DMA Test Guide

Andy Shevchenko <andriy.shevchenko@linux.intel.com>

This small document introduces how to test DMA drivers using dmatest module.

The dmatest module tests DMA memcpy, memset, XOR and RAID6 P+Q operations using various lengths and various offsets into the source and destination buffers. It will initialize both buffers with a repeatable pattern and verify that the DMA engine copies the requested region and nothing more. It will also verify that the bytes aren't swapped around, and that the source buffer isn't modified.

The dmatest module can be configured to test a specific channel. It can also test multiple channels at the same time, and it can start multiple threads competing for the same channel.

> **Note**
>
> The test suite works only on the channels that have at least one capability of the following: DMA_MEMCPY (memory-to-memory), DMA_MEMSET (const-to-memory or memory-to-memory, when emulated), DMA_XOR, DMA_PQ.

> **Note**
>
> In case of any related questions use the official mailing list dmaengine@vger.kernel.org.

Part 1 - How to build the test module

The menuconfig contains an option that could be found by following path:

> Device Drivers -> DMA Engine support -> DMA Test client

In the configuration file the option called CONFIG_DMATEST. The dmatest could be built as module or inside kernel. Let's consider those cases.

Part 2 - When dmatest is built as a module

Example of usage:

```
% modprobe dmatest timeout=2000 iterations=1 channel=dma0chan0 run=1
```

...or:

```
% modprobe dmatest
% echo 2000 > /sys/module/dmatest/parameters/timeout
% echo 1 > /sys/module/dmatest/parameters/iterations
% echo dma0chan0 > /sys/module/dmatest/parameters/channel
% echo 1 > /sys/module/dmatest/parameters/run
```

...or on the kernel command line:

```
dmatest.timeout=2000 dmatest.iterations=1 dmatest.channel=dma0chan0 dmatest.run=1
```

Example of multi-channel test usage (new in the 5.0 kernel):

```
% modprobe dmatest
% echo 2000 > /sys/module/dmatest/parameters/timeout
% echo 1 > /sys/module/dmatest/parameters/iterations
% echo dma0chan0 > /sys/module/dmatest/parameters/channel
% echo dma0chan1 > /sys/module/dmatest/parameters/channel
% echo dma0chan2 > /sys/module/dmatest/parameters/channel
% echo 1 > /sys/module/dmatest/parameters/run
```

> **Note**
>
> For all tests, starting in the 5.0 kernel, either single- or multi-channel, the channel parameter(s) must be set after all other parameters. It is at that time that the existing parameter values are acquired for use by the thread(s). All other parameters are shared. Therefore, if changes are made to any of the other parameters, and an additional channel specified, the (shared) parameters used for all threads will use the new values. After the channels are specified, each thread is set as pending. All threads begin execution when the run parameter is set to 1.

> **Hint**
>
> A list of available channels can be found by running the following command:
>
> ```
> % ls -1 /sys/class/dma/
> ```

Once started a message like " dmatest: Added 1 threads using dma0chan0" is emitted. A thread for that specific channel is created and is now pending, the pending thread is started once run is to 1.

Note that running a new test will not stop any in progress test.

The following command returns the state of the test.

```
% cat /sys/module/dmatest/parameters/run
```

To wait for test completion userpace can poll 'run' until it is false, or use the wait parameter. Specifying 'wait=1' when loading the module causes module initialization to pause until a test run has completed, while reading /sys/module/dmatest/parameters/wait waits for any running test to complete before returning. For example, the following scripts wait for 42 tests to complete before exiting. Note that if 'iterations' is set to 'infinite' then waiting is disabled.

Example:

```
% modprobe dmatest run=1 iterations=42 wait=1
% modprobe -r dmatest
```

...or:

```
% modprobe dmatest run=1 iterations=42
% cat /sys/module/dmatest/parameters/wait
% modprobe -r dmatest
```

Part 3 - When built-in in the kernel

The module parameters that is supplied to the kernel command line will be used for the first performed test. After user gets a control, the test could be re-run with the same or different parameters. For the details see the above section Part 2 - When dmatest is built as a module.

In both cases the module parameters are used as the actual values for the test case. You always could check them at run-time by running

```
% grep -H . /sys/module/dmatest/parameters/*
```

Part 4 - Gathering the test results

Test results are printed to the kernel log buffer with the format:

```
"dmatest: result <channel>: <test id>: '<error msg>' with src_off=<val> dst_off=<val> len=<val> (<err code
```

Example of output:

```
% dmesg | tail -n 1
dmatest: result dma0chan0-copy0: #1: No errors with src_off=0x7bf dst_off=0x8ad len=0x3fea (0)
```

The message format is unified across the different types of errors. A number in the parentheses represents additional information, e.g. error code, error counter, or status. A test thread also emits a summary line at completion listing the number of tests executed, number that failed, and a result code.

Example:

```
% dmesg | tail -n 1
dmatest: dma0chan0-copy0: summary 1 test, 0 failures 1000 iops 100000 KB/s (0)
```

The details of a data miscompare error are also emitted, but do not follow the above format.

Part 5 - Handling channel allocation

Allocating Channels

Channels do not need to be configured prior to starting a test run. Attempting to run the test without configuring the channels will result in testing any channels that are available.

Example:

```
% echo 1 > /sys/module/dmatest/parameters/run
dmatest: No channels configured, continue with any
```

Channels are registered using the "channel" parameter. Channels can be requested by their name, once requested, the channel is registered and a pending thread is added to the test list.

Example:

```
% echo dma0chan2 > /sys/module/dmatest/parameters/channel
dmatest: Added 1 threads using dma0chan2
```

More channels can be added by repeating the example above. Reading back the channel parameter will return the name of last channel that was added successfully.

Example:

```
% echo dma0chan1 > /sys/module/dmatest/parameters/channel
dmatest: Added 1 threads using dma0chan1
% echo dma0chan2 > /sys/module/dmatest/parameters/channel
dmatest: Added 1 threads using dma0chan2
% cat /sys/module/dmatest/parameters/channel
dma0chan2
```

Another method of requesting channels is to request a channel with an empty string, Doing so will request all channels available to be tested:

Example:

```
% echo "" > /sys/module/dmatest/parameters/channel
dmatest: Added 1 threads using dma0chan0
dmatest: Added 1 threads using dma0chan3
dmatest: Added 1 threads using dma0chan4
dmatest: Added 1 threads using dma0chan5
dmatest: Added 1 threads using dma0chan6
dmatest: Added 1 threads using dma0chan7
dmatest: Added 1 threads using dma0chan8
```

At any point during the test configuration, reading the "test_list" parameter will print the list of currently pending tests.

Example:

```
% cat /sys/module/dmatest/parameters/test_list
dmatest: 1 threads using dma0chan0
dmatest: 1 threads using dma0chan3
dmatest: 1 threads using dma0chan4
dmatest: 1 threads using dma0chan5
dmatest: 1 threads using dma0chan6
dmatest: 1 threads using dma0chan7
dmatest: 1 threads using dma0chan8
```

Note: Channels will have to be configured for each test run as channel configurations do not carry across to the next test run.

Releasing Channels

Channels can be freed by setting run to 0.

Example:

```
% echo dma0chan1 > /sys/module/dmatest/parameters/channel
dmatest: Added 1 threads using dma0chan1
% cat /sys/class/dma/dma0chan1/in_use
1
% echo 0 > /sys/module/dmatest/parameters/run
% cat /sys/class/dma/dma0chan1/in_use
0
```

Channels allocated by previous test runs are automatically freed when a new channel is requested after completing a successful test run.