

Gowin SGDMA Linux Driver & Sample Applications Guide

This package contains a Linux driver and sample applications for Gowin SGDMA.

Please note that this driver and associated software are supplied to give a basic generic reference implementation only. Customers may have specific use-cases and/or requirements for which this driver is not suitable.

Operating System Support

- Operating System Architecture: x86_64
- Linux kernel: 5.15
- Ubuntu: 20.04 LTS

Directory Structure

```
gowin_sgdma_linux_vx.x.x/
|__ gdma/                - This directory contains the Gowin SGDMA kernel module
|                        driver files.
|__ tests/               - This directory contains the following scripts
|                        and directories.
|   |__ load_driver.sh    - This script loads the kernel module and creates the
|                        necessary kernel nodes used by the provided software.
|                        The kernel device nodes will be created under /dev/ with
|                        names starting with 'gdma0_'.
|                        Root permissions will be required to run this script.
|__ tools/               - Contains sample application source code.
|   |__ gdma_rw           - Sample tool reading/writing to/from sgdma device nodes
|                        such as control, user, h2c_0, c2h_0 etc.
|   |__ perf             - Sample tool for evaluating channel performance.
|__ README.md            - This file.
|__ README.pdf           - PDF version of this file.
```

Load Driver

Load the kernel module driver

`load_driver.sh` accepts an optional parameter to select the loading mode (run `./load_driver.sh help` for a summary):

```
cd tests
./load_driver.sh [poll|msi]
```

mode	Description
msi	MSI interrupt mode
poll	Poll mode, no interrupts

Example. Load the driver in MSI interrupt mode:

```
./load_driver.sh msi
```

Load the driver in poll mode:

```
./load_driver.sh poll
```

After the driver is loaded successfully, you will see the following prompt:

```
Loading driver...insmod gdma.ko interrupt_mode=1 ... (msi mode)
Loading driver...insmod gdma.ko poll_mode=1 ... (poll mode)
The kernel module installed correctly and the gdma devices were recognized.
```

Sample Applications

gdma_rw

This program can open any device node and perform read/write operations. It is typically used to access the memory space of the *control* or *user* PCIe BARs, and it can also perform single DMA operations via the *h2c** and *c2h** nodes, where the asterisk (*) indicates the channel index (0-3).

Usage

```
gdma_rw.exe <DEVNODE> <read|write> <ADDR> [OPTIONS] [DATA]
- DEVNODE : control | user | h2c_* | c2h_*,
            where * is a numeric wildcard (0-3),
            or an absolute /dev/gdma0_xxx path.
- ADDR :    Target offset address of the read/write operation.
            Can be in decimal or hexadecimal.
- OPTIONS :
            -a ALIGN buffer alignment (default: PAGE_SIZE)
            -b open file as binary. only use with the -f option below.
            -e C2H: open with O_TRUNC for EOP flush (streaming),
                tolerate underflow (short read) without error
            --no-mmap force read/write for control/user devices
                (default: auto-use mmap when available)
            -c CARD card index for /dev/gdma<card>_* (default 0)
            -f PATH use contents of file at PATH as input or write output
into file
            -l LEN length of data to read/write (default: 4, or file size for
write)
            -v more verbose output
- DATA :   Space-separated byte data in decimal or hex (big endian).
            e.g. for the 4-byte value 0x44332211 (decimal 1144201745),
            DATA can be either: 0x11 0x22 0x33 0x44
            or: 17 34 51 68
```

Examples

Read a 4-byte control register at offset 0x1000:

```
gdma_rw control read 0x1000 -l 4
```

Write 2 bytes (0x1234) to a control register at offset 0x2004:

```
gdma_rw control write 0x2004 0x34 0x12
```

Read 4KB from C2H_0 starting at offset 0 into a binary file (my_data.bin):

```
gdma_rw c2h_0 read 0 -l 0x1000 -b -f my_data.bin
```

Read 4KB of data from C2H channel 0 at offset 0x100:

```
gdma_rw c2h_0 read 0x100 -l 0x1000
```

Write the contents of my_data.bin to H2C channel 0 at offset 0x0:

```
gdma_rw h2c_0 write 0x0 -b -f my_data.bin
```

perf

This program measures H2C and C2H channel performance. You can specify the packet size and test duration.

Usage

```
perf -d <h2c_X|c2h_X> [-l <bytes|K|M|G>] [-t <seconds>] [-c <card>] [-a <align>]  
[-e] [-v]
```

ARGS:

- d h2c_* or c2h_*, where * is a numeric wildcard (0-3), or an absolute /dev/gdma0_xxx path.
- l Packet size per transfer. Supports K/M/G suffixes and hexadecimal.
- t Test duration in seconds
- c Device card index (default 0)
- a Buffer alignment in bytes (default 4096)
- e C2H: open with O_TRUNC for EOP flush (streaming), tolerate underflow without error
- v More verbose output
- h|-? show help

Examples

Test H2C_0 performance with 1MB packets for 5 seconds:

```
perf -d /dev/gdma0_h2c_0 -l 1M -t 5
```

Test C2H_0 performance with 128KB packets for 5 seconds:

```
perf -d /dev/gdma0_c2h_0 -l 128K -t 4
```

Reference Performance

H2C reference:

```
Transferred : 1793064960 bytes in 5.001 s
Packet size : 1048576 bytes
Throughput   : 2.868 Gbps (358.6 MB/s)
Efficiency   : 71.7 % (vs PCIe Gen2 x1 = 4.0 Gbps)
```

C2H reference:

```
Transferred : 2214592512 bytes in 5.001 s
Packet size : 1048576 bytes
Throughput   : 3.542 Gbps (442.8 MB/s)
Efficiency   : 88.6 % (vs PCIe Gen2 x1 = 4.0 Gbps)
```

User Interrupt Events

The driver creates one user event character device per user interrupt: `/dev/gdma0_events_0` to `/dev/gdma0_events_3`. A read on an event device blocks until the corresponding user interrupt is triggered by the FPGA, then returns the 4-byte data. This can be tested with `dd`:

```
dd if=/dev/gdma0_events_0 bs=4 count=1 | xxd
```

The command blocks until a user event occurs. When it returns, the data read is the event data (e.g. `01 00 00 00` for event 0).