**A picture containing engine, airplane

Description automatically generated**

**IF Arbiter Verification plan**

Raghda Kais

**25.01.2024**

**REV 1.0**

This document contains Intellectual Property (IP) that is owned and belongs to GuardKnox Cyber Technologies Ltd. GuardKnox Cyber Technologies owns this document and its information and therefore it is forbidden to copy, duplicate, transfer and/or disclose any information, fully or partially, that is included in this document without written consent from GuardKnox Cyber Technologies Ltd.



**© GuardKnox Cyber Technologies Ltd.**

Business Classification: Confidential

Changes History Table

| **Rev.** | **Date** | **initiator** | **Change description** | **Approver** |
| --- | --- | --- | --- | --- |
| **1** | **25/01/2024** | **Raghda Kais** | **Review 1** |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Content

1 Introduction 4

1.1 Scope and Objectives 4

1.2 Definitions, Acronyms & Abbreviations 4

1.3 Applicable Standard and Regulations 4

1.4 References 5

2 Dut Overview 6

2.1 Block diagram 6

2.2 Block Connection 6

2.3 Interfaces 7

2.4 Registers 8

3 Verification Plan 9

4 Verification Environment 10

4.1 Verification Environment Detailed Description 11

4.1.1 Verification Components Table 12

4.1.2 UVM Seq item 13

4.1.3 Simulation configuration 15

4.1.4 Compare testpoints 15

4.1.5 Interfaces vectors 15

4.1.6 Reference models 15

4.1.7 Data Flow and Operation 16

4.1.8 Pass/Fail criteria 16

5 Test Plan 17

6 Coverage 26

6.1 Code Coverage 26

6.2 Functional Coverage 26

Track functional coverage to ensure that the testcases cover different functional scenarios and configurations. 26

LIST OF FIGURES

[Figure 1 – Top verification Environment diagram 7](#_Toc139288314)

[Figure 2 – testbench diagram 8](#_Toc139288315)

[Figure 3 – Reference model diagram 8](#_Toc139288316)

LIST OF TABLES

Table 1 – Definitions Table 4

Table 2 – Standard and Regulations Table 4

Table 2 – references documents Table 4

Table 4 – iterfaces Table 5

Table 5 – Registers Table 5

Table 6 – Verification Components Table 8

Table 6 – Testoints list Table 8

Table 7 – Test plan Table 10

Introduction

The document describes the verification goals for testing and the underlying requirements for generation, reference modelling and coverage.

## Scope and Objectives

This document describes the required functionality to be tested as well as required checkers, assertions. This document provides a detailed description of the verification environment(s) used to cover the verification of the design under test.

Definitions, Acronyms & Abbreviations

| **Acronym** | **Definition** |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

***Table 1 – Definitions Table***

## Applicable Standard and Regulations

| **Ref.** | **Doc Name** | **Link to ref doc** |
| --- | --- | --- |
| **[A1]** | Automotive SPICE 3.1 VDA-HIS scope |  |
| **[A2]** | ISO 9001:2015 |  |
| **[A3]** | ISO 26262:2018 Functional Safety |  |

***Table 2 – Standard and Regulations Table***

## References

The following references contain useful information concerning the standard compatibility and other supportive documentation. The table contains links to all related documents, DUT specification, reference models specification, tests list spreadsheet, etc.

| **Ref.** | **Doc Name** | **Author(s)** | **safety item** | **Link to ref doc** |
| --- | --- | --- | --- | --- |
|  | Detailed Design Interface Arbiter |  |  | <https://guardknox365.sharepoint.com/:w:/r/sites/hwteam/_layouts/15/Doc.aspx?sourcedoc=%7B7CDFA0F9-E6C2-4E98-8875-EE6DD3940A63%7D&file=Detailed%20Design%20Interface%20Arbiter.docx&action=default&mobileredirect=true> |
|  | Testplan |  |  |  |
| **[R3]** |  |  |  |  |
| **[R4]** |  |  |  |  |
| **[R5]** |  |  |  |  |
| **[R6]** |  |  |  |  |

***Table 3 – references documents Table***

Dut Overview

The ‘Interface Arbiter’ determines which interface the data is received from next, making sure no interface is starved.

Each of the input source can be configured to either High (H) or Low (L) priority.

The arbiter prefers to select an interface request from the high priority group.

If there is no request from the high priority interface group (no data to send or neither one of the interfaced is configured as high priority) the arbiter selects from the low priority interfaces group.

If high priority and low priority interfaces groups were set together for long period of time (the high priority wins the selections), to avoid starvation of the low priority inputs, after predefined successive requests a selection from the low priority interfaces will be generated.

Within a priority, the selection of an interface is done using Round-Robin (RR) mechanism.

The ‘Interface Arbiter’ receives 3 data buses, each bit signifies one interface, the busses are:

                   ‘Interface Priority’ – indicates for each interface if it’s ‘High Priority or ‘Low Priority’

                   ‘Interface Status’ - indicates for each interface if it has anything to transmit

                   ‘Interface Enabled’ – indicates for each interface if it is enabled or not

The ‘Interface Arbiter’ outputs the number (based on the interfaces bit number in the bus) of interface the data should be received from

## Block diagram

A screenshot of a computer

Description automatically generated

**Figure 1 – High Level Block Diagram**

## Block Connection

A screenshot of a computer

Description automatically generated

**Figure 2 – L3 L4 checksum block connections**

## Interfaces

|  |
| --- |
| **Arbiter controller** |
|  |  |  |  |  |
| **idx** | **Name** | **Type** | **Dir** | **note** | |
| **1** | **interface\_priority\_bus** | **bit [if\_count-1:0]** | **Input** |  | |
| **2** | **interface\_status\_bus** | **bit [if\_count-1:0]** | **Input** |  | |
| **3** | **interface\_enabled\_bus** | **bit [if\_count-1:0]** | **Input** |  | |
| **4** | **low\_prio\_timeout** | **bit [$clog2(if\_count)-1:0]** | **Input** |  | |
| **5** | **selected\_prio** | **bit** | **output** |  | |

***Table 1 – iterfaces Table***

|  |
| --- |
| **axis\_iarb\_if** |
|  |  |  |  |  |
| **idx** | **Name** | **Type** | **Dir** | **note** | |
| **1** | **m\_axis\_iarb\_ready** | **bit** | **Input** |  | |
| **2** | **m\_axis\_iarb\_valid** | **bit** | **output** |  | |
| **3** | **m\_axis\_iarb\_interfaceid** | **bit [76:0]** | **output** |  | |

***Table 2 – iterfaces Table***

|  |
| --- |
| **Clk&rst** |
|  |  |  |  |  |
| **idx** | **Name** | **Type** | **Dir** | **note** | |
| **1** | **Sys\_clk** | **bit** | **input** | **The ststem clok** | |
| **2** | **srst\_n** | **bit** | **input** | **Synchronous reset\_n** | |
| **3** | **asrst\_n** | **bit** | **input** | **ASynchronous reset\_n** | |

***Table 8 – iterfaces Table***



Verification Plan

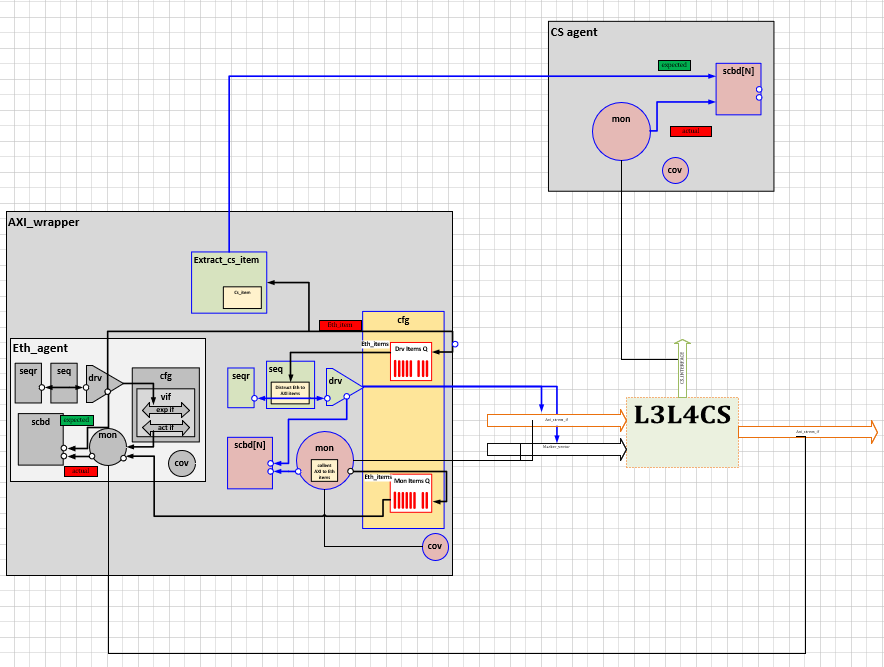
**Verification environmant:**

**Verification methodolgy:**

Verification Environment

***Figure 3 – Top verification*** ***Environment diagram***

Verification Environment Detailed Description

******

***Figure 4 –verification*** ***Environment diagram***

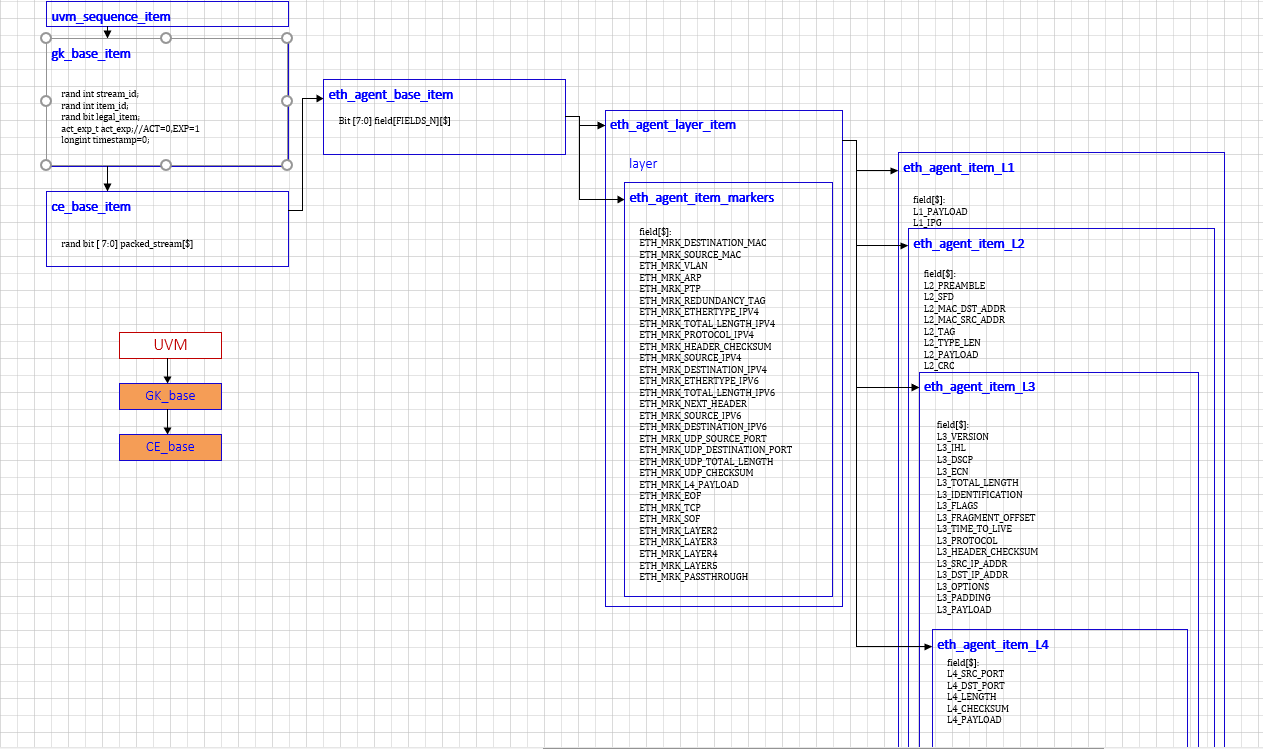
### Verification Components Table

verification components with detailed description

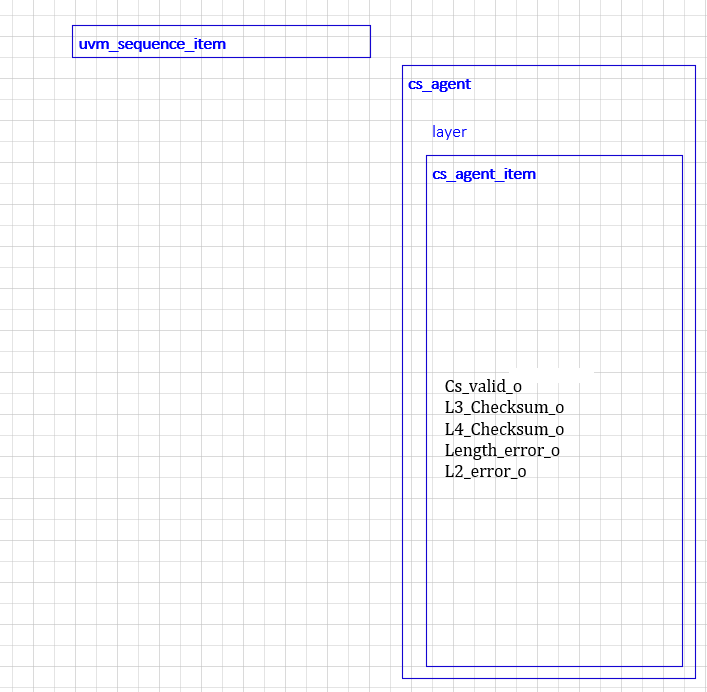
| **idx** | **verification comp** | **Type** | **Description** |
| --- | --- | --- | --- |
| **1** | Axi\_wrapper\_agent | Uvm\_agent | Agent class itself is a top-level container class for the components and they are the bottom of test bench hierarchy. the components are eth\_driver, eth\_sequencer, eth\_monitor. |
| **2** | Axi\_wrapper \_driver | Uvm\_drvier | eth\_driver Collects the eth frame data from the sequencer and sends to the dut via the axi interfacel. |
| **3** | Axi\_wrapper monitor | Uvm\_monitor | eth\_monitor is a component which receives eth frame via axi inf from the DUT and and parse it and then send to the scoreboard via analysis port. |
| **4** | Eth\_item\_coverage | Uvm\_subscriber | Eth\_item\_coverage collects eth\_items for collection of a coverpoints. |
| **5** | CS agent | Uvm\_agent | Agent class itself is a top-level container class for the components and they are the bottom of test bench hierarchy. the components in this agent is only cs monitor |
| **6** | CS\_monitor | Uvm\_monitor | cs\_monitor is a component which receives cs outputs from the send it to the scoreboard via analysis port. |
| **7** | CS\_item\_coverage | Uvm\_subscriber | Cs\_coverage collects cs\_items for a collection of a coverpoints. |
|  | Cs\_Scoreboard | Uvm\_scoreboard | Reciverd transcation of actual cs and compare with the transaction of the expected cs that generated in extract cs item inside axi wrapper agent. |
| **8** | test | Uvm\_test | The test extends uvm\_test. It consists of instances for sequences, environments and interfaces such as axi\_stream\_intf, further in the build phase object is created for components.  In the run phase the basic sequence is created and started on the sequencer within phase. raise\_objection and phase. drop\_objection. |
| **9** | Env | Uvm\_env | environment is a container component for assembling sub component of test like CS\_agent and axi\_wrapper\_agent. |

***Table 12 – Verification*** ***Components Table***

### UVM Seq item

******

***Figure 5 –eth\_seq\_item\_frames***

******

***Figure 6 –cs\_seq\_item***

### Simulation configuration

### Compare testpoints

| **idx** | **Testpoint name** | **RTL loaction** | **Agent associate** | **Description** |
| --- | --- | --- | --- | --- |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |
| **4** |  |  |  |  |
| **5** |  |  |  |  |
| **6** |  |  |  |  |

***Table 6 – Testoints list Table***

### Interfaces vectors

### Reference models

**A picture containing sketch, diagram, design, clipart

Description automatically generated**

***Figure 3 – Reference model diagram***

### Data Flow and Operation

The data flow in the L3L4CS verification environment follows these major steps:

1.Assert the “srst\_n” signal to low for 10 us to reset the DUT.

2.Release the “srst\_n” signal (set it high) to start the DUT operation.

3. The Axi\_wrapper agent generates Ethernet packets (eth\_item) according the test plan.

The axi\_wrapper agent responsible drives the eth\_item to the DUT using the AXI protocol.

4.the extract\_cs\_item component inside Axi\_wrapper agents generate expected cs\_item and pass it to the scoreboard via the analysis port inside cs agent.

5. the scoreboard inside the cs agent make the compression between the expected cs item to the actual cs item.

6.the scoreboard inside the axi\_wrappermake the compression between the expected eth\_item to the actual eth item.

### Pass/Fail criteria

The verification process will be considered complete and ready for signoff when:

1. All Test Cases in the Testplan have been excuted and passed successfully.
2. Code Coverage Goals have been met.
3. Functional Coverage Goals have been achieved, covering the required Scenarios and configurations.
4. Any reported Bugs have been resovled and verifed.

Test Plan

**1. Packets without VLAN IPV4 UDP TEST:**

Create random numbers of eth\_itemes with diff size payloads (min 4 frames) with these fields:

* Eth\_item. L2. with\_vlan=0 //without vlan
* Eth\_item. L2.ethertype=0800 // ipv4
* Eth\_item. L4=8’h11 //udp
* Eth\_item. L3\_TOTAL\_LEN [ 20: 65535]

In each packet the cs can be correct or wrong in L3 and L4.

**Expected Result**:

* cs\_l3\_o and cs\_l4\_o should be '01' or ‘10’, indicating correct CS or wrong CS.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

EX :

|  |  |  |  |
| --- | --- | --- | --- |
| Layer Number | Layer 2 | | |
| Title | MAC destination | MAC source | Ethertype |
| Number of Octets | 6 octets | 6 octets | 2 octets |
| Subtitle |  |  |  |
| Number of Octets |  |  | 0x0800 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 3 | | | | | | | | | |
| Title | IPv4 Header | | | | | | | | | |
| Number of Octets | 20 Octets | | | | | | | | | |
| Subtitle | Version+IHL | DSCP+ECN | Total Length | Identification | Flags+ Offset | Time to Live | Protocol | Header Checksum | Source IP | Destination IP |
| Number of Octets | 1 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets | 1 Octet | 1 Octet | 2 Octets | 4 Octets | 4 Octets |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 4 | | | | | Layer 2 |
| Title | UDP Header | | | | Payload | Frame check sequence (32bit CRC) |
| Number of Octets | 8 Octets | | | | 1-65,508 Octets | 4 octets |
| Subtitle | Source Port | Destination Port | Length | Checksum |  |  |
| Number of Octets | 2 Octets | 2 Octets | 2 Octets | 2 Octets |  |  |

**2. Packets with VLAN IPV4 UDP TEST:**

Create random numbers of eth\_itemes with diff size payloads (min 4 frames) with these fields:

* Eth\_item. L2.with\_vlan=1 //with vlan
* Eth\_item. L2.ethertype=0800 // ipv4
* Eth\_item. L4=8’h11 //udp
* Eth\_item. L3\_TOTAL\_LEN [ 20: 65535]

In each packet the cs can be correct or wrong in L3 and L4.

**Expected Result**:

* cs\_l3\_o and cs\_l4\_o should be '01' or ‘10’, indicating correct CS or wrong CS.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer Number |  | Layer 2 | | |
| Title | MAC destination | MAC source | VLAN | Ethertype |
| Number of Octets | 6 octets | 6 octets | 4 octets | 2 octets |
| Subtitle |  |  |  |  |
| Number of Octets |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 3 | | | | | | | | | |
| Title | IPv4 Header | | | | | | | | | |
| Number of Octets | 20 Octets | | | | | | | | | |
| Subtitle | Version+IHL | DSCP+ECN | Total Length | Identification | Flags+ Offset | Time to Live | Protocol | Header Checksum | Source IP | Destination IP |
| Number of Octets | 1 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets | 1 Octet | 1 Octet | 2 Octets | 4 Octets | 4 Octets |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 4 | | | | | Layer 2 |
| Title | UDP Header | | | | Payload | Frame check sequence (32bit CRC) |
| Number of Octets | 8 Octets | | | | 1-65,508 Octets | 4 octets |
| Subtitle | Source Port | Destination Port | Length | Checksum |  |  |
| Number of Octets | 2 Octets | 2 Octets | 2 Octets | 2 Octets |  |  |

**3. Packets without VLAN IPV4 TCP TEST:**

Create random numbers of eth\_itemes with diff size payloads (min 4 frames) with these fields:

* Eth\_item. L2. with\_vlan=0 //without vlan
* Eth\_item. L2.ethertype=0800 // ipv4
* Eth\_item. L4=8’h06 //udp
* Eth\_item. L3\_TOTAL\_LEN [ 20: 65535]

In each packet the cs can be correct or wrong in L3 and L4.

**Expected Result**:

* cs\_l3\_o and cs\_l4\_o should be '01' or ‘10’, indicating correct CS or wrong CS.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

**Packet without VLAN IPV4 TCP:**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer Number | Layer 2 | | |
| Title | MAC destination | MAC source | Ethertype |
| Number of Octets | 6 octets | 6 octets | 2 octets |
| Subtitle |  |  |  |
| Number of Octets |  |  | 0x0800 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 3 | | | | | | | | | |
| Title | IPv4 Header | | | | | | | | | |
| Number of Octets | 20 Octets | | | | | | | | | |
| Subtitle | Version+IHL | DSCP+ECN | Total Length | Identification | Flags+ Offset | Time to Live | Protocol | Header Checksum | Source IP | Destination IP |
| Number of Octets | 1 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets | 1 Octet | 1 Octet | 2 Octets | 4 Octets | 4 Octets |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | |  | |  | | |  | | |  | Layer 4 | | | | | | | | Layer2 | |
| Title | |  | | TCP header | | |  | | | | | | | | | | Payload | | Frame check sequence (32bit CRC) | |
| Number of Octets | | 20 Octets | | | | | | | | | | | | | | | 1-65,508 Octets | | 4 octets | |
| Subtitle | | Source\_port | | | dest\_port | Sequence number | | Acknowledge number | | | Data offset | reserved | flags | window | checksum | Urgent pointer |  | |  | |
| Number of Octets | | 2 Octet | | | 2 Octet | 4 Octets | | 4 Octets | | | 0.5 Octets | 0.5 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets |  | |  | |

**4.Packets with VLAN IPV4 TCP TEST:**

Create random numbers of eth\_itemes with diff size payloads (min 4 frames) with these fields:

* Eth\_item. L2.with\_vlan=1 //with vlan
* Eth\_item. L2.ethertype=0800 // ipv4
* Eth\_item. L4=8’h06 //udp
* Eth\_item. L3\_TOTAL\_LEN [ 20: 65535]

In each packet the cs can be correct or wrong in L3 and L4.

**Expected Result**:

* cs\_l3\_o and cs\_l4\_o should be '01' or ‘10’, indicating correct CS or wrong CS.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer Number |  | Layer 2 | | |
| Title | MAC destination | MAC source | VLAN | Ethertype |
| Number of Octets | 6 octets | 6 octets | 4 octets | 2 octets |
| Subtitle |  |  |  |  |
| Number of Octets |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | Layer 3 | | | | | | | | | |
| Title | IPv4 Header | | | | | | | | | |
| Number of Octets | 20 Octets | | | | | | | | | |
| Subtitle | Version+IHL | DSCP+ECN | Total Length | Identification | Flags+ Offset | Time to Live | Protocol | Header Checksum | Source IP | Destination IP |
| Number of Octets | 1 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets | 1 Octet | 1 Octet | 2 Octets | 4 Octets | 4 Octets |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer Number | |  | |  | | |  | | |  | Layer 4 | | | | | | | | Layer2 | |
| Title | |  | | TCP header | | |  | | | | | | | | | | Payload | | Frame check sequence (32bit CRC) | |
| Number of Octets | | 20 Octets | | | | | | | | | | | | | | | 1-65,508 Octets | | 4 octets | |
| Subtitle | | Source\_port | | | dest\_port | Sequence number | | Acknowledge number | | | Data offset | reserved | flags | window | checksum | Urgent pointer |  | |  | |
| Number of Octets | | 2 Octet | | | 2 Octet | 4 Octets | | 4 Octets | | | 0.5 Octets | 0.5 Octet | 1 Octet | 2 Octets | 2 Octets | 2 Octets |  | |  | |

**5. Length\_error TEST:**

Generate an Ethernet packet with valid L3 and L4 headers and their corresponding correct checksums. Introduce an error in the total length of the packet.

*for ex: L3\_total\_len lower 20 or L3\_ IHL<5.*

**Expected Result:**

* len\_error\_o should be '1', indicating a length error.
* L2\_error\_o should be '0', indicating no L2 error.

**6. invalid\_L2\_FCS TEST:**

Generate an Ethernet packet with valid L3 and L4 headers and their corresponding correct checksums. Introduce an error in the L2 FCS after the L4 payload.

**Expected Result:**

* len\_error\_o should be '0', indicating no length error.
* L2\_error\_o should be '1', indicating an error in the L2 FCS after the L4 payload.

**7. NO\_Packet\_received TEST:**

Do not generate any Ethernet packet.

**Expected Result:**

* cs\_l3\_o and cs\_l4\_o should be '11', indicating that no packet was received.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.
* Cs valid should be in zero.

**8. Non-IP L3\_Header TEST:**

Generate an Ethernet packet with a non-IP L3 header (e.g., ARP, PTP) and with correct cs in L4.

**Expected Result:**

* cs\_l3\_o should be '00', indicating a non-IP L3 header.
* cs\_l4\_o should be '00', indicating a correct checksum in L4.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

**9. Non-TCP/UDP L4\_Header TEST:**

Generate an Ethernet packet with a non-TCP/UDP L4 header (e.g.,) and with correct cs in L3.

**Expected Result:**

* cs\_l3\_o should be '01', indicating a correct checksum in L3.
* cs\_l4\_o should be '00', indicating a non-TCP/UDP L4 header.
* len\_error\_o and L2\_error\_o should be '0', indicating no length error or L2 error.

**RANDOM TESTS:**

**1. Continuous\_Packet\_Stream TEST:**

Generate a continuous stream of Ethernet packets with different combinations of L3 and L4 headers and min supported rate.

**Expected Result:**

* Verify that the DUT can handle the continuous stream without errors and report any issues correctly.

**2. random\_payload\_len TEST:**

Generate Ethernet packets with random payload lengths (within the validrange) and correct checksums in L3 and L4 headers.

**Expected Result:**

* Verify that the DUT can handle packets with various payload lengths without errors.

**CORNER TESTS:**

**1. Boundary values TEST:**

Generate Ethernet packets with checksum values at the maximum and minimum allowable range.

**Expected Result:**

* Ensure that the DUT handles boundary values and detects errors.

**2. handling Malformed Packets TEST:**

Verify the DUT's response when processing malformed packets with invalid or corrupted headers. Generate Ethernet packets with invalid or corrupted L3 and L4 headers.

**Expected Result:**

* Verify that the DUT handles malformed packets without any assertion or critical errors. It should produce appropriate output values for cs\_l3\_o, cs\_l4\_o, len\_error\_o, and L2\_error\_o.

**3. Invalid\_len\_& L2 & Incorrect CS Packets TEST:**

Verify the DUT's response when the packet length is incorrect, L2 is incorrect, and there is also an incorrect checksum in L3, L4.

Generate an Ethernet packet with an incorrect total length and incorrect checksums in both L3 and L4 headers.

**Expected Result:**

* cs\_l3\_o and cs\_l4\_o should be '10', indicating incorrect checksums in both headers.
* len\_error\_o and L2\_error\_o should be '1', indicating length error and L2 error.

**4. Jumbo frame TEST:**

Generate Ethernet packets with max payload length for creating jumbo frame with different combinations of L3 and L4 CS fileds.

**Expected Result:**

* Ensure that the DUT handles the frames and detects cs results.

**5. srst\_during\_frame TEST:**

Generate Ethernet packets and toggling the srst during the hdr transmission of in the transmission.

**Expected Result:**

Ensure that the DUT detects srst and react correctly.

Coverage

Code Coverage

Functional Coverage

Track functional coverage to ensure that the testcases cover different functional scenarios and configurations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **cover group** | **Cover Point** | **Width** | **cover bins** | **status** |
| Cvg\_cs\_outputs | Cvr\_l3\_cs | 2 bits | l3\_cs\_o= {01,10,00,11}; |  |
| Cvg\_cs\_outputs | Cvr\_l4\_cs | 2 bits | L4\_cs\_o= {01,10,00,11}; |  |
| Cvg\_cs\_outputs | Cvr\_len\_er\_o | bit | len\_er\_o {0,1}; |  |
| Cvg\_cs\_outputs | Cvr\_L2\_er\_o | bit | L2\_er\_o {0,1}; |  |
| Cvg\_eth | Cvr\_eth\_L3\_length | 2 bytes | L3\_length |  |
| Cvg\_eth | Cvr\_eth\_L3\_IHL | 4bits | L3\_IHL {[0:5} [5:10] |  |
| Cvg\_eth | Cvr\_eth\_L3\_protocol | byte | L3\_protocol {8’h11,8’h06}  {arp, ptp,0…} |  |
| Cvg\_eth | Cvr\_eth\_L2\_FCS | 4 bytes | L2.FCS= |  |
| Cvg\_eth | Cvr\_eth\_L4\_length | 2 bytes | L4.length=] |  |