Alveo Data Center Accelerator Card Test

User Guide

UG1361 (v5.0) February 11, 2021

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
02/11/2021 Version 5.0	
General Updates	Clarifying edits in various sections.
Display Modes	Introduces a new user interface called dynamic display mode.
Memory Test Case Description	Support for asymmetric memory.
GT MAC Test Case Description	Support for switch at 10GbE and 25GbE lane rates.
Test JSON File Structure	New file structure allowing independent control of each element under test.
10/28/2020 Version 4.0	
General Updates	Clarifying edits in various sections.
Alveo Data Center Accelerator Card Compatibility	Removed table and linked to answer record.
7/15/2020 Version 4.0	
General Updates	Added support for u50_3x4 and u50lv_3x4
3/11/2020 Version 4.0	
General Updates	• Standardization and simplification of test json: memory and test environment • New message display (use of ID number) • New installation procedure
3/02/2020 Version 3.3.0	
General Updates	• Add hardware watchdog support for all compute units • Software enhancement: json parsing, simplification of memory test configuration • Minor bug fixes
General Updates	Editorial/Style updates
11/06/2019 Version 3.2.1	
Power Test Case Description	Updates to Power Test Case
General Updates	Editorial/Style updates
10/25/2019 Version 3.2.1	
Chapter 5: Platform Definition	Added Shell Definition json file concept and usage information.
Chapter 4: Test Case Description	Updated test case information for clarity.
10/18/2019 Version 3.2	
Initial Xilinx release.	N/A

Table of Contents

Overview

The Alveo™ card validation test solution application can be used to validate that the Alveo card hardware is operating correctly within the host server environment. The application monitors system health and validates the functionality of the essential hardware and software components of the platform.

The Alveo card validation test solution is also referred to as `xbtest`.

`xbtest` can be configured to run the following tests:

- Host can communicate with:
  - **On-board memories:** DMA
  - **Compute units (CUs):** Validate the CUs
- Dissipate a programmable amount of power.
- Check the GT transceivers at 10GbE and 25GbE lane rates.
- Verify that CUs can communicate at the required rate with:
  - **On-board memories:** For example, DDR or HBM.
  - **Host memory:** If the slave bridge feature is available in your card (refer to the documentation for your card).

IMPORTANT! *Prior to using `xbtest`, your Alveo™ card hardware and software must be installed (refer to the documentation for your card).*

What's New

`xbtest` v5 major updates compared to v4 are:

- New user interface.
- Support for asymmetric memory.
- Support for network interface. `xbtest` GT MAC CU can be connected to the switch.
- New test `JSON` file structure allowing independent control of each element under test.

Alveo Data Center Accelerator Card Compatibility

`xbtest` supports the Alveo cards with compatible Xilinx® Runtime (XRT) and target platforms. The specific Alveo cards and deployment platforms compatible with `xbtest` are listed in *Alveo - xbtest Version Platform Support* (AR#75656).

For Alveo card documentation, refer to the documentation for your card.

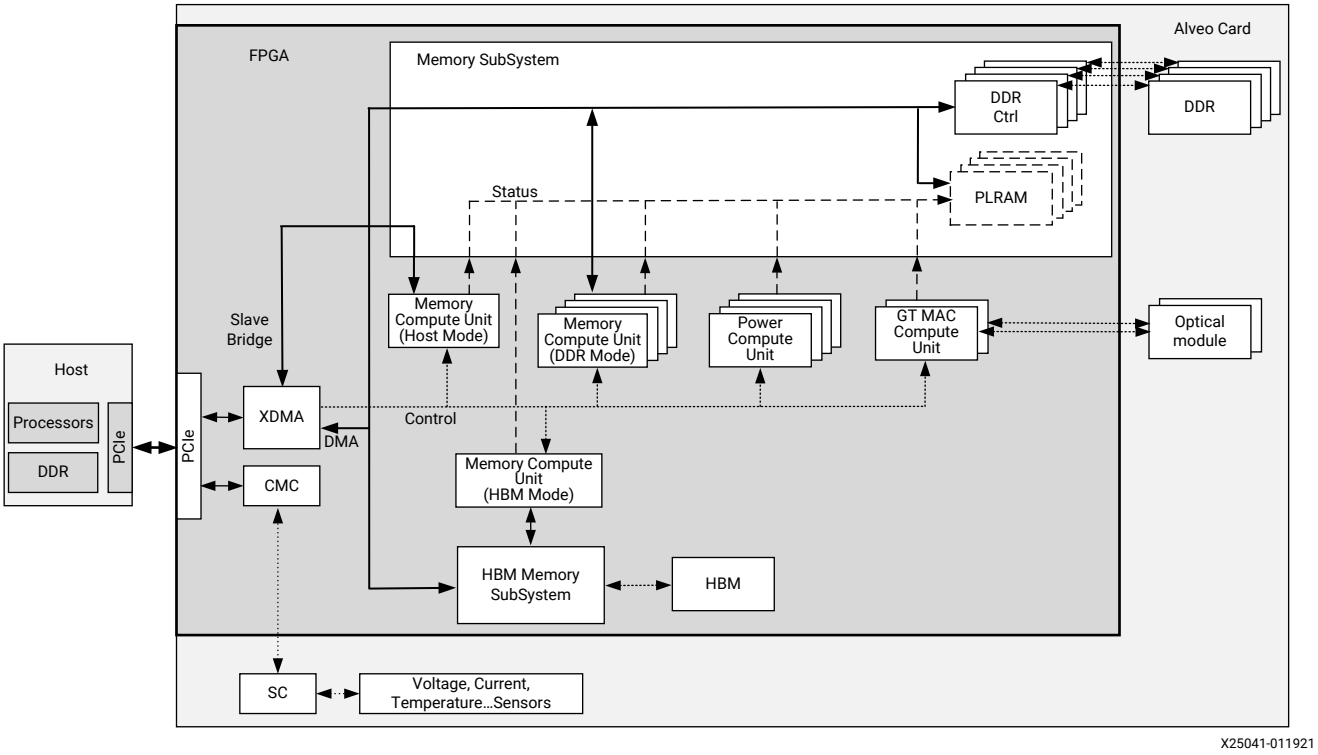
Architecture Overview

The solution comprises:

- **Application software:** Runs on the host
- **`xclbin`:** Downloaded to the card and can contain power, memory, and GT MAC compute units.

The following block diagram shows an Alveo card with an example of deployment platform and compute units (part of the `xclbin`). Per CU type (power, memory, and GT MAC), the presence and the quantity of available CUs depends on card and platform capabilities (refer to the respective documentation).

Figure 1: **Alveo Card Block Diagram**



Application software and `xclbin` are used in conjunction to test on-board memories (for example DDR, HBM) and GTs while the card is consuming a programmable amount of power. Host memory can also be tested separately if slave bridge feature is available.

Hardware Overview

The following CU types test different areas of the card hardware:

- **Power:** Throttles the clock of flip-flops, DSPs, block RAMs, and UltraRAMs, present in the logic of the `xclbin`, to control their power consumption.
- **Memory:** Measures the read and write bandwidths while performing a data integrity check on the data transmitted and received. The read and write latencies are also measured.
- **GT MAC:** Checks GT transceivers of the Alveo card at 10 Gigabit Ethernet (10 GbE) and 25 Gigabit Ethernet (25 GbE) lane rates.

The following hardware safety mechanisms are present in CUs:

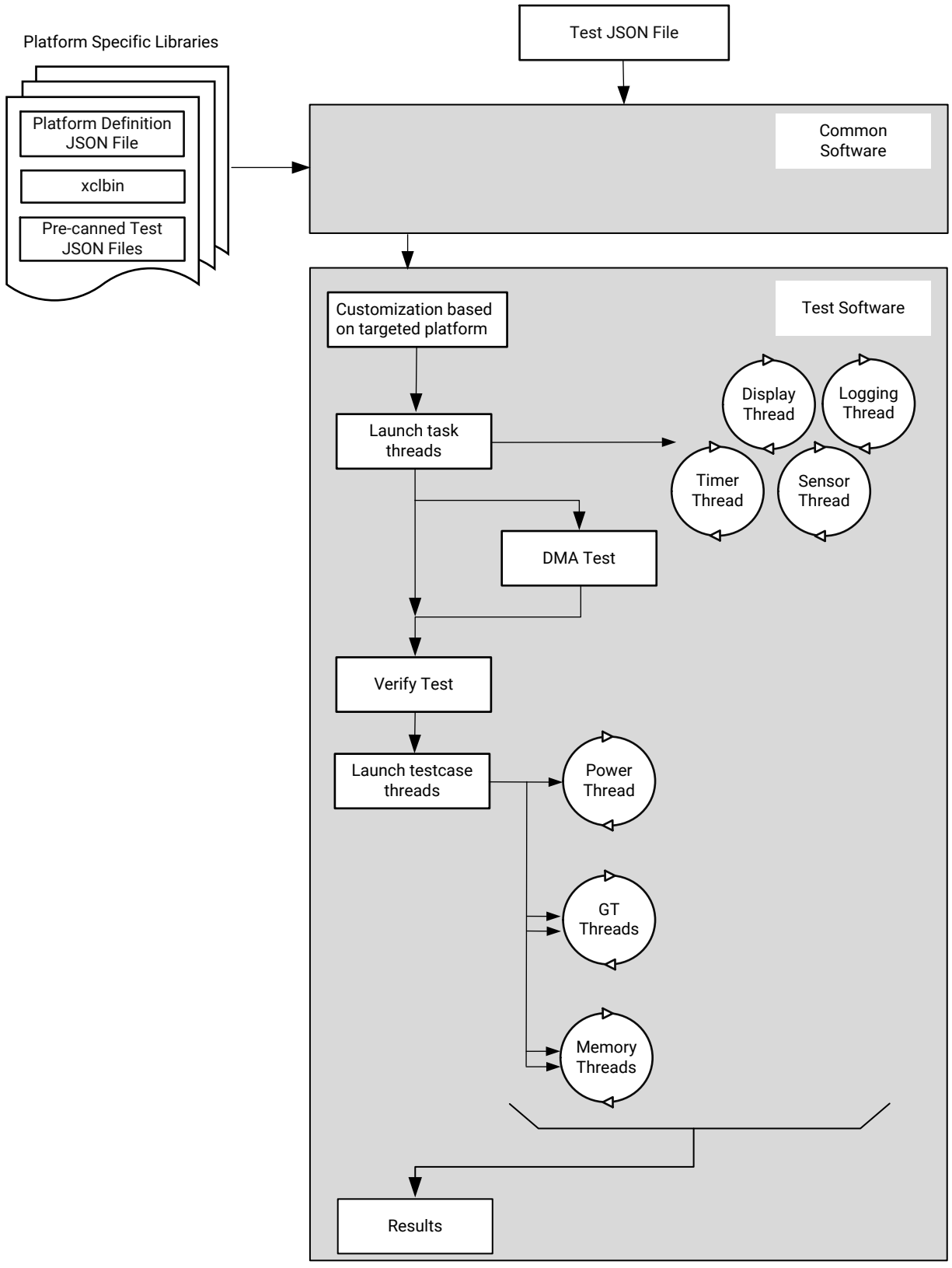
- **Watchdog:** Stops the CU activity after 15 seconds in the case of the application software failing to perform the watchdog reset.

- **Status Register:** Detects and prevents multiple instances of application software trying to control/access the same Alveo card.

Software Overview

The `xbtest` is common for all supported platforms. Multiple Alveo cards can be tested simultaneously by running additional instances of the application. The application software automatically detects the number and type of compute units (CUs) present in the `xclbin` (power, memory, or GT MAC).

Figure 2: **Software Model**



To enable simultaneous support for multiple versions of the software, the `xbtest` application software is split into two different applications:

- **`Common` software:** In charge of dispatching the test:
  - It selects, configures, and launches the `Test` software based on the selected deployment platform under test.
  - `xbtest` must be run from the `Common` software.
- **`Test` software:** The actual software which performs the tests:
  - Each test case (power, memory and GT MAC) runs independently.
  - The DMA test case runs prior to all test cases.
  - The `Test` software is only run by the `Common` software.

The application software supports heterogeneous Alveo card installation by the addition of `xbtest` platform specific libraries.

IMPORTANT! *The platform specific library files should not be edited or modified. See Install xbtest section.*

The `Test` software uses two `JSON` configuration files:

- **Test `JSON` file:** Contains the descriptions of the tests to perform. User can provide his own test or use one of the pre-canned tests available in the platform specific library (see Command Line Options).
- **Platform definition `JSON` file:** Specifies the platform specific characteristics and limits of `xbtest`. This file is delivered in the platform specific library and is automatically selected.

`xbtest` performs the following steps to launch a test:

1. Checks the command line options for validity and unsupported combinations.
2. Checks integrity of all platform specific libraries currently installed.
3. Selects from the targeted `<Platform>` (`-d` option). See Command Line Options.
   - Platform definition `JSON` file.
   - `xclbin` file.
   - Supported `Test` software version.
4. Launches `Test` software on the selected card with:
   - Pre-canned test (`-c` option) or the provided test `JSON` file (`-j` option). See Command Line Options.
   - Platform definition `JSON` file and `xclbin` selected previously. See Chapter 5: Platform Definition.

The `Test` software also manages the watchdog present in the different CUs and checks that the Alveo card is not in use by another instance of the application.

Installation

xbtest is delivered as various archive files that contain the installation packages (RPM and DEB). For each deployment platform, the two archives files provided are:

- **Software archive:** Contains two software applications:
  - **Common software:** Select, configure, and launch the Test software based on the platform under test.
  - **Test software:** Actual software running the tests.
- **Platform library archive:** Contains platform-specific files.

For links to xbtest software and platform library archive files, go to Alveo Data Center Accelerator Card Compatibility.

Dependencies

The application software depends on the json-glib-1.0 and ncurses packages. These two dependencies are part of the xbtest RPM or DEB packages, and they will be resolved by the OS package manager (for example, yum or apt).

IMPORTANT! *Although the OS package manager will resolve the dependencies between the xbtest packages, xbtest requires a correctly installed and operating card. For instructions on how to install Xilinx® Runtime (XRT) and the deployment platform, refer to the documentation for your card.*

CAUTION! *xbtest v4 and xbtest v5 cannot be installed simultaneously. Before you can install xbtest v5, you must remove xbtest v4.*

Install xbtest

xbtest is delivered as various archive files containing the RPM or DEB packages to be installed, which are available in Alveo Data Center Accelerator Card Compatibility. This section explains how to identify and install the xbtest RPM or DEB packages using your OS package manager.

The high-level installation steps are first listed, and then specific details are outlined in the following section:

1. Get `xbtest` software and platform library archive files compatible with your setup.
2. Install `xbtest` software RPM or DEB packages compatible with your setup.

   ***Note*:** The software packages are only installed once for all target platforms in the following order:

   1. `Common` software named `xilinx-xbtest-common`.
   2. `Test` software named `xilinx-xbtest-sw-5`.
3. Install the platform library RPM or DEB package compatible with your setup.

   ***Note*:** You can install multiple platform library packages (based on what is available on your system).

Step 1: Get Compatible Archives

Identify your deployment platform package name and version. For example, if you are targeting a U50 LV card, run the command given in the following table based on your OS:

Table 1: **Identify your deployment platform**

OS	Command	Output
Red Hat / CentOS	`$ yum list | grep 'xilinx-u50lv*'`	`xilinx-u50lv-gen3x4-xdma-base.noarch 2-2902115 installed`
Ubuntu	`$ apt list | grep 'xilinx-u50lv*'`	`xilinx-u50lv-gen3x4-xdma-base/now 2-2902115 all [installed,local]`

From the output of these commands, the deployment package can be identified:

- Name is xilinx-u50lv-gen3x4-xdma-base
- Version is 2

Finally, download and extract `xbtest` software and platform library archive files compatible with this deployment package (see Alveo Data Center Accelerator Card Compatibility section).

Step 2: Install Software Packages

`xbtest` software RPM and DEB packages depend on OS version and system architecture. Identify compatible `xbtest` package `<architecture>` which is returned by the command given in the following table:

Table 2: **Identify `xbtest` Package Architecture**

OS	Command	Output
Red Hat / CentOS	`$ uname -p`	`x86_64`
Ubuntu	`$ dpkg --print-architecture`	`amd64`

Identify compatible `xbtest` package `<OS release>` depending on your distribution. The following table gives two examples of `xbtest` package `<OS release>`:

Table 3: **Identify `xbtest` Package OS Release**

OS	`xbtest` package OS release
Red Hat / CentOS 8.x	8.1.1911
Ubuntu 18.04	18.04

Note: You can check the OS release of your distribution by running the command:

```
$ lsb_release -rs
```

Based on `<architecture>` and `<OS release>`, install first the `Common` software and then the `Test` software. The following table gives two examples of commands:

Table 4: **Install `xbtest` Software Packages**

Distribution	Architecture	Command
Red Hat / CentOS 8.x	x86_64	`$ sudo yum install xbtest-common-1.1-1.x86_64_8.1.1911.rpm` `$ sudo yum install xbtest-sw-5-0-1.x86_64_8.1.1911.rpm`
Ubuntu 18.04	amd64	`$ sudo apt install xbtest-common_1.1-1_amd64_18.04.deb` `$ sudo apt install xbtest-sw-5_0-1_amd64_18.04.deb`

Note: Software packages are named with the suffix `<architecture>_<OS release>.<extension>`

Step 3: Install Platform Specific Library Package

Platform library packages do not depend on OS release and architecture. Install platform library DEB or RPM package. The following table gives the installation command for the deployment platform identified in the first step:

Table 5: **Install `xbtest` Platform Library Packages**

Distribution	Command
Red Hat / CentOS	`$ sudo yum install xbtest-xilinx-u50lv-gen3x4-xdma-base-2-5.0-3031405.noarch.rpm`
Ubuntu	`$ sudo apt install xbtest-xilinx-u50lv-gen3x4-xdma-base-2_5.0-3031405_all.deb`

***Note*:** Platform packages are named with the suffixes: `_all.deb` or `.noarch.rpm`

Installed Content

After `xbtest` software packages installation, two executable files are present:

- **`Common` software:** `/opt/xilinx/xbtest/bin/xbtest`
- **`Test` software:** `/opt/xilinx/xbtest/5/bin/xbtest`

IMPORTANT! *To ensure that the correct executable is used, the `xbtest` environment must be set up before running `xbtest`. Refer to Set Up `xbtest`, which also describes how to run a quick check of the installation.*

After the platform library installation, the following content is present:

- One hardware application file (`xbtest_stress.xclbin`) containing specific compute units (GT MAC, memory and power).
- A set of pre-canned test `JSON` files, for example, `verify`, `dma`, `memory`, `gt_mac`, `power` and `stress`.
- Platform definition (`xbtest_pfm_def.json`) and configuration `JSON` files.

CAUTION! *These files are specific to each deployment platform. Do not edit them.*

Various platform libraries can be installed and used simultaneously. The following shows an example of some installed files and directory structure for two platforms, `<Platform A>` and `<Platform B>`.

```
/opt/xilinx/xbtest/
├── bin/
│   └── xbtest*
├── setup.csh
├── setup.sh
├── 5/
│   ├── bin/
│   │   └── xbtest*
└── lib/
    ├── Platform A/
    │   ├── config.json
    │   ├── xbtest_pfm_def.json
    │   ├── xclbin/
    │   │   └── xbtest_stress.xclbin
    │   └── test/
    │       └── verify, dma, gt_mac, switch, memory, power, stress.json
    └── Platform B/
        ├── config.json
        ├── xbtest_pfm_def.json
        ├── xclbin/
        │   └── xbtest_stress.xclbin
        └── test/
            └── verify, dma, gt_mac, switch, memory_host, memory, power,
stress.json
```

Where:

```
*: identifies an executable file
/: identifies a folder
```

***Note*:** The folder `test/` contains the pre-canned test `JSON` files. The quantity and content of the pre-canned test `JSON` files depends on the deployment platform.

Removal

To fully remove `xbtest` run the command given in the following table based on your OS:

Table 6: **`xbtest` Packages Removal**

Distribution	Command
Red Hat / CentOS	`$ sudo yum remove 'xbtest*'`
Ubuntu	`$ sudo apt remove 'xbtest*'`

CAUTION! *This will remove* `xbtest` *for all platforms.*

Usage

This chapter describes the basic commands to:

- Set up `xbtest` and check if its installation was correct.
- Launch a pre-canned test.
- Launch `xbtest` while targeting one or more deployment platforms.

Before using `xbtest`, your card should be installed and operating correctly, including XRT. For installation instructions, see the following:

- Xilinx Runtime (XRT) documentation
- Documentation for your card

IMPORTANT! *Ensure you run `xbtest` from a folder where you have write permission.*

Set Up `xbtest`

Use the following commands to set up `xbtest`:

Table 7: **Set Up `xbtest`**

Shell	Command
csh	`$ source /opt/xilinx/xbtest/setup.csh`
bash	`$ source /opt/xilinx/xbtest/setup.sh`

This script adds the location of the `xbtest` executable (`/opt/xilinx/xbtest/bin/`) to the `PATH` environment variable.

If `xbtest` is installed and set up correctly, the following command displays the `xbtest` help instructions.

```
$ xbtest -h
```

Command Line Options

The following table details the available `xbtest` command line options:

Table 8: **`xbtest` Command Line Options**

Option	Usage	Description

`-v`[1]	• `Common` software:
	`$ xbtest -v`
	• `Test` software:
	`$ xbtest -d <BDF> -v`
		Display the version. There are two version levels:
		• **`Common` software:** Display `xbtest` version.
		• **`Test` software:** Display `xbtest` version for the selected platform.

`-h`[1]	• `Common` software:
	`$ xbtest -h`
	• `Test` software:
	`$ xbtest -d <BDF> -h`
		Display the help message and command line options available. There are two levels of messages:
		• **`Common` software:** Display `xbtest` command line options and the list of all installed platforms supported by `xbtest` with their respective pre-canned tests.
		• **`Test` software:** Display the command line options for the specific version `xbtest` used by the selected platform.

`-d`	`$ xbtest -d <BDF> -j <PathTo/mytests.json>`	Select the Alveo card identified by its `<BDF>`. This option is mandatory to run a test (see Identifying a Deployment Platform).
`-g`[1]	`$ xbtest -d <BDF> -g <test case>`	Display a guide for the given `<test case>`. This guide includes `JSON` members description, test sequence definition with examples and basic test `JSON` file examples. To obtain the list of test cases available use the command `$ xbtest -d <BDF> -h` ***Note*:** This option must be used with `-d` command line option.
`-j`[2]	`$ xbtest -d <BDF> -j <PathTo/mytests.json>`	Select test `JSON` file `<PathTo/mytests.json>` defining the test cases to run.
`-c`[2]	`$ xbtest -d <BDF> -c <pre-canned test>`	Select a pre-canned test `<pre-canned test>` to run. To obtain the list of pre-canned tests available and their location in the installation directory use command: `$ xbtest -h`. See Pre-Canned Tests for more information on the quantity and content of these platform specific tests.
`-l`[3]	`$ xbtest -l <PathTo/LogDir> -d <BDF> -j <PathTo/mytests.json>`	Define a logging directory `<PathTo/LogDir>` in which all output files of any test case will be stored. When not specified, logging directory is still generated with a default name. See Result Directory section for more information on the different directories and files generated by `xbtest`.

Table 8: `xbtest` **Command Line Options** *(cont'd)*

Option	Usage	Description
`-L`[3]	`$ xbtest -L -d <BDF> -j <PathTo/mytests.json>`	Disable logging directory generation in which all output files of any test case will be stored. No message displayed during the execution will be stored.
`-m`[1]	`$ xbtest -d <BDF> -m <message ID>`	Display message information for the given `<message ID>`: severity, details or resolution (when applicable).
`-F`	`$ xbtest -F -d <BDF> -j <PathTo/mytests.json>`	Enable classic display mode (similar to `xbtest` v4). When not provided, `xbtest` defaults to the dynamic display mode. See Display Modes section for more information on the messages displayed by `xbtest` in the console.

Notes:

1. Prevents any test being launched.
2. `-c` and `-j` cannot be combined.
3. `-l` and `-L` cannot be combined.

Identifying a Deployment Platform

After having installed and set up XRT (*Xilinx Runtime (XRT) Documentation*), use the following command to get the list of available platforms:

```
$ xbutil scan
```

The command will output results similar to the following example. The card index is the integer value given in the square brackets `[<index>]` followed by the card BDF `<bus>:<device>.<function>` and target deployment platform.

```
[0] 0000:d8:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=133)
[1] 0000:af:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=132)
[2] 0000:86:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=131)
[3] 0000:5e:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=130)
[4] 0000:3b:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=129)
[5] 0000:18:00.1 xilinx_u50_gen3x16_xdma_201920_3 user(inst=128)
```

In this example, for the third card, BDF is `0000:86:00.1` and index is `2`.

Targeting Multiple Deployment Platforms Simultaneously

When the system contains multiple deployment platforms, multiple instances of `xbtest` can be run simultaneously, each instance targeting a different deployment platform using the `-d` command line option (see Command Line Options).

The same test `JSON` file specified using `-j` command line option can be used by the different instances of `xbtest`. In addition, `-l` option can be used to provide other logging directory names than the default ones.

The various instances of `xbtest` can be run for example from the same console using the command line operator `&`. The following example shows how to run multiple instances of `xbtest` in one command, using the same test `JSON` file, on the deployment platforms with device BDFs `0000:d8:00.1`, `0000:af:00.1` and `0000:86:00.1`.

***Note*:** By default, `xbtest` uses the dynamic display mode which clears the console regularly. Instead, it is recommended to use the `-F` command line option, which enables the classic display mode when multiple `xbtest` are run simultaneously from the same console.

```
$ xbtest -F -j test.json -d 0000:d8:00.1 & xbtest -F -j test.json -d
0000:af:00.1 & xbtest -F -j test.json -d 0000:86:00.1
```

Display Modes

An execution of `xbtest` will typically contain the following messages:

- **Start banner:** Information about the run: start time and versions (for example, `xbtest` and XRT).
- **Messages from test cases:**
  - Start
  - Status and results
  - End
- **End banner:**
  - Information about the run: end time and versions
  - Summary of all test cases run and their results
  - Global result of `xbtest` run

`xbtest` supports two display modes, which are described in the following sections:

- **dynamic:** Default mode
- **classic:** As per previous version of `xbtest`.

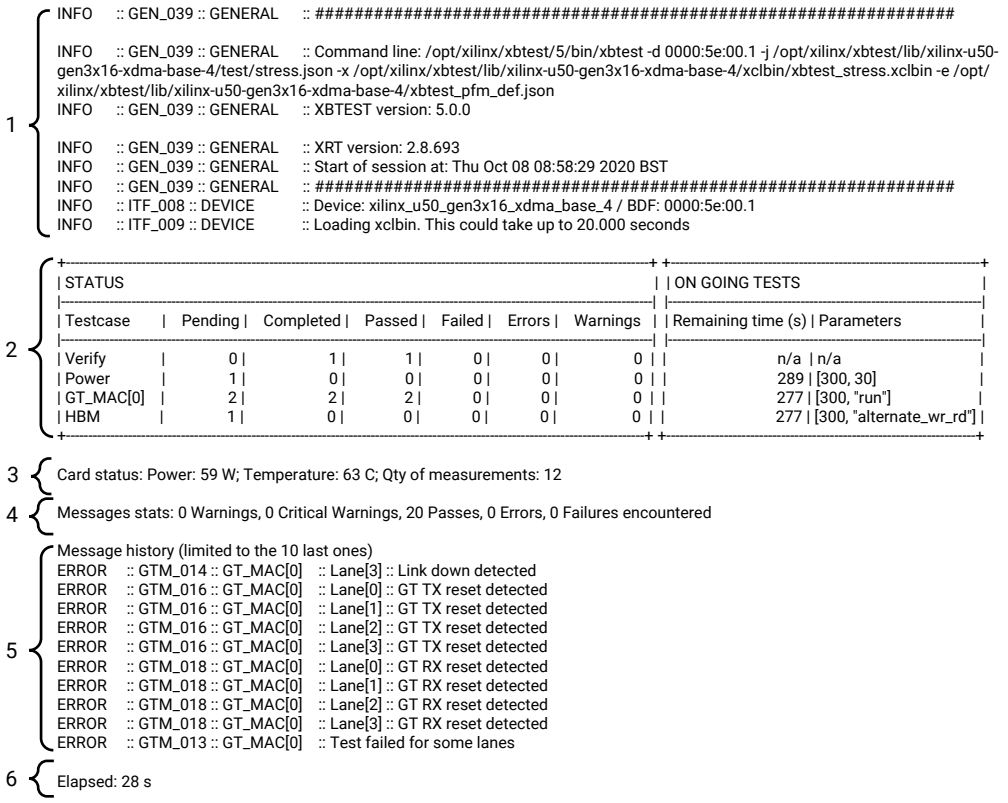
Dynamic Display Mode

By default, the console dynamically displays `xbtest` messages. Regularly, the entire console is cleared, and its content is refreshed. The content displayed in the console contains the following:

- **Messages displayed before the first test case starts:**
  - Start banner
  - Card selection
- **Test cases information:** A table presenting the following:
  - **Test cases status:** Results of previous tests, quantity of warning, pass, error, and failure messages
  - **Ongoing test information:** Duration and current test running
- **Card status:** Basic sensor information (power consumption and temperature of the card).
- **Message statistics:** Quantity of messages available for each severity, which are in `xbtest.log`.
- **Message history:** 10 last important messages (warning, error, and failure)

At the end of `xbtest` execution, the console returns to the initial state, and the last content displayed in the dynamic mode is repeated.

Figure 3: **Example of information available in dynamic display mode**



Legend:

1. Messages displayed before first test case starts
2. Test cases status and information about ongoing tests
3. Card status
4. Messages statistics
5. Messages history
6. Elapsed time

Classic Display Mode

When using `-F` command line option, all messages are displayed consecutively, and the console is never cleared (as per previous version of `xbtest`). The information displayed in this mode is the same as the information available in `summary.log`.

Understanding `xbtest` Messages

The application software outputs messages into log files. Each message is generated with the format `<severity> :: <ID> :: <headers> :: <message>`. The following table describes these message fields, separated by "::".

Table 9: **Message Fields**

Field	Example	Description

`<severity>`	`INFO`	Possible severities are:
		• **`INFO`:** General information about the progress of a test; for example, the start of a particular step of the test sequence.
		• **`STATUS`:** Message showing intermediate progress of a test; for example, measurements.
		• **`WARNING`:** Message that does not alter the progress of the test. User action might be taken or might be reserved.
		• **`CRIT_WARN`:** Message that does alter the progress or the results of the test. User action is strongly recommended.
		• **`PASS`:** Message returned in the case of successful check.
		• **`ERROR`:** Message returned in the case of failed check. This does not interrupt any test cases or tests.
		• **`FAILURE`:** The only message level which aborts all test cases/tests. For example, in cases of:
		  • SW/HW incompatibility
		  • Test `JSON` file typo or wrong settings/members
		  • User interruption: CTRL+C
		  • Other interrupt: for example, card reset

Table 9: **Message Fields** *(cont'd)*

Field	Example	Description

<ID>	GEN_039	Unique identifier for a message. The ID is formed as follows: <string>_<number>, where:
		• **<string>:** One string as a 3-letter code which identifies the type of the message:
		  • **GEN:** (general) message not specific to any test case.
		  • **ITF:** (interface) message related to device interface (for example OpenCL) and Test JSON file.
		  • **JPR:** JSON file parser.
		  • **MGT:** (management) control and safety features.
		  • **CMN:** (common) used by different test cases.
		  • **DMA:** DMA test case specific.
		  • **VER:** Verify test case specific.
		  • **MEM:** Memory test case specific.
		  • **PWR:** Power test case specific.
		  • **ETH:** GT MAC test case specific.
		• **<number>:** One number per type. This is a unique 3-digit number identifying the message.

<headers>	GENERAL	Identifier of the message originator. In most case it is composed by a single level <header>, but it can also be composed by different levels: <header> :: <2nd header> or even <header> :: <2nd header> :: <3rd header>
		• **<header>:** Originator. For example, POWER messages originate from the Power test case.
		• **<2nd header>:** This second header is used to distinguish each individual message when multiple test cases of same type can run in parallel. For example:
		  • **Memory test cases:** Multiple memory types identified by <memory type name>, for example HBM, DDR.
		  • **GT_MAC test cases:** Multiple GT lanes identified by Lane[<idx>] with <idx> the index of the lane.
		• **<3rd header>:** This third header is used in memory test case to distinguish each individual message when multiple memory of same type can run in parallel. For example:
		  • **DDR memory type:** <tag> is used, for example DDR[0], DDR[1].

Table 9: **Message Fields** *(cont'd)*

Field	Example	Description
<message>	Start of session at: <time>	Actual message content.

To get more information about a message, use `-m` command option and see messages displayed with `GEN_033` message ID. For example, running the command `$ xbtest -d 0000:d9:00.1 -m MEM_023` will display:

```
INFO :: GEN_033 :: COMMAND_LINE :: The message "MEM_023" is:
INFO :: GEN_033 :: COMMAND_LINE :: - Severity    : ERROR
INFO :: GEN_033 :: COMMAND_LINE :: - Content     : Data integrity test fail
<optional channel info>
INFO :: GEN_033 :: COMMAND_LINE :: - Details     : Global message reporting
that the data integrity was not maintained during the test. An error has
been detected in the read data
INFO :: GEN_033 :: COMMAND_LINE :: - Resolution : Check previous error for
details about which section of the test failed
```

This presents the following information:

- **Severity:** Level of the message.
- **Content:** A generic version of the message content. As the same message can be used by multiple tests, or multiple times by the same test, when described in the console, some sections will be kept generic (usage of "<>" to describe their content). During a test, these sections will be replaced by actual value related to the test.
- **Details:** Contains more information about the message, for example context, reason.
- **Resolution:** Possible origin or solution to the message. This is only applicable for the message with the following severity levels: `WARNING`, `CRIT_WARN`, `ERROR` and `FAILURE`.

Result Directory

With any display that is used, `xbtest` stores all test case specific result files into one directory, in which `xbtest` also stores all messages into two log files:

- **`xbtest.log`:** Contains all messages and results.
- **`summary.log`:** Only the most important messages and results.

The results directory can be specified or disabled (see Command Line Options). By default, each `xbtest` run creates its own result directory `./xbtest_logs/<DATE>_<TIME>_<BDF>` in the current working directory, where the sub-directory is named with the following, separated with an underscore "_":

- **<date>:** Start of `xbtest` session date (year, month and day, separated with a dash "-").
- **<time>:** Start of `xbtest` session time (hour, minute and second, separated with a dash "-").
- **<BDF>:** Card BDF (see Identifying a Deployment Platform).

For example, the default logging directory can be: `./xbtest_logs/2020-10-13_08-46-32_0000-5e-00-1`.

Host Memory (Slave Bridge) Set Up

Slave bridge test requires the following set up procedure.

1. Before loading `xbtest xclbin`, you must allocate the host memory with the following command.

   ```
   $ sudo xbutil host_mem --enable --size <size> -d <BDF>
   ```

   Where:

   - `<BDF>` is the card BDF (see Identifying a Deployment Platform)
   - `<size>` is the size of the host memory (for example, `1G`

   Note: For host memory size requirement and enabling hugepage, refer to XRT.

2. Force a reload of `xbtest xclbin` and validate your card with the following command:

   ```
   $ xbutil validate -d <BDF>
   ```

   Where:

   - `<BDF>` is the card BDF (see Identifying a Deployment Platform)

3. Start `xbtest` test:

   ```
   $ xbtest -d <BDF> -j <PathTo/mytests.json>
   ```

Note: Steps 1 and 2 must be done *after* any server reboot, but they do not need to be redone in between each `xbtest` test.

Pre-Canned Tests

`xbtest` comes with a set of pre-canned tests which use one or more of the available test cases (verify, DMA, power, memory, or GT MAC). The pre-canned tests can be used as templates to create your own tests.

The exact list of available pre-canned tests is specific to each platform. The `test_sequence` used by these various tests is displayed when run and depends on the targeted platform. Use `$ xbtest -h` to list and locate the pre-canned test files (see Command Line Options). Any pre-canned test requires `xbtest` to be set up correctly (see Set Up `xbtest`).

IMPORTANT! *For each platform, all pre-canned tests have been validated in DELL R740 server with its fans spinning at 100% in a 25°C room.*

Verify pre-canned test

Checks the integrity of the `xclbin` (see Verify Test Case).

DMA pre-canned test

The DMA pre-canned test performs a basic DMA test case of all memories available on the card (for example, DDR and/or HBM). For more information about host ↔ memory test capabilities, see DMA Test Case Description.

In this test, the DMA bandwidths are only reported as information and are not checked against any thresholds. The following are two examples of how a pre-canned `test_sequence` might look. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
test_sequence: [[5,"HBM"], [5, "DDR"]]
test_sequence: [[5,"PL_DDR"], [5, "PS_DDR"]]
```

Memory pre-canned test

The memory pre-canned test performs a basic memory test case of all memories available on the card (for example DDR or HBM). See Memory Test Case Description for more information about CU ↔ memory test capabilities. Here is an example of how a pre-canned `test_sequence` may look like. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
"test_sequence": [
    [30, "alternate_wr_rd"],
    [30, "only_wr"],
    [30, "only_rd"],
    [30, "simultaneous_wr_rd"]
]
```

memory_host pre-canned test (slave bridge)

The host memory must be enabled before running the memory_host pre-canned test (see Host Memory (Slave Bridge) Set Up).

The memory_host pre-canned test performs a basic memory test case of the allocated host memory. See Memory Test Case Description for more information about CU ↔ memory test capabilities. Here is an example of how a pre-canned `test_sequence` may look like. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
"test_sequence": [
    [30, "alternate_wr_rd"],
    [30, "only_wr"],
    [30, "only_rd"],
    [30, "simultaneous_wr_rd"]
]
```

***Note*:** In the memory_host pre-canned test, write and read bandwidths are not compared against any thresholds. Bandwidths are only reported as information.

Power Pre-Canned Test

The power pre-canned test performs a basic power test case (see Power Test Case Description). Here is an example of how a pre-canned `test_sequence` may look like. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
"test_sequence": [ [100, 10], [100, 20], [100, 30], [100, 40] ]
```

GT Pre-Canned Tests

Each GT pre-canned test requires a different set up (see GT Test Set Up).

gt_mac Pre-Canned Test
The gt_mac pre-canned test uses the QSFP passive electrical loopback module.

Here is an example of how a pre-canned `test_sequence` may look like. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
"test_sequence" : [
    [1,  "conf_25gbe_rs_fec"],
    [1,  "clear_status"],
    [10, "run"],
    [1,  "check_status"],

    [1,  "conf_10gbe_c74_fec"],
    [1,  "clear_status"],
    [10, "run"],
    [1,  "check_status"]
]
```

***Note*:** The usage of QSFP passive electrical loopback module allows 10/25GbE rate update during the test.

switch Pre-Canned Tests
The switch pre-canned tests require to be connected to a 10GbE or 25GbE switch port. See GT MAC Test Case Description for supported test sequence, lane mapping, switch and cables.

These tests perform basic tests at 10GbE or 25GbE rate while the GT is connected to a 10 or 25GbE switch port. The traffic is looped back by the switch. As targeted cards can have 1 or 2 GTs and as each GT can be connected to a 10GbE or 25GbE port, multiple combinations of tests are provided. They all use the same naming convention based on lane rate, GT under test and number of GTs supported by the platform:

- **2 GTs:** switch_<rate>_gt<index(es)> (for example switch_10gbe_gt01, switch_25gbe_gt1).
- **1 GT:** switch_<rate> (for example switch_10gbe, switch_25gbe).

Here is an example of how a pre-canned `test_sequence` may look like. Refer to the pre-canned test provided in the platform specific library for accurate content.

```
"gt_mac": {
    "0": {
    "global_config": {
        "utilisation"    : 99,
        "match_tx_rx"    : true,
        "test_sequence" : [
            [1,  "conf_10gbe_no_fec"],
            [1,  "clear_status"],
            [60, "run"],
            [1,  "check_status"]
        ]
    },
    "lane_config": {
        "0": {
            "tx_mapping" : 1
        },
        "1": {
            "tx_mapping" : 0
        },
        "2": {
            "tx_mapping" : 3
        },
        "3": {
            "tx_mapping" : 2
        }
    }
}
```

Stress Pre-Canned Test

The stress pre-canned test requires GTs to be connected to a 25GbE switch port (see GT Test Set Up).

The stress pre-canned test combines multiple test cases:

- verify
- switch_25gbe
- memory
- power

Refer to the pre-canned test provided in the platform specific library for accurate content.

IMPORTANT! *If the GTs are not connected to a 25GbE switch port, this pre-canned test will fail.*

Test Case Description

Test Cases Overview

Five different test cases can be run by `xbtest` depending on the deployment platform and the `xclbin` capabilities: verify, DMA, power, memory, and GT MAC. With the exception of the Verify test case, a test case is composed of, at minimum, a test sequence which contains the duration and some configurations.

Verify

- Detects and reports available CUs.
- Checks compatibility.
- Displays platform definition.
- Checks capabilities of all CUs.
- Checks basic communication between the host and the CUs: read/write of a scratch register.

DMA

- Checks data transfer between the host and the memories available on the card (for example, DDR/HBM).
- Each memory is tested individually.
- Uses counter for data integrity check.
- Measures read and write bandwidths between the host and the memories available on the card (for example, DDR/HBM).

***Note*:** The DMA test case does not use any CUs, but it requires an `xclbin` with connection to the memory under test.

Memory

- Checks data transfer between the CUs and:
  - memories available on the card (for example, DDR HBM).

    - memories available on the host via Slave Bridge.
- All memories are tested separately with different a configuration for each memory type.
- Uses PRBS31 for data integrity check and linear addressing.
- Measure read and write bandwidths and latencies.

Power

- Sets the power consumed by the card by controlling the toggle rate of the resources present in the CU.

Power CU only exercises the logic of the `xclbin`. The maximum and minimum power depend on the following:

- Selected Alveo™ card.
- Platform used (see Chapter 5: Platform Definition).
- Configuration of other tests running at the same time as the power test case.

GT MAC

- Checks the GT transceiver of the Alveo card at 10 Gigabit Ethernet (10 GbE) and 25 Gigabit Ethernet (25 GbE) lane rates.
- Uses the Xilinx XXV Ethernet IP core (see *10G/25G High Speed Ethernet Subsystem Product Guide* (PG210)).
- Includes a packet generator which allows an Alveo card with simple loopback cables to generate packets, and to verify that these packets are received error free.
- Supports configurable GT settings which allows the card to connect to a switch.

Verify Test Case

The verify test case is always executed at the start of any test and is not configurable. If any of the described checks fail, then none of the test cases are executed. `xbtest` software automatically detects the compute units (CUs) present in the `xclbin`. The verify test case crosschecks the contents of `xclbin` for the following:

- Its compatibility with the application software.
- The content of the test `JSON` file.

Compute Units Verification

Once uniquely identified, each CU present in the `xclbin` is checked for:

- Compatibility with application software version.
- Basic communication where multiple read/write data transfers to each CU scratch register are performed.

Test JSON File Verification

An initial sanity check of the test `JSON` file is performed when the application software starts. All `JSON` members are checked for validity and compatibility (value type and range) with the application software and the platform definition `JSON` file. This also includes a check of the `test_sequence` parameters for each test case. If a test case is described in the test `JSON` file, but the `xclbin` does not contain the associated CU, then the verify test case fails. If a test case is not described in the test `JSON` file, the associated CU will stay in its idle state, and it is not considered to be a verify test case failure.

Test `JSON` File Structure

CAUTION! *Test `JSON` files from previous versions of `xbtest` are not compatible with this version. Contact Xilinx Support for information on how to convert test `JSON` files from previous versions.*

The following is a test `JSON` file example. The various schemas present in test cases can be listed in any order. Schemas can be removed/added to disable/enable test cases. Test cases are executed in parallel with exception of:

- **Verify:** Executed prior to any tests, as a preliminary check.
- **DMA:** Executed prior to all other listed tests (memory, power, or GT MAC).

The test `JSON` file has the following properties:

- If the same member is repeated throughout your file, only the last value is used.
- Comments can be added or removed anywhere in test `JSON` file using member `comment`.

```
{
    "comment": "This is an example of test JSON file",
    "comment": "You can use this example as template for your own tests",
    "comment": "Use comment to detail your test if necessary (you can also
remove these lines)",
    "testcases": {
        "dma": {
            "global_config": {
                "comment": "Use comment to detail your test if necessary",
```

```
                "test_sequence":[
                    [10, "DDR"],
                    [10, "HBM"]
                ]
            }
        },
        "power": {
            "comment": "Use comment to detail your test if necessary",
            "global_config": {
                "test_sequence":[
                    [60, 15],
                    [120, 50]
                ]
            }
        },
        "memory": {
            "DDR": {
                "comment": "Use comment to detail your test if necessary",
                "global_config": {
                    "test_sequence":[
                        [60, "alternate_wr_rd"]
                    ]
                }
            },
            "HBM": {
                "global_config": {
                    "comment": "Use comment to detail your test if
necessary",
                    "test_sequence":[
                        [60, "alternate_wr_rd"]
                    ]
                }
            }
        },
        "gt_mac": {
            "comment": "Use comment to detail your test if necessary",
            "0": {
                "global_config": {
                    "test_sequence" :[
                        [1, "conf_10gbe_c74_fec"],
                        [1, "clear_status"],
                        [10, "run"],
                        [1, "check_status"]
                    ]
                }
            },
            "1": {
                "global_config": {
                    "test_sequence" :[
                        [1, "conf_10gbe_c74_fec"],
                        [1, "clear_status"],
                        [10, "run"],
                        [1, "check_status"]
                    ]
                }
            }
        }
    }
}
```

Test Environment Members

Table 10: **Test Environment Members**

Member	Mandatory/ Optional	Description
comment	Optional	If necessary, to detail your test, use comments at any level in the test JSON file.
testcases	Optional	List of the test cases to be executed, which are performed in parallel. A failure in a test case does not stop other test cases. Define which test is performed. Because test cases run in parallel, you can define 0, 1, or more. When not defined, the verify test case is still performed. Possible values include the following: • **dma:** DMA Test Case Description • **memory:** Memory Test Case Description • **power:** Power Test Case Description • **gt_mac:** GT MAC Test Case Description

DMA Test Case Description

The goal of this test case is to check communication and available bandwidth between host and memories available on the card (for example DDR/HBM) through the PCIe®. Data integrity and write/read bandwidths are measured.

The DMA test case consists of writing and reading back data to and from the entire range of the memory under test over and over during a certain period. A write-read-check cycle is never interrupted, meaning that:

- Data is always fully sent and read back to and from the entire range of the memory and checked for data integrity
- If required, a test duration can be extended to perform all write-read-check cycle operations.

The data sent and read back to and from the memory is:

- Generated via an 8-bit counter that is randomly initialized at the beginning of each write-read-check cycle.
- Split into buffers which are transferred (via OpenCL™) to or from the Alveo™ card.

The write/read bandwidths are computed after all write/read data transfers in each write-read-check cycle and the values are averaged over the test duration.

IMPORTANT! *By default, the average read and write bandwidths are not checked against any pass/fail criteria, but this can be overruled (see DMA Test JSON Members).*

Required Test Parameters

The following test configuration parameters must be used, although others are available.

- `duration`: The duration of the test (in seconds).
- `mem_type_tag`: Name of the memory type (for example, DDR/HBM) or tag (for example, DDR[0]/HBM[12]) of memory to access.

Related Information

DMA Test JSON Members

Main Test Steps

For each test configuration, the following steps are repeated:

1. Allocate `N` host buffers of 256 MB (aligned with memory page size), where `N` equals the quantity of data to be sent and read back (in MB) divided by 256.
2. Allocate and initialize the reference buffer used to check data integrity.
3. Write-read-check cycles are repeated for the duration of the test. One cycle consists of the following steps:
   a. Set host buffers with reference data (8-bit counter).
   b. Write host buffers to the card memory, measure bandwidth.
   c. Reset host buffers to 0.
   d. Read from the card memory, measure bandwidth.
   e. Check that the host buffers contain the same data as the reference buffers (data integrity).

   Note: These steps constitute a write-read-check cycle which is always entirely executed.
4. Compute write and read minimum, maximum, and average bandwidths.
5. If enabled, compare the average read and write bandwidths against their thresholds.
6. Release all host buffers.

Low DMA Bandwidth Troubleshooting

The DMA test always reports the read and write bandwidths. There are many factors influencing these bandwidths:

- Server PCIe architecture.
- PCIe load:
  - Running other DMA tests.
  - Loading `xclbin` on other cards.
- PCIe training: if all PCIe are lane trained:
  - The command `xbutil validate` reports a message if the PCIe is not operating at its best rate.
  - The Linux command `lspci` can also report the speed and the quantity of lanes in use.
- CPU affinity (NUMA nodes): the CPU affinity of the card is reported by the command `xbutil query` (or `xbutil dump`). Use the Linux command `taskset` to run `xbtest` on the desired CPUs. For example:

```
$ taskset -c 0,2,4,6,8,10 xbtest -d <BDF> -c dma
```

DMA Test JSON Members

The following table shows all members available for this test case. More details are provided for each member in the subsequent sections.

Table 11: **DMA Test Case Members**

Member	Memory Type Override	Mandatory/ Optional	Description
test_sequence	no	mandatory	Describes the sequence of tests to perform.
check_bw	yes	optional	Enable bandwidth checking. Disabled by default.
hi_thresh_wr hi_thresh_rd	only	optional	Overwrite high threshold of the write/read bandwidth (MB/s) for specified memory type.
lo_thresh_wr lo_thresh_rd	only	optional	Overwrite low threshold of the write/read bandwidth (MB/s) for specified memory type.

Basic Example

The following is a basic example of a DMA test case targeting all DDR and HBM memories available on the card. All memories are tested *serially*.

```
"dma": {
    "global_config": {
        "test_sequence": [
            [10,"DDR"],
            [10,"HBM"]
              |    └──────> mem_type_tag
              └───────────> duration
        ]
    }
}
```

Some test `JSON` members can be overwritten for all memories based on memory type using the test `JSON` member `memory_type_config` which child members are memory type names. Following is an example of DMA test cases where the comparison of the average read and write bandwidths against thresholds is enabled only for all memories of type HBM.

```
"dma": {
    "global_config": {
        "test_sequence": [
            [10,"DDR"],
            [10,"HBM"]
        ]
    },
    "memory_type_config": {
        "HBM": {
            "check_bw"    : true,
            "hi_thresh_wr": 13000,
            "hi_thresh_rd": 13000,
            "lo_thresh_wr": 9000,
            "lo_thresh_rd": 9000
        }
    }
}
```

***Note*:** By default, bandwidths are not checked, so `"check_bw": true` is present.

test_sequence

Mandatory. Describes the sequence of tests to perform. Tests are performed serially, and a failure in one test does not stop the sequence (the next test will be launched). There is no limit to the length of the `test_sequence`.

This field contains a list of tests, each test being defined by a list of parameters: `[ [], [], [] ]`.

The values are interpreted as: `[duration, mem_type_tag]`.

Where:

- `duration`: The duration of the test in seconds: Range [1, 2^{32}-1]
- `mem_type_tag`: Name of the memory type (for example DDR/HBM) or tag (for example DDR[0]/HBM[12]) of memory to access:
  - The index must be within the range specified in the platform definition file.
    - When a test is defined by memory type. One test is created for each memory tag of the memory type.

      Note: The memory tags applicable for each memory type are displayed when the DMA test case starts.
    - The test fails when the name provided does not match any of the memory type available in the `xclbin` or if the memory tag to test is not connected in the `xclbin`.

      TIP: *Memory information can be retrieved using the xbutil query and checking the tag associated with each memory.*

Note: The different Alveo™ cards have different memory types. The following command can be used to identify the names of available memory types and associated memory tags on the card selected with card BDF `<BDF>` (see Command Line Options):

```
$ xbtest -d <BDF> -g dma
```

For example:

- Single test:
  - `[[50,"DDR"]]`
  - `[[50,"HBM"]]`
  - `[[50,"DDR[0]"]]`
  - `[[50,"HBM[1]"]]`
- Multiple test:
  - `[[50,"DDR[0]"],[1,"DDR[1]"],[10,"DDR[2]"]]`

  Note: If, in an `xclbin`, the memory tags DDR[0] and DDR[1] are available for the memory type named DDR, then the sequence set to `[[5,"DDR[0]"], [5,"DDR[1]"]]` is equivalent to the sequence `[[5,"DDR"]]`

check_bw

check_bw

Optional. Type: boolean. Possible values: `false` or `true`; default `false`.

By default, none of bandwidth values (read or write) are checked against any pass/fail criteria; they are just reported as information. By setting this member to **true**, all bandwidth results will be compared to thresholds specified inside the platform definition file. These thresholds can be overwritten (see the following section).

hi_thresh_wr, hi_thresh_rd

hi_thresh_wr, hi_thresh_rd

Optional. Type: integer. Possible values: from 1 to 2^{32}-1; default: specified in the platform definition `JSON` file.

Overwrite high threshold of the write/read bandwidth (MB/s) specified in the platform definition `JSON` file for specified memory type (see Chapter 5: Platform Definition). After all bandwidth measurements made during the test duration are complete, if the measured bandwidth is greater than this threshold, the test fails.

lo_thresh_wr, lo_thresh_rd

lo_thresh_wr, lo_thresh_rd

Optional. Type: integer. Possible values: from 1 to 2^{32}-1; default: specified in the platform definition `JSON` file.

Overwrite low threshold of the DDR/HBM write/read bandwidth (MB/s) specified in the platform definition `JSON` file for specified memory type (see Chapter 5: Platform Definition). After all bandwidth measurements made during the test duration are complete, if the measured bandwidth is lower than this threshold, the test fails. Low threshold must be lower than high threshold.

Output Files

All DMA measurements are stored in output `CSV` files which are generated in `xbtest` logging directory. The values are stored in `CSV` type format with one column for each information type.

IMPORTANT! *If the `-L` command line option is used while calling the application software, no output file is generated.*

In the DMA test case, two different `CSV` files are used to store all test results. They are named with the following convention:

- `dma_detail.csv`
- `dma_result.csv`

`dma_detail.csv` Output File

This file contains all intermediate bandwidth measurements for all memory types available on the card (for example DDR/HBM). There is one line of result for every write-read-check cycle of each test of the `test_sequence`. The following table summarizes the content of this file, where the following columns represent groups of columns actually present in the file for a platform containing a memory type named DDR with the two tags DDR[0] and DDR[1] associated and containing also a memory type named HBM with the 32 tags associated: HBM[0] to HBM[31]:

- **write results:** Group of columns for DMA write results.
- **read results:** Group of columns for DMA read results.

Table 12: **Example:** `dma_detail.csv`

Test	memory tag	buffer size (MB)	Cycle ID	write results	read results
1	DDR[0]	256	0	...	...
1	DDR[0]	256	1	...	...
	...	...	...	...	...
2	DDR[1]	256	0	...	...
2	DDR[1]	256	1	...	...
...	...	...	...	...	...
3	HBM[0]	256	0	...	...
...	...	...	...	...	...
4	HBM[1]	256	0	...	...
...	...	...	...	...	...
34	HBM[31]	256	0	...	...

- **Test:** Index of current test within the `test_sequence`. Index of first test is 1.
- **memory tag:** Tested memory name.
- **buffer size (MB):** Size of buffers transferred during the test.
- **Cycle ID:** Index of the write-read-check cycle: the number of cycles depends on duration and quantity of data transferred.
- **write results:** This group contains the following columns:
  - **live write BW (MB/s):** DMA write BW measurements for the current write-read-check cycle.
  - **minimum write BW (MB/s):** Minimum of DMA write BW measurements.
  - **average write BW (MB/s):** Average of DMA write BW measurements.
  - **maximum write BW (MB/s):** Maximum of DMA write BW measurements.

- **read results:** This group contains the following columns:
  - **live read BW (MB/s):** DMA read BW measurements for the current write-read-check cycle.
  - **minimum read BW (MB/s):** Minimum of DMA read BW measurements.
  - **average read BW (MB/s):** Average of DMA read BW measurements.
  - **maximum read BW (MB/s):** Maximum of DMA read BW measurements.

`dma_result.csv` Output File

For each test of the `test_sequence`, a new row containing the test configuration and results computed is present in this file. The following table summarizes the content of this file, where the following columns represent groups of columns actually present in the file (see description below) for a platform containing a memory type named DDR with the two tags DDR[0] and DDR[1] associated and containing also a memory type named HBM with the 32 tags associated HBM[0] to HBM[31]:

- **configuration:** Group of columns for DMA write/read configuration.
- **write results:** Group of columns for DMA write results.
- **read results:** Group of columns for DMA read results.

Table 13: **Example:** `dma_result.csv`

Test	memory tag	configuration	Number of cycles	write results	read results
1	DDR[0]	...	...	...	...
2	DDR[1]	...	...	...	...
3	HBM[0]	...	...	...	...
4	HBM[1]	...	...	...	...
...	...	...	...	...	...
34	HBM[31]	...	...	...	...

- **Test:** Index of current test within the `test_sequence`. Index of first test is 1.
- **memory tag:** Tested memory name.
- **configuration:** This group contains the following columns:
  - **buffer size (MB):** Size of buffers transferred during the test.
  - **number of buffers:** Quantity of buffers transferred in each write-read-check cycle.
  - **total size (MB):** Total quantity of data transferred in each write-read-check cycle.

- **Number of cycles:** Total number of write-read-check cycles performed: the number of cycles depends on duration and quantity of data transferred.
- **write results:** This group contains the following columns:
  - **minimum write BW (MB/s):** Minimum of DMA write BW measurements.
  - **average write BW (MB/s):** Average of DMA write BW measurements.
  - **maximum write BW (MB/s):** Maximum of DMA write BW measurements.
- **read results:** This group contains the following columns:
  - **minimum read BW (MB/s):** Minimum of DMA read BW measurements.
  - **average read BW (MB/s):** Average of DMA read BW measurements.
  - **maximum read BW (MB/s):** Maximum of DMA read BW measurements.

Memory Test Case Description

The goal of this test case is to check communications between the memory CUs and memories available in the following places:

- **On the card:** Typically, DDRs.
- **In the FPGA:** For example, HBM.
- **On the host (with PCIe slave-bridge):** Slave-bridge provides access between the memory CU and host memory (HOST) via PCIe. The memory CU can use the host memory the same way it currently uses the other memories.

***Note*:** In the DMA test case, `xbtest` software controls data transfers between host and memories available on the card and in the FPGA. In the memory test case, `xbtest` commands the memory CUs to transfer data between the CU and their associated memories located on the card, in the FPGA, or on the host.

The memory test case includes the following features:

- All memories are tested in parallel.
- All memories of a type (for example DDR, HBM, or HOST) are tested with the same sequence (see Memory CU Types).
- Data integrity is checked using PRBS31 generator/checker within the CU.
- Write and read bandwidths and latencies are measured.
- Mode of data transfer is configurable. Available modes are only/alternate/simultaneous write/read.

- Write and read rates of the memory CU data transfers are configurable separately.
- All transfers are performed using linear addressing.

> **IMPORTANT!** *Host memory must be allocated prior being tested (see Host Memory (Slave Bridge) Set Up).*

Test Parameters

The mandatory test configuration parameters are as follows (see Memory Test JSON Members for more details):

- `duration`: Specifies the test duration in seconds.
- `test_mode`: Describes the data transfer mode: `alternate_wr_rd`, `only_wr`, `only_rd`, `simultaneous_wr_rd`.

> **IMPORTANT!** *By default, `xbtest` loads a valid PRBS31 data sequence in the memory before any `only_rd` or `simultaneous_wr_rd` test.*

The following optional parameters may also be specified (see Memory Test JSON Members):

- `wr_rate`: Write data transfer rate.
- `rd_rate`: Read data transfer rate.

> **IMPORTANT!** *For some memory types, the memory CU has been intentionally designed to exceed the power capacity of the card. Your server/workstation may reboot or `xbtest` may be interrupted if you try to use a high write or read data transfer rates. Nominal write and read data transfer rates are specified in platform definition file (see Chapter 5: Platform Definition).*

Main Test Steps

For each test configuration, the following steps are repeated:

1. The test is run for at least the defined duration. The entire range of the memory is always checked, meaning that, if needed, the test duration is extended.
2. Every second, the application software requests status and measurements from the CU:
   - Read/write bandwidths and latencies.
   - Data integrity status.
3. After the test completes, the application software displays the average read/write bandwidths. The bandwidths are also checked against thresholds when the test duration is greater than 20s. Latencies are always reported as information and are not checked against any thresholds.

Memory CU Types

Different Alveo™ cards support various memory types of different sizes (for example, multiple 16 GB of DDR, 8 GB of HBM, 2 GB of PL-DDR, and/or 4 GB of PS-DDR) which are automatically detected by `xbtest` during the verify test case. Each of these memories is tested with either of these different types of memory CU:

- **Single-channel (SC):** One or more CU with 1 channel (1 AXI interface). Each CU targets a different memory entirely. All memory test cases for each SC memory CU run in parallel. For example, DDR, PL_DDR or HOST are SC memory types.
- **Multi-channel (MC):** Only one CU with up to 32 channels targeting different areas of the same memory. All memory test cases for each channel of the MC memory CU run in parallel. For example, HBM, PS_DDR are MC memory types.

  ***Note*:** For example, as the HBM stack can be split into pseudo channels (PCs) (see *AXI High Bandwidth Controller LogiCORE IP Product Guide* (PG276)), a MC memory CU is used. Each channel of the memory CU can be connected either one or more PCs.

For each memory types, the size of AXI W/RDATA buses can be different. The maximum data size is typically used (512 bits) but it can be lower for example for PS_DDR memory type (128 bits).

***Note*:** Different Alveo™ cards might have different memory types. To identify the names of available memory types on the card selected with card BDF <BDF>, use the following command (see Command Line Options):

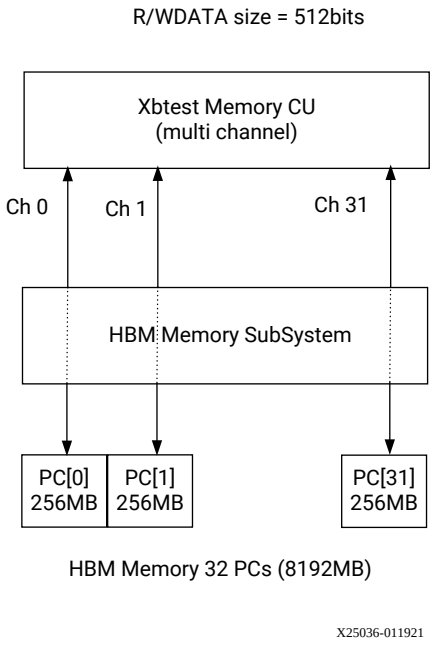
```
$ xbtest -d <BDF> -g memory
```

More information on the configuration is also available in `xbtest.log`, with message ID `ITF_058` for each memory type and message ID `ITF_100` for each CU.

32-channel HBM

The following figure represents one 32-channel memory CU, with each channel targets a different PC (different memory tags).

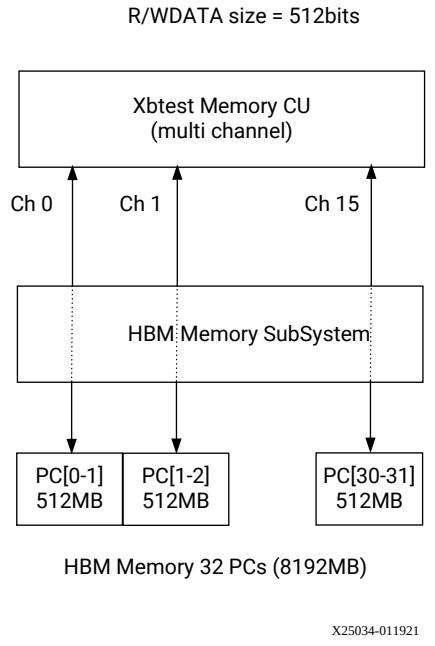
Figure 4: **32-channel HBM Memory CU type**



16-channel HBM

The following figure represents one 16-channel memory CU, with each channel targeting a different series of two PCs (different memory tags).

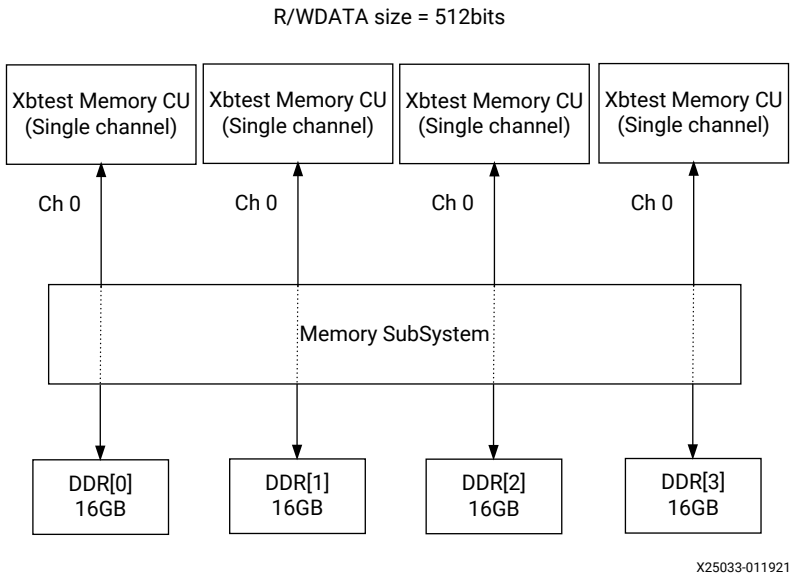
Figure 5: **16-channel HBM Memory CU type**



DDR

The following figure represents four SC memory CUs, with each CU targeting a different DDR.

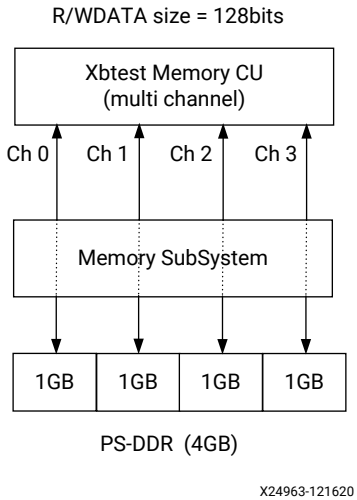
Figure 6: **DDR Memory CU type**



PS_DDR

The following figure represents one 4-channel memory CU, with each channel targeting a different area of the same memory (same memory tag).

Figure 7: **PS_DDR Memory CU type**



Write and Read Ranges/Blocks

Data is transferred from/to the memory via multiple AXI bursts, which address linearly the entire range of the memory. By default, the entire range of the memory is tested. When the memory CU is:

- **Single-channel:** The range is simply the size of the memory.
- **Multi-channel:** The range is based on the quantity of memories connected to each channel of the memory CU. For example with:
  - **32-channel HBM:** Each channel of the memory CU has a range of 256 MB (size of one HBM Pseudo Channel).
  - **16-channel HBM:** Each channel of the memory CU has a range of 512 MB (size of two HBM Pseudo Channel).

Throughout the documentation, the terminology of *block* is also used to refer to the range under test.

A block of data is fully:

- Written to the memory, when the last burst of data is acknowledged (`BVALID`) to the memory CU.
- read from the memory, when the last data (`RLAST`) of the last burst is received by the memory CU.

For more information about AXI protocol, refer to *Vivado Design Suite: AXI Reference Guide* (UG1037).

Note: When a block of data is fully read/written, the memory CU repeats the transfer of data during the entire duration, but the memory range is always entirely transferred. The duration of the test is automatically increased accordingly, meaning that the reading/writing of a block is never interrupted.

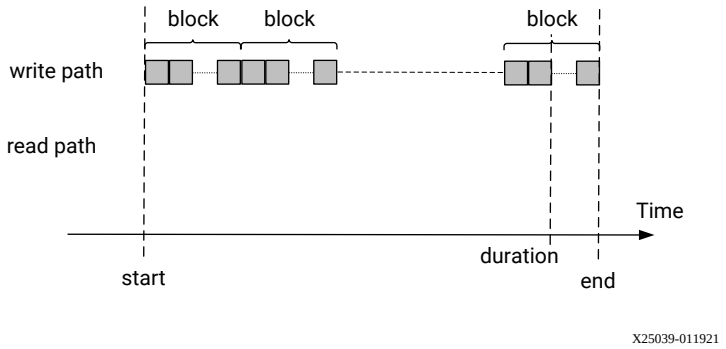
Test Modes

The memory CUs can be configured to write and/or read data to/from the targeted memory in four different modes.

`only_wr` Test Mode

The memory range (block) is fully written over and over during the entire duration of the test. The following figure represents the blocks transferred during an `only_wr` test.

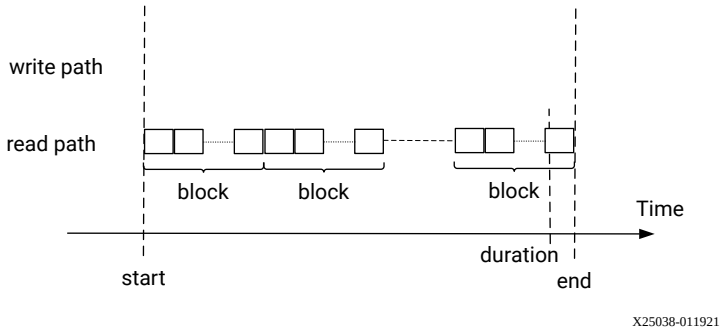
Figure 8: `only_wr` **Test Mode Operations**



`only_rd` Test Mode

The memory range (block) is fully read over and over during the entire duration of the test. A single preliminary write (with known PRBS31 data) of the memory is performed prior starting the reading. The following figure represents the blocks transferred during an `only_rd` test.

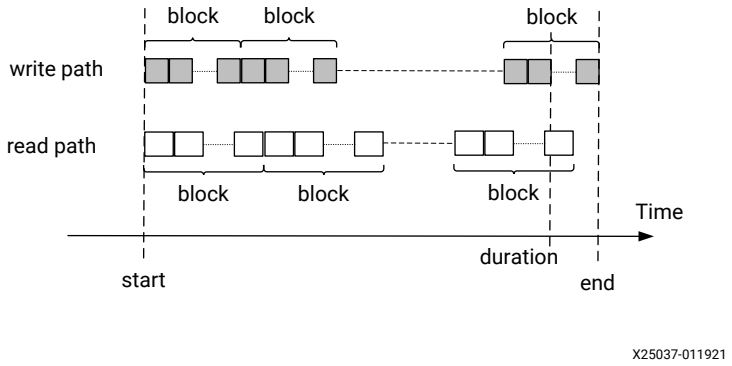
Figure 9: `only_rd` **Test Mode Operations**



`simultaneous_wr_rd` Test Mode

The first half of the memory range is fully written and the second half of the memory range is read simultaneously. A single preliminary write (with known PRBS31 data) of the second half of the memory is performed prior starting the reading. The following figure represents the blocks transferred during a `simultaneous_wr_rd` test. In this mode, a block represents half of the memory range.

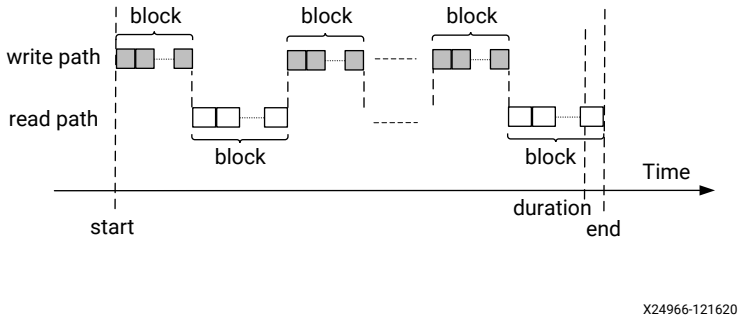
Figure 10: `simultaneous_wr_rd` **Test Mode Operations**



`alternate_wr_rd` Test Mode

The full range of the memory is written, then fully read. The following figure represents the blocks transferred during an `alternate_wr_rd` test:

Figure 11:



AXI Bursts

The memory CU communicates with the memory via one or more AXI interfaces (channels). The size of the AXI burst is typically 4 KB. For more information about AXI protocol, refer to *Vivado Design Suite: AXI Reference Guide* (UG1037).

Table 14: **AXI Data size vs. AXI Beat**

AXI Data Size (in bits)	AXI Beat Quantity	AXI Burst Size (in KB)
512	64	4
256	128	4
128	128	2

Measurements

The memory CUs computes burst time and latency. The bandwidth is computed every second by the `Test` software after requesting measurements from the CU.

Burst Time

The following sections describes the start and end point for burst time measurement, which are similar in the write and read directions. The memory test case reports minimum, average and maximum burst time, which are not check against any thresholds.

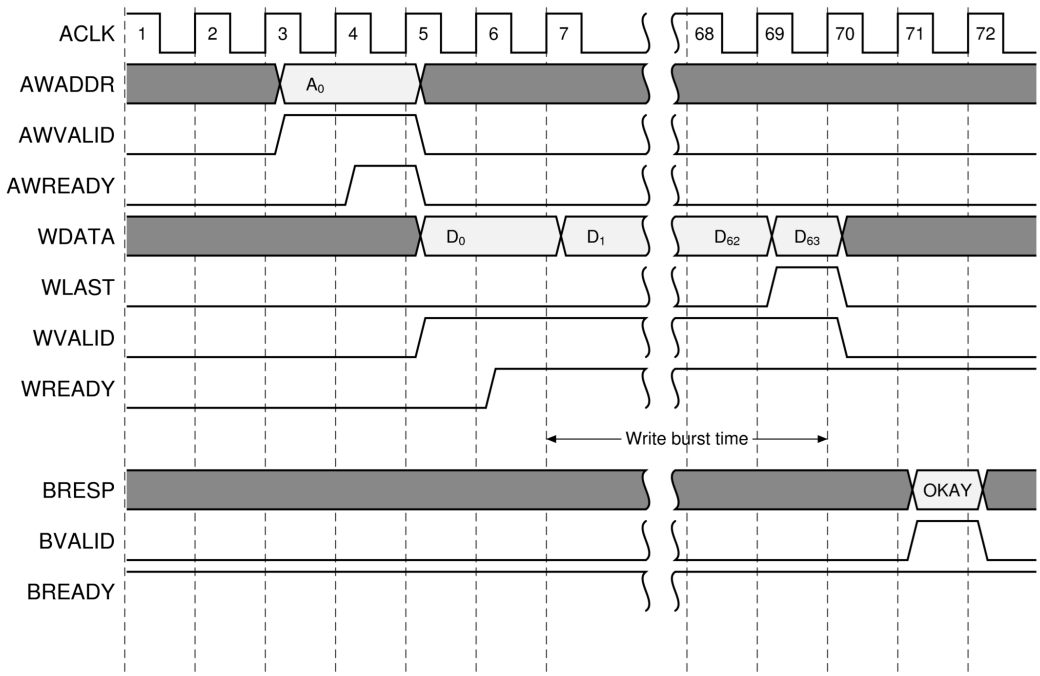
Burst Time

The write burst time is measured for a given burst using the following:

- **Start point:** First time `WVALID` and `WREADY` are asserted.
- **End point:** `WVALID`, `WLAST` and `WREADY` are asserted.

The following figure represents the write burst time measurement:

Figure 12: **Write Burst Time**



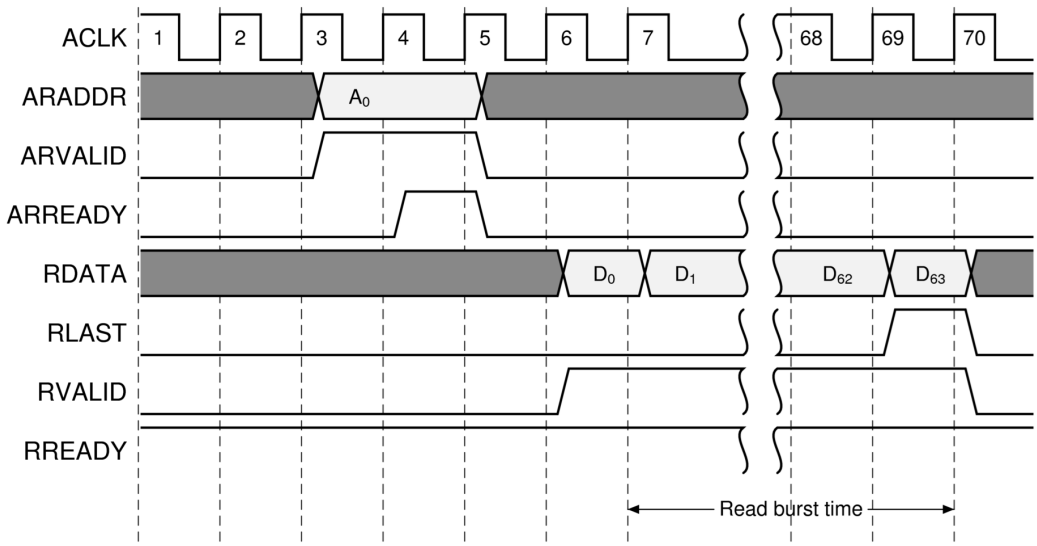
Read Burst Time

The read burst time is measured for a given burst using the following:

- **Start point:** First time `RVALID` and `RREADY` are asserted.
- **End point:** `RVALID`, `RLAST` and `RREADY` are asserted.

The following figure represents the read burst time measurement:

Figure 13: **Read Burst Time**



Bandwidth

The average write/read bandwidth is computed as write/read burst size divided by average write/read burst time. The measured average bandwidth can be checked against thresholds when enabled.

IMPORTANT! *The bandwidth checks are disabled by default when the host memory is targeted.*

Latency

The following sections describes the start and end point for latency measurement. The memory test case reports minimum, average and maximum burst latency. The measured average latency can be checked against thresholds when enabled.

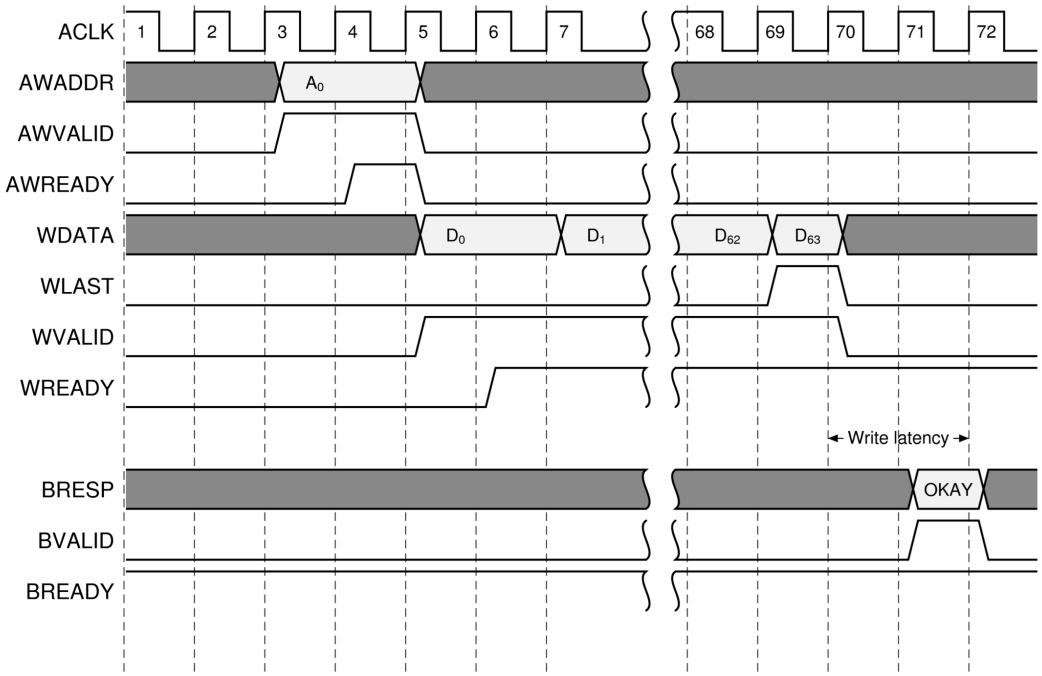
Write Latency

The write burst latency is measured for a given burst using the following:

- **Start point:** `WVALID`, `WLAST` and `WREADY` are asserted.
- **End point:** `BVALID` and `BREADY` are asserted.

The following figure represents the write latency measurement:

Figure 14: **Write Burst Latency**



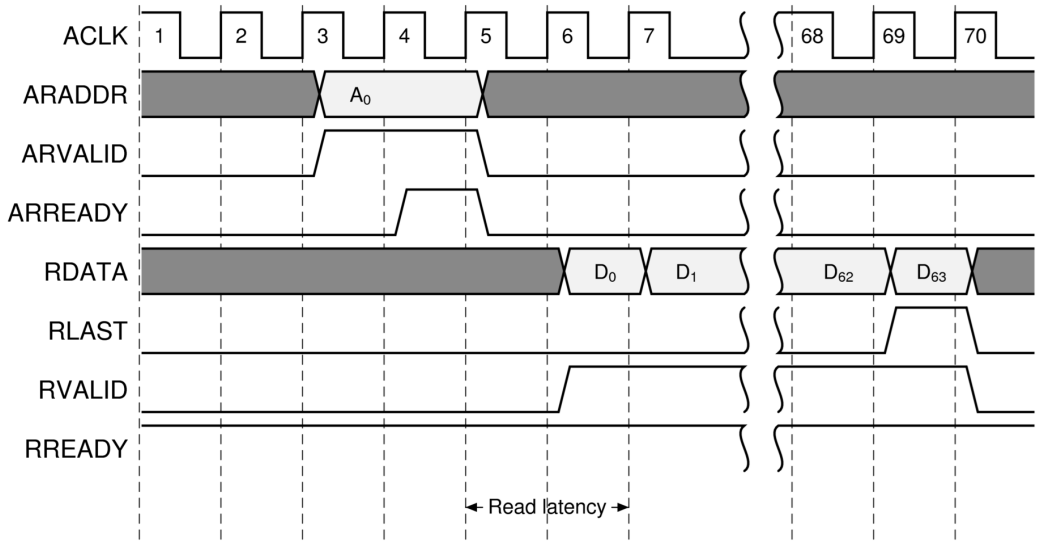
Read Latency

The read burst latency is measured for a given burst using the following:

- **Start point:** `ARVALID` and `ARREADY` are asserted.
- **End point:** First time `RVALID` and `RREADY` are asserted.

The following figure represents the read latency measurement.

Figure 15: **Read Burst Latency**



Memory Test JSON Members

Target Memories on the Card

Following is an example of memory test cases targeting all memories of type DDR and HBM available on the card. Note that all memories are tested in parallel.

```
"memory" : {
    "DDR": {
        "global_config": {
            "test_sequence": [
                [30, "simultaneous_wr_rd"],
                [30, "alternate_wr_rd"]
                  |                 └──────> test_mode
                  └────────────────────────> duration
            ]
        }
    },
    "HBM": {
        "global_config": {
            "test_sequence": [
                [30, "only_wr"],
                [30, "only_rd"]
            ]
        }
    }
}
```

Target Memories on the Host

Following is an example of memory test case targeting all memories of type HOST:

```
"memory" : {
    "HOST": {
        "global_config": {
            "test_sequence": [
                [30, "simultaneous_wr_rd"],
                [30, "only_wr"],
                [30, "only_rd"]
            ]
        }
    }
}
```

IMPORTANT! *The memory test case targeting memories available on the host should not be run in parallel with other memory, power or GT MAC test cases.*

Definition

In the memory test case, valid test `JSON` members depend on the configuration of the memory CUs (see Memory CU Types), which are automatically detected by the application software during the verify test case. The following table shows all members available for this test case. More details are provided for each member in the subsequent sections:

Table 15: **Memory Test Case Members**

Member	Mandatory/ Optional	Description
test_sequence	mandatory	Describes the sequence of tests to perform.
check_bw	optional	Enable the check of the bandwidths. This check is by default for memories available: • **On the card:** Enabled. • **On the host:** Disabled.
hi_thresh_alt_wr hi_thresh_alt_rd hi_thresh_only_wr hi_thresh_only_rd hi_thresh_simul_wr_bw hi_thresh_simul_rd_bw	optional	Overwrite high threshold of write/read bandwidth (MB/s).
lo_thresh_alt_wr lo_thresh_alt_rd lo_thresh_only_wr lo_thresh_only_rd lo_thresh_simul_wr_bw lo_thresh_simul_rd_bw	optional	Overwrite low threshold of the write/read bandwidth (MB/s).
check_latency	optional	Enable the check of the latencies.

Table 15: **Memory Test Case Members** *(cont'd)*

Member	Mandatory/ Optional	Description
hi_thresh_alt_wr_lat hi_thresh_alt_rd_lat hi_thresh_only_wr_lat hi_thresh_only_rd_lat hi_thresh_simul_wr_lat hi_thresh_simul_rd_lat	optional	Overwrite high threshold of write/read latency (MB/s).
lo_thresh_alt_wr_lat lo_thresh_alt_rd_lat lo_thresh_only_wr_lat lo_thresh_only_rd_lat lo_thresh_simul_wr_lat lo_thresh_simul_rd_lat	optional	Overwrite low threshold of the write/read latency (MB/s).

test_sequence

Mandatory. Describes the sequence of tests to perform. Tests are performed serially and a failure in one test does not stop the sequence (the next test will be launched). There is no limit to the length of the `test_sequence`.

This field contains a list of tests, each test being defined by a list of parameters: [[], [], []].

The values are interpreted as:

- [`duration`, `test_mode`] when:
  - nominal write and read rates are used.
- [`duration`, `test_mode`, `wr_rate`, `rd_rate`] when:
  - write and read rates are overwritten.
  - `test_mode` is `alternate_wr_rd` or `simultaneous_wr_rd`
- [`duration`, `test_mode`, `wr/rd_rate`] when:
  - write/read rate is overwritten.
  - `test_mode` is `only_wr/rd` .

Where:

- **`duration`**: Test duration in seconds; Range: [1, 2^{32}-1].
- **`test_mode`**: Test mode; Possible value: `alternate_wr_rd`, `only_wr`, `only_rd` and `simultaneous_wr_rd`.

- **`wr/rd_rate`:** Optional write/read rate in percent; Range: [1, 100]. Overwrites nominal rate specified in platform definition file (see Chapter 5: Platform Definition).

For example:

- **Single test:** `[[60,"alternate_wr_rd"]]`
- **Multiple tests:** `[[60, "alternate_wr_rd"], [60,"only_wr"], [60,"only_rd"]]`
- **Nominal rate override:** `[[60,"only_rd", 25], [60,"simultaneous_wr_rd", 20, 25]]`

check_bw

Optional. Type: boolean. Possible values: `true` or `false`. Default: for memories available:

- **On the card:** `true`.
- **On the host:** `false`.

By setting this member to `false`, no average bandwidth measurement will be compared against defined thresholds.

Default bandwidth limit may be defined by the Platform Definition `JSON` file (see Chapter 5: Platform Definition) but can be overwritten by the user (see the following). The default values are displayed at the beginning of each test.

hi_thresh_alt_wr, hi_thresh_alt_rd, hi_thresh_only_wr, hi_thresh_only_rd, hi_thresh_simul_wr_bw, hi_thresh_simul_rd_bw

Optional. Type: integer. Range: [1, 2^{32}-1]. Overwrite high threshold of the average write/read bandwidth (MB/s) specified in platform definition `JSON` file (see Chapter 5: Platform Definition).

After all bandwidth measurements made during the test duration are complete, if the measured average bandwidth is greater than this threshold, the test fails.

The following table gives the write and read bandwidth high thresholds based on `test_mode`:

Table 16: **Bandwidth High Thresholds**

Test mode	Write	Read
`alternate_wr_rd`	`hi_thresh_alt_wr_bw`	`hi_thresh_alt_rd_bw`
`only_wr`	`hi_thresh_only_wr_bw`	n/a
`only_rd`	n/a	`hi_thresh_only_rd_bw`
`simultaneous_wr_rd`	`hi_thresh_simul_wr_bw`	`hi_thresh_simul_rd_bw`

lo_thresh_alt_wr, lo_thresh_alt_rd, lo_thresh_only_wr, lo_thresh_only_rd

Optional. Type: integer. Range: [1, 2^{32}-1]. Overwrite low threshold of the average write/read bandwidth (MB/s) specified in platform definition `JSON` file (see Chapter 5: Platform Definition).

After all bandwidth measurements made during the test duration are complete, if the measured average bandwidth is lower than this threshold, the test fails.

Low threshold must be lower than high threshold.

The following table gives the write and read bandwidth low thresholds based on `test_mode`:

Table 17: **Bandwidth Low Thresholds**

Test mode	Write	Read
`alternate_wr_rd`	`lo_thresh_alt_wr_bw`	`lo_thresh_alt_rd_bw`
`only_wr`	`lo_thresh_only_wr_bw`	n/a
`only_rd`	n/a	`lo_thresh_only_rd_bw`
`simultaneous_wr_rd`	`lo_thresh_simul_wr_bw`	`lo_thresh_simul_rd_bw`

check_latency

Optional. Possible values: `true` or `false`; default: `false`.

By setting this member to `false`, no latency measurement (read or write) is checked against pass/failed criteria.

Default latency limit may be defined by the platform definition `JSON` file (see Chapter 5: Platform Definition) but can be overwritten by the user (see the following). The default values are displayed at the beginning of each test.

TIP: *For some platforms, latency high and low thresholds may have been defined respectively very high and low. Consider defining thresholds in your test `JSON` file.*

hi_thresh_alt_wr_lat, hi_thresh_alt_rd_lat, hi_thresh_only_wr_lat, hi_thresh_only_rd_lat, hi_thresh_simul_wr_lat, hi_thresh_simul_rd_lat

Optional. Type: integer. Range: [1, 2^{32}-1]. Overwrite high threshold of the average write/read latency (ns) specified in platform definition `JSON` file (see Chapter 5: Platform Definition).

After all latency measurements made during the test duration are complete, if the measured average latency is greater than this threshold, the test fails.

The following table gives the write and read latency high thresholds based on `test_mode`:

Table 18: **Latency High Thresholds**

Test mode	Write	Read
alternate_wr_rd	hi_thresh_alt_wr_lat	hi_thresh_alt_rd_lat
only_wr	hi_thresh_only_wr_lat	n/a
only_rd	n/a	hi_thresh_only_rd_lat
simultaneous_wr_rd	hi_thresh_simul_wr_lat	hi_thresh_simul_rd_lat

lo_thresh_alt_wr_lat, lo_thresh_alt_rd_lat, lo_thresh_only_wr_lat, lo_thresh_only_rd_lat, lo_thresh_simul_wr_lat, lo_thresh_simul_rd_lat

Optional. Type: integer. Range: [1, 2^{32}-1]. Overwrite low threshold of the average write/read latency (ns) specified in platform definition `JSON` file (see Chapter 5: Platform Definition).

After all latency measurements made during the test duration are complete, if the measured average latency is lower than this threshold, the test fails.

Low threshold must be lower than high threshold.

The following table gives the write and read latency high thresholds based on `test_mode`:

Table 19: **Latency Low Thresholds**

Test mode	Write	Read
alternate_wr_rd	lo_thresh_alt_wr_lat	lo_thresh_alt_rd_lat
only_wr	lo_thresh_only_wr_lat	n/a
only_rd	n/a	lo_thresh_only_rd_lat
simultaneous_wr_rd	lo_thresh_simul_wr_lat	lo_thresh_simul_rd_lat

Output Files

All memory measurements are stored in output `CSV` files which are generated in `xbtest` logging directory. The values are stored in `CSV` type format with one column for each information type.

IMPORTANT! *If the `-L` command line option is used while calling the application software, no output file is generated.*

In the memory test case, different `CSV` files are used to store all memory test case results. They are named with a prefix the following convention base on memory type:

- **Single-channel:** `memory_<tag>`.
- **Multi-channel:** `memory_<tag>_ch_<channel>`.

Where:

- `<tag>` is the formatted tag of the memory targeted by the memory CU (lower case letters, numbers, square brackets or underscores are used).
- `<channel>` is the index of the memory CU channel.

In addition, for MC memory types, another prefix is used: **memory_<type name>_combined,** where **<type name>** is the memory type name. These files contain the combined results for all channels of the memory CU:

- **Bandwidth:** Sum of the BW of each channel.
- **Latency:** Average of the latency of each channel.

For each of these prefixes, two files are generated with `_detail.csv` (see `_detail.csv` Output Files) and `_result.csv` (see `_results.csv` Output Files) suffixes.

Additionally, a file, named with suffix `_power.csv` is generated for each memory CU (see `_power.csv` Output Files).

For example, for a given deployment platform, if `xbtest` supports the following memory types:

- One SC memory type named DDR
  - **Configuration:** Two memory CUs are instantiated:
    - One CU is connected to DDR[0].
    - One CU is connected to DDR[1].
  - **Generated output files:**
    - `memory_ddr[0]_detail.csv`.
    - `memory_ddr[0]_result.csv`.
    - `memory_ddr[1]_detail.csv`.
    - `memory_ddr[1]_result.csv`.
    - `memory_DDR_ddr[0]_power.csv`.
    - `memory_DDR_ddr[1]_power.csv`.
- One MC memory type named HBM
  - **Configuration:** One 16-channel memory CU is instantiated:
    - Channel 0 is connected to HBM[0:1].
    - Channel 1 is connected to HBM[2:3].
    - ...

    - Channel 15 is connected to HBM[30:31].
- **Generated output files:**
    - `memory_hbm[0_1]_ch_0_detail.csv`.
    - `memory_hbm[0_1]_ch_0_result.csv`.
    - `memory_hbm[2_3]_ch_1_detail.csv`.
    - `memory_hbm[2_3]_ch_1_result.csv`.
    - ...
    - `memory_hbm[30_31]_ch_15_detail.csv`.
    - `memory_hbm[30_31]_ch_15_result.csv`.
    - `memory_HBM_combined_detail.csv`.
    - `memory_HBM_combined_result.csv`.
    - `memory_HBM_power.csv`.

`_results.csv` Output Files

For each test of the `test_sequence`, a new row containing the test configuration and results computed is present in this file. All columns present in the file are defined as:

- **Test:** Index of current test within the `test_sequence`. Index of first test is 1.
- **duration (s):** Duration of current test within the `test_sequence`.
- **test mode:** Mode of current test within the `test_sequence`.
- **data integrity:** Overall data integrity result: set to `KO` if data integrity error was detected, otherwise set to `OK`.
- **average total write+read BW (MBps):** Sum of `average write BW (MBps)` and `average read BW (MBps)`.
- **write configuration:** This group contains the following columns. These columns are set to `n/a` when `test_mode` is `only_rd`.
    - **write rate (%):** Rate used for writing the memory.
    - **write start address offset (MB):** Starting address of the memory to be written.
    - **write end address offset (MB):** Ending address of the memory to be written.
    - **write burst size (Bytes):** Size of each bursts to be written in the memory.
    - **write block size (MB):** Size of the block to be written in the memory.

  - **AXI write data size (Bytes):** Size of an AXI beat in a burst.
  - **number of write transfers per burst:** Quantity of AXI beats per burst for write direction.
  - **total number of write bursts per block:** Quantity of bursts per block for write direction.
  - **total number of write transfers per block:** Quantity of AXI beats to write the memory based on `write block size (MB)` and `AXI write data size (Bytes)`.
- **write results:** This group contains the following columns. These columns are set to `n/a` when `test_mode` is `only_rd`.
  - **average write BW (MBps):** Average write BW (MB/s).
  - **average number of write bursts per second:** Average quantity of bursts written in the memory every second.
  - **minimum write burst time (ns):** Minimum time needed for writing a burst.
  - **average write burst time (ns):** Average time needed for writing a burst.
  - **maximum write burst time (ns):** Maximum time needed for writing a burst.
  - **minimum write burst latency (ns):** Minimum latency for writing a burst.
  - **average write burst latency (ns):** Minimum latency for writing ting a burst.
  - **maximum write burst latency (ns):** Maximum latency for writing a burst.
- **read configuration:** Same columns as `write configuration` group but for read direction. These columns are set to `n/a` when `test_mode` is `only_wr`.
- **read results:** Same columns as `write results` group but for read direction. These columns are set to `n/a` when `test_mode` is `only_wr`.

`_detail.csv` Output Files

This file contains all intermediate bandwidths computed every second, based on information retrieved from the memory CU. There is one line of result for every second of each test of the `test_sequence`. All columns present in the file are defined as:

- **Test:** Index of current test within the `test_sequence`. Index of first test is 1.
- **test mode:** Mode of current test within the `test_sequence`.
- **Measurement ID:** Measurement identifier. ID of first measurement is 0.
- **data integrity results:** This group contains the following columns. These columns are set to `n/a` when `test_mode` is `only_wr`.
  - **data integrity:** Overall data integrity result: set to `KO` as soon as data integrity error is detected, otherwise set to `OK`.

  - **live data integrity:** Live data integrity result: set to `KO` when data integrity error is detected, otherwise set to `OK`.

- **write+read BW:** This group contains the following columns.
  - **average total write+read BW (MBps):** Sum of `average write BW (MBps)` and `average read BW (MBps)`.
  - **live total write+read BW (MBps):** Sum of `live write BW (MBps)` and `live read BW (MBps)`.
- **write results:** This group contains the following groups. These columns are set to `n/a` when `test_mode` is `only_rd`.
  - **write BW:** This group contains the following columns.
    - **average write BW (MBps):** Current average value of write BW measurements.
    - **live write BW (MBps):** Current live measurement of write BW measurement.
  - **number of write bursts:** This group contains the following columns.
    - **average number of write bursts per second:** Current average number of write bursts per second performed by the memory CU.
    - **live number of write bursts per second:** Current live number of write bursts per second performed by the memory CU.
  - **write burst time:** This group contains the following columns.
    - **minimum write burst time (ns):** Current average value of minimum write burst time measurements.
    - **live minimum write burst time (ns):** Current live measurement of minimum write burst time.
    - **average write burst time (ns):** Current average value of average write burst time measurements.
    - **live average write burst time (ns):** Current live measurement of average write burst time.
    - **maximum write burst time (ns):** Current average value of maximum write burst time measurements.
    - **live maximum write burst time (ns):** Current live measurement of maximum write burst time.
  - **write burst latency:** This group contains the following columns.
    - **minimum write burst latency (ns):** Current average value of minimum write burst latency measurements.

  - **live minimum write burst latency (ns):** Current live measurement of minimum write burst latency.
  - **average write burst latency (ns):** Current average value of average write burst latency measurements.
  - **live average write burst latency (ns):** Current live measurement of average write burst latency.
  - **maximum write burst latency (ns):** Current average value of maximum write burst latency measurements.
  - **live maximum write burst latency (ns):** Current live measurement of maximum write burst latency.
- **read results:** Same columns as `write results` group but for read direction. These columns are set to `n/a` when `test_mode` is `only_wr`.

***Note*:** The live values are for internal use only of the application software.

`_power.csv` Output Files

For each memory CU, all power measurements are stored in an output `CSV` file named depending on the memory type:

- **Single-channel:** `memory_<type_name>_<tag>_power.csv`.
- **Multi-channel:** `memory_<type_name>_power.csv`.

Where:

- `<type name>` is the memory type name.
- `<tag>` is the formatted tag of the memory targeted by the memory CU (lower case letters, numbers, square brackets or underscores are used).

A new line is written in this file every time the memory CU status are available.

At a minimum, the following values are recorded:

- **Test:** Index of current test within the `test_sequence`. Index of first test is 1.
- **test mode:** Mode of current test within the `test_sequence`.
- **Measurement ID:** Measurement identifier. ID of first measurement is 0.
- **write rate (%):** Rate used for writing the memory.
- **read rate (%):** Rate used for reading the memory.
- **fan speed (rpm):** For passive cards, the column contains 0.

- **Temperature measurements:** Group of one or more columns recording temperatures. The temperature sources to monitor are defined in the platform definition `JSON` file (see Chapter 5: Platform Definition). The platform definition `JSON` file also contains their respective names, which are defined like those found when using the XRT command `xbutil query`. For more information, refer to the documentation for your card. Typically, temperature measurements are recorded in degree Celsius.
- **Power measurements:** Group of one or more columns recording detailed power measurements for each power rail. The power rails (voltage and current) to monitor are defined in the platform definition `JSON` file (see Chapter 5: Platform Definition). The platform definition `JSON` file also contains their respective names, which are defined like those found when using the XRT command `xbutil query`. For more information, refer to the documentation for your card. Typically, current measurements are recorded in mA, voltage in mV and power in W.

Power Test Case Description

The goal of this test is to allow the user to control the power consumption of the card. This is achieved by adjusting the toggle rate of the clock, which drives all flip-flops, DSPs, block RAMs, and UltraRAMs of the `xclbin` present in the power CU. The power is measured every second. It is computed based on the current and voltage measurements available via Xilinx® Runtime (XRT) library.

Test Parameters

The mandatory test configuration parameters are listed below. For more information, see Power Test JSON Members.

- `duration`: The duration of the test, measured in seconds.
- `percent`: The toggle rate, specified in `%`, driving the sites of the xclbin present in the power CU.

Main Test Steps

The measurement of the card rest power, which lasts a few seconds, is always performed when the power test case starts. For each test configuration, the following steps are repeated:

1. The toggle rate is set to the power CU.
2. For the defined duration, sensor values are read and reported every second.
3. When the test completes, it always passes because no checks are made on the consumed power. The user is responsible for monitoring the consumed power, which is displayed by the application software.

WARNING! *The power CU has been intentionally designed to exceed the power capacity of the card. You might damage your card and cause your server/workstation to reboot if you try to for example (but not limited to):*

- Use a high toggle rate.
- Use a particularly demanding test sequence (for example, alternating between a low and a high toggle rate for short periods of time).

IMPORTANT! *`xbtest` reports the entire power consumed by the card (for active card, fan speed is also reported).*

Power And Temperature Limits

To limit potential damage to the Alveo™ card in cases of accidental misuse or demanding test environmental conditions, the following basic safety mechanisms are in place.

- **Temperature Limit:** A critical warning is generated when the temperature limit is reached.
- **Power Limit:** A critical warning is generated when the power limit is reached.

Temperature and power limits are defined in the platform definition JSON.

Related Information

Platform Definition

Power Budget And Calibration

To establish the relationship between toggle rate and power, a simple calibration method can be used. An example of calibration is starting from 0%, increasing the toggle rate by 5%, and for each toggle rate step, letting the power and temperature stabilize for two minutes. There are numerous considerations to take into account when creating this relationship.

IMPORTANT!

- `xbtest` always reports the total power of the card (sum of the various power rails).
- Ensure that the environmental conditions (for example, temperature) used during calibration are similar to the conditions used in testing.

Actual Power Available

The distribution of the power across the various regulators also limits which power is available for `xbtest` to control. For example, on an Alveo U50 card, although the power budget of the card is 75W, up to 10W are reserved for the HBM. This means that Power CU can only control up to 65W. It also means that the Memory CU must be in use to have a card power consumption higher than 65W.

Moreover, the total power budget of the card is not entirely available. The actual power available would be impacted by the efficiency and current limitation of the various regulators. For information about various sensors and power rails limits, refer to the specific documentation for your card. With the same example (U50 card), the actual power thresholds will be lower than 65W and 10W.

Components Present On The Card

The card might be fitted with other ICs (such as co-processor, memories, and so forth) on which `xbtest` has no control. During calibration, ensure these components behave like similarly to normal test operations.

***Note*:** `xbtest` can only control power of memory directly connected to the FPGA.

Tests Running

Other test cases, like memory (DDR or HBM) and GT MAC, have a significant impact on the power consumed. Make sure that the calibration is done while using these other feature as per nominal load.

- **DDR:** When running four DDRs simultaneously, the memory test consumes approximately 20W (write mode) or 15W (read mode).
- **HBM:** For example, when eight HBM ports are used, the memory test consumes between 7W and 8W.
- **GT MAC:** When two GT MAC CUs are present, the 25 GbE mode uses ±6W more than the 10 GbE mode.
- **Logic:** Memory and GT MAC CUs, and the memory subsystem also consume several watts.

***Note*:** These values are indicative and might vary from card to card. They also depend on test environmental conditions, such as cooling.

Care must be taken when mixing test case types or when changing the mode of other tests while the power test is running. For example, power varies when the memory test changes from `only_rd` to `only_wr` mode. Power will decrease when a memory test case ends. `simultaneous_wr_rd` mode is usually the memory test mode consuming the most power.

Power Test JSON Members

The following is an example of a Power test case running for 60 seconds at a toggle rate of 15%.

```
"power": {
    "global_config": {
        "test_sequence":[[60,15]]
                          |  └──> percent
                          └─────> duration
    }
}
```

The following table shows all members available for this test case. More details are provided for each member in the subsequent sections.

Table 20: **Power Test Case Members**

Member	Mandatory/ Optional	Description
test_sequence	mandatory	Describes the sequence of tests to perform. A test is defined by the following values: • Duration: In seconds • Percent: Toggle rate

test_sequence

Mandatory. Describes the test to perform. Tests are performed serially and an error in one test does not stop the sequence (the next test will be launched). There is no limitation to the length of the `test_sequence`.

This field contains a list of tests, each test being defined by a list of parameters: `[ [], [], [] ]`

The values are interpreted as: `[duration, target]`

- `duration`: The duration of the test in seconds; range [1,2^{32}-1]
- `percent`: Toggle rate (in %); range [0, 100]

For example:

- Single test: `[ [40,75] ]`
- Multiple test: `[ [40,15], [240,30], [120, 40], [20,50] ]`

Output Files

All power measurements are stored in an output `CSV` file named `power.csv` which is generated in `xbtest` logging directory. All measurements from all `test_sequence` are combined into a single file.

IMPORTANT! *If the `-L` command line option is used while calling the application software, no output file is generated.*

A new line is written in this file every time power measurements are available. At a minimum, the following values are recorded:

- **test:** Index of current test within the `test_sequence`. Index of first test is 1. The first rows of the file with test and toggle rate set to 0 corresponds to the measurement of the card rest power.

- **time (s):** Timestamp of the measurement. Timestamp of first measurement is 0 for a given test within the `test_sequence`.
- **measurement ID:** Measurement identifier. ID of first measurement does not necessarily equal 1.
- **fan speed (rpm):** For passive cards, the column contains 0.
- **Temperature measurements:** Group of one or more columns recording temperatures. The temperature sources to monitor are defined in the platform definition `JSON` file (see Chapter 5: Platform Definition). The platform definition `JSON` file also contains their respective names, which are defined like those found when using the XRT command `xbutil query`. For more information, refer to the documentation for your card. Typically, temperature measurements are recorded in degree Celsius.
- **Power measurements:** Group of one or more columns recording detailed power measurements for each power rail. The power rails (voltage and current) to monitor are defined in the platform definition `JSON` file (see Chapter 5: Platform Definition). The platform definition `JSON` file also contains their respective names, which are defined like those found when using the XRT command `xbutil query`. For more information, refer to the documentation for your card. Typically, current measurements are recorded in mA, voltage in mV and power in W.
- **raw power:** Total measured power. Typically, power measurements are recorded in W.
- **toggle rate:** Toggle rate in % currently set to the power CU.

GT MAC Test Case Description

The goal of this test case is to allow verification of GT transceivers on Alveo cards at 10GbE and 25GbE lane rates. Each GT transceiver supports 4 lanes.

This compute unit (CU) instantiates the 10G/25G High Speed Ethernet Subsystem IP core and allows the core to be configured from a Test `JSON` file (see *10G/25G High Speed Ethernet Subsystem Product Guide* (PG210)). GT MAC traffic is layer 2 type traffic:

- Ethertype: 0x88b5, local traffic only.
- Source and Destination MAC addresses are inserted.
- No IP address is present as there is no layer 3.

In case of multiple CUs, the rate can be select individually per CU, but the selected rate applies to the 4 lanes of the CU.

The CU includes a packet generator, which allows a card with simple electrical or optical loopback cables to generate packets. The CU also verify that the generated packets have been received back, error free. The packet generator can be configured for each lane individually.

Note: Under some circumstances the packet generator might not be able to send packets at the requested rate. This is most likely to occur when operating at 25GbE and with small packets (for example <128-bytes packet). Reducing the number of active MACs and/or increasing the packet size should allow the maximum rate to be achieved (see GT MAC Test JSON Members).

The CU also includes a packet receiver, which counts, per lane, the quantity of packets and bytes received with the expected source and destination MAC addresses.

Main Test Steps

A test is generally composed of four steps and a definition of the hardware environment (see GT MAC Test JSON Members). The following are typical test steps:

1. Configuration
2. Clear status
3. Run
4. Report/check status

WARNING!

- By default, after `xclbin` downloads, the GT MAC CU generates IDLE packets at 25GbE rate.
- When the test sequence is over, the GT MAC CU continues to send traffic as per its last configuration.

GT Test Set Up

GT testing can be achieved by using one of the following methods.

- The use of a QSFP passive electrical loopback module. The module must be compliant to 100GbE (25GbE per lane) and have 0 dB insertion loss. This is the preferred method, the GTs having being validated using a QSFP28 module provided by MultiLane (ML4002-28-C5).

  Note: This module also has the capability of providing a QSFP temperature reading and a programmable power dissipation up to 5W. However, these are not required to pass the GT tests.

- The use of a switch and active optical cables or passive copper cables.
- The use of a QSFP optical module with suitably connected fiber loopback. The module must be compatible with the traffic rate being tested.

  Note: This is an active component the electrical interface between the GTs and the module will need to be validated to ensure optimum performance.

- The use of a protocol analizer with a compatible electrical or optical interface. This is the most complex method of connection as not only will the interfaces require validation with the GTs, the RX and TX paths will be independent. The test `JSON` file needs to be modified to reflect that the RX and TX packet/byte counts might not be the same.

The configuration of the GT MAC test case depends on the set up (see GT MAC Test JSON Members).

Switch Set Up

`xbtest` has been validated with the following Cisco hardware:

- Switch Nexus 3232c, 32 port 100G
- Cables
  - **10/25GbE:** QSFP-100G-CU3M - 100GBASE-CR4 Passive Copper Cable, 3m
  - **25GbE:** QSFP-100G-AOC3M- 100GBASE QSFP Active Optical Cable, 3m

The following is the switch configuration:

- **port 1 - 16:** 10GbE; no FEC
- **port 17 - 32:** 25GbE; clause 74 FEC

The configuration can be obtained via the following command:

```
$ interface breakout module 1 port 1-16 map 10g-4x
$ interface breakout module 1 port 17-32 map 25g-4x
```

Table 21: **Example of switch port configuration**

10GbE Port 1	25GbE Port 17

• **Interface Ethernet1/1/1:**
  • Switchport.
  • Switchport access VLAN 3000.
  • Spanning-tree port type edge.
  • Spanning-tree bpduguard enable.
  • MTU 9216.
  • Speed 10000.
  • Duplex full.
  • No shutdown.
• **Interface Ethernet1/1/2:** Identical to Ethernet1/1/1.
• **Interface Ethernet1/1/3:** Identical to Ethernet1/1/1.
• **Interface Ethernet1/1/4:** Identical to Ethernet1/1/1.
	• **Interface Ethernet1/17/1:**
	  • Switchport.
	  • Switchport access VLAN 3000.
	  • Spanning-tree port type edge.
	  • Spanning-tree bpduguard enable.
	  • MTU 9216.
	  • Speed 25000.
	  • FEC FC-FEC
	  • No shutdown.
	• **Interface Ethernet1/17/2:** Identical to Ethernet1/17/1.
	• **Interface Ethernet1/17/3:** Identical to Ethernet1/17/1.
	• **Interface Ethernet1/17/4:** Identical to Ethernet1/17/1.

***Note*:** The switch port should be on their own VLAN to avoid traffic leak/broadcast.

Source MAC address

The source MAC addresses of the card can be obtained via the command `xbutil dump -d <BDF>`. The GT MAC CU uses all valid source MAC addresses (one per lane). If multiple GT MAC CUs are present in the `xclbin`, they'll split all valid MAC addresses in a round robin manner over all lanes of all GT MAC CU's.

Table 22: **Source MAC Address Round Robin Selection**

MAC Address index	GT/lane index
0	GT[0] Lane 0
1	GT[1] Lane 0
2	GT[0] Lane 1
3	GT[1] Lane 1
4	GT[0] Lane 2
...	...

If there is not enough valid MAC addresses, lanes will be disabled following the same round robin manner. In the table above, if MAC address index 4 is the last one available, GT[0] lane 3 and GT[1] lanes 2/3 will be disabled. It's possible to re-organize the MAC address distribution of the GT via the `source_addr` option. Source MAC addresses are displayed with message ID `ETH_031`.

Destination MAC address - Lane Mapping

The destination MAC addresses are defined via the lane mapping. By default, each lane is configured loop back to itself. So, per lane, the destination MAC address is identical to the Source MAC address. This default mapping is the one to use with loopback module as each lane is physically loopbacked to itself. Destination MAC addresses are displayed with message ID `ETH_031`.

When using a switch, lane traffic can't be loopback to itself, source and destination MAC addresses must be different. `xbtest` only supports paired mapping lanes in order to compare RX status with TX status.

Table 23: **Supported Lane Mapping**

Source	Loopback Module	Switch		
	Default Pairing	Pairing A	Pairing B	Pairing C
Lane[0]	Lane[0]	Lane[1]	Lane[2]	Lane[3]
Lane[1]	Lane[1]	Lane[0]	Lane[3]	Lane[2]
Lane[2]	Lane[2]	Lane[3]	Lane[0]	Lane[1]
Lane[3]	Lane[3]	Lane[2]	Lane[1]	Lane[0]

The mapping is defined using `tx_mapping` option. Here is how the switch pairing A is configured in the test `JSON` file:

```
"lane_config": {
    "0": {
        "tx_mapping": 1
    },
    "1": {
        "tx_mapping": 0
    },
    "2": {
        "tx_mapping": 3
    },
    "3": {
        "tx_mapping": 2
    }
}
```

CAUTION! *When connected to switch, the lane mapping must be defined in pair in order to be able to cross check RX and TX status counters*

Although destination addresses are automatically selected based on `tx_mapping`, it's possible to overwrite it with `dest_addr` member.

GT Settings

As there are multiple ways to connect to the GTs, default settings have been defined. There are 2 modes:

- **module:** For loopback module and active optical cable. Loopback module is actually an electrical track from TX to RX. An active optical cable is terminates the electrical TX track at the optic module input. I also has the electric RX track from the optic module output. Resulting in electrical track length twice the size compared to the loopback module, but from experience, it has not major impact and GT settings can be common for these 2 types (loopback module or active optical cable).
- **cable:** For copper cable

The selection is made using the `gt_settings` member, by default `cable` settings are selected. Each mode defines values for the following GT transceiver settings (see *UltraScale Architecture GTY Transceivers User Guide* (UG578)):

- `rx_equaliser`
- `tx_differential_swing_control`
- `tx_pre_emphasis`
- `tx_post_emphasis`
- `gt_tx_polarity`

The actual values are defined in the platform definition `JSON` file and are also displayed in the `xbtest.log` file with message ID `CMN_021`. It's possible to overwrite these settings for all lanes (included into `global_config`) or selectively for some lanes (part of `lane_config`) via the following test `JSON` file members:

- `gt_tx_diffctrl`
- `gt_tx_pre_emph`
- `gt_tx_post_emph`
- `gt_rx_use_lpm`
- `gt_tx_polarity`

WARNING! *When connected to switch, using a wrong setting for one single lane might result in traffic interruption on all lanes. The switch might try to reset its whole module because it sees that a link is down.*

Status

GT MAC CU provides 3 kind of status/counters:

- A subset of the status registers from the 10G/25G High Speed Ethernet Subsystem IP core (see *10G/25G High Speed Ethernet Subsystem Product Guide* (PG210)). Some status are reported as information (counter registers), others are checked against a null value (error register).
- An indication that the quantity of bytes and packets transmitted is within a range based on the test `duration` and `mode` (lane rate), `utilisation`, and `packet_cfg` configurations.

- A comparison between the quantity of packets and bytes sent by a lane and received by its destination lane. Per lane, the packet receiver checks the source and destination MAC address for each packet received. This is only enabled when `match_tx_rx` is set to `true`.

Matching TX RX

The optional parameter `match_tx_rx` causes comparison between some registers of the 10G/25G High Speed Ethernet Subsystem IP core:

- `RX_TOTAL_GOOD_PACKETS` with `TX_TOTAL_PACKETS`.
- `RX_TOTAL_GOOD_BYTES` with `TX_TOTAL_BYTES`.

The packet receiver also compares the quantity of packets and bytes sent (`TX_TOTAL_PACKETS`, `TX_TOTAL_BYTES`) with their respective received quantities when source and destination MAC addresses are matching. `tx_mapping` option defines which lane values are compared together.

In addition, if the `RX_TOTAL_GOOD_PACKETS` count is equal to 0, then an error message ID `ETH_007` is reported to inform that no good packets were received, and the test will fail.

MAC_STAT Status Registers Description

Each time the `check_status` command is executed, the following registers are read from the MAC hardware for each active MAC lane. All registers are stored in hardware using 48 bits, and extended to 64 bits when read by software.

***Note*:** The 48-bit counters `RX_TOTAL_BYTES`, `RX_TOTAL_GOOD_BYTES` and `TX_TOTAL_BYTES` could saturate after approximately 25 hours of maximum rate operation at 25GbE. It is therefore recommended that the test duration does not exceed 24 hours between two `check_status` commands.

In the table following table, the register type column represents how the register is verified:

- **Check:** The content of the register must be null. Any other value will generate an error (message ID `ETH_004`).
- **Info:** The content of the register is displayed as information (no verification performed).

Table 24: **MAC_STAT Status Description**

Register Name	Register Type	Description
`CYCLE_COUNT`	Info	Number of transceiver clock domain cycles (approximately 1.5625e8 / sec at 10GbE and 3.90612e8 / sec at 25GbE). ***Note*:** A value of 0 means that clocks were not active during the test and other registers should be ignored. Any non-zero result indicates the clocks were active.
`FEC_INC_CANT_CORRECT_COUNT`	Check	This count indicates how many uncorrected bit errors in the corresponding Clause 74 FEC Frame.

Table 24: **MAC_STAT Status Description** *(cont'd)*

Register Name	Register Type	Description
FEC_INC_CORRECT_COUNT	Check	This count indicates how many corrected bit errors in the corresponding Clause 74 FEC Frame.
RX_BAD_CODE	Check	This count indicates how many cycles the RX PCS receive state machine is in the RX_E state as defined by IEEE Std. 802.3.
RX_BAD_FCS	Check	The value of this count indicates packets received with a bad FCS, but not a stomped FCS. A stomped FCS is defined as the bitwise inverse of the expected good FCS.
RX_BROADCAST	Info	Increment for good broadcast packets.
RX_ERROR	Check	This count indicates a mismatch occurred for the test pattern in the RX core.
RX_FRAGMENT	Check	Increment for packets shorter than stat_rx_min_packet_len with bad FCS.
RX_FRAMING_ERR	Check	This count is used to keep track of sync header errors. The stat_rx_framing_err output indicates how many sync header errors were received.
RX_INRANGEERR	Check	Increment for packets with Length field error but with good FCS.
RX_JABBER	Check	Increment for packets longer than ctl_rx_max_packet_len with bad FCS.
RX_MULTICAST	Info	Increment for good multicast packets.
RX_OVERSIZE	Check	Increment for packets longer than ctl_rx_max_packet_len with good FCS.
RX_PACKET_64_BYTES	Info	Increment for good and bad packets received that contain 64 bytes.
RX_PACKET_65_127_BYTES	Info	Increment for good and bad packets received that contain 65 to 127 bytes.
RX_PACKET_128_255_BYTES	Info	Increment for good and bad packets received that contain 128 to 255 bytes.
RX_PACKET_256_511_BYTES	Info	Increment for good and bad packets received that contain 256 to 511 bytes.
RX_PACKET_512_1023_BYTES	Info	Increment for good and bad packets received that contain 512 to 1,023 bytes.
RX_PACKET_1024_1518_BYTES	Info	Increment for good and bad packets received that contain 1,024 to 1,518 bytes.
RX_PACKET_1519_1522_BYTES	Info	Increment for good and bad packets received that contain 1,519 to 1,522 bytes.
RX_PACKET_1523_1548_BYTES	Info	Increment for good and bad packets received that contain 1,523 to 1,548 bytes
RX_PACKET_1549_2047_BYTES	Info	Increment for good and bad packets received that contain 1,549 to 2,047 bytes.
RX_PACKET_2048_4095_BYTES	Info	Increment for good and bad packets received that contain 2,048 to 4,095 bytes.
RX_PACKET_4096_8191_BYTES	Info	Increment for good and bad packets received that contain 4,096 to 8,191 bytes.
RX_PACKET_8192_9215_BYTES	Info	Increment for good and bad packets received that contain 8,192 to 9,215 bytes.
RX_PACKET_BAD_FCS	Check	Increment for packets between 64 and ctl_rx_max_packet_lenbytes that have FCS errors.

Table 24: **MAC_STAT Status Description** *(cont'd)*

Register Name	Register Type	Description
RX_PACKET_LARGE	Info	Increment for all packets that are more than 9,215 bytes long.
RX_PACKET_SMALL	Check	Increment for all packets that are less than 64 bytes long. Packets that are less than 4 bytes are dropped.
RX_PAUSE	Info	Increment for 802.3x Ethernet MAC Pause packet with good FCS.
RX_RSFEC_CORRECTED_CW_INC	Check	This count will increment if the RS-FEC decoder detected and corrected a bit errors in the corresponding frame.
RX_RSFEC_ERR_COUNT0_INC	Check	Increment for RS-FEC detected errors.
RX_RSFEC_UNCORRECTED_CW_INC	Check	This count will increment if the RS-FEC decoder detected uncorrectable bit errors in the corresponding frame.
RX_STOMPED_FCS	Check	The value of this count indicates packets were received with a stomped FCS. A stomped FCS is defined as the bitwise inverse of the expected good FCS.
RX_TEST_PATTERN_MISMATCH	Check	This count indicates how many mismatches occurred for the test pattern in the RX core.
RX_TOOLONG	Check	Increment for packets longer than `ctl_rx_max_packet_len` with good and bad FCS.
RX_TOTAL_BYTES	Info	Increment for the total number of bytes received.
RX_TOTAL_GOOD_BYTES	Info	Increment for the total number of good bytes received. This value is only non-zero when a packet is received completely and contains no errors.
RX_TOTAL_GOOD_PACKETS	Info	Increment for the total number of good packets received. This value is only non-zero when a packet is received completely and contains no errors.
RX_TOTAL_PACKETS	Info	Increment for the total number of packets received.
RX_TRUNCATED	Check	This count indicates that the number of packets truncated due to their length exceeding `ctl_rx_max_packet_len[14:0]`.
RX_UNDERSIZE	Check	Increment for packets shorter than `stat_rx_min_packet_len` with good FCS.
RX_UNICAST	Info	Increment for good unicast packets.
RX_USER_PAUSE	Info	Increment for priority-based pause packets with good FCS.
RX_VLAN	Info	Increment for good 802.1Q tagged VLAN packets.
TX_TOTAL_BYTES	Info	Increment for the total number of bytes transmitted by the packet generator.
TX_TOTAL_PACKETS	Info	Increment for the total number of packets transmitted by the packet generator.

GT MAC Test JSON Members

Following are examples of GT MAC test cases. Some test `JSON` members can be overwritten for each lane using the test `JSON` member `lane_config` which child members are lane indexes.

Electrical/Optical Loopback Example

***Note*:** The default TX/RX lane mapping is used with loopback module.

```
"gt_mac": {
    "0": {
        "global_config": {
            "utilisation"  : 100,
            "match_tx_rx"  : true,
            "gt_settings"  : "module",
            "test_sequence": [
                [1,  "conf_25gbe_rs_fec"],
                [1,  "clear_status"],
                [60, "run"],
                [1,  "check_status"]
            ]
        }
    }
}
```

Switch Example

***Note*:** Lane pairing must be used when connected to a switch.

Here is an example of 2 GT CUs:

- GT[0] is connected to a 10GbE port with a fixed packet size of 1024 bytes

- GT[1] is a 25GbE with the default `sweep` packet size

```
"gt_mac": {
    "0": {
        "global_config": {
            "utilisation" : 99,
            "match_tx_rx" : true,
            "packet_cfg" : "1024",
            "test_sequence" : [
                [1,  "conf_10gbe_no_fec"],
                [1,  "clear_status"],
                [60, "run"],
                [1,  "check_status"]
            ]
        },
        "lane_config": {
            "0": {
                "tx_mapping": 1
            },
            "1": {
                "tx_mapping": 0
            },
            "2": {
                "tx_mapping": 3
            },
            "3": {
                "tx_mapping": 2
            }
        }
    },
    "1": {
        "global_config": {
            "utilisation" : 99,
            "match_tx_rx" : true,
            "test_sequence" : [
                [1,  "conf_25gbe_c74_fec"],
                [1,  "clear_status"],
                [60, "run"],
                [1,  "check_status"]
            ]
        },
        "lane_config": {
            "0": {
                "tx_mapping": 1
            },
            "1": {
                "tx_mapping": 0
            },
            "2": {
                "tx_mapping": 3
            },
            "3": {
                "tx_mapping": 2
            }
        }
    }
}
```

Definition

The following table shows all members available for this test case. More details are provided for each member in the subsequent sections.

Table 25: **GT MAC Test JSON Members**

Member	Lane override	Mandatory/ Optional	Description
test_sequence	no	mandatory	Describes the sequence of tests to perform.
gt_settings	no	optional	Select GT settings for all lanes.
tx_mapping	only	optional	Specify lane mapping. It defines: • Destination MAC address. • TX lane index which will be checked against RX status.
disable_lane	only	optional	Disable a lane.
source_addr	only	optional	Overwrite default source MAC address - lane mapping.
dest_addr	only	optional	Overwrite default destination MAC address.
utilisation	yes	optional	Transmit utilization.
traffic_type	yes	optional	Define the content of the payload area of the packets.
packet_cfg	yes	optional	Define the packet length.
match_tx_rx	yes	optional	Enable RX and TX packet count match when loopback is present.
gt_tx_diffctrl	yes	optional	Configure the GTY transceiver `TXDIFFCTRL` input to transmitter. See *UltraScale Architecture GTY Transceivers User Guide* (UG578) for details.
gt_tx_pre_emph	yes	optional	Configure the GTY transceiver `TXPRECURSOR` input to transmitter. See *UltraScale Architecture GTY Transceivers User Guide* (UG578) for details.
gt_tx_post_emph	yes	optional	Configure the GTY transceiver `TXPOSTCURSOR` input to transmitter. See *UltraScale Architecture GTY Transceivers User Guide* (UG578) for details.
gt_rx_use_lpm	yes	optional	Configure the GTY transceiver `RXLPMEN` input to transmitter. See *UltraScale Architecture GTY Transceivers User Guide* (UG578) for details.
gt_tx_polarity	yes	optional	Configure the GTY transceiver `TXPOLARITY` input to transmitter. See *UltraScale Architecture GTY Transceivers User Guide* (UG578) for details.

test_sequence

Mandatory. Describes the sequence of tests to perform. Tests are performed serially, and a failure in one test does not stop the sequence (the next test will be launched).

This field contains a list of tests, each test being defined by a list of parameters: `[ [], [], [] ]`

The values are interpreted as: `[duration, command]`

- **duration:** The duration of the test in seconds: range [1, $2^{32}-1$].
- **command:** See the following table.

Table 26: `test_sequence` **Command Possible Values**

Command	Description

conf_<mode>	Apply the settings specified in the configuration parameters to the MAC hardware. Part of the configuration process is to issue a reset to the MAC hardware, so the `conf_<mode>` operation will always result in the Ethernet link dropping and restarting, even if the configuration is identical to the previous test.
	Configurations are available for various lane rates and FEC modes:
	• **Lane rates:**
	  • **10GbE:** 10.3125 Gb/s.
	  • **25GbE:** 25.78125 Gb/s.
	• **FEC modes:**
	  • **none:** Disables both FEC options, and uses 66-b words with 2-bit sync headers.
	  • **clause_74:** Enables the Forward Error Correction mode specified in IEEE 802.3 Clause 74. It can be used for both 10GbE and 25GbE lane rates.
	  • **rs_fec:** Enables the Forward Error Correction mode specified in IEEE 802.3by Clause 91. It can only be used in 25GbE mode. If selected in 10GbE mode, it will be ignored, and the link will operate in 66b mode (equivalent to none).
	Possible values of **conf_<mode>** are:
	• **conf_10gbe_no_fec:** Lane rate: `10GbE`, FEC mode: `none`.
	• **conf_10gbe_c74_fec:** Lane rate: `10GbE`, FEC mode: `clause_74`.
	• **conf_25gbe_no_fec:** Lane rate: `25GbE`, FEC mode: `none`.
	• **conf_25gbe_c74_fec:** Lane rate: `25GbE`, FEC mode: `clause_74`.
	• **conf_25gbe_rs_fec:** Lane rate: `25GbE`, FEC mode: `rs_fec`.
	WARNING! *The switch only supports `conf_10gbe_no_fec` or `conf_25gbe_c74_fec` (see GT Test Set Up)*

clear_status	Read and clear the MAC status registers, but ignore the values returned in the counters. It is intended to be used after a `conf_<mode>` operation to clear the status errors caused by the link dropping and restarting.
run	Enable the packet generator. Any test sequence entry without a run will disable the packet generator. If the final `test_sequence` entry contains a run then packet generation will continue after execution of `xbtest` has terminated
check_status	Read the MAC status registers, and for any MAC instances that have `active_mac` set to `true`, Check for any counter values that indicate an error has occurred, or the received number of packets indicates a fault on the link. If an error is detected the overall test will be flagged as a fail.

An example of a `test_sequence` is:

```
"test_sequence" :[
    [1,"conf_10gbe_no_fec"] ,
    [1,"clear_status"] ,
    [60,"run"],
    [1,"check_status"]
]
```

This will:

- Apply the configuration to the MACs, reset them and wait for 1 seconds.
- Clear the status registers and wait for 1 seconds.
- Start the packet generators for any active MACs and wait for 60 seconds.
- Read the status registers and check the results. Then, clear the status registers, stop the packet generator and wait for 1 second before exiting.

gt_settings

Optional. Type: string. Possible values: `module` or `cable`; default: `cable`.

This configuration can only be applied to all lanes of the GT MAC CU at the `global_config` level.

Select the default settings for the GT transceivers (see GT Settings).

tx_mapping

Optional. Type: integer. Possible values: `0` to `3`; default: `n` where `n` represents the lane index within the `lane_config` (so the default is equivalent to a lane loopback to itself).

Specifies the lane index of the n^{th} TX which will be checked against RX status.

This configuration can only be applied individually to one or more of the four lanes connected to the individual transceivers at the `lane_config` level.

disable_lane

Optional. Type: boolean. Possible values: `true` or `false`; default: `false`

This configuration can only be applied individually to one or more of the four lanes connected to the individual transceivers at the `lane_config` level.

When `false` the packet generator of the selected lane will be enabled, and receiver statistics will be gathered and used to determine the overall pass/fail result of the test.

When `true` no packets are generated by the selected lane, and receiver statistics from that instance are ignored.

source_addr

Optional. Type: string. Possible values: `board_mac_addr_<i>`

This configuration can only be applied individually to one or more of the four lanes connected to the individual transceivers at the `lane_config` level.

This option allows the overwrite of the default round robin MAC address vs. lane mapping.

All available MAC addresses are listed in `summary.log` (or in `xbtest.log`) file via the message ID `ETH_036`:

```
INFO :: ETH_030 :: GT_MAC[0] :: Board MAC address list:
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_0: 00:0A:35:06:9F:D2
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_1: 00:0A:35:06:9F:D3
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_2: 00:0A:35:06:9F:D4
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_3: 00:0A:35:06:9F:D5
```

In this particular case, there are 4 addresses listed, so the possible values for `source_addr` are `board_mac_addr_0/1/2/3`.

TIP: *Using `board_mac_addr_<i>` address keeps the test generic across different cards as the actual value of the address is not used in the test `JSON` file.*

dest_addr

Optional. Type: string. Possible values: `board_mac_addr_<i>` or any valid six-octet value (for example: `01:0A:BC:DE:F0:20`).

This configuration can only be applied individually to one or more of the four lanes connected to the individual transceivers at the `lane_config` level.

Although the destination address is automatically selected defining `tx_mapping`, it is possible to overwrite the selection and use another board address or any valid MAC address.

All available MAC addresses are listed in `summary.log` (or in `xbtest.log`) file via the message ID `ETH_036`:

```
INFO :: ETH_030 :: GT_MAC[0] :: Board MAC address list:
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_0: 00:0A:35:06:9F:D2
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_1: 00:0A:35:06:9F:D3
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_2: 00:0A:35:06:9F:D4
INFO :: ETH_036 :: GT_MAC[0] :: -board_mac_addr_3: 00:0A:35:06:9F:D5
```

In this particular case, there are 4 addresses listed, so the possible values for `dest_addr` are `board_mac_addr_0/1/2/3`.

TIP: *Using `board_mac_addr_<i>` addresses keeps the test generic across different cards as the actual value of the address is not used in the test `JSON` file.*

The `dest_addr` overwrite also supports any valid MAC address. The address must be composed of 6 octets separated by columns ":". Only hexadecimal characters are supported. For example: `01:0A:BC:DE:F0:20`.

utilisation

Optional. Type: integer. Possible values: from `0` to `100`; default: `50`.

This sets the transmit utilization of the link in the range 0 to 100 (percent). The parameter is used to set the approximate link utilization for the packet generator, by adjusting the delay between packets.

Setting the utilization to 100 causes packets to be generated at the maximum achievable rate.

Setting the utilization to 0 disables the packet generator completely.

> **CAUTION!** *Never use a utilization of 100 when connected to the switch. The switch also sends some maintenance packets which will take priority over any traffic, resulting in lost packets.*

traffic_type

Optional. Type: string. Possible values: `0x00`, `0xff`, `count` or `pattern`; default: `count`.

The test packets produced by the traffic generator consist of a statically configured destination address (48 bits), source address (48 bits) and ethertype (16 bits) followed by a payload area and a CRC (32 bits).

The content of the payload area is controlled by this parameter:

- **0x00:** The whole payload area will be filled with bytes of value `0x00`.
- **count:** The payload area will be filled with a byte count sequence. The byte following Ethertype will be `0x00`, the next 0x01, with each successive byte incrementing to `0xff`, and rolling over to `0x00` and repeating to the end of the payload area.
- **pattern:** The payload area will be filled with the pattern (`0x00`, `0x55`, `0xaa`, `0xff`) repeating for the number of bytes in the payload area.
- **0xff:** The whole payload area will be filled with bytes of value `0xff`.

packet_cfg

Optional. Type: string. Possible values: `sweep` or value from `64` to `1535` or from `9000` to `10000`; default: `sweep`.

When `sweep` is used, a sequence of 1455 packets will be generated continuously with sizes between 64 and 1518 bytes

If a single numeric value is used (in the range 64 to 1535 or in the range 9000 to 10000) is supplied, then all generated packets shall be this size.

WARNING!

- Only even values are supported for packet sizes between 9000 and 10000 (jumbo frame).
- The switch only supports jumbo frames up to 9216 bytes (see GT Test Set Up).

***Note*:** Note that the receive MTU is adjusted to match the configured transmit packet size. If the transmit size is:

- Lower than or equal to 1518, then the receive MTU is set to 1518.
- Greater than 1518 but lower than or equal to 9600, then the receive MTU is set to 9600.
- Greater than 9600, then the receive MTU is set to 10000.

match_tx_rx

Optional. Type: boolean. Possible values: `true` or `false`; default: `false`

Specifies if RX is checked against TX.

If the transmit and receive interfaces of each active MAC instance are is looped back, then the transmitted packet and byte counts are expected to exactly match the equivalent received good packet and byte counters. Setting this parameter to `true` enables this comparison to be made and included in the overall pass/fail assessment of the link.

When set to `false`, the comparison of TX and RX counters is not included in the overall pass/fail assessment of the link.

gt_tx_diffctrl

Optional. Type: integer. Possible values: from `0` to `31`; default: defined in the platform definition `JSON` file (see Chapter 5: Platform Definition).

This parameter sets the `TXDIFFCTRL` input to transmitter.

Further details can be found in Chapter 3 of *UltraScale Architecture GTY Transceivers User Guide* (UG578).

gt_tx_pre_emph

Optional. Type: integer. Possible values: from `0` to `31`; default: defined in the platform definition `JSON` file (see Chapter 5: Platform Definition).

This parameter sets the `TXPRECURSOR` input to transmitter.

Further details can be found in Chapter 3 of *UltraScale Architecture GTY Transceivers User Guide* (UG578).

gt_tx_post_emph

Optional. Type: integer. Possible values: from `0` to `31`; default: defined in the platform definition `JSON` file (see Chapter 5: Platform Definition).

This parameter sets the `TXPOSTCURSOR` input to transmitter.

Further details can be found in Chapter 3 of *UltraScale Architecture GTY Transceivers User Guide* (UG578).

gt_rx_use_lpm

Optional. Type: boolean. Possible values: `false` or `true`; default defined in the platform definition `JSON` file (see Chapter 5: Platform Definition).

When set to `false`, the GTY transceiver to use the DFE receive equalizer.

When set to `true`, the GTY transceiver to use the LPM receive equalizer.

Further details can be found in Chapter 3 of *UltraScale Architecture GTY Transceivers User Guide* (UG578).

gt_tx_polarity

Optional. Type: string. Possible values: `normal` or `inverted`; default `normal`.

When set to `normal`, TXP is positive, and TXN is negative.

When set to `inverted`, TXP is negative, and TXN is positive.

Further details can be found in Chapter 3 of *UltraScale Architecture GTY Transceivers User Guide* (UG578).

Output Files

All GT measurements are stored in an output `CSV` file generated in `xbtest` logging directory. The values are stored in `CSV` type format with one column for each information type.

> **IMPORTANT!** *If the `-L` command line option is used while calling the application software, no output files are generated.*

For each GT MAC CU, one file is generated per lane with the suffix/extension `_gt<gt_cu_index>_<lane_index>.csv` where:

- `<gt_cu_index>` is the index of the GT MAC CU.

- `<lane_index>` is the index of the lane.

For each test of the `test_sequence`, a new row containing the test results and status is present in this file. All columns present in the file are defined as:

- **Overall result:** Set to `FAIL` as soon as one test fails, otherwise set to `PASS`.
- **Test result:** Set to `FAIL` if the current test fails, otherwise set to `PASS`.
- **Status:** This group of columns is composed of one column for each status registers (see Status).

Platform Definition

The same `Test` software is used for all supported platforms. It is automatically configured at run-time based on the targeted platform using:

- Platform definition `JSON` file.
- `xclbin`.
- `xbutil dump` command

As part of this platform specific customization, the supported parameters a user can specify in the test `JSON` file are computed. For example, no GT MAC test case can be run if there is no GT defined for the platform under test.

TIP: *All platform definition parameters are displayed in `xbtest.log` (as information with the header `XBT_SW_CFG`).*

Platform definition `JSON` file

`xbtest` uses a platform definition `JSON` file specific to each deployment platform. It specifies characteristics, default setting. limits and pass/fail criteria of `xbtest` for a platform. It defines for example:

- Power and temperature sources definition.
- Default GT settings.
- Default bandwidth thresholds of the DMA test case.
- Default bandwidth thresholds of the memory test case.
- Nominal rates for the memory test case.

Other settings are set with default values internally within the `Test` software.

The platform definition `JSON` file is part of the platform library package (see Installed Content) and the specification of its content and structure of is beyond the scope of this document.

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. Alveo Card Documentation
2. *Vivado Design Suite: AXI Reference Guide* (UG1037)
3. *10G/25G High Speed Ethernet Subsystem Product Guide* (PG210)
4. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
5. *AXI High Bandwidth Controller LogiCORE IP Product Guide* (PG276)

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2019–2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.