GRIFO Test Environment - Technical Analysis

Document Version: 1.0
Date: 2026-01-29
Purpose: Technical documentation of the GRIFO automatic test environment architecture

---

Table of Contents

Environment Overview

The GRIFO test environment is a Python-based automated test system for the GRIFO-F/TH radar system. It provides:

- **1553 bus interface** for radar communication (via native C++ library)
- **Serial terminal interface** for radar diagnostics monitoring
- **Power control** via BrainBox interface
- **PDF report generation** for test results
- **Comprehensive logging** and data recording

Key Technologies

- **Python 3.x** (local installation in environment)
- **SWIG bindings** for native C++ 1553 library
- **Serial communication** (pyserial)
- **PDF generation** (fpdf2)
- **Custom test framework** (leo_grifo_* modules)

Directory Structure

```
GrifoAutomaticTestEnv/
│
├── PlatformSimulator/                # Native 1553 interface components
│   ├── bin/
```

```
│   │   ├── interpreter.py            # SWIG-generated Python wrapper
│   │   ├── _interpreter.pyd          # Native C++ DLL (1553 hardware driver)
│   │   └── help/                     # Documentation files (.chm)
│   └── (additional platform files)
│
├── TestEnvironment/
│   ├── env/                          # Python libraries and utilities
│   │   ├── leo_grifo_1553.py        # 1553 interface wrapper
│   │   ├── leo_grifo_terminal.py    # Serial terminal handler
│   │   ├── leo_grifo_io_box.py      # Power control (BrainBox)
│   │   ├── leo_grifo_common.py      # Common utilities
│   │   ├── leo_grifo_core.py        # Core framework (recorder)
│   │   ├── leo_grifo_test_report.py # PDF report generation
│   │   ├── test_common_function.py  # Test helper functions
│   │   ├── logger.py                # Logging setup
│   │   ├── site-packages/           # Local Python packages
│   │   │   ├── defusedxml/
│   │   │   ├── fpdf/
│   │   │   ├── PIL/
│   │   │   ├── pyvisa/
│   │   │   ├── serial/
│   │   │   └── ...
│   │   └── __pycache__/
│   │
│   ├── scripts/                      # Test scripts
│   │   ├── __init__.py              # Environment setup (sys.path config)
│   │   ├── GRIFO_M_PBIT.py          # Main PBIT test script
│   │   ├── GRIFO_M_PBIT_mock.py     # Simulation mode mock (NEW)
│   │   └── (other test scripts)
│   │
│   ├── json/                         # Configuration files
│   │   └── (test configurations)
│   │
│   ├── LOG/                          # Test execution logs
│   └── pdf_reports/                  # Generated test reports
│
├── Documents/                         # Project documentation
└── run_batch.bat                      # Batch execution scripts
```

Core Components

1. `__init__.py` - Environment Bootstrap

Location: `TestEnvironment/scripts/__init__.py`

Purpose: Configure Python import paths for test scripts

```python
import os, sys
LIB_DIR = os.path.join(os.path.dirname(__file__), '..', 'env')
```

```
sys.path.append(LIB_DIR)
sys.path.append(os.path.join(LIB_DIR, 'site-packages'))
```

Key Function:

- Adds `env/` to Python path (for leo_grifo_* modules)
- Adds `env/site-packages/` to Python path (for dependencies)
- Enables relative imports from test scripts

1553 Interface Architecture

Native Layer: `interpreter` Module

Location: `PlatformSimulator/bin/interpreter.py` + `_interpreter.pyd`

Type: SWIG-generated Python bindings for C++ library

Key Classes:

- `PyInterfaceManager` - Main interface factory
- `Grifo 1553 Interface` - Specific 1553 hardware interface

Functionality:

- Low-level 1553 bus communication
- Message transmission/reception
- Field access and manipulation
- Hardware-specific drivers

Python Wrapper: `leo_grifo_1553.py`

Location: `TestEnvironment/env/leo_grifo_1553.py`

Class: `GrifoInstrumentInterface(TestCommonInterface, ABC)`

Key Methods:

```
def __init__(self, _timeout=0.5):
    """Initialize interface and get hardware reference"""
    mgr = interpreter.PyInterfaceManager()
    index = mgr.indexOf('Grifo 1553 Interface')
    self.grifo_1553 = mgr.getInterface(index)
    self.run(True)  # Start message reception

def check(self, expected_result, *fields, **kwargs) -> (bool, Any, AnyStr):
    """
    Verify message field value with optional timing and retry logic.

    Args:
        expected_result: Expected value or range (tuple for range)
```

```
        *fields: (message_name, field_name)
        **kwargs:
            - timeout: Max wait time for message reception
            - step: Polling interval for retry logic

    Returns:
        (success: bool, value: Any, error: str)
    """

def set(self, value, *fields, **kwargs) -> (bool, AnyStr):
    """
    Assign value to TX message field and optionally send.

    Args:
        value: Field value (None = send last committed message)
        *fields: (message_name, field_name)
        **kwargs:
            - commitChanges: Send message after assignment
            - errorInject: Inject CRC error for testing
    """

def get(self, *fields, **kwargs) -> (Any, AnyStr):
    """
    Read message field value.

    Args:
        *fields: (message_name, field_name)

    Returns:
        (value: Any, error: str)
    """

def getInterface(self):
    """Return raw 1553 interface object for advanced operations"""
    return self.grifo_1553

def run(self, enable: bool):
    """Start/stop message reception"""
    if enable:
        self.grifo_1553.start()
    else:
        self.grifo_1553.stop()
```

Global Singleton:

```
theGrifo1553 = GrifoInstrumentInterface(0.2)
```

Usage Pattern in Tests:

```
from leo_grifo_1553 import theGrifo1553

# Get interface object
interface = theGrifo1553.getInterface()

# Read message counter
count =
interface.getSingleMessageReceivedSz("B6_MsgRdrSettingsAndParametersTellback")

# Read field value
value = interface.getMessageFieldValue("B6_MsgRdrSettingsAndParametersTellback",
                                       "radar_health_status_...")

# Check field with automatic retry logic
success, value, error = theGrifo1553.check(
    "false",  # Expected value
    "B6_MsgRdrSettingsAndParametersTellback",
    "radar_health_status_...",
    timeout=5.0,  # Wait up to 5s for message
    step=0.1      # Poll every 100ms
)
```

Serial Communication

Base Class: `Terminal2Serial`

Location: `TestEnvironment/env/leo_terminal_serial.py`

Purpose: Generic serial communication wrapper using pyserial

Features:

- Configurable port, baud rate, stop bits, parity
- Line-feed mode for text-based protocols
- Callback-based message reception

GRIFO Implementation: `GrifoSerialTerminal`

Location: `TestEnvironment/env/leo_grifo_terminal.py`

Class: `GrifoSerialTerminal(Terminal2Serial)`

Configuration: Loaded from JSON config

```
{
    "serial_terminal": {
        "port": "COM2",
        "speed": "9600",
        "stop_bit": "1",
        "bytesize": 8,
```

```
        "parity": "N",
        "mode": "EXPECT_LF"
    }
}
```

Message Pattern Detection:

- `%%E` - Error messages (logged as critical)
- `%%F` - Fatal messages (logged as critical)
- `RECYCLE` - System restart keyword (case-insensitive)

Statistics Tracking:

```
self._serial_stats = {
    'total_messages': 0,        # Total messages received
    'error_messages': 0,        # %%E count
    'fatal_messages': 0,        # %%F count
    'recycle_count': 0,         # RECYCLE keyword occurrences
    'error_details': [],        # [(timestamp, message), ...]
    'fatal_details': [],        # [(timestamp, message), ...]
    'recycle_details': [],      # [(timestamp, message), ...]
}
```

API Methods:

```
def connect(self):
    """Connect to serial port"""

def disconnect(self):
    """Disconnect from serial port"""

def get_serial_statistics(self) -> dict:
    """Get current statistics snapshot"""
    return dict copy of _serial_stats

def reset_serial_statistics(self):
    """Reset statistics for new test run"""
```

Usage Pattern:

```
import leo_grifo_terminal

terminal = leo_grifo_terminal.GrifoSerialTerminal()
terminal.connect()

# At test run start
terminal.reset_serial_statistics()
```

```
# During test execution
# (messages received automatically in background)

# At test run end
stats = terminal.get_serial_statistics()
print(f"Errors: {stats['error_messages']}")
print(f"Fatal: {stats['fatal_messages']}")
print(f"Recycles: {stats['recycle_count']}")

terminal.disconnect()
```

Power Control

BrainBox Interface: `leo_grifo_io_box.py`

Location: `TestEnvironment/env/leo_grifo_io_box.py`

Purpose: Control radar power via external control box

Global Singleton:

```
theBrainBox = BrainBoxInterface()
```

Usage (via helper functions):

```
from test_common_function import power_grifo_on, power_grifo_off

# Power on radar with 3s wait
power_grifo_on(wait_after=3)

# Power off radar
power_grifo_off(wait_after=0)
```

Implementation:

```
def power_grifo_on(wait_after=0):
    setValue(theBrainBox, True, 'MAIN_POWER')
    ret, err = check(theBrainBox, 1, 'MAIN_POWER')
    time.sleep(wait_after)
    return ret

def power_grifo_off(wait_after=0):
    setValue(theBrainBox, False, 'MAIN_POWER')
    ret, err = check(theBrainBox, 0, 'MAIN_POWER', timeout=0.1)
```

```
    time.sleep(wait_after)
    return ret
```

Test Framework

Test Report: `leo_grifo_test_report.py`

Class: `testReport`

Methods:

```
def __init__(self, test_name):
    """Initialize report with test name"""

def open_session(self, session_name):
    """Start new test session section"""

def close_session(self):
    """End current session"""

def add_comment(self, comment):
    """Add text to report"""

def generate_pdf(self):
    """Generate final PDF report"""
```

Usage Pattern:

```
from leo_grifo_test_report import testReport

report = testReport(sys.argv[0])

report.open_session('Test Initialization')
report.add_comment('Starting test...')
# ... test operations ...
report.close_session()

report.generate_pdf()
```

Common Functions: `test_common_function.py`

Key Utilities:

```
def check(interface, expected, *fields, **kwargs):
    """Universal check function for any interface"""
```

```
def setValue(interface, value, *fields, **kwargs):
    """Universal set function for any interface"""

def get_test_name(file):
    """Extract test name from script file path"""

def startTest() -> datetime:
    """Log test start time"""

def stopTest() -> datetime:
    """Log test stop time"""
```

Core Recorder: `leo_grifo_core.py`

Global Object:

```
theRecorder = DataRecorder()
```

Purpose: Record test steps for later analysis/playback

Methods:

```
def add_step(self, data, success, description, error):
    """Record test step with outcome"""

def logStart(self, verbosity, log_dir):
    """Start recording session"""

def logStop(self):
    """Stop recording session"""
```

Python Environment

Local Packages (site-packages)

Location: `TestEnvironment/env/site-packages/`

Installed Packages:

- `defusedxml` - Secure XML parsing
- `fpdf2` - PDF generation
- `Pillow (PIL)` - Image processing
- `pyvisa` - Instrument control (if needed)
- `pyserial` - Serial communication

Python Version: Python 3.x (exact version determined by environment)

Import Resolution Order:

1. Test script directory
2. `env/` directory (leo_grifo_* modules)
3. `env/site-packages/` (dependencies)
4. System Python paths

Message Protocol

1553 Message Structure

Message Types Used in Tests:

B6: `B6_MsgRdrSettingsAndParametersTellback`

- **Direction:** RX (from radar to test system)
- **Purpose:** Radar settings confirmation + LRU status
- **Frequency:** 10 Hz (100ms period)
- **Key Fields:**
  - `bit_report_available` - BIT completion flag
  - `radar_fail_status` - Overall radar health (enum)
  - `array_status`, `pedestal_status`, etc. - LRU status (inverse logic)
  - `pressurization_status`, temperature alarms

LRU Status Fields (12 total):

```
# Inverse logic: false = OK, true = FAIL
"array_status"
"pedestal_status"
"processor_status"
"receiver_status"
"rx_front_end_status"
"servoloop_status"
"trasmitter_status"
"pressurization_status"
"processor_over_temperature_alarm"
"servoloop_over_temperature_alarm"
"trasmitter_over_temperature_alarm"

# Enum: RDR_OK / RDR_FAIL
"radar_fail_status"
```

B8: `B8_MsgBitReport`

- **Direction:** RX (from radar to test system)
- **Purpose:** Detailed BIT diagnostic results
- **Frequency:** 10 Hz (100ms period)
- **Key Field Categories:**

- Degradation conditions (12 fields)
- SRU failure locations (43 fields across 6 subsystems)
- Test results (118 fields across 10 test types)

Field Categories:

1. **Degradation Conditions** (12 fields)
   - System-level failures (BCN, groups, total radar fail)
2. **SRU Failure Locations** (43 fields)
   - Pedestal (5 SRUs)
   - Processor (14 SRUs)
   - Receiver (7 SRUs)
   - RX Frontend (6 SRUs)
   - Servoloop (3 SRUs)
   - Transmitter (8 SRUs)
3. **Test Results** (118 fields)
   - AGC tests (11)
   - Data Processor tests (14)
   - Integrated System tests (15)
   - Post Processor tests (8)
   - Power Supply tests (2)
   - Receiver/RX Frontend tests (7)
   - RX Module tests (6)
   - Servoloop tests (15)
   - Signal Processor tests (14)
   - Transmitter tests (26)

B9: Message (target data)

- **Direction:** RX (from radar)
- **Purpose:** Target track data
- **Frequency:** 50 Hz (20ms period)

Message Field Naming Convention

Pattern: `{subsystem}_{structure}_{field_name}`

Example:

```
radar_health_status_and_bit_report_valid_RdrHealthStatusAndBitReport_processor_sta
tus
|                                                                    |
└─ Subsystem/Group                                                   └─ Field
name
```

Simulation Mode

Overview

Simulation mode allows test execution without physical hardware using mock objects.

Activation:

```
python GRIFO_M_PBIT.py --simulate
```

Architecture

Mock Module: `GRIFO_M_PBIT_mock.py`

Key Components:

1. `MockGrifo1553Interface`
   - Simulates 1553 message reception
   - Configurable BIT timing (15-25s default)
   - Scenario-based field values (normal/pedestal_fail/random_failures)
2. `MockSerialTerminal`
   - Simulates serial message reception
   - Generates %%E, %%F, RECYCLE messages
   - Compatible API with real terminal
3. `MockBrainBox`
   - Simulates power control (no-op)
   - Logs power state changes
4. `setup_simulation()`
   - Monkey-patches global singletons
   - Replaces `theGrifo1553` with mock
   - Replaces `theBrainBox` with mock

Configuration Options

```
# In GRIFO_M_PBIT_mock.py

# BIT completion timing
PBIT_TIME_MIN = 15.0
PBIT_TIME_MAX = 25.0

# Test scenario
SIMULATION_SCENARIO = 'normal'  # or 'pedestal_fail' or 'random_failures'
```

```
# Serial message generation
SIMULATE_SERIAL_ERRORS = True   # Generate %%E messages
SIMULATE_SERIAL_FATAL = False   # Generate %%F messages
SIMULATE_RECYCLE_EVENTS = True  # Generate RECYCLE at power-on
```

Usage in Tests

Minimal Changes Required:

```
def test_proc():
    # Simulation mode detection
    if '--simulate' in sys.argv:
        from GRIFO_M_PBIT_mock import setup_simulation, create_mock_terminal
        setup_simulation()
        use_mock_terminal = True
    else:
        use_mock_terminal = False

    # ... test initialization ...

    # Terminal creation (conditional)
    if use_mock_terminal:
        terminal = create_mock_terminal()
    else:
        terminal = leo_grifo_terminal.GrifoSerialTerminal()

    # Rest of test remains UNCHANGED
```

Key Benefits:

- ☑ Zero impact on production test code
- ☑ No modification to environment Python
- ☑ No dependency changes
- ☑ Transparent operation (same API)
- ☑ Reproducible test scenarios
- ☑ Faster test development/debugging

Known Issues and Limitations

1. Inverse Logic Fields

Many status fields use **inverse logic**:

- `false` = OK/PASS
- `true` = FAIL

Affected Fields:

- All LRU status fields in B6
- Most test result fields in B8

Reason: Hardware convention for active-high failure flags

2. Known Hardware Setup Limitations

Some tests may consistently fail in certain setups:

- **Pedestal status**: Often fails when physical pedestal unit not connected
- **Pressurization status**: Requires sealed system

Solution: Use `KNOWN_FAILURES` configuration list to track expected failures

3. Native Library Dependencies

The `interpreter` module requires:

- `_interpreter.pyd` native DLL
- Compatible hardware drivers
- Proper 1553 bus interface card

Not portable across different systems without hardware.

4. Timing Considerations

- **BIT Completion:** Typically 15-30s, but can vary
- **Message Periods:** Fixed by radar firmware
  - B6: 100ms (10 Hz)
  - B7: 40ms (25 Hz)
  - B8: 100ms (10 Hz)
  - B9: 20ms (50 Hz)

Test Execution Flow

Standard Execution (with Hardware)

```
1. Initialize environment
   ├─ Load Python modules
   ├─ Connect to 1553 interface
   ├─ Connect to serial terminal
   └─ Initialize report

2. Test preparation
   ├─ Power off radar
   └─ Wait stabilization

3. For each test repetition:
   ├─ Power on radar
   ├─ Wait for BIT completion (180s timeout)
```

```
    ├─ Verify B6 LRU status (12 fields)
    ├─ If failures: drill-down B8 diagnostics (185 fields)
    ├─ Collect serial statistics
    ├─ Generate per-run report
    └─ Power off radar

4. Generate final statistics report
    ├─ Aggregate all runs
    ├─ Timing analysis
    ├─ Failure analysis
    └─ Test verdict

5. Cleanup
    ├─ Disconnect serial
    ├─ Generate PDF
    └─ Exit
```

Simulated Execution (without Hardware)

Same flow, but with mock objects:

- No actual 1553 communication
- No actual serial port access
- No actual power control
- Simulated timing and responses

Execution Time: ~5 minutes for 10 runs (vs 30+ minutes real)

Troubleshooting

Common Issues

1. **Import Error: `interpreter` module not found**
    - Ensure `PlatformSimulator/bin` is accessible
    - Check Python path configuration
    - Verify native DLL exists
2. **Serial Port Error: Cannot open COM port**
    - Check port configuration in JSON
    - Verify port permissions
    - Ensure port not in use by another application
3. **BIT Never Completes (timeout)**
    - Check 1553 message reception
    - Verify radar power state
    - Check `bit_report_available` field polling
4. **Field Value Always `None`**

- Verify message name spelling
- Check field name (case-sensitive)
- Ensure message being received (check counter)

Debug Techniques

Enable Verbose Logging:

```
logging.basicConfig(level=logging.DEBUG)
```

Check Message Reception:

```
for i in range(100):
    count = interface.getSingleMessageReceivedSz("B6_...")
    print(f"Message count: {count}")
    time.sleep(0.1)
```

Dump Field Value:

```
value = interface.getMessageFieldValue("B6_...", "field_name")
print(f"Raw value: {value!r}")  # Note: !r for repr
```

Future Enhancements

Potential Improvements

1. **Configuration File Support**
   - JSON-based test configuration
   - Scenario definitions
   - Field value expectations
2. **Data Logging**
   - CSV export of all field values
   - Time-series recording
   - Replay capability
3. **Enhanced Mock**
   - Load real message captures
   - Timing-accurate playback
   - Failure injection scenarios
4. **Integration with pybusmonitor1553**

- Use your module as message source
- UDP-based simulation
- Full protocol stack testing

References

Related Files

- **C++ ICD Header:** `___OLD/_old/cpp/GrifoScope/GrifoSdkEif/pub/TH/th_b1553_icd.h`
- **Message Definitions:** `pybusmonitor1553/Grifo_E_1553lib/messages/`
- **ICD Documentation:** `doc/ICD-Implementation-Notes.md`

Contact Information

For questions about this environment, refer to:

- Original test developers (Genova team)
- GRIFO radar system documentation
- MIL-STD-1553 protocol specifications

Appendix: Quick Reference

Command Line

```
# Run test with hardware
python GRIFO_M_PBIT.py

# Run test in simulation mode
python GRIFO_M_PBIT.py --simulate

# Change number of repetitions (edit script)
NUMBER_OF_REPETITIONS = 10
```

Key Global Objects

```
from leo_grifo_1553 import theGrifo1553          # 1553 interface
from leo_grifo_io_box import theBrainBox         # Power control
from leo_grifo_core import theRecorder           # Data recorder
```

Common Code Patterns

```
# Check field value
success, value, error = theGrifo1553.check(
    expected_value,
```

```
    message_name,
    field_name,
    timeout=5.0,
    step=0.1
)

# Power cycle
power_grifo_off(wait_after=3)
power_grifo_on(wait_after=0)

# Report section
report.open_session('Test Phase 1')
report.add_comment('Performing checks...')
report.close_session()
```

End of Document