FlexIO SDK API

Reference Manual

Table of Contents

Chapter 1. Change Log

This chapter list changes in API that were introduced to the library.

1.3.0

- Field Groups, GPU Groups, and field watches created with a handle returned from dcgmConnect() are now cleaned up upon disconnect. dcgmConnect_v2() can be used to get the old behavior of objects persisting after disconnect.
- dcgmConnect_v2() was added as a method for specifying additional connection options when connecting to the host engine.
- dcgmUnwatchFields() was added as a method of unwatching fields that were previously watched with dcgmWatchFields()
- dcgmActionValidate_v2() was added to be able to pass more parameters to the DCGM GPU Diagnostic.
- dcgmDiagResponse_t was increased from v2 to v3. See dcgmDiagResponse_v3 for details

1.2.3

- No API changes in this version.

1.1.1

- dcgmGetAllSupportedDevices() was added as a method to get DCGM-supported GPU Ids. dcgmGetAllDevices() can still be used to get all GPU Ids in the system.

1.0.0

- Initial Release.

Chapter 2. Modules

Here is a list of all modules:

- DPATOOLSSDK
- Flex IO SDK host
- Flex IO SDK dev
  - Flex IO SDK dev error handling
  - Flex IO SDK dev queue access
  - Flex IO SDK dev queue types
- <stdio.h>: Standard IO facilities

2.1. DPATOOLSSDK

DPA Performance tools SDK host API Mostly used for DPA read processes, threads amd statistics.

struct dpa_stats_capabilities

struct dpa_stats_perf_cumul_list

struct dpa_stats_perf_event_list

struct dpa_stats_perf_event_sample

struct dpa_stats_process_info

struct dpa_stats_process_list

struct dpa_stats_thread_cumul_info

struct dpa_stats_thread_info

struct dpa_stats_thread_list

enum dpa_stats_counter_state

List of the performance counters state.

Values

DPA_STATS_COUNTER_STATE_ACTIVE = 1
DPA_STATS_COUNTER_STATE_INACTIVE = 2
DPA_STATS_COUNTER_STATE_RESET = 3

enum dpa_stats_event_sample_type

List of the performance event samples type.

Values

DPA_STATS_EVENT_SAMPLE_TYPE_EMPTY_SAMPLE = 0x0
DPA_STATS_EVENT_SAMPLE_TYPE_SCHEDULE_IN = 0x1
DPA_STATS_EVENT_SAMPLE_TYPE_SCHEDULE_OUT = 0x2
DPA_STATS_EVENT_SAMPLE_TYPE_BUFFER_FULL = 0xff

enum dpa_stats_sample_type

List of the performance counters samples type.

Values

DPA_STATS_SAMPLE_TYPE_CUMULATIVE_EVENT = 0
DPA_STATS_SAMPLE_TYPE_EVENT_TRACER = 1

enum dpa_stats_status

DPA stats API return code.

Values

DPA_STATS_STATUS_OK = 0
DPA_STATS_STATUS_ERROR_IN_MFT = 1
DPA_STATS_STATUS_NO_SUCH_PROCESS = 2
DPA_STATS_STATUS_NO_SUCH_THREADS = 3
DPA_STATS_STATUS_NO_SUCH_EVENTS = 4
DPA_STATS_STATUS_ERROR_IN_ALLOC = 10
DPA_STATS_STATUS_INVALID_PARAMS = 20
DPA_STATS_STATUS_NULL_POINTER = 21
DPA_STATS_STATUS_VALUE_OUT_OF_ENUM = 22

DPA_STATS_EXPERIMENTAL void dpa_stats_close (dpa_stats_handler_t *dpa_handler)

Close previously opened DPA device for reading statistics.

Parameters

dpa_handler
- a pointer of opened device.

Description

This function closes DPA device for reading statistics.

dpa_stats_compare_4_perf_cumul (const void *elem1, const void *elem2)

Compare two cumulative info elements.

Parameters

elem1
- pointer to first element.

elem2
- pointer to second element.

Returns

result of the compare.

Description

This function compares two cumulative info elements by their process and thread id.

dpa_stats_compare_4_processes (const void *elem1, const void *elem2)

Compare two process elements.

Parameters

elem1
- pointer to first element.

elem2
- pointer to second element.

Returns

results of the compare.

Description

This function compares two process elements by their process id.

dpa_stats_compare_4_threads (const void *elem1, const void *elem2)

Compare two thread elements.

Parameters

elem1
- pointer to first element.

elem2
- pointer to second element.

Returns

results of the compare.

Description

This function compares two thread elements by their process and thread id.

DPA_STATS_EXPERIMENTAL void dpa_stats_free_cumul_info_list (dpa_stats_perf_cumul_list *cumul_info_list)

Free previously allocated list of the cumulative info counters.

Parameters

cumul_info_list
- a list of the cumulative info counters.

Description

This function frees the previously allocated list of cumulative info counters.

DPA_STATS_EXPERIMENTAL void dpa_stats_free_perf_event_list (dpa_stats_perf_event_list *perf_event_list)

Free previously allocated event counters list.

Parameters

perf_event_list
- a list of the events counters.

Description

This function frees the previously allocated event counters list.

DPA_STATS_EXPERIMENTAL void dpa_stats_free_process_list (dpa_stats_process_list *process_list)

Free previously allocated process list.

Parameters

process_list
- a list of the processes.

Description

This function frees the previously allocated process list.

DPA_STATS_EXPERIMENTAL void dpa_stats_free_thread_list (dpa_stats_thread_list *thread_list)

Free previously allocated thread list.

Parameters

thread_list
- a list of the threads.

Description

This function frees the previously allocated thread list.

dpa_stats_status dpa_stats_get_counter_state (dpa_stats_handler_t *dpa_handler, uint32_t process_id, dpa_stats_counter_state *state, dpa_stats_sample_type *type)

Get counter state and type.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id.

state
- counter state.

type
- counter type.

Returns

dpa status value.

Description

This function gets event counters state and type for specific process.

dpa_stats_status dpa_stats_get_error_code (dpa_stats_handler_t *dpa_handler)

Report error code.

Parameters

dpa_handler
- a pointer of opened device.

Returns

last error code.

Description

This function returns last error code reported by the opened device. If no error - returns DPA_STATS_STATUS_OK.

DPA_STATS_EXPERIMENTAL char *dpa_stats_get_error_location (dpa_stats_handler_t *dpa_handler)

Report error location.

Parameters

dpa_handler
- a pointer of opened device.

Returns

last error location.

Description

This function return info of last error location reported by the opened device. If no error - returns empty string.

DPA_STATS_EXPERIMENTAL char *dpa_stats_get_error_message (dpa_stats_handler_t *dpa_handler)

Report error message.

Parameters

dpa_handler
- a pointer of opened device.

Returns

last error message.

Description

This function returns last error message reported by the opened device. If no error - returns empty string.

DPA_STATS_EXPERIMENTAL dpa_stats_handler_t *dpa_stats_open (const char *dpa_device)

Open DPA device for reading statistics.

Parameters

dpa_device
- name of the dpa_device.

Returns

return pointer of the opened device or NULL.

Description

This function open DPA device for reading statistics. On success - return pointer of opened device. On Failure - return NULL.

dpa_stats_status dpa_stats_read_caps (dpa_stats_handler_t *dpa_handler, dpa_stats_capabilities *caps)

Read DPA device capabilities.

Parameters

dpa_handler
- a pointer of opened device

caps
- a pointer to the structure that will be filled with device capabilities.

Returns

dpa status value.

Description

This function reads capabilities of DPA device and counters to the structure.

dpa_stats_status dpa_stats_read_cumul_info_list (dpa_stats_handler_t *dpa_handler, dpa_stats_process_list *process_list, dpa_stats_perf_cumul_list **cumul_info_list)

Read list of the cumulative info counters for a list of processes.

Parameters

dpa_handler
- a pointer of opened device.

process_list
- previously read list of processes.

cumul_info_list
- pointer to the list of the cumulative info counters struct pointer.

Returns

dpa status value.

Description

This function reads list of the cumulative info counters for a given list of processes.

dpa_stats_status dpa_stats_read_perf_event_list (dpa_stats_handler_t *dpa_handler, dpa_stats_process_list *process_list, dpa_stats_perf_event_list **perf_event_list)

Read list of the performance event tracer counters for a list of processes.

Parameters

dpa_handler
- a pointer of opened device.

process_list
- previously read list of the processes.

perf_event_list
- pointer to the list of the performance event tracer counters struct pointer.

Returns

dpa status value.

Description

This function reads list of the performance event tracer counters for a given list of processes.

dpa_stats_status dpa_stats_read_process_cumul_info_list (dpa_stats_handler_t *dpa_handler, uint32_t process_id, uint32_t thread_id, dpa_stats_perf_cumul_list **cumul_info_list)

Read list of the cumulative info counters.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id or DPA_STATS_ALL_PROCESSES.

thread_id
- DPA thread id or DPA_STATS_ALL_THREADS. If process_id is DPA_STATS_ALL_PROCESSES the thread_id must be DPA_STATS_ALL_THREADS.

cumul_info_list
- pointer to the list of the cumulative info counters struct pointer.

Returns

dpa status value.

Description

This function reads list of the cumulative info counters for specific process or all processes and specific thread or all threads.

dpa_stats_status dpa_stats_read_process_perf_event_list (dpa_stats_handler_t *dpa_handler, uint32_t process_id, uint32_t thread_id, dpa_stats_perf_event_list **perf_event_list)

Read list of the performance event tracer counters.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id or DPA_STATS_ALL_PROCESSES.

thread_id
- DPA thread id or DPA_STATS_ALL_THREADS. If process_id is DPA_STATS_ALL_PROCESSES the thread_id must be DPA_STATS_ALL_THREADS.

perf_event_list
- pointer to the list of the performance event tracer counters struct pointer.

Returns

dpa status value.

Description

This function reads list of the performance event tracer counters for specific process or all processes and specific thread or all threads.

dpa_stats_status dpa_stats_read_process_thread_list (dpa_stats_handler_t *dpa_handler, uint32_t process_id, uint32_t thread_id, dpa_stats_thread_list **thread_list)

Read threads info.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id or DPA_STATS_ALL_PROCESSES.

thread_id
- DPA thread id or DPA_STATS_ALL_THREADS. If process_id is DPA_STATS_ALL_PROCESSES the thread_id must be DPA_STATS_ALL_THREADS.

thread_list
- pointer to thread list struct pointer.

Returns

dpa status value.

Description

This function reads the threads info for specific process or all processes and specific thread or all threads.

dpa_stats_status dpa_stats_read_processes_list (dpa_stats_handler_t *dpa_handler, uint32_t process_id, dpa_stats_process_list **process_list)

Read the processes info.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id or DPA_STATS_ALL_PROCESSES.

process_list
- pointer to a processes list struct pointer.

Returns

dpa status value.

Description

This function reads the processes info for a specific process or all processes.

dpa_stats_status dpa_stats_read_thread_list (dpa_stats_handler_t *dpa_handler, dpa_stats_process_list *process_list, dpa_stats_thread_list **thread_list)

Read threads info for a list of processes.

Parameters

dpa_handler
- a pointer of opened device.

process_list
- previously read list of processes.

thread_list
- pointer to thread list struct pointer..

Returns

dpa status value.

Description

This function reads threads info for a given list of processes.

dpa_stats_status dpa_stats_set_counter_state (dpa_stats_handler_t *dpa_handler, uint32_t process_id, dpa_stats_counter_state state, dpa_stats_sample_type type)

Set counter state and type.

Parameters

dpa_handler
- a pointer of opened device.

process_id
- DPA process id or DPA_STATS_ALL_PROCESSES.

state
- counter state.

type
- counter type.

Returns

dpa status value.

Description

This function sets event counters for specific process or all processes with appropriate state and type.

dpa_stats_sort_perf_cumul_list (dpa_stats_perf_cumul_list *cumul_list)

Sort cumulative info list.

Parameters

cumul_list
- list of the cumulative info.

dpa_stats_sort_process_list (dpa_stats_process_list *process_list)

Sort process list.

Parameters

process_list
- list of processes.

dpa_stats_sort_thread_list (dpa_stats_thread_list *thread_list)

Sort thread list.

Parameters

thread_list
- list of the thread.

#define DPA_STATS_EXPERIMENTAL __attribute__((deprecated("Symbol is defined as experimental"), \ section(".text.experimental"))) \ DPA_STATS_STABLE

To set a Symbol (or specifically a function) as experimental.

2.2. Flex IO SDK host

Flex IO SDK host API for DPA programs. Mostly used for DPA resource management and invocation of DPA programs.

struct flexio_affinity

struct flexio_app_attr

struct flexio_cmdq_attr

struct flexio_cq_attr

struct flexio_event_handler_attr

struct flexio_heap_mem_info

struct flexio_mkey_attr

struct flexio_msg_stream_attr_t

struct flexio_outbox_attr

struct flexio_process_attr

struct flexio_qmem

struct flexio_qp_attr

struct flexio_qp_attr_opt_param_mask

struct flexio_wq_attr

struct flexio_wq_rq_attr

struct flexio_wq_sq_attr

enum flexio_affinity_type

Flex IO thread affinity types.

Values

FLEXIO_AFFINITY_NONE = 0
FLEXIO_AFFINITY_STRICT
FLEXIO_AFFINITY_GROUP

enum flexio_cmdq_state

Flex IO command queue states.

Values

FLEXIO_CMDQ_STATE_PENDING = 0
FLEXIO_CMDQ_STATE_RUNNING = 1

enum flexio_cq_period_mode

Flex IO CQ CQE compression period modes.

Values

FLEXIO_CQ_PERIOD_MODE_EVENT = 0x0
FLEXIO_CQ_PERIOD_MODE_CQE = 0x1

enum flexio_cqe_comp_type

Flex IO CQ CQE compression modes.

Values

FLEXIO_CQE_COMP_NONE = 0x0
FLEXIO_CQE_COMP_ENABLE = 0x2

enum flexio_log_lvl_t

Flex IO SDK host logging levels

Values

FLEXIO_LOG_LVL_ERR = 0
FLEXIO_LOG_LVL_WARN = 1
FLEXIO_LOG_LVL_INFO = 2
FLEXIO_LOG_LVL_DBG = 3

enum flexio_memtype

Flex IO memory types.

Values

FLEXIO_MEMTYPE_DPA = 0
FLEXIO_MEMTYPE_HOST = 1

enum flexio_msg_dev_sync_mode

Flex IO device messaging synchronization modes.

Values

FLEXIO_LOG_DEV_SYNC_MODE_SYNC = 0
FLEXIO_LOG_DEV_SYNC_MODE_ASYNC = 1
FLEXIO_LOG_DEV_SYNC_MODE_BATCH = 2
FLEXIO_LOG_DEV_SYNC_MODE_TRACER = 3

enum flexio_qp_op_types

Flex IO QP operation types.

Values

FLEXIO_QP_WR_RDMA_WRITE = 0x4
FLEXIO_QP_WR_RDMA_READ = 0x8
FLEXIO_QP_WR_ATOMIC_CMP_AND_SWAP = 0x10

enum flexio_qp_qpc_mtu

Flex IO QP possible MTU values.

Values

FLEXIO_QP_QPC_MTU_BYTES_256 = 0x1
FLEXIO_QP_QPC_MTU_BYTES_512 = 0x2
FLEXIO_QP_QPC_MTU_BYTES_1K = 0x3
FLEXIO_QP_QPC_MTU_BYTES_2K = 0x4
FLEXIO_QP_QPC_MTU_BYTES_4K = 0x5

enum flexio_qp_state

Flex IO QP states.

Values

FLEXIO_QP_STATE_RST = 0x0
FLEXIO_QP_STATE_INIT = 0x1
FLEXIO_QP_STATE_RTR = 0x2
FLEXIO_QP_STATE_RTS = 0x3
FLEXIO_QP_STATE_ERR = 0x6

enum flexio_qp_transport_type

Flex IO QP states.

Values

FLEXIO_QPC_ST_RC = 0x0
FLEXIO_QPC_ST_UC = 0x1
FLEXIO_QPC_ST_UD = 0x2
FLEXIO_QPC_ST_XRC = 0x3
FLEXIO_QPC_ST_IBL2 = 0x4
FLEXIO_QPC_ST_DCI = 0x5
FLEXIO_QPC_ST_QP0 = 0x7
FLEXIO_QPC_ST_QP1 = 0x8
FLEXIO_QPC_ST_RAW_DATAGRAM = 0x9
FLEXIO_QPC_ST_REG_UMR = 0xc
FLEXIO_QPC_ST_DC_CNAK = 0x10

enum flexio_status

Flex IO API function return codes.

Values

FLEXIO_STATUS_SUCCESS = 0
FLEXIO_STATUS_FAILED = 1
FLEXIO_STATUS_TIMEOUT = 2
FLEXIO_STATUS_FATAL_ERR = 3

enum flexio_tracer_transport

Flex IO device messaging tracer transport modes.

Values

FLEXIO_TRACER_TRANSPORT_QP = 0
FLEXIO_TRACER_TRANSPORT_WINDOW = 1

enum flexio_wq_end_pad_mode

Flex IO supported WQ types.

Values

FLEXIO_WQ_END_PAD_NONE = 0x0
FLEXIO_WQ_END_PAD_ALIGN = 0x1

enum flexio_wq_type

Flex IO supported WQ types.

Values

FLEXIO_WQ_TYPE_LINKED_LIST = 0x0
FLEXIO_WQ_TYPE_CYCLIC = 0x1

typedef void (flexio_func_arg_pack_fn_t)

Callback function to pack the arguments for a function.

This function is called internally from the FlexIO runtime upon user making a call (e.g., flexio_process_call). It packs the arguments for a user function into the argument buffer provided in `argbuf`. The argument list can be arbitrarily long and is represented by `ap`. The correct usage of this function requires the caller to initialize the list using `va_start`.

typedef void (flexio_func_t)

Flex IO application function prototype.

typedef uint32_t flexio_uar_device_id

Flex IO UAR extension ID prototype.

typedef uint64_t flexio_uintptr_t

Flex IO address type.

flexio_status flexio_app_create (flexio_app_attr *fattr, flexio_app **app)

Create a container for a FlexIO App.

Parameters

fattr

- A pointer to the application attributes struct.

app

- Created app.

Returns

flexio status value.

Description

This function creates a named app with a given ELF buffer. It is called from within the constructor generated by the compiler.

flexio_status flexio_app_destroy (flexio_app *app)

Destroy a flexio app.

Parameters

app

- App that was created before.

Returns

flexio status value.

Description

This function destroys the state associated with the app and all registered functions. This function will free the internal elf buffer. It is called from within the destructor generated by the compiler.

flexio_status flexio_app_get_elf (flexio_app *app, uint64_t *bin_buff, size_t bin_size)

Retrieve ELF binary associated with application.

Parameters

app
- App that created before.

bin_buff
- Pointer to buffer to copy ELF binary.

bin_size
- Size of buffer pointed by bin_buff. If parameter is smaller than ELF binary size function will fail.

Returns

flexio status value.

Description

This function registers the function name, stub address with the runtime. Compiler calls this from within the constructor.

FLEXIO_EXPERIMENTAL size_t flexio_app_get_elf_size (flexio_app *app)

Gets a Flex IO application size.

Parameters

app
- A pointer to a Flex IO application.

Returns

the application's size (bytes) or NULL on error.

flexio_status flexio_app_get_list (flexio_appapp_list, uint32_t *num_apps)

Get a list of FlexIO Apps that are available.

Parameters

app_list
- A list of apps that are available.

num_apps
- number of apps to obtain / obtained.

Returns

flexio status value.

Description

This function returns a list of Flex IO apps that are loaded.

const FLEXIO_EXPERIMENTAL char *flexio_app_get_name (flexio_app *app)

Gets a Flex IO application name.

Parameters

app
- A pointer to a Flex IO application.

Returns

the application's name or NULL on error.

flexio_status flexio_app_list_free (flexio_app **apps_list)

Free the list of flexio apps.

Parameters

apps_list
- list obtained previously.

Returns

flexio status value.

Description

This function frees the list of apps obtained from `flexio_app_get_list`.

flexio_status flexio_buf_dev_alloc (flexio_process *process, size_t buff_bsize, flexio_uintptr_t *dest_daddr_p)

Allocates a buffer on Flex IO heap memory.

Parameters

process

- A pointer to the Flex IO process context.

buff_bsize

- The size of the buffer to allocate.

dest_daddr_p

- A pointer to the Flex IO address, where the buffer was allocated.

Returns

flexio status value.

Description

This function allocates a buffer with the requested size on the Flex IO heap memory. On success - sets dest_daddr_p to the start address of the allocated buffer. On Failure - sets dest_daddr_p to 0x0.

flexio_status flexio_buf_dev_free (flexio_process *process, flexio_uintptr_t daddr)

Deallocates Flex IO heap memory buffer.

Parameters

process

- A pointer to the Flex IO process context.

daddr

- A pointer to an address of allocated memory on the Flex IO heap. Zero value is valid argument.

Returns

flexio status value.

Description

This function frees Flex IO heap memory buffer by address.

flexio_status flexio_buf_dev_memset (flexio_process *process, int value, size_t buff_bsize, flexio_uintptr_t dest_daddr)

Sets DPA heap memory buffer to a given value.

Parameters

process
- A pointer to the Flex IO process context.

value
- A value to set the DPA heap memory buffer to.

buff_bsize
- The size of the Flex IO heap memory buffer.

dest_daddr
- Flex IO heap memory buffer address to set.

Returns

flexio status value.

flexio_status flexio_cmdq_create (flexio_process *process, flexio_cmdq_attr *fattr, flexio_cmdq **cmdq)

Create asynchronous rpc command queue.

Parameters

process
- A pointer to the process context.

fattr
- A pointer to the command queue attributes struct.

cmdq
- A pointer to the created command queue context pointer.

Returns

flexio status value.

Description

This function creates the asynchronous rpc command queue infrastructure allowing background tasks execution.

flexio_status flexio_cmdq_destroy (flexio_cmdq *cmdq)

Destroy the command queue infrastructure.

Parameters

cmdq

- A pointer to the command queue context.

Returns

flexio status value.

Description

This function destroy the command queue infrastructure and release all its resources.

FLEXIO_EXPERIMENTAL int flexio_cmdq_is_empty (flexio_cmdq *cmdq)

Check if command queue is empty.

Parameters

cmdq

- A pointer to the command queue context.

Returns

boolean.

Description

This function checks if the command queue is empty and all jobs up to this point where performed.

flexio_status flexio_cmdq_state_running (flexio_cmdq *cmdq)

Move command queue to running state.

Parameters

cmdq

- A pointer to the command queue context.

Returns

flexio status value.

Description

This function moves the command queue to running state in the case the queue was create in pending state. Otherwise has no affect.

flexio_status flexio_cmdq_task_add (flexio_cmdq *cmdq, flexio_func_t *host_func, uint64_t arg)

Add a task to the asynchronous rpc command queue.

Parameters

cmdq

- A pointer to the command queue context.

host_func

- host stub function for DPA function to execute.

arg

- user argument to function.

Returns

flexio status value.

Description

This function adds a task to the asynchronous rpc command queue to be executed by DPA in background. allowing background jobs execution.

flexio_status flexio_copy_from_host (flexio_process *process, void *src_haddr, size_t buff_bsize, flexio_uintptr_t *dest_daddr_p)

Copy from host memory to Flex IO heap memory buffer.

Parameters

process

- A pointer to the Flex IO process context.

src_haddr

- An address of the buffer on the host memory.

buff_bsize
- The size of the buffer to copy.

dest_daddr_p
- A pointer to the Flex IO address, where the buffer was copied to.

Returns

flexio status value.

Description

This function copies data from a buffer on the host memory to the Flex IO memory. The function allocates memory on the device heap which dest_address points to. It is the caller responsibility to deallocate this memory when it is no longer used.

flexio_status flexio_coredump_create (flexio_process *process, const char *outfile)

Create a DPA core dump of the process.

Parameters

process
- A pointer to a flexio_process

outfile
- pathname to write ELF formatted core dump data too. If NULL - filename will be generated in form flexio_dev.NNN.core, where NNN is the process id. If outfile is not NULL - suffix .NNN.core will be added. If outfile starts from slash (/pathname) - it will be passed with suffix described above to fopen() otherwise outfile will be created in the current directory or (if failed) in /tmp directory

Returns

flexio status value.

Description

This function creates a core dump image of a process and all it's threads, and is intended to be used after a fatal error or abnormal termination to allow the user to debug DPA application code.

There must be sufficient free memory to allocate 2-3 times the maximum core file size for intermediate processing before the elf file is written.

Memory windows that may be referenced by DPA code are *not* dumped by this code and must be handled separately if the data is desired.

flexio_status flexio_cq_create (flexio_process *process, ibv_context *ibv_ctx, const flexio_cq_attr *fattr, flexio_cq **cq)

Creates a Flex IO CQ.

Parameters

process

- A pointer to the Flex IO process.

ibv_ctx

- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

fattr

- A pointer to the CQ attributes struct.

cq

- A pointer to the created CQ context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO CQ.

flexio_status flexio_cq_destroy (flexio_cq *cq)

Destroys a Flex IO CQ.

Parameters

cq

- A pointer to a CQ context.

Returns

flexio status value.

Description

This function destroys a Flex IO CQ.

FLEXIO_EXPERIMENTAL uint32_t flexio_cq_get_cq_num (flexio_cq *cq)

Gets the Flex IO CQ number.

Parameters

cq
- A pointer to a Flex IO CQ.

Returns

the CQ number or UINT32_MAX on error.

mlx5dv_devx_obj *flexio_cq_get_obj (flexio_cq *cq)

Get the Flex IO CQ object.

Parameters

cq
- A pointer to the CQ context.

Returns

the CQ devx object.

Description

This function returns the Flex IO CQ object.

flexio_status flexio_cq_modify_moderation (flexio_cq *cq, uint16_t max_count, uint16_t period, uint16_t mode)

Modifies a Flex IO CQ moderation configuration.

Parameters

cq
- A pointer to a CQ context.

max_count
- CQ moderation max count value.

period
- CQ moderation period value.

mode
- CQ moderation mode value.

Returns

flexio status value.

flexio_status flexio_cq_query_moderation (flexio_cq *cq, uint16_t *max_count, uint16_t *period, uint16_t *mode)

Queries a Flex IO CQ moderation configuration.

Parameters

cq
- A pointer to a CQ context.

max_count
- A pointer to the CQ moderation max count value.

period
- A pointer to the CQ moderation period value.

mode
- A pointer to the CQ moderation mode value.

Returns

flexio status value.

flexio_status flexio_crash_data (flexio_process *process, const char *outfile)

Provide crash info in textual form.

Parameters

process
- A pointer to a flexio_process

outfile
- pathname to write ELF formatted core dump data too. If NULL - filename will be generated in form flexio_dev.NNN.crash, where NNN is the process id. If outfile is not NULL - suffix .NNN.crash will be added. If outfile starts from slash (/pathname) - it will be passed with suffix described above to fopen() otherwise outfile will be created in the current directory or (if failed) in /tmp directory

Returns

flexio status value.

Description

This function displays useful crash info in textual form. Info will be printed on console and duplicated to outfile

flexio_status flexio_device_mkey_create (flexio_process *process, flexio_mkey_attr *fattr, flexio_mkey **mkey)

Creates an Mkey to the process device UMEM.

Parameters

process
- A pointer to the Flex IO process context.

fattr
- A pointer to a Flex IO MKey attribute struct.

mkey
- A pointer to a pointer to the created MKey struct.

Returns

flexio status value.

Description

This function creates an MKey over the provided PD for the provided process device UMEM. The mkey_id will point to the field in the containing flexio_mkey object.

flexio_status flexio_device_mkey_destroy (flexio_mkey *mkey)

destroys an MKey object containing the given ID

Parameters

mkey
- A pointer to the Flex IO MKey to destroy. NULL is a valid value.

Returns

flexio status value.

Description

This function destroys an Mkey object containing the given ID.

FLEXIO_EXPERIMENTAL int flexio_err_handler_fd (flexio_process *process)

Get file descriptor for error handler.

Parameters

process

- A pointer to the Flex IO process.

Returns

- file descriptor.

Description

User should get fd in order to monitor for nonrecoverable errors

User can poll all created processes, using select/poll/epoll functions family.

FLEXIO_EXPERIMENTAL int flexio_err_status_get (flexio_process *process)

Check if unrecoverable error occurred.

Parameters

process

- A pointer to the Flex IO process. NULL is a valid value.

Returns

- nonzero value if error happen. See explanation of function flexio_dev_error() for agreement regarding error codes ranges (FW errors, FlexIO errors, User errors)

Description

It is suggested to check error status if file from flexio_err_handler_fd() reports about existence data to read.

Check error status before finishing process as well.

flexio_status flexio_event_handler_create (flexio_process *process, flexio_event_handler_attr *fattr, flexio_event_handler **event_handler_ptr)

Creates a Flex IO event handler.

Parameters

process

- A pointer to the Flex IO process.

fattr

- A pointer to the event handler attributes struct.

event_handler_ptr

- A pointer to the created event handler context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO event handler for an existing Flex IO process.

flexio_status flexio_event_handler_destroy (flexio_event_handler *event_handler)

Destroys a Flex IO event handler.

Parameters

event_handler

- A pointer to an event handler context.

Returns

flexio status value.

Description

This function destroys a Flex IO event handler.

FLEXIO_EXPERIMENTAL uint32_t flexio_event_handler_get_activation_id (flexio_event_handler *event_handler)

Query the Flex IO event handler activation id.

Parameters

event_handler

- A pointer to an event handler context.

Returns

the activation id or UINT32_MAX in case of error.

Description

This function returns the needed activation id in order to activate this event handler by another thread of the same process.

FLEXIO_EXPERIMENTAL uint32_t flexio_event_handler_get_id (flexio_event_handler *event_handler)

Gets the ID from a Flex IO event handler's thread metadata.

Parameters

event_handler

- A pointer to a Flex IO event handler.

Returns

the event handler's thread ID or UINT32_MAX on error.

FLEXIO_EXPERIMENTAL uint32_t flexio_event_handler_get_obj_id (flexio_event_handler *event_handler)

Gets the object ID of a Flex IO event handler.

Parameters

event_handler

- A pointer to a Flex IO event handler.

Returns

the event handler's thread object ID or UINT32_MAX on error.

FLEXIO_EXPERIMENTAL flexio_thread *flexio_event_handler_get_thread (flexio_event_handler *event_handler)

Gets a Flex IO thread object from a Flex IO event handler.

Parameters

event_handler
- A pointer to a Flex IO event handler.

Returns

the event handler's thread or NULL on error.

FLEXIO_EXPERIMENTAL mlx5dv_devx_obj *flexio_event_handler_get_thread_obj (flexio_event_handler *event_handler)

Gets the thread object of a Flex IO event handler.

Parameters

event_handler
- A pointer to a Flex IO event handler.

Returns

the event handler's thread object or null on error.

flexio_status flexio_event_handler_run (flexio_event_handler *event_handler, uint64_t user_arg)

Run a Flex IO event handler.

Parameters

event_handler
- A pointer to an event handler context.

user_arg
- A 64 bit argument for the event handler's thread.

Returns

flexio status value.

Description

This function makes a Flex IO event handler start running.

flexio_status flexio_func_get_register_info (flexio_app *app, flexio_func_t *host_stub_func_addr, uint32_t *pup, char *dev_func_name, char *dev_unpack_func_name, size_t func_name_size, size_t *argbuf_size, flexio_func_arg_pack_fn_t **host_pack_func, flexio_uintptr_t *dev_func_addr, flexio_uintptr_t *dev_unpack_func_addr)

Obtain info for previously registered function.

Parameters

app
- FlexIO app.

host_stub_func_addr
- Known host stub func addr.

pup
- Whether function has been registered with pack/unpack support (0: No, 1:Yes).

dev_func_name
- Name of device function.

dev_unpack_func_name
- Name of unpack routine on device, NA if pup == 0.

func_name_size
- Size of function name len allocated.

argbuf_size
- Size of argument buffer, NA if pup == 0.

host_pack_func
- Function pointer to host packing routine, NA if pup == 0.

dev_func_addr
- address of device function.

dev_unpack_func_addr
- address of device unpack function.

Returns

flexio status value.

Description

This function is used to obtain info about a previously registered function. It is used to compose higher-level libraries on top of DPACC / FlexIO interface. It is not intended to be used directly by the user.

The caller must ensure that the string pointers have been allocated and are at least `FLEXIO_MAX_NAME_LEN + 1` long to ensure that the call doesn't fail to copy full function name.

flexio_status flexio_func_pup_register (flexio_app *app, const char *dev_func_name, const char *dev_unpack_func_name, flexio_func_t *host_stub_func_addr, size_t argbuf_size, flexio_func_arg_pack_fn_t *host_pack_func)

Register a function name at application start.

Parameters

app
- App that created before.

dev_func_name
- The device function name (entry point). Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

dev_unpack_func_name
- The device wrapper function that unpacks the argument buffer. Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

host_stub_func_addr
- The host stub function that is used by the application to reference the device function.

argbuf_size
- Size of the argument buffer required by this function.

host_pack_func
- Host callback function that packs the arguments.

Returns

flexio status value.

Description

This function registers the function name, stub address with the runtime. It is called from within the constructor generated by the compiler.

flexio_status flexio_func_register (flexio_app *app, const char *dev_func_name, flexio_func_t **out_func)

Register a function to be used later.

Parameters

app

- previously created flexio app.

dev_func_name

- name of flexio function on device that will be called. Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

out_func

- opaque handle to use with flexio_process_call(), flexio_event_handler_create(), ...

Returns

flexio status value.

Description

This function is intended to be called directly by user in the situation where they don't desire pack/unpack support that is typically done by the compiler interface.

It is the user's responsibility to ensure that a function was annotated for event handler with __dpa_global__. The runtime will not provide any type checking. A mismatched call will result in undefined behavior.

flexio_status flexio_host2dev_memcpy (flexio_process *process, void *src_haddr, size_t buff_bsize, flexio_uintptr_t dest_daddr)

Copy from host memory to a pre-allocted Flex IO heap memory buffer.

Parameters

process

- A pointer to the Flex IO process context.

src_haddr

- An address of the buffer on the host memory.

buff_bsize
- The size of the buffer to copy.

dest_daddr
- Flex IO heap memory buffer address to copy to.

Returns

flexio status value.

Description

This function copies data from a buffer on the host memory to a buffer on the Flex IO heap memory.

flexio_status flexio_log_dev_destroy (flexio_process *process)

Destroys a flexio device messaging default stream environment.

Parameters

process
- A pointer to the Flex IO process.

Returns

flexio status value.

Description

This function destroys and releases all resources, allocated for process messaging needs, which were allocated by flexio_log_dev_init() in purpose of serving the default stream.

flexio_status flexio_log_dev_flush (flexio_process *process)

Flush the default msg stream's buffer in case of asynchronous messaging mode.

Parameters

process
- A pointer to the Flex IO process.

Returns

flexio status value.

Description

All data from the default msg stream buffer will be flushed to the file defined in flexio_log_dev_init().

In case of synchronous device messaging this functions does nothing. This function allocates resources to support messaging from Flex IO to HOST.

flexio_status flexio_log_dev_init (flexio_process *process, flexio_msg_stream_attr_t *stream_fattr, FILE *out, pthread_t *ppthread)

Create environment to support messages output from DPA.

Parameters

process
- A pointer to the Flex IO process.

stream_fattr
- A pointer to the messaging attributes struct.

out
- file to save data from Flex IO. Use stdout if you want receive data on HOST's console

ppthread
- A pointer to receive pthread ID of created thread. May be NULL if user doesn't need it.

Returns

flexio status value.

Description

This function allocates resources to support messages output from Flex IO to HOST. It can only allocate and create the default stream.

Device messaging works in the following modes: synchronous or asynchronous. Under synchronous mode, a dedicated thread starts to receive data and outputs it immediately. When asynchronous mode is in operation, all message stream buffers will be flushed by flexio_log_dev_flush(). Buffer can be overrun.

This function doesn't have a "destroy" procedure. All messaging infrastructure will be closed and the resources will be released using the flexio_process_destroy() function.

FLEXIO_EXPERIMENTAL enum flexio_log_lvl flexio_log_lvl_set (enum flexio_log_lvl lvl)

Sets host SDK logging level.

Parameters

lvl

- logging level to set. All entries with this or higher priority level will be printed.

Returns

the previous host logging level.

Description

This function sets the host logging level. Changing the logging level may change the visibility of some logging entries in the SDK code.

FLEXIO_EXPERIMENTAL uint32_t flexio_mkey_get_id (flexio_mkey *mkey)

Gets the Flex IO MKey ID.

Parameters

mkey

- A pointer to a Flex IO MKey.

Returns

the Flex IO mkey ID or UINT32_MAX on error.

flexio_status flexio_msg_stream_create (flexio_process *process, flexio_msg_stream_attr_t *stream_fattr, FILE *out, pthread_t *ppthread, flexio_msg_stream **stream)

Create a Flex IO msg stream that can contain output messages sent from the DPA.

Parameters

process

- A pointer to the Flex IO process.

stream_fattr
- A pointer to the flexio_msg_stream attributes struct.

out
- file to save data from Flex IO. Use stdout if you want receive data on HOST's console

ppthread
- A pointer to receive pthread ID of created thread. May be NULL if user doesn't need it.

stream
- A pointer to the created stream context pointer.

Returns

flexio status value.

Description

This function can create a flexio_msg_stream that could have device messages directed to it. Directing messages from the device to the host, could be done to any and all open streams, including the default stream.

The function creates the same resources created in flexio_log_dev_init for any new stream. It can also create the default stream. It creates it with the FLEXIO_MSG_DEV_INFO stream level, and that could be modified using flexio_msg_stream_level_set.

flexio_status flexio_msg_stream_destroy (flexio_msg_stream *stream)

Destroys a Flex IO msg stream.

Parameters

stream
- A pointer to the stream context.

Returns

flexio status value.

Description

This function destroys any Flex IO msg stream.

flexio_status flexio_msg_stream_flush (flexio_msg_stream *stream)

Flush a msg stream's buffer in case of asynchronous messaging mode.

Parameters

stream

- A pointer to the Flex IO msg stream.

Returns

flexio status value.

Description

All data from the msg stream buffer will be flushed to the file defined in flexio_msg_stream_create().

In case of synchronous device messaging this functions does nothing. This function allocates resources to support messaging from Flex IO to HOST.

FLEXIO_EXPERIMENTAL int flexio_msg_stream_get_id (flexio_msg_stream *stream)

Gets the Flex IO device message stream's ID (aka file descriptor).

Parameters

stream

- A pointer to a Flex IO message stream.

Returns

the stream_id or -1 in case of error.

Description

Using this function on a destroyed stream will result in unpredictable behavior.

flexio_status flexio_msg_stream_level_set (flexio_msg_stream *stream, flexio_msg_dev_level level)

Change the provided device message stream's level.

Parameters

stream

- A pointer to a Flex IO message stream.

level

- The new desired level, ranges between FLEXIO_MSG_DEV_NO_PRINT FLEXIO_MSG_DEV_DEBUG. FLEXIO_MSG_DEV_ALWAYS_PRINT cannot be used here.

Returns

flexio status value.

Description

The default stream's level cannot be altered. Note that modifying the stream's level while messages are being sent may result in missing or unwanted messages.

flexio_status flexio_outbox_create (flexio_process *process, flexio_outbox_attr *fattr, flexio_outbox **outbox)

Creates a Flex IO outbox.

Parameters

process

- A pointer to the Flex IO process.

fattr

- A pointer to the outbox attributes struct.

outbox

- A pointer to the created outbox context pointer.

Returns

flexio status value.

Description

This function Creates a Flex IO outbox for the given process.

flexio_status flexio_outbox_destroy (flexio_outbox *outbox)

Destroys a Flex IO outbox.

Parameters

outbox

- A pointer to a outbox context.

Returns

flexio status value.

Description

This function destroys a Flex IO outbox.

FLEXIO_EXPERIMENTAL uint32_t flexio_outbox_get_id (flexio_outbox *outbox)

Gets the Flex IO outbox ID.

Parameters

outbox

- A pointer to a Flex IO outbox.

Returns

the Flex IO outbox ID or UINT32_MAX on error.

FLEXIO_EXPERIMENTAL flexio_uar *flexio_outbox_get_uar (flexio_outbox *outbox)

Gets a Flex IO UAR object from a Flex IO outbox.

Parameters

outbox

- A pointer to a Flex IO outbox.

Returns

the Flex IO outbox UAR object or NULL on error.

flexio_status flexio_process_call (flexio_process *process, flexio_func_t *host_func, uint64_t *func_ret, ...)

Calls a Flex IO process.

Parameters

process
- A pointer to the Flex IO process to run.

host_func
- The host stub function that is used by the application to reference the device function.

func_ret
- A pointer to the ELF function return value.

Returns

flexio status value.

flexio_status flexio_process_create (ibv_context *ibv_ctx, flexio_app *app, const flexio_process_attr *process_attr, flexio_process **process_ptr)

Create a new Flex IO process.

Parameters

ibv_ctx
- A pointer to a device context.

app
- Device side application handle.

process_attr
- Optional, process attributes for create. Can be NULL.

process_ptr
- A pointer to the created process pointer.

Returns

flexio status value.

Description

This function creates a new Flex IO process with requested image.

flexio_status flexio_process_destroy (flexio_process *process)

Destroys a Flex IO process.

Parameters

process

- A pointer to a process. NULL is a valid value.

Returns

flexio status value.

Description

This function destroys a Flex IO process.

flexio_status flexio_process_error_handler_set (flexio_process *process, flexio_func_t *error_handler)

Set the Flexio process error handler.

Parameters

process

- A pointer to a process

error_handler

- The host stub function that is used as a reference to the error handler function.

Returns

flexio status value.

Description

This function sets the Flex IO process error handler. The error handler must be set after the process is created, and before the first thread is created. The function registered for error handler should be annotated with __dpa_global__.

FLEXIO_EXPERIMENTAL uint32_t flexio_process_get_dumem_id (flexio_process *process)

Gets the Flex IO process DUMEM ID.

Parameters

process

- A pointer to a Flex IO process.

Returns

the Flex IO process DUMEM ID or UINT32_MAX on error.

FLEXIO_EXPERIMENTAL ibv_pd *flexio_process_get_pd (flexio_process *process)

Gets a Flex IO IBV PD object from a Flex IO process.

Parameters

process

- A pointer to a Flex IO process.

Returns

the process's PD object or NULL on error.

FLEXIO_EXPERIMENTAL flexio_uar *flexio_process_get_uar (flexio_process *process)

Gets a Flex IO UAR object from a Flex IO process.

Parameters

process

- A pointer to a Flex IO process.

Returns

the Flex IO process UAR object or NULL on error.

flexio_status flexio_process_mem_info_get (const flexio_process *process, flexio_heap_mem_info *info)

Get process memory info.

Parameters

process

- A pointer to the Flex IO process context.

info

- A pointer to flexio_heap_mem_info struct to fill info.

Returns

flexio status value.

Description

This function returns the process heap memory base address and its available size.

FLEXIO_EXPERIMENTAL uint64_t flexio_process_udbg_token_get (flexio_process *process)

Get token for Flex IO process debug access.

Parameters

process

- A pointer to the Flex IO process context.

Returns

the requested token. Zero value means - User Debug access for the process is not allowed.

Description

This function returns the token, needed for user debug syscalls access.

flexio_status flexio_qp_create (flexio_process *process, ibv_context *ibv_ctx, flexio_qp_attr *qp_fattr, flexio_qp **qp_ptr)

Creates a Flex IO QP.

Parameters

process
- A pointer to the Flex IO process.

ibv_ctx
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

qp_fattr
- A pointer to the QP attributes struct.

qp_ptr
- A pointer to the created QP context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO QP.

flexio_status flexio_qp_destroy (flexio_qp *qp)

Destroys a Flex IO QP.

Parameters

qp
- A pointer to the QP context.

Returns

flexio status value.

Description

This function destroys a Flex IO QP.

FLEXIO_EXPERIMENTAL uint32_t flexio_qp_get_qp_num (flexio_qp *qp)

Gets the Flex IO QP number.

Parameters

qp
- A pointer to a Flex IO QP.

Returns

the QP number or UINT32_MAX on error.

flexio_status flexio_qp_modify (flexio_qp *qp, flexio_qp_attr *fattr, flexio_qp_attr_opt_param_mask *mask)

Modify Flex IO QP.

Parameters

qp
- A pointer to the QP context.

fattr
- A pointer to the QP attributes struct that will also define the QP connection.

mask
- A pointer to the optional QP attributes mask.

Returns

flexio status value.

Description

This function modifies Flex IO QP and transition it between states. At the end of the procedure Flex IO QP would have moved from it's current state to to next state, given in the fattr, if the move is a legal transition in the QP's state machine.

flexio_qp_state flexio_qp_state_get (flexio_qp *qp)

retrieve the device QP state.

Parameters

qp
- A pointer to a Flex IO QP.

Returns

enum flexio_qp_state.

Description

This function return the device QP state it is currently in.

flexio_status flexio_recoverable_buf_dev_alloc (flexio_process *process, size_t buff_bsize, uint32_t mkey, flexio_uintptr_t *dest_daddr_p)

Allocates a recoverable buffer on Flex IO heap memory.

Parameters

process
- A pointer to the Flex IO process context.

buff_bsize
- The size of the buffer to allocate.

mkey
- An MKey ID to the user buffer for memory dump on object destroy.

dest_daddr_p
- A pointer to the Flex IO address, where the buffer was allocated.

Returns

flexio status value.

Description

This function allocates a recoverable buffer with the requested size on the Flex IO heap memory. This buffer will force a memory object to be created and on object destroy will dump its content to the user supplied buffer.

Notes: (1) The created memory object is in the resolution of DPA memory block size so if buff_bsize is not a multiple of DPA memory block size the actual allocation would round up. (2) MKey size must match the buffer size (including rounding-up as mentioned above).

On success - sets dest_daddr_p to the start address of the allocated buffer. On Failure - sets dest_daddr_p to 0x0.

flexio_status flexio_rmp_create (flexio_process *process, ibv_context *ibv_ctx, const flexio_wq_attr *fattr, flexio_rmp **flexio_rmp_ptr)

Creates a Flex IO RMP.

Parameters

process
- A pointer to the Flex IO process.

ibv_ctx
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

fattr
- A pointer to the WQ attributes struct.

flexio_rmp_ptr
- A pointer to the created RMP context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO RMP.

flexio_status flexio_rmp_destroy (flexio_rmp *flexio_rmp)

Destroys a Flex IO RMP.

Parameters

flexio_rmp
- A pointer to an RMP context.

Returns

flexio status value.

Description

This function destroys a Flex IO RMP.

FLEXIO_EXPERIMENTAL uint32_t flexio_rmp_get_wq_num (flexio_rmp *rmp)

Gets the Flex IO RMP number.

Parameters

rmp
- A pointer to a Flex IO RMP.

Returns

the RQ number or UINT32_MAX on error.

flexio_status flexio_rq_create (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_rq **flexio_rq_ptr)

Creates a Flex IO RQ.

Parameters

process
- A pointer to the Flex IO process.

ibv_ctx
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

cq_num
- A CQ number.

fattr
- A pointer to the RQ WQ attributes struct.

flexio_rq_ptr
- A pointer to the created RQ context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO RQ.

flexio_status flexio_rq_create_cross_dev (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_rq **flexio_rq_ptr)

Creates a Flex IO RQ assuming cross device.

Parameters

process
- A pointer to the Flex IO process.

ibv_ctx
- A pointer to an IBV device context (should be different than process'). If NULL or same as process - will result in an error.

cq_num
- A CQ number.

fattr
- A pointer to the RQ WQ attributes struct.

flexio_rq_ptr
- A pointer to the created RQ context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO RQ assuming cross VHCA_ID without checking.

flexio_status flexio_rq_destroy (flexio_rq *flexio_rq)

Destroys a Flex IO RQ.

Parameters

flexio_rq
- A pointer to an RQ context.

Returns

flexio status value.

Description

This function destroys a Flex IO RQ.

FLEXIO_EXPERIMENTAL mlx5dv_devx_obj *flexio_rq_get_object (flexio_rq *rq)

Get the Flex IO RQ object.

Parameters

rq
- A pointer to the RQ context.

Returns

the RQ devx object.

Description

This function returns the Flex IO RQ object.

FLEXIO_EXPERIMENTAL mlx5dv_devx_obj *flexio_rq_get_tir (flexio_rq *rq)

Gets the Flex IO RQ TIR object.

Parameters

rq
- A pointer to a Flex IO RQ.

Returns

the RQ TIR object or NULL on error.

FLEXIO_EXPERIMENTAL uint32_t flexio_rq_get_wq_num (flexio_rq *rq)

Gets the Flex IO RQ number.

Parameters

rq
- A pointer to a Flex IO RQ.

Returns

the RQ number or UINT32_MAX on error.

flexio_status flexio_rq_set_err_state (flexio_rq *rq)

Sets a Flex IO RQ to error state.

Parameters

rq
- A pointer to the RQ context to move to error state.

Returns

flexio status value.

Description

This function sets a Flex IO RQ to error state.

flexio_status flexio_sq_create (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_sq **flexio_sq_ptr)

Creates a Flex IO SQ.

Parameters

process
- A pointer to the Flex IO process.

ibv_ctx
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

cq_num
- A CQ number (can be Flex IO or host CQ).

fattr
- A pointer to the SQ attributes struct.

flexio_sq_ptr
- A pointer to the created SQ context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO SQ.

flexio_status flexio_sq_create_cross_dev (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_sq **flexio_sq_ptr)

Creates a Flex IO SQ assuming cross device.

Parameters

process

- A pointer to the Flex IO process.

ibv_ctx

- A pointer to an IBV device context (should be different than process'). If NULL or same as process - will result in an error.

cq_num

- A CQ number (can be Flex IO or host CQ).

fattr

- A pointer to the SQ attributes struct.

flexio_sq_ptr

- A pointer to the created SQ context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO SQ assuming cross VHCA_ID without checking.

flexio_status flexio_sq_destroy (flexio_sq *flexio_sq)

Destroys a Flex IO SQ.

Parameters

flexio_sq

- A pointer to an SQ context.

Returns

flexio status value.

Description

This function destroys a Flex IO SQ.

FLEXIO_EXPERIMENTAL uint32_t flexio_sq_get_wq_num (flexio_sq *sq)

Gets the Flex IO SQ number.

Parameters

sq
- A pointer to a Flex IO SQ.

Returns

the SQ number or UINT32_MAX on error.

flexio_status flexio_sq_tis_create (ibv_context *ibv_ctx, flexio_transport_domain *td, flexio_sq_tis **tis)

Creates a Flex IO SQ transport interface send (TIS) object.

Parameters

ibv_ctx
- A pointer to an IBV device context. must be the same as the SQ's that will use this TIS.

td
- A pointer to a Flex IO transport domain struct. TD must be created for the same IBV device context provided for TIS creation.

tis
- A pointer to the created SQ TIS context pointer.

Returns

flexio status value.

Description

This function creates a Flex IO SQ TIS for allowing steering from an SQ.

flexio_status flexio_sq_tis_destroy (flexio_sq_tis *tis)

Destroys a Flex IO SQ TIS.

Parameters

tis
- A pointer to an SQ TIS context.

Returns

flexio status value.

Description

This function destroys a Flex IO TIS.

flexio_status flexio_transport_domain_create (ibv_context *ibv_ctx, flexio_transport_domain **td)

Create a Flex IO transport domain (TD).

Parameters

ibv_ctx
- A pointer to an IBV device context.

td
- A pointer to the created TD context pointer.

Returns

flexio status value.

Description

This function creates a TD for the given IBV device context.

flexio_status flexio_transport_domain_destroy (flexio_transport_domain *td)

Destroy a Flex IO transport domain (TD).

Parameters

td
- A pointer to a TD context.

Returns

flexio status value.

Description

This function destroys a Flex IO TD.

flexio_status flexio_uar_create (flexio_process *process, flexio_uar **flexio_uar)

Creates a Flex IO UAR object.

Parameters

process

- A pointer to the Flex IO process context.

flexio_uar

- A pointer to a pointer to the created Flex IO UAR struct.

Returns

flexio status value.

Description

This function creates a Flex IO UAR object.

flexio_status flexio_uar_destroy (flexio_uar *uar)

destroys a Flex IO UAR object

Parameters

uar

- A pointer to the Flex IO UAR to destroy.

Returns

flexio status value.

Description

This function destroys a Flex IO UAR object.

flexio_status flexio_uar_extend (flexio_uar *in_uar, ibv_context *to_extend, flexio_uar **extended)

Extend UAR to an ibv context.

Parameters

in_uar

- A pointer to the Flex IO uar.

to_extend
- A pointer to an IBV device context to be extended to.

extended
- A pointer to the UAR context pointer.

Returns

flexio status value.

Description

This function extend the UAR to an ibv context to allow handling its queues.

flexio_uar_device_id flexio_uar_get_extended_id (flexio_uar *uar)

Gets the Flex IO extended UAR ID.

Parameters

uar
- A pointer to a Flex IO extended UAR.

Returns

the Flex IO UAR extended ID or UINT32_MAX on error.

FLEXIO_EXPERIMENTAL uint32_t flexio_uar_get_id (flexio_uar *uar)

Gets the Flex IO UAR ID.

Parameters

uar
- A pointer to a Flex IO UAR.

Returns

the Flex IO UAR ID or UINT32_MAX on error.

flexio_status flexio_version_set (uint64_t version)

Set version for flexio.

Parameters

version
- version to set in format FLEXIO_VER(major, minor, 0).

Returns

flexio status value.

Description

This function allows the library to determine which version it should work with. The function is called without error only once; if called again with a different version, it returns an error. The function also returns an error if it is called with a version greater than the FLEXIO_CURRENT_VERSION.

flexio_status flexio_window_create (flexio_process *process, ibv_pd *pd, flexio_window **window)

Creates a Flex IO window.

Parameters

process

- A pointer to the Flex IO process.

pd

- A pointer to a protection domain struct to the memory the window should access.

window

- A pointer to the created window context pointer.

Returns

flexio status value.

Description

This function Creates a Flex IO window for the given process.

flexio_status flexio_window_destroy (flexio_window *window)

Destroys a Flex IO window.

Parameters

window

- A pointer to a window context.

Returns

flexio status value.

Description

This function destroys a Flex IO window.

FLEXIO_EXPERIMENTAL uint32_t flexio_window_get_id (flexio_window *window)

Gets the Flex IO window ID.

Parameters

window

- A pointer to a Flex IO window.

Returns

the Flex IO window ID or UINT32_MAX on error.

#define FLEXIO_EXPERIMENTAL __attribute__((deprecated("Symbol is defined as experimental"), \ section(".text.experimental"))) \ FLEXIO_STABLE

To set a Symbol (or specifically a function) as experimental.

#define FLEXIO_MAX_NAME_LEN (256)

Maximum length of application and device function names

2.3. Flex IO SDK dev

Flex IO SDK device API for DPA programs. Includes services for DPA programs.

struct spinlock_s

Flex IO SDK dev error handling

Flex IO SDK dev queue access

Flex IO SDK dev queue types

enum cq_ce_mode

Flex IO dev CQ CQE creation modes.

Values

MLX5_CTRL_SEG_CE_CQE_ON_CQE_ERROR = 0x0
MLX5_CTRL_SEG_CE_CQE_ON_FIRST_CQE_ERROR = 0x1
MLX5_CTRL_SEG_CE_CQE_ALWAYS = 0x2
MLX5_CTRL_SEG_CE_CQE_AND_EQE = 0x3

enum flexio_dev_nic_counter_ids

Flex IO dev NIC counters ID enumeration.

Values

FLEXIO_DEV_NIC_COUNTER_PORT0_RX_BYTES = 0x10
FLEXIO_DEV_NIC_COUNTER_PORT1_RX_BYTES = 0x11
FLEXIO_DEV_NIC_COUNTER_PORT2_RX_BYTES = 0x12
FLEXIO_DEV_NIC_COUNTER_PORT3_RX_BYTES = 0x13
FLEXIO_DEV_NIC_COUNTER_PORT0_TX_BYTES = 0x20
FLEXIO_DEV_NIC_COUNTER_PORT1_TX_BYTES = 0x21
FLEXIO_DEV_NIC_COUNTER_PORT2_TX_BYTES = 0x22
FLEXIO_DEV_NIC_COUNTER_PORT3_TX_BYTES = 0x23

enum flexio_dev_status_t

Return status of Flex IO dev API functions.

Values

FLEXIO_DEV_STATUS_SUCCESS = 0
FLEXIO_DEV_STATUS_FAILED = 1

enum flexio_window_entity

Flex IO dev windows entity.

Values

FLEXIO_DEV_WINDOW_ENTITY_0 = 0
FLEXIO_DEV_WINDOW_ENTITY_1 = 1
FLEXIO_DEV_WINDOW_ENTITY_NUM = 2

typedef uint64_t (flexio_dev_arg_unpack_func_t)

Unpack the arguments and call the user function.

This callback function is used at runtime to unpack the arguments from the call on Host and then call the function on DPA. This function is called internally from flexio dev.

argbuf - Argument buffer that was written by Host. func - Function pointer to user function.

return uint64_t - result of the RPC function.

typedef void (flexio_dev_async_rpc_handler_t)

Asynchronous RPC handler callback function type.

Defines an RPC handler callback function.

arg - argument of the RPC function.

return void.

typedef void (flexio_dev_event_handler_t)

Event handler callback function type.

Defines an event handler callback function. On handler function end, need to call flexio_dev_process_finish() instead of a regular return statement, in order to properly release resources back to the OS.

thread_arg - an argument for the executing thread.

return void.

typedef uint64_t (flexio_dev_rpc_handler_t)

RPC handler callback function type.

Defines an RPC handler for most useful callback function.

arg - argument of the RPC function.

return uint64_t - result of the RPC function.

typedef uint32_t flexio_uar_device_id

Flex IO UAR extension ID prototype.

flexio_dev_status_t flexio_dev_cross_device_ring_db (flexio_uar_device_id device_id, uint32_t qpn, uint32_t pi)

send a doorbell to a QP on another device ID

Parameters

device_id
- The device ID.

qpn
- QP number to send doorbell on.

pi
- doorbell producer index.

Returns

flexio_dev_status_t

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_event_handler_activate (uint32_t activation_id)

Activate an event handler thread.

Parameters

activation_id
- The event handler activation ID.

Returns

void.

Description

Using activation id, activate (trigger) the event handler with that activation id. Note that the activated event handler must be of same process as the activating thread.

FLEXIO_DEV_EXPERIMENTAL uint64_t flexio_dev_get_pcc_table_base (uint16_t vhca_id)

get programable congestion control table base address

Parameters

vhca_id

- PCC table VHCA_ID.

Returns

PCC table base address for the given VHCA_ID.

Description

This function gets the programable congestion control table base address.

FLEXIO_DEV_EXPERIMENTAL int flexio_dev_get_thread_ctx (flexio_dev_thread_ctx **dtctx)

Request thread context.

Parameters

dtctx

- A pointer to a pointer of flexio_dev_thread_ctx structure.

Returns

0 on success negative value on failure.

Description

This function requests the thread context. Should be called for every start of thread.

FLEXIO_DEV_EXPERIMENTAL uint32_t flexio_dev_get_thread_id (flexio_dev_thread_ctx *dtctx)

Get thread ID from thread context.

Parameters

dtctx

- A pointer to a flexio_dev_thread_ctx structure.

Returns

thread ID value.

Description

This function queries a thread context for its thread ID (from thread metadata).

flexio_uintptr_t flexio_dev_get_thread_local_storage (flexio_dev_thread_ctx *dtctx)

Get thread local storage address from thread context.

Parameters

dtctx

- A pointer to a flexio_dev_thread_ctx structure.

Returns

thread local storage value.

Description

This function queries a thread context for its thread local storage (from thread metadata).

FLEXIO_DEV_EXPERIMENTAL int flexio_dev_msg (int stream_id, flexio_msg_dev_level level, const char *format, ...)

Creates message entry and outputs from the device to the host side.

Parameters

stream_id
- the relevant msg stream, created and passed from the host.

level
- messaging level.

format
- same as for regular printf.

Returns

- same as from regular printf.

Description

Same as a regular printf but with protection from simultaneous print from different threads.

flexio_dev_status_t flexio_dev_multi_window_config (flexio_window_entity win_entity, uint16_t window_config_id, uint32_t mkey)

Config thread window object.

Parameters

win_entity
- The window entity to configure.

window_config_id
- The window object id.

mkey
- The mkey id.

Returns

flexio_dev_status_t.

Description

This function updates the thread window object of the given thread context.

flexio_dev_status_t flexio_dev_multi_window_copy_from_host (flexio_window_entity win_entity, void *daddr, uint64_t haddr, uint32_t size)

Copy a buffer from host memory to device memory.

Parameters

win_entity
- The window entity to configure.

daddr
- A pointer to the device memory buffer.

haddr
- A pointer to the host memory allocated buffer.

size
- Number of bytes to copy.

Returns

flexio_dev_status_t.

Description

This function copies specified number of bytes from host memory to device memory. UNSUPPORTED at this time.

flexio_dev_status_t flexio_dev_multi_window_copy_to_host (flexio_window_entity win_entity, uint64_t haddr, const void *daddr, uint32_t size)

Copy a buffer from device memory to host memory.

Parameters

win_entity
- The window entity to configure.

haddr
- A pointer to the host memory allocated buffer.

daddr
- A pointer to the device memory buffer.

size
- Number of bytes to copy.

Returns

flexio_dev_status_t.

Description

This function copies specified number of bytes from device memory to host memory.

flexio_dev_status_t flexio_dev_multi_window_mkey_config (flexio_window_entity win_entity, uint32_t mkey)

Config thread window mkey object.

Parameters

win_entity
- The window entity to configure.

mkey
- The mkey id.

Returns

flexio_dev_status_t.

Description

This function updates the thread window mkey object of the given thread context.

flexio_dev_status_t flexio_dev_multi_window_ptr_acquire (flexio_window_entity win_entity, uint64_t haddr, flexio_uintptr_t *daddr)

Generate device address from host allocated memory.

Parameters

win_entity
- The window entity to configure.

haddr
- Host allocated address.

daddr
- A pointer to write the device generated matching address.

Returns

flexio_dev_status_t.

Description

This function generates a memory address to be used by device to access host side memory, according to already create window object. from a host allocated address.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_nic_counters_config (uint32_t *counter_values, uint32_t *counter_ids, uint32_t num_counters)

Prepare a list of counters to read.

Parameters

counter_values
- buffer to store counters values (32b) read by flexio_dev_nic_counters_sample().

counter_ids
- An array of counter ids.

num_counters
- number of counters in the counter_ids array.

Returns

void process crashes in case of: counters_ids too large bad pointers of values, counter_ids unknown counter

Description

The list is stored in kernel memory. A single counters config per process is supported. Note that arrays memory must be defined in global or heap memory only.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_nic_counters_sample (void)

Sample counters according to the prior configuration call.

Returns

void. process crashes in case of: flexio_dev_config_nic_counters() never called

Description

Sample counter_ids, num_counters and values buffer provided in the last successful call to flexio_dev_config_nic_counters(). This call ensures fastest sampling on a pre-checked counter ids and buffers.

flexio_dev_status_t flexio_dev_outbox_config (flexio_dev_thread_ctx *dtctx, uint16_t outbox_config_id)

Config thread outbox object.

Parameters

dtctx

- A pointer to flexio_dev_thread_ctx structure.

outbox_config_id

- The outbox object config id.

Returns

flexio_dev_status_t.

Description

This function updates the thread outbox object of the given thread context.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_outbox_config_fast (flexio_dev_thread_ctx *dtctx, uint16_t outbox_config_id)

Config thread outbox object without any checks.

Parameters

dtctx

- A pointer to flexio_dev_thread_ctx structure.

outbox_config_id

- The outbox object config id.

Returns

Function does not return.

Description

This function updates the thread outbox object of the given thread context, but it doesn't check for correctness or redundancy (same ID as current configured).

flexio_dev_status_t flexio_dev_outbox_config_uar_extension (flexio_dev_thread_ctx *dtctx, flexio_uar_device_id device_id)

set extension ID for outbox

Parameters

dtctx

- A pointer to a flexio_dev_thread_ctx structure.

device_id

- The device ID.

Returns

flexio_dev_status_t.

Description

This function sets the device ID for the outbox to operate on.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_process_finish (void)

Exit flexio process (no errors).

Returns

Function does not return.

Description

This function releases resources back to OS and returns '0x40' in dpa_process_status. All threads for the current process will stop executing and no new threads will be able to trigger for this process. Threads state will NOT be changes to 'finished' (will remain as is).

FLEXIO_DEV_EXPERIMENTAL int flexio_dev_puts (flexio_dev_thread_ctx *dtctx, char *str)

Put a string to messaging queue.

Parameters

dtctx

- A pointer to a pointer of flexio_dev_thread_ctx structure.

str

- A pointer to string.

Returns

length of messaged string.

Description

This function puts a string to host's default stream messaging queue. This queue has been serviced by host application. Would have no effect, if the host application didn't configure device messaging stream environment. In order to initialize/configure device messaging environment - On HOST side - after flexio_process_create, a stream should be created, therefore flexio_msg_stream_create should be called, and the default stream should be created. On DEV side - before using flexio_dev_puts, the thread context is needed, therefore flexio_dev_get_thread_ctx should be called before.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_thread_finish (void)

Exit from a thread, mark it as finished.

Returns

Function does not return.

Description

This function releases resources back to OS. The thread will be marked as finished so next DUAR will not trigger it.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_thread_reschedule (void)

Exit from a thread, leave process active.

Returns

Function does not return.

Description

This function releases resources back to OS. For the next DUAR the thread will restart from the beginning.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_thread_retrigger (void)

Exit from a thread, and retrigger it.

Returns

Function does not return.

Description

This function asks the OS to retrigger the thread. The thread will not wait for the next DUAR to be triggered but will be triggered immediately.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_yield (flexio_dev_thread_ctx *dtctx)

exit point for continuable event handler routine

Parameters

dtctx

- A pointer to a flexio_dev_thread_ctx structure.

Returns

Function does not return.

Description

This function is used to mark the exit point on continuable event handler where user wishes to continue execution on next event. In order to use this API the event handler must be created with continuable flag enabled, otherwise call will have no effect.

#define FLEXIO_DEV_EXPERIMENTAL __attribute__((deprecated("Symbol is defined as experimental"), \ section(".text.experimental"))) \ FLEXIO_DEV_STABLE

To set a Symbol (or specifically a function) as experimental.

#define flexio_dev_msg_broadcast flexio_dev_msg(FLEXIO_MSG_DEV_BROADCAST_STREAM, lvl, __VA_ARGS__)

Create message entry and outputs from the device to all of the host's open streams. Same as a regular printf but with protection from simultaneous print from different threads.

#define flexio_dev_msg_dflt flexio_dev_msg(FLEXIO_MSG_DEV_DEFAULT_STREAM_ID, lvl, __VA_ARGS__)

Create message entry and outputs from the device to host's default stream. Same as a regular printf but with protection from simultaneous print from different threads.

#define flexio_dev_print flexio_dev_msg(FLEXIO_MSG_DEV_DEFAULT_STREAM_ID, FLEXIO_MSG_DEV_INFO, __VA_ARGS__)

Create message entry and outputs from the device to host's default stream, with FLEXIO_MSG_DEV_INFO message level. Same as a regular printf but with protection from simultaneous print from different threads.

#define spin_init __atomic_store_n(&((lock)->locked), 0, __ATOMIC_SEQ_CST)

Initialize a spinlock mechanism.

Initialize a spinlock mechanism, must be called before use.

#define spin_lock do { \ while (__atomic_exchange_n(&((lock)->locked), 1, __ATOMIC_SEQ_CST)) {;} \ } while (0)

Lock a spinlock mechanism.

Lock a spinlock mechanism.

#define spin_trylock __atomic_exchange_n(&((lock)->locked), 1, __ATOMIC_SEQ_CST)

Atomic try to catch lock.

makes attempt to take lock. Returns immediately.

#define spin_unlock __atomic_store_n(&((lock)->locked), 0, __ATOMIC_SEQ_CST)

Unlock a spinlock mechanism.

Unlock a spinlock mechanism.

2.3.1. Flex IO SDK dev error handling

Flex IO SDK dev

Flex IO SDK device API for DPA programs error handling.

enum flexio_dev_error_t

Flex IO dev errors.

Values

FLEXIO_DEV_ERROR_ILLEGAL_ERR = 0x42
Illegal user error code

FLEXIO_DEV_EXPERIMENTAL __attribute__ ((__noreturn__))

Exit the process and return a user (fatal) error code.

Returns

- function does not return

Description

Error codes returned to the host in the dpa_process_status field of the DPA_PROCESS object are defined as follows: 0: OK 1-63: RTOS or Firmware errors 64-127: Flexio-SDK errors 128-255: User defined

FLEXIO_DEV_EXPERIMENTAL uint64_t flexio_dev_get_and_rst_errno (flexio_dev_thread_ctx *dtctx)

Get and Reset thread error flag (errno) of recoverable (non fatal) error.

Parameters

dtctx
- A pointer to a flexio_dev_thread_ctx structure.

Returns

- thread error code.

FLEXIO_DEV_EXPERIMENTAL uint64_t flexio_dev_get_errno (flexio_dev_thread_ctx *dtctx)

Get thread error flag (errno) of recoverable (non fatal) error.

Parameters

dtctx
- A pointer to a flexio_dev_thread_ctx structure.

Returns

thread error code.

Description

This function queries an errno field from thread context.

FLEXIO_DEV_EXPERIMENTAL void flexio_dev_rst_errno (flexio_dev_thread_ctx *dtctx)

Reset thread error flag (errno) of recoverable (non fatal) error.

Parameters

dtctx

- A pointer to a flexio_dev_thread_ctx structure.

Returns

- void.

2.3.2. Flex IO SDK dev queue access

Flex IO SDK dev

Flex IO SDK device API for DPA programs queue access. Provides an API for handling networking queues (WQs/CQs).

enum flexio_ctrl_seg_t

Flex IO dev WQE control segment types.

Values

FLEXIO_CTRL_SEG_SEND_EN = 0
FLEXIO_CTRL_SEG_SEND_RC = 1
FLEXIO_CTRL_SEG_LDMA = 2
FLEXIO_CTRL_SEG_RDMA_WRITE = 3
FLEXIO_CTRL_SEG_RDMA_READ = 4
FLEXIO_CTRL_SEG_ATOMIC_COMPARE_AND_SWAP = 5
FLEXIO_CTRL_SEG_LSO = 6
FLEXIO_CTRL_SEG_NOP = 7
FLEXIO_CTRL_SEG_RDMA_WRITE_IMM = 8
FLEXIO_CTRL_SEG_TRANSPOSE = 9

enum flexio_dev_cc_db_next_act_t

Flex IO dev congestion control next action types.

Values

CC_DB_NEXT_ACT_SINGLE = 0x0
CC_DB_NEXT_ACT_MULTIPLE = 0x1
CC_DB_NEXT_ACT_FW = 0x2

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_cc_ring_db (uint16_t ccq_id, uint32_t rate, uint32_t rtt_req, flexio_dev_cc_db_next_act_t next_act)

Rings CC doorbell.

Parameters

ccq_id
- CC queue ID to update.

rate
- Rate to set.

rtt_req
- RTT measure request to set.

next_act
- Next action to set.

Returns

void.

Description

This function rings CC doorbell for the requested CC queue, which sets the requested rate, RTT request and next action.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_cc_ring_db_high64bit (uint64_t value)

Write to high 64 bits of CC doorbell register.

Parameters

value
- 64 bits value to write.

Returns

void.

Description

This function writes to high 64 bits of CC doorbell register. Register value should be prepared by caller according to register layout.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_cc_ring_db_low64bit (uint64_t value)

Write to low 64 bits of CC doorbell register.

Parameters

value

- 64 bits value to write.

Returns

void.

Description

This function writes to low 64 bits of CC doorbell register. Register value should be prepared by caller according to register layout.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_comp_cq_init (flexio_dev_cqe64 *cqe, int num_cqes, uint8_t validity_iteration_count)

init CQEs according to compressed feature requirement.

Parameters

cqe

- first CQE in range to init.

num_cqes

- Number of CQEs in range.

validity_iteration_count

- validity_iteration_count field value to init.

Returns

void.

FLEXIO_DEV_ALWAYS_INLINE uint64_t flexio_dev_comp_cqe_get_comp_cqe (flexio_dev_cqe64 *cqe, int comp_cqe_index)

Get a mini CQE from CQE.

Parameters

cqe
- CQE to parse.

comp_cqe_index
- index of mini CQE to return from array.

Returns

uint64_t - the mini CQE value.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_comp_cqe_get_num_comp_cqes (flexio_dev_cqe64 *cqe)

Get number of mini CQEs in CQE.

Parameters

cqe
- CQE to parse.

Returns

uint8_t - number of mini CQEs in array.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_comp_cqe_get_validity_byte (flexio_dev_cqe64 *cqe)

Get the validity iteration count byte value from CQE.

Parameters

cqe
- CQE to parse.

Returns

uint8_t - validity byte value.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_comp_cqe_is_comp_cqe_array (flexio_dev_cqe64 *cqe)

returns true if CQE is a mini CQE array

Parameters

cqe
- CQE to parse.

Returns

uint8_t - true for mini cqe array.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_cq_arm (uint32_t ci, uint32_t qnum)

Arm CQ function.

Parameters

ci
- Current CQ consumer index.

qnum
- Number of the CQ to arm.

Returns

void.

Description

Moves a CQ to 'armed' state. This means that next CQE created for this CQ will result in an EQE on the relevant EQ.

FLEXIO_DEV_ALWAYS_INLINE uint32_t flexio_dev_cqe_get_byte_cnt (flexio_dev_cqe64 *cqe)

Get byte count field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint32_t - Byte count field value of the CQE.

Description

Parse a CQE for its byte count field.

FLEXIO_DEV_ALWAYS_INLINE uint16_t flexio_dev_cqe_get_csum_ok (flexio_dev_cqe64 *cqe)

Get csum OK field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint16_t - csum_ok field value of the CQE.

Description

Parse a CQE for its csum OK field.

FLEXIO_DEV_ALWAYS_INLINE uint32_t flexio_dev_cqe_get_err_synd (flexio_dev_cqe64 *cqe)

Get error syndrome field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint32_t - Error syndrome field value of the CQE.

Description

Parse a CQE for its error syndrome field.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_cqe_get_opcode (flexio_dev_cqe64 *cqe)

Get the opcode field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint8_t - Opcode field value of the CQE.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_cqe_get_owner (flexio_dev_cqe64 *cqe)

Get owner field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint8_t - Owner field value of the CQE.

Description

Parse a CQE for its owner field.

FLEXIO_DEV_ALWAYS_INLINE uint32_t flexio_dev_cqe_get_qpn (flexio_dev_cqe64 *cqe)

Get QP number field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint32_t - QP number field value of the CQE.

Description

Parse a CQE for its QP number field.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_cqe_get_type (flexio_dev_cqe64 *cqe)

Get the type of CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint8_t - the type of the CQE. 0 - no inline data 1 - inline data in the data 32 segment 2 - inline data in the data 64 segment 3 - Compressed CQE

FLEXIO_DEV_ALWAYS_INLINE uint32_t flexio_dev_cqe_get_user_index (flexio_dev_cqe64 *cqe)

Get the user index field from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint32_t - User index field value of the CQE.

FLEXIO_DEV_ALWAYS_INLINE uint16_t flexio_dev_cqe_get_wqe_counter (flexio_dev_cqe64 *cqe)

Get WQE counter filed from CQE function.

Parameters

cqe
- CQE to parse.

Returns

uint16_t - WQE counter field value of the CQE.

Description

Parse a CQE for its WQE counter field.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_db_ctx_arm (uint32_t qnum, uint32_t emu_ctx_id)

arm the emulation context

Parameters

qnum
- Number of the queue provided by host.

emu_ctx_id
- Emulation context ID, provided by a call on the host to flexio_emu_db_to_cq_ctx_get_id.

Returns

void.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_db_ctx_force_trigger (uint32_t cqn, uint32_t emu_ctx_id)

force trigger of emulation context

Parameters

cqn
- CQ number provided by host.

emu_ctx_id
- Emulation context ID, provided by a call on the host to flexio_emu_db_to_cq_ctx_get_id.

Returns

void.

flexio_dev_status_t flexio_dev_dbr_cq_set_ci (uint32_t *cq_dbr, uint32_t ci)

Set consumer index value for a CQ function.

Parameters

cq_dbr
- A pointer to the CQ's doorbell record address.

ci
- The consumer index value to update.

Returns

flexio_dev_status_t.

Description

Writes an updated consumer index number to a CQ's doorbell record

flexio_dev_status_t flexio_dev_dbr_rq_inc_pi (uint32_t *rq_dbr)

Increment producer index of an RQ by 1 function.

Parameters

rq_dbr

- A pointer to the CQ's doorbell record address.

Returns

flexio_dev_status_t.

Description

Mark a WQE for reuse by incrementing the relevant RQ producer index by 1

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_eq_update_ci (uint32_t ci, uint32_t qnum)

Update an EQ consumer index function.

Parameters

ci

- Current EQ consumer index.

qnum

- Number of the EQ to update.

Returns

void.

Description

Updates the consumer index of an EQ after handling an EQE.

FLEXIO_DEV_ALWAYS_INLINE uint32_t flexio_dev_eqe_get_cqn (flexio_dev_eqe *eqe)

Get CQ number field from EQE function.

Parameters

eqe
- EQE to parse.

Returns

uint32_t - CQ number field value of the EQE.

Description

Parse an EQE for its CQ number field.

FLEXIO_DEV_ALWAYS_INLINE uint8_t flexio_dev_eqe_get_owner (flexio_dev_eqe *eqe)

Get owner field from EQE function.

Parameters

eqe
- EQE to parse.

Returns

uint32_t - owner field value of the EQE.

Description

Parse an EQE for its owner field.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_qp_sq_ring_db (uint16_t pi, uint32_t qnum)

QP/SQ ring doorbell function.

Parameters

pi
- Current queue producer index.

qnum
- Number of the queue to update.

Returns

void.

Description

Rings the doorbell of a QP or SQ in order to alert the HW of pending work.

FLEXIO_DEV_ALWAYS_INLINE void *flexio_dev_rwqe_get_addr (flexio_dev_wqe_rcv_data_seg *rwqe)

Get address field from receive WQE function.

Parameters

rwqe
- WQE to parse.

Returns

void* - Address field value of the receive WQE.

Description

Parse a receive WQE for its address field.

flexio_dev_status_t flexio_dev_swqe_seg_atomic_set (flexio_dev_sqe_seg *swqe, uint64_t swap_or_add_data, uint64_t compare_data)

Fill out an Atomic send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

swap_or_add_data
- The data that will be swapped in or the data that will be added.

compare_data
- The data that will be compared with. Unused in fetch & add operation.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (2 DWORDs) with Atomic segment information. This segment can service a compare & swap or fetch & add operation.

flexio_dev_status_t flexio_dev_swqe_seg_ctrl_set (flexio_dev_sqe_seg *swqe, uint32_t sq_pi, uint32_t sq_number, uint32_t ce, flexio_ctrl_seg_t ctrl_seg_type)

Fill out a control send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

sq_pi
- Producer index of the send WQE.

sq_number
- SQ number that holds the WQE.

ce
- wanted CQ policy for CQEs. Value is taken from cq_ce_mode enum.

ctrl_seg_type
- Type of control segment.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with control segment information. This should always be the 1st segment of the WQE. Note: For RDMA write immediate WQE - user should fill the immediate data information in the control segment.

flexio_dev_status_t flexio_dev_swqe_seg_eth_set (flexio_dev_sqe_seg *swqe, uint16_t cs_swp_flags, uint16_t mss, uint16_t inline_hdr_bsz, uint8_t inline_hdrs)

Fill out an ethernet send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

cs_swp_flags
- Flags for checksum and swap, see PRM section 8.9.4.2, Send WQE Construction Summary.

mss
- Maximum Segment Size - For LSO WQEs - the number of bytes in the TCP payload to be transmitted in each packet. Must be 0 on non LSO WQEs.

inline_hdr_bsz
- Length of inlined packet headers in bytes. This includes the headers in the inline_data segment as well.

inline_hdrs
- First 2 bytes of the inlined packet headers.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with Ethernet segment information.

flexio_dev_status_t flexio_dev_swqe_seg_inline_data_set (flexio_dev_sqe_seg *swqe, uint32_t data_sz, uint32_t *data)

Fill out an inline data send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

data_sz
- Size of the data.

data
- Inline data array (3 DWORDs).

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with inline data segment information.

flexio_dev_status_t flexio_dev_swqe_seg_mem_ptr_data_set (flexio_dev_sqe_seg *swqe, uint32_t data_sz, uint32_t lkey, uint64_t data_addr)

Fill out a memory pointer data send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

data_sz
- Size of the data.

lkey
- Local memory access key for the data operation.

data_addr
- Address of the data for the data operation.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with memory pointer data segment information.

flexio_dev_status_t flexio_dev_swqe_seg_rdma_set (flexio_dev_sqe_seg *swqe, uint32_t rkey, uint64_t raddr)

Fill out an RDMA send queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

rkey
- Remote memory access key for the RDMA operation.

raddr
- Address of the data for the RDMA operation.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with RDMA segment information.

flexio_dev_status_t flexio_dev_swqe_seg_shared_receive_set (flexio_dev_sqe_seg *swqe, uint16_t next_wqe_index, uint8_t signature)

Fill out a Shared receive queue wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

next_wqe_index
- The next wqe index.

signature
- The signature.

Returns

flexio_dev_status_t.

Description

Fill the fields of a linked list shared receive WQE segment.

flexio_dev_status_t flexio_dev_swqe_seg_transpose_set (flexio_dev_sqe_seg *swqe, uint8_t element_size, uint8_t num_of_cols, uint8_t num_of_rows)

Fill out a Transpose send wqe segment function.

Parameters

swqe
- Send WQE segment to fill.

element_size
- The Matrix element_size.

num_of_cols
- Number of columns in the matrix.

num_of_rows
- Number of rows in the matrix.

Returns

flexio_dev_status_t.

Description

Fill the fields of a send WQE segment (4 DWORDs) with Transpose segment information.

FLEXIO_DEV_ALWAYS_INLINE void flexio_dev_zcqe_gen (uint32_t cqn)

Generate a zero CQE on the given CQ.

Parameters

cqn

- CQ number to trigger. Trigger is done via currently configured outbox, this can be changed with outbox config API according to CQ.

Returns

void.

Description

This function trigger the given CQ by creating a zero CQE on it. In turn, this may activate any handler connected to the CQ, most commonly another thread or an EQ (and MSIX).

#define FLEXIO_DEV_COMP_CQE_GET_RX_HASH_RESULT be32_to_cpu((uint32_t)((_x) & 0xFFFFFFFFULL))

Get the RX hash result from a mini CQE.

#define flexio_dev_msix_send flexio_dev_zcqe_gen(cqn)

Generate an MSI-X from the given CQ.

This function trigger an MSI-X interrupt connected to the given CQ.

2.3.3. Flex IO SDK dev queue types

Flex IO SDK dev

Flex IO SDK device queue types for DPA programs. Defines basic networking elements structure.

struct flexio_dev_cqe64

struct flexio_dev_eqe

struct flexio_dev_mini_cqe64

union flexio_dev_sqe_seg

struct flexio_dev_wqe_atomic_seg

struct flexio_dev_wqe_ctrl_seg

struct flexio_dev_wqe_eth_seg

struct flexio_dev_wqe_inline_data_seg

struct flexio_dev_wqe_inline_send_data_seg

struct flexio_dev_wqe_mem_ptr_send_data_seg

struct flexio_dev_wqe_rcv_data_seg

struct flexio_dev_wqe_rdma_seg

struct flexio_dev_wqe_shared_receive_seg

struct flexio_dev_wqe_transpose_seg

enum flexio_dev_wqe_eth_seg_cs_swp_flags_t

Flex IO dev ethernet segment bitmask for CS / SWP flags

Values

FLEXIO_ETH_SEG_L4CS = 0x8000
FLEXIO_ETH_SEG_L3CS = 0x4000
FLEXIO_ETH_SEG_L4CS_INNER = 0x2000
FLEXIO_ETH_SEG_L3CS_INNER = 0x1000
FLEXIO_ETH_SEG_TRAILER_ALIGN = 0x0200

FLEXIO_ETH_SEG_SWP_OUTER_L4_TYPE = 0x0040
FLEXIO_ETH_SEG_SWP_OUTER_L3_TYPE = 0x0020
FLEXIO_ETH_SEG_SWP_INNER_L4_TYPE = 0x0002
FLEXIO_ETH_SEG_SWP_INNER_L3_TYPE = 0x0001

struct flexio_dev_eqe ::packed

Describes Flex IO dev EQE.

Describes Flex IO dev CQE.

Describes Flex IO dev compressed CQE.

Describes Flex IO dev WQE memory pointer send data segment.

Describes Flex IO dev WQE inline send data segment.

Describes Flex IO dev WQE receive data segment.

Describes Flex IO dev shared receive WQE.

Describes Flex IO dev WQE control segment.

Describes Flex IO dev WQE ethernet segment.

Describes Flex IO dev WQE inline data segment.

Describes Flex IO dev WQE RDMA segment.

Describes Flex IO dev WQE ATOMIC segment.

Describes Flex IO dev WQE transpose segment.

#define LOG_SQE_NUM_SEGS 2

SQ depth (log_sq_depth) is measured in WQEBBs, each one is 64B. We have to understand difference between wqe_idx and seg_idx. For example wqe with index 5 built from 4 segments with indexes 20, 21, 22 and 23.

2.4. <stdio.h>: Standard IO facilities

```
#include <stdio.h>
```

Introduction to the Standard IO facilities

This file declares the standard IO facilities that are implemented in `flexio-libc`. Due to the nature of the underlying hardware, only a limited subset of standard IO is implemented. There is no actual file implementation available, so only device IO can be performed. Since there's no operating system, the application needs to provide enough details about their devices in order to make them usable by the standard IO facilities.

Due to space constraints, some functionality has not been implemented at all (like some of the `printf` conversions that have been left out). Nevertheless, potential users of this implementation should be warned: the `printf` and `scanf` families of functions, although usually associated with presumably simple things like the famous "Hello, world!" program, are actually fairly complex which causes their inclusion to eat up a fair amount of code space. Also, they are not fast due to the nature of interpreting the format string at run-time. Whenever possible, resorting to the (sometimes non-standard) predetermined conversion facilities that are offered by flexio-libc will usually cost much less in terms of speed and code size.

Tunable options for code size vs. feature set

In order to allow programmers a code size vs. functionality tradeoff, the function vfprintf() which is the heart of the printf family can be selected in different flavours using linker options. See the documentation of vfprintf() for a detailed description. The same applies to vfscanf() and the `scanf` family of functions.

Outline of the chosen API

The standard streams `stdin`, `stdout`, and `stderr` are provided, but contrary to the C standard, since flexio-libc has no knowledge about applicable devices, these streams are not already pre-initialized at application startup. Also, since there is no notion of "file" whatsoever to flexio-libc, there is no function `fopen()` that could be used to associate a stream to some device. (See note 1.) Instead, the function `fdevopen()` is provided to associate a stream to a device, where the device needs to provide a function to send a character, to receive a character, or both. There is no differentiation between "text" and "binary" streams inside flexio-libc. Character `\n` is sent literally down to the device's `put()` function. If the device requires a carriage return (`\r`) character to be sent before the linefeed, its `put()` routine must implement this (see note 2).

As an alternative method to fdevopen(), the macro fdev_setup_stream() might be used to setup a user-supplied FILE structure.

It should be noted that the automatic conversion of a newline character into a carriage return - newline sequence breaks binary transfers. If binary transfers are desired, no automatic conversion should be performed, but instead any string that aims to issue a CR-LF sequence must use `"\r\n"` explicitly.

stdin, stdout and stderr are undefined global FILE pointers. If you want to use this, your application must define these variables and initialize them. They are declared 'const' so that you can place them in ROM if you don't need to modify it after startup. FILEs cannot be placed in ROM as they have values which are modified during runtime.

Running stdio without malloc()

By default, fdevopen() requires malloc(). As this is often not desired in the limited environment of a microcontroller, an alternative option is provided to run completely without malloc().

The macro fdev_setup_stream() is provided to prepare a user-supplied FILE buffer for operation with stdio.

Example

```
#include <stdio.h>

      static int uart_putchar(char c, FILE *stream);

      static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
                                               _FDEV_SETUP_WRITE);

      static int
      uart_putchar(char c, FILE *stream)
      {

        if (c == '\n')
          uart_putchar('\r', stream);
        loop_until_bit_is_set(UCSRA, UDRE);
        UDR = c;
        return 0;
      }

      int
      main(void)
      {
        init_uart();
        stdout = &mystdout;
        printf("Hello, world!\n");

        return 0;
      }
```

This example uses the initializer form FDEV_SETUP_STREAM() rather than the function-like fdev_setup_stream(), so all data initialization happens during C start-up.

If streams initialized that way are no longer needed, they can be destroyed by first calling the macro fdev_close(), and then destroying the object itself. No call to fclose() should be issued for these streams. While calling fclose() itself is harmless, it will cause an undefined reference to free() and thus cause the linker to link the malloc module into the application.

Notes

Note 1:

It might have been possible to implement a device abstraction that is compatible with `fopen()` but since this would have required to parse a string, and to take all the information needed either out of this string, or out of an additional table that would need to be provided by the application, this approach was not taken.

Note 2:

This basically follows the Unix approach: if a device such as a terminal needs special handling, it is in the domain of the terminal device driver to provide this functionality. Thus, a simple

function suitable as `put()` for `fdevopen()` that talks to a UART interface might look like this:

```
    int
         uart_putchar(char c, FILE *stream)
         {

           if (c == '\n')
             uart_putchar('\r', stream);
           loop_until_bit_is_set(UCSRA, UDRE);
           UDR = c;
           return 0;
         }
```

Note 3:

This implementation has been chosen because the cost of maintaining an alias is considerably smaller than the cost of maintaining full copies of each stream. Yet, providing an implementation that offers the complete set of standard streams was deemed to be useful. Not only that writing `printf()` instead of `fprintf(mystream, ...)` saves typing work, but since the compiler needs to resort to pass all arguments of variadic functions on the stack (as opposed to passing them in registers for functions that take a fixed number of parameters), the ability to pass one parameter less by implying `stdin` or stdout will also save some execution time.

FILE *const ::stderr

Stream destined for error output. Unless specifically assigned, identical to `stdout`.

FILE *const ::stdin

Stream that will be used as an input stream by the simplified functions that don't take a `stream` argument.

FILE *::stdout

Stream that will be used as an output stream by the simplified functions that don't take a `stream` argument.

int int int int int int int asprintf (char **strp, const char *fmt, ...)

Description

Variant of `printf()` that sends the formatted characters to allocated string `*strp`.

clearerr (FILE *__stream)

Description

Clear the error and end-of-file flags of `stream`.

feof (FILE *__stream)

Description

Test the end-of-file flag of `stream`. This flag can only be cleared by a call to clearerr().

ferror (FILE *__stream)

Description

Test the error flag of `stream`. This flag can only be cleared by a call to clearerr().

fflush (FILE *stream)

Description

Flush `stream`.

If the stream provides a flush hook, use that. Otherwise return 0.

int int int int int int int int int fprintf (FILE *__stream, const char *__fmt, ...)

Description

The function `fprintf` performs formatted output to `stream`. See `vfprintf()` for details.

int int int int int int int int int fputs (const char *__str, FILE *__stream)

Description

Write the string pointed to by `str` to stream `stream`.

Returns 0 on success and EOF on error.

printf (const char *__fmt, ...)

Description

The function `printf` performs formatted output to stream `stdout`. See `vfprintf()` for details.

puts (const char *__str)

Description

Write the string pointed to by `str`, and a trailing newline character, to `stdout`.

int int int int snprintf (char *__s, size_t __n, const char *__fmt, ...)

Description

Like `sprintf()` , but instead of assuming `s` to be of infinite size, no more than `n` characters (including the trailing NUL character) will be converted to `s`.

Returns the number of characters that would have been written to `s` if there were enough space.

int int int sprintf (char *__s, const char *__fmt, ...)

Description

Variant of `printf()` that sends the formatted characters to string `s`.

int int int int int int int int vasprintf (char **strp, const char *fmt, va_list ap)

Description

Variant of `vprintf()` that sends the formatted characters to allocated string `*strp`.

int int vprintf (const char *__fmt, va_list __ap)

Description

The function `vprintf` performs formatted output to stream `stdout`, taking a variable argument list as in vfprintf().

See vfprintf() for details.

int int int int int int vsnprintf (char *__s, size_t __n, const char *__fmt, va_list ap)

Description

Like `vsprintf()` , but instead of assuming `s` to be of infinite size, no more than `n` characters (including the trailing NUL character) will be converted to `s`.

Returns the number of characters that would have been written to `s` if there were enough space.

int int int int int vsprintf (char *__s, const char *__fmt, va_list ap)

Description

Like `sprintf()` but takes a variable argument list for the arguments.

#define __FORMAT_ATTRIBUTE__

`vfprintf` is the central facility of the `printf` family of functions. It outputs values to `stream` under control of a format string passed in `fmt`. The actual values to print are passed as a variable argument list `ap`.

`vfprintf` returns the number of characters written to `stream`, or `EOF` in case of an error. Currently, this will only happen if `stream` has not been opened with write intent.

The format string is composed of zero or more directives: ordinary characters (not `%`), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the `%` character. The arguments must properly correspond (after type promotion) with the conversion specifier. After the `%`, the following appear in sequence:

- Zero or more of the following flags:

  - # The value should be converted to an "alternate form". For c, d, i, s, and u conversions, this option has no effect. For o conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For x and X conversions, a non-zero result has the string `0x' (or `0X' for X conversions) prepended to it.
  - 0 (zero) Zero padding. For all conversions, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (d, i, o, u, i, x, and X), the 0 flag is ignored.
  - – A negative field width flag; the converted value is to be left adjusted on the field boundary. The converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A - overrides a 0 if both are given.
  - ' ' (space) A blank should be left before a positive number produced by a signed conversion (d, or i).
  - + A sign must always be placed before a number produced by a signed conversion. A + overrides a space if both are used.
- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given) to fill out the field width.
- An optional precision, in the form of a period . followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for d, i, o, u, x, and X conversions, or the maximum number of characters to be printed from a string for s conversions.
- An optional `l` or `h` length modifier, that specifies that the argument for the d, i, o, u, x, or X conversion is a `"long int"` rather than `int`. The `h` is ignored, as `"short int"` is equivalent to `int`.
- A character that specifies the type of conversion to be applied.

The conversion specifiers and their meanings are:

- `diouxX` The int (or appropriate variant) argument is converted to signed decimal (d and i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal (x and X) notation. The letters "abcdef" are used for x conversions; the letters "ABCDEF" are used for X conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.
- `p` The `void *` argument is taken as an unsigned integer, and converted similarly as a `%#x` command would do.
- `c` The `int` argument is converted to an `"unsigned char"`, and the resulting character is written.
- `s` The `"char *"` argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating NUL character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not

specified, or is greater than the size of the array, the array must contain a terminating NUL character.

- `%` A `%` is written. No argument is converted. The complete conversion specification is "%%".
- `eE` The double argument is rounded and converted in the format `"[-]d.ddde±dd"` where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An E conversion uses the letter `'E'` (rather than `'e'`) to introduce the exponent. The exponent always contains two digits; if the value is zero, the exponent is 00.
- `fF` The double argument is rounded and converted to decimal notation in the format `"[-]ddd.ddd"`, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.
- `gG` The double argument is converted in style `f` or `e` (or `F` or `E` for `G` conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style `e` is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.
- `S` Similar to the `s` format, except the pointer is expected to point to a program-memory (ROM) string instead of a RAM string.

In no case does a non-existent or small field width cause truncation of a numeric field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Since the full implementation of all the mentioned features becomes fairly large, three different flavours of vfprintf() can be selected using linker options. The default vfprintf() implements all the mentioned functionality except floating point conversions. A minimized version of vfprintf() is available that only implements the very basic integer and string conversion facilities, but only the # additional option can be specified using conversion flags (these flags are parsed correctly from the format specification, but then simply ignored). This version can be requested using the following compiler options:

```
-Wl,-u,vfprintf -lprintf_min
```

Limitations:

- The specified width and precision can be at most 255.

Notes:

- Floating point is not supported in flexio-libc
- The `hh` length modifier is ignored (`char` argument is promouted to `int`). More exactly, this realization does not check the number of `h` symbols.

- But the `ll` length modifier will to abort the output, as this realization does not operate `long long` arguments.
- The variable width or precision field (an asterisk `*` symbol) is not realized and will to abort the output.

#define _FDEV_EOF (-2)

Return code for an end-of-file condition during device read.

To be used in the get function of fdevopen().

#define _FDEV_ERR (-1)

Return code for an error condition during device read.

To be used in the get function of fdevopen().

#define _FDEV_SETUP_READ __SRD

fdev_setup_stream() with read intent

#define _FDEV_SETUP_RW (__SRD|__SWR)

fdev_setup_stream() with read/write intent

#define _FDEV_SETUP_WRITE __SWR

fdev_setup_stream() with write intent

#define EOF (-1)

`EOF` declares the value that is returned by various standard IO functions in case of an error. Since the AVR platform (currently) doesn't contain an abstraction for actual files, its origin as "end of file" is somewhat meaningless here.

#define PICOLIBC_STDIO_GLOBALS

This symbol is defined when stdin/stdout/stderr are global variables. When undefined, the old __iob array is used which contains the pointers instead

#define putc fputc(__c, __stream)

The macro `putc` used to be a "fast" macro implementation with a functionality identical to fputc(). For space constraints, in `flexio-libc`, it is just an alias for `fputc`.

#define putchar fputc(__c, stdout)

The macro `putchar` sends character `c` to `stdout`.

Chapter 3. Data Structures

Here are the data structures with brief descriptions:

3.1. __file Struct Reference

`FILE` is the opaque structure that is passed around between the various standard IO functions.

3.2. dpa_stats_capabilities Struct Reference

Describes fields of the single process structure.

uint8_t dpa_stats_capabilities::dpa_supported

1 if DPA caps are supported

uint32_t dpa_stats_capabilities::performance_sample_type

bits capability of which types of performance counters are supported

uint8_t dpa_stats_capabilities::process_performance_counters

1 if performance counters are supported

3.3. dpa_stats_perf_cumul_list Struct Reference

Describes fields of the cumulative info list.

dpa_stats_thread_cumul_info *dpa_stats_perf_cumul_list::samples

Array of thread cumulative info samples with the size of samples_num

uint32_t dpa_stats_perf_cumul_list::samples_num

Size of the samples array

3.4. dpa_stats_perf_event_list Struct Reference

Describes fields of the performance events tracer list.

dpa_stats_perf_event_sample *dpa_stats_perf_event_list::samples

Array of performance event samples with the size of samples_num

uint32_t dpa_stats_perf_event_list::samples_num

Size of the samples array

3.5. dpa_stats_perf_event_sample Struct Reference

Describes fields of the single performance events tracer structure.

uint64_t dpa_stats_perf_event_sample::cycles

Stamp of total Execution Unit cycles

uint32_t dpa_stats_perf_event_sample::dpa_process_id

Global DPA process Id

uint32_t dpa_stats_perf_event_sample::dpa_thread_id

Global DPA thread Id

uint16_t dpa_stats_perf_event_sample::eu_id

EU id

uint32_t dpa_stats_perf_event_sample::instructions

Stamp of total number of instructions of this DPA EU

uint16_t dpa_stats_perf_event_sample::sample_id_in_eu

Running sample id per Execution Unit. A single sample_id is assigned to both schedule in and out samples

uint64_t dpa_stats_perf_event_sample::time

Timestamp in usec

enumdpa_stats_event_sample_type dpa_stats_perf_event_sample::type

Type of event sample

3.6. dpa_stats_process_info Struct Reference

Describes fields of the single process structure.

uint32_t dpa_stats_process_info::dpa_process_id

Global DPA process Id

uint32_t dpa_stats_process_info::num_of_threads

Number of threads in process

char dpa_stats_process_info::process_name

The name of the process

3.7. dpa_stats_process_list Struct Reference

Describes fields of the process list.

uint32_t dpa_stats_process_list::process_num

Size of the process info's array

dpa_stats_process_info *dpa_stats_process_list::processes

Array of process info with the size of process_num

3.8. dpa_stats_thread_cumul_info Struct Reference

Describes fields of the single cumulative info structure.

uint64_t dpa_stats_thread_cumul_info::cycles

Total EU cycles the thread was used

uint32_t dpa_stats_thread_cumul_info::dpa_process_id

Global DPA process Id

uint32_t dpa_stats_thread_cumul_info::dpa_thread_id

Global DPA thread Id

uint64_t dpa_stats_thread_cumul_info::instructions

Total number of instructions the thread executed

uint64_t dpa_stats_thread_cumul_info::num_executions

Total number of thread invocations

uint64_t dpa_stats_thread_cumul_info::time

Total time in ticks the thread was active

3.9. dpa_stats_thread_info Struct Reference

Describes fields of the single thread structure.

uint32_t dpa_stats_thread_info::dpa_process_id

Global DPA process Id

uint32_t dpa_stats_thread_info::dpa_thread_id

Global DPA thread Id

char dpa_stats_thread_info::thread_name

The name of the thread

3.10. dpa_stats_thread_list Struct Reference

Describes fields of the thread list.

dpa_stats_thread_info *dpa_stats_thread_list::threads

Array of thread info with the size of threads_num

uint32_t dpa_stats_thread_list::threads_num

Size of the thread info's array

3.11. flexio_affinity Struct Reference

Describes Flex IO thread affinity information.

uint32_t flexio_affinity::id

ID of the chosen resource (EU / DPA EU group). Reserved if affinity type none is set.

enumflexio_affinity_type flexio_affinity::type

Affinity type to use for a Flex IO thread (none, strict or group).

3.12. flexio_app_attr Struct Reference

Describes process attributes for creating a Flex IO application.

size_t flexio_app_attr::app_bsize

DPA application size (bytes).

const char *flexio_app_attr::app_name

DPA application name.

void *flexio_app_attr::app_ptr

Pointer to a buffer holds the DPA application.

uint64_t flexio_app_attr::dpa_api_version

RTOS version.

uint64_t *flexio_app_attr::flexio_dev_versions

Array of flexio_dev_versions.

size_t flexio_app_attr::flexio_dev_versions_len_size

Length of array of flexio_dev_versions.

size_t flexio_app_attr::sig_bsize

DPA application signature buffer size (bytes). sig_bsize == 0 indicates no signature.

void *flexio_app_attr::sig_ptr

Pointer to a buffer holds the signature of the application.

3.13. flexio_cmdq_attr Struct Reference

Describes process attributes for creating a Flex IO command queue (async RPC).

int flexio_cmdq_attr::batch_size

Number of tasks to be executed to completion by invoked thread.

enumflexio_cmdq_state flexio_cmdq_attr::state

Command queue initial state.

int flexio_cmdq_attr::workers

Number of available workers, each worker can handle up to batch_size number of tasks in a single invocation.

3.14. flexio_cq_attr Struct Reference

Describes attributes for creating a Flex IO CQ.

uint8_t flexio_cq_attr::always_armed

Indication to always arm for the created CQ

bool flexio_cq_attr::cc

Indication to enable collapsed CQE for the created CQ.

flexio_uintptr_t flexio_cq_attr::cq_dbr_daddr

DBR memory address for the created CQ.

uint16_t flexio_cq_attr::cq_max_count

CQE moderation max count (number of CQEs before creating an event).

uint16_t flexio_cq_attr::cq_period

CQE moderation period (number of usecs before creating an event).

enumflexio_cq_period_mode flexio_cq_attr::cq_period_mode

CQE moderation period mode (by CQE or by event).

struct flexio_qmem flexio_cq_attr::cq_ring_qmem

Ring memory info for the created CQ.

enum flexio_cqe_comp_format flexio_cq_attr::cqe_comp_format

CQE compression mini CQE format.

enumflexio_cqe_comp_type flexio_cq_attr::cqe_comp_type

CQE compression type to use for the CQ.

uint8_t flexio_cq_attr::element_type

Type of the element attached to the created CQ (thread, EQ, none, emulated EQ).

uint32_t flexio_cq_attr::emulated_eqn

Emulated EQ number to attach to the created CQ

uint8_t flexio_cq_attr::log_cq_depth

Log number of entries for the created CQ.

bool flexio_cq_attr::no_arm

Indication to not arm the CQ on creation.

uint8_t flexio_cq_attr::overrun_ignore

Indication to ignore overrun for the created CQ.

flexio_thread *flexio_cq_attr::thread

Thread object to attach to the created CQ (only valid for element type thread).

void *flexio_cq_attr::uar_base_addr

CQ UAR base address, relevant for devx UAR only, otherwise must be NULL.

uint32_t flexio_cq_attr::uar_id

CQ UAR ID (devx UAR ID for host queues, otherwise flexio_uar).

3.15. flexio_dev_cqe64 Struct Reference

Describes Flex IO dev CQE.

__be32 flexio_dev_cqe64::byte_cnt

0Bh - Byte count.

__be16 flexio_dev_cqe64::csum_ok

05h 16..31 - checksum ok bits.

__be32 flexio_dev_cqe64::err_syndrome

0Dh Error syndrome

__be32 flexio_dev_cqe64::imm_inval_pkey

09h - immediate / invalidate key / pkey

uint8_t flexio_dev_cqe64::op_own

0Fh 0 - Ownership bit.

__be32 flexio_dev_cqe64::qpn

0Eh - QPN.

uint8_t flexio_dev_cqe64::qpn24

0Eh 0..23 - qpn with 24 bits

__be32 flexio_dev_cqe64::rsvd

0Ah - Reserved.

__be32 flexio_dev_cqe64::rsvd00

00h..04h - Reserved.

__be32 flexio_dev_cqe64::rsvd14

06h..07h - Reserved.

uint8_t flexio_dev_cqe64::rsvd22

05h 0..15 - Reserved.

__be32 flexio_dev_cqe64::rsvd48

0Ch.Reserved.

uint8_t flexio_dev_cqe64::signature

0Fh 8..15 - Signature/validity.

uint8_t flexio_dev_cqe64::sop_rdrop

0Eh 24..31 - send_wqe_opcode/rx_drop_counter

__be32 flexio_dev_cqe64::srqn_uidx

08h - SRQ number or user index.

__be16 flexio_dev_cqe64::wqe_counter

0Fh 16..31 - WQE counter.

3.16. flexio_dev_eqe Struct Reference

Describes Flex IO dev EQE.

__be32 flexio_dev_eqe::cqn

18h 24 lsb - CQN.

flexio_dev_eqe::@10 flexio_dev_eqe::event_data

20h - Event data.

uint8_t flexio_dev_eqe::owner

3Fh - Owner.

__be32 flexio_dev_eqe::rsvd00

00h..17h - Reserved.

uint8_t flexio_dev_eqe::rsvd00

00h - Reserved.

uint8_t flexio_dev_eqe::rsvd02

02h - Reserved.

__be16 flexio_dev_eqe::rsvd3c

3Ch - Reserved.

uint8_t flexio_dev_eqe::rsvd4

04h..1fh - Reserved.

uint8_t flexio_dev_eqe::signature

3Eh - Signature.

uint8_t flexio_dev_eqe::sub_type

03h - Sub type.

uint8_t flexio_dev_eqe::type

01h - EQE type.

3.17. flexio_dev_mini_cqe64 Struct Reference

Describes Flex IO dev compressed CQE.

__be64 flexio_dev_mini_cqe64::mini_cqe

00h..37h mini cqe array.

uint8_t flexio_dev_mini_cqe64::num_and_type

3fh..3fh no' of mini cqes, mini cqe format.

uint8_t flexio_dev_mini_cqe64::rsvd0

38h..3dh mini cqe 7.

uint8_t flexio_dev_mini_cqe64::validity_iteration_count

3eh..3eh validity iteration count.

3.18. flexio_dev_sqe_seg Union Reference

Describes Flex IO dev send WQE segments. Only one segment can be set at a given time.

struct flexio_dev_wqe_atomic_seg flexio_dev_sqe_seg::atomic

Atomic segment.

struct flexio_dev_wqe_ctrl_seg flexio_dev_sqe_seg::ctrl

Control segment.

struct flexio_dev_wqe_eth_seg flexio_dev_sqe_seg::eth

Ethernet segment.

struct flexio_dev_wqe_inline_data_seg flexio_dev_sqe_seg::inline_data

Inline data segment.

struct flexio_dev_wqe_inline_send_data_seg flexio_dev_sqe_seg::inline_send_data

Inline send data segment.

struct flexio_dev_wqe_mem_ptr_send_data_seg flexio_dev_sqe_seg::mem_ptr_send_data

Memory pointer send data segment.

struct flexio_dev_wqe_rdma_seg flexio_dev_sqe_seg::rdma

RDMA segment.

struct flexio_dev_wqe_shared_receive_seg flexio_dev_sqe_seg::shared_receive

Shared receive.

struct flexio_dev_wqe_transpose_seg flexio_dev_sqe_seg::transpose

Transpose segment.

3.19. flexio_dev_wqe_atomic_seg Struct Reference

Describes Flex IO dev WQE ATOMIC segment.

__be64 flexio_dev_wqe_atomic_seg::compare_data

02h..03h - Compare operation data.

__be64 flexio_dev_wqe_atomic_seg::swap_or_add_data

00h..01h - Swap or Add operation data.

3.20. flexio_dev_wqe_ctrl_seg Struct Reference

Describes Flex IO dev WQE control segment.

__be32 flexio_dev_wqe_ctrl_seg::general_id

03h - Control general ID.

__be32 flexio_dev_wqe_ctrl_seg::idx_opcode

00h - WQE index and opcode.

__be32 flexio_dev_wqe_ctrl_seg::qpn_ds

01h - QPN and number of data segments.

__be32 flexio_dev_wqe_ctrl_seg::signature_fm_ce_se

02h - Signature, fence mode, completion mode and solicited event.

3.21. flexio_dev_wqe_eth_seg Struct Reference

Describes Flex IO dev WQE ethernet segment.

__be16 flexio_dev_wqe_eth_seg::cs_swp_flags

01h 16..31 - CS and SWP flags.

__be16 flexio_dev_wqe_eth_seg::inline_hdr_bsz

03h 16..31 - Inline headers size (bytes).

uint8_t flexio_dev_wqe_eth_seg::inline_hdrs

03h 0..15 - Inline headers (first two bytes).

__be16 flexio_dev_wqe_eth_seg::mss

01h 0..15 - Max segment size.

__be32 flexio_dev_wqe_eth_seg::rsvd0

00h - Reserved.

__be32 flexio_dev_wqe_eth_seg::rsvd2

02h - Reserved.

3.22. flexio_dev_wqe_inline_data_seg Struct Reference

Describes Flex IO dev WQE inline data segment.

uint8_t flexio_dev_wqe_inline_data_seg::inline_data

00h..03h - Inline data.

3.23. flexio_dev_wqe_inline_send_data_seg Struct Reference

Describes Flex IO dev WQE inline send data segment.

__be32 flexio_dev_wqe_inline_send_data_seg::byte_count

00h - Byte count.

__be32 flexio_dev_wqe_inline_send_data_seg::data_and_padding

01h..03h - Data and padding array.

3.24. flexio_dev_wqe_mem_ptr_send_data_seg Struct Reference

Describes Flex IO dev WQE memory pointer send data segment.

__be64 flexio_dev_wqe_mem_ptr_send_data_seg::addr

02h..03h - Address.

__be32 flexio_dev_wqe_mem_ptr_send_data_seg::byte_count

00h - Byte count.

__be32 flexio_dev_wqe_mem_ptr_send_data_seg::lkey

01h - Local key.

3.25. flexio_dev_wqe_rcv_data_seg Struct Reference

Describes Flex IO dev WQE receive data segment.

__be64 flexio_dev_wqe_rcv_data_seg::addr

02h..03h - Address.

__be32 flexio_dev_wqe_rcv_data_seg::byte_count

00h - Byte count.

__be32 flexio_dev_wqe_rcv_data_seg::lkey

01h - Local key.

3.26. flexio_dev_wqe_rdma_seg Struct Reference

Describes Flex IO dev WQE RDMA segment.

__be64 flexio_dev_wqe_rdma_seg::raddr

00h..01h - Remote address.

__be32 flexio_dev_wqe_rdma_seg::rkey

02h - Remote key.

__be32 flexio_dev_wqe_rdma_seg::rsvd0

03h - Reserved.

3.27. flexio_dev_wqe_shared_receive_seg Struct Reference

Describes Flex IO dev shared receive WQE.

__be16 flexio_dev_wqe_shared_receive_seg::next_wqe_index

Index (pointer) in the WQE buffer to the next WQE to be executed.

uint8_t flexio_dev_wqe_shared_receive_seg::rsvd0

Reserved bits for memory alignment.

uint8_t flexio_dev_wqe_shared_receive_seg::rsvd1

Reserved bits for memory alignment.

uint8_t flexio_dev_wqe_shared_receive_seg::signature

WQE signature.

3.28. flexio_dev_wqe_transpose_seg Struct Reference

Describes Flex IO dev WQE transpose segment.

uint8_t flexio_dev_wqe_transpose_seg::element_size

00h 0..7 - Matrix element size.

uint8_t flexio_dev_wqe_transpose_seg::num_of_cols

01h 16..22 - Number of columns in matrix (7b).

uint8_t flexio_dev_wqe_transpose_seg::num_of_rows

01h 0..6 - Number of rows in matrix (7b).

uint8_t flexio_dev_wqe_transpose_seg::rsvd0

00h 8..31 - Reserved.

uint8_t flexio_dev_wqe_transpose_seg::rsvd1

01h - Reserved.

uint8_t flexio_dev_wqe_transpose_seg::rsvd2

01h - Reserved.

uint8_t flexio_dev_wqe_transpose_seg::rsvd4

02h..03h - Reserved.

3.29. flexio_event_handler_attr Struct Reference

Describes attributes for creating a Flex IO event handler.

struct flexio_affinity flexio_event_handler_attr::affinity

Thread's affinity information.

uint64_t flexio_event_handler_attr::arg

Thread argument.

int flexio_event_handler_attr::continuable

Thread continuable flag.

flexio_func_t *flexio_event_handler_attr::host_stub_func

Stub for the entry function of the thread.

const char *flexio_event_handler_attr::name

Name of event handler.

flexio_uintptr_t flexio_event_handler_attr::thread_local_storage_daddr

Address of the local storage buffer of the thread.

3.30. flexio_heap_mem_info Struct Reference

Describes process heap memory information

size_t flexio_heap_mem_info::allocated

Process heap memory allocated in bytes.

uint64_t flexio_heap_mem_info::base_addr

Process heap memory base address.

size_t flexio_heap_mem_info::requested

Process heap memory requested in bytes.

size_t flexio_heap_mem_info::size

Process heap memory size in bytes.

3.31. flexio_mkey_attr Struct Reference

Describes process attributes for creating a Flex IO MKey.

int flexio_mkey_attr::access

access contains the access mask for the MKey (Expected values: IBV_ACCESS_REMOTE_WRITE, IBV_ACCESS_LOCAL_WRITE).

flexio_uintptr_t flexio_mkey_attr::daddr

DPA address the MKey is created for.

size_t flexio_mkey_attr::len

Length of the address space the MKey is created for.

ibv_pd *flexio_mkey_attr::pd

IBV protection domain information for the created MKey.

3.32. flexio_msg_stream_attr_t Struct Reference

Describes DPA msg thread attributes for messaging from the Device to the Host side.

size_t flexio_msg_stream_attr_t::data_bsize

Size of buffer, used for data transfer from Flex IO to HOST MUST be power of two and be at least 2Kb.

flexio_msg_dev_level flexio_msg_stream_attr_t::level

Log level of the stream with the ranges between FLEXIO_MSG_DEV_NO_PRINT and FLEXIO_MSG_DEV_DEBUG. FLEXIO_MSG_DEV_ALWAYS_PRINT cannot be used.

struct flexio_affinity flexio_msg_stream_attr_t::mgmt_affinity

EU affinity for stream management operations creation, modification and destruction Passing a nullified struct will set affinity type to 'NONE'.

char *flexio_msg_stream_attr_t::stream_name

The name of the stream.

flexio_msg_dev_sync_mode flexio_msg_stream_attr_t::sync_mode

Select sync mode scheme.

enumflexio_tracer_transport flexio_msg_stream_attr_t::tracer_mode

Tracer transport mode.

**flexio_msg_stream_attr_t::tracer_msg_formats

Tracer print format templates array, last entry must be NULL. Device message format ID is used as index to this array.

flexio_uar *flexio_msg_stream_attr_t::uar

Deprecated field. Value will be ignored. flexio_process UAR be used instead.

3.33. flexio_outbox_attr Struct Reference

Describes attributes for creating a Flex IO outbox.

uint32_t flexio_outbox_attr::en_pcc

Create outbox with support for CC operations.

flexio_uar *flexio_outbox_attr::uar

Deprecated field. Value will be ignored. flexio_process UAR will be used instead.

3.34. flexio_process_attr Struct Reference

Describes attributes for creating a Flex IO process.

int flexio_process_attr::en_pcc

Enable PCC configuration for the created process.

const char *flexio_process_attr::name

Name of the process.

ibv_pd *flexio_process_attr::pd

IBV protection domain information for the created process. Passing NULL will result in an internal PD being created and used for the process.

3.35. flexio_qmem Struct Reference

Describes queue memory, which may be either host memory or DPA memory

flexio_uintptr_t flexio_qmem::daddr

DPA address of the queue memory (only valid for memtype FLEXIO_MEMTYPE_DPA).

uint64_t flexio_qmem::humem_offset

Address offset in the umem of the queue memory (only valid for memtype FLEXIO_MEMTYPE_HOST).

enumflexio_memtype flexio_qmem::memtype

Type of memory to use (FLEXIO_MEMTYPE_DPA or FLEXIO_MEMTYPE_HOST).

uint32_t flexio_qmem::umem_id

UMEM ID of the queue memory.

3.36. flexio_qp_attr Struct Reference

Describes attributes for creating a Flex IO QP.

uint8_t *flexio_qp_attr::dest_mac

Destination MAC address to set for the modified QP

uint8_t flexio_qp_attr::fl

Indication to enable force loopback for the modified QP.

uint8_t flexio_qp_attr::gid_table_index

GID table index to set for the modified QP

uint8_t flexio_qp_attr::grh

GRH to set for the modified QP.

uint8_t flexio_qp_attr::isolate_vl_tc

When set, the QP will transmit on an isolated VL/TC if available.

int flexio_qp_attr::log_rq_depth

Log number of entries of the QP's RQ.

uint8_t flexio_qp_attr::log_rra_max

Log of the number of allowed outstanding RDMA read/atomic operations

int flexio_qp_attr::log_sq_depth

Log number of entries of the QP's SQ.

uint8_t flexio_qp_attr::log_sra_max

Log of the number of allowed outstanding RDMA read/atomic operations as requester

uint32_t flexio_qp_attr::min_rnr_nak_timer

Minimal RNR NACK timer to set for the modified QP.

uint32_t flexio_qp_attr::next_rcv_psn

Next receive PSN to set for the modified QP.

uint32_t flexio_qp_attr::next_send_psn

Next send PSN to set for the modified QP.

enumflexio_qp_state flexio_qp_attr::next_state

QP state to move the QP to (reset, init, RTS, RTR).

int flexio_qp_attr::no_sq

Indication to create the QP without an SQ.

int flexio_qp_attr::ops_flag

deprecated.

enumflexio_qp_qpc_mtu flexio_qp_attr::path_mtu

Path MTU to set for the modified QP.

ibv_pd *flexio_qp_attr::pd

IBV protection domain information for the created QP.

int flexio_qp_attr::qp_access_mask

QP's access permission (Expected values: IBV_ACCESS_REMOTE_WRITE, IBV_ACCESS_REMOTE_READ, IBV_ACCESS_REMOTE_ATOMIC, IBV_ACCESS_LOCAL_WRITE).

struct flexio_qmem flexio_qp_attr::qp_wq_buff_qmem

Ring memory info for the created QP's WQ.

struct flexio_qmem flexio_qp_attr::qp_wq_dbr_qmem

DBR memory info for the created QP's WQ.

uint32_t flexio_qp_attr::remote_qp_num

Remote QP number to set for the modified QP.

uint8_t flexio_qp_attr::retry_count

Retry count to set for the modified QP.

ibv_gid flexio_qp_attr::rgid_or_rip

Remote GID or remote IP to set for the modified QP.

uint16_t flexio_qp_attr::rlid

Remote LID to set for the modified QP.

uint32_t flexio_qp_attr::rmpqn

RMP queue number, relevant only if QP RQ is RMP.

uint32_t flexio_qp_attr::rq_cqn

CQ number of the QP's RQ. Not relevant for RMP

int flexio_qp_attr::rq_type

QP's RQ type (regular, RMP, zero-RQ)

uint32_t flexio_qp_attr::sq_cqn

CQ number of the QP's SQ.

uint32_t flexio_qp_attr::transport_type

QP's transport type (currently only FLEXIO_QPC_ST_RC is supported).

uint32_t flexio_qp_attr::uar_id

QP UAR ID.

uint16_t flexio_qp_attr::udp_sport

UDP port to set for the modified QP.

uint32_t flexio_qp_attr::user_index

User defined user_index for the created QP.

uint8_t flexio_qp_attr::vhca_port_num

VHCA port number to set for the modified QP.

3.37. flexio_qp_attr_opt_param_mask Struct Reference

Describes QP modify operation mask.

bool flexio_qp_attr_opt_param_mask::min_rnr_nak_timer

Indication to modify the QP's min_rnr_nak_timer field.

bool flexio_qp_attr_opt_param_mask::qp_access_mask

Indication to modify the QP's qp_access_mask field.

3.38. flexio_wq_attr Struct Reference

Describes attributes for creating a Flex IO WQ.

uint8_t flexio_wq_attr::log_wq_depth

Log number of entries for the created WQ.

uint8_t flexio_wq_attr::log_wq_stride

Log size of entry for the created WQ. If this parameter is not provided, it will be set to default value 4.

ibv_pd *flexio_wq_attr::pd

IBV protection domain struct to use for creating the WQ.

struct flexio_wq_rq_attr flexio_wq_attr::rq

RQ attributes (used only for RQs).

struct flexio_wq_sq_attr flexio_wq_attr::sq

SQ attributes (used only for SQs).

uint32_t flexio_wq_attr::uar_id

WQ UAR ID.

uint32_t flexio_wq_attr::user_index

User defined user_index for the created WQ.

struct flexio_qmem flexio_wq_attr::wq_dbr_qmem

DBR memory address for the created WQ.

struct flexio_qmem flexio_wq_attr::wq_ring_qmem

Ring memory info for the created WQ.

3.39. flexio_wq_rq_attr Struct Reference

Describes attributes for creating a Flex IO RQ.

enumflexio_wq_end_pad_mode flexio_wq_rq_attr::end_pad_mode

RQ's WQ end padding mode.

flexio_transport_domain *flexio_wq_rq_attr::td

TD (transport domain) object created for the RQ. If null, a TD object will be created on RQ creation. TD is used for TIR creation.

uint8_t flexio_wq_rq_attr::vlan_strip_disable

When set, the RQ vlan strip is disables.

enumflexio_wq_type flexio_wq_rq_attr::wq_type

RQ's WQ type.

3.40. flexio_wq_sq_attr Struct Reference

Describes attributes for creating a Flex IO SQ.

uint8_t flexio_wq_sq_attr::allow_multi_pkt_send_wqe

Indication enable multi packet send WQE for the created SQ.

flexio_sq_tis *flexio_wq_sq_attr::tis

TIS (transport interface send) object created for the SQ. If null, a TIS object will be created on SQ creation.

3.41. spinlock_s Struct Reference

Describes Flex IO dev spinlock.

uint32_t spinlock_s::locked

Indication for spinlock lock state.

Chapter 4. Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

A

access
flexio_mkey_attr
addr
flexio_dev_wqe_mem_ptr_send_data_seg
flexio_dev_wqe_rcv_data_seg
affinity
flexio_event_handler_attr
allocated
flexio_heap_mem_info
allow_multi_pkt_send_wqe
flexio_wq_sq_attr
always_armed
flexio_cq_attr
app_bsize
flexio_app_attr
app_name
flexio_app_attr
app_ptr
flexio_app_attr
arg
flexio_event_handler_attr
atomic
flexio_dev_sqe_seg

B

base_addr
flexio_heap_mem_info
batch_size
flexio_cmdq_attr

C

D

E

F

fl
- flexio_qp_attr

flexio_dev_versions
- flexio_app_attr

flexio_dev_versions_len_size
- flexio_app_attr

G

general_id
- flexio_dev_wqe_ctrl_seg

gid_table_index
- flexio_qp_attr

grh
- flexio_qp_attr

H

host_stub_func
- flexio_event_handler_attr

humem_offset
- flexio_qmem

I

id
- flexio_affinity

idx_opcode
- flexio_dev_wqe_ctrl_seg

imm_inval_pkey
- flexio_dev_cqe64

inline_data
- flexio_dev_wqe_inline_data_seg
- flexio_dev_sqe_seg

inline_hdr_bsz
- flexio_dev_wqe_eth_seg

inline_hdrs
- flexio_dev_wqe_eth_seg

inline_send_data
- flexio_dev_sqe_seg

instructions
- dpa_stats_thread_cumul_info
- dpa_stats_perf_event_sample

L

M

N

O

P

Q

R

rkey
flexio_dev_wqe_rdma_seg
rlid
flexio_qp_attr
rmpqn
flexio_qp_attr
rq
flexio_wq_attr
rq_cqn
flexio_qp_attr
rq_type
flexio_qp_attr
rsvd
flexio_dev_cqe64
rsvd0
flexio_dev_wqe_eth_seg
flexio_dev_wqe_rdma_seg
flexio_dev_wqe_transpose_seg
flexio_dev_mini_cqe64
flexio_dev_wqe_shared_receive_seg
rsvd00
flexio_dev_eqe
flexio_dev_cqe64
rsvd02
flexio_dev_eqe
rsvd1
flexio_dev_wqe_shared_receive_seg
flexio_dev_wqe_transpose_seg
rsvd14
flexio_dev_cqe64
rsvd2
flexio_dev_wqe_eth_seg
flexio_dev_wqe_transpose_seg
rsvd22
flexio_dev_cqe64
rsvd3c
flexio_dev_eqe
rsvd4
flexio_dev_eqe
flexio_dev_wqe_transpose_seg
rsvd48
flexio_dev_cqe64

S

T

U

V

W

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com