COMP3301 A2 Device Interface

COMP3301 S2 2025

Version $Revision: 526 $

Table of Contents

Notation used in this document

Unless otherwise noted, structure diagrams in this document show offsets from the beginning of the structure in hexadecimal bytes along the left hand side, and offsets from the beginning of each row in bytes along the top.

The structure diagrams shown in e.g. the PCI interface or Descriptor rings sections all have 64-bit (8-byte) rows.

1. Introduction

The COMP3301 A2 device is a virtual PCI device for transporting packets across a simplified network named "Ductnet".

It supports:

- A simple descriptor ring DMA structure
- Full-duplex simultaneous transmit and receive operation
- Dual-interrupt with 64-bit MSI-X support, with always-enabled coalescing

2. Ductnet

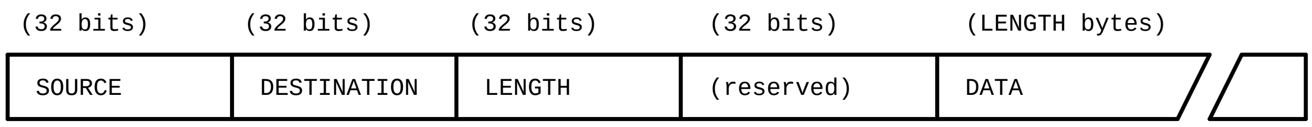
The simple network used in this assignment is an example of a packet-switched network with a bus topology, where all stations on the network can receive all traffic.

Information over the network is exchanged in "packets", which have a defined length, a source address (the station sending the packet) and a destination address (the station or service which should receive the packet).

Packets can be addressed either to a specific other station on the network (unicast) or they can be addressed to a multicast group address (when multiple other stations are expected to receive the packet).

Addresses in Ductnet are 32-bit integers, where the most significant bit is set to indicate a multicast group address.

Packets consist of a header followed by data. The header contains 4 32-bit fields:



All fields are in little-endian byte order on the wire.

Each station on the Ductnet network has its address programmed into the hardware at the time of manufacture (to assist with testing, however, your virtual Ductnet hardware will change address each time you reboot).

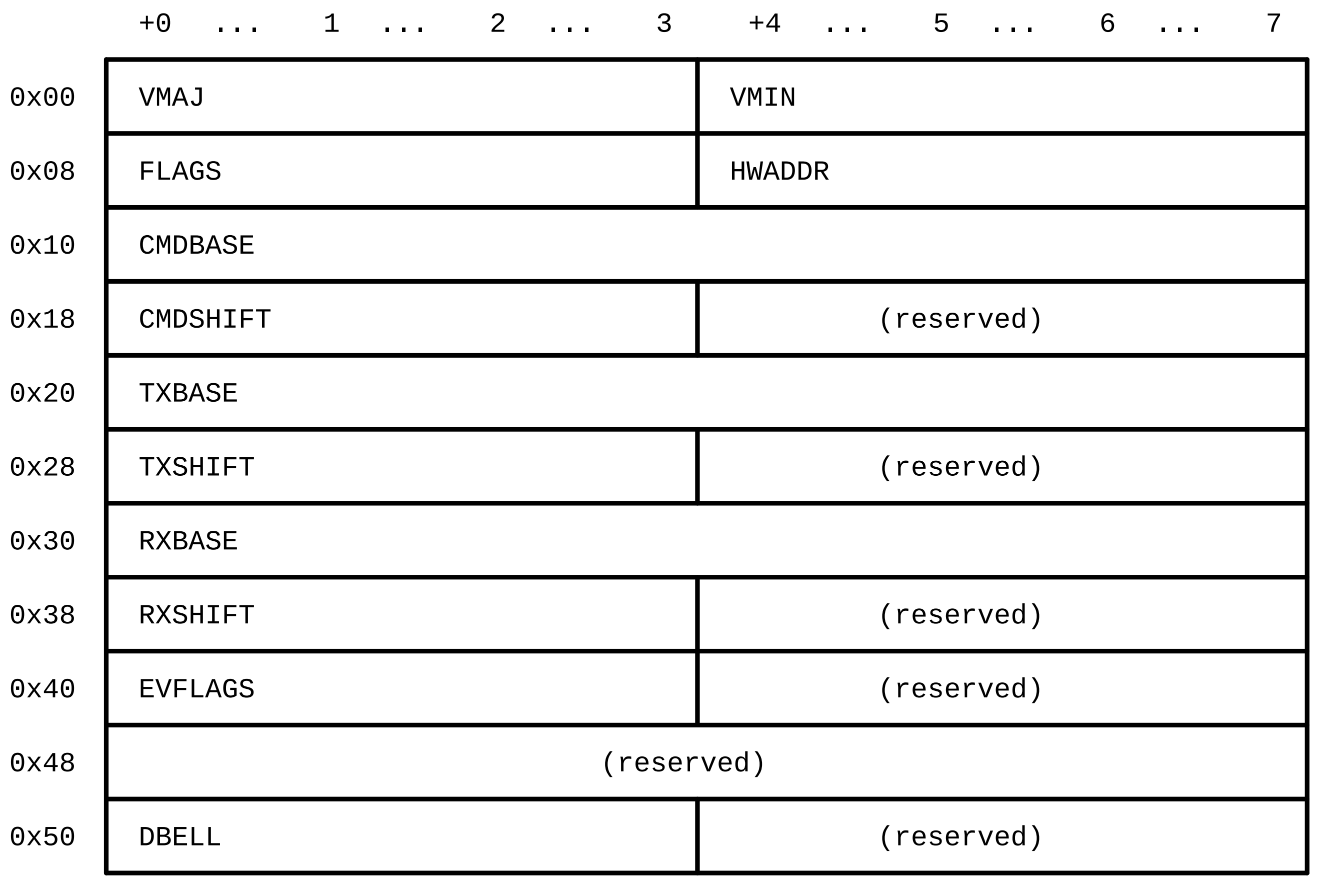
3. PCI interface

All COMP3301 devices use the PCI Vendor ID `0x3301`. The A2 device should be recognised by its use of the device ID `0x2000` in the PCI configuration space Type-0 header.

The A2 device has two PCI BARs:

BAR index	Type-0 offset	Type	Contents
`0`	`0x10`	MEM32	Main memory BAR for operating the device
`1`	`0x18`	MEM32	MSI-X configuration table

The main BAR, of length `0x80`, has the register structure shown below. All other bytes of the BAR not included in this diagram are reserved for future use (RFU). Reserved bytes, including those shown on the diagram must not be read from or written to by the device driver.



The registers `VMAJ` and `VMIN` are 32-bit, read-only, and contain the specification major and minor versions respectively. This specification is major 2, minor 0. You may use the content of this specification to communicate with any device with the same major number as this specification, and any minor number at least as high.

The register `HWADDR` contains the hardware address of this station on the Ductnet network. This is the address which will be inserted into the headers of transmitted packets, and also the address which will appear as the destination of unicast packets received by this station.

The register `FLAGS` contains error flags related to invalid operating conditions for the device. The section Error flags describes the possible values of this register. Once such a condition has been reached, the device will fire MSI-X interrupt vector 1 (the second vector), then it will stop and need to be reset, which is described in Resetting the device.

The registers `CMDBASE` and `CMDSHIFT` contain the base linear (physical) address of the command descriptor ring structure for the device, and the bit shift to calculate its size in number of descriptors (which must be a power of 2). These are to be written by the device driver. This ring is used for control commands (such as setting up and managing filters on the RX ring). See the section Command ring for more details.

The registers `TXBASE` and `TXSHIFT` contain the base linear (physical) address of the transmit descriptor ring structure for the device, and the bit shift to calculate its size. Like `CMDBASE` and `CMDSHIFT`, these are to be written by the device driver. This ring is used to transmit packets out onto the Ductnet network. See the section Descriptor rings for more details.

The registers `RXBASE` and `RXSHIFT` contain the base linear (physical) address of the receive descriptor ring structure for the device, and the bit shift to calculate its size. Like `TXBASE` and `TXSHIFT`, these are to be written by the device driver. This ring is used to provide buffers to the Ductnet hardware for it to write received packets into. See the section Descriptor rings for more details.

The `EVFLAGS` register contains a set of flags which are set by the device to indicate the cause of the last interrupt it issued on the event interrupt vector (MSI-X vector 0). This register is "clear on read", meaning that when read by the host CPU, its value will also be set to zero atomically. As a result, if a subsequent read returns non-zero, then those flags have been set *again* by the device after the previous read completed. See the section Handling events for more details.

The `DBELL` register is written by the device driver to wake up the device when new entries on the command or transmit descriptor rings have been written by the driver and are available for device use. This is described further in [Executing a control command] and Transmitting a packet. The high bit (`0x80000000`) of the `DBELL` register must be set when writing the ring index of a transmit descriptor, and unset when writing the ring index of a command descriptor.

All multi-byte integers processed by the device are stored in little-endian byte order.

4. Interrupt vectors

The device supports two MSI-X interrupt vectors:

MSI vector	When triggered
0	New events have occurred
1	Fatal error; bits in the `FLAGS` register can be used to diagnose

Legacy PCI interrupts and MSI are not supported.

5. Flow of operation

At startup, the driver should check the contents of the `VMAJ` and `VMIN` fields for device compatibility. It should also read the `HWADDR` field so that it may accurately report its hardware address.

Then, it should allocate linear contiguous memory for the command, transmit, receive and event rings, and initialise all descriptors in each.

After initialising the descriptors, it should write the `CMDBASE`/`CMDSHIFT`, `TXBASE`/`TXSHIFT`, and `RXBASE` /`RXSHIFT` registers in the BAR.

To begin normal device operation, including reception of packets, the driver should then produce a `START` command in the command ring. Once this command completes, normal operation has begun.

To transmit a packet, the driver should write the packet into the next available transmit ring descriptor and change its `OWNER` value, then write the ring index of the descriptor into the `DBELL` register.

Once the packet has been transmitted and the buffers can be released, ownership of the descriptor will be changed back to host ownership, and an event will be produced by the device, by setting the `TXCOMP` flag in `EVFLAGS` and triggering an interrupt on MSI-X vector 0.

Before receiving packets, the driver will need to use the `ADDFILT` command to add a receive filter for the device's own unicast hardware address (and any other multicast addresses desired).

To receive packets, the driver should set up a suitable number of descriptors in the receive descriptor ring to point at buffers it has allocated for packet data. When a packet is received, the device will look at the next descriptor in the receive ring. If it is set to device ownership, it will write packet data into the referenced buffer, then set the descriptor to host ownership and produce an event interrupt with the `RXCOMP` flag set.

At each event interrupt, the driver should look at the `EVFLAGS` register descriptor to determine which ring or rings caused the interrupt. On each ring, the driver should begin at the last position the device wrote to, and continue processing and checking any subsequent completions available until it finds one set to device ownership.

Once it has processed all the rings requiring attention, it should return and await the next interrupt.

To stop normal packet operation, the device should produce a `STOP` command in the command ring. Once this command completes, all ring polling for receive and transmit rings, and packet-related activity has ceased. The driver can safely clear the contents of the transmit and receive descriptor rings and release any buffers.

If issuing a subsequent `START` command to restart normal operation later, the driver must return the transmit and receive rings to their initial state before producing the command.

6. Descriptor rings

Each descriptor ring for the A2 device consists of a region of contiguous linear memory allocated by the device driver. Each ring's address is written into the `TXBASE`, `RXBASE` or `CMDBASE` register in BAR 0, and its size (as a bit shift value) is written into `TXSHIFT`, `RXSHIFT` or `CMDSHIFT`.

Each ring consists of a power of two number of descriptors. For the transmit and receive rings, each entry is 64 bytes in length. The device driver is expected to initialise the entire ring before writing to the relevant `BASE` register (see Initial descriptor state below).

The size of the ring is written into the `SHIFT` register as a bit shift value. For example, the driver has allocated a transmit ring made up of a single 4k page, which has 64 descriptors. The driver would write the number `6` into the `TXSHIFT` register for this ring, as `1 << 6 == 64`.

The `OWNER` field of each descriptor is set to indicate whether the host or device is currently in charge of each descriptor. A descriptor with `OWNER` set to the device ownership value on the transmit ring indicates a packet ready for transmission. On the receive ring, a descriptor with `OWNER` set to the host ownership value indicates a received packet ready for the driver to pick up.

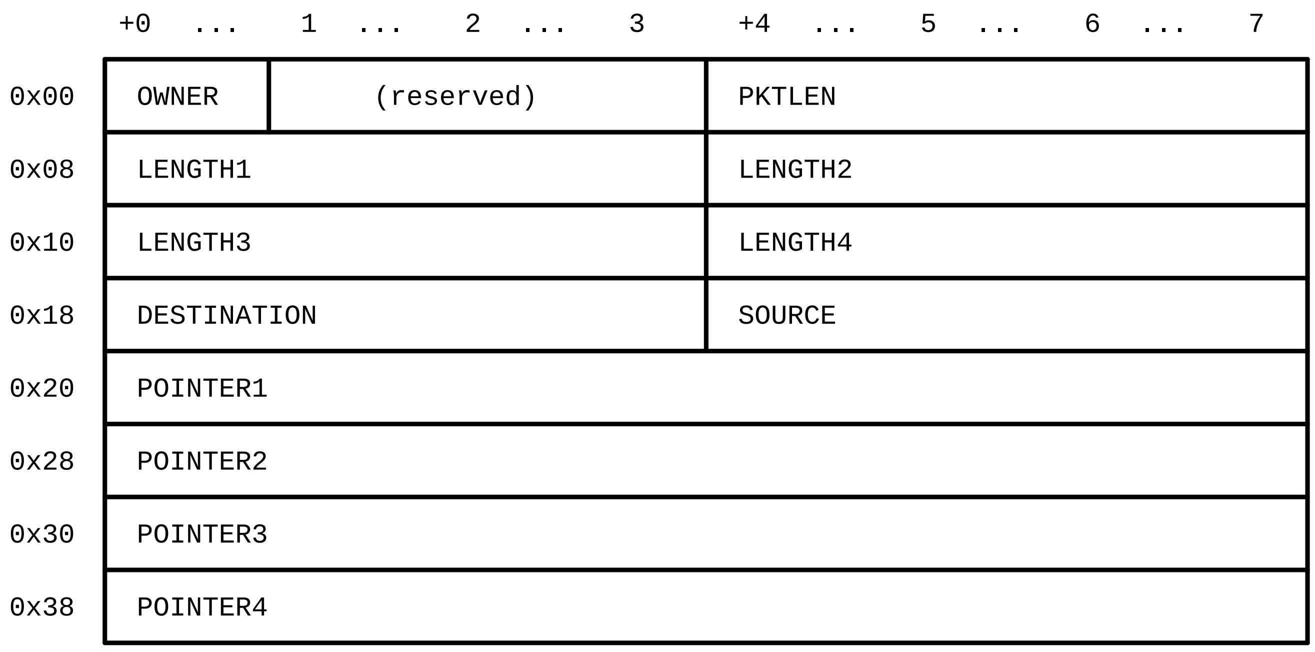
Descriptors in the ring are used in ascending order beginning at index 0 (the first descriptor). It is a circular buffer, meaning that when the end of the ring is reached, the producer "wraps" around to index 0 again (hence the term "ring"). The consumer is only ever expected to read from the next descriptor index after the last one they consumed.

On the transmit ring, as soon as a descriptor has been set to device ownership, the device may begin transmitting the packet immediately (before any write to the `DBELL` register in BAR 0, since the device may be polling the ring). It is important to ensure that all writes to other fields of the descriptor are flushed out to memory before writing to the `OWNER` field.

Note that both transmit and receive descriptors will be used strictly in order: i.e., the device will not transmit the packet in descriptor 2 until the packet in descriptor 1 has been transmitted and its `OWNER` field updated.

6.1. Transmit and receive descriptor

The descriptor structure for command and reply rings is shown below:



The `OWNER` field should be set either to the value `DEVICE` (`0x55`) or `HOST` (`0xAA`).

The `PKTLEN` field is written by the device on receive descriptors to indicate the total length of the received packet. It is not used for transmit descriptors.

The `DESTINATION` field is used for both transmit and receive descriptors. On the transmit ring, it is used to specify the destination the packet is being sent to (and will be written into the header transmitted by the device). On the receive ring, it will be filled out with the destination address of the received packet.

The `SOURCE` field is used only on the receive ring, and will be filled out with the source address of the received packet when the receive has completed.

The `LENGTH` and `POINTER` fields are scatter or gather pointers to buffers containing the command or reply data.

All pointer fields in descriptor rings contain linear (physical) addresses.

The scatter/gather pointers will be used in the order shown (`POINTER1`, then `POINTER2`, etc). The `LENGTH` fields for any unused pointers should be set to zero.

On the reply ring, descriptors ready for fulfillment must be given to the device with at least one valid scatter pointer.

6.2. Initial descriptor state

The host device driver must initialise the transmit and receive rings by:

1. Setting the `OWNER` field on each descriptor to `HOST`, and
2. Filling all other bytes with zero

For the receive and transmit rings, initialisation of the rings must be done prior to executing a `START` command, but may occur either before or after writing the `BASE` and `SHIFT` registers.

For the command ring (see Command ring), all descriptors must be initialised *before* `CMDBASE` is written by a host driver.

6.3. Transmitting a packet

Suppose the host driver wants to transmit a packet, to station address `0x12345678`. The datagram is located at linear address `0xabcd1200` and is of length `0x10`. The next free descriptor in the command ring has index `5`

To enqueue the packet for transmission, the driver must take the following steps:

1. Locate the next free descriptor in the transmit ring
2. Check that the descriptor's owner field is set to `HOST`
3. Write the fields of the descriptor other than `OWNER`: `DESTINATION = 0x12345678`, `LENGTH1 = 0x10`, `POINTER1 = 0xabcd1200`, and all the other `LENGTH` fields to zero
4. Perform a store memory barrier to prevent the previous stores crossing the next one
5. Write the `OWNER` field to `DEVICE`
6. Perform a store memory barrier again
7. Write the index of the descriptor (`0x5`) to the `DBELL` register in the BAR

Once the packet has been transmitted, the device will set the `TXCOMP` flag in `EVFLAGS` and emit an MSI-X interrupt on vector 0 (an event interrupt). Once the this flag has been read, and the driver has verified that the descriptor has changed back to `HOST` ownership, the packet data buffers may be released for other use (the device will not read from them again).

6.4. Receiving packets

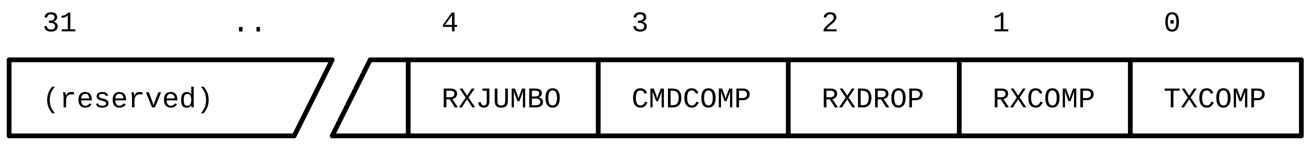
Suppose the host driver is ready to receive packets. It has allocated two 4k pages for receiving up to 8k of data, located at `0xabcd1000` and `0xabcd5000` (they are linearly discontinuous). The next free descriptor in the receive ring has index `0x31`

To use this buffer to receive a datagram, the driver must take the following steps:

1. Locate the next free descriptor in the receive ring
2. Check that the descriptor's owner field is set to `HOST`
3. Write the fields of the descriptor other than `OWNER`: `LENGTH1 = 0x1000`, `POINTER1 = 0xabcd1000`, `LENGTH2 = 0x1000`, `POINTER2 = 0xabcd5000`, all other `LENGTH` fields to zero
4. Perform a store memory barrier to prevent the previous stores crossing the next one
5. Write the `OWNER` field to `DEVICE`
6. Perform a store memory barrier again

6.5. Handling events

The following types of events are supported in this version of the specification, as shown by the structure of the EVFLAGS register:



- TXCOMP — completed descriptors are available on the transmit ring
- RXCOMP — completed descriptors are available on the receive ring
- CMDCOMP — completed descriptors are available on the command ring
- RXDROP — received packets have been dropped due to lack of available buffers on the receive ring
- RXJUMBO — received packets have been dropped due to lack of sufficient space for the packet data in the descriptor at the head of the receive ring (i.e. only X bytes worth of buffers were in the descriptor, but the packet received was >= X+1 bytes in size)

Suppose that the driver has not handled a completion interrupt yet, and is receiving one for the first time.

The driver must take the following steps:

1. Wait for an MSI on vector 0 from the device
2. Check the EVFLAGS register to see which rings have completed descriptors available.
3. Start at the descriptor after the last descriptor the device processed
4. Read its OWNER field to verify that it has changed to HOST
5. Perform a load memory barrier to prevent the load of the OWNER field crossing any subsequent loads
6. Read the other fields of the descriptor as needed to determine what action to take or read incoming packet data
7. Check the subsequent descriptor’s OWNER field and repeat until the next is marked DEVICE

To prevent ring underflow and dropped packets (RXDROP) it is also recommended to periodically issue a store barrier while in the loop at step 7.

7. Command ring

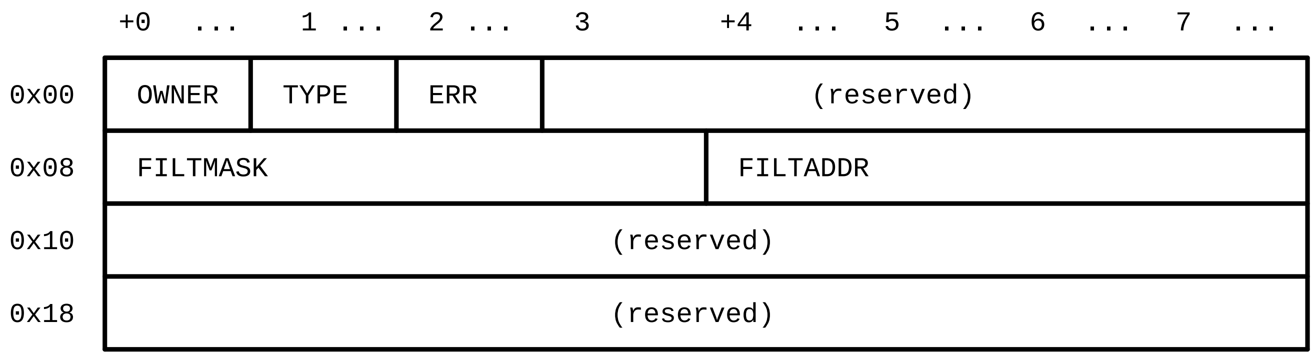
Much like the receive and transmit descriptor rings, the command descriptor ring consists of a region of contiguous linear memory allocated by the device driver. Its address is written into the CMDBASE register in BAR 0, and its size (as a bit shift value) in CMDSHIFT.

Descriptors in the command ring are 32 bytes in length, but otherwise behave similarly to RX/TX ring descriptors.

#	Mnemonic	Used for
1	`START`	Starts normal packet operation
2	`STOP`	Stops normal packet operation and any TX/RX ring polling
3	`ADDFILT`	Add an address filter to the receive ring
4	`RMFILT`	Remove an address filter from the receive ring

7.1. Command descriptor

The descriptor structure for the completion ring is shown below:



The `OWNER` field should be set either to the value `DEVICE` (`0x55`) or `HOST` (`0xAA`).

The `TYPE` field should be written to contain one of the command numbers shown in the table in the previous section.

The `ERR` field will be written by the device after processing each command with a result code. The result codes `0x00` (mnemonic `OK`) and `0xFF` (mnemonic `NOTSUP`) are standard across all commands; all other result codes are considered a type of error and have a per-command meaning listed below.

The `FILTMASK` and `FILTADDR` fields are to be used only for the `ADDFILT` and `RMFILT` commands, and contain the filter mask and filter address value to add or remove from the receive ring (see <<`ADDFILT` and `RMFILT` commands>>).

7.2. `START` and `STOP` commands

These commands start and stop packet processing on the device. The `START` command can only be validly issued when:

- The `RXBASE`/`RXSHIFT` and `TXBASE`/`TXSHIFT` registers have been written with valid values
- All descriptors on the RX and TX descriptor rings have been initialised to their required initial values (see Initial descriptor state).
- The `EVFLAGS` register has been read since the last `STOP` command (to reset it to zero)
- The `FLAGS` register is clear

Violations of these conditions will generally result in a device halt and setting the `FLAGS` register (see Error flags), but the following `START` specific error codes are also used:

#	Meaning
1	Already running; the `START` command was already previously issued

The `STOP` command is the inverse of `START`, i.e. it stops all packet processing on the device. While packet processing is stopped, the device is also guaranteed not to use the contents of the RX and TX descriptor rings in any way (i.e. all polling activity stops). This is useful to allow for freeing up any buffers still on those rings when the device is not in use.

The `STOP` command can only be validly issued when:

- The `START` command has been issued and succeeded previously

After issuing the `STOP` command, and before the command has been completed (i.e. the `OWNER` and `ERR` fields are written by the device, and an event generated), packet reception and transmission may still take place. Activity is guaranteed to cease only once the command has succeeded and its descriptor has been updated. This means that other events may be generated while the `STOP` command is executing and result in interrupts or buffers being used.

The following `STOP`-specific error codes are used:

#	Meaning
1	Already stopped; the `START` command was never issued or `STOP` was already previously issued

7.3. `ADDFILT` and `RMFILT` commands

These commands add and remove address filters from the receive ring.

When the device receives packets from the Ductnet network, it matches their destination address fields against a list of "filters". If any of the filters match, then the packet is placed onto the receive ring. Otherwise, the packet is discarded.

After a device reset, no filters are present on the receive ring. This means that no received packets will be placed on the receive ring until at least one `ADDFILT` command has been executed.

Each filter contains two fields: a bitmask (`FILTMASK`) and an address value (`FILTADDR`). The mask is combined with the packet destination address in a bitwise AND operation before being checked against the address value.

The condition which "matches" a filter is thus: `(DESTINATION & FILTMASK) = FILTADDR`.

When removing a filter using `RMFILT`, both fields must match exactly the values given previously in an `ADDFILT`. One `RMFILT` command will undo the effects of exactly one matching `ADDFILT`.

It is legal to add overlapping filters to the filter chain, and no validation will be carried out by the device to prevent the addition of overlapping or duplicate filters.

Filters may be added or removed at any time, irrespective of whether the packet processing engine is running (i.e. these commands may be executed either before or after any `START` or `STOP` commands), but will only be applied to incoming packets while the packet processing engine is enabled.

As of version 2.0, the device supports up to 16 hardware receive filters.

The following `ADDFILT`-specific error codes are used:

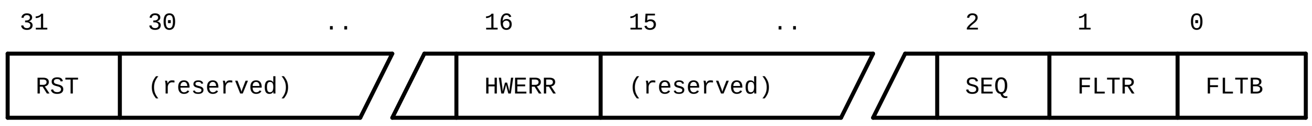
#	Meaning
1	No more filter space available

The following `RMFILT`-specific error codes are used:

#	Meaning
1	No exactly matching filter found to remove

8. Error flags

The `FLAGS` field in the BAR has the following bit structure:



- `FLTB` — the device encountered a page fault, bus fault or access violation while chasing a pointer provided by the driver in the BAR (e.g. `RXBASE`)
- `FLTR` — the device encountered a page fault, bus fault or access violation while chasing a pointer provided by the driver in a descriptor ring
- `SEQ` — the device detected an operation out of sequence (e.g. a write to the `DBELL` register before the ring base and shift registers have been set to valid values)
- `HWERR` — a miscellaneous hardware error occurred
- `RST` — set by the driver on write as part of the reset procedure (see Resetting the device). Always reads as zero.

Once any bit in the `FLAGS` register is set, the device will stop, and a reset will be required to resume service (see Resetting the device).

Encountering any condition which sets a bit in `FLAGS` will also trigger MSI-X interrupt vector 1, if enabled. This is a separate interrupt vector to that used for normal completion traffic.

As well as checking this register in the handler for MSI-X vector 1, a defensively written host driver could also perform a periodic check of the register, or check it at any time when it detects a "hang" where operations are not completing as normal.

Writes to the `FLAGS` register are allowed only as part of the reset procedure, and must be full 32-bit writes. Only the `RST` bit can be written: all other bits in a write will be ignored.

9. Resetting the device

A full device reset can be carried out in case of error. The steps required are:

1. Write the value `0x80000000` (`RST = 1`) to the `FLAGS` register in the BAR
2. Busy-poll the `FLAGS` register until its entire 32-bit value returns to `0`

To account for a hardware or bus fault, the driver may wish to limit the amount of time it busy-polls the register (or fall back to a periodic timer after some count of iterations).

All outstanding operations and DMA by the device will be abandoned, but some in-flight operations may still complete between the write to `FLAGS` and the point in time where it reads as `0` (and these may result in an MSI on vector 0 being issued). Once the `FLAGS` register has read `0`, all outstanding operations have either completed or been abandoned.