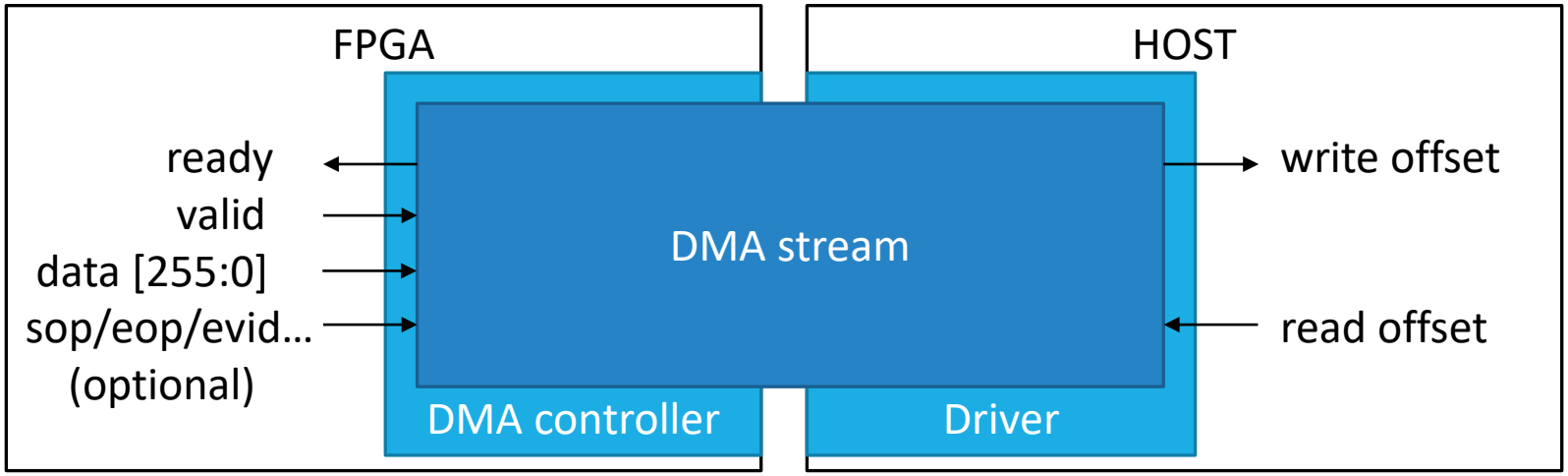
PCIe40: new PCIe fw/sw interface

PAOLO DURANTE

Building blocks

- The main building block of the PCIe module is the **DMA stream**
- Streams are unidirectional (FPGA→HOST, “DAQ” direction)
  - An implementation of the opposite direction exists but it is not tested in production
- A stream can be used as a black box
  - Input: arbitrary bytes + datavalid signal, Output: circular buffer in host memory
  - Multichannel DMA, data scheduling, memory management are hidden by firmware and software
- All the streams on a given PCIe interface are instantiated in a **DMA controller**



Current firmware “flavors”

- **MINIDAQ flavor**
  - 4 DMA streams
    - 2x “MAIN” data stream (FPGA:32 KiB buffer → HOST:4 GiB buffer, 56 Gbps max)
    - 2x “ODIN” data stream (FPGA:4 KiB buffer → HOST:1 GiB buffer, 10 Gbps max)
    - 2x “META” data stream (“special” stream type, see slides on metadata mechanism)
- **TELL40 flavor**
  - 2 DMA streams
    - 2x “MAIN” data stream (32 KiB → 4 GiB buffer, 56 Gbps max)
    - 2x “META” data stream
- **ODIN flavor**
  - 10 DMA streams
    - 2x “ODIN0...ODIN4” data streams (4 KiB → 1 GiB buffer each, 10 Gbps max)
    - 2x “META” data stream
- **NONE flavor**
  - No streams (used for SOL40 firmware that does not use DMA)
  - BAR0/BAR2 for control system (ECS) and low-level interface (LLI)
  - Optional RBAR (“reverse bar”) for ECS offload of specific functions

Stream operating modes

- **Byte mode** (also “raw mode”)
  - Completely data format agnostic
  - FPGA side: avalon streaming input → HOST side: circular memory buffer
- **Packet mode** (also “fragment mode”)
  - Same as above, plus:
  - Avalon streaming data is delimited by SOP/EOP signals (Start/End of Packet)
  - Each packet must have
    - 64-bit **EVID**
    - 8-bit **TYPE**
    - 16-bit **SIZE** (in bytes)
  - Supports truncation compensation and throttle compensation
- **Block mode** (also “MFP mode”)
  - Same as above, plus:
  - Supports fragment realignment and metadata packing

Extra features

- **Truncation compensation**
  - Used in PACKET MODE only
  - If backpressure causes a packet to be cut “in the middle”, automatically adjusts fragment to prevent parsing errors in data stream consumer
- **Throttle compensation**
  - Used in PACKET MODE and BLOCK MODE
  - If backpressure causes packets to be lost, inserts empty packets in the sequence to ensure EVID counter remains strictly monotonic without holes
- **Fragment realignment**
  - Used in BLOCK MODE only
  - Optimizes bandwidth by realigning packets at a smaller boundary than the default
    - default alignment is 256 bits (32 Bytes)
- **Metadata packing**
  - Used in BLOCK MODE only
  - Optimizes bandwidth by compressing EVID counter (saves 64b per packet)
  - Optimizes memory accesses by packing multiple TYPE and SIZE fields together and by precomputing total block size in FPGA

Effective PCIe throughput

Theoretical bandwidth

Packet efficiency

$$\rho = \frac{Lane\ rate \times Lane\ width}{Encoding} \times \frac{MPS}{MPS+Headers}$$

Example: Gen2 x8, 128 Bytes MPS

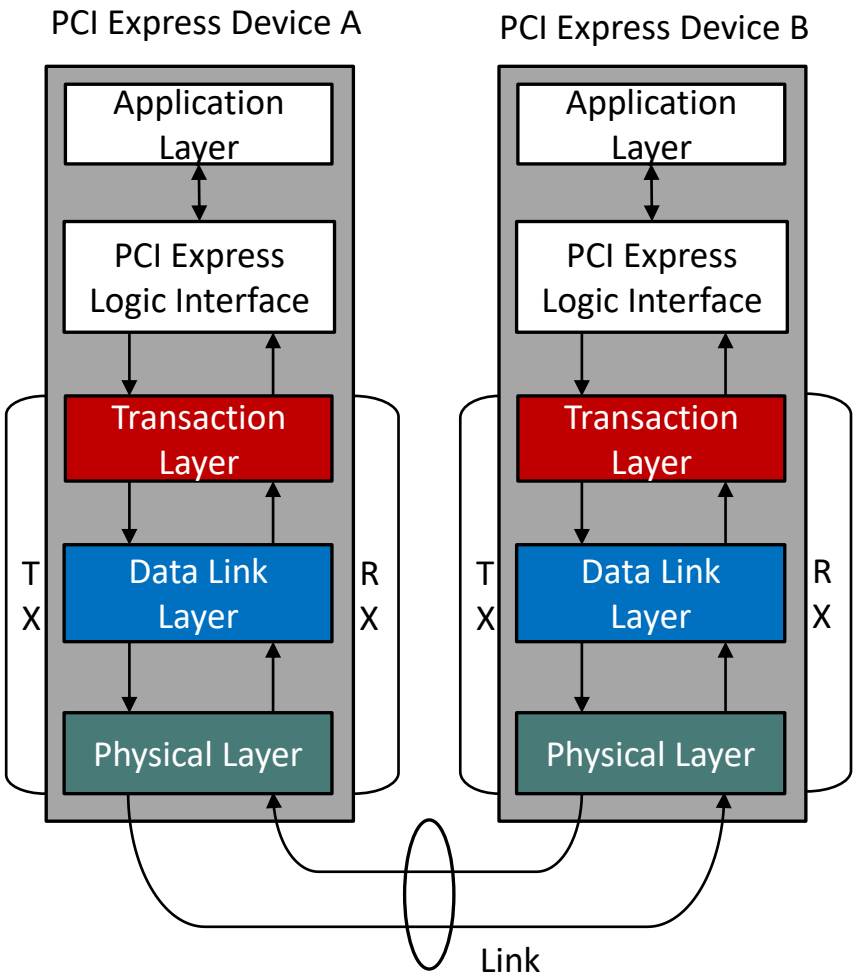
- $\rho = 40 \times 0.8 \times \frac{128}{128+24}$ = 32 x 0.84 = 26.9 Gb/s

Example: Gen3 x8, 128 Bytes MPS

- $\rho = 64 \times 0.98 \times \frac{128}{128+24}$ = 62.7 x 0.84 = 52.6 Gb/s

Example: Gen3 x8, 256 Bytes MPS

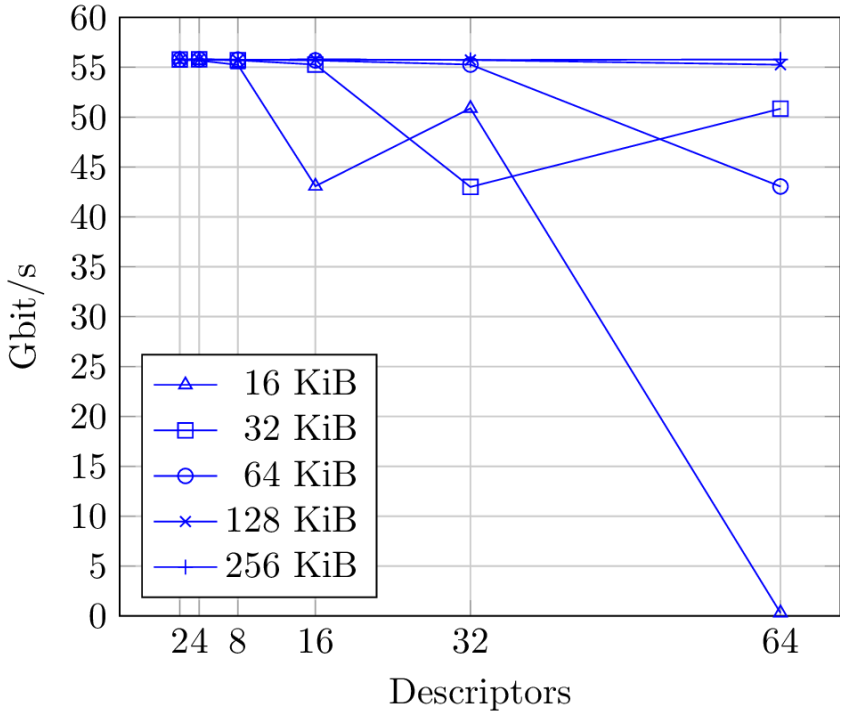
- $\rho = 64 \times 0.98 \times \frac{256}{256+24}$ = 62.7 x 0.91 = **57 Gb/s**



PCIe data throughput

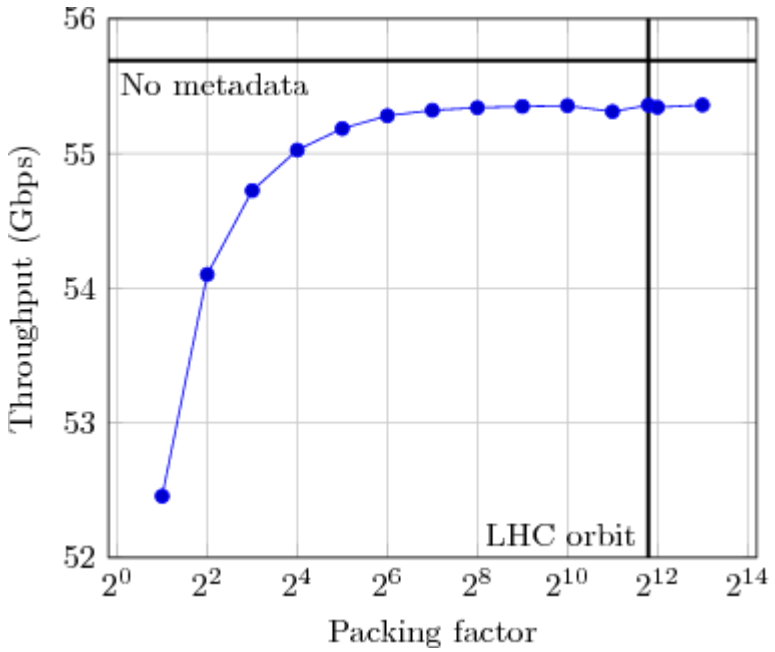
- PCIe transfer parameters already optimized to minimize on-chip memory and maximize throughput.
  - 256 Bytes TLP payload
  - 32 KiB fpga buffer
  - 8 KiB per descriptor
  - 4 GiB host buffer
- ~56 Gbit per second per x8 link
- ~112 Gbit per second per PCIe40

These figures measure how much data can be transferred overall, it's up to us to use all these bits efficiently



PCIe metadata throughput

- Some information in the data is redundant ($EVID_{t+1} = EVID_t + 1$)
  - 64 bit EVID x 40 MHz = 2.56 Gbps
  - **4.5% of max bandwidth**
  - ➢ Group fragments in blocks and store only first EVID
- Some information in the data is expensive (slow) to look up
  - Locating a specific fragment in memory requires linear scanning of the entire stream
  - ➢ Store fragment types and sizes in dedicated lookup tables
- Metadata Packing Factor = number of fragments described by one metadata block
- Metadata throughput < 1% of data

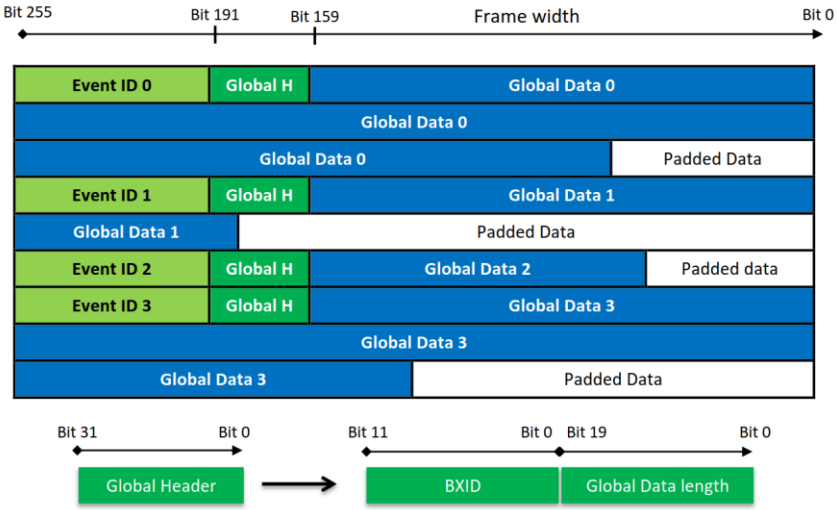


TELL40 throughput example

Data processing example (TDET)

For each accepted bunch crossing:

1. Align all fibers to the same BXID
2. Remove the BXID
3. Concatenate all fibers
4. Pad to 32 bytes
5. Add EVID
6. Truncate (if necessary)
7. Send to output



Stream of fragments

Example (1 MHz no MFP):

12 widebus links @ 1 MHz + 4 GiB buffer

- 100 bits x 12 fibers = 1200 bits
- 1200 bits + 32 bits header = 1232 bits = 154 bytes
- 154 bytes + 8 bytes EVID = 162 bytes
- 162 bytes + 30 bytes padding = 192 bytes
- 192 bytes x 1 MHz = 192 MBps = ~1.5 Gbps
- 4 GiB / 1.5 Gbps = ~23 seconds
- 4 GiB / 192 bytes = ~22.3 Mevents

Example (40 MHz no MFP):

12 widebus links @ 40 MHz + 4 GiB buffer

- 100 bits x 12 fibers = 1200 bits
- 1200 bits + 32 bits header = 1232 bits = 154 bytes
- 154 bytes + 8 bytes EVID = 162 bytes
- 162 bytes + 30 bytes padding = 192 bytes
- 192 bytes x 35 MHz = 7.68 GBps = ~61 Gbps
- 4 GiB / 61 Gbps = ~0.5 seconds
- 4 GiB / 192 bytes = ~22.3 Mevents

Optimizing throughput

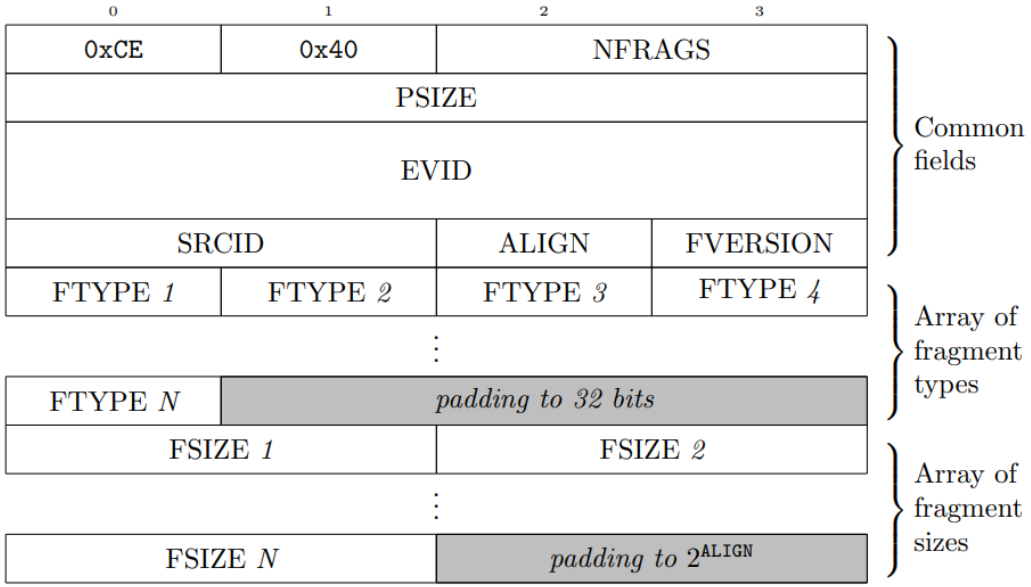
PREVIOUS OUTPUT FORMAT

- EVID repeated in each fragment
- Fragments aligned to 32 bytes
- Fragment types and sizes scattered in memory (header of each fragment)

CURRENT OUTPUT FORMAT

- One EVID per block (MFP)
- Thousands of events per block
- Fragments **realigned** to 16 bytes (half padding overhead)
- Fragment types and sizes can be found in constant time (**lookup tables** in MFP header)

Metadata format (“MFP”)

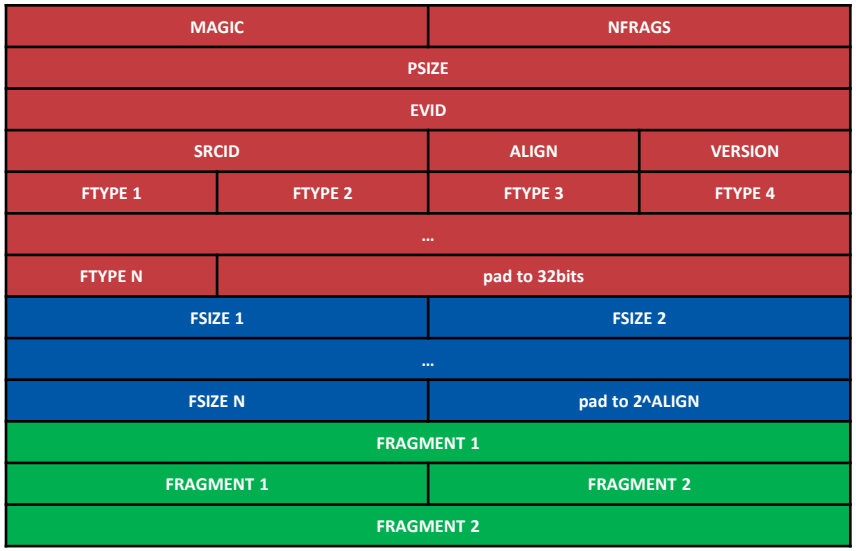


Field	Width (bits)	Meaning
NFRAGS	16	Number of fragments in this packet
PSIZE	32	Size of this packet, including the header, in bytes
EVID	64	Event ID of first fragment in this packet
SRCID	16	Source ID of the fragments in this packet
ALIGN	8	Fragments in this packet are aligned to $2^{\texttt{ALIGN}}$
FVERSION	8	Version of the data format of the fragments
FTYPE n	8	Type of the n-th fragment in the packet
FSIZE n	16	Size in bytes of the n-th fragment in the packet
All fields are represented as unsigned integers in little-endian byte order.		

Table 2: The multiple-fragment packet (MFP) header

Metadata implementation

- dma_stream_daq.vhd implements a DMA stream in the DAQ direction
- dma_stream_mfp.vhd implements the MFP data format
- MFP format can be enabled at runtime by writing a register, in this mode both daq and mfp streams write in parallel to the same circular buffer
  - Metadata stream does not own any host memory
  - Requires careful synchronization between streams and between clock domains to avoid data corruption



Each color represents a “DMA descriptor group”. Descriptor groups write data in parallel.

← “meta_0” DMA descriptor group

← “meta_1” DMA descriptor group

← realigned by “META” stream, but written by “MAIN” or “ODIN” descriptor group

Example (1 MHz MFP):

12 widebus links @ 1 MHz + 4 GiB buffer

- 100 bits x 12 fibers = 1200 bits
- 1200 bits + 32 bits header = 1232 bits = 154 bytes
- 154 bytes ~~**+ 8 bytes EVID**~~ = 154 bytes
- 154 bytes **+ 6 bytes padding** = 160 bytes **(-17% wrt no MFP)**
- 160 bytes x 1 MHz = 160 MBps = ~1.28 Gbps **(-15% wrt no MFP)**
- 4 GiB / 1.28 Gbps = ~27 seconds
- 4 GiB / 160 bytes = ~26.8 Mevents

Example (40 MHz MFP):

12 widebus links @ 40 MHz + 4 GiB buffer

- 100 bits x 12 fibers = 1200 bits
- 1200 bits + 32 bits header = 1232 bits = 154 bytes
- 154 bytes ~~**+ 8 bytes EVID**~~ = 154 bytes
- 154 bytes **+ 6 bytes padding** = 160 bytes **(-17% wrt no MFP)**
- 160 bytes x 40 MHz = 6.4 GBps = 51.2 Gbps **(-17% wrt no MFP)**
- 4 GiB / 51.2 Gbps = ~0.7 seconds
- 4 GiB / 160 bytes = ~26.8 Mevents

Current status

- MFP support is included in latest firmware and software releases
- Fully integrated in EventBuilder software
- Firmware and software support **BOTH** MFP and non-MFP data format
  - MFP -> buffer_type=2 in WinCC RunInfo
  - Non-MFP -> buffer_type=3 in WinCC RunInfo
- Global partition always uses MFP format
  - Local partitions used non-MFP by default, until this week
- Work ongoing in mfp_commissioning branches to improve stability, timing, logic and memory utilization
- RDMA support for Infiniband memory sharing
- Tested up to 20MHz in TDET, up to 10MHz in global
- Next full-scale and full-bandwith test at Point 8 this week

Driver architecture

The driver is split in three “sub-drivers”:

- ID subdriver (/dev/pcie40_*N*_id)
  - Basic device identification
- ECS subdriver (/dev/pcie40_*N*_bar0, /dev/pcie40_*N*_bar2)
  - Control system interface (access to BAR0/BAR2/virtual RBAR)
- DAQ subdriver (/dev/pcie40_*N*_ctrl)
  - Access to DAQ streams

The first two are always available (API is frozen) to ensure the board can always be configured and monitored.

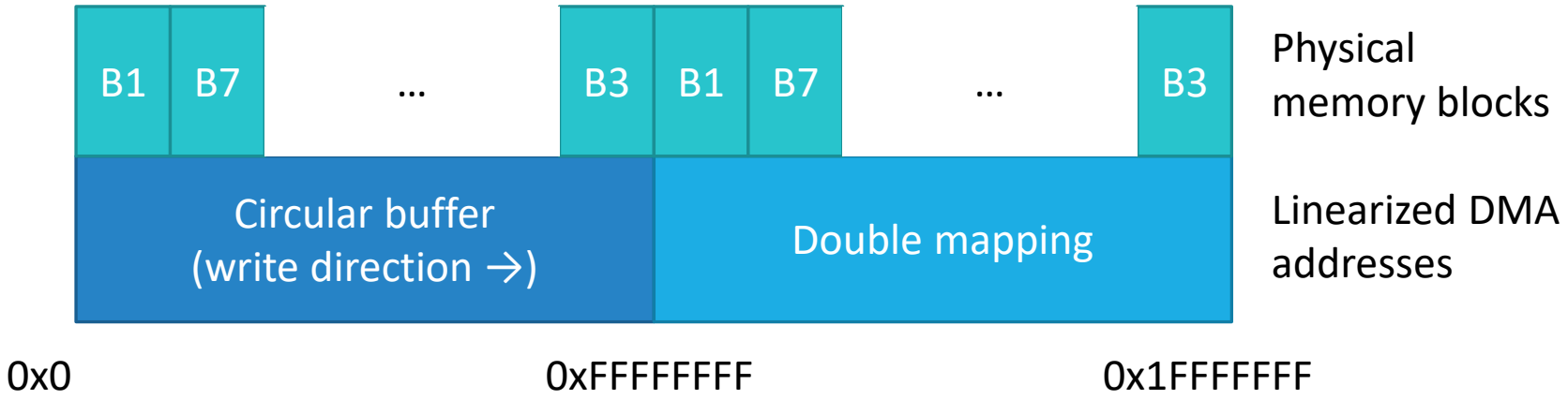
DAQ subdriver will only be loaded if firmware (DMA controller) and software (driver) versions are compatible (driver is backwards compatible and supports older firmwares)

- FW = 6.0 + KD = 6.1: OK
- FW = 6.1 + KD = 6.0: NO
- FW = 6.1 + KD = 6.1: OK

```
Interface: 2
  FPGA: 00-54-01-90-38-85-FE-0C
  Name: tdtel013_0 (tdtel013)
  Source: 0x17804 (version: 1, subsystem: 15 'TDET', number: 4)
  Version: FW 6.1 (2124) / KD 6.1 / SW 6.1 (PASS)
  Link: 0 (23:00.0)
  Leds: off (___, ___)
```

DMA memory management

- DMA memory is allocated in physically contiguous blocks (pci_alloc_consistent)
  - Max block size supported by kernel is 4 MiB
  - We need to buffer a lot more data than what one block can contain
  - Solution: implement “virtual memory map” in FPGA
    - Memory blocks are chained in a linear list, independent from their physical address
    - DMA stream in FPGA dynamically jumps between memory blocks when needed
    - Driver remaps memory so that user software sees the same linear address space as the FPGA
    - Circular buffer in user memory is double mapped to simplify wrap-around logic



Driver RDMA support

- RDMA (Remote DMA) support allows InfiniBand HCAs to read PCIe40 DMA buffers directly without copying data
- Very important optimization for LHCb, but can be useful anywhere
- Available since 6.1
- #define RDMA_SUPPORT in driver to enable (enabled by default)
- Implementation details outside the scope of this presentation, but ask me if necessary