NAME

mlx5dv_query_device − Query device capabilities specific to mlx5

SYNOPSIS

#include <infiniband/mlx5dv.h>

int mlx5dv_query_device(struct ibv_context *ctx_in,
struct mlx5dv_context *attrs_out);

DESCRIPTION

mlx5dv_query_device() Query HW device-specific information which is important for data-path, but isn't provided by **ibv_query_device**(3).

This function returns version, flags and compatibility mask. The version represents the format of the internal hardware structures that mlx5dv.h represents. Additions of new fields to the existed structures are handled by comp_mask field.

```
struct mlx5dv_sw_parsing_caps {
        uint32_t sw_parsing_offloads; /* Use enum mlx5dv_sw_parsing_offloads */
        uint32_t supported_qpts;
};

struct mlx5dv_striding_rq_caps {
        uint32_t min_single_stride_log_num_of_bytes; /* min log size of each stride */
        uint32_t max_single_stride_log_num_of_bytes; /* max log size of each stride */
        uint32_t min_single_wqe_log_num_of_strides; /* min log number of strides per WQE */
        uint32_t max_single_wqe_log_num_of_strides; /* max log number of strides per WQE */
        uint32_t supported_qpts;
};

struct mlx5dv_context {
        uint8_t         version;
        uint64_t        flags;
        uint64_t        comp_mask; /* Use enum mlx5dv_context_comp_mask */
        struct mlx5dv_cqe_comp_caps     cqe_comp_caps;
        struct mlx5dv_sw_parsing_caps sw_parsing_caps;
        uint32_t tunnel_offloads_caps;
        uint32_t        max_dynamic_bfregs /* max blue-flame registers that can be dynamiclly allocated */
};

enum mlx5dv_context_flags {
        /*
         * This flag indicates if CQE version 0 or 1 is needed.
         */
         MLX5DV_CONTEXT_FLAGS_CQE_V1 = (1 << 0),
         MLX5DV_CONTEXT_FLAGS_OBSOLETE   =  (1 << 1), /* Obsoleted, don't use */
         MLX5DV_CONTEXT_FLAGS_MPW_ALLOWED  = (1 << 2), /* Multi packet WQE is allowed */
         MLX5DV_CONTEXT_FLAGS_ENHANCED_MPW = (1 << 3), /* Enhanced multi packet WQE is
         MLX5DV_CONTEXT_FLAGS_CQE_128B_COMP = (1 << 4), /* Support CQE 128B compression
         MLX5DV_CONTEXT_FLAGS_CQE_128B_PAD = (1 << 5), /* Support CQE 128B padding */
         MLX5DV_CONTEXT_FLAGS_PACKET_BASED_CREDIT_MODE = (1 << 6), /* Support packet
};

enum mlx5dv_context_comp_mask {
        MLX5DV_CONTEXT_MASK_CQE_COMPRESION      = 1 << 0,
        MLX5DV_CONTEXT_MASK_SWP                 = 1 << 1,
        MLX5DV_CONTEXT_MASK_STRIDING_RQ         = 1 << 2,
        MLX5DV_CONTEXT_MASK_TUNNEL_OFFLOADS     = 1 << 3,
```

```
                        MLX5DV_CONTEXT_MASK_DYN_BFREGS         = 1 << 4,
                        MLX5DV_CONTEXT_MASK_CLOCK_INFO_UPDATE  = 1 << 5,
};

enum enum mlx5dv_sw_parsing_offloads {
                        MLX5DV_SW_PARSING        = 1 << 0,
                        MLX5DV_SW_PARSING_CSUM   = 1 << 1,
                        MLX5DV_SW_PARSING_LSO    = 1 << 2,
};

enum mlx5dv_tunnel_offloads {
                        MLX5DV_RAW_PACKET_CAP_TUNNELED_OFFLOAD_VXLAN  = 1 << 0,
                        MLX5DV_RAW_PACKET_CAP_TUNNELED_OFFLOAD_GRE    = 1 << 1,
                        MLX5DV_RAW_PACKET_CAP_TUNNELED_OFFLOAD_GENEVE = 1 << 2,
};

enum mlx5dv_flow_action_cap_flags {
                        MLX5DV_FLOW_ACTION_FLAGS_ESP_AES_GCM              = 1 << 0, /* Flow action ESP (with
                        MLX5DV_FLOW_ACTION_FLAGS_ESP_AES_GCM_REQ_METADATA  = 1 << 1, /* Flow actio
                        MLX5DV_FLOW_ACTION_FLAGS_ESP_AES_GCM_SPI_STEERING  = 1 << 2, /* ESP (with A
                        MLX5DV_FLOW_ACTION_FLAGS_ESP_AES_GCM_FULL_OFFLOAD  = 1 << 3, /* Flow actio
                        MLX5DV_FLOW_ACTION_FLAGS_ESP_AES_GCM_TX_IV_IS_ESN  = 1 << 4, /* Flow action
};
```

RETURN VALUE

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

NOTES

* Compatibility mask (comp_mask) is in/out field.

SEE ALSO

mlx5dv(7), **ibv_query_device**(3)

AUTHORS

Leon Romanovsky <leonro@mellanox.com>