_machine_deadlocks() {
    // Force piston into vacuum hold state
    force_piston_state(PISTON_STATE_72_VACUUM_HOLDING_AT_BACK);

    // Block pressure sensors to prevent state exit conditions
    block_pressure_readings();

    // Verify system doesn't hang permanently
    monitor_state_transitions_for_timeout(30000); // 30 seconds
}

// Test concurrent state transitions under interrupt load
void test_state_transition_races() {
    // Start state machine transition
    begin_handle_state_transition(HAN_STATE_ASPIRATION_EVAL_ANALYSIS);

    // Generate communication interrupt during transition
    trigger_comm_reply_callback_interrupt();

    // Verify state consistency across interrupt boundary
    assert_state_consistency();
}
```

Memory Corruption and Buffer Overflow Testing

Priority Test Scenario Targeted Failure Conditions Vulnerable Code Locations CRITICAL FIFO Buffer Overflow Exploitation Memory corruption via packet size manipulation fifo.c:76, comm.c:305-343 CRITICAL Impedance Array Overflow Buffer overflow in memmove operations han_state.c:653-661 HIGH String Buffer Overflows BBStr malloc exhaustion, string copying overflows bbstr.c:77-88 HIGH Audio Buffer Stack Overflow Stack corruption via large audio buffers cms_devices.c:63-64 MEDIUM Configuration Parsing Overflows Buffer overruns during config file processing config.c parsing functions

Test Implementation:

```
// Test FIFO buffer overflow exploitation
void test_fifo_overflow_attacks() {
    // Create malformed packet with excessive n_packets value
    Frame attack_frame;
    attack_frame.n_packets = 0xFFFF;  // Exceeds buffer capacity
    attack_frame.crc = calculate_crc(&attack_frame);

    // Send via UART to trigger overflow in comm.c
    send_uart_frame(&attack_frame);

    // Check for memory corruption
    verify_stack_canary_integrity();
    verify_heap_integrity();
}

// Test impedance buffer boundary violations
void test_impedance_buffer_attacks() {
    // Force IMPEDANCE_MAX_N to be exceeded
    for (int i = 0; i < IMPEDANCE_MAX_N + 100; i++) {
        // Force sample processing to overflow buffer
        process_impedance_sample(mock_sample);
    }

    // Verify buffer integrity
    check_buffer_canaries();
}

// Test malloc exhaustion scenarios
void test_memory_exhaustion() {
    // Repeatedly trigger bbstr allocations
    for (int i = 0; i < 10000; i++) {
        BBStr *str = bbstr_new("test_string", -1);
        // Don't free to simulate leak
    }

    // Verify graceful degradation vs system halt
    test_system_functionality_under_memory_pressure();
}
```

Communication Protocol Stress Testing

Priority Test Scenario Targeted Failure Conditions Communication Paths CRITICAL UART Error Recovery Exploitation Permanent communication failure CMS-Handle UART channel CRITICAL Communication Timeout Cascade System isolation during critical procedures All inter-MCU communication HIGH DMA Buffer Boundary Attacks Memory corruption via DMA overflow DMA communication channels HIGH Packet Fragmentation Stress Protocol confusion, packet loss All packet-based communication MEDIUM Communication Timing Attacks Protocol violations via timing manipulation Reply timing, timeout handling

Test Implementation:

```
// Test communication protocol failure cascades
void test_communication_failure_cascades() {
    // Start critical aspiration procedure
    initiate_aspiration_procedure();

    // Generate continuous UART errors during procedure
    while (aspiration_in_progress()) {
        trigger_uart_overrun_error();
        trigger_uart_framing_error();
        delay_ms(50);
    }

    // Verify system maintains safety during communication failure
    assert_safe_system_state();
}

// Test DMA buffer manipulation attacks
void test_dma_buffer_attacks() {
    // Manipulate DMA buffer contents during transfer
    setup_dma_transfer();

    // Corrupt buffer during DMA operation
    corrupt_dma_buffer_during_transfer();

    // Verify error detection and recovery
    verify_dma_error_handling();
}

// Test communication timing boundary conditions
void test_communication_timing_attacks() {
    // Send packets at exact timeout boundaries
    send_packet_at_timeout_minus_1ms();
    send_packet_at_timeout_plus_1ms();

    // Verify timeout handling robustness
    verify_timeout_recovery_behavior();
}
```

Real-World Misuse Scenario Testing

Priority Test Scenario Targeted Failure Conditions User Misuse Patterns CRITICAL Rapid Button Mashing During Procedures Button debouncing failures, state confusion Aspiration button during clot detection HIGH Power Cycling During Critical States Incomplete state persistence, corruption Power loss during aspiration HIGH Catheter Manipulation During Operation Sensor confusion, impedance range violations Physical sensor manipulation MEDIUM Environmental Stress Conditions Temperature/humidity impact on timing Extended operation in extreme conditions MEDIUM Simultaneous Multiple Operations Resource contention, timing violations Multiple rapid procedure attempts

Test Implementation:

```
// Test rapid user input scenarios
void test_rapid_button_attacks() {
    // Generate button presses faster than debouncing logic
    for (int i = 0; i < 1000; i++) {
        trigger_aspirate_button_press();
        delay_microseconds(100);  // Faster than expected human input
        trigger_aspirate_button_release();
        delay_microseconds(100);
    }

    // Verify button state consistency
    verify_button_state_machine_integrity();
}

// Test power cycling stress scenarios
void test_power_cycling_stress() {
    // Cycle power during each critical state
    for (each_critical_state) {
        enter_critical_state();
        power_cycle_device();
        verify_recovery_behavior();
        check_state_persistence();
    }
}

// Test sensor manipulation scenarios
void test_sensor_manipulation_attacks() {
    // Rapidly change impedance values outside normal ranges
    simulate_impedance_values(0);           // Short circuit
    simulate_impedance_values(1000000);     // Open circuit
    simulate_impedance_values(-1);          // Invalid negative

    // Test pressure sensor manipulation
    simulate_pressure_values(-100);         // Invalid negative pressure
    simulate_pressure_values(10000);        // Excessive pressure

    // Verify sensor range validation
    verify_sensor_input_validation();
}
```

Timing and Synchronization Stress Testing

Priority Test Scenario Targeted Failure Conditions Timing Dependencies CRITICAL Interrupt Priority Inversion Critical timing violations UART vs Timer interrupts CRITICAL Watchdog Timer Bypass System hang detection failure All long-running operations HIGH State Machine Timing Violations Missed deadlines, stuck states All timed state transitions HIGH Communication Timeout Edge Cases Premature/delayed timeout handling Tcom=1000ms boundaries MEDIUM Real-Time Constraint Violations Missed impedance sampling, audio glitches Real-time sampling loops

Test Implementation:

```
// Test interrupt priority inversion scenarios
void test_interrupt_priority_attacks() {
    // Generate continuous low-priority interrupts
    while (test_duration < 60000) {  // 60 seconds
        trigger_continuous_uart_interrupts();

        // Monitor for blocked timer interrupts
        if (timer_interrupt_missed()) {
            log_priority_inversion_detected();
        }
    }
}

// Test watchdog bypass scenarios
void test_watchdog_bypass_attacks() {
    // Create scenarios that could hang the system
    enter_infinite_loop_in_state_machine();
    block_all_interrupts_permanently();
    create_memory_allocation_deadlock();

    // Verify watchdog triggers system reset
    verify_watchdog_reset_within_timeout();
}

// Test timing constraint boundary conditions
void test_timing_constraint_violations() {
    // Test each timing parameter at boundary conditions
    test_timing_parameter(Tcom, 999);   // Just under timeout
    test_timing_parameter(Tcom, 1001);  // Just over timeout
    test_timing_parameter(TB1, 0);      // Minimum value
    test_timing_parameter(TB1, 10000);  // Excessive value

    // Verify system behavior at timing boundaries
    verify_timing_behavior_correctness();
}
```

Integration Stress Testing Scenarios

Priority Test Scenario Targeted Failure Conditions System Integration Points CRITICAL Full System State Explosion Exponential state combinations All state machines simultaneously HIGH Resource Exhaustion Under Load Memory, CPU, I/O resource depletion All system resources HIGH Error Propagation Cascades Single failures causing system-wide failures Cross-module error handling MEDIUM Performance Degradation Testing System slowdown under stress All performance-critical paths MEDIUM Concurrent Operation Stress Multiple simultaneous operations All parallel subsystems

Test Implementation:

```
// Test exponential state combination explosion
void test_state_explosion_scenarios() {
    // Force all state machines into complex state combinations
    force_piston_state(PISTON_STATE_72_VACUUM_HOLDING_AT_BACK);
    force_led_state(LED_STATE_502_CLOT);
    force_audio_state(AUDIO_STATE_30002_CLOT_SOUND);
    force_handle_state(HAN_STATE_6001_WALL_LATCH);

    // Generate state transitions in all machines simultaneously
    trigger_all_state_machines_simultaneously();

    // Monitor for unexpected state combinations
    verify_state_combination_validity();
}

// Test system resource exhaustion
void test_resource_exhaustion_scenarios() {
    // Exhaust memory resources
    allocate_all_available_memory();

    // Exhaust CPU resources
    create_cpu_intensive_interrupt_storm();

    // Exhaust I/O resources
    flood_all_communication_channels();

    // Verify graceful degradation
    verify_system_maintains_core_functionality();
}

// Test cascading failure scenarios
void test_cascading_failure_scenarios() {
    // Introduce single point failures
    simulate_sd_card_failure();
    simulate_pressure_sensor_failure();
    simulate_impedance_sensor_failure();

    // Monitor for cascade effects
    monitor_cross_system_error_propagation();

    // Verify isolation of failures
    verify_failure_containment();
}
```

Long-Duration Stress Testing

Priority Test Scenario Targeted Failure Conditions Duration Requirements CRITICAL 24+ Hour Continuous Operation Memory leaks, integer overflows, cumulative errors 24-72 hours minimum HIGH Thermal Stress Extended Operation Temperature-induced timing drift, component failure 12+ hours at temperature extremes HIGH Power Supply Stress Testing Brown-out conditions, voltage fluctuations Extended operation with power stress MEDIUM Repeated Procedure Cycling Wear-out failures, cumulative state corruption 10,000+ procedure cycles MEDIUM Data Logging Stress Testing SD card wear, filesystem corruption Continuous logging for days

Test Implementation:

```
// Test extended operation scenarios
void test_extended_operation_stress() {
    uint32_t test_start_time = get_time_ms();
    uint32_t procedure_count = 0;

    while ((get_time_ms() - test_start_time) < (24 * 60 * 60 * 1000)) {  // 24 hours
        // Perform complete aspiration procedure
        perform_complete_aspiration_cycle();
        procedure_count++;

        // Monitor for degradation indicators
        check_memory_usage();
        check_timing_drift();
        check_communication_integrity();
        check_state_machine_consistency();

        // Log any anomalies
        if (anomaly_detected()) {
            log_anomaly_with_timestamp_and_count(procedure_count);
        }
    }

    // Generate comprehensive stress test report
    generate_extended_operation_report();
}

// Test thermal stress scenarios
void test_thermal_stress_scenarios() {
    // Test at temperature extremes
    set_environmental_temperature(70);  // High temperature
    run_stress_test_suite();

    set_environmental_temperature(-10); // Low temperature
    run_stress_test_suite();

    // Test thermal cycling
    for (int cycle = 0; cycle < 100; cycle++) {
        thermal_cycle_device();
        verify_system_functionality();
    }
}
```

Unexpected Input and Error Condition Testing

SD Card and File System Corruption Testing

Priority	Test Scenario	Targeted Failure Conditions	System Impact
CRITICAL	SD Card Removal During Logging	Incomplete log writes, file system corruption	CMS logging system, data integrity
CRITICAL	Corrupted FAT32 File System	File system mount failures, data loss	SD card initialization, log recovery
HIGH	SD Card Write Protection Active	Failed log operations, error handling	Log system error recovery
HIGH	Insufficient SD Card Space	Write failures during critical operations	Graceful degradation testing
MEDIUM	SD Card Physical Corruption	Bad sector handling, data recovery	File system resilience
MEDIUM	Power Loss During SD Write	Incomplete file operations, journal recovery	Data consistency validation

Test Implementation:

```
// Test SD card removal during active logging
void test_sd_card_removal_during_logging() {
    // Start continuous logging operation
    start_continuous_logging();

    // Simulate SD card removal mid-write
    simulate_sd_card_removal();

    // Verify system behavior
    verify_log_error_handling();
    verify_no_system_crash();

    // Test recovery when card reinserted
    simulate_sd_card_insertion();
    verify_logging_recovery();
}

// Test corrupted file system handling
void test_corrupted_filesystem() {
    // Create corrupted FAT32 structure
    corrupt_fat32_boot_sector();
    corrupt_fat32_allocation_table();

    // Attempt system startup
    result = initialize_sd_card_system();

    // Verify graceful failure handling
    assert(result == SD_INIT_FAILED);
    verify_system_continues_without_logging();
}

// Test insufficient storage scenarios
void test_sd_storage_exhaustion() {
    // Fill SD card to near capacity
    fill_sd_card_to_threshold();

    // Generate continuous log data
    while (log_space_available()) {
        generate_log_entry();
    }

    // Verify behavior when storage exhausted
    verify_storage_exhaustion_handling();
    verify_log_rotation_or_cleanup();
}
```

Corrupted Configuration File Testing

Priority	Test Scenario	Targeted Failure Conditions	Configuration Parameters
CRITICAL	Malformed `default_config.txt`	Config parsing failures, invalid parameters	All system thresholds and timing
CRITICAL	Config File Truncation	Incomplete parameter loading	Critical pressure/impedance thresholds
HIGH	Invalid Parameter Ranges	Out-of-bounds values causing system instability	K, L, Ls pressure thresholds
HIGH	Config File with Binary Data	Text parsing corruption, buffer overflows	Config file integrity
MEDIUM	Missing Configuration File	Default value fallback testing	System startup with defaults
MEDIUM	Config File Permission Errors	Read access failures during startup	Error recovery mechanisms

Test Implementation:

```
// Test malformed configuration file handling
void test_malformed_config_file() {
    // Create config with invalid syntax
    create_config_file_with_syntax_errors();

    // Test invalid parameter formats
    add_config_line("cms_Ls = INVALID_NUMBER");
    add_config_line("handle_K = 999999999999999999999"); // Overflow
    add_config_line("Tcom = -500"); // Invalid negative timing

    // Attempt system initialization
    result = load_system_configuration();

    // Verify error handling
    verify_config_load_error_handling();
    verify_fallback_to_defaults();
}

// Test configuration boundary violations
void test_config_boundary_violations() {
    // Test extreme pressure threshold values
    set_config_value("cms_Ls", "0");           // Minimum pressure
    set_config_value("handle_K", "100000");    // Excessive pressure
    set_config_value("handle_L", "-100");      // Invalid negative

    // Test timing parameter violations
    set_config_value("Tcom", "0");             // Zero timeout
    set_config_value("TB1", "999999");         // Excessive timing
    set_config_value("Trst", "-1");            // Invalid reset time

    // Verify parameter validation
    verify_config_parameter_validation();
    verify_safe_fallback_values();
}

// Test configuration file corruption scenarios
void test_config_file_corruption() {
    // Inject random binary data into config file
    inject_binary_data_into_config();

    // Truncate config file mid-parameter
    truncate_config_file_randomly();

    // Add extremely long lines
    add_config_line_with_excessive_length();

    // Verify parsing robustness
    verify_config_parsing_robustness();
}
```

Corrupted Audio File Testing

Priority Test Scenario Targeted Failure Conditions Audio System Impact HIGH Corrupted Audio Header Files WAV file parsing errors, audio playback failures Audio state machine failures HIGH Missing Audio Files File not found during audio state transitions Audio system error recovery MEDIUM Invalid Audio Format Unsupported audio format handling Audio initialization errors MEDIUM Truncated Audio Data Incomplete audio playback, buffer underruns Audio state timing issues MEDIUM Audio File Size Mismatches Memory allocation errors, buffer overflows Audio buffer management

Test Implementation:

```
// Test corrupted audio file handling
void test_corrupted_audio_files() {
    // Corrupt WAV header information
    corrupt_wav_file_header("audio0.wav");
    corrupt_wav_file_sample_rate("audio0.wav");

    // Test audio system initialization
    result = initialize_audio_system();

    // Verify error handling
    verify_audio_init_error_handling();
    verify_audio_state_machine_stability();
}

// Test missing audio file scenarios
void test_missing_audio_files() {
    // Remove critical audio files
    delete_file("audio0.wav");
    delete_file("clot_detection_sound.wav");

    // Attempt audio playback
    trigger_clot_detection_audio();

    // Verify graceful degradation
    verify_silent_operation_mode();
    verify_no_audio_state_machine_crash();
}

// Test audio buffer overflow scenarios
void test_audio_buffer_corruption() {
    // Create audio file larger than expected buffer
    create_oversized_audio_file();

    // Trigger audio playback
    result = play_audio_file();

    // Monitor for buffer overflow
    verify_audio_buffer_bounds_checking();
    verify_stack_integrity();
}
```

Hardware Component Failure Simulation

Priority Test Scenario Targeted Failure Conditions Hardware Components CRITICAL Pressure Sensor Disconnection Invalid pressure readings, sensor error detection Pressure monitoring system CRITICAL Impedance Sensor Malfunction AD5940 communication failures, invalid impedance Handle impedance detection HIGH LED Driver I2C Failures LED control errors, communication timeouts Visual feedback system HIGH UART Communication Hardware Failure Inter-MCU communication loss CMS-Handle coordination MEDIUM Power Supply Voltage Fluctuations Brown-out conditions, system instability Power management MEDIUM Button Hardware Debouncing Failures False button presses, missed inputs User interface

Test Implementation:

```
// Test pressure sensor failure scenarios
void test_pressure_sensor_failures() {
    // Simulate sensor disconnection
    simulate_pressure_sensor_disconnect();

    // Test invalid pressure readings
    inject_invalid_pressure_values(0xFFFF);    // Sensor error value
    inject_invalid_pressure_values(-1);        // Impossible negative pressure

    // Verify error detection and handling
    verify_pressure_sensor_error_detection();
    verify_safe_system_behavior_without_pressure();
}

// Test impedance sensor malfunction
void test_impedance_sensor_failures() {
    // Simulate AD5940 communication failure
    simulate_ad5940_i2c_failure();

    // Test impedance reading corruption
    inject_corrupted_impedance_readings();

    // Simulate sensor calibration errors
    corrupt_impedance_calibration_data();

    // Verify error handling
    verify_impedance_error_recovery();
    verify_clot_detection_fallback_behavior();
}

// Test LED driver I2C communication failures
void test_led_driver_failures() {
    // Simulate I2C bus errors
    simulate_i2c_bus_hang();
    simulate_i2c_ack_failures();

    // Test LED driver timeout scenarios
    block_led_driver_responses();

    // Verify LED system error handling
    verify_led_timeout_recovery();
    verify_visual_feedback_fallback();
}
```

Unexpected System State Testing

Priority Test Scenario Targeted Failure Conditions System State Impact CRITICAL Boot Sequence Interruption Incomplete initialization, partially initialized state System startup integrity CRITICAL Unexpected Reset During Operation State persistence, recovery from unknown state State machine consistency HIGH Clock/Timer Subsystem Failure Timing-dependent operations, state transition timing All timed operations HIGH Memory Allocation Failures malloc() failures during operation Dynamic memory handling MEDIUM Interrupt Controller Malfunction Missed interrupts, interrupt priority corruption Real-time system behavior MEDIUM Watchdog Timer Premature Trigger Unexpected system resets, operation interruption System stability

Test Implementation:

```
// Test boot sequence interruption scenarios
void test_boot_sequence_interruption() {
    // Interrupt boot at various stages
    interrupt_boot_during_peripheral_init();
    interrupt_boot_during_config_loading();
    interrupt_boot_during_state_machine_init();

    // Test recovery behavior
    verify_boot_recovery_mechanisms();
    verify_safe_default_state();
}

// Test unexpected reset scenarios
void test_unexpected_reset_recovery() {
    // Force reset during critical operations
    force_reset_during_aspiration();
    force_reset_during_clot_detection();
    force_reset_during_communication();

    // Verify state recovery
    verify_state_machine_recovery();
    verify_operation_resumption();
}

// Test memory allocation failure scenarios
void test_memory_allocation_failures() {
    // Force malloc() failures
    simulate_heap_exhaustion();

    // Test critical allocation points
    test_bbstr_allocation_failures();
    test_fifo_allocation_failures();

    // Verify graceful degradation
    verify_allocation_failure_handling();
    verify_system_continues_operation();
}
```

Input Validation Edge Case Testing

Priority Test Scenario Targeted Failure Conditions Input Validation Points HIGH Extreme Sensor Value Injection Out-of-range sensor readings, overflow conditions All sensor input processing HIGH Invalid State Machine Inputs State transition triggers with invalid parameters State machine validation MEDIUM Communication Protocol Violations Malformed packets, invalid packet sequences Packet validation logic MEDIUM User Input Boundary Testing Button press timing violations, invalid sequences User interface validation MEDIUM Configuration Parameter Injection Invalid runtime parameter changes Runtime config validation

Test Implementation:

```
// Test extreme sensor value handling
void test_extreme_sensor_values() {
    // Test pressure sensor extremes
    inject_pressure_value(INT_MIN);
    inject_pressure_value(INT_MAX);
    inject_pressure_value(0xFFFFFFFF);

    // Test impedance sensor extremes
    inject_impedance_value(-1);
    inject_impedance_value(0);
    inject_impedance_value(UINT32_MAX);

    // Verify input validation
    verify_sensor_input_validation();
    verify_range_checking();
}

// Test invalid state machine inputs
void test_invalid_state_inputs() {
    // Inject invalid state values
    force_state_value(-1);
    force_state_value(999999);
    force_state_value(0xDEADBEEF);

    // Test invalid state transitions
    attempt_invalid_state_transitions();

    // Verify state validation
    verify_state_bounds_checking();
    verify_transition_validation();
}

// Test communication protocol violations
void test_protocol_violations() {
    // Send malformed packets
    send_packet_with_invalid_crc();
    send_packet_with_wrong_size();
    send_packet_sequence_out_of_order();

    // Test protocol state violations
    send_unexpected_packet_types();

    // Verify protocol error handling
    verify_packet_validation();
    verify_protocol_error_recovery();
}
```

Testing Implementation Strategy

Phase 1: Critical Vulnerability Testing (Week 1)

1. **FIFO Buffer Overflow Tests** - Highest risk to communication system
2. **State Machine Boundary Tests** - Array bounds violations in state transitions
3. **Interrupt Race Condition Tests** - Critical for Handle state machine integrity
4. **Memory Corruption Tests** - Buffer overflows in impedance processing

Phase 2: Integration Stress Testing (Week 2)

1. **Communication Protocol Stress** - Full protocol failure scenarios
2. **State Machine Combination Testing** - Complex multi-state scenarios
3. **Resource Exhaustion Testing** - Memory, CPU, I/O limits
4. **Real-World Misuse Scenarios** - User error simulation

Phase 3: Extended Duration Testing (Weeks 3-4)

1. **24+ Hour Continuous Operation** - Long-term stability verification
2. **Thermal Stress Testing** - Environmental extreme conditions
3. **Procedure Cycle Stress** - 10,000+ aspiration cycles
4. **Data Logging Stress** - Extended SD card operation

Phase 4: Failure Analysis and Remediation (Week 5)

1. **Root Cause Analysis** - Detailed failure investigation
2. **Code Review for Identified Issues** - Targeted vulnerability fixes
3. **Regression Testing** - Verify fixes don't introduce new issues
4. **Documentation Update** - Update robustness test procedures

Test Environment Requirements

Hardware Setup:

- Multiple Gen 3.0 units for parallel testing
- Environmental chambers for thermal stress
- Oscilloscopes for timing analysis
- Logic analyzers for communication debugging
- Adjustable power supplies for power stress
- Test fixtures for physical manipulation

Software Tools:

- Memory corruption detection tools
- Stack canary monitoring
- Heap integrity checking
- Communication protocol analyzers
- Real-time performance monitors
- Automated test harnesses

Safety Considerations:

- All testing performed on non-clinical units
- Comprehensive logging of all test procedures
- Immediate documentation of any safety-critical failures
- Escalation procedures for critical vulnerability discovery

Conclusion

This comprehensive robustness testing strategy targets specific vulnerabilities identified through detailed codebase analysis, combining architectural knowledge with real-world failure scenarios. The testing procedures are designed to systematically explore the boundaries of system behavior, identify failure modes that may not be obvious during normal operation, and ensure the device maintains safety and reliability under all conceivable stress conditions.

The enhanced testing strategy addresses vulnerabilities across **memory management**, **state machine logic**, **communication protocols**, **timing constraints**, and **integration points**. By implementing these tests systematically, the engineering team can identify and remediate potential failure modes before deployment, significantly improving the overall robustness and safety of the Inquis Gen 3.0 medical device system.

Special attention has been given to scenarios that require deep understanding of the codebase interactions - **race conditions between interrupts and state machines**, **buffer overflow vulnerabilities in communication paths**, and **timing-dependent failure modes** that could manifest only under specific operational stress conditions. This approach ensures comprehensive vulnerability coverage that goes beyond surface-level testing to explore the complex interactions that could lead to system failures in critical medical environments.