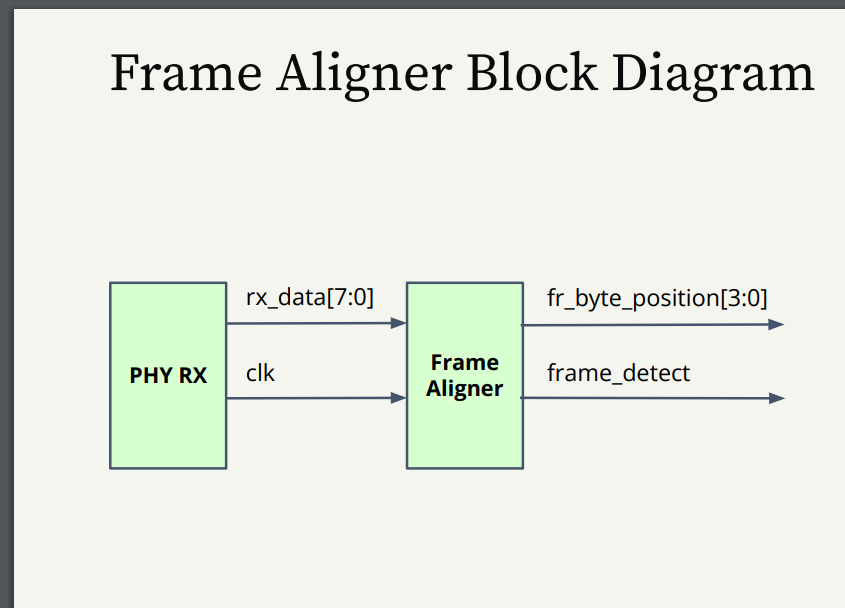
Frame Aligner report

1. **Frame Aligner Module Design Overview**
2. **Goals of Verification**
3. **Verification Environment**
4. **Verification Plan**
5. **Test Scenarios**
6. **Results and Analysis**
7. **Conclusion**
8. **Simulation Environment**
9. **Simulation Flow**
10. **Verification Environment**
11. **Monitors and Scoreboard**
12. **Coverage**
13. **Features to Verify**
14. **Misalignment Detection**
15. **Byte Counter**
16. **Header Coverage**
17. **Byte Position Coverage**
18. **Frame Detection Coverage**
19. **Functional Coverage**
20. **Bug Description**

**1. Design Overview 1. Introduction**

**The Frame Aligner module is designed to monitor an incoming data stream for specific header patterns (0xAFAA and 0xBA55). When these patterns are detected in three consecutive frames, the module declares the data stream as aligned. Conversely, if four consecutive frames do not contain the expected header, the module signals that the stream is out of alignment.**



**2. Goals of Verification**

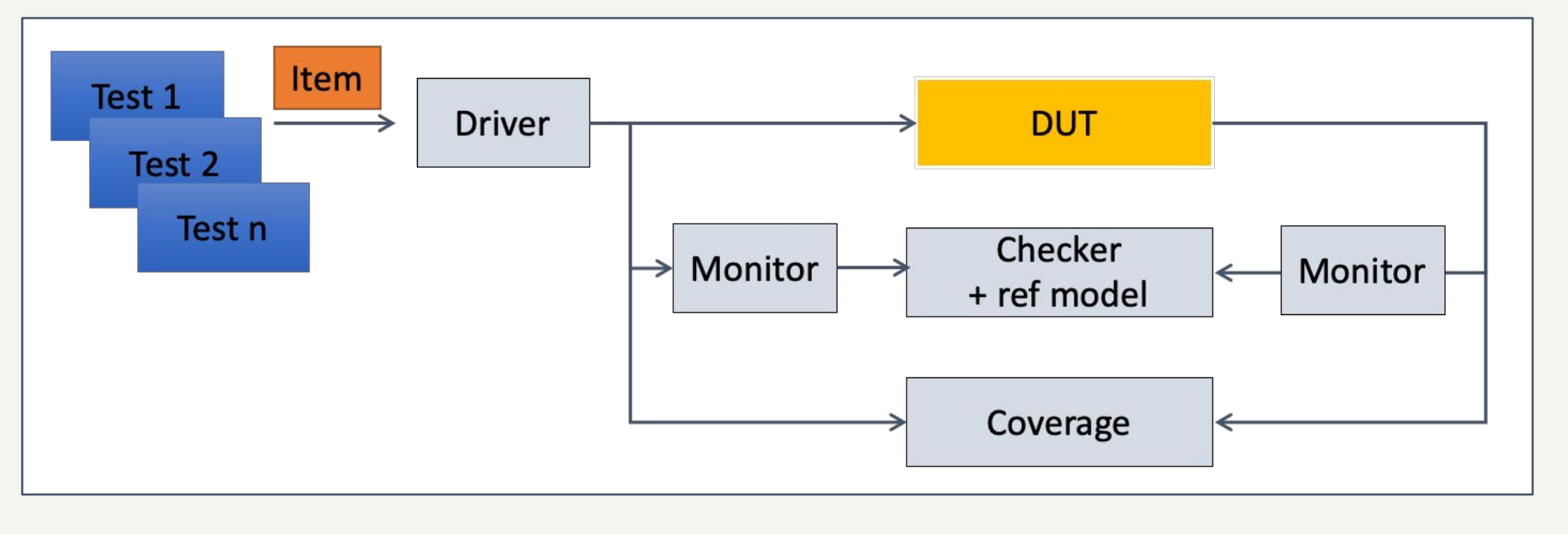
**The primary goals of the verification process are:**

* **Header Pattern Detection: Confirm that the module correctly identifies the specified header patterns within the data stream.**
* **Alignment Verification: Ensure that the module correctly indicates alignment when three consecutive headers are detected.**
* **Misalignment Handling: Verify that the module signals misalignment when four consecutive frames lack the required header.**
* **Robustness and Resilience: Ensure the module's reliability in various scenarios, including noise and partial matches in data.**

**3. Verification Environment**

**The verification environment consists of several key components:**

* **Driver: Sends a range of pre-defined and randomized data sequences to the Frame Aligner module.**
* **Monitor: Observes and records outputs, such as alignment signals and frame detection status.**
* **Checker: Compares observed outputs against expected behavior to determine if the module meets the verification goals.**
* **Scoreboard: Tracks overall pass/fail results for each test case.**

****

**4. Verification Plan**

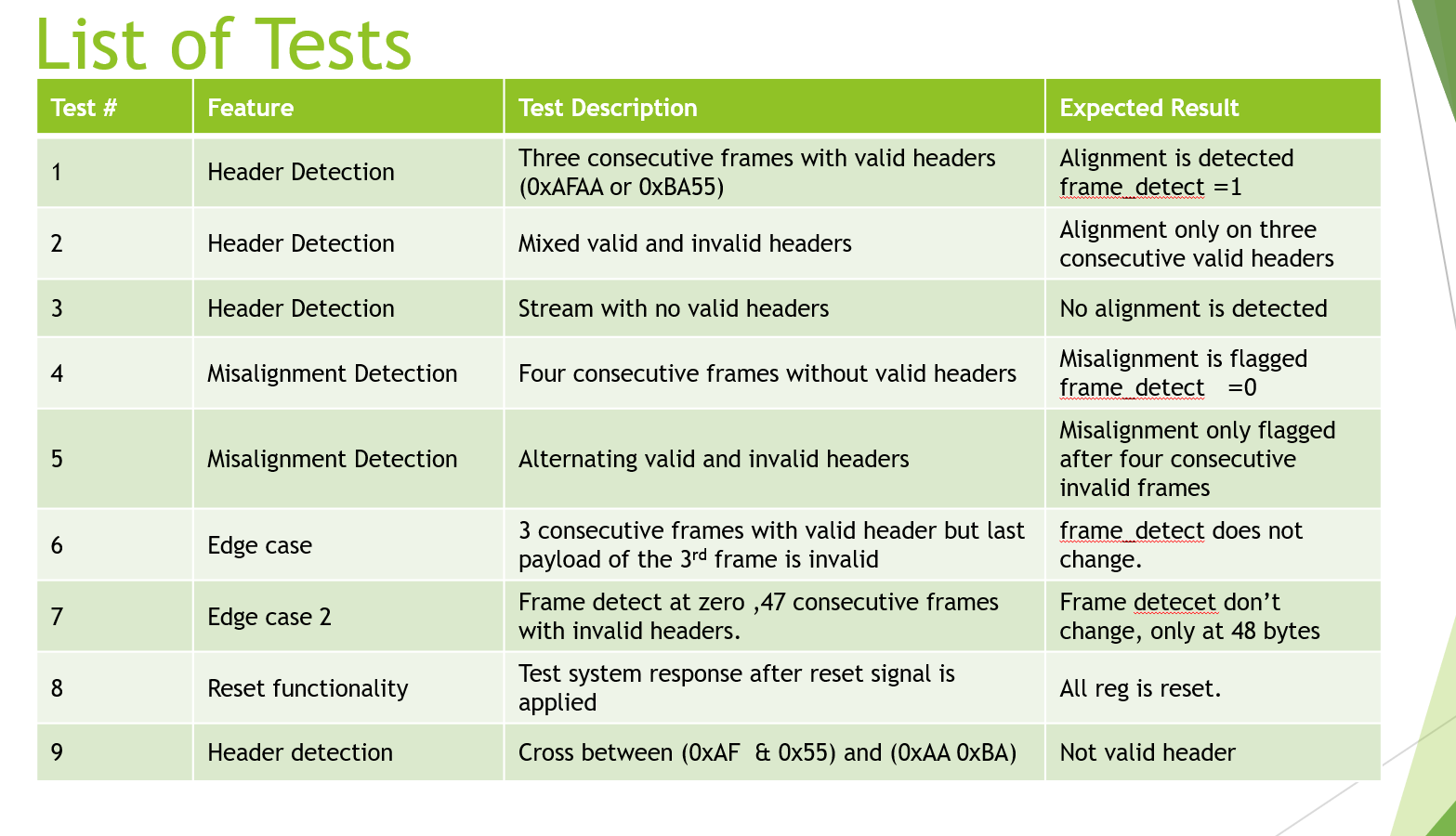
**The plan includes the following elements:**

* **Coverage Metrics:**
  + **Detection of both 0xAFAA and 0xBA55 header patterns.**
  + **Detection of misalignment when headers are missing.**
* **Assertions and Properties:**
  + **Properties to confirm alignment occurs only after three valid consecutive headers.**
  + **Properties to detect a misalignment signal after four frames without headers.**

**5. Test Scenarios**

**The test scenarios cover the following conditions:**

* **Basic Test: A simple data stream with consistent headers to check alignment detection.**
* **Edge Cases:**
  + **Streams that start aligned but lose headers midway to test misalignment detection.**
  + **Streams with noise and non-header data to verify that the module does not falsely detect alignment.**
* **Stress Tests: Long data streams with mixed valid and invalid headers to ensure performance under extended conditions.**

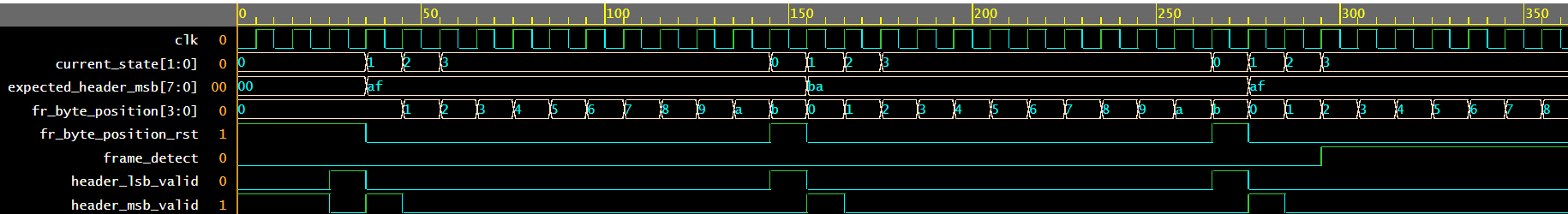
****

**6. Results and Analysis**

**The results of the simulation tests are summarized below:**

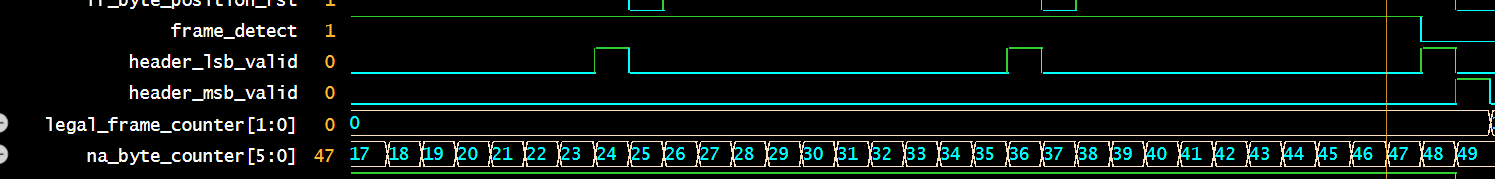
* **Pass/Fail Results: Test outcomes based on the observed outputs.**
* **Discrepancies Analysis: Identifies discrepancies and their causes.**
* **Coverage Analysis: Graphical representation of coverage metrics.**

3 consecutive valid header -> frame detect ( 0 -> 1)

****



4 consecutive Invalid header -> frame detect ( 1 -> 0)

****



**7. Conclusion**

**The design met the verification goals with the exception of certain edge cases, including the ambiguity regarding the byte position when a valid header is detected in the payload of a frame without a valid header at the start.**

**8. Simulation Environment**

**The simulation is carried out using a SystemVerilog-based simulation environment, which includes a dedicated testbench for the Frame Aligner module. The testbench simulates the behavior of the module under various conditions by feeding it different data streams, ensuring accurate monitoring of the alignment and misalignment conditions.**

**9. Simulation Flow**

**The simulation is structured in phases:**

* **Phase 1: Basic Functionality: Simple data streams containing continuous, valid headers to verify the module's alignment detection.**
* **Phase 2: Header Variability: Data streams with varied and mixed headers (0xAFAA, 0xBA55).**
* **Phase 3: Misalignment Checks: Data sequences with four consecutive frames missing valid headers.**
* **Phase 4: Noise and Edge Conditions: Data streams with random non-header bytes.**

**Each phase evaluates the module's outputs, focusing on alignment and misalignment detection.**

**10. Verification Environment**

**Verification Components:**

* **Device Under Test (DUT): The DUT is the Frame Aligner module, which processes the incoming data stream and aligns frames based on detecting specific header patterns (0xAFAA, 0xBA55).**
* **Driver: Feeds a continuous stream of data to the DUT, mimicking real-world data behavior with both valid and invalid header patterns.**
* **Monitor: Observes and records the DUT’s outputs, ensuring that all relevant data is collected for further analysis.**
* **Scoreboard: Compares the DUT's outputs against expected results, logging discrepancies for further analysis.**

**11. Monitors and Scoreboard**

* **Monitors: Continuously observe input signals (e.g., rx\_data) and output signals (e.g., frame\_detect, fr\_byte\_position).**
* **Scoreboard: Compares the DUT's outputs with expected values, logging discrepancies and raising alerts for errors.**

**12. Coverage**

**Coverage Metrics:**

* **Header Pattern Detection: Verifies that 0xAFAA and 0xBA55 are detected correctly.**
* **State Transitions: Ensures correct transitions between alignment and misalignment states.**
* **Misalignment Handling: Verifies correct misalignment detection after four consecutive frames with no header.**

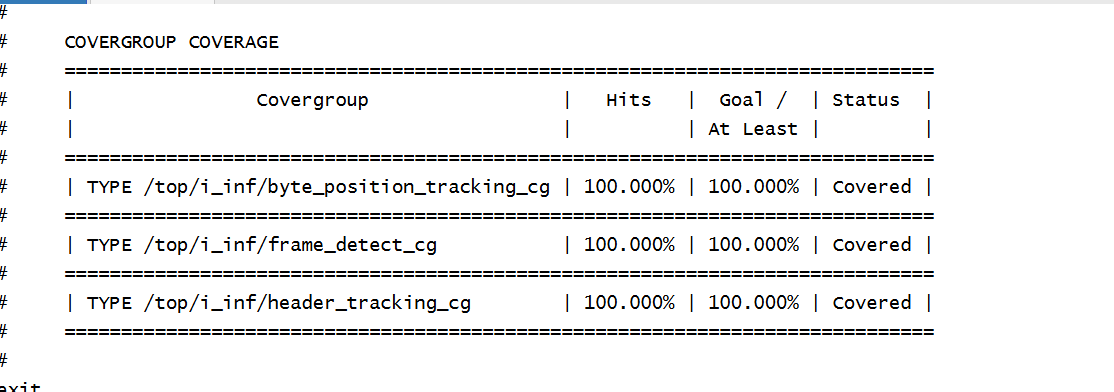
**Coverage Collection:**

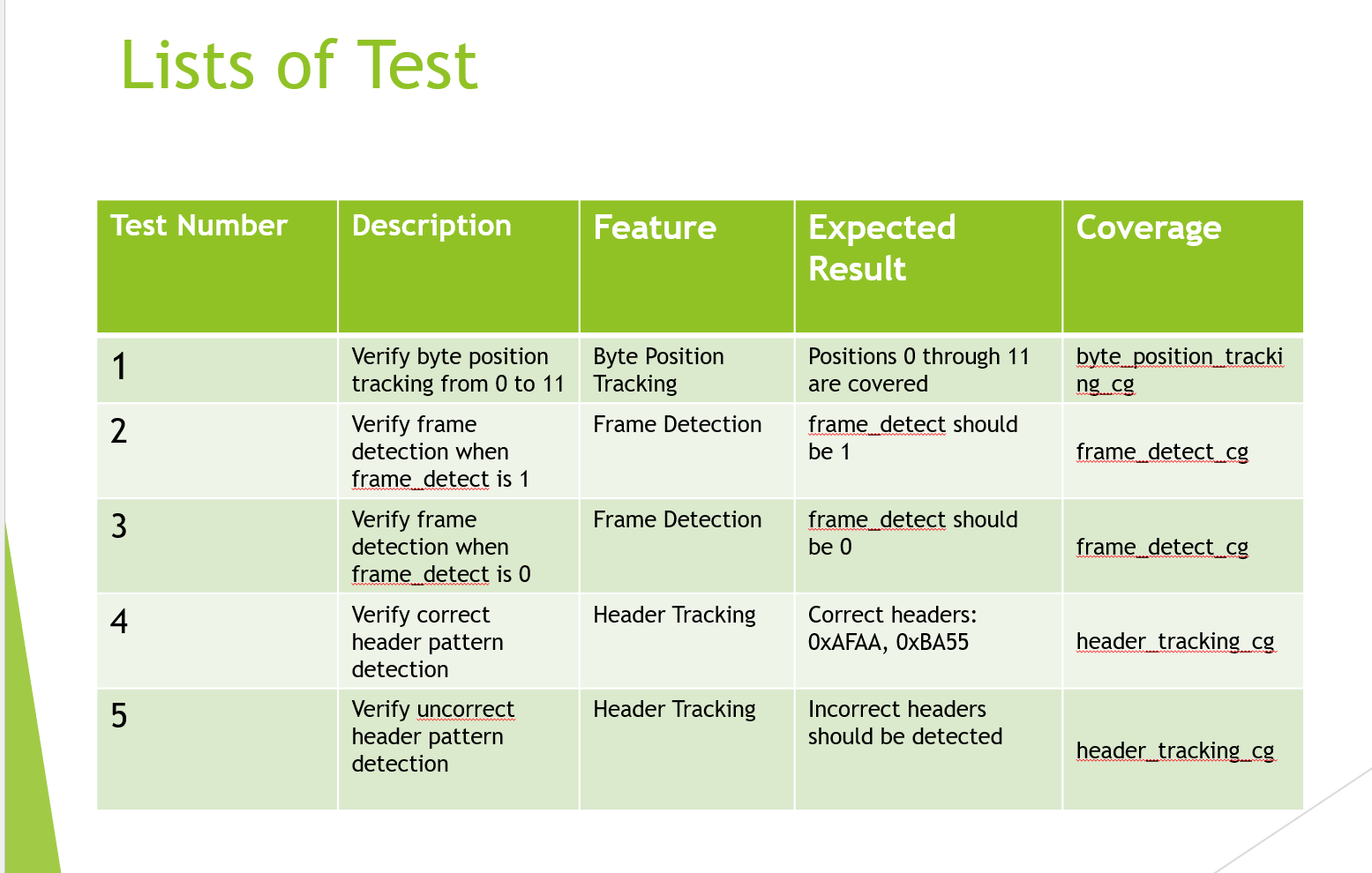
* **Header Detection: Monitors the occurrence of valid headers.**
* **State Transitions: Tracks when alignment and misalignment states change.**
* **Misalignment Handling: Verifies misalignment detection after 48 invalid bytes.**

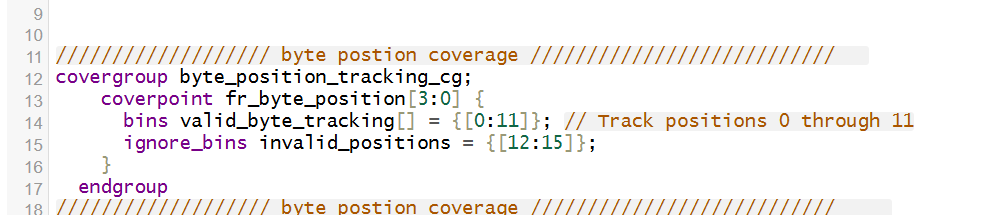
**13. Features to Verify**

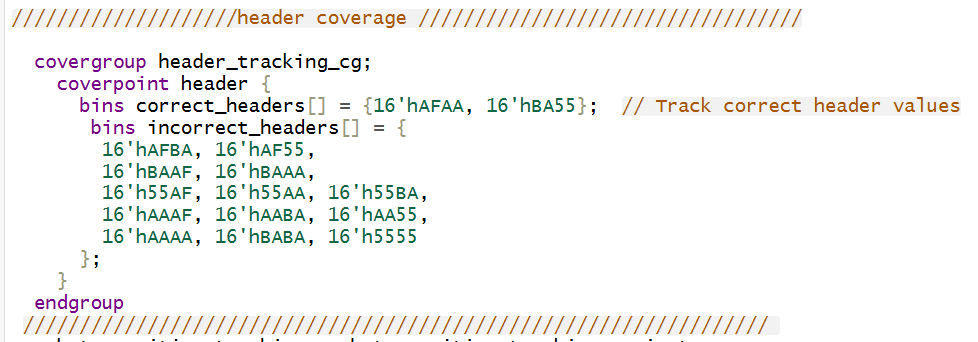
**Key features to verify include:**

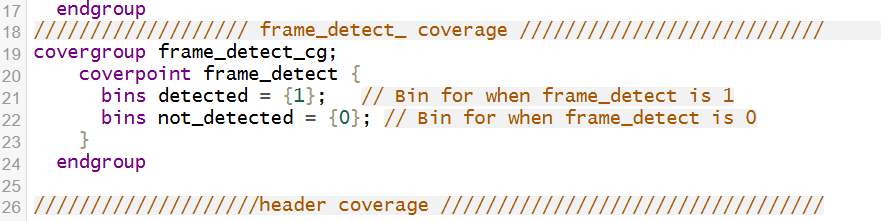
* **Frame Detection and Alignment: Ensuring correct header detection and alignment signaling.**
* **Misalignment Handling: Correct signaling when frames are misaligned.**
* **Byte Counter Functionality: Verifying the behavior of the na\_byte\_counter when no valid headers are detected for 48 consecutive bytes.**

****

****

****

****

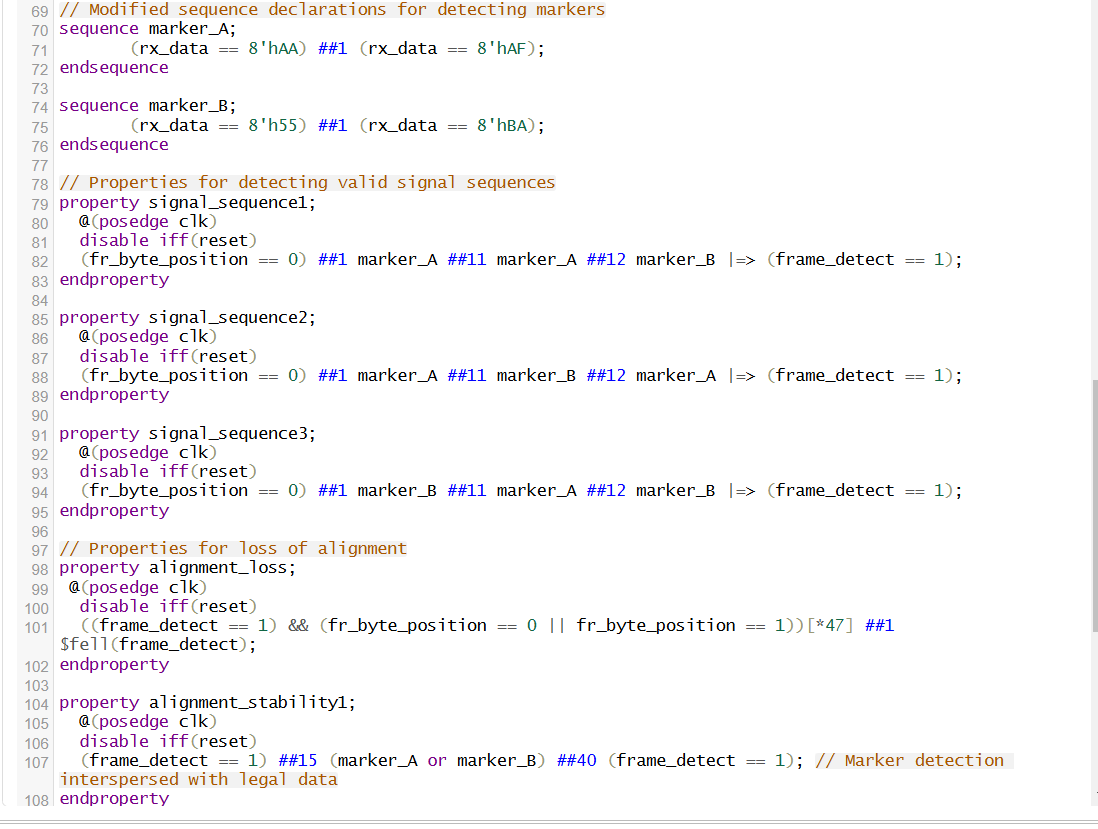
****

**14. Misalignment Detection**

* **Four Consecutive Frames without Valid Headers: Ensures the module detects misalignment after four consecutive invalid frames.**
* **Alternating Valid/Invalid Headers: Verifies that misalignment is detected only after four consecutive invalid frames.**

**15. Byte Counter**

* **48 Consecutive Bytes without Valid Header: Ensures that after 48 invalid bytes, the system signals misalignment.**
* **Valid Headers Resetting the Counter: Verifies that valid headers reset the byte counter.**

****

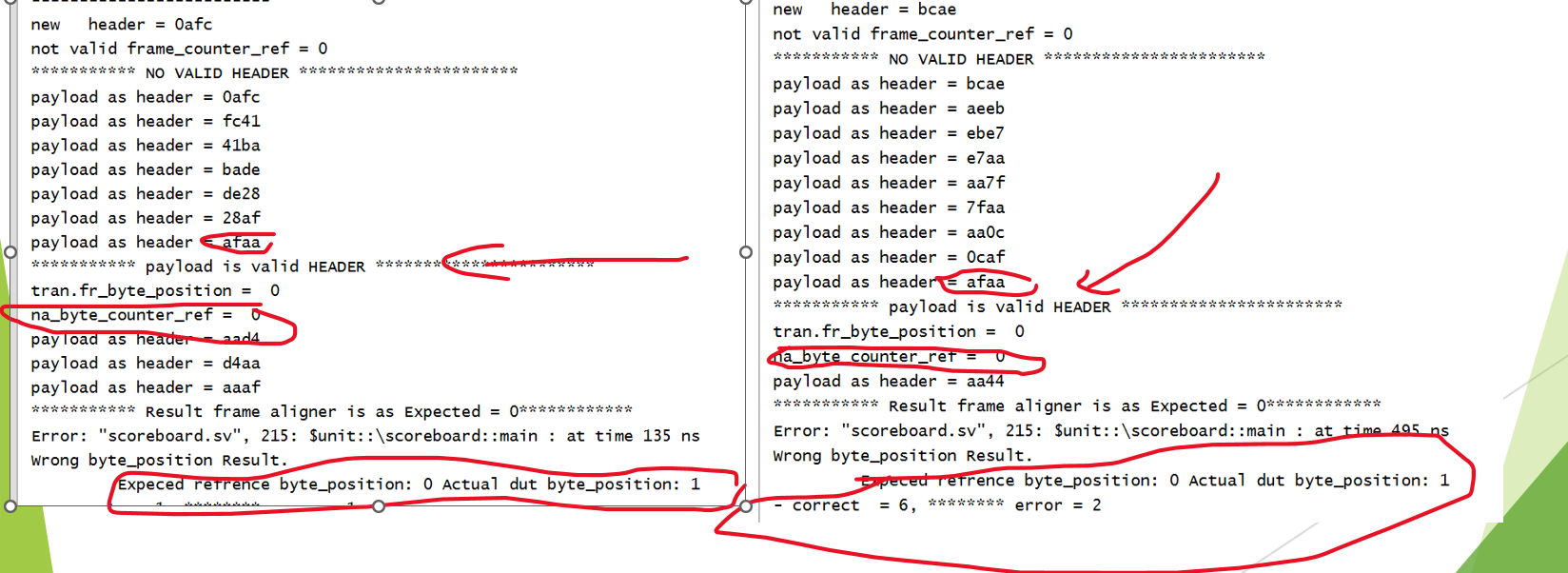
**Bug Description**

**The issue arises when the payload contains two bytes that together form a valid header (e.g., 0xAFAA or 0xBA55), but the frame itself lacks a valid header. The specification does not clearly define the behavior of the byte position when a valid header is detected within the payload of a frame that does not have a valid header at the start.**

**Issue Details:**

* **Valid Header in Payload: The module detects a valid header in the payload, such as 0xAFAA or 0xBA55, but the frame itself is not aligned (i.e., no valid header at the start).**
* **Byte Position Ambiguity: The spec does not clarify what the byte position should be when a valid header is detected in the payload but not at the start of the frame.**
* **Expected Behavior: The na\_byte\_counter should reset when a valid header is detected in the payload, but the byte position should also be reset to zero. However, this behavior is not defined in the specification.**

**This leads to errors in the system, as the byte position may not be correctly handled when a valid header is detected mid-frame.**

****

**16. Recommendations**

* **Clarify Byte Position Handling: The specification should explicitly define how the byte position is handled when a valid header is detected in the payload of a frame without a valid header at the start.**
* **Update the Specification: Include more detailed information on how byte positions are tracked and reset during header detection.**