NAME

ibv_query_device_ex − query an RDMA device's attributes

SYNOPSIS

#include <infiniband/verbs.h>

int ibv_query_device_ex(struct ibv_context **context***,**
struct ibv_device_attr_ex **attr***);**

DESCRIPTION

ibv_query_device_ex() returns the attributes of the device with context *context*. The argument *attr* is a pointer to an ibv_device_attr_ex struct, as defined in <infiniband/verbs.h>.

```
struct ibv_device_attr_ex {
                        struct ibv_device_attr orig_attr;
                        uint32_t              comp_mask;               /* Compatibility mask that defines which of the following va
                        struct ibv_odp_caps   odp_caps;                /* On-Demand Paging capabilities */
                        uint64_t              completion_timestamp_mask;  /* Completion timestamp mask (0 = unsupported) */
                        uint64_t              hca_core_clock;          /* The frequency (in kHZ) of the HCA (0 = unsupported) */
                        uint64_t              device_cap_flags_ex;      /* Extended device capability flags */
                        struct ibv_tso_caps   tso_caps;                /* TCP segmentation offload capabilities */
                        struct ibv_rss_caps   rss_caps;                /* RSS capabilities */
                        uint32_t              max_wq_type_rq;          /* Max Work Queue from type RQ */
                        struct ibv_packet_pacing_caps packet_pacing_caps; /* Packet pacing capabilities */
                        uint32_t              raw_packet_caps;         /* Raw packet capabilities, use enum ibv_raw_packet_caps
                        struct ibv_tm_caps    tm_caps;                 /* Tag matching capabilities */
                        struct ibv_cq_moderation_caps  cq_mod_caps;      /* CQ moderation max capabilities */
                        uint64_t                  max_dm_size;                  /* Max Device Memory size (in bytes) availab
                        struct ibv_pci_atomic_caps atomic_caps;          /* PCI atomic operations capabilities, use enum ibv_
                        uint32_t              xrc_odp_caps;            /* Mask with enum ibv_odp_transport_cap_bits to know wh
};

struct ibv_odp_caps {
    uint64_t general_odp_caps;    /* Mask with enum ibv_odp_general_cap_bits */
    struct {
        uint32_t rc_odp_caps; /* Mask with enum ibv_odp_tranport_cap_bits to know which operations are supported.
        uint32_t uc_odp_caps; /* Mask with enum ibv_odp_tranport_cap_bits to know which operations are supported.
        uint32_t ud_odp_caps; /* Mask with enum ibv_odp_tranport_cap_bits to know which operations are supported.
    } per_transport_caps;
};

enum ibv_odp_general_cap_bits {
    IBV_ODP_SUPPORT = 1 << 0, /* On demand paging is supported */
    IBV_ODP_SUPPORT_IMPLICIT = 1 << 1, /* Implicit on demand paging is supported */
};

enum ibv_odp_transport_cap_bits {
    IBV_ODP_SUPPORT_SEND     = 1 << 0, /* Send operations support on-demand paging */
    IBV_ODP_SUPPORT_RECV     = 1 << 1, /* Receive operations support on-demand paging */
    IBV_ODP_SUPPORT_WRITE    = 1 << 2, /* RDMA-Write operations support on-demand paging */
    IBV_ODP_SUPPORT_READ     = 1 << 3, /* RDMA-Read operations support on-demand paging */
    IBV_ODP_SUPPORT_ATOMIC   = 1 << 4, /* RDMA-Atomic operations support on-demand paging */
    IBV_ODP_SUPPORT_SRQ_RECV = 1 << 5, /* SRQ receive operations support on-demand paging */
};

struct ibv_tso_caps {
```

```
        uint32_t max_tso;       /* Maximum payload size in bytes supported for segmentation by TSO engine.*/
        uint32_t supported_qpts; /* Bitmap showing which QP types are supported by TSO operation. */
};

struct ibv_rss_caps {
        uint32_t supported_qpts;                /* Bitmap showing which QP types are supported RSS */
        uint32_t max_rwq_indirection_tables;    /* Max receive work queue indirection tables */
        uint32_t max_rwq_indirection_table_size;   /* Max receive work queue indirection table size */
        uint64_t rx_hash_fields_mask;           /* Mask with enum ibv_rx_hash_fields to know which incoming packet's fiel
        uint8_t  rx_hash_function;              /* Mask with enum ibv_rx_hash_function_flags to know which hash functions a
};

struct ibv_packet_pacing_caps {
      uint32_t qp_rate_limit_min; /* Minimum rate limit in kbps */
      uint32_t qp_rate_limit_max; /* Maximum rate limit in kbps */
      uint32_t supported_qpts;    /* Bitmap showing which QP types are supported. */
};

enum ibv_raw_packet_caps {
                        IBV_RAW_PACKET_CAP_CVLAN_STRIPPING   = 1 << 0, /* CVLAN stripping is supported */
                        IBV_RAW_PACKET_CAP_SCATTER_FCS               = 1 << 1, /* FCS scattering is supporte
                        IBV_RAW_PACKET_CAP_IP_CSUM           = 1 << 2, /* IP CSUM offload is supported */
};

enum ibv_tm_cap_flags {
                        IBV_TM_CAP_RC   = 1 << 0,       /* Support tag matching on RC transport */
};

struct ibv_tm_caps {
                        uint32_t        max_rndv_hdr_size;   /* Max size of rendezvous request header */
                        uint32_t        max_num_tags;       /* Max number of tagged buffers in a TM-SRQ matching list */
                        uint32_t        flags;              /* From enum ibv_tm_cap_flags */
                        uint32_t        max_ops;            /* Max number of outstanding list operations */
                        uint32_t        max_sge;            /* Max number of SGEs in a tagged buffer */
};

struct ibv_cq_moderation_caps {
            uint16_t max_cq_count;
            uint16_t max_cq_period;
};

enum ibv_pci_atomic_op_size {
                        IBV_PCI_ATOMIC_OPERATION_4_BYTE_SIZE_SUP = 1 << 0,
                        IBV_PCI_ATOMIC_OPERATION_8_BYTE_SIZE_SUP = 1 << 1,
                        IBV_PCI_ATOMIC_OPERATION_16_BYTE_SIZE_SUP = 1 << 2,
};

struct ibv_pci_atomic_caps {
                        uint16_t fetch_add;         /* Supported sizes for an atomic fetch and add operation, use enum ibv_pc
                        uint16_t swap;              /* Supported sizes for an atomic unconditional swap operation, use enum i
                        uint16_t compare_swap;      /* Supported sizes for an atomic compare and swap operation, use enum ib
};
```

Extended device capability flags (device_cap_flags_ex):

IBV_DEVICE_PCI_WRITE_END_PADDING

Indicates the device has support for padding PCI writes to a full cache line.

Padding packets to full cache lines reduces the amount of traffic required at the memory controller at the expense of creating more traffic on the PCI-E port.

Workloads that have a high CPU memory load and low PCI-E utilization will benefit from this feature, while workloads that have a high PCI-E utilization and small packets will be harmed.

For instance, with a 128 byte cache line size, the transfer of any packets less than 128 bytes will require a full 128 transfer on PCI, potentially doubling the required PCI-E bandwidth.

This feature can be enabled on a QP or WQ basis via the IBV_QP_CREATE_PCI_WRITE_END_PADDING or IBV_WQ_FLAGS_PCI_WRITE_END_PADDING flags.

RETURN VALUE

ibv_query_device_ex() returns 0 on success, or the value of errno on failure (which indicates the failure reason).

NOTES

The maximum values returned by this function are the upper limits of supported resources by the device. However, it may not be possible to use these maximum values, since the actual number of any resource that can be created may be limited by the machine configuration, the amount of host memory, user permissions, and the amount of resources already in use by other users/processes.

SEE ALSO

ibv_query_device(3), **ibv_open_device**(3), **ibv_query_port**(3), **ibv_query_pkey**(3), **ibv_query_gid**(3)

AUTHORS

Majd Dibbiny <majd@mellanox.com>