VideoListener

User's Manual

Contents

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors

Chapter 1

Revision History

Date	Revision	Author	Changes
02-Sep-2016	0.1	Ludovit Minarik	Initial version
06-Sep-2016	0.2	Ludovit Minarik	Prepared for BETA 0.9.0
08-Sep-2016	0.3	Ludovit Minarik	Correction of review findings

© 2016 NXP Semiconductors

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

© 2016 NXP Semiconductors

Chapter 3

File Index

3.1 File List

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

© 2016 NXP Semiconductors

Chapter 4

Module Documentation

4.1 Decoder Feeding Driver

4.1.1 Detailed Description

The Decoder Feeding module is intended to act as a JPEG/H264 decoder input handling driver. It manages input queue of the decoder and provides interface allowing user to enqueue frames to be decoded. Module also contains auxiliary API enabling access to current queue and error status of the peripheral.

Since only input of the decoder is being handled within this module it needs to be used within an environment where a complete decoder driver takes care of decoder's configuration and output management.

Note

A supported decoder may contain multiple stream instances what means that is able to process multiple separated data streams in parallel. The feeding driver uses term "stream index" to enumerate particular HW instances.

Theory of operation

First of all the module needs to be correctly configured. Configuration is done by setting values of configuration macros within the dec_feed_cfg.h to desired values and providing runtime configuration via:

- DECFEED_SetBaseAddress()
- DECFEED_SetRegionLength()
- DECFEED_SetBurstLength()
- DECFEED_SetNumOfStreams()

When properly configured, module needs to be initialized by DECFEED_Init() and started by DECFEED_Start(). From now the user can enqueue frames to be decoded via DECFEED_Push(). Queue is being maintained by periodic calls to the DECFEED_HandleProcessedFrms() and user shall ensure that the function is called often enough to keep the operation fluent.

Module can be stopped by DECFEED_Stop() and re-initialized by following call to the DECFEED_Init().

Files

- file dec_feed_cfg.h

  Static configuration of video decoder driver.
- file dec_feed.h

  API of the decoder driver.
- file dec_feed.c

  The decoder feeding driver.

Defines and configuration options

- #define DECFEED_CFG_DECODER_VARIANT

  The desired decoder variant to be supported. Currently supported options are: DECODER_JPEG, DECODER_H264.
- #define DECFEED_CFG_INIT_CHECK

  Enable/Disable module initialization check. If enabled (TRUE) API verifies that module already has been initialized and thus requested call can be executed.
- #define DEC_FEED_CFG_MAX_STREAMS

  Number of streams being supported by the Decoder Feeding module.

Functions

- dec_feed_ret_t DECFEED_CheckConfig (void)

  Function checks module configuration for typical errors.
- dec_feed_ret_t DECFEED_SetBaseAddress (uint32_t u32Addr)

  Configure base address of decoder register region.
- dec_feed_ret_t DECFEED_SetRegionLength (uint32_t u32Length)

  Configure length of decoder register region.
- dec_feed_ret_t DECFEED_GetMemoryBase (uint32_t ∗pu32Base, uint32_t ∗pu32Length)

  Read configured base address and length of decoder register region.
- dec_feed_ret_t DECFEED_SetBurstLength (uint8_t u8Count)

  Configure Burst Length.
- dec_feed_ret_t DECFEED_GetBurstLength (uint8_t ∗pu8Count)

  Read configured Burst Length.
- dec_feed_ret_t DECFEED_SetNumOfStreams (uint8_t u8NumberOfStreams)

  Configure number of streams to work with.
- dec_feed_ret_t DECFEED_GetDecoderStatus (uint32_t ∗pu32StatusReg)

  Get current error status of decoder hardware.
- dec_feed_ret_t DECFEED_GetFreeSpace (uint8_t u8StreamIdx, uint32_t ∗pu32FreeEntriesNum)

  Gets number of buffers which may be passed to the decoder.
- dec_feed_ret_t DECFEED_HandleProcessedFrms (uint8_t u8StreamIdx, uint32_t ∗pu32ProcessedNum)

  Processing of number of processed frames.
- dec_feed_ret_t DECFEED_Push (uint8_t u8StreamIdx, uint32_t u32Addr, uint16_t u16Length)

  Push one buffer to decoder HW queue.
- dec_feed_ret_t DECFEED_Init (void)

  Module initialization.
- dec_feed_ret_t DECFEED_Start (void)

  Start feeding decoder.
- dec_feed_ret_t DECFEED_Stop (void)

  Stop feeding decoder.

© 2016 NXP Semiconductors

4.1.2 Function Documentation

4.1.2.1 dec_feed_ret_t DECFEED_CheckConfig (void)

Function checks module configuration for typical errors.

© 2016 NXP Semiconductors

Return values

DECFEED_E_INVALID_CONFIG	Error - At least one parameter has wrong value.
DECFEED_E_OK	Success.

4.1.2.2 dec_feed_ret_t DECFEED_SetBaseAddress (uint32_t *u32Addr*)

Configure base address of decoder register region.

Parameters

in	*u32Addr*	Physical base address.

Return values

DECFEED_E_MULTIPLE_CONFIG	Error - attempt to change the address.
DECFEED_E_OK	Success.

Note

Once the address is set, it may not be changed.

4.1.2.3 dec_feed_ret_t DECFEED_SetRegionLength (uint32_t *u32Length*)

Configure length of decoder register region.

Parameters

in	*u32Length*	Length in bytes.

Return values

DECFEED_E_MULTIPLE_CONFIG	Error - attempt to change the length.
DECFEED_E_OK	Success.

Note

Once the length is set, it may not be changed.

4.1.2.4 dec_feed_ret_t DECFEED_GetMemoryBase (uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Read configured base address and length of decoder register region.

It also checks validity of the configuration. Values are provided only if the check passed..

© 2016 NXP Semiconductors

Parameters

out	*pu32Base*	Register region base address will be written here.
out	*pu32Length*	Register region length will be written here.

Returns

If configuration check fails, then return value from DECFEED_CheckConfig is passed. Otherwise this function returns DECFEED_E_OK.

4.1.2.5 dec_feed_ret_t DECFEED_SetBurstLength (uint8_t *u8Count*)

Configure Burst Length.

Parameters

in	*u8Count*	Maximal number of Ethernet frames that may be passed to decoder in one application iteration.

Return values

DECFEED_E_OK	Success.

4.1.2.6 dec_feed_ret_t DECFEED_GetBurstLength (uint8_t * *pu8Count*)

Read configured Burst Length.

Parameters

in	*pu8Count*	Pointer to variable where result shall be written.

Return values

DECFEED_E_OK	Success.

4.1.2.7 dec_feed_ret_t DECFEED_SetNumOfStreams (uint8_t *u8NumberOfStreams*)

Configure number of streams to work with.

Parameters

in	*u8NumberOfStreams*	Number of streams.

© 2016 NXP Semiconductors

Return values

DECFEED_E_INVALID_CONFIG	Error - unsupported number of streams.
DECFEED_E_OK	Success.

4.1.2.8 dec_feed_ret_t DECFEED_GetDecoderStatus (uint32_t * *pu32StatusReg*)

Get current error status of decoder hardware.

This function shall be used to periodically check HW error flags. DEC_FEED module must be initialized before this function is called.

Parameters

out	*pu32StatusReg*	Hardware specific error mask will be written here. Zero means no error.

Return values

DECFEED_E_NOT_INITIALIZED	Unable to read the status due to missing configuration.
DECFEED_E_OK	Success.

4.1.2.9 dec_feed_ret_t DECFEED_GetFreeSpace (uint8_t *u8StreamIdx,* uint32_t * *pu32FreeEntriesNum*)

Gets number of buffers which may be passed to the decoder.

This function calculates free space based on number of buffers that were passed into the decoder and number of processed buffers that were freed by function DECFEED_HandleProcessedFrms.

Parameters

in	*u8StreamIdx*	Selects stream to calculate with.
out	*pu32FreeEntriesNum*	Number of free entries will be written here.

Return values

DECFEED_E_NOT_INITIALIZED	Error - module was not properly initialized.
DECFEED_E_INVALID_PARAMETER	Error - invalid stream index.
DECFEED_E_OK	Success.

4.1.2.10 dec_feed_ret_t DECFEED_HandleProcessedFrms (uint8_t *u8StreamIdx,* uint32_t * *pu32ProcessedNum*)

Processing of number of processed frames.

This function calculates number of frames that HW decoder processed since last call and updates counter of freed frames (side effect).

© 2016 NXP Semiconductors

Parameters

in	*u8StreamIdx*	Selects stream to work with.
out	*pu32ProcessedNum*	Number of buffers processed by HW decoder since last call will be written here.

Return values

DECFEED_E_NOT_INITIALIZED	Error - module was not properly initialized.
DECFEED_E_INVALID_PARAMETER	Error - invalid stream index.
DECFEED_E_FIFO_ERROR	Error - Number of processed frames seems to be negative. Unexpected state.
DECFEED_E_OK	Success.

4.1.2.11 dec_feed_ret_t DECFEED_Push (uint8_t *u8StreamIdx,* uint32_t *u32Addr,* uint16_t *u16Length*)

Push one buffer to decoder HW queue.

Parameters

in	*u8StreamIdx*	Selects stream to work with.
in	*u32Addr*	Address of the buffer.
in	*u16Length*	Length of the buffer in bytes.

Return values

DECFEED_E_NOT_INITIALIZED	Error - module was not properly initialized.
DECFEED_E_INVALID_PARAMETER	Error - invalid stream index.
DECFEED_E_FIFO_FULL	Error - There is no free space in the FIFO.
DECFEED_E_OK	Success.

4.1.2.12 dec_feed_ret_t DECFEED_Init (void)

Module initialization.

This function checks configuration of the module and resets queue counters.

Returns

On success it returns DECFEED_E_OK, otherwise return value from DECFEED_CheckConfig is passed.

Note

The decoder HW is not restarted here. It must be done from Linux side, from VSDK.

4.1.2.13 dec_feed_ret_t DECFEED_Start (void)

Start feeding decoder.

Currently this function just checks whether module is initialized.

© 2016 NXP Semiconductors

Return values

DECFEED_E_NOT_INITIALIZED	Error - module was not properly initialized.
DECFEED_E_OK	Success.

4.1.2.14 dec_feed_ret_t DECFEED_Stop (void)

Stop feeding decoder.

Currently this function just checks whether module is initialized and clears this flag, so then DECFEED_Init must be called before the module is started again.

Return values

DECFEED_E_NOT_INITIALIZED	Error - module was not properly initialized.
DECFEED_E_OK	Success.

© 2016 NXP Semiconductors

4.2 Ethernet Queue Driver

4.2.1 Detailed Description

Purpose of the Ethernet Queue module is to provide a partial data and control interface to the ENET networking peripheral within handling of a single allocated HW queue.

Architecture

The ENET peripheral contains multiple queues used for data reception. ENET driver configures, enables and takes care about the peripheral and may use various number of RX queues. This driver is aimed only to queue handling without need to manage the complete ENET peripheral. It means that this driver can be used only within an environment where some full ENET driver takes care about the HW and a single RX queue is allocated to be handled by this Ethernet Queue module.

Theory of operation

Before the module can be used it must be properly initialized by setting values of configuration macros within the eth_queue_cfg.h and providing additional runtime configuration via:

- ETHQ_SetBaseAddr()
- ETHQ_SetRegionLength()
- ETHQ_SetBufferSize()
- ETHQ_SetRingAddr()

Once configuration is properly prepared, module can be initialized by the ETHQ_Init() call.

Remarks

Since the associated HW queue has not been configured to receive any Ethernet traffic yet, user should at this point enable the reception. It is done via ETHQ_AddVlanClassification() by setting VLAN priority(ies) identifying traffic to be received into the managed queue.

Due to nature of the HW, module needs to synchronize its internal state with the HW queue to know entry indicating start of the ring. For this purpose the module provides ETHQ_PreRunIteration() API which needs to be called upon initialization periodically until the pre-run phase is finished (indicated by the ETHQ_E_OK return value). From this point the queue is ready to receive Ethernet frames.

Frame reception is done via the ETHQ_GetNextRxBDIdx() returning index of buffer descriptor of a received frame. Each received frame is locked and user shall unlock it by ETHQ_UnlockRxBD() as soon as possible to enable further reception. User should also call ETHQ_WriteRDAR() each time when some descriptor(s) have been unlocked to keep the reception running.

The ETHQ_RemoveAllVlanClassifications() can be called to stop reception by instructing the VLAN classifier to not accept any frames. To stop the module with FIFO flush and cleaning the queue the ETHQ_Stop() shall be used.

© 2016 NXP Semiconductors

Files

- file eth_queue_cfg.h

  Statical configuration of Ethernet Rx Queue driver.
- file eth_queue.h

  API of Ethernet Rx Queue driver.
- file eth_queue.c

  Ethernet Rx Queue driver.

Defines and configuration options

- #define ETHQ_CFG_RX_BD_RING_LEN

  Number of buffers and BDs in our Rx Ethernet queue.
- #define ETHQ_CFG_USED_RX_QUEUE

  Selects which Rx queue shall be used.
- #define ETHQ_CFG_PRERUN_MAX_CYCLES

  Limit maximal number of cycles of pre-run code.
- #define ETHQ_CFG_INIT_CHECK

  Enable/Disable module initialization check. If enabled (TRUE) API verifies that module has already been initialized and thus requested call can be executed. If module has not been initialized yet an error code is returned.

Functions

- eth_queue_ret_t ETHQ_CheckConfig (void)

  Checks the module configuration for typical errors.
- eth_queue_ret_t ETHQ_SetBaseAddr (uint32_t u32Addr)

  Configure base address of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetRegionLength (uint32_t u32Len)

  Configure length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_GetMemoryBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured base address and length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetBufferSize (uint32_t u32BufSize)

  Configure length of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_SetRingAddr (uint32_t u32Addr)

  Configure address of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_GetBufferBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured address and length of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_Init (void)

  Initialization of Ethernet Rx queue.
- eth_queue_ret_t ETHQ_Empty (void)

  This function is called when all Rx frames in the queue shall be discarded.
- eth_queue_ret_t ETHQ_PreRunIteration (void)

  This function sets the Rx queue and this driver into defined state.
- eth_queue_ret_t ETHQ_AddVlanClassification (uint8_t u8PCP)

  Enables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveVlanClassification (uint8_t u8PCP)

  Disables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveAllVlanClassifications (void)

  Disables all VLAN priorities in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_SuspendVlanClassifications (void)

© 2016 NXP Semiconductors

Temporarily disables whole VLAN classifier without clearing table of enabled priorities.

- eth_queue_ret_t ETHQ_ResumeVlanClassifications (void)

  Re-enables whole VLAN classifier after it was previously suspended.

- eth_queue_ret_t ETHQ_Stop (void)

  Stops reception on used queue and empties Ethernet controller FIFO.

- eth_queue_ret_t ETHQ_GetNextRxBDIdx (sint16_t *ps16Idx)

  This function returns index of next Rx BD that shall be read first.

- eth_queue_ret_t ETHQ_UnlockRxBD (uint32_t u32BDIdx)

  Unlocks given Ethernet Rx BD so it can receive another frame.

- eth_queue_ret_t ETHQ_WriteRDAR (void)

  Notifies Ethernet controller that there are new free BDs. The Ethernet controller stops if it encounters locked BD. In this case this function will make it resume.

4.2.2 Function Documentation

4.2.2.1 eth_queue_ret_t ETHQ_CheckConfig (void)

Checks the module configuration for typical errors.

Return values

ETHQ_E_INVALID_CONFIG	Error - At least one parameter has wrong value.
ETHQ_E_OK	Success.

4.2.2.2 eth_queue_ret_t ETHQ_SetBaseAddr (uint32_t *u32Addr*)

Configure base address of Ethernet controller register region.

Parameters

in	*u32Addr*	Physical base address.

Return values

ETHQ_E_MULTIPLE_CONFIG	Error - attempt to change the address.
ETHQ_E_OK	Success.

Note

Once the address is set, it may not be changed.

4.2.2.3 eth_queue_ret_t ETHQ_SetRegionLength (uint32_t *u32Len*)

Configure length of Ethernet controller register region.

© 2016 NXP Semiconductors

Parameters

in	*u32Length*	Length in bytes.

Return values

ETHQ_E_MULTIPLE_CONFIG	Error - attempt to change the length.
ETHQ_E_OK	Success.

Note

Once the length is set, it may not be changed.

4.2.2.4 eth_queue_ret_t ETHQ_GetMemoryBase (uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Read configured base address and length of Ethernet controller register region.

It also checks validity of the configuration. Values are provided only if the check passed.

Parameters

out	*pu32Base*	Register region base address will be written here.
out	*pu32Length*	Register region length will be written here.

Returns

If configuration check fails, then return value from ETHQ_CheckConfig is passed. Otherwise this function returns ETHQ_E_OK.

4.2.2.5 eth_queue_ret_t ETHQ_SetBufferSize (uint32_t *u32BufSize*)

Configure length of Ethernet Rx buffers region.

Parameters

in	*u32Length*	Length in bytes.

Return values

ETHQ_E_MULTIPLE_CONFIG	Error - attempt to change the length.
ETHQ_E_OK	Success.

Note

Once the length is set, it may not be changed.

© 2016 NXP Semiconductors

4.2.2.6 eth_queue_ret_t ETHQ_SetRingAddr (uint32_t *u32Addr*)

Configure address of Ethernet Rx buffers region.

Parameters

in	*u32Addr*	Physical address.

Return values

ETHQ_E_MULTIPLE_CONFIG	Error - attempt to change the address.
ETHQ_E_OK	Success.

Note

Once the address is set, it may not be changed.

4.2.2.7 eth_queue_ret_t ETHQ_GetBufferBase (uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Read configured address and length of Ethernet Rx buffers region.

It also checks validity of the configuration. Values are provided only if the check passed.

Parameters

out	*pu32Base*	Rx buffer region address will be written here.
out	*pu32Length*	Rx buffer region length will be written here.

Returns

If configuration check fails, then return value from ETHQ_CheckConfig is passed. Otherwise this function returns ETHQ_E_OK.

4.2.2.8 eth_queue_ret_t ETHQ_Init (void)

Initialization of Ethernet Rx queue.

Performs configuration of Rx queue in Ethernet controller. Initializes Rx buffer descriptor (BD) ring.

Return values

ETHQ_E_INVALID_CONFIG	Error - Configuration checking failed.
ETHQ_E_OK	Success.

© 2016 NXP Semiconductors

4.2.2.9 eth_queue_ret_t ETHQ_Empty (void)

This function is called when all Rx frames in the queue shall be discarded.

It just unlocks all non-empty Rx BDs. This function keeps the queue usable.

Return values

ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.
ETHQ_E_OK	Success.

4.2.2.10 eth_queue_ret_t ETHQ_PreRunIteration (void)

This function sets the Rx queue and this driver into defined state.

Note

This function requires number of Ethernet buffer descriptors to be odd.

This function shall be called periodically as long as it returns ETHQ_E_AGAIN. During several iterations it determines active Rx Buffer Descriptor. Then it needs some more iterations to empty the Ethernet FIFO, which might contain old frames. Once it is done, the function returns ETHQ_E_OK and it shall not be called again.

Return values

ETHQ_E_OK	Done
ETHQ_E_AGAIN	Keep trying
ETHQ_E_TIMEOUT	Time-out error
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

4.2.2.11 eth_queue_ret_t ETHQ_AddVlanClassification (uint8_t *u8PCP*)

Enables given VLAN priority in Rx filter of used Ethernet queue.

Up to 4 various VLAN priorities may be added. Incoming Ethernet frames with matching priority will then be received into used Ethernet queue.

Parameters

in	*u8PCP*	VLAN priority to be enabled in VLAN classifier.

Return values

ETHQ_E_OK	Done
ETHQ_E_MULTIPLE_CONFIG	Error - This value was already added.
ETHQ_E_OVERFLOW	Error - Failed to add, table is full.
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

© 2016 NXP Semiconductors

4.2.2.12 eth_queue_ret_t ETHQ_RemoveVlanClassification (uint8_t *u8PCP*)

Disables given VLAN priority in Rx filter of used Ethernet queue.

Removes a value that was previously added. Ethernet frames with given VLAN priority will no longer be received, except frames that are already in Ethernet controller FIFO.

Parameters

in	*u8PCP*	VLAN priority to be disabled in VLAN classifier.

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_FOUND	Error - There is no such priority in the table.
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

4.2.2.13 eth_queue_ret_t ETHQ_RemoveAllVlanClassifications (void)

Disables all VLAN priorities in Rx filter of used Ethernet queue.

Removes all values. Ethernet frames will no longer be received, except frames that are already in Ethernet controller FIFO.

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

4.2.2.14 eth_queue_ret_t ETHQ_SuspendVlanClassifications (void)

Temporarily disables whole VLAN classifier without clearing table of enabled priorities.

Ethernet frames will no longer be received, except frames that are already in Ethernet controller FIFO.

Return values

ETHQ_E_OK	Done.
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

Note

If this function is called with empty filter, state of the filter will be undefined after resume function will be called. After the filter is suspended, filter configuration must NOT be changed before it is resumed. Otherwise the state of the filter will be undefined.

© 2016 NXP Semiconductors

4.2.2.15 eth_queue_ret_t ETHQ_ResumeVlanClassifications (void)

Re-enables whole VLAN classifier after it was previously suspended.

Will restore state before function ETHQ_SuspendVlanClassifications was called. Frames will be received again.

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

4.2.2.16 eth_queue_ret_t ETHQ_Stop (void)

Stops reception on used queue and empties Ethernet controller FIFO.

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

Note

This function takes longer time, it busy-waits for the Ethernet FIFO to flush.

4.2.2.17 eth_queue_ret_t ETHQ_GetNextRxBDIdx (sint16_t * *ps16Idx*)

This function returns index of next Rx BD that shall be read first.

If there is at least one received frame, then it returns index of Rx BD of first of those frames and increments internal counter. Next time this function is called, BD index of another frame is returned.

Returns

It returns negative number if there is no received frame, non-negative BD index otherwise.

Note

Once the caller obtains a valid index, it is responsible for unlocking it.

4.2.2.18 eth_queue_ret_t ETHQ_UnlockRxBD (uint32_t *u32BDIdx*)

Unlocks given Ethernet Rx BD so it can receive another frame.

Parameters

in	*u32BDIdx*	Index of buffer descriptor to be unlocked.

© 2016 NXP Semiconductors

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

4.2.2.19 eth_queue_ret_t ETHQ_WriteRDAR (void)

Notifies Ethernet controller that there are new free BDs. The Ethernet controller stops if it encounters locked BD. In this case this function will make it resume.

Return values

ETHQ_E_OK	Done
ETHQ_E_NOT_INITIALIZED	Error - Function ETHQ_Init was not run or it failed.

© 2016 NXP Semiconductors

4.3 GIC Driver

4.3.1 Detailed Description

This module contains GICv2 abstraction to provide way to manage interrupts on HW level within both Interrupt Distributor and CPU Interface components of the GIC peripheral.

Remarks

Software expects that API is called on appropriate exception level with regards to the desired security level.

Files

- file gic.h

  GIC driver header file.
- file gic.c

  The GIC driver.

Functions

- void gic_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)

  Get memory region occupied by the GIC peripheral.
- void gicd_disable_all (void)

  Disable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_enable_all (void)

  Enable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_disable_grp0 (void)

  Disable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_disable_grp1 (void)

  Disable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_enable_grp0 (void)

  Enable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_enable_grp1 (void)

  Enable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_all_to_grp1 (void)

  Move all interrupts to Group1.
- void gicd_set_target (const uint32_t u32Irq, const uint32_t u32Target)

  Set interrupt target.
- void gicd_set_priority (const uint32_t u32Irq, const uint32_t u32Priority)

  Set interrupt priority.
- void gicd_set_sensitivity (const uint32_t u32Irq, const gicd_sensitivity_t eConfig)

  Set interrupt sensitivity.
- void gicd_set_group0 (const uint32_t u32Irq)

  Assign an interrupt to Group0.
- void gicd_set_group1 (const uint32_t u32Irq)

  Assign an interrupt to Group1.
- void gicd_enable (const uint32_t u32Irq)

  Enable IRQ within the distributor.
- void gicd_disable (const uint32_t u32Irq)

© 2016 NXP Semiconductors

Disable IRQ within the distributor.

- void gicc_enable_fiq (void)

  Enable FIQ signalling.
- void gicc_disable_fiq (void)

  Disable FIQ signalling.
- void gicc_disable_all (void)

  Disable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.
- void gicc_enable_all (void)

  Enable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.
- void gicd_send_sgi_to_this_core (const uint32_t u32Irq, const uint8_t u8NSATT)

  Send SGI to the current core.

4.3.2 Function Documentation

4.3.2.1 void gic_get_memory_base (uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Get memory region occupied by the GIC peripheral.

Parameters

out	*pu32Base*	Pointer to memory where base address will be stored
out	*pu32Length*	Pointer to memory where length of the region will be stored

4.3.2.2 void gicd_set_target (const uint32_t *u32Irq,* const uint32_t *u32Target*)

Set interrupt target.

Parameters

in	*u32Irq*	The IRQ number
in	*u32Target*	Target mask to be set

4.3.2.3 void gicd_set_priority (const uint32_t *u32Irq,* const uint32_t *u32Priority*)

Set interrupt priority.

Parameters

in	*u32Irq*	The IRQ number
in	*u32Priority*	Priority value to be set

4.3.2.4 void gicd_set_sensitivity (const uint32_t *u32Irq,* const gicd_sensitivity_t *eConfig*)

Set interrupt sensitivity.

© 2016 NXP Semiconductors

Parameters

in	*u32Irq*	The interrupt number
in	*eConfig*	Sensitivity configuration to be set

4.3.2.5 void gicd_set_group0 (const uint32_t *u32Irq*)

Assign an interrupt to Group0.

Parameters

in	*u32Irq*	The interrupt number

4.3.2.6 void gicd_set_group1 (const uint32_t *u32Irq*)

Assign an interrupt to Group1.

Parameters

in	*u32Irq*	The interrupt number

4.3.2.7 void gicd_enable (const uint32_t *u32Irq*)

Enable IRQ within the distributor.

Parameters

in	*u32Irq*	The interrupt number

4.3.2.8 void gicd_disable (const uint32_t *u32Irq*)

Disable IRQ within the distributor.

Parameters

in	*u32Irq*	The interrupt number

4.3.2.9 void gicd_send_sgi_to_this_core (const uint32_t *u32Irq,* const uint8_t *u8NSATT*)

Send SGI to the current core.

© 2016 NXP Semiconductors

Parameters

in	*u32Irq*	The interrupt number (SGIINTID)
in	*u8NSATT*	Security value of the SGI

© 2016 NXP Semiconductors

4.4 MMU Driver

4.4.1 Detailed Description

Purpose of the MMU module is to provide a simple interface to the virtual memory subsystem. It contains functionality enabling user to create virtual to physical memory mappings, setting various memory attributes, enabling and disabling the memory management unit and perform certain cache-related operations.

Translation parameters are configured statically and refers to 3 translation levels with the smallest granule size 4kB. Input addresses are limited to be at most 32 bits long due to translation tables size reduction, output address is always 32 bit long.

Module operation

Module expects to be properly initialized by mmu_init() before any other operation is performed.

Note

Currently only the EL3 is supported and call to mmu_init() on different exception level will cause the MMU_↩E_SYSTEM error.

When properly initialized user calls the mmu_add_mapping() to prepare translation tables according to desired setup and is responsible for correct memory segmentation. Size of the translation tables is limited by number of entries defined by MMU_CFG_ENTRY_POOL_SIZE and should be kept as small as possible to reduce the memory footprint. To prepare the attributes mask parameters for the mmu_add_mapping() one can use the mmu_get_↩attr() to properly combine chosen memory type and memory attributes as they are defined within mmu_cfg.h and mmu_mem_attr.h headers. Once all mappings are prepared the MMU can be started by mmu_start().

Translation can be stopped by mmu_stop().

Note

To ensure memory coherency after the MMU is stopped the caches should be cleaned up. User can clean caches for various memory ranges by calls of cache_d_clean_by_va_range().

Memory attributes

Each mapping requires a set of memory attributes when its being created by the mmu_add_mapping(). The helper function mmu_get_attr() accepts two parameters: memory type and attributes mask. The attributes mask can contain various attributes joined by bitwise OR operation. Currently supported memory types are configurable and provided by the mmu_cfg.h in form of macros prefixed with MTYPE_. Memory attributes are provided by mmu_↩mem_attr.h as macros prefixed with MATTR_. Example use of the mmu_get_attr() can look like this:

```
mem_attr_t theAttrMask = mmu_get_attr(MTYPE_NORMAL,
      MA_EL2_EL3_RW | MA_NON_EXEC)
```

© 2016 NXP Semiconductors

Files

- file mmu_cfg.h

  The MMU module configuration header file.
- file mmu.h

  The MMU driver header file.
- file mmu_exception.h

  The MMU exception handler header file.
- file mmu_mem_attr.h

  Memory attributes header file.
- file mmu.c

  The MMU driver.
- file mmu_exception.c

  The MMU exception handler.

Defines and configuration options

- #define MMU_CFG_ENTRY_POOL_SIZE

  Number of entries allocated to be used by translation tables.
- #define MTYPE_NORMAL

  Memory type: Normal.
- #define MTYPE_NORMAL_NC

  Memory type: Normal, not cached.
- #define MTYPE_DEVICE

  Memory type: Device.
- #define mmu_get_attr(type, attr)

  Function to prepare attribute mask combining memory type and memory attributes values.
- #define MA_EL2_EL3_RW

  Memory attribute for EL2/EL3 read/write access.
- #define MA_EL2_EL3_RO

  Memory attribute for EL2/EL3 read-only access.
- #define MA_NON_SECURE

  Memory attribute to create non-secure mapping.
- #define MA_NON_EXEC

  Memory attribute to create non-executable mapping.

Functions

- mmu_ret_t mmu_init (void)

  Initialization function.
- mmu_ret_t mmu_add_mapping (const va_t VA, const pa_t PA, const mlen_t Size, const mem_attr_t Attr)

  API to install new mapping.
- mmu_ret_t mmu_check_mapping (const va_t VA, const pa_t PA, const mlen_t Size)

  API to check status of a mapping.
- mmu_ret_t mmu_start (void)

  Start MMU within the EL3.
- mmu_ret_t mmu_stop (void)

  Disable MMU within the EL3.
- mmu_ret_t mmu_get_region_size (const uint32_t u32Level, mlen_t ∗const RegionSize)

  Get mapping region size at given translation level.
- void cache_d_clean_by_va_range (va_t VA, mlen_t length)

  Clean data cache by address to Point of Coherency.
- void mmu_exception_handler (uint32_t u32esr_elx)

  The exception handler.

© 2016 NXP Semiconductors

4.4.2 Function Documentation

4.4.2.1 mmu_ret_t mmu_init (void)

Initialization function.

Return values

MMU_E_OK	Success
MMU_E_SYSTEM	System error
MMU_E_CONFIGURATION	Wrong configuration

4.4.2.2 mmu_ret_t mmu_add_mapping (const va_t *VA*, const pa_t *PA*, const mlen_t *Size*, const mem_attr_t *Attr*)

API to install new mapping.

Parameters

in	*VA*	Virtual address
in	*PA*	Physical address
in	*Size*	Length of mapping
in	*Attr*	Attributes

Return values

MMU_E_OK	Success
MMU_E_CONFLICT	Mapping conflict (output address or attributes)
MMU_E_INVALID_DESCRIPTOR	Invalid descriptor detected
MMU_E_CONFIGURATION	Wrong configuration

4.4.2.3 mmu_ret_t mmu_check_mapping (const va_t *VA*, const pa_t *PA*, const mlen_t *Size*)

API to check status of a mapping.

Parameters

in	*VA*	Start of the input range to be checked
in	*PA*	Output range start
in	*Size*	Size of the range

Return values

MMU_E_OK	Given VA range already mapped to the given PA range
MMU_E_NOT_MAPPED	Range not mapped yet

© 2016 NXP Semiconductors

Return values

MMU_E_CONFLICT	Given VA range or a part of the given VA range is mapped to different PA range
MMU_E_INVALID_DESCRIPTOR	Invalid descriptor detected

4.4.2.4 mmu_ret_t mmu_start (void)

Start MMU within the EL3.

Return values

MMU_E_OK	Success
MMU_E_NOT_INITIALIZED	Not initialized yet

4.4.2.5 mmu_ret_t mmu_stop (void)

Disable MMU within the EL3.

Also disables caches

Return values

MMU_E_OK	Success

4.4.2.6 mmu_ret_t mmu_get_region_size (const uint32_t *u32Level,* mlen_t *const *RegionSize*)

Get mapping region size at given translation level.

Parameters

`in`	*u32Level*	The desired translation level
`out`	*RegionSize*	Pointer to memory where the size will be written

Return values

MMU_E_OK	Success
MMU_E_INVALID_PARAMETER	Invalid input parameter

4.4.2.7 void cache_d_clean_by_va_range (va_t *VA,* mlen_t *length*)

Clean data cache by address to Point of Coherency.

© 2016 NXP Semiconductors

Parameters

in	*VA*	Start of the virtual address range to be processed
in	*length*	Length of the range

4.4.2.8 void mmu_exception_handler (uint32_t *u32esr_elx*)

The exception handler.

Function in intended to process MMU faults and shall be hooked to the sync exception handler. Note that current implementation is just a stub and can be used for debugging purposes only.

Parameters

in	*u32esr_elx*	The current ESR_ELx value mmu_exception_c_REF_3 MISRA rule 8.10

© 2016 NXP Semiconductors

4.5 PIT Driver

4.5.1 Detailed Description

This is driver for the Periodic Interrupt Timer peripheral enabling various PIT channels control and configurations. Package also contains API for getting elapsed time (useful for measurement of short time intervals) and is able to determine the PIT frequency considering current platform, PLL setup and CPU mode so the user does not need to care about that.

Files

- file pit.h

  PIT driver header file.
- file pit.c

  The PIT driver.

Functions

- void pit_get_memory_base (const uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)

  Get memory region occupied by the PIT peripheral.
- void mc_me_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)

  Get memory region occupied by the MC_ME peripheral.
- void mc_cgm_get_memory_base (uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)

  Get memory region occupied by the MC_CGM peripheral.
- pit_ret_t pit_start (const uint8_t u8Instance, const uint8_t u8Channel)

  Start the PIT.
- pit_ret_t pit_stop (const uint8_t u8Instance, const uint8_t u8Channel)

  Stop PIT.
- pit_ret_t pit_confirm_irq (const uint8_t u8Instance, const uint8_t u8Channel)

  Confirm interrupt.
- pit_ret_t pit_set_period (const uint8_t u8Instance, const uint8_t u8Channel, const uint32_t u32PeriodNs)

  Set up timer period.
- pit_ret_t pit_enable_irq (const uint8_t u8Instance, const uint8_t u8Channel)

  Enable interrupt request generation.
- pit_ret_t pit_disable_irq (const uint8_t u8Instance, const uint8_t u8Channel)

  Disable interrupt request generation.
- pit_ret_t pit_is_timeout (const uint8_t u8Instance, const uint8_t u8Channel)

  Check if timeout has occurred.
- pit_ret_t pit_get_elapsed_ns (const uint8_t u8Instance, const uint8_t u8Channel, uint32_t *const pu32Result)

  Get number of ns since timer has been started.

4.5.2 Function Documentation

4.5.2.1 void pit_get_memory_base (const uint8_t *u8Instance*, uint32_t * *pu32Base*, uint32_t * *pu32Length*)

Get memory region occupied by the PIT peripheral.

© 2016 NXP Semiconductors

Parameters

in	*u8Instance*	The desired PIT instance
out	*pu32Base*	Pointer to memory where base address will be stored
out	*pu32Length*	Pointer to memory where length of the region will be stored

4.5.2.2 void mc_me_get_memory_base (uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Get memory region occupied by the MC_ME peripheral.

Parameters

out	*pu32Base*	Pointer to memory where base address will be stored
out	*pu32Length*	Pointer to memory where length of the region will be stored

4.5.2.3 void mc_cgm_get_memory_base (uint8_t *u8Instance,* uint32_t * *pu32Base,* uint32_t * *pu32Length*)

Get memory region occupied by the MC_CGM peripheral.

Parameters

in	*u8Instance*	The desired MC_CGM instance
out	*pu32Base*	Pointer to memory where base address will be stored
out	*pu32Length*	Pointer to memory where length of the region will be stored

4.5.2.4 pit_ret_t pit_start (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Start the PIT.

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel within the instance to be started

Return values

PIT_E_OK	Success
PIT_E_PARAMETER	Invalid configuration

4.5.2.5 pit_ret_t pit_stop (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Stop PIT.

© 2016 NXP Semiconductors

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be stopped

Return values

PIT_E_OK	Success
PIT_E_PARAMETER	Invalid configuration

4.5.2.6 pit_ret_t pit_confirm_irq (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Confirm interrupt.

Function is intended to be called each time the PIT interrupt has occurred to clear the interrupt flag.

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to confirm interrupt for

Return values

PIT_E_OK	Success
PIT_E_PARAMETER	Invalid configuration

4.5.2.7 pit_ret_t pit_set_period (const uint8_t *u8Instance,* const uint8_t *u8Channel,* const uint32_t *u32PeriodNs*)

Set up timer period.

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be configured
in	*u32PeriodNs*	New timer period in ns

Return values

PIT_E_OK	Success, no timeout
PIT_E_PARAMETER	Invalid configuration

4.5.2.8 pit_ret_t pit_enable_irq (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Enable interrupt request generation.

© 2016 NXP Semiconductors

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be enabled to generate IRQ

Return values

PIT_E_OK	Success
PIT_E_PARAMETER	Invalid configuration

4.5.2.9 pit_ret_t pit_disable_irq (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Disable interrupt request generation.

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be disabled to generate IRQ

Return values

PIT_E_OK	Success
PIT_E_PARAMETER	Invalid configuration

4.5.2.10 pit_ret_t pit_is_timeout (const uint8_t *u8Instance,* const uint8_t *u8Channel*)

Check if timeout has occurred.

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be checked

Return values

PIT_E_OK	Success, no timeout
PIT_E_TIMEOUT	Timeout has occurred
PIT_E_PARAMETER	Invalid configuration

4.5.2.11 pit_ret_t pit_get_elapsed_ns (const uint8_t *u8Instance,* const uint8_t *u8Channel,* uint32_t *const *pu32Result*)

Get number of ns since timer has been started.

© 2016 NXP Semiconductors

Parameters

in	*u8Instance*	The desired PIT instance
in	*u8Channel*	Channel to be checked
out	*pu32Result*	Pointer to memory where number of nanoseconds will be written

Return values

PIT_E_OK	Success, no timeout
PIT_E_TIMEOUT	Timeout has occurred, result is not accurate
PIT_E_PARAMETER	Invalid configuration

© 2016 NXP Semiconductors

4.6 SM Core

4.6.1 Detailed Description

This module is intended to be a software platform operating on the highest exception level of the CPU and providing all the basic functionality like memory mapping preparation, stack initialization, installation of vector tables and exception handlers as well as providing linker script describing the memory. Module prepares suitable environment and provides API for user's application(s) which can run hooked to the core.

List of provided services includes following items:

- provides vector table (sm_vector_table.S) as an internal component intended to be installed by a boot-loader SW into specified memory location before module can be used
- performs initial, one-time firmware initialization of the environment and user's applications
- provides synchronous exception dispatcher and routes system calls to dedicated user's applications handlers
- provides FIQ/vFIQ exception dispatcher and routes interrupt requests to dedicated user's applications handlers
- provides mechanism of data exchange with SW running on different exception level(s) in form of additional system call parameters
- provides mechanism of asynchronous notifications where hosted user's applications can send messages to SW running on different exception levels
- provides text debug output accessible by user's applications and directed to shared memory which can be read by applications on different exception levels/contexts

The user's application life cycle

An user's application is linked with the SM core using the SM_ENTRY_POINT macro provided by sm.h where parameter is the user's application entry point function. This function is then called single time once the SM Core has started and can contain various one-time initializations.

User's application can then hook-in various SMC system call handlers as well as FIQ handlers using dedicated API (sm_install_sc_handler() or sm_install_fiq_handler()). Handlers are then called each time when appropriate system call or interrupt request will reach the exception level the SM Core is running in.

Note

Currently there is no shut-down procedure or way how to uninstall already installed handlers.

System calls and data exchange format

Module expects system calls done via the SMC instruction with ISS equal to the system call identifier. System call identifiers are used by the dispatcher code to route the SC to appropriate handlers if some are installed. Since the argument of the SMC instruction is used to carry that identifier, data exchange is done using CPU registers X0 and X1. Invoker of the system call shall therefore store current values of X0 and X1 and write the registers by values passed to the SM Core before the SMC instruction is executed:

```
...
stp x0, x1, [sp, #-16]!
smc #0x1002
str x0, x15
str x1, x16
ldp x0, x1, [sp], #16
...
```

Register values are then accessible during the SC via the sm_sc_get_params() API in form of the sm_sc_param_t structure where are mapped to its members as: u64Val0 = X0, u64Val1 = X1. Each system call returns its status to the invoker within the X0 register and if an hosted application needs to send some additional data back, it is written into the X1. The SC invoker then should read both registers, check return values and recover original values.

© 2016 NXP Semiconductors

The asynchronous notification feature

The SM Core module provides API (sm_send_async_msg()) enabling user's applications to send asynchronous signals and data to SW on different exception levels. The mechanism uses software generated interrupts and the approach is fully configurable within the sm_cfg.h as definition of following items:

```
sm_cfg_enable_notification()
sm_cfg_send_notification()
sm_cfg_confirm_notification()
sm_cfg_disable_notification()
```

By default the feature is disabled. If a SW on different exception level or within different context wants to receive asynchronous notifications from the SM Core environment it needs to enable this feature by performing the SM↩ _CFG_LL_ENABLE_EVENT_KEY system call with a non-zero X0 value. From this point each call of sm_send↩ _async_msg() within the SM Core environment will trigger the notification as defined by configuration (mostly an interrupt).

If the feature is enabled the receiving SW shall be prepared to receive such notifications and contain related interrupt handler. If the notification is received the handler must confirm the message by performing the SM_CFG_LL_CON↩ FIRM_EVENT_KEY system call. The call does not need additional parameters but returns the message arguments in X0 and X1 registers as: X0 = Reason, X1 = UsersValue.

Once the notification is not needed is shall be disabled by invoking the SM_CFG_LL_ENABLE_EVENT_KEY system call with X0 equal to zero.

Integration notes

- Before the module can operate, it must be properly written to memory location defined by the linker script.
- Once the module is in the memory and the module's target exception level is EL3 then shall be ensured that the SMC instruction is accessible and executable from lower exception levels considering sending system calls to the module.
- First system call performed to reach the SM Core is expected to be the one identified by the SM_CFG_LL↩ _INIT_KEY. This call will also return the address (X1) and size (X0) of the text debug buffer.
- For proper shut down of the SM Core module user needs to perform the SM_CFG_LL_SHUTDOWN_KEY system call and check X0 value if success (zero).

  Warning

  Re-initialization attempt without shutting the SM core down may lead to MMU faults.

Files

- file dbgb_output_cfg.h

  Configuration file for the dbgb_output.c.
- file sm_cfg.h

  The SM core configuration header file.
- file dbgb_output.h

  Header for the dbgb_output.c.
- file mmap.h

  The memory map header file.
- file sm.h

  The SM core header file.
- file dbgb_output.c

  Debug output buffer feeding module.
- file sm.c

  The SM core.
- file sm_mmap.c

  The SM core memory management abstraction and configuration module.

© 2016 NXP Semiconductors

Defines and configuration options

- #define DBGB_CFG_BUF_SIZE

  Configures size of the debug buffer.
- #define SM_CFG_SC_H_COUNT

  Maximum number of SC handlers.
- #define SM_CFG_FIQ_H_COUNT

  Maximum number of FIQ handlers.
- #define SM_CFG_ENABLE_MMU

  If TRUE then MMU is engaged @ EL3.
- #define SM_CFG_LL_INIT_KEY 0xffffU

  SC identifier of the low-level initialization status request.
- #define SM_CFG_LL_CONFIRM_EVENT_KEY 0xfffeU

  SC identifier of the event confirmation.
- #define SM_CFG_LL_ENABLE_EVENT_KEY 0xfffdU

  SC identifier of the enable/disable request.
- #define SM_CFG_LL_SHUTDOWN_KEY 0xfffcU

  SC identifier of the SM code shut-down request.
- #define sm_cfg_send_notification()

  Called when module needs to send asynchronous notification.
- #define sm_cfg_confirm_notification()

  Called by the SM Core module to confirm notification.
- #define sm_cfg_enable_notification()

  Called by the SM Core module when asynchronous notification feature shall be enabled.
- #define sm_cfg_disable_notification()

  Called by the SM Core module to disable the notification.

Functions

- sint32_t sm_install_sc_handler (const uint32_t u32Key, const sm_handler_t pfHandler)

  Install system call handler.
- sint32_t sm_install_fiq_handler (const uint32_t u32IrqID, const sm_handler_t pfHandler)

  Install FIQ handler.
- sm_sc_param_t * sm_sc_get_params (void)

  Retrieve additional system call parameters.
- void sm_sc_set_query_result (uint64_t u64Result)

  Set additional system call return value.
- void sm_send_async_msg (uint64_t u64Reason, uint64_t u64UserVal)

  Send asynchronous message to the current core.

4.6.2 Function Documentation

4.6.2.1 sint32_t sm_install_sc_handler (const uint32_t *u32Key,* const sm_handler_t *pfHandler*)

Install system call handler.

API to enable user's application to install its own system call handler

© 2016 NXP Semiconductors

Parameters

in	*u32Key*	The SC identifier
in	*pfHandler*	Pointer to the handler

Return values

SM_E_OK	Success
SM_E_FAILURE	Failure

4.6.2.2 sint32_t sm_install_fiq_handler (const uint32_t *u32IrqID,* const sm_handler_t *pfHandler*)

Install FIQ handler.

API to enable user's application to install its own FIQ handler

Parameters

in	*u32IrqID*	The FIQ identifier
in	*pfHandler*	Pointer to the handler

Return values

SM_E_OK	Success
SM_E_FAILURE	Failure

4.6.2.3 sm_sc_param_t * sm_sc_get_params (void)

Retrieve additional system call parameters.

Additional parameters of a system call can be passed to the SM Core via dedicated registers at the time the system call is performed. This function is intended to get those and provide them to the user's application.

Returns

Pointer to structure containing the parameters

4.6.2.4 void sm_sc_set_query_result (uint64_t *u64Result*)

Set additional system call return value.

An user's application may need to return custom value at the end of a system call (for example when the system call invoker needs to get some user's application specific value). This function provides way how to set up the additional return value to be accessible by the current system call invoker.

© 2016 NXP Semiconductors

Parameters

in	*u64Result*	Value to be returned to the system call invoker when the system call is finished

4.6.2.5 void sm_send_async_msg (uint64_t *u64Reason,* uint64_t *u64UserVal*)

Send asynchronous message to the current core.

This is the way how an user's application can notify current core that some event has occurred or when just needs to send some data to SW at different exception level. The receiving environment shall be prepared to receive such message according to chosen notification approach (if SM Core is configured to send asynchronous messages in form of interrupt the receiving SW needs to contain related interrupt handler; please see the sm_cfg_send_↩notification() implementation within the sm_cfg.h).

Parameters

in	*u64Reason*	Value interpreted as reason. Will be accessible via the SM_CFG_LL_CONFIRM_EVENT system call.
in	*u64UserVal*	Some user's value. Will be accessible via the SM_CFG_LL_CONFIRM_EVENT too.

© 2016 NXP Semiconductors

4.7 Streaming Core

4.7.1 Detailed Description

The Streaming Core module forms a VideoListener application core connecting all underlying modules and provides all the streaming functionality of the VideoListener firmware. Main purpose is to create a bridge between networking peripheral and a chosen hardware decoder to transfer received video data belonging to a logical stream to the input of HW decoder peripheral.

Module uses the Ethernet Queue Driver to access received data, Decoder Feeding Driver to communicate with decoder and MMU Driver to access and configure memory subsystem. It is state-based SW optimized to be run within a periodically executed context.

Theory of operation

Basic configuration of the module is done via the stream_core_cfg.h header which provides configuration options allowing to change stream type or size of internal Ethernet frames storage. When the static configuration is done, user needs to provide the runtime parameters using the SCORE_SetConf() API. Note that many of runtime parameters are required and without setting all the values the module can't be started. Following parameter values need to be set up via the SCORE_SetConf():

- APP_KEY_STRM_FETCH_THRESHOLD
- APP_KEY_STRM_SOI_MARK_VALUE
- APP_KEY_STRM_SOI_MARK_OFFSET
- APP_KEY_STRM_SOI_MARK_MASK
- APP_KEY_STRM_SEQ_NUM_OFFSET
- APP_KEY_STRM_STRM_ID_OFFSET
- APP_KEY_STRM_NUMBER_OF_STRMS
- APP_KEY_STRM_FRM_DATA_OFFSET
- APP_KEY_STRM_STREAM_ID_0
- APP_KEY_STRM_STREAM_ID_1
- APP_KEY_STRM_STREAM_ID_2
- APP_KEY_STRM_STREAM_ID_3
- APP_KEY_STRM_VLAN_PRIO_ADD
- APP_KEY_STRM_DROP_OUT_THRESHOLD
- APP_KEY_ETHQ_BASE_ADDR
- APP_KEY_ETHQ_REGION_LENGTH
- APP_KEY_ETHQ_SIZE_OF_BUFF
- APP_KEY_ETHQ_BUFF_RING_PTR
- APP_KEY_DEC_BASE_ADDR
- APP_KEY_DEC_REGION_LENGTH

© 2016 NXP Semiconductors

Once configured, module must be initialized by the SCORE_Init(). The initialization function checks configuration and returns values different from SCORE_E_OK if an error has been detected. In such case the configuration must be completed/updated until initialization is executed correctly.

Module is started by SCORE_Start(). The call changes state of the module which enters the SCORE_ST_PRERUN state. From now the module is driven by execution of the SCORE_Iteration() API. This function is intended to by the body of the module and user shall ensure its periodic calls within a non-preemptive context.

Runtime errors are stored internally and are accessible via SCORE_GetCurrentState() and SCORE_GetError↩Mask() API. Both function can be called after each iteration (the SCORE_Iteration() call) to verify if no runtime error has occurred.

Files

- file stream_core_cfg.h

  The StreamCore module configuration file.
- file h264_proc.h

  H264 stream processor header file.
- file stream_core.h

  API of core of the Video Listener Streaming Application running in FIQ.
- file h264_proc.c

  H264 stream processor.
- file stream_core.c

  Core of the Video Listener Streaming Application running in FIQ.
- file application_cfg.h

  Static part of Video Listener streaming application configuration.

Defines and configuration options

- #define SCORE_CFG_STREAM_TYPE

  Type of stream to be processed. Supported values are: H264_OVER_AVB, JPEG_OVER_AVB.
- #define SCORE_PREPROCESS_H264_STREAM

  If TRUE then stream preprocessing is engaged to prepare compatible input of the decoder HW. Only valid for H264↩_OVER_AVB stream type.
- #define SCORE_CFG_STREAM_BD_RING_LEN

  Buffer for BDs for each stream.
- #define APP_KEY_STRM_FETCH_THRESHOLD

  Maximum number of frames stored after SOI is detected in FETCH state. If state is not changed then the stream buffer is freed.
- #define APP_KEY_STRM_SOI_MARK_VALUE

  Specification of the StartOfImage marker. This value is being compared with the one within the Ethernet frames to determine start of image. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SOI_MARK_OFFSET

  Offset of the StartOfImage marker within Ethernet frame. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SOI_MARK_MASK

  Mask of the StartOfImage marker within the 32 bits. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SEQ_NUM_OFFSET

  Offset of the Sequence Number within Ethernet frame.
- #define APP_KEY_STRM_STRM_ID_OFFSET

  Offset of the Stream ID within Ethernet frame.

© 2016 NXP Semiconductors

- #define APP_KEY_STRM_NUMBER_OF_STRMS

  Requested number of streams.
- #define APP_KEY_STRM_FRM_DATA_OFFSET

  Offset of the Frame Data (payload) within Ethernet frame. If AVB is used as transport protocol this is the AVB payload offset.
- #define APP_KEY_STRM_VLAN_PRIO_CLEAN

  Request to clear all VLAN entries.
- #define APP_KEY_STRM_VLAN_PRIO_ADD

  Request to add VLAN priority value to identify traffic to be received as a video stream. Frames without VLAN tag or with VLAN priority value not equal to this added value will not be accepted.
- #define APP_KEY_STRM_DROP_OUT_THRESHOLD

  Number of iterations to tolerate while no stream data is being received. Used to detect stream drop-outs.
- #define APP_KEY_STRM_STREAM_ID_0

  Stream ID value of AVB frames identifying packets for logical stream number 0.
- #define APP_KEY_STRM_STREAM_ID_1

  Stream ID value of AVB frames identifying packets for logical stream number 1.
- #define APP_KEY_STRM_STREAM_ID_2

  Stream ID value of AVB frames identifying packets for logical stream number 2.
- #define APP_KEY_STRM_STREAM_ID_3

  Stream ID value of AVB frames identifying packets for logical stream number 3.
- #define APP_KEY_ETHQ_BASE_ADDR

  Base address of the ENET peripheral to be used by the ETHQ module.
- #define APP_KEY_ETHQ_REGION_LENGTH

  Length of the ENET peripheral registers region to be used by the ETHQ module.
- #define APP_KEY_ETHQ_SIZE_OF_BUFF

  Size of the Ethernet RX buffers region. This is the memory buffer used to store received Ethernet frames.
- #define APP_KEY_ETHQ_BUFF_RING_PTR

  Starting address of the Ethernet RX buffers memory.
- #define APP_KEY_DEC_BASE_ADDR

  Base address of the desired HW decoder peripheral to be used by the DECFEED module.
- #define APP_KEY_DEC_REGION_LENGTH

  Length of the HW decoder registers region to be used by the DECFEED module.
- #define APP_KEY_DEC_FEED_AT_ONCE

  Configuration parameter specifying number of frames to be put into the decoder's input FIFO within a single iteration. Can be used to prevent decoder feeding bursts.

Functions

- h264_proc_ret_t H264PROC_PreprocessPacket (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr, const uint16_t u16Length, uint32_t ∗const pu32ProcPacketAddr, uint16_t ∗const pu16ProcLength)

  When data stream from sensor does not match input format of the HW decoder this function may be used to prepare data for the decoder in acceptable format.
- h264_proc_ret_t H264PROC_IsStartOfFrame (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr)

  Function indicates whether current packet is a leading packet of a h264 frame.
- stream_core_ret_t SCORE_CheckConfig (void)

  Checks whether all required configuration parameters were set.
- stream_core_ret_t SCORE_Init (void)

  Initialization function of stream_core module.
- stream_core_ret_t SCORE_GetConf (uint64_t u64Var, uint64_t ∗pu64Val)

  Function for getting current values of runtime configuration parameters.
- stream_core_ret_t SCORE_SetConf (uint64_t u64Var, uint64_t u64Val)

© 2016 NXP Semiconductors

Function for setting values of runtime configuration parameters.

- stream_core_ret_t SCORE_Start (void)

  Changes stream_core state from READY to PRERUN.

- stream_core_ret_t SCORE_Stop (void)

  Stops other modules and changes stream_core state to READY.

- uint32_t SCORE_GetErrorMask (void)

  Function for getting current value of stream_core error mask.

- void SCORE_ClearErrorMask (void)

  Function for cleaning stream_core error mask.

- SCORE_tenState SCORE_GetCurrentState (void)

  Function for getting current state of stream_core.

- void SCORE_Iteration (void)

  Performs single iteration of autonomous stream processing.

4.7.2 Enumeration Type Documentation

4.7.2.1 enum SCORE_tenState

Definition of possible values of current stream_core state.

Enumerator

SCORE_ST_READY Waiting for START or CONFIGURE command

SCORE_ST_FETCH Waiting for frames with start of image/key-frame

SCORE_ST_PRERUN The pre-run phase

SCORE_ST_RUN Running

SCORE_ST_ERROR Fatal error encountered, reconfiguration needed

4.7.2.2 enum stream_core_ret_t

Definition of return values of stream_core functions.

This values are used by functions which return their error state. There are also functions that can not fail and they return void or different information.

4.7.3 Function Documentation

4.7.3.1 h264_proc_ret_t H264PROC_PreprocessPacket (const uint8_t *u8StreamIdx,* const uint32_t *u32PacketAddr,* const uint16_t *u16Length,* uint32_t ∗const *pu32ProcPacketAddr,* uint16_t ∗const *pu16ProcLength*)

When data stream from sensor does not match input format of the HW decoder this function may be used to prepare data for the decoder in acceptable format.

As stream packets are flowing in, function is sequentially modifying headers and the StreamIdx information is used to distinguish between various stream instances.

© 2016 NXP Semiconductors

Parameters

in	*u8StreamIdx*	Identifier of stream the preprocessing is being performed for
in	*u32PacketAddr*	Pointer to the original h264 packet
in	*u16Length*	Length of the original packet
in	*pu32ProcPacketAddr*	Pointer to the preprocessed h264 packet
in	*pu16ProcLength*	Length of the preprocessed packet

Return values

H264PROC_E_OK	Success
H264PROC_E_UNSUPPORTED_U↩ NIT	Unsupported NAL unit detected

4.7.3.2 h264_proc_ret_t H264PROC_IsStartOfFrame (const uint8_t *u8StreamIdx,* const uint32_t *u32PacketAddr*)

Function indicates whether current packet is a leading packet of a h264 frame.

Parameters

in	*u8StreamIdx*	The stream identifier
in	*u32PacketAddr*	Address of the h264 packet to be checked

Return values

H264PROC_E_AGAIN	Start of frame not found
H264PROC_E_OK	Start of frame found
H264PROC_E_UNSUPPORTED_U↩ NIT	Unsupported NAL unit detected

4.7.3.3 stream_core_ret_t SCORE_CheckConfig (void)

Checks whether all required configuration parameters were set.

Return values

SCORE_E_INVALID_CONFIG	Error - At least one parameter was not set.
SCORE_E_OK	Success.

4.7.3.4 stream_core_ret_t SCORE_Init (void)

Initialization function of stream_core module.

This function must be called before each call of SCORE_Start. It checks validity of configurations of stream_core, dec_feed, and eth_queue modules, applies stream_core configuration, and initializes queues and other variables in all 3 mentioned modules.

© 2016 NXP Semiconductors

Returns

If configuration check fails, then return value from SCORE_CheckConfig is passed. Otherwise this function returns either error value or on success SCORE_E_OK.

4.7.3.5 stream_core_ret_t SCORE_GetConf (uint64_t *u64Var,* uint64_t * *pu64Val*)

Function for getting current values of runtime configuration parameters.

Parameters

in	*u64Var*	Selects which parameter (which variable) shall be read.
in	*pu64Val*	Current value of selected parameter will be written here.

Return values

SCORE_E_UNKNOWN_PARAMETER	Error - parameter u64Var was not recognized. Since this function is not implemented yet, this is the only possible return value.

4.7.3.6 stream_core_ret_t SCORE_SetConf (uint64_t *u64Var,* uint64_t *u64Val*)

Function for setting values of runtime configuration parameters.

Parameters

in	*u64Var*	Selects which parameter (which variable) shall be written.
in	*u64Val*	Value to be assigned to selected parameter.

Return values

SCORE_E_UNKNOWN_PARAMETER	Error - parameter given in argument u64Var was not recognized.
SCORE_E_INVALID_CONFIG	Error - value of parameter given in argument pu64Val is not valid.
SCORE_E_OK	Parameter value was successfully set.

4.7.3.7 stream_core_ret_t SCORE_Start (void)

Changes stream_core state from READY to PRERUN.

Return values

SCORE_E_NOT_INITIALIZED	Error - function SCORE_Init shall be executed before and it shall not fail.
SCORE_E_OK	Success.

© 2016 NXP Semiconductors

4.7.3.8 stream_core_ret_t SCORE_Stop (void)

Stops other modules and changes stream_core state to READY.

Return values

SCORE_E_NOT_INITIALIZED	Error - function SCORE_Init shall be executed before and it shall not fail.
SCORE_E_OK	Success.

4.7.3.9 uint32_t SCORE_GetErrorMask (void)

Function for getting current value of stream_core error mask.

Returns

Value of stream_core error mask. Each set bit determines that corresponding kind of error or warning was detected in the stream_core during run-time.

4.7.3.10 void SCORE_ClearErrorMask (void)

Function for cleaning stream_core error mask.

This function shall be used to delete all error flags after all errors were handled by upper layers.

4.7.3.11 SCORE_tenState SCORE_GetCurrentState (void)

Function for getting current state of stream_core.

Returns

Current state of stream_core. Find possible values in documentation of enum SCORE_tenState.

4.7.3.12 void SCORE_Iteration (void)

Performs single iteration of autonomous stream processing.

Once everything is configured, SCORE_Start was successfully executed, and stream packets are coming into our Ethernet queue, stream_core is able to work autonomously. It requires just periodical calling of this function and checking stream_core state accessible through function SCORE_GetCurrentState. In case of error: main application shall be notified and calling of this function stopped.

© 2016 NXP Semiconductors

4.8 The VideoListener Firmware

4.8.1 Detailed Description

This is the final VideoListener Firmware application intended to be run out of Linux context within the highest available exception level as a hosted application of the SM Core module. It does not contain any API to be called from external environment.

Application provides an entry point for the SM Core and performs initial initialization of PIT and GIC peripherals which are then configured to be used as periodic interrupt generator to drive the application's core. The application's core consists of the Streaming Core which takes care of the main functionality and a simple error handler verifying correct operation.

Control events are passed from the SM Core in form of callbacks to start, stop and configure the application. All system calls accepted by the firmware are:

- APP_KEY_START: start the application
- APP_KEY_STOP: stop the application
- APP_KEY_SET_CFG: set a configuration parameter
- APP_KEY_GET_CFG: get a configuration parameter

and are defined within the application_cfg.h.

The APP_CFG_PIT_IRQ_ID is used to identify interrupt request number triggering the application's body and is being configured and enabled within the GIC. User shall ensure that interrupt number configured in APP_CFG_P↩IT_IRQ_ID does not conflict with any other interrupt within the system.

Application performs detection of runtime errors. In case when some is detected it triggers asynchronous notification event as implemented and described by the SM Core module which provides error reason and error flags to a possible event listener.

Integration notes

For this application applies the same assumptions as for the SM Core module. Needs to be loaded to memory specified by the current linker script.

Files

- file application.c

  Main file of the VideoListener firmware.
- file application.h

  Header of main file of application running in Secure Monitor CPU mode.

© 2016 NXP Semiconductors

4.9 The VideoListener Application Example

4.9.1 Detailed Description

This is the demonstrator of the complete VideoListener application providing video play back on the HW screen from one or more camera sensors connected with the target via Ethernet link. Application depends on the The VideoListener Firmware and the The VideoListener Firmware Driver which also needs to be present within the system. Additionally, there is a list of Vision components needed to be supplied too:

- sram.ko
- seq.ko
- oal_cma.ko
- csi.ko
- fdma.ko
- fsl_jpegdcd.ko (only if application is built with APP_CFG_DECODER_VARIANT == DECODER_JPEG)
- h264dcd.ko (only if the application is built with APP_CFG_DECODER_VARIANT == DECODER_H264)

When all required components are loaded, application can be executed. Note that user shall ensure that Ethernet traffic from the cameras is visible by the host and is properly VLAN tagged. It shall also be ensured that all VideoListener components are built with the same configuration of the decoder variant, either as JPEG or as H264.

Once started the application can be terminated by the Ctrl+c command.

Files

- file main.cpp

  This is the VideoListener video play back demonstration application.

© 2016 NXP Semiconductors

4.10 The VideoListener Firmware Driver

4.10.1 Detailed Description

This Linux driver is intended to be used as the VideoListener firmware boot-loader and interface between user-space application and the firmware. It contains the asynchronous event listener as implemented by the SM Core and provides access to the debug text memory in form of readable device file.

Module expects that the current device tree configuration contains an allocated memory region prepared to be occupied by the firmware and that the SMC instruction is accessible and executable. Both tasks are done by patching the u-boot and the Linux kernel source code by provided patches.

In time when the driver module is being insmod-ed it expects to have the firmware binary within the current directory named theA5App.bin. After successful insmod the module creates entry within the /dev folder with the name of the device: /dev/sm_drv. This can be used to read the text debug output produced by the firmware. The device file also provides ioctl interface supporting following commands:

- SM_DRV_IOCTL_INIT: Initialize the firmware
- SM_DRV_IOCTL_START: Start the firmware
- SM_DRV_IOCTL_STOP: Stop the firmware
- SM_DRV_IOCTL_SET_CFG: Set a configuration parameter
- SM_DRV_IOCTL_GET_CFG: Get a configuration parameter
- SM_DRV_IOCTL_REG_SIG: Register asynchronous signal listener
- SM_DRV_IOCTL_UNREG_SIG: Cancel registration of asynchronous signal listener
- SM_DRV_IOCTL_ENABLE_EVENTS: Enable/Disable asynchronous events reporting by the firmware

Files

- file sm_drv_types.h

  IOCTL interface definition for the firmware driver.
- file main.c

  This is the VideoListener firmware driver.

© 2016 NXP Semiconductors

Chapter 5

File Documentation

5.1 application.c File Reference

Main file of the VideoListener firmware.

```
#include "common_types.h"
#include "common.h"
#include "mmap.h"
#include "sm_cfg.h"
#include "sm.h"
#include "mmu_mem_attr.h"
#include "mmu_cfg.h"
#include "mmu.h"
#include "gic.h"
#include "pit.h"
#include "application_cfg.h"
#include "application.h"
#include "stream_core.h"
#include "debug.h"
```

5.1.1 Detailed Description

Main file of the VideoListener firmware.

Project Video Listener Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.2 application.h File Reference

Header of main file of application running in Secure Monitor CPU mode.

5.2.1 Detailed Description

Header of main file of application running in Secure Monitor CPU mode.

Project Video Listener Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.3 application_cfg.h File Reference

Static part of Video Listener streaming application configuration.

Macros

- #define APP_KEY_STRM_FETCH_THRESHOLD

  Maximum number of frames stored after SOI is detected in FETCH state. If state is not changed then the stream buffer is freed.
- #define APP_KEY_STRM_SOI_MARK_VALUE

  Specification of the StartOfImage marker. This value is being compared with the one within the Ethernet frames to determine start of image. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SOI_MARK_OFFSET

  Offset of the StartOfImage marker within Ethernet frame. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SOI_MARK_MASK

  Mask of the StartOfImage marker within the 32 bits. Value is being used for the JPEG_OVER_AVB stream type only.
- #define APP_KEY_STRM_SEQ_NUM_OFFSET

  Offset of the Sequence Number within Ethernet frame.
- #define APP_KEY_STRM_STRM_ID_OFFSET

  Offset of the Stream ID within Ethernet frame.
- #define APP_KEY_STRM_NUMBER_OF_STRMS

  Requested number of streams.
- #define APP_KEY_STRM_FRM_DATA_OFFSET

  Offset of the Frame Data (payload) within Ethernet frame. If AVB is used as transport protocol this is the AVB payload offset.
- #define APP_KEY_STRM_VLAN_PRIO_CLEAN

  Request to clear all VLAN entries.
- #define APP_KEY_STRM_VLAN_PRIO_ADD

  Request to add VLAN priority value to identify traffic to be received as a video stream. Frames without VLAN tag or with VLAN priority value not equal to this added value will not be accepted.
- #define APP_KEY_STRM_DROP_OUT_THRESHOLD

  Number of iterations to tolerate while no stream data is being received. Used to detect stream drop-outs.
- #define APP_KEY_STRM_STREAM_ID_0

  Stream ID value of AVB frames identifying packets for logical stream number 0.
- #define APP_KEY_STRM_STREAM_ID_1

  Stream ID value of AVB frames identifying packets for logical stream number 1.
- #define APP_KEY_STRM_STREAM_ID_2

  Stream ID value of AVB frames identifying packets for logical stream number 2.
- #define APP_KEY_STRM_STREAM_ID_3

Stream ID value of AVB frames identifying packets for logical stream number 3.

- #define APP_KEY_ETHQ_BASE_ADDR

  Base address of the ENET peripheral to be used by the ETHQ module.
- #define APP_KEY_ETHQ_REGION_LENGTH

  Length of the ENET peripheral registers region to be used by the ETHQ module.
- #define APP_KEY_ETHQ_SIZE_OF_BUFF

  Size of the Ethernet RX buffers region. This is the memory buffer used to store received Ethernet frames.
- #define APP_KEY_ETHQ_BUFF_RING_PTR

  Starting address of the Ethernet RX buffers memory.
- #define APP_KEY_DEC_BASE_ADDR

  Base address of the desired HW decoder peripheral to be used by the DECFEED module.
- #define APP_KEY_DEC_REGION_LENGTH

  Length of the HW decoder registers region to be used by the DECFEED module.
- #define APP_KEY_DEC_FEED_AT_ONCE

  Configuration parameter specifying number of frames to be put into the decoder's input FIFO within a single iteration. Can be used to prevent decoder feeding bursts.

5.3.1 Detailed Description

Static part of Video Listener streaming application configuration.

Project Video Listener Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.4 autolibc.c File Reference

Custom implementation of some standard functions from libc.

```
#include "common_types.h"
#include "autolibc.h"
```

5.4.1 Detailed Description

Custom implementation of some standard functions from libc.

Version

0.0.2.0

Note

Functions are safe, as far as given pointers (with respect to their lengths) point to valid memory ranges and strings (except for strncpy) are zero terminated. Also avoid arrays occupying last 4 bytes of address space (0xFFFFFFFB to 0xFFFFFFFF). Some functions (strlen, strcpy...) are in some cases reading up to 3 bytes behind terminating nul byte.

This module provides some of standard functions usually provided by a compiler library set. Module is intended to provide only functions necessary for compilation of other modules. All functions here are optimized for 32-bit PPC.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2012-2016 Freescale Semiconductor Inc. Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.5 autolibc.h File Reference

Header file for the AutoLibc.c.

5.5.1 Detailed Description

Header file for the AutoLibc.c.

Version

0.0.2.0

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2012-2016 Freescale Semiconductor Inc. Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.6 dbgb_output.c File Reference

Debug output buffer feeding module.

```
#include "common_types.h"
#include "sm_cfg.h"
#include "mmap.h"
#include "dbgb_output_cfg.h"
#include "dbgb_output.h"
```

5.6.1 Detailed Description

Debug output buffer feeding module.

This module provides an output buffer for the fsl_printf module to write the debug messages.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.7 dbgb_output.h File Reference

Header for the dbgb_output.c.

5.7.1 Detailed Description

Header for the dbgb_output.c.

Provides interface to the debug buffer

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.8 dbgb_output_cfg.h File Reference

Configuration file for the dbgb_output.c.

Macros

- #define DBGB_CFG_BUF_SIZE

  Configures size of the debug buffer.

5.8.1 Detailed Description

Configuration file for the dbgb_output.c.

Provides interfaces needed by legacy fsl_printf module

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.9 dec_feed.c File Reference

The decoder feeding driver.

```
#include "common_types.h"
#include "dec_feed_cfg.h"
#include "dec_feed.h"
```

Functions

- dec_feed_ret_t DECFEED_CheckConfig (void)

  Function checks module configuration for typical errors.
- dec_feed_ret_t DECFEED_SetBaseAddress (uint32_t u32Addr)

  Configure base address of decoder register region.
- dec_feed_ret_t DECFEED_SetRegionLength (uint32_t u32Length)

  Configure length of decoder register region.
- dec_feed_ret_t DECFEED_GetMemoryBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured base address and length of decoder register region.
- dec_feed_ret_t DECFEED_SetBurstLength (uint8_t u8Count)

  Configure Burst Length.
- dec_feed_ret_t DECFEED_GetBurstLength (uint8_t *pu8Count)

  Read configured Burst Length.
- dec_feed_ret_t DECFEED_GetDecoderStatus (uint32_t *pu32StatusReg)

  Get current error status of decoder hardware.
- dec_feed_ret_t DECFEED_GetFreeSpace (uint8_t u8StreamIdx, uint32_t *pu32FreeEntriesNum)

  Gets number of buffers which may be passed to the decoder.
- dec_feed_ret_t DECFEED_SetNumOfStreams (uint8_t u8NumberOfStreams)

  Configure number of streams to work with.
- dec_feed_ret_t DECFEED_HandleProcessedFrms (uint8_t u8StreamIdx, uint32_t *pu32ProcessedNum)

  Processing of number of processed frames.
- dec_feed_ret_t DECFEED_Push (uint8_t u8StreamIdx, uint32_t u32Addr, uint16_t u16Length)

  Push one buffer to decoder HW queue.
- dec_feed_ret_t DECFEED_Init (void)

  Module initialization.
- dec_feed_ret_t DECFEED_Start (void)

  Start feeding decoder.
- dec_feed_ret_t DECFEED_Stop (void)

  Stop feeding decoder.

5.9.1 Detailed Description

The decoder feeding driver.

This driver handles only the input of the decoder. It cooperates with Linux driver which handles the rest, mainly initialization and output of the decoder.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.10 dec_feed.h File Reference

API of the decoder driver.

Enumerations

Functions

- dec_feed_ret_t DECFEED_CheckConfig (void)
  Function checks module configuration for typical errors.
- dec_feed_ret_t DECFEED_SetBaseAddress (uint32_t u32Addr)
  Configure base address of decoder register region.
- dec_feed_ret_t DECFEED_SetRegionLength (uint32_t u32Length)
  Configure length of decoder register region.
- dec_feed_ret_t DECFEED_GetMemoryBase (uint32_t *pu32Base, uint32_t *pu32Length)
  Read configured base address and length of decoder register region.
- dec_feed_ret_t DECFEED_SetBurstLength (uint8_t u8Count)
  Configure Burst Length.
- dec_feed_ret_t DECFEED_GetBurstLength (uint8_t *pu8Count)
  Read configured Burst Length.
- dec_feed_ret_t DECFEED_SetNumOfStreams (uint8_t u8NumberOfStreams)
  Configure number of streams to work with.
- dec_feed_ret_t DECFEED_GetDecoderStatus (uint32_t *pu32StatusReg)
  Get current error status of decoder hardware.
- dec_feed_ret_t DECFEED_GetFreeSpace (uint8_t u8StreamIdx, uint32_t *pu32FreeEntriesNum)
  Gets number of buffers which may be passed to the decoder.
- dec_feed_ret_t DECFEED_HandleProcessedFrms (uint8_t u8StreamIdx, uint32_t *pu32ProcessedNum)
  Processing of number of processed frames.
- dec_feed_ret_t DECFEED_Push (uint8_t u8StreamIdx, uint32_t u32Addr, uint16_t u16Length)
  Push one buffer to decoder HW queue.
- dec_feed_ret_t DECFEED_Init (void)
  Module initialization.
- dec_feed_ret_t DECFEED_Start (void)
  Start feeding decoder.
- dec_feed_ret_t DECFEED_Stop (void)
  Stop feeding decoder.

5.10.1 Detailed Description

API of the decoder driver.

This driver handles only the input of the decoder. It cooperates with Linux driver which handles the rest, mainly output of the decoder.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.11 dec_feed_cfg.h File Reference

Static configuration of video decoder driver.

Macros

- #define DECFEED_CFG_DECODER_VARIANT

  The desired decoder variant to be supported. Currently supported options are: DECODER_JPEG, DECODER_H264.
- #define DECFEED_CFG_INIT_CHECK

  Enable/Disable module initialization check. If enabled (TRUE) API verifies that module already has been initialized and thus requested call can be executed.
- #define DEC_FEED_CFG_MAX_STREAMS

  Number of streams being supported by the Decoder Feeding module.

5.11.1 Detailed Description

Static configuration of video decoder driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.12 eth_queue.c File Reference

Ethernet Rx Queue driver.

```
#include "common_types.h"
#include "common.h"
#include "sm_cfg.h"
#include "mmap.h"
#include "eth_queue_cfg.h"
#include "eth_queue.h"
#include "autolibc.h"
```

Functions

- eth_queue_ret_t ETHQ_CheckConfig (void)

  Checks the module configuration for typical errors.
- eth_queue_ret_t ETHQ_GetNextRxBDIdx (sint16_t *ps16Idx)

  This function returns index of next Rx BD that shall be read first.
- eth_queue_ret_t ETHQ_GetMemoryBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured base address and length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetBaseAddr (uint32_t u32Addr)

  Configure base address of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetRegionLength (uint32_t u32Len)

  Configure length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_GetBufferBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured address and length of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_SetBufferSize (uint32_t u32BufSize)

Configure length of Ethernet Rx buffers region.

- eth_queue_ret_t ETHQ_SetRingAddr (uint32_t u32Addr)

  Configure address of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_Init (void)

  Initialization of Ethernet Rx queue.
- eth_queue_ret_t ETHQ_Empty (void)

  This function is called when all Rx frames in the queue shall be discarded.
- eth_queue_ret_t ETHQ_PreRunIteration (void)

  This function sets the Rx queue and this driver into defined state.
- eth_queue_ret_t ETHQ_AddVlanClassification (uint8_t u8PCP)

  Enables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveVlanClassification (uint8_t u8PCP)

  Disables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveAllVlanClassifications (void)

  Disables all VLAN priorities in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_SuspendVlanClassifications (void)

  Temporarily disables whole VLAN classifier without clearing table of enabled priorities.
- eth_queue_ret_t ETHQ_ResumeVlanClassifications (void)

  Re-enables whole VLAN classifier after it was previously suspended.
- eth_queue_ret_t ETHQ_Stop (void)

  Stops reception on used queue and empties Ethernet controller FIFO.
- eth_queue_ret_t ETHQ_UnlockRxBD (uint32_t u32BDIdx)

  Unlocks given Ethernet Rx BD so it can receive another frame.
- eth_queue_ret_t ETHQ_WriteRDAR (void)

  Notifies Ethernet controller that there are new free BDs. The Ethernet controller stops if it encounters locked BD. In this case this function will make it resume.

Variables

- uint8_t ETHQ_aau8BDRing [ETHQ_CFG_RX_BD_RING_LEN][ETHQ_RX_BD_LENGTH]

5.12.1 Detailed Description

Ethernet Rx Queue driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.13 eth_queue.h File Reference

API of Ethernet Rx Queue driver.

Enumerations

Functions

- eth_queue_ret_t ETHQ_CheckConfig (void)

  Checks the module configuration for typical errors.
- eth_queue_ret_t ETHQ_SetBaseAddr (uint32_t u32Addr)

  Configure base address of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetRegionLength (uint32_t u32Len)

  Configure length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_GetMemoryBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured base address and length of Ethernet controller register region.
- eth_queue_ret_t ETHQ_SetBufferSize (uint32_t u32BufSize)

  Configure length of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_SetRingAddr (uint32_t u32Addr)

  Configure address of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_GetBufferBase (uint32_t *pu32Base, uint32_t *pu32Length)

  Read configured address and length of Ethernet Rx buffers region.
- eth_queue_ret_t ETHQ_Init (void)

  Initialization of Ethernet Rx queue.
- eth_queue_ret_t ETHQ_Empty (void)

  This function is called when all Rx frames in the queue shall be discarded.
- eth_queue_ret_t ETHQ_PreRunIteration (void)

  This function sets the Rx queue and this driver into defined state.
- eth_queue_ret_t ETHQ_AddVlanClassification (uint8_t u8PCP)

  Enables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveVlanClassification (uint8_t u8PCP)

  Disables given VLAN priority in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_RemoveAllVlanClassifications (void)

  Disables all VLAN priorities in Rx filter of used Ethernet queue.
- eth_queue_ret_t ETHQ_SuspendVlanClassifications (void)

  Temporarily disables whole VLAN classifier without clearing table of enabled priorities.
- eth_queue_ret_t ETHQ_ResumeVlanClassifications (void)

  Re-enables whole VLAN classifier after it was previously suspended.
- eth_queue_ret_t ETHQ_Stop (void)

  Stops reception on used queue and empties Ethernet controller FIFO.
- eth_queue_ret_t ETHQ_GetNextRxBDIdx (sint16_t *ps16Idx)

  This function returns index of next Rx BD that shall be read first.
- eth_queue_ret_t ETHQ_UnlockRxBD (uint32_t u32BDIdx)

  Unlocks given Ethernet Rx BD so it can receive another frame.
- eth_queue_ret_t ETHQ_WriteRDAR (void)

  Notifies Ethernet controller that there are new free BDs. The Ethernet controller stops if it encounters locked BD. In this case this function will make it resume.

Variables

- uint8_t ETHQ_aau8BDRing [ETHQ_CFG_RX_BD_RING_LEN][ETHQ_RX_BD_LENGTH]

© 2016 NXP Semiconductors

5.13.1 Detailed Description

API of Ethernet Rx Queue driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.14 eth_queue_cfg.h File Reference

Statical configuration of Ethernet Rx Queue driver.

Macros

- #define ETHQ_CFG_RX_BD_RING_LEN

  Number of buffers and BDs in our Rx Ethernet queue.
- #define ETHQ_CFG_USED_RX_QUEUE

  Selects which Rx queue shall be used.
- #define ETHQ_CFG_PRERUN_MAX_CYCLES

  Limit maximal number of cycles of pre-run code.
- #define ETHQ_CFG_INIT_CHECK

  Enable/Disable module initialization check. If enabled (TRUE) API verifies that module has already been initialized and thus requested call can be executed. If module has not been initialized yet an error code is returned.

5.14.1 Detailed Description

Statical configuration of Ethernet Rx Queue driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.15 fsl_printf.c File Reference

Module serves to printing debug messages.

```
#include "common_types.h"
#include "fsl_printf.h"
#include "fsl_printf_cfg.h"
#include <stdarg.h>
```

Functions

- void fsl_printf_init (void)

  Not needed at the moment.
- void fsl_printf (const char_t ∗pcocStr,...)

  Print data to the interface selected in configuration.

5.15.1 Detailed Description

Module serves to printing debug messages.

Version

0.0.0.0

Project Video Listener Platform S32V234

SWVersion 0.9.0

Copyright (c) 2014-2016 Freescale Semiconductor Inc. Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.15.2 Function Documentation

5.15.2.1 void fsl_printf (const char_t ∗ *pcocStr,* ...)

Print data to the interface selected in configuration.

Function is used to print data through another module, which is joined in configuration. It serves for printing debug messages.

Parameters

in	∗*pcou8Str*	String which should be printed. It can contain format specifiers which will be substituted by the values of the variables that are passed to the function through variable count of parameters. Format is same as in standard function printf. Conversion specifiers are either: • supported: diuoxXcsp% • with configurable support: fFeEgG (They are ignored if the support is disabled) • always ignored: aA • not recognized: all others (not recognized conversion specifiers will cause the conversion to abort) • Support of "long long" is configurable - it is ignored if it is disabled. • Long float is ignored. fsl_printf_REF_9 MISRA 2004 Required Rule 16.1 fsl_printf_REF_1 MISRA 2004 Advisory Rule 6.3

5.16 fsl_printf.h File Reference

Module serves to printing debug messages.

```
#include "common_types.h"
```

Functions

- void fsl_printf_init (void)

  Not needed at the moment.
- void fsl_printf (const char_t ∗pcocStr,...)

  Print data to the interface selected in configuration.

5.16.1 Detailed Description

Module serves to printing debug messages.

Version

0.0.0.0

Project Video Listener Platform S32V234

SWVersion 0.9.0

Copyright (c) 2014-2016 Freescale Semiconductor Inc. Copyright (c) 2016 NXP Semiconductors All Rights Reserved

5.16.2 Function Documentation

5.16.2.1 void fsl_printf (const char_t ∗ *pcocStr,* ...)

Print data to the interface selected in configuration.

Function is used to print data through another module, which is joined in configuration. It serves for printing debug messages.

Parameters

in	**pcou8Str*	String which should be printed. It can contain format specifiers which will be substituted by the values of the variables that are passed to the function through variable count of parameters. Format is same as in standard function printf. Conversion specifiers are either: • supported: diuoxXcsp% • with configurable support: fFeEgG (They are ignored if the support is disabled) • always ignored: aA • not recognized: all others (not recognized conversion specifiers will cause the conversion to abort) • Support of "long long" is configurable - it is ignored if it is disabled. • Long float is ignored. fsl_printf_REF_9 MISRA 2004 Required Rule 16.1 fsl_printf_REF_1 MISRA 2004 Advisory Rule 6.3

5.17 fsl_printf_cfg.h File Reference

Fsl_printf module configuration file.

```
#include "dbgb_output.h"
```

5.17.1 Detailed Description

Fsl_printf module configuration file.

Version

0.0.0.0

Project Video Listener Platform S32V234

SWVersion 0.9.0

(c) Copyright 2014 Freescale Semiconductor Inc. All Rights Reserved.

5.18 gic.c File Reference

The GIC driver.

```
#include "common_types.h"
#include "common.h"
#include "gic.h"
```

Functions

- void gic_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the GIC peripheral.
- void gicd_disable_all (void)
  Disable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_enable_all (void)
  Enable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_disable_grp0 (void)
  Disable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_disable_grp1 (void)
  Disable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_enable_grp0 (void)
  Enable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_enable_grp1 (void)
  Enable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_all_to_grp1 (void)
  Move all interrupts to Group1.
- void gicc_enable_fiq (void)
  Enable FIQ signalling.
- void gicc_disable_fiq (void)
  Disable FIQ signalling.
- void gicd_set_priority (const uint32_t u32Irq, const uint32_t u32Priority)
  Set interrupt priority.
- void gicd_set_target (const uint32_t u32Irq, const uint32_t u32Target)
  Set interrupt target.
- void gicd_set_sensitivity (const uint32_t u32Irq, const gicd_sensitivity_t eConfig)
  Set interrupt sensitivity.
- void gicd_enable (const uint32_t u32Irq)
  Enable IRQ within the distributor.
- void gicd_disable (const uint32_t u32Irq)
  Disable IRQ within the distributor.
- void gicd_set_group0 (const uint32_t u32Irq)
  Assign an interrupt to Group0.
- void gicd_set_group1 (const uint32_t u32Irq)
  Assign an interrupt to Group1.
- void gicd_send_sgi_to_this_core (const uint32_t u32Irq, const uint8_t u8NSATT)
  Send SGI to the current core.
- void gicc_disable_all (void)
  Disable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.
- void gicc_enable_all (void)
  Enable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.

5.18.1 Detailed Description

The GIC driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.19 gic.h File Reference

GIC driver header file.

Functions

- void gic_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)

  Get memory region occupied by the GIC peripheral.
- void gicd_disable_all (void)

  Disable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_enable_all (void)

  Enable forwarding of interrupts to CPU interfaces for both groups.
- void gicd_disable_grp0 (void)

  Disable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_disable_grp1 (void)

  Disable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_enable_grp0 (void)

  Enable forwarding of interrupts to CPU interfaces for Group0.
- void gicd_enable_grp1 (void)

  Enable forwarding of interrupts to CPU interfaces for Group1.
- void gicd_all_to_grp1 (void)

  Move all interrupts to Group1.
- void gicd_set_target (const uint32_t u32Irq, const uint32_t u32Target)

  Set interrupt target.
- void gicd_set_priority (const uint32_t u32Irq, const uint32_t u32Priority)

  Set interrupt priority.
- void gicd_set_sensitivity (const uint32_t u32Irq, const gicd_sensitivity_t eConfig)

  Set interrupt sensitivity.
- void gicd_set_group0 (const uint32_t u32Irq)

  Assign an interrupt to Group0.
- void gicd_set_group1 (const uint32_t u32Irq)

  Assign an interrupt to Group1.
- void gicd_enable (const uint32_t u32Irq)

  Enable IRQ within the distributor.
- void gicd_disable (const uint32_t u32Irq)

  Disable IRQ within the distributor.
- void gicc_enable_fiq (void)

  Enable FIQ signalling.
- void gicc_disable_fiq (void)

  Disable FIQ signalling.
- void gicc_disable_all (void)

  Disable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.
- void gicc_enable_all (void)

  Enable signalling of Group0 and Group1 interrupts to the processor by the CPU interface.
- void gicd_send_sgi_to_this_core (const uint32_t u32Irq, const uint8_t u8NSATT)

  Send SGI to the current core.

© 2016 NXP Semiconductors

5.19.1 Detailed Description

GIC driver header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.20 h264_proc.c File Reference

H264 stream processor.

```
#include "common_types.h"
#include "common.h"
#include "h264_proc.h"
```

Functions

- h264_proc_ret_t H264PROC_IsStartOfFrame (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr)

  Function indicates whether current packet is a leading packet of a h264 frame.
- h264_proc_ret_t H264PROC_PreprocessPacket (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr, const uint16_t u16Length, uint32_t ∗const pu32ProcPacketAddr, uint16_t ∗const pu16ProcLength)

  When data stream from sensor does not match input format of the HW decoder this function may be used to prepare data for the decoder in acceptable format.

5.20.1 Detailed Description

H264 stream processor.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.21 h264_proc.h File Reference

H264 stream processor header file.

Functions

- h264_proc_ret_t H264PROC_PreprocessPacket (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr, const uint16_t u16Length, uint32_t *const pu32ProcPacketAddr, uint16_t *const pu16ProcLength)

  When data stream from sensor does not match input format of the HW decoder this function may be used to prepare data for the decoder in acceptable format.
- h264_proc_ret_t H264PROC_IsStartOfFrame (const uint8_t u8StreamIdx, const uint32_t u32PacketAddr)

  Function indicates whether current packet is a leading packet of a h264 frame.

5.21.1 Detailed Description

H264 stream processor header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.22 main.c File Reference

This is the VideoListener firmware driver.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/ioport.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <asm/segment.h>
#include <linux/buffer_head.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/kthread.h>
#include <linux/semaphore.h>
#include <linux/mutex.h>
#include <linux/completion.h>
#include <linux/delay.h>
#include <sm_drv_types.h>
#include <linux/interrupt.h>
```

5.22.1 Detailed Description

This is the VideoListener firmware driver.

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

5.23 main.cpp File Reference

This is the VideoListener video play back demonstration application.

```
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <stdint.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <ctype.h>
#include <pthread.h>
#include "sm_drv_types.h"
#include "vl_sram.h"
#include "application_cfg.h"
#include "eth_queue_cfg.h"
#include "cfg_file.h"
#include "video_app.h"
```

5.23.1 Detailed Description

This is the VideoListener video play back demonstration application.

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

THIS SOFTWARE IS PROVIDED BY NXP "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INC↩ LUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL NXP OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (IN↩ CLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIAB↩ ILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5.24 mmap.h File Reference

The memory map header file.

5.24.1 Detailed Description

The memory map header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.25 mmu.c File Reference

The MMU driver.

```
#include "common_types.h"
#include "mmu_mem_attr.h"
#include "mmu_cfg.h"
#include "mmu.h"
```

Functions

- mmu_ret_t mmu_init (void)
  Initialization function.
- mmu_ret_t mmu_get_region_size (const uint32_t u32Level, mlen_t ∗const RegionSize)
  Get mapping region size at given translation level.
- mmu_ret_t mmu_check_mapping (const va_t VA, const pa_t PA, const mlen_t Size)
  API to check status of a mapping.
- mmu_ret_t mmu_add_mapping (const va_t VA, const pa_t PA, const mlen_t Size, const mem_attr_t Attr)
  API to install new mapping.
- mmu_ret_t mmu_start (void)
  Start MMU within the EL3.
- void cache_d_clean_by_va_range (va_t VA, mlen_t length)
  Clean data cache by address to Point of Coherency.
- mmu_ret_t mmu_stop (void)
  Disable MMU within the EL3.

5.25.1 Detailed Description

The MMU driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.26 mmu.h File Reference

The MMU driver header file.

Macros

- #define mmu_get_attr(type, attr)

  Function to prepare attribute mask combining memory type and memory attributes values.

Functions

- mmu_ret_t mmu_init (void)

  Initialization function.
- mmu_ret_t mmu_add_mapping (const va_t VA, const pa_t PA, const mlen_t Size, const mem_attr_t Attr)

  API to install new mapping.
- mmu_ret_t mmu_check_mapping (const va_t VA, const pa_t PA, const mlen_t Size)

  API to check status of a mapping.
- mmu_ret_t mmu_start (void)

  Start MMU within the EL3.
- mmu_ret_t mmu_stop (void)

  Disable MMU within the EL3.
- mmu_ret_t mmu_get_region_size (const uint32_t u32Level, mlen_t ∗const RegionSize)

  Get mapping region size at given translation level.
- void cache_d_clean_by_va_range (va_t VA, mlen_t length)

  Clean data cache by address to Point of Coherency.

5.26.1 Detailed Description

The MMU driver header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.27 mmu_cfg.h File Reference

The MMU module configuration header file.

Macros

- #define MMU_CFG_ENTRY_POOL_SIZE

  Number of entries allocated to be used by translation tables.
- #define MTYPE_NORMAL

  Memory type: Normal.
- #define MTYPE_NORMAL_NC

  Memory type: Normal, not cached.
- #define MTYPE_DEVICE

  Memory type: Device.

5.27.1 Detailed Description

The MMU module configuration header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.28 mmu_exception.c File Reference

The MMU exception handler.

```
#include "common_types.h"
#include "common.h"
#include "mmu_exception.h"
```

Functions

- void mmu_exception_handler (uint32_t u32esr_elx)

  The exception handler.

5.28.1 Detailed Description

The MMU exception handler.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.29 mmu_exception.h File Reference

The MMU exception handler header file.

5.29.1 Detailed Description

The MMU exception handler header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.30 mmu_mem_attr.h File Reference

Memory attributes header file.

Macros

- #define MA_EL2_EL3_RW

  Memory attribute for EL2/EL3 read/write access.
- #define MA_EL2_EL3_RO

  Memory attribute for EL2/EL3 read-only access.
- #define MA_NON_SECURE

  Memory attribute to create non-secure mapping.
- #define MA_NON_EXEC

  Memory attribute to create non-executable mapping.

5.30.1 Detailed Description

Memory attributes header file.

File contains definitions of supported memory attributes and related internal constants

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.31 pit.c File Reference

The PIT driver.

```
#include "common_types.h"
#include "common.h"
#include "pit.h"
```

Functions

- void mc_me_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the MC_ME peripheral.
- void mc_cgm_get_memory_base (uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the MC_CGM peripheral.
- void pit_get_memory_base (const uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the PIT peripheral.
- pit_ret_t pit_start (const uint8_t u8Instance, const uint8_t u8Channel)
  Start the PIT.
- pit_ret_t pit_enable_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Enable interrupt request generation.
- pit_ret_t pit_disable_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Disable interrupt request generation.
- pit_ret_t pit_stop (const uint8_t u8Instance, const uint8_t u8Channel)
  Stop PIT.
- pit_ret_t pit_confirm_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Confirm interrupt.
- pit_ret_t pit_is_timeout (const uint8_t u8Instance, const uint8_t u8Channel)
  Check if timeout has occurred.
- pit_ret_t pit_get_elapsed_ns (const uint8_t u8Instance, const uint8_t u8Channel, uint32_t *const pu32Result)
  Get number of ns since timer has been started.
- pit_ret_t pit_set_period (const uint8_t u8Instance, const uint8_t u8Channel, const uint32_t u32PeriodNs)
  Set up timer period.

5.31.1 Detailed Description

The PIT driver.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.32 pit.h File Reference

PIT driver header file.

Functions

- void pit_get_memory_base (const uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the PIT peripheral.
- void mc_me_get_memory_base (uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the MC_ME peripheral.
- void mc_cgm_get_memory_base (uint8_t u8Instance, uint32_t *pu32Base, uint32_t *pu32Length)
  Get memory region occupied by the MC_CGM peripheral.
- pit_ret_t pit_start (const uint8_t u8Instance, const uint8_t u8Channel)
  Start the PIT.
- pit_ret_t pit_stop (const uint8_t u8Instance, const uint8_t u8Channel)
  Stop PIT.
- pit_ret_t pit_confirm_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Confirm interrupt.
- pit_ret_t pit_set_period (const uint8_t u8Instance, const uint8_t u8Channel, const uint32_t u32PeriodNs)
  Set up timer period.
- pit_ret_t pit_enable_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Enable interrupt request generation.
- pit_ret_t pit_disable_irq (const uint8_t u8Instance, const uint8_t u8Channel)
  Disable interrupt request generation.
- pit_ret_t pit_is_timeout (const uint8_t u8Instance, const uint8_t u8Channel)
  Check if timeout has occurred.
- pit_ret_t pit_get_elapsed_ns (const uint8_t u8Instance, const uint8_t u8Channel, uint32_t *const pu32Result)
  Get number of ns since timer has been started.

5.32.1 Detailed Description

PIT driver header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.33 sm.c File Reference

The SM core.

```
#include "common_types.h"
#include "common.h"
#include "sm_cfg.h"
#include "sm.h"
#include "dbgb_output.h"
#include "gic.h"
#include "mmu_mem_attr.h"
#include "mmu_exception.h"
#include "mmu_cfg.h"
#include "mmu.h"
```

Functions

- sint32_t sm_install_sc_handler (const uint32_t u32Key, const sm_handler_t pfHandler)
  Install system call handler.
- sint32_t sm_install_fiq_handler (const uint32_t u32IrqID, const sm_handler_t pfHandler)
  Install FIQ handler.
- sm_sc_param_t * sm_sc_get_params (void)
  Retrieve additional system call parameters.
- void sm_sc_set_query_result (uint64_t u64Result)
  Set additional system call return value.
- void sm_send_async_msg (uint64_t u64Reason, uint64_t u64UserVal)
  Send asynchronous message to the current core.

Variables

- void * __sm_init_vec_start
- void * __sm_init_vec_end
- void * __sm_sc_prm_start
- void * __sm_sc_qr_start

5.33.1 Detailed Description

The SM core.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.34 sm.h File Reference

The SM core header file.

Functions

- sint32_t sm_install_sc_handler (const uint32_t u32Key, const sm_handler_t pfHandler)
  Install system call handler.
- sint32_t sm_install_fiq_handler (const uint32_t u32IrqID, const sm_handler_t pfHandler)
  Install FIQ handler.
- sm_sc_param_t * sm_sc_get_params (void)
  Retrieve additional system call parameters.
- void sm_sc_set_query_result (uint64_t u64Result)
  Set additional system call return value.
- void sm_send_async_msg (uint64_t u64Reason, uint64_t u64UserVal)
  Send asynchronous message to the current core.

5.34.1 Detailed Description

The SM core header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.35 sm_cfg.h File Reference

The SM core configuration header file.

Macros

- #define SM_CFG_SC_H_COUNT
  Maximum number of SC handlers.
- #define SM_CFG_FIQ_H_COUNT
  Maximum number of FIQ handlers.
- #define SM_CFG_ENABLE_MMU
  If TRUE then MMU is engaged @ EL3.
- #define SM_CFG_LL_INIT_KEY 0xffffU
  SC identifier of the low-level initialization status request.
- #define SM_CFG_LL_CONFIRM_EVENT_KEY 0xfffeU
  SC identifier of the event confirmation.
- #define SM_CFG_LL_ENABLE_EVENT_KEY 0xfffdU
  SC identifier of the enable/disable request.
- #define SM_CFG_LL_SHUTDOWN_KEY 0xfffcU
  SC identifier of the SM code shut-down request.
- #define sm_cfg_send_notification()
  Called when module needs to send asynchronous notification.
- #define sm_cfg_confirm_notification()
  Called by the SM Core module to confirm notification.
- #define sm_cfg_enable_notification()
  Called by the SM Core module when asynchronous notification feature shall be enabled.
- #define sm_cfg_disable_notification()
  Called by the SM Core module to disable the notification.

5.35.1 Detailed Description

The SM core configuration header file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.36 sm_drv_types.h File Reference

IOCTL interface definition for the firmware driver.

5.36.1 Detailed Description

IOCTL interface definition for the firmware driver.

Copyright (c) 2016 NXP Semiconductors

All Rights Reserved.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

5.37 sm_mmap.c File Reference

The SM core memory management abstraction and configuration module.

```
#include "common_types.h"
#include "common.h"
#include "sm_cfg.h"
#include "sm.h"
#include "mmu_mem_attr.h"
#include "mmu_exception.h"
#include "mmu_cfg.h"
#include "mmu.h"
#include "gic.h"
```

Variables

- void * __sm_vector_table_start
- sm_mmap_long_t __sm_vector_table_size
- void * __sm_t_tables_start
- sm_mmap_long_t __sm_t_tables_size
- void * __sm_stack_start
- sm_mmap_long_t __sm_stack_size
- void * __sm_rodata_cached_start
- sm_mmap_long_t __sm_rodata_cached_size
- void * __sm_text_cached_start
- sm_mmap_long_t __sm_text_cached_size
- void * __sm_data_cached_start
- sm_mmap_long_t __sm_data_cached_size
- void * __sm_data_non_cached_start
- sm_mmap_long_t __sm_data_non_cached_size

5.37.1 Detailed Description

The SM core memory management abstraction and configuration module.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.38 stream_core.c File Reference

Core of the Video Listener Streaming Application running in FIQ.

```
#include "common_types.h"
#include "common.h"
#include "mmap.h"
#include "sm_cfg.h"
#include "sm.h"
#include "mmu_mem_attr.h"
#include "mmu_cfg.h"
#include "mmu.h"
#include "application_cfg.h"
#include "eth_queue_cfg.h"
#include "dec_feed_cfg.h"
#include "stream_core_cfg.h"
#include "stream_core.h"
#include "h264_proc.h"
#include "eth_queue.h"
#include "dec_feed.h"
#include "debug.h"
```

Functions

- uint32_t SCORE_GetErrorMask (void)

  Function for getting current value of stream_core error mask.
- void SCORE_ClearErrorMask (void)

  Function for cleaning stream_core error mask.
- SCORE_tenState SCORE_GetCurrentState (void)

  Function for getting current state of stream_core.
- stream_core_ret_t SCORE_Init (void)

  Initialization function of stream_core module.
- stream_core_ret_t SCORE_GetConf (uint64_t u64Var, uint64_t ∗pu64Val)

  Function for getting current values of runtime configuration parameters.
- stream_core_ret_t SCORE_SetConf (uint64_t u64Var, uint64_t u64Val)

  Function for setting values of runtime configuration parameters.
- stream_core_ret_t SCORE_Start (void)

  Changes stream_core state from READY to PRERUN.
- stream_core_ret_t SCORE_Stop (void)

  Stops other modules and changes stream_core state to READY.
- stream_core_ret_t SCORE_CheckConfig (void)

  Checks whether all required configuration parameters were set.
- void SCORE_Iteration (void)

  Performs single iteration of autonomous stream processing.

5.38.1 Detailed Description

Core of the Video Listener Streaming Application running in FIQ.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.39 stream_core.h File Reference

API of core of the Video Listener Streaming Application running in FIQ.

Enumerations

Functions

- stream_core_ret_t SCORE_CheckConfig (void)
  Checks whether all required configuration parameters were set.
- stream_core_ret_t SCORE_Init (void)
  Initialization function of stream_core module.
- stream_core_ret_t SCORE_GetConf (uint64_t u64Var, uint64_t *pu64Val)
  Function for getting current values of runtime configuration parameters.
- stream_core_ret_t SCORE_SetConf (uint64_t u64Var, uint64_t u64Val)
  Function for setting values of runtime configuration parameters.
- stream_core_ret_t SCORE_Start (void)
  Changes stream_core state from READY to PRERUN.
- stream_core_ret_t SCORE_Stop (void)
  Stops other modules and changes stream_core state to READY.
- uint32_t SCORE_GetErrorMask (void)
  Function for getting current value of stream_core error mask.
- void SCORE_ClearErrorMask (void)
  Function for cleaning stream_core error mask.
- SCORE_tenState SCORE_GetCurrentState (void)
  Function for getting current state of stream_core.
- void SCORE_Iteration (void)
  Performs single iteration of autonomous stream processing.

5.39.1 Detailed Description

API of core of the Video Listener Streaming Application running in FIQ.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

5.40 stream_core_cfg.h File Reference

The StreamCore module configuration file.

Macros

- #define SCORE_CFG_STREAM_TYPE

  Type of stream to be processed. Supported values are: H264_OVER_AVB, JPEG_OVER_AVB.
- #define SCORE_PREPROCESS_H264_STREAM

  If TRUE then stream preprocessing is engaged to prepare compatible input of the decoder HW. Only valid for H264_OVER_AVB stream type.
- #define SCORE_CFG_STREAM_BD_RING_LEN

  Buffer for BDs for each stream.

5.40.1 Detailed Description

The StreamCore module configuration file.

Project Video Listener

Platform S32V234

SWVersion 0.9.0

Copyright (c) 2016 NXP Semiconductors All Rights Reserved.

© 2016 NXP Semiconductors

Index

© 2016 NXP Semiconductors

© 2016 NXP Semiconductors