MVA™ Application Programming Interface

Version 2.0.1.0

May 14, 2021

Contents

Chapter 1

MVA Documentation

1.1 Introduction

The Myricom Machine Vision Accelerator MVA™ solution greatly improves the performance of machine vision applications processing data from GigE Vision devices. MVA dramatically reduces the host processor overhead while providing maximum throughput when receiving GigE Vision Stream Protocol (GVSP) content.

1.2 Terminology

Readers of this document should be familiar with the AIA GigE Vision Specification version 1.x and the GenICam™ Standard. See www.machinevisiononline.org and www.genicam.org respectively for more detail. "Application" in this document describes the consumer of the MVA interface, commonly GigE Vision libraries or development kits.

1.3 Overview

MVA leverages Myri-10G programmable 10-Gigabit Ethernet network interface cards (NICs) with custom firmware to divert GVSP data directly to user-space memory, bypassing the operating system and legacy network software stacks. MVA offloads the reassembly of GVSP data blocks from individual packets on the wire, avoiding intermediate memory copies and context switch overhead. Optionally, MVA can handle GVSP reliability in NIC firmware, requesting retransmission of lost packets in real-time without host involvement.

MVA is composed of a user library, driver, and firmware running on the embedded processor of the Myri-10G network adapter.

1.4 Streams

An application opens a GVSP stream by specifing a destination address and port using the GigE Vision SCDx and S-CPx registers. A matching MVA stream is created by passing the same parameters to an mva_open_stream() function, including mva_open_stream_pr(). The destination address should match the Ethernet interface address of an MVA--Enabled Myri-10G NIC. Only GVSP traffic associated to MVA streams is handled by the MVA stack, all other GVSP packets are directed to the legacy network stack in the operating system.

1.5 Memory management

Memory directly accessible by the network adapter DMA engine must be pinned to physical pages. The application uses mva_alloc() to allocate such memory. MVA buffers can be of any size, large enough to contain one or more GVSP data blocks. It is an application error to free MVA memory while it in use.

1.6 Receiving data blocks

MVA delivers GVSP data blocks into buffers queued into the NIC using mva_queue_buffer(). Multiple buffers can be queued for a particular stream and they are used in the order they are enqueued. Buffers should be large enough to contain the corresponding data block payload. For example, this size could be the PayloadSize value in the GenICam Device Description file. Once a buffer is queued, its ownership is transfered to the MVA library.

The application should ensure that there is always a buffer available to receive incoming data for a given stream. If no buffer is available, all the packets related to the current data block are dropped. Queing multiple buffers allows for consecutive data blocks to be received on a stream without host involvement.

MVA provides several modes for receiving data from GigE Vision devices. Drop mode instructs the NIC to jettison the entire GVSP data block if one or more packet is missing. In Zeroloss mode, the NIC firmware will request retransmission of missing packets according to configurable timeouts.

The application uses mva_poll_recv() to wait for the reception of the next GVSP data block on a given MVA stream. This function returns when a data block has been received into the next queued buffer or when the timeout has expired, whichever comes first. If successful, the address of the corresponding buffer and the actual size of the data block is returned, along with the related GVSP metadata. Once mva_poll_recv() indicates that a data block has been received into a buffer, the ownership of this buffer is transfered back to the application. When its content has been processed, the buffer can safely be queued again for any stream.

Chapter 2

Module Index

2.1 API Reference

Here is a list of all modules:

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

Module Documentation

5.1 Initialization

MVA Initialization function.

Macros

- #define MVA_VERSION_API 0x0102

  MVA API version number (16 bits)

Enumerations

- enum mva_link_state { **MVA_LINK_DOWN** = 0, **MVA_LINK_UP** = 1 }
- enum mva_timesource_state { MVA_TIMESOURCE_LOCAL = 0, MVA_TIMESOURCE_EXT_UNSYNC-ED, MVA_TIMESOURCE_EXT_SYNCED, MVA_TIMESOURCE_EXT_FAILED }

Functions

- mva_init (uint16_t api_version)

  Initialize MVA library.

5.1.1 Detailed Description

MVA Initialization function.

5.1.2 Macro Definition Documentation

5.1.2.1 #define MVA_VERSION_API 0x0102

MVA API version number (16 bits)

LSB increases for minor backwards compatible changes in the API. MSB increases for incompatible changes in the API.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum mva_link_state

Link state enumeration, returned by mva_get_link_state.

5.1.3.2 enum mva_timesource_state

Timesource state (for -SYNC NICs), returned by mva_get_timesource_state.

Enumerator

MVA_TIMESOURCE_LOCAL Local timesource (no external). Returned if there is no available external timesource or if its use was explicitly disabled.

MVA_TIMESOURCE_EXT_UNSYNCED External Timesource: not synchronized (yet).

MVA_TIMESOURCE_EXT_SYNCED External Timesource: synchronized.

MVA_TIMESOURCE_EXT_FAILED External Timesource: NIC failure to connect to source.

5.1.4 Function Documentation

5.1.4.1 mva_init (uint16_t *api_version*)

Initialize MVA library.

This function initializes the MVA library, verifies driver and linked library compatibility, and checks the license and allocates device-independent resources.

Parameters

`in`	*api_version*	Must always be MVA_VERSION_API.

Return values

EINVAL	Library already initialized with different API version. Incompatible library/API version.
ENXIO	Incompatible driver/library.
ENODEV	No driver found.
ENOMEM	Not enough available memory.

Remarks

This function should be invoked prior to any other MVA calls. It can be called multiple times, as long as the API version is the same.

5.2 Streams

Functions related to MVA Streams.

Data Structures

- struct mva_packet_resend_config

Macros

- #define MVA_OPEN_IPV6 0x1
- #define MVA_OPEN_ZEROLOSS 0x2
- #define MVA_OPEN_DROP_INCOMPLETE 0x4
- #define MVA_OPEN_RETURN_INCOMPLETE 0x8
- #define MVA_OPEN_GEV2X 0x10
- #define MVA_BOOO_ALWAYS_DROP 0x1
- #define MVA_BOOO_ALWAYS_RETURN 0x2
- #define MVA_BOOO_LT_ALWAYS_RETURN 0x3
- #define MVA_BOOO_GT_ALWAYS_RETURN 0x4

Typedefs

- typedef struct mva_stream ∗ mva_stream_t

Functions

- mva_open_stream (void ∗in_addr, uint16_t dest_port, mva_stream_t ∗mva_stream, int flags, MVA_OS_HANDLE ∗os_handle)

  Open a GigE Vision stream for MVA acceleration.
- mva_open_stream_pr (void ∗in_addr, uint16_t dest_port, mva_stream_t ∗mva_stream, struct mva_packet_resend_config ∗prc, MVA_OS_HANDLE ∗os_handle)

  Open a GigE Vision stream for MVA acceleration with Packet Resend support.
- mva_open_mcast_stream (void ∗in_addr, void ∗mcast_in_addr, uint16_t dest_port, mva_stream_t ∗mva_stream, int flags, MVA_OS_HANDLE ∗os_handle)

  Open a GigE Vision Multicast group stream for MVA acceleration. The Ethernet addess must match the interface address of a Myri-10G network adapter (NIC) with MVA enabled.
- mva_close_stream (mva_stream_t mva_stream)

  Close an MVA Stream.
- mva_get_link_state (mva_stream_t strm, enum mva_link_state ∗state)
- mva_get_timesource_state (mva_stream_t strm, enum mva_timesource_state ∗state)

5.2.1 Detailed Description

Functions related to MVA Streams.

5.2.2 Macro Definition Documentation

5.2.2.1 #define MVA_BOOO_ALWAYS_DROP 0x1

The packet resend config object is used by the open stream with packet resend allowed function. It's used to config how resend will work. The object will remain in effect until the stream is closed.

Always drop out-of-order blocks.

5.2.2.2 #define MVA_BOOO_ALWAYS_RETURN 0x2

Always return out-of-order blocks.

5.2.2.3 #define MVA_BOOO_GT_ALWAYS_RETURN 0x4

Only return out-of-order blocks if the next packet ID is greater than the expected one.

5.2.2.4 #define MVA_BOOO_LT_ALWAYS_RETURN 0x3

Only return out-of-order blocks if the next packet ID is less than the expected one.

5.2.2.5 #define MVA_OPEN_DROP_INCOMPLETE 0x4

Stream operates on "drop incomplete" mode.

5.2.2.6 #define MVA_OPEN_GEV2X 0x10

GEV 2.x Flag

5.2.2.7 #define MVA_OPEN_IPV6 0x1

Address speciefied to mva_open_stream() is an IPv6 address (default: IPv4).

5.2.2.8 #define MVA_OPEN_RETURN_INCOMPLETE 0x8

Stream operates on "return incomplete" mode.

5.2.2.9 #define MVA_OPEN_ZEROLOSS 0x2

Steam operates in "zeroloss" modes (default: "drop" mode).

5.2.3 Typedef Documentation

5.2.3.1 typedef struct mva_stream∗ mva_stream_t

Opaque stream handle structure.

5.2.4 Function Documentation

5.2.4.1 mva_close_stream (mva_stream_t *mva_stream*)

Close an MVA Stream.

This function closes a stream from MVA acceleration.

Parameters

`in`	*mva_stream*	MVA stream handle.

Postcondition

The MVA stream handle is no longer valid and cannot be used for any other functions. All queued buffers are released, and their ownership is transferred back to the application.

5.2.4.2 mva_get_link_state (mva_stream_t *strm*, enum mva_link_state ∗ *state*)

Get link status on opened handle.

Parameters

strm	Stream handle.
state	Returns one of MVA_LINK_DOWN or MVA_LINK_UP.

Remarks

The cost of retrieving the link state requires a function call that reads state kept in kernel host memory (i.e., no PCI bus reads).

5.2.4.3 mva_get_timesource_state (mva_stream_t *strm*, enum mva_timesource_state ∗ *state*)

Get timesource information from opened handle.

Parameters

strm	Stream handle.
state	Returns one of mva_timesource_state.

Remarks

The cost of retrieving the timesource state requires a function call that reads state kept in kernel host memory (i.e., no PCI bus reads).

5.2.4.4 mva_open_mcast_stream (void ∗ *in_addr,* void ∗ *mcast_in_addr,* uint16_t *dest_port,* mva_stream_t ∗ *mva_stream,* int *flags,* MVA_OS_HANDLE ∗ *os_handle*)

Open a GigE Vision Multicast group stream for MVA acceleration. The Ethernet addess must match the interface address of a Myri-10G network adapter (NIC) with MVA enabled.

This function is identical to mva_open_stream, except that a Multicast group address is specified as an additional parameter. The multicast address is joined to the interface address specified by the in_addr parameter. GVSP packets with the specified multicast address will be accepted. If the multicast address is NULL, the function behaves exactly like mva_open_stream.

Parameters

`in`	*in_addr*	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): • IPv4: (struct in_addr ∗) • IPv6: (struct in6_addr ∗)
`in`	*mcast_addr*	Pointer to a multicast address structure for the address that will receive GigE Vision frames. • IPv4: (struct in_addr ∗) • IPv6: (struct in6_addr ∗)
`in`	*dest_port*	Destination port for the stream (from SCP register).
`in`	*flags*	Flags are single bit values; so, binary or (|) can be used to combine them. Possible values are: • MVA_OPEN_IPV6 • MVA_OPEN_ZEROLOSS
`out`	*mva_stream*	MVA stream handle.
`out`	*os_handle*	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor, on Windows it is a HANDLE. Specify NULL if handle is not needed.

Return values

EINVAL	The dest_addr was not an interface address for a Myri-10G MVA-enabled network adapter, or the multicast address was not valid.

Postcondition

The MVA stream handle is valid and can be used by other functions.

5.2.4.5 mva_open_stream (void ∗ *in_addr,* uint16_t *dest_port,* mva_stream_t ∗ *mva_stream,* int *flags,* MVA_OS_HANDLE ∗ *os_handle*)

Open a GigE Vision stream for MVA acceleration.

This function opens a GigE Vision stream channel for acceleration. The addess must match an Ethernet interface address of a Myri-10G network adapter (NIC) with MVA enabled.

By default, streams operate in "drop" mode, where the entire data block is dropped if one related packet is lost. If the MVA_OPEN_ZEROLOSS flag is specified, the NIC will make resonable attempts to retrieve any missing packets.

Parameters

`in`	*in_addr*	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): • IPv4: (struct in_addr ∗) • IPv6: (struct in6_addr ∗)
`in`	*dest_port*	Destination port for the stream (from SCP register)
`in`	*flags*	Flags are single bit values; so, binary or (|) can be used to combine them. Possible values are: • MVA_OPEN_IPV6 • MVA_OPEN_ZEROLOSS
`out`	*mva_stream*	MVA stream handle.
`out`	*os_handle*	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor; on Windows it is a HANDLE. Specify NULL if handle is not needed.

Return values

EINVAL	The dest_addr was not an interface address for a Myri-10G MVA-enabled network adapter.

Postcondition

The MVA stream handle is valid and can be used by other functions.

5.2.4.6 mva_open_stream_pr (void ∗ *in_addr*, uint16_t *dest_port*, mva_stream_t ∗ *mva_stream*, struct mva_packet_resend_config ∗ *prc*, MVA_OS_HANDLE ∗ *os_handle*)

Open a GigE Vision stream for MVA acceleration with Packet Resend support.

This function opens a GigE Vision stream channel for acceleration. The addess must match an Ethernet interface address of a Myri-10G network adapter (NIC) with MVA enabled.

By default, streams operate in "drop" mode, where the entire data block is dropped if one related packet is lost. If the MVA_OPEN_ZEROLOSS flag is specified, the NIC will make resonable attempts to retrieve any missing packets. MVA_OPEN_RETURN_INCOMPLETE can also be set to return incomplete streams.

Parameters

`in`	*in_addr*	Pointer to an interface address structure for the address of an open GigE Vision stream (from the SCDA register): • IPv4: (struct in_addr ∗) • IPv6: (struct in6_addr ∗)
`in`	*dest_port*	Destination port for the stream (from SCP register).
`in`	*prc*	The packet resend config object for the stream.

out	*mva_stream*	MVA stream handle.
out	*os_handle*	OS-specific file descriptor which can be passed to poll() or select() to block on receive data available. For UNIX systems, this is a file descriptor; on Windows it is a HANDLE. Specify NULL if handle is not needed.

Return values

EINVAL	The dest_addr was not an interface address for a Myri-10G MVA-enabled network adapter.

Postcondition

The MVA stream handle is valid and can be used by other functions.

5.3 Memory allocation and deallocation.

Functions related to MVA Memory allocation and deallocation.

Functions

- mva_alloc (mva_stream_t mva_stream, size_t size, mva_buf_t ∗buf)

  Allocate MVA memory.
- mva_free (mva_buf_t buf)

  Free MVA memory.

5.3.1 Detailed Description

Functions related to MVA Memory allocation and deallocation.

5.3.2 Function Documentation

5.3.2.1 mva_alloc (mva_stream_t *mva_stream,* size_t *size,* mva_buf_t ∗ *buf*)

Allocate MVA memory.

This function allocates an MVA zero-copy buffer of the specified size and returns a pointer to the corresponding memory. MVA buffers are pinned in physical memory to allow direct access by the DMA engine of the Myricom network adapter. After a buffer has been allocated, it can be queued to MVA. See mva_queue_buffer().

Parameters

`in`	*mva_stream*	An open MVA stream on which the buffer will be queued.
`in`	*size*	Size of the buffer to allocate.
`out`	*buf*	MVA buffer handle.

Return values

0	Success.
EINVAL	Invalid mva_stream or size parameter.
ENOMEM	Out of resources.

Postcondition

The MVA buffer handle is valid and can be enqueued on a stream with mva_queue_buffer().

5.3.2.2 mva_free (mva_buf_t *buf*)

Free MVA memory.

This function frees memory previously allocated by mva_alloc().

Parameters

`in`	*buf*	Buffer handle.

Return values

0	Success.

Postcondition

The MVA buffer can no longer be used.

5.4 Enqueuing and Receiving data blocks.

Functions related enqueuing and receiving GVSP data blocks.

Data Structures

- struct mva_block
- struct mva_block_status
- struct mva_info
- struct mva_stats

Macros

- #define MVA_BLOCK_MAX_FILENAME 32
- #define MVA_BLOCK_STATUS_NO_BLOCK_RECEIVED 0x0
- #define **MVA_BLOCK_STATUS_BLOCK_DROPPED** 0x1
- #define **MVA_BLOCK_STATUS_BLOCK_RETURNED_COMPLETE** 0x2
- #define **MVA_BLOCK_STATUS_BLOCK_RETURNED_INCOMPLETE** 0x3

Functions

- mva_queue_buffer (mva_buf_t buf)

  Queue an MVA buffer.
- mva_poll_recv (mva_stream_t mva_stream, struct mva_block ∗block, int timeout)

  Wait for the next GVSP data block.
- mva_poll_recv_pr (mva_stream_t mva_stream, struct mva_block ∗block, uint16_t req_id, struct mva_block_-status ∗block_status)

  Wait for the next GVSP data block.
- mva_info (mva_stream_t mva_stream, struct mva_info ∗info)

  Get device info.
- mva_get_stats (mva_stream_t mva_stream, struct mva_stats ∗stats)

  Get stats.
- mva_clear_stats (mva_stream_t mva_stream)

  Clear stats.

5.4.1 Detailed Description

Functions related enqueuing and receiving GVSP data blocks.

5.4.2 Macro Definition Documentation

5.4.2.1 #define MVA_BLOCK_MAX_FILENAME 32

The maximum filename size of a file payload

5.4.2.2 #define MVA_BLOCK_STATUS_NO_BLOCK_RECEIVED 0x0

mva_block_status is used by mva_poll_recv_pr.

5.4.3 Function Documentation

5.4.3.1 mva_clear_stats (mva_stream_t *mva_stream*)

Clear stats.

This function clears the various information, counters, and statistics that are returned by mva_get_stats for a given stream.

Parameters

in	*mva_stream*	MVA stream for which the statistics are cleared.

Return values

0	Success.

5.4.3.2 mva_get_stats (mva_stream_t *mva_stream*, struct mva_stats ∗ *stats*)

Get stats.

This function returns various information, counters, and statistics for a given MVA stream.

Parameters

in	*mva_stream*	MVA stream for which statistics are retrieved.
out	*info*	MVA stats structure.

Return values

0	Success.

5.4.3.3 mva_info (mva_stream_t *mva_stream*, struct mva_info ∗ *info*)

Get device info.

This function returns various information, counters, and statistics for a given MVA stream.

Parameters

in	*mva_stream*	The MVA stream for which statistics are retrieved.
out	*info*	MVA stream info structure.

Return values

0	Success.

5.4.3.4 mva_poll_recv (mva_stream_t *mva_stream,* struct mva_block ∗ *block,* int *timeout*)

Wait for the next GVSP data block.

Wait for the next GVSP data block on the specified stream. If the next data block has not been completely received, the function waits until completion or until the timeout expires. On success, the corresponding mva_block structure is updated with the information specific to the received GVSP payload.

Parameters

in	*mva_stream*	MVA stream.
in,out	*block*	MVA block structure allocated by the application and initialized with information about received block upon completion.
in	*timeout*	Timeout in msecs. If this parameter is 0, the function will return immediately. If the parameter is negative, the function will block indefinitely.

Return values

0	Success.
EINVAL	Invalid mva_stream.
EAGAIN	Timeout expired and no block was available. Fields in mva_info will be invalid.
EINTR	Signal was received. Fields in mva_block will be invalid.
ENOMEM	The queued buffer was not large enough to contain the block.

Postcondition

The buffer handle can be used to free or re-queue the buffer on the stream. After the buffer is freed or re-queued, the payload_data pointer in the block is no longer valid.

5.4.3.5 mva_poll_recv_pr (mva_stream_t *mva_stream,* struct mva_block ∗ *block,* uint16_t *req_id,* struct mva_block_status ∗ *block_status*)

Wait for the next GVSP data block.

Wait for the next GVSP data block on the specified stream. If the next data block has not been completely received, the function waits until completion or until the timeout expires. On success, the corresponding mva_block structure is updated with the information specific to the received GVSP payload.

Parameters

in	*mva_stream*	MVA stream.
in,out	*block*	MVA block structure allocated by the application and initialized with information about the received block upon completion.
in	*req_id*	The first request ID to be issued to the device if a packet resend request is issued.
out	*block_status*	The resulting status returned from the function.

Return values

0	Success.
EINVAL	Invalid mva_stream.
EAGAIN	Timeout expired and no block was available. Fields in mva_info will be invalid.
EINTR	Signal was received. Fields in mva_block will be invalid.

ENOMEM	The queued buffer was not large enough to contain the block
ETIMEDOUT	A packet resend timeout expired and no block was available.
EIO	The block was out of order and dropped based on the BOOO parameter.

Postcondition

The buffer handle can be used to free or re-queue the buffer on the stream. After the buffer is freed or re-queued, the payload_data pointer in the block is no longer valid.

5.4.3.6 mva_queue_buffer (mva_buf_t *buf*)

Queue an MVA buffer.

Queue an MVA buffer for receiving a GVSP data block from a specific MVA stream. Once a buffer is enqueued, it should not be accessed until it is returned by mva_poll_recv(). Buffers are automatically dequeued when mva_poll_-recv() returns.

Parameters

`in`	*buffer*	Buffer handle.

Return values

0	Success.
EINVAL	Invalid buffer address.
EAGAIN	Max amount of queue ahead reached.

Postcondition

Buffer is queued to receive GVSP block data from stream from which the buffer was allocated.

Remarks

Once the buffer has been queued, it should not be read or written until it is returned by mva_poll_recv().

5.5 Manage the name of an MVA adapter.

Functions related to configuring the name of an MVA adapter.

Functions

- mva_set_name (const char ∗which_adapter, const char ∗name)

  Set the name of an adapter.
- mva_get_name (const char ∗which_adapter, const char ∗∗name)

  Get the name of an adapter.
- mva_reset_name (const char ∗which_adapter)

  Reset the name of an adapter.
- mva_reset_all_name (void)

  Reset the names of all adapters.

5.5.1 Detailed Description

Functions related to configuring the name of an MVA adapter.

5.5.2 Function Documentation

5.5.2.1 mva_get_name (const char ∗ *which_adapter,* const char ∗∗ *name*)

Get the name of an adapter.

This function returns the name of an MVA adapter.

Parameters

in	*which_adapter*	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.
out	*name*	The name of the adapter.

Return values

0	Success.

5.5.2.2 mva_reset_all_name (void)

Reset the names of all adapters.

This function resets the name of all MVA adapters to the default of <adapter-model>.<SN>.

Return values

0	Success.

5.5.2.3 mva_reset_name (const char ∗ *which_adapter*)

Reset the name of an adapter.

This function resets the name of an MVA adapter to the default of <adapter-model>.<SN>.

Parameters

`in`	*which_adapter*	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.

Return values

0	Success.

5.5.2.4 mva_set_name (const char ∗ *which_adapter,* const char ∗ *name*)

Set the name of an adapter.

This function sets the name of the adapter.

Parameters

`in`	*which_adapter*	This specifies a card by its MAC or SN based on the format S=477237 for serial numbers or M=00:60:dd:43:a3:61 for MAC addresses.
`in`	*name*	The name to set for the adapter.

Return values

0	Success.

5.6 receive worker thread.

MVA receive worker thread (RWDT) component.

Data Structures

- struct mva_receive_stream_worker_status

Typedefs

- typedef void(∗ mva_get_request_id_cb)(mva_stream_t mva_stream, uint16_t ∗starting_request_id, uint16_t ∗max_requests)

Functions

- mva_create_recv_worker (mva_stream_t mva_stream, int cpu, mva_get_request_id_cb callback_func)

  Create an MVA receive worker for the specified stream.
- mva_destroy_recv_worker (mva_stream_t mva_stream)

  Destroy an MVA receive worker for the specified stream.
- mva_block_recv_worker (mva_stream_t mva_stream)

  Block the receive worker thread.
- mva_unblock_recv_worker (mva_stream_t mva_stream)

  Unblock the Receive Worker thread.
- mva_stop_recv_worker (mva_stream_t mva_stream)

  Stop the receive worker thread.
- mva_start_recv_worker (mva_stream_t mva_stream)

  Start the receive worker thread.
- mva_get_status_recv_worker (mva_stream_t mva_stream, struct mva_receive_stream_worker_status ∗recv_-worker_status)

  Get the status of the receive worker thread.
- mva_get_camera_id_stream (mva_stream_t mva_stream, uint64_t ∗camera_id)

  Get Camera ID.
- mva_drop_packet (mva_stream_t mva_stream, uint32_t block, uint32_t packet)

  Drop packet.

5.6.1 Detailed Description

MVA receive worker thread (RWDT) component.

5.6.2 Typedef Documentation

5.6.2.1 typedef void(∗ mva_get_request_id_cb)(mva_stream_t mva_stream, uint16_t ∗starting_request_id, uint16_t ∗max_requests)

This is the callback used by the calling application for the worker thread to obtain the starting request ID and maximum number of requests.

5.6.3 Function Documentation

5.6.3.1 mva_block_recv_worker (mva_stream_t *mva_stream*)

Block the receive worker thread.

This function allows the user to block processing of the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

Parameters

`in`	*mva_stream*	MVA stream to block.

Return values

0	Success. #retval EINVAL The receive worker thread has not been created.

5.6.3.2 mva_create_recv_worker (mva_stream_t *mva_stream*, int *cpu*, mva_get_request_id_cb *callback_func*)

Create an MVA receive worker for the specified stream.

This function creates a receive worker for the specified stream. If the stream is not opened, the function will return non-zero.

Parameters

`in`	*mva_stream*	The stream for which to create a worker thread.
`in`	*cpu*	The CPU to which the worker thread is affinitized.
`in`	*callback_func*	The callback function used by the worker thread to obtain the next request ID.

Return values

0	Success.
ENOENT	The mva_stream is not open.
EINVAL	Invalid callback function.
EINVAL	Invalid CPU.
ENOMEM	Failed to allocate the receive worker.

5.6.3.3 mva_destroy_recv_worker (mva_stream_t *mva_stream*)

Destroy an MVA receive worker for the specified stream.

Stop and destroy a created receive worker thread.

Parameters

`in`	*mva_stream*	The stream where the worker thread will be destroyed.

Return values

0	Success.

5.6.3.4 mva_drop_packet (mva_stream_t *mva_stream*, uint32_t *block*, uint32_t *packet*)

Drop packet.

This function drops a packet on the the given stream.

Parameters

in	*mva_stream*	MVA stream associated with the drop.
in	*block*	Block ID containing the packet to drop.
in	*packet*	ID of the packet to drop.

Return values

0	Success.

5.6.3.5 mva_get_camera_id_stream (mva_stream_t *mva_stream*, uint64_t ∗ *camera_id*)

Get Camera ID.

This function returns the camera ID for the given stream.

Parameters

in	*mva_stream*	The MVA stream from which to obtain the camera ID.

Return values

0	Success.

5.6.3.6 mva_get_status_recv_worker (mva_stream_t *mva_stream*, struct mva_receive_stream_worker_status ∗ *recv_worker_status*)

Get the status of the receive worker thread.

This function allows the user to get the state of the stream worker thread.

Parameters

in	*mva_stream*	MVA stream to start.

Return values

0	Success.

5.6.3.7 mva_start_recv_worker (mva_stream_t *mva_stream*)

Start the receive worker thread.

This function allows the user to start processing the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

Parameters

in	*mva_stream*	MVA stream to start.

Return values

0	Success. #retval EINVAL The receive worker thread has not been created.

5.6.3.8 mva_stop_recv_worker (mva_stream_t *mva_stream*)

Stop the receive worker thread.

This function allows the user to stop processing the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

Parameters

in	*mva_stream*	MVA stream to stop.

Return values

0	Success. #retval EINVAL The receive worker thread has not been created.

5.6.3.9 mva_unblock_recv_worker (mva_stream_t *mva_stream*)

Unblock the Receive Worker thread.

This function allows the user to unblock processing of the stream worker thread for a given stream. If this function is called and the worker thread is not created, it will return EINVAL.

Parameters

in	*mva_stream*	MVA stream to unblock.

Return values

0	Success. #retval EINVAL The receive worker thread has not been created.

Chapter 6

Namespace Documentation

6.1 mva Namespace Reference

6.1.1 Detailed Description

MachineVisionAccelerator

Author

Myricom, Inc.

Chapter 7

Data Structure Documentation

7.1 mva_block Struct Reference

Data Fields

- uint16_t payload_type
- uint32_t payload_length
- void ∗ payload_data
- mva_buf_t mva_buf
- uint16_t status
- uint64_t block_id
- uint64_t timestamp
- uint64_t nsecs
- uint32_t crc
- uint32_t chunk_data_payload_Length
- uint32_t chunk_layout_id
- union {
    struct {
      uint32_t **pixel_type**
      uint32_t **size_x**
      uint32_t **size_y**
      uint32_t **offset_x**
      uint32_t **offset_y**
      uint32_t **padding_x**
      uint32_t **padding_y**
      uint32_t **trailer_size_y**
    } **image**
    struct {
      uint64_t **payload_data_size**
    } **raw**
    struct {
      uint64_t **payload_data_size**
      char **filename** [MVA_BLOCK_MAX_FILENAME]
    } **file**

```
    struct {
      uint64_t data_payload_length
    } chunk
    struct {
      uint8_t chunk_flag
      uint32_t pixel_type
      uint32_t size_x
      uint32_t size_y
      uint32_t offset_x
      uint32_t offset_y
      uint32_t padding_x
      uint32_t padding_y
      uint32_t trailer_size_y
      uint64_t data_payload_length
      uint32_t chunk_layout_id
    } extended_chunk
  } meta
```

7.1.1 Detailed Description

GVSP block.

7.1.2 Field Documentation

7.1.2.1 uint64_t mva_block::block_id

Block ID.

7.1.2.2 uint32_t mva_block::chunk_data_payload_Length

Chunk Data Payload Length

7.1.2.3 uint32_t mva_block::chunk_layout_id

Chunk Layout ID

7.1.2.4 uint32_t mva_block::crc

Block CRC (for internal testing).

7.1.2.5 union { ... } mva_block::meta

Payload type specific metadata.

7.1.2.6 mva_buf_t mva_block::mva_buf

MVA buffer to use with mva_queue_buffer() or mva_free().

7.1.2.7 uint64_t mva_block::nsecs

If SYNC NIC in use, time since Epoc in nanoseconds of the arrival of the Leader packet; else 0.

7.1.2.8 void∗ mva_block::payload_data

Pointer to block data.

7.1.2.9 uint32_t mva_block::payload_length

Total length of block data.

7.1.2.10 uint16_t mva_block::payload_type

Payload type.

7.1.2.11 uint16_t mva_block::status

Status of block transaction.

7.1.2.12 uint64_t mva_block::timestamp

Timestamp.

7.2 mva_block_status Struct Reference

Data Fields

- uint32_t status
- uint64_t block_id
- uint32_t dropped_packet_id
- uint32_t resend_request
- uint16_t next_req_id

7.2.1 Field Documentation

7.2.1.1 uint64_t mva_block_status::block_id

The ID of the returned block.

7.2.1.2 uint32_t mva_block_status::dropped_packet_id

If the block was incomplete, the dropped packet ID; otherwise UINT32_MAX.

7.2.1.3 uint16_t mva_block_status::next_req_id

The next request ID.

7.2.1.4 uint32_t mva_block_status::resend_request

The number of times a packet resesnd request was performed.

7.2.1.5 uint32_t mva_block_status::status

The status of the returned block.

7.3 mva_info Struct Reference

Data Fields

- uint32_t port_link_up
- uint32_t port_active
- uint64_t blocks_dropped
- uint64_t blocks_received

7.3.1 Detailed Description

Various information related to MVA stream.

7.3.2 Field Documentation

7.3.2.1 uint64_t mva_info::blocks_dropped

Number of GVSP data blocks dropped or with a non-zero status.

7.3.2.2 uint64_t mva_info::blocks_received

Number of GVSP data blocks received with status equal to 0.

7.3.2.3 uint32_t mva_info::port_active

Active port on failover NIC.

7.3.2.4 uint32_t mva_info::port_link_up

Bitmap of ports with link up.

7.4 mva_packet_resend_config Struct Reference

Data Fields

- int flags
- uint32_t block_timeout
- uint32_t resend_timeout
- uint32_t resend_retries
- uint32_t block_out_of_order
- uint64_t camera_id
- uint32_t camera_ipv4
- uint16_t camera_port
- uint16_t stream_channel_id

7.4.1 Field Documentation

7.4.1.1 uint32_t mva_packet_resend_config::block_out_of_order

The Block Out of Order (BOOO) flag.

7.4.1.2 uint32_t mva_packet_resend_config::block_timeout

The time, in milliseconds, the API waits for a resend after a first incomplete block.

7.4.1.3 uint64_t mva_packet_resend_config::camera_id

Unique camera app identifacation field.

7.4.1.4 uint32_t mva_packet_resend_config::camera_ipv4

The Unicast IPv4 Address to which the packet resend requests are issued.

7.4.1.5 uint16_t mva_packet_resend_config::camera_port

The port to use when issuing the packet resend requests.

7.4.1.6 int mva_packet_resend_config::flags

The MVA open flags.

7.4.1.7 uint32_t mva_packet_resend_config::resend_retries

The maximum number of packet resend retries allowed.

7.4.1.8 uint32_t mva_packet_resend_config::resend_timeout

The time, in milliseconds, the API waits for a single packet resend request response.

7.4.1.9 uint16_t mva_packet_resend_config::stream_channel_id

The stream ID for the stream.

7.5 mva_receive_stream_worker_status Struct Reference

Data Fields

- uint32_t **thread_state**
- uint32_t **thread_mode**
- uint32_t **block_processing_mode**

7.6 mva_stats Struct Reference

Data Fields

- uint64_t blocks_dropped
- uint64_t blocked_blocks_dropped
- uint64_t blocks_returned_complete
- uint64_t blocks_returned_incomplete
- uint64_t blocks_not_available
- uint64_t resend_requests
- uint64_t resend_requests_successful
- uint64_t resend_requests_failed
- uint64_t block_time_outs
- uint64_t resend_time_outs
- uint64_t max_resend_retries
- uint64_t buf_allocated
- uint64_t buf_freed
- uint64_t buf_enqueued
- uint64_t stream_stopped
- uint64_t stream_started
- uint64_t stream_blocked
- uint64_t stream_unblocked
- uint64_t callbacks_invoked

7.6.1 Detailed Description

MVA stats

7.6.2 Field Documentation

7.6.2.1 uint64_t mva_stats::block_time_outs

Number of block timeouts.

7.6.2.2 uint64_t mva_stats::blocked_blocks_dropped

Number of GVSP data blocks dropped when in the blocked stream state.

7.6.2.3 uint64_t mva_stats::blocks_dropped

Number of GVSP data blocks dropped or with a non-zero status.

7.6.2.4 uint64_t mva_stats::blocks_not_available

Number of GVSP data blocks returned as not available.

7.6.2.5 uint64_t mva_stats::blocks_returned_complete

Number of GVSP data blocks returned as completed.

7.6.2.6 uint64_t mva_stats::blocks_returned_incomplete

Number of GVSP data blocks returned as incomplete.

7.6.2.7 uint64_t mva_stats::buf_allocated

Number of buffers allocated.

7.6.2.8 uint64_t mva_stats::buf_enqueued

Number of buffers enqueued.

7.6.2.9 uint64_t mva_stats::buf_freed

Number of buffers freed.

7.6.2.10 uint64_t mva_stats::callbacks_invoked

Number of times the get starting request ID callback function was invoked.

7.6.2.11 uint64_t mva_stats::max_resend_retries

Number of times the resend retries counter was exceeded.

7.6.2.12 uint64_t mva_stats::resend_requests

Number of GVCP Packet Resend requests issued.

7.6.2.13 uint64_t mva_stats::resend_requests_failed

Number of GVCP Packet Resend requests failed.

7.6.2.14 uint64_t mva_stats::resend_requests_successful

Number of GVCP Packet Resend requests successful.

7.6.2.15 uint64_t mva_stats::resend_time_outs

Number of resend timeouts.

7.6.2.16 uint64_t mva_stats::stream_blocked

Number of RWDT block states entered.

7.6.2.17 uint64_t mva_stats::stream_started

Number of RWDT start states entered.

7.6.2.18 uint64_t mva_stats::stream_stopped

Number of RWDT stop states entered.

7.6.2.19 uint64_t mva_stats::stream_unblocked

Number of RWDT unblock states entered.

Index