Intel(R) IXP400 Software API Reference Manual

Automatically generated from sources, February 26, 2007.

*Other names and brands may be claimed as the property of others.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Intel (R) IXP400 Software Component Co−exist

component coexist information

Defines

#define **IX_ETH_HSS_COM_MUT_UNLOCK**()
macros for common mutex control for ETH & HSS co−existence

#define **IX_ETH_HSS_COM_MUT_LOCK**(result)
macros for common mutex control for ETH & HSS co−existence

Functions

IxEthNpeStatus **ixEthNpePortMapCreate** (void)
Select the default port parameters for particular Intel(R) IXP4XX Product Line, setup lookup tables for port conversions Ethernet PortId => LogicalId, LogicalId => Ethernet PortId, Ethernet PortId => Physical Address.

Detailed Description

component coexist information

Define Documentation

```
#define IX_ETH_HSS_COM_MUT_LOCK ( result )
```

macros for common mutex control for ETH & HSS co−existence

Definition at line **78** of file **IxAccCommon.h**.

```
#define IX_ETH_HSS_COM_MUT_UNLOCK (                    )
```

macros for common mutex control for ETH & HSS co−existence

Definition at line **66** of file **IxAccCommon.h**.

Function Documentation

ixEthNpePortMapCreate (void)

Select the default port parameters for particular Intel(R) IXP4XX Product Line, setup lookup tables for port conversions Ethernet PortId => LogicalId, LogicalId => Ethernet PortId, Ethernet PortId => Physical Address.

Parameters:

none

Returns:

IxEthAccStatus IX_ETH_NPE_SUCCESS − port mapping lookup table successfully built
IX_ETH_NPE_FAIL − unknown featureCtrl device ID, failed to get default port mapping

Intel (R) IXP400 Software Assertion Macros (IxAssert) API

Assertion support.

Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) API

The public API for the IXP400 Atm Driver Data component.

Defines

#define **IX_ATMDACC_WARNING**
Warning return code.

#define **IX_ATMDACC_BUSY**
Busy return code.

#define **IX_ATMDACC_RESOURCES_STILL_ALLOCATED**
Disconnect return code.

#define **IX_ATMDACC_DEFAULT_REPLENISH_COUNT**
Default resources usage for RxVcFree replenish mechanism.

#define **IX_ATMDACC_OAM_TX_VPI**
The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to dsicriminate the OAM channel, and should be chosen so that it does not coencide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

#define **IX_ATMDACC_OAM_TX_VCI**
The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to dsicriminate the OAM channel, and should be chosen so that it does not coencide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

#define **IX_ATMDACC_OAM_RX_PORT**
The reserved dummy PORT used for all dedicated OAM Rx connections. Note that this is not a real port but must have a value that lies within the valid range of port values.

#define **IX_ATMDACC_OAM_RX_VPI**
The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coencide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

#define **IX_ATMDACC_OAM_RX_VCI**
The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coencide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

Typedefs

typedef
unsigned int **IxAtmdAccUserId**
User−supplied Id.

typedef void(* **IxAtmdAccRxVcRxCallback**)(**IxAtmLogicalPort** port, **IxAtmdAccUserId** userId, **IxAtmdAccPduStatus** status, **IxAtmdAccClpStatus** clp, IX_OSAL_MBUF *mbufPtr)
Rx callback prototype.

typedef void(* **IxAtmdAccRxVcFreeLowCallback**)(**IxAtmdAccUserId** userId)
Callback prototype for free buffer level is low.

typedef void(* **IxAtmdAccTxVcBufferReturnCallback**)(**IxAtmdAccUserId** userId, IX_OSAL_MBUF *mbufPtr)
Buffer callback prototype.

Enumerations

enum **IxAtmdAccPduStatus** {
IX_ATMDACC_AAL0_VALID,
IX_ATMDACC_OAM_VALID,
IX_ATMDACC_AAL2_VALID,
IX_ATMDACC_AAL5_VALID,
IX_ATMDACC_AAL5_PARTIAL,
IX_ATMDACC_AAL5_CRC_ERROR,
IX_ATMDACC_MBUF_RETURN
}
IxAtmdAcc Pdu status :.

enum **IxAtmdAccAalType** {
IX_ATMDACC_AAL5,
IX_ATMDACC_AAL2,
IX_ATMDACC_AAL0_48,
IX_ATMDACC_AAL0_52,
IX_ATMDACC_OAM,
IX_ATMDACC_MAX_SERVICE_TYPE
}
IxAtmdAcc AAL Service Type :.

enum **IxAtmdAccClpStatus** {
IX_ATMDACC_CLP_NOT_SET,
IX_ATMDACC_CLP_SET
}
IxAtmdAcc CLP indication.

Functions

PUBLIC
IX_STATUS **ixAtmdAccInit** (void)
Initialise the IxAtmdAcc Component.

PUBLIC
IX_STATUS **ixAtmdAccUninit** (void)
Uninitialise the IxAtmdAcc Component.

PUBLIC void **ixAtmdAccShow** (void)
Show IxAtmdAcc configuration on a per port basis.

PUBLIC void **ixAtmdAccStatsShow** (void)
Show all IxAtmdAcc stats.

PUBLIC void **ixAtmdAccStatsReset** (void)
Reset all IxAtmdAcc stats.

PUBLIC**ixAtmdAccRxVcConnect** (**IxAtmLogicalPort** port, unsigned int vpi, unsigned int vci,
IX_STATUS **IxAtmdAccAalType** aalServiceType, **IxAtmRxQueueId** rxQueueId, **IxAtmdAccUserId** userCallbackId, **IxAtmdAccRxVcRxCallback** rxCallback, unsigned int minimumReplenishCount, **IxAtmConnId** *connIdPtr, **IxAtmNpeRxVcId** *npeVcIdPtr)
Connect to a Aal Pdu receive service for a particular port/vpi/vci, and service type.

PUBLIC
IX_STATUS **ixAtmdAccRxVcFreeReplenish** (**IxAtmConnId** connId, IX_OSAL_MBUF *mbufPtr)
Provide free mbufs for data reception on a connection.

PUBLIC**ixAtmdAccRxVcFreeLowCallbackRegister** (**IxAtmConnId** connId, unsigned int
IX_STATUS numberOfMbufs, **IxAtmdAccRxVcFreeLowCallback** callback)
Configure the Rx Free threshold value and register a callback to handle threshold notifications.

PUBLIC**ixAtmdAccRxVcFreeEntriesQuery** (**IxAtmConnId** connId, unsigned int
IX_STATUS *numberOfMbufsPtr)
Get the number of Rx mbufs the system can accept to replenish the the Rx reception mechanism on a particular channel.

PUBLIC
IX_STATUS **ixAtmdAccRxVcEnable** (**IxAtmConnId** connId)
Start the Rx service on a VC.

PUBLIC
IX_STATUS **ixAtmdAccRxVcDisable** (**IxAtmConnId** connId)
Stop the Rx service on a VC.

PUBLIC
IX_STATUS **ixAtmdAccRxVcTryDisconnect** (**IxAtmConnId** connId)
Disconnect a VC from the Rx service.

PUBLIC **ixAtmdAccTxVcConnect** (**IxAtmLogicalPort** port, unsigned int vpi, unsigned int vci,
IX_STATUS **IxAtmdAccAalType** aalServiceType, **IxAtmdAccUserId** userId,
IxAtmdAccTxVcBufferReturnCallback bufferFreeCallback, **IxAtmConnId** *connIdPtr)
Connect to a Aal Pdu transmit service for a particular port/vpi/vci and service type.

PUBLIC **ixAtmdAccTxVcPduSubmit** (**IxAtmConnId** connId, IX_OSAL_MBUF *mbufPtr,
IX_STATUS **IxAtmdAccClpStatus** clp, unsigned int numberOfCells)
Submit a Pdu for transmission on connection.

PUBLIC
IX_STATUS **ixAtmdAccTxVcTryDisconnect** (**IxAtmConnId** connId)
Disconnect from a Aal Pdu transmit service for a particular port/vpi/vci.

Detailed Description

The public API for the IXP400 Atm Driver Data component.

IxAtmdAcc is the low level interface by which AAL0/AAL5 and OAM data gets transmitted to,and received from the Utopia bus.

For AAL0/AAL5 services transmit and receive connections may be established independantly for unique combinations of port,VPI,and VCI.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0_52 is a raw cell service the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

OAM cells cannot be received over the AAL0 service but instead are received over a dedicated OAM service.

For the OAM service an "OAM Tx channel" may be enabled for a port by establishing a single dedicated OAM Tx connection on that port. A single "OAM Rx channel" for all ports may be enabled by establishing a dedicated OAM Rx connection.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC−10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI,VCI, and (in the case of Rx) port values defined below. These values are used purely to descriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particluar VPI/VCIs by manipulating the VPI,VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc. Note that AtmdAcc does not validate the cell headers in a submitted OAM PDU.

This part is related to the User datapath processing

Define Documentation

#define IX_ATMDACC_BUSY

Busy return code.

This constant is used to tell IxAtmDAcc user that the request is correct, but cannot be processed because the IxAtmAcc resources are already used. The user has to retry its request later

Definition at line **107** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_DEFAULT_REPLENISH_COUNT

Default resources usage for RxVcFree replenish mechanism.

This constant is used to tell IxAtmDAcc to allocate and use the minimum of resources for Rx free replenish.

See also:
ixAtmdAccRxVcConnect

Definition at line **138** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_OAM_RX_PORT

The reserved dummy PORT used for all dedicated OAM Rx connections. Note that this is not a real port but must have a value that lies within the valid range of port values.

Definition at line **179** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_OAM_RX_VCI

The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coencide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

Definition at line **203** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_OAM_RX_VPI

The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coencide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

Definition at line **191** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_OAM_TX_VCI

The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to dsicriminate the OAM channel, and should be chosen so that it does not coencide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

Definition at line **167** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_OAM_TX_VPI

The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to dsicriminate the OAM channel, and should be chosen so that it does not coencide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

Definition at line **154** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_RESOURCES_STILL_ALLOCATED

Disconnect return code.

This constant is used to tell IxAtmDAcc user that the disconnect functions are not complete because the resources used by the driver are not yet released. The user has to retry the disconnect call later.

Definition at line **123** of file **IxAtmdAcc.h**.

#define IX_ATMDACC_WARNING

Warning return code.

This constant is used to tell IxAtmDAcc user about a special case.

Definition at line **92** of file **IxAtmdAcc.h**.

Typedef Documentation

typedef void(* IxAtmdAccRxVcFreeLowCallback)(**IxAtmdAccUserId** userId)

Callback prototype for free buffer level is low.

IxAtmdAccRxVcFreeLowCallback is the prototype of the user function which get called on a per−VC basis, when more mbufs are needed to continue the ATM data reception. This function is likely to supply more available mbufs by one or many calls to the replenish function ***ixAtmdAccRxVcFreeReplenish()***

This function is called when the number of available buffers for reception is going under the threshold level as defined in ***ixAtmdAccRxVcFreeLowCallbackRegister()***

This function is called inside an Qmgr dispatch context. No system resource or interrupt−unsafe feature should be used inside this callback.

See also:

ixAtmdAccRxVcFreeLowCallbackRegister

IxAtmdAccRxVcFreeLowCallback

ixAtmdAccRxVcFreeReplenish

ixAtmdAccRxVcFreeEntriesQuery

ixAtmdAccRxVcConnect

Parameters:

userId **IxAtmdAccUserId** [in] − user Id provided in the call to ***ixAtmdAccRxVcConnect()***

Returns:

None

Definition at line **372** of file **IxAtmdAcc.h**.

typedef void(* IxAtmdAccRxVcRxCallback)(**IxAtmLogicalPort** port, **IxAtmdAccUserId** userId, **IxAtmdAccPduStatus** status, **IxAtmdAccClpStatus** clp, IX_OSAL_MBUF * mbufPtr)

Rx callback prototype.

IxAtmdAccRxVcRxCallback is the prototype of the Rx callback user function called once per PDU to pass a receive Pdu to a user on a partilcular connection. The callback is likely to push the mbufs to a protocol layer, and recycle the mbufs for a further use.

Note:

−This function is called ONLY in the context of the ***ixAtmdAccRxDispatch()*** function

See also:

ixAtmdAccRxDispatch

ixAtmdAccRxVcConnect

Parameters:

port **IxAtmLogicalPort** [in] − the port on which this PDU was received a logical PHY port [*IX_UTOPIA_PORT_0 .. IX_UTOPIA_MAX_PORTS* − 1]

userId **IxAtmdAccUserId** [in] − user Id provided in the call to ***ixAtmdAccRxVcConnect()***

status **IxAtmdAccPduStatus** [in] − an indication about the PDU validity. In the case of AAL0 the only possibile value is AAL0_VALID, in this case the client may optionally determine that an Rx timeout occured by checking if the mbuf is compleletly or only

partially filled, the later case indicating a timeout. In the case of OAM the only possible value is OAM valid. The status is set to *IX_ATMDACC_MBUF_RETURN* when the mbuf is released during a disconnect process.

clp **IxAtmdAccClpStatus** [in] − clp indication for this PDU. For AAL5/AAL0_48 this information is set if the clp bit of any Rx cell is set For AAL0−52/OAM the client may inspect the CLP in individual cell headers in the PDU, and this parameter is set to 0.

**mbufPtr* [in] − depending on the servive type a pointer to an mbuf (AAL5/AAL0/OAM) or mbuf chain (AAL5 only), that comprises the complete PDU data.

This parameter is guaranteed not to be a null pointer.

Definition at line **334** of file **IxAtmdAcc.h**.

typedef void(* IxAtmdAccTxVcBufferReturnCallback)(**IxAtmdAccUserId** userId, IX_OSAL_MBUF * mbufPtr)

Buffer callback prototype.

This function is called to relinguish ownership of a transmitted buffer chain to the user.

Note:

−In the case of a chained mbuf the AmtdAcc component can chain many user buffers together and pass ownership to the user in one function call.

Parameters:

userId **IxAtmdAccUserId** [in] − user If provided at registration of this callback.

mbufPtr [in] − a pointer to mbufs or chain of mbufs and is guaranteed not to be a null pointer.

Definition at line **397** of file **IxAtmdAcc.h**.

IxAtmdAccUserId

User−supplied Id.

IxAtmdAccUserId is passed through callbacks and allows the IxAtmdAcc user to identify the source of a call back. The range of this user−owned Id is [0...2^32−1)].

The user provides this own Ids on a per−channel basis as a parameter in a call to ***ixAtmdAccRxVcConnect()*** or ***ixAtmdAccRxVcConnect()***

See also:

ixAtmdAccRxVcConnect

ixAtmdAccTxVcConnect

Definition at line **286** of file **IxAtmdAcc.h**.

Enumeration Type Documentation

enum IxAtmdAccAalType

IxAtmdAcc AAL Service Type :.

IxAtmdAccAalType defines the type of traffic to run on this VC

Enumeration values:

IX_ATMDACC_AAL5	ITU−T AAL5.
IX_ATMDACC_AAL2	ITU−T AAL2 **reserved** for future use.
IX_ATMDACC_AAL0_48	AAL0 48 byte payloads (cell header is added by NPE).
IX_ATMDACC_AAL0_52	AAL0 52 byte cell data (HEC is added by NPE).
IX_ATMDACC_OAM	OAM cell transport service (HEC is added by NPE).
IX_ATMDACC_MAX_SERVICE_TYPE	not a service, used for parameter validation

Definition at line **241** of file **IxAtmdAcc.h**.

enum IxAtmdAccClpStatus

IxAtmdAcc CLP indication.

IxAtmdAccClpStatus defines the CLP status of the current PDU

Enumeration values:

IX_ATMDACC_CLP_NOT_SET	CLP indication is not set.
IX_ATMDACC_CLP_SET	CLP indication is set.

Definition at line **262** of file **IxAtmdAcc.h**.

enum IxAtmdAccPduStatus

IxAtmdAcc Pdu status :.

IxAtmdAccPduStatus is used during a Rx operation to indicate the status of the received PDU

Enumeration values:

IX_ATMDACC_AAL0_VALID	aal0 pdu
IX_ATMDACC_OAM_VALID	OAM pdu.

IX_ATMDACC_AAL2_VALID	aal2 pdu **reserved** for future use
IX_ATMDACC_AAL5_VALID	aal5 pdu complete and trailer is valid
IX_ATMDACC_AAL5_PARTIAL	aal5 pdu not complete, trailer is missing
IX_ATMDACC_AAL5_CRC_ERROR	aal5 pdu not complete, crc error/length error
IX_ATMDACC_MBUF_RETURN	empty buffer returned to the user

Definition at line **218** of file **IxAtmdAcc.h**.

Function Documentation

ixAtmdAccInit (void)

Initialise the IxAtmdAcc Component.

This function initialise the IxAtmdAcc component. This function shall be called before any other function of the API. Its role is to initialise all internal resources of the IxAtmdAcc component.

The ixQmgr component needs to be initialized prior the use of ***ixAtmdAccInit()***

Parameters:

none Failing to initilialize the IxAtmdAcc API before any use of it will result in a failed status. If the specified component is not present, a success status will still be returned, however, a warning indicating the NPE to download to is not present will be issued.

Returns:

◊ IX_SUCCESS initialisation is complete (in case of component not being present, a warning is clearly indicated)

◊ IX_FAIL unable to process this request either because this IxAtmdAcc is already initialised or some unspecified error has occrred.

```
ixAtmdAccRxVcConnect ( IxAtmLogicalPort          port,
                       unsigned int              vpi,
                       unsigned int              vci,
                       IxAtmdAccAalType          aalServiceType,
                       IxAtmRxQueueId            rxQueueId,
                       IxAtmdAccUserId           userCallbackId,
                       IxAtmdAccRxVcRxCallback   rxCallback,
                       unsigned int              minimumReplenishCount,
                       IxAtmConnId *             connIdPtr,
                       IxAtmNpeRxVcId *          npeVcIdPtr
                     )
```

Connect to a Aal Pdu receive service for a particular port/vpi/vci, and service type.

This function allows a user to connect to an Aal5/Aal0/OAM Pdu receive service for a particular port/vpi/vci. It registers the callback and allocates internal resources and a Connection Id to be used in further API calls related to this VCC.

The function will setup VC receive service on the specified Rx queue.

This function is blocking and makes use internal locks, and hence should not be called from an interrupt context.

On return from ***ixAtmdAccRxVcConnect()*** with a failure status, the connection Id parameter is unspecified. Its value cannot be used. A connId is the reference by which IxAtmdAcc refers to a connected VC. This identifier is the result of a succesful call to a connect function. This identifier is invalid after a sucessful call to a disconnect function.

Calling this function for the same combination of Vpi, Vci and more than once without calling ***ixAtmdAccRxVcTryDisconnect()*** will result in a failure status.

If this function returns success the user should supply receive buffers by calling ***ixAtmdAccRxVcFreeReplenish()*** and then call ***ixAtmdAccRxVcEnable()*** to begin receiving pdus.

There is a choice of two receive Qs on which the VC pdus could be receive. The user must associate the VC with one of these. Essentially having two qs allows more flexible system configuration such as have high prioriy traffic on one q (e.g. voice) and low priority traffic on the other (e.g. data). The high priority Q could be serviced in preference to the low priority Q. One queue may be configured to be serviced as soon as there is traffic, the other queue may be configured to be serviced by a polling mechanism running at idle time.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Received AAL0 PDUs will be be a multiple of the cell data size (48/52). AAL0_52 is a raw cell service and includes an ATM cell header (excluding HEC) at the start of each cell.

A single "OAM Rx channel" for all ports may be enabled by establishing a dedicated OAM Rx connection.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC−10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI,VCI, and (in the case of Rx) port values. These values are used purely to descriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particluar VPI/VCIs by manipulating the VPI,VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc.

Calling this function prior to enable the port will fail.

See also:

ixAtmdAccRxDispatch

ixAtmdAccRxVcEnable

ixAtmdAccRxVcDisable

ixAtmdAccRxVcTryDisconnect

ixAtmdAccPortEnable

Parameters:

port	**IxAtmLogicalPort** [in] − VC identification : logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]
vpi	unsigned int [in] − VC identification : ATM Vpi [0..255] or IX_ATMDACC_OAM_VPI
vci	unsigned int [in] − VC identification : ATM Vci [0..65535] or IX_ATMDACC_OAM_VCI
aalServiceType	**IxAtmdAccAalType** [in] − type of service: AAL5, AAL0_48, AAL0_52, or OAM
rxQueueId	**IxAtmRxQueueId** [in] − this identifieds which of two Qs the VC should use.when icoming traffic is processed
userCallbackId	**IxAtmdAccUserId** [in] − user Id used later as a parameter to the supplied rxCallback.
rxCallback	[in] − function called when mbufs are received. This parameter cannot be a null pointer.
bufferFreeCallback	[in] − function to be called to return ownership of buffers to IxAtmdAcc user.
minimumReplenishCount	unsigned int [in] − For AAL5/AAL0 the number of free mbufs to be used with this channel. Use a high number when the expected traffic rate on this channel is high, or when the user's mbufs are small, or when the RxVcFreeLow Notification has to be invoked less often. When this value is IX_ATMDACC_DEFAULT_REPLENISH_COUNT, the minimum of resources will be used. Depending on traffic rate, pdu size and mbuf size, rxfree queue size, polling/interrupt rate, this value may require to be replaced by a different value in the range 1−128 For OAM the rxFree queue size is fixed by atmdAcc and this parameter is ignored.
connIdPtr	**IxAtmConnId** [out] − pointer to a connection Id This parameter cannot be a null pointer.
npeVcIdPtr	**IxAtmNpeRxVcId** [out] − pointer to an npe Vc Id This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS successful call to IxAtmdAccRxVcConnect

◊ IX_ATMDACC_BUSY cannot process this request : no VC is available

◊ IX_FAIL parameter error, VC already in use, attempt to connect AAL service on reserved OAM VPI/VCI, attempt to connect OAM service on VPI/VCI other than the reserved OAM VPI/VCI, port is not initialised, or some other error occurs during processing.

ixAtmdAccRxVcDisable (**IxAtmConnId** *connId*)

Stop the Rx service on a VC.

This functions stops the traffic reception for a particular VC connection.

Once invoked, incoming Pdus are discarded by the hardware. Any Pdus pending will be freed to the user

Hence once this function returns no more receive callbacks will be called for that VC. However, buffer free callbacks will be invoked until such time as all buffers supplied by the user have been freed back to the user

Calling this function doe not invalidate the connId. ***ixAtmdAccRxVcEnable()*** can be invoked to enable Pdu reception again.

If the traffic is already stopped, this function returns IX_SUCCESS.

This function is not reentrant and should not be used inside an interrupt context.

See also:

ixAtmdAccRxVcConnect

ixAtmdAccRxVcEnable

ixAtmdAccRxVcDisable

Parameters:

connId **IxAtmConnId** [in] − connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

Returns:

◊ IX_SUCCESS successful call to ***ixAtmdAccRxVcDisable()***.

◊ IX_ATMDACC_WARNING the channel is already disabled

◊ IX_FAIL invalid parameters or some unspecified internal error occured

ixAtmdAccRxVcEnable (**IxAtmConnId** *connId*)

Start the Rx service on a VC.

This functions kicks−off the traffic reception for a particular VC. Once invoked, incoming PDUs will be made available by the hardware and are eventually directed to the ***IxAtmdAccRxVcRxCallback()*** callback registered for the connection.

If the traffic is already running, this function returns IX_SUCCESS. This function can be invoked many times.

IxAtmdAccRxVcFreeLowCallback event will occur only after ***ixAtmdAccRxVcEnable()*** function is invoked.

Before using this function, the ***ixAtmdAccRxVcFreeReplenish()*** function has to be used to replenish the Rx

Free queue. If not, incoming traffic may be discarded.and in the case of interrupt driven reception the ***IxAtmdAccRxVcFreeLowCallback()*** callback may be invoked as a side effect during a replenish action.

This function is not reentrant and should not be used inside an interrupt context.

For an VC connection this function can be called after a call to ***ixAtmdAccRxVcDisable()*** and should not be called after ***ixAtmdAccRxVcTryDisconnect()***

See also:

ixAtmdAccRxVcDisable

ixAtmdAccRxVcConnect

ixAtmdAccRxVcFreeReplenish

Parameters:

connId **IxAtmConnId** [in] − connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

Returns:

◊ IX_SUCCESS successful call to ixAtmdAccRxVcEnable

◊ IX_ATMDACC_WARNING the channel is already enabled

◊ IX_FAIL invalid parameters or some unspecified internal error occured.

```
ixAtmdAccRxVcFreeEntriesQuery ( IxAtmConnId connId,
                                unsigned int * numberOfMbufsPtr
                              )
```

Get the number of Rx mbufs the system can accept to replenish the the Rx reception mechanism on a particular channel.

The ixAtmdAccRxVcFreeEntriesQuery function is used to retrieve the current number of available mbuf entries for reception, on a per−VC basis. This function can be used to know the number of mbufs which can be provided using ***ixAtmdAccRxVcFreeReplenish()***.

This function can be used from a timer context, or can be associated with a threshold event, or can be used inside an active polling mechanism which is under user control.

This function is reentrant and does not use system resources and can be invoked from an interrupt context.

Parameters:

connId	**IxAtmConnId** [in] − connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*
numberOfMbufsPtr	unsigned int [out] − Pointer to the number of available entries. . This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the current number of mbufs not yet used for incoming traffic

◊ IX_FAIL invalid parameter

See also:

ixAtmdAccRxVcFreeReplenish

ixAtmdAccRxVcFreeLowCallbackRegister (	**IxAtmConnId**	*connId*,
	unsigned int	*numberOfMbufs*,
	IxAtmdAccRxVcFreeLowCallback	*callback*
)		

Configure the Rx Free threshold value and register a callback to handle threshold notifications.

The function ixAtmdAccRxVcFreeLowCallbackRegister sets the threshold value for a particular Rx VC. When the number of buffers reaches this threshold the callback is invoked.

This function should be called once per VC before Rx traffic is enabled.This function will fail if the curent level of the free buffers is equal or less than the threshold value.

See also:

ixAtmdAccRxVcFreeLowCallbackRegister

IxAtmdAccRxVcFreeLowCallback

ixAtmdAccRxVcFreeReplenish

ixAtmdAccRxVcFreeEntriesQuery

ixAtmdAccRxVcConnect

Parameters:

connId **IxAtmConnId** [in] − connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

numberOfMbufs unsigned int [in] − threshold number of buffers. This number has to be a power of 2, one of the values 0,1,2,4,8,16,32.... The maximum value cannot be more than half of the rxFree queue size (which can be retrieved using ***ixAtmdAccRxVcFreeEntriesQuery()*** before any use of the ***ixAtmdAccRxVcFreeReplenish()*** function)

callback **IxAtmdAccRxVcFreeLowCallback** [in] − function telling the user that the number of free buffers has reduced to the threshold value.

Returns:

◊ IX_SUCCESS Threshold set successfully.

◊ IX_FAIL parameter error or the current number of free buffers is less than or equal to the threshold supplied or some unspecified error has occrred.

Note:

− the callback will be called when the threshold level will drop from exactly (numberOfMbufs + 1) to (numberOfMbufs).

```
ixAtmdAccRxVcFreeReplenish ( IxAtmConnId      connId,
                             IX_OSAL_MBUF *  mbufPtr
                           )
```

Provide free mbufs for data reception on a connection.

This function provides mbufs for data reception by the hardware. This function needs to be called by the user on a regular basis to ensure no packet loss. Providing free buffers is a connection−based feature; each connection can have different requirements in terms of buffer size number of buffers, recycling rate. This function could be invoked from within the context of a ***IxAtmdAccRxVcFreeLowCallback()*** callback for a particular VC

Mbufs provided through this function call can be chained. They will be unchained internally. A call to this function with chained mbufs or multiple calls with unchained mbufs are equivalent, but calls with unchained mbufs are more efficients.

Mbufs provided to this interface need to be able to hold at least one full cell payload (48/52 bytes, depending on service type). Chained buffers with a size less than the size supported by the hardware will be returned through the Rx callback provided during the connect step.

Failing to invoke this function prior to enabling the Rx traffic can result in packet loss.

This function is not reentrant for the same connId.

This function does not use system resources and can be invoked from an interrupt context.

Note:

− Over replenish is detected, and extra mbufs are returned through the Rx callback provided during the connect step.

− Mbuf provided to the replenish function should have a length greater or equal to 48/52 bytes according to service type.

− The memory cache of mMbuf payload should be invalidated prior to Mbuf submission. Flushing the Mbuf headers is handled by IxAtmdAcc.

− When a chained mbuf is provided, this function process the mbufs up to the hardware limit and invokes the user−supplied callback to release extra buffers.

See also:

ixAtmdAccRxVcFreeLowCallbackRegister

IxAtmdAccRxVcFreeLowCallback

ixAtmdAccRxVcConnect

Parameters:

connId **IxAtmConnId** [in] − connection Id as returned from a succesfull call to *IxAtmdAccRxVcConnect()*

mbufPtr [in] − pointer to a mbuf structure to be used for data reception. The mbuf pointed to by this parameter can be chained to an other mbuf.

Returns:

◊ IX_SUCCESS successful call to ***ixAtmdAccRxVcFreeReplenish()*** and the mbuf is now ready to use for incoming traffic.

◊ IX_ATMDACC_BUSY cannot process this request because the max number of outstanding free buffers has been reached or the internal resources have exhausted for this VC. The user is responsible for retrying this request later.

◊ IX_FAIL cannot process this request because of parameter errors or some unspecified internal error has occurred.

Note:

− It is not always guaranteed the replenish step to be as fast as the hardware is consuming Rx Free mbufs. There is nothing in IxAtmdAcc to guarantee that replenish reaches the rxFree threshold level. If the threshold level is not reached, the next rxFree low notification for this channel will not be triggered. The preferred ways to replenish can be as follows (depending on applications and implementations) :

◊ Replenish in a rxFree low notification until the function **ixAtmdAccRxVcFreeReplenish()** returns IX_ATMDACC_BUSY

◊ Query the queue level using

See also:

ixAtmdAccRxVcFreeEntriesQuery, then , replenish using ***ixAtmdAccRxVcFreeReplenish()***, then query the queue level again, and replenish if the threshold is still not reached.

◊ Trigger replenish from an other event source and use rxFree starvation to throttle the Rx traffic.

ixAtmdAccRxVcTryDisconnect (**IxAtmConnId** *connId*)

Disconnect a VC from the Rx service.

This function deregisters the VC and guarantees that all resources associated with this VC are free. After its execution, the connection Id is not available.

This function will fail until such time as all resources allocated to the VC connection have been freed. The user is responsible to delay and call again this function many times until a success status is returned.

This function needs internal locks and should not be called from an interrupt context

Parameters:

connId **IxAtmConnId** [in] − connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

Returns:

◊ IX_SUCCESS successful call to ixAtmdAccRxVcDisable

◊ IX_ATMDACC_RESOURCES_STILL_ALLOCATED not all resources associated with the connection have been freed.

◊ IX_FAIL cannot process this request because of a parameter error

ixAtmdAccShow (void)

Show IxAtmdAcc configuration on a per port basis.

Parameters:

none

Returns:

none

Note:

− Display use printf() and are redirected to stdout

ixAtmdAccStatsReset (void)

Reset all IxAtmdAcc stats.

Parameters:

none

Returns:

none

ixAtmdAccStatsShow (void)

Show all IxAtmdAcc stats.

Parameters:

none

Returns:

none

Note:

− Stats display use printf() and are redirected to stdout

```
ixAtmdAccTxVcConnect ( IxAtmLogicalPort                    port,
                       unsigned int                        vpi,
                       unsigned int                        vci,
                       IxAtmdAccAalType                    aalServiceType,
                       IxAtmdAccUserId                     userId,
                       IxAtmdAccTxVcBufferReturnCallback   bufferFreeCallback,
                       IxAtmConnId *                       connIdPtr
                     )
```

Connect to a Aal Pdu transmit service for a particular port/vpi/vci and service type.

This function allows a user to connect to an Aal5/Aal0/OAM Pdu transmit service for a particular port/vpi/vci. It registers the callback and allocates internal resources and a Connection Id to be used in further API calls related to this VC.

The function will setup VC transmit service on the specified on the specified port. A connId is the reference by which IxAtmdAcc refers to a connected VC. This identifier is the result of a succesful call to a connect function. This identifier is invalid after a sucessful call to a disconnect function.

This function needs internal locks, and hence should not be called from an interrupt context.

On return from ***ixAtmdAccTxVcConnect()*** with a failure status, the connection Id parameter is unspecified. Its value cannot be used.

Calling this function for the same combination of port, Vpi, Vci and more than once without calling ***ixAtmdAccTxVcTryDisconnect()*** will result in a failure status.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0_52 is a raw cell service the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

For the OAM service an "OAM Tx channel" may be enabled for a port by establishing a single dedicated OAM Tx connection on that port.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC−10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI,VCI, and (in the case of Rx) port values. These values are used purely to descriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particluar VPI/VCIs by manipulating the VPI,VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc.

Calling this function before enabling the port will fail.

See also:

ixAtmdAccTxVcTryDisconnect

ixAtmdAccPortTxScheduledModeEnable

ixAtmdAccPortEnable

Parameters:

port	**IxAtmLogicalPort** [in] − VC identification : logical PHY port [*IX_UTOPIA_PORT_0 .. IX_UTOPIA_MAX_PORTS* − 1]
vpi	unsigned int [in] − VC identification : ATM Vpi [0..255] or IX_ATMDACC_OAM_VPI
vci	unsigned int [in] − VC identification : ATM Vci [0..65535] or IX_ATMDACC_OAM_VCI
aalServiceType	**IxAtmdAccAalType** [in] − type of service AAL5, AAL0_48, AAL0_52, or OAM
userId	**IxAtmdAccUserId** [in] − user id to be used later during callbacks related to this channel
bufferFreeCallback	**IxAtmdAccTxVcBufferReturnCallback** [in] − function called when mbufs transmission is complete. This parameter cannot be a null pointer.
connIdPtr	**IxAtmConnId** [out] − Pointer to a connection Id. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS successful call to *IxAtmdAccRxVcConnect()*.

◊ IX_ATMDACC_BUSY cannot process this request because no VC is available

◊ IX_FAIL parameter error, VC already in use, attempt to connect AAL service on reserved OAM VPI/VCI, attempt to connect OAM service on VPI/VCI other than the reserved OAM VPI/VCI, port is not initialised, or some other error occurs during processing.

```
ixAtmdAccTxVcPduSubmit ( IxAtmConnId          connId,
                         IX_OSAL_MBUF *       mbufPtr,
                         IxAtmdAccClpStatus   clp,
                         unsigned int         numberOfCells
                       )
```

Submit a Pdu for transmission on connection.

A data user calls this function to submit an mbufs containing a Pdu to be transmitted. The buffer supplied can be chained and the Pdu it contains must be complete.

The transmission behavior of this call depends on the operational mode of the port on which the connection is made.

In unscheduled mode the mbuf will be submitted to the hardware immediately if sufficent resource is available. Otherwise the function will return failure.

In scheduled mode the buffer is queued internally in IxAtmdAcc. The cell demand is made known to the traffic shaping entity. Cells from the buffers are MUXed onto the port some time later as dictated by the traffic shaping entity. The traffic shaping entity does this by sending transmit schedules to IxAtmdAcc via

ixAtmdAccPortTxProcess() function call.

Note that the dedicated OAM channel is scheduled just like any other channel. This means that any OAM traffic relating to an active AAL0/AAL5 connection will be scheduled independantly of the AAL0/AAL5 traffic for that connection.

When transmission is complete, the Tx Done mechanism will give the ownership of these buffers back to the customer. The Tx done mechanism must be in operation before transmission is attempted.

For AAL0/OAM submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0_52 and OAM are raw cell services, and the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

See also:

IxAtmdAccTxVcBufferReturnCallback

ixAtmdAccTxDoneDispatch

Parameters:

connId	**IxAtmConnId** [in] − connection Id as resulted from a succesfull call to ***ixAtmdAccTxVcConnect()***
mbufPtr	[in] − pointer to a chained structure of mbufs to transmit. This parameter cannot be a null pointer.
clp	**IxAtmdAccClpStatus** [in] − clp indication for this PDU. All cells of this pdu will be sent with the clp bit set
numberOfCells	unsigned int [in] − number of cells in the PDU.

Returns:

◊ IX_SUCCESS successful call to ***ixAtmdAccTxVcPduSubmit()*** The pdu pointed by the mbufPtr parameter will be transmitted

◊ IX_ATMDACC_BUSY unable to process this request because internal resources are all used. The caller is responsible for retrying this request later.

◊ IX_FAIL unable to process this request because of error in the parameters (wrong connId supplied, or wrong mbuf pointer supplied), the total length of all buffers in the chain should be a multiple of the cell size (48/52 depending on the service type), or unspecified error during processing

Note:

− This function in not re−entrant for the same VC (e.g. : two thread cannot send PDUs for the same VC). But two threads can safely call this function with a different connection Id

− In unscheduled mode, this function is not re−entrant on a per port basis. The size of pdus is limited to 8Kb.

− 0−length mbufs should be removed from the chain before submission. The total length of the pdu (sdu + padding +trailer) has to be updated in the header of the first mbuf of a chain of mbufs.

− Aal5 trailer information (UUI, CPI, SDU length) has to be supplied before submission.

− The payload memory cache should be flushed, if needed, prior to transmission. Mbuf headers are flushed by IxAtmdAcc

− This function does not use system resources and can be used inside an interrupt context

ixAtmdAccTxVcTryDisconnect (**IxAtmConnId** *connId*)

Disconnect from a Aal Pdu transmit service for a particular port/vpi/vci.

This function deregisters the VC and guarantees that all resources associated with this VC are free. After its execution, the connection Id is not available.

This function will fail until such time as all resources allocated to the VC connection have been freed. The user is responsible to delay and call again this function many times until a success status is returned.

After its execution, the connection Id is not available.

Parameters:

connId **IxAtmConnId** [in] − connection Id as resulted from a succesfull call to ***ixAtmdAccTxVcConnect()***

Returns:

◊ IX_SUCCESS successful call to ***ixAtmdAccTxVcTryDisconnect()***

◊ IX_ATMDACC_RESOURCES_STILL_ALLOCATED not all resources associated with the connection have been freed. This condition will disappear after Tx and TxDone is complete for this channel.

◊ IX_FAIL unable to process this request because of errors in the parameters (wrong connId supplied)

Note:

− This function needs internal locks and should not be called from an interrupt context

− If the *IX_ATMDACC_RESOURCES_STILL_ALLOCATED* error does not clear after a while, this may be linked to a previous problem of cell overscheduling. Diabling the port and retry a disconnect will free the resources associated with this channel.

See also:

ixAtmdAccPortTxProcess

ixAtmdAccUninit (void)

Uninitialise the IxAtmdAcc Component.

This function uninitialises the IxAtmdAcc component. Its role is to uninitialise all internal resources of the IxAtmdAcc component. It de−allocatess all the buffers allocated during intialization and also destroys the mutex taken. This should be the last function to be called to clean up all resources.

Parameters:

none

Returns:

◊ IX_SUCCESS uninitialisation is complete

◊ IX_FAIL unable to process this request either because this IxAtmdAcc is already uninitialised or some unspecified error has occurred.

Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Control API

The public API for the IXP400 Atm Driver Control component.

Modules

Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API

The public API for the IXP400 Atm Driver Control component.

Defines

#define **IX_ATMDACC_PORT_DISABLE_IN_PROGRESS**

Port enable return code.

#define **IX_ATMDACC_ALLPDUS**

All PDUs.

Typedefs

typedef**IxAtmdAccRxDispatcher**)(**IxAtmRxQueueId** rxQueueId, unsigned int IX_STATUS(* numberOfPdusToProcess, unsigned int *reservedPtr)

Callback prototype for notification of available PDUs for an Rx Q.

typedef**IxAtmdAccTxDoneDispatcher**)(unsigned int numberOfPdusToProcess, unsigned int IX_STATUS(* *reservedPtr)

Callback prototype for transmitted mbuf when threshold level is crossed.

typedef void(* **IxAtmdAccPortTxLowCallback**)(**IxAtmLogicalPort** port, unsigned int numberOfAvailableCells)

Notification that the threshold number of scheduled cells remains in a port's transmit Q.

typedef**IxAtmdAccTxVcDemandUpdateCallback**)(**IxAtmLogicalPort** port, int vcId, unsigned IX_STATUS(* int numberOfCells)

Prototype to submit cells for transmission.

typedef void(* **IxAtmdAccTxVcDemandClearCallback**)(**IxAtmLogicalPort** port, int vcId)

prototype to remove all currently queued cells from a registered VC

typedef**IxAtmdAccTxSchVcIdGetCallback**)(**IxAtmLogicalPort** port, unsigned int vpi, unsigned IX_STATUS(* int vci, **IxAtmConnId** connId, int *vcId)

prototype to get a scheduler vc id

Functions

PUBLIC **ixAtmdAccRxDispatcherRegister** (**IxAtmRxQueueId** queueId,
IX_STATUS **IxAtmdAccRxDispatcher** callback)

Register a notification callback to be invoked when there is at least one entry on a particular Rx queue.

PUBLIC
IX_STATUS **ixAtmdAccRxDispatcherUnregister** (**IxAtmRxQueueId** atmdQueueId)

UnRegister a notification callback to be invoked when there is at least one entry on a particular Rx queue.

PUBLIC **ixAtmdAccRxDispatch** (**IxAtmRxQueueId** rxQueueId, unsigned int
IX_STATUS numberOfPdusToProcess, unsigned int *numberOfPdusProcessedPtr)

Control function which executes Rx processing for a particular Rx stream.

PUBLIC **ixAtmdAccRxLevelQuery** (**IxAtmRxQueueId** rxQueueId, unsigned int
IX_STATUS *numberOfPdusPtr)

Query the number of entries in a particular Rx queue.

PUBLIC **ixAtmdAccRxQueueSizeQuery** (**IxAtmRxQueueId** rxQueueId, unsigned int
IX_STATUS *numberOfPdusPtr)

Query the size of a particular Rx queue.

PUBLIC **ixAtmdAccPortTxFreeEntriesQuery** (**IxAtmLogicalPort** port, unsigned int
IX_STATUS *numberOfCellsPtr)

Get the number of available cells the system can accept for transmission.

PUBLIC **ixAtmdAccPortTxCallbackRegister** (**IxAtmLogicalPort** port, unsigned int
IX_STATUS numberOfCells, **IxAtmdAccPortTxLowCallback** callback)

Configure the Tx port threshold value and register a callback to handle threshold notifications.

PUBLIC void **ixAtmdAccPortTxCallbackUnregister** (**IxAtmLogicalPort** port)

Unregister a callback to handle threshold notifications.

PUBLIC **ixAtmdAccPortTxScheduledModeEnable** (**IxAtmLogicalPort** port,
IX_STATUS **IxAtmdAccTxVcDemandUpdateCallback** vcDemandUpdateCallback,
IxAtmdAccTxVcDemandClearCallback vcDemandClearCallback,
IxAtmdAccTxSchVcIdGetCallback vcIdGetCallback)

Put the port into Scheduled Mode.

PUBLIC
IX_STATUS **ixAtmdAccPortTxScheduledModeDisable** (**IxAtmLogicalPort** port)

Put the port into UnScheduled Mode.

PUBLIC **ixAtmdAccPortTxProcess** (**IxAtmLogicalPort** port, **IxAtmScheduleTable**
IX_STATUS *scheduleTablePtr)

Transmit queue cells to the H/W based on the supplied schedule table.

PUBLIC IX_STATUS **ixAtmdAccTxDoneDispatch** (unsigned int numberOfPdusToProcess, unsigned int *numberOfPdusProcessedPtr)
Process a number of pending transmit done pdus from the hardware.

PUBLIC IX_STATUS **ixAtmdAccTxDoneLevelQuery** (unsigned int *numberOfPdusPtr)
Query the current number of transmit pdus ready for recycling.

PUBLIC IX_STATUS **ixAtmdAccTxDoneQueueSizeQuery** (unsigned int *numberOfPdusPtr)
Query the TxDone queue size.

PUBLIC IX_STATUS **ixAtmdAccTxDoneDispatcherRegister** (unsigned int numberOfPdus, **IxAtmdAccTxDoneDispatcher** notificationCallback)
Configure the Tx Done stream threshold value and register a callback to handle threshold notifications.

PUBLIC void **ixAtmdAccTxDoneDispatcherUnregister** (void)
Unregister a callback to handle threshold notifications.

PUBLIC IX_STATUS **ixAtmdAccUtopiaConfigSet** (const **IxAtmdAccUtopiaConfig** *ixAtmdAccUtopiaConfigPtr)
Send the configuration structure to the Utopia interface.

PUBLIC IX_STATUS **ixAtmdAccUtopiaStatusGet** (**IxAtmdAccUtopiaStatus** *ixAtmdAccUtopiaStatus)
Get the Utopia interface configuration.

PUBLIC IX_STATUS **ixAtmdAccPortEnable** (**IxAtmLogicalPort** port)
enable a PHY logical port

PUBLIC IX_STATUS **ixAtmdAccPortDisable** (**IxAtmLogicalPort** port)
disable a PHY logical port

PUBLIC BOOL **ixAtmdAccPortDisableComplete** (**IxAtmLogicalPort** port)
disable a PHY logical port

PUBLIC IX_STATUS **ixAtmdAccUtopiaConfigReset** (const **IxAtmdAccUtopiaConfig** *utConfig)
Reset the configuration structure to the Utopia interface initializes configuration registers and write to NPE−A Load and get response from NPE−A and resets the Utopia interface.

Detailed Description

The public API for the IXP400 Atm Driver Control component.

IxAtmdAcc is the low level interface by which AAL PDU get transmitted to,and received from the Utopia bus

This part is related to the Control configuration

Define Documentation

#define IX_ATMDACC_ALLPDUS

All PDUs.

This constant is used to tell IxAtmDAcc to process all PDUs from the Rx queue or the Tx Done

See also:

IxAtmdAccRxDispatcher

IxAtmdAccTxDoneDispatcher

Definition at line **74** of file **IxAtmdAccCtrl.h**.

#define IX_ATMDACC_PORT_DISABLE_IN_PROGRESS

Port enable return code.

This constant is used to tell IxAtmDAcc user that the port disable functions are not complete. The user can call **ixAtmdAccPortDisableComplete()** to find out when the disable has finished. The port enable can then proceed.

Definition at line **57** of file **IxAtmdAccCtrl.h**.

Typedef Documentation

typedef void(* IxAtmdAccPortTxLowCallback)(**IxAtmLogicalPort** port, unsigned int numberOfAvailableCells)

Notification that the threshold number of scheduled cells remains in a port's transmit Q.

The is the prototype for of the user notification function which gets called on a per−port basis, when the number of remaining scheduled cells to be transmitted decreases to the threshold level. The number of cells passed as a parameter can be used for scheduling purposes as the maximum number of cells that can be passed in a schedule table to the ***ixAtmdAccPortTxProcess()*** function.

See also:

ixAtmdAccPortTxCallbackRegister

ixAtmdAccPortTxProcess

ixAtmdAccPortTxFreeEntriesQuery

Parameters:

port	**IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]
numberOfAvailableCells	unsigned int [in] − number of available cell entries.for the port

Note:

− This functions shall not use system resources when used inside an interrupt context.

Definition at line **188** of file **IxAtmdAccCtrl.h**.

typedef IX_STATUS(* IxAtmdAccRxDispatcher)(**IxAtmRxQueueId** rxQueueId, unsigned int numberOfPdusToProcess, unsigned int *reservedPtr)

Callback prototype for notification of available PDUs for an Rx Q.

This a protoype for a function which is called when there is at least one Pdu available for processing on a particular Rx Q.

This function should call ***ixAtmdAccRxDispatch()*** with the aprropriate number of parameters to read and process the Rx Q.

See also:

ixAtmdAccRxDispatch

ixAtmdAccRxVcConnect

ixAtmdAccRxDispatcherRegister

Parameters:

rxQueueId	**IxAtmRxQueueId** [in] indicates which Rx queue to has Pdus to process.
numberOfPdusToProcess	unsigned int [in] indicates the minimum number of PDUs available to process all PDUs from the queue.
reservedPtr	unsigned int* [out] pointer to a int location which can be written to, but does not retain written values. This is provided to make this prototype compatible with ***ixAtmdAccRxDispatch()***

Returns:

◊ int − ignored.

Definition at line **108** of file **IxAtmdAccCtrl.h**.

typedef IX_STATUS(* IxAtmdAccTxDoneDispatcher)(unsigned int numberOfPdusToProcess, unsigned int *reservedPtr)

Callback prototype for transmitted mbuf when threshold level is crossed.

IxAtmdAccTxDoneDispatcher is the prototype of the user function which get called when pdus are completely transmitted. This function is likely to call the ***ixAtmdAccTxDoneDispatch()*** function.

This function is called when the number of available pdus for reception is crossing the threshold level as defined in ***ixAtmdAccTxDoneDispatcherRegister()***

This function is called inside an Qmgr dispatch context. No system resource or interrupt−unsafe feature should be used inside this callback.

Transmitted buffers recycling implementation is a sytem−wide mechanism and needs to be set before any traffic is started. If this threshold mechanism is not used, the user is responsible for polling the transmitted buffers with ***ixAtmdAccTxDoneDispatch()*** and ***ixAtmdAccTxDoneLevelQuery()*** functions.

See also:

ixAtmdAccTxDoneDispatcherRegister

ixAtmdAccTxDoneDispatch

ixAtmdAccTxDoneLevelQuery

Parameters:

numberOfPdusToProcess	unsigned int [in] − The current number of pdus currently available for recycling
**reservedPtr*	unsigned int [out] − pointer to a int location which can be written to but does not retain written values. This is provided to make this prototype compatible with ***ixAtmdAccTxDoneDispatch()***

Returns:

◊ IX_SUCCESS This is provided to make this prototype compatible with ***ixAtmdAccTxDoneDispatch()***

◊ IX_FAIL invalid parameters or some unspecified internal error occured. This is provided to make this prototype compatible with ***ixAtmdAccTxDoneDispatch()***

Definition at line **159** of file **IxAtmdAccCtrl.h**.

typedef IX_STATUS(* IxAtmdAccTxSchVcIdGetCallback)(**IxAtmLogicalPort** port, unsigned int vpi, unsigned int vci, **IxAtmConnId** connId, int *vcId)

prototype to get a scheduler vc id

IxAtmdAccTxSchVcIdGetCallback is the prototype of the function to get a scheduler vcId

See also:

IxAtmdAccTxVcDemandUpdateCallback

IxAtmdAccTxVcDemandClearCallback

IxAtmdAccTxSchVcIdGetCallback

ixAtmdAccPortTxScheduledModeEnable

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM logical port on which the VC is established

vpi unsigned int [in] − For AAL0/AAL5 specifies the ATM vpi on which the VC is established. For OAM specifies the dedicated "OAM Tx channel" VPI.

vci unsigned int [in] − For AAL0/AAL5 specifies the ATM vci on which the VC is established. For OAM specifies the dedicated "OAM Tx channel" VCI.

connId **IxAtmConnId** [in] − specifies the IxAtmdAcc connection Id already associated with this VC

vcId int* [out] − pointer to a vcId

Returns:

◊ IX_SUCCESS the function is returning a Scheduler vcId for this VC

◊ IX_FAIL the function cannot process scheduling for this VC. the contents of vcId is unspecified

Definition at line **288** of file **IxAtmdAccCtrl.h**.

typedef void(* IxAtmdAccTxVcDemandClearCallback)(**IxAtmLogicalPort** port, int vcId)

prototype to remove all currently queued cells from a registered VC

IxAtmdAccTxVcDemandClearCallback is the prototype of the function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero. After the use of this callback, the scheduler shall not schedule more cells for this VC.

This callback function is called during a VC disconnection ***ixAtmdAccTxVcTryDisconnect()***

See also:

IxAtmdAccTxVcDemandUpdateCallback

IxAtmdAccTxVcDemandClearCallback

IxAtmdAccTxSchVcIdGetCallback

ixAtmdAccPortTxScheduledModeEnable

ixAtmdAccTxVcTryDisconnect

Parameters:

port

IxAtmLogicalPort [in] − Specifies the ATM port on which the VC to be cleared is established

vcId int [in] − Identifies the VC to be cleared. This is the value returned by the ***IxAtmdAccTxSchVcIdGetCallback()*** call .

Returns:

none

Definition at line **253** of file **IxAtmdAccCtrl.h**.

typedef IX_STATUS(* IxAtmdAccTxVcDemandUpdateCallback)(**IxAtmLogicalPort** port, int vcId, unsigned int numberOfCells)

Prototype to submit cells for transmission.

IxAtmdAccTxVcDemandUpdateCallback is the prototype of the callback function used by AtmD to notify an ATM Scheduler that the user of a VC has submitted cells for transmission.

See also:

IxAtmdAccTxVcDemandUpdateCallback

IxAtmdAccTxVcDemandClearCallback

IxAtmdAccTxSchVcIdGetCallback

ixAtmdAccPortTxScheduledModeEnable

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the VC to be updated is established

vcId int [in] − Identifies the VC to be updated. This is the value returned by the ***IxAtmdAccTxSchVcIdGetCallback()*** call .

numberOfCells unsigned int [in] − Indicates how many ATM cells should be added to the queue for this VC.

Returns:

◊ IX_SUCCESS the function is registering the cell demand for this VC.

◊ IX_FAIL the function cannot register cell for this VC : the scheduler maybe overloaded or misconfigured

Definition at line **219** of file **IxAtmdAccCtrl.h**.

Function Documentation

ixAtmdAccPortDisable (**IxAtmLogicalPort** *port*)

disable a PHY logical port

This function disable the transmission over one port.

When a port is disabled, the cell transmission to the Utopia interface is stopped.

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

Returns:

◊ IX_SUCCESS disable is complete

◊ IX_ATMDACC_WARNING port already disabled

◊ IX_FAIL disable failed, wrong parameter .

Note:

− This function needs internal locks and should not be called from an interrupt context

− The response from hardware is done through the txDone mechanism to ensure the synchrnisation with Tx resources. Therefore, the txDone mechanism needs to be serviced to make a PortDisable complete.

See also:

ixAtmdAccPortEnable

ixAtmdAccPortDisableComplete

ixAtmdAccTxDoneDispatch

ixAtmdAccPortDisableComplete (**IxAtmLogicalPort** *port*)

disable a PHY logical port

This function indicates if the port disable for a port has completed. This function will return TRUE if the port has never been enabled.

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

Returns:

◊ TRUE disable is complete

◊ FALSE disable failed, wrong parameter .

Note:

− This function needs internal locks and should not be called from an interrupt context

See also:

ixAtmdAccPortEnable

ixAtmdAccPortDisable

ixAtmdAccPortEnable (**IxAtmLogicalPort** *port*)

enable a PHY logical port

This function enables the transmission over one port. It should be called before accessing any resource from this port and before the establishment of a VC.

When a port is enabled, the cell transmission to the Utopia interface is started. If there is no traffic already running, idle cells are sent over the interface.

This function can be called multiple times.

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

Returns:

◊ IX_SUCCESS enable is complete

◊ IX_ATMDACC_WARNING port already enabled

◊ IX_FAIL enable failed, wrong parameter, or cannot initialise this port (the port is maybe already in use, or there is a hardware issue)

Note:

− This function needs internal locks and should not be called from an interrupt context

See also:

ixAtmdAccPortDisable

ixAtmdAccPortTxCallbackRegister (**IxAtmLogicalPort** *port*,
unsigned int *numberOfCells*,
IxAtmdAccPortTxLowCallback *callback*
)

Configure the Tx port threshold value and register a callback to handle threshold notifications.

This function sets the threshold in cells

See also:

ixAtmdAccPortTxCallbackRegister

ixAtmdAccPortTxProcess

ixAtmdAccPortTxFreeEntriesQuery

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

numberOfCells unsigned int [in] − threshold value which triggers the callback invocation, This number has to be one of the values 0,1,2,4,8,16,32 .... The maximum value cannot be more than half of the txVc queue size (which can be retrieved using ***ixAtmdAccPortTxFreeEntriesQuery()*** before any Tx traffic is sent for this port)

callback **IxAtmdAccPortTxLowCallback** [in] − callback function to invoke when the threshold level is reached. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS Successful call to ***ixAtmdAccPortTxCallbackRegister()***

◊ IX_FAIL error in the parameters, Tx channel already set for this port threshold level is not correct or within the range regarding the queue size:or unspecified error during processing:

Note:

− This callback function get called when the threshold level drops from (numberOfCells+1) cells to (numberOfCells) cells

− This function should be called during system initialisation, outside an interrupt context

ixAtmdAccPortTxCallbackUnregister (**IxAtmLogicalPort** *port*)

Unregister a callback to handle threshold notifications.

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

Returns:

◊ none

Note:

− This function should be called during system un−initialisation, outside an interrupt context

ixAtmdAccPortTxFreeEntriesQuery (**IxAtmLogicalPort** *port*,
unsigned int * *numberOfCellsPtr*
)

Get the number of available cells the system can accept for transmission.

The function is used to retrieve the number of cells that can be queued for transmission to the hardware.

This number is based on the worst schedule table where one cell is stored in one schedule table entry, depending on the pdus size and mbuf size and fragmentation.

This function doesn't use system resources and can be used from a timer context, or can be associated with a threshold event, or can be used inside an active polling mechanism

Parameters:

port	**IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]
numberOfCellsPtr	unsigned int* [out] − number of available cells. This parameter cannot be a null pointer.

See also:

ixAtmdAccPortTxProcess

Returns:

◊ IX_SUCCESS numberOfCellsPtr contains the number of cells that can be scheduled for this port.

◊ IX_FAIL error in the parameters, or some processing error occured.

```
ixAtmdAccPortTxProcess ( IxAtmLogicalPort       port,
                         IxAtmScheduleTable *   scheduleTablePtr
                       )
```

Transmit queue cells to the H/W based on the supplied schedule table.

This function ***ixAtmdAccPortTxProcess()*** process the schedule table provided as a parameter to the function. As a result cells are sent to the underlaying hardware for transmission.

The schedule table is executed in its entirety or not at all. So the onus is on the caller not to submit a table containing more cells than can be transmitted at that point. The maximum numbers that can be transmitted is guaranteed to be the number of cells as returned by the function ***ixAtmdAccPortTxFreeEntriesQuery()***.

When the scheduler is invoked on a threshold level, IxAtmdAcc gives the minimum number of cells (to ensure the callback will fire again later) and the maximum number of cells that ***ixAtmdAccPortTxProcess()*** will be able to process (assuming the ATM scheduler is able to produce the worst−case schedule table, i.e. one entry per cell).

When invoked ouside a threshold level, the overall number of cells of the schedule table should be less than the number of cells returned by the ***ixAtmdAccPortTxFreeEntriesQuery()*** function.

After invoking the ***ixAtmdAccPortTxProcess()*** function, it is the user choice to query again the queue level with the function ***ixAtmdAccPortTxFreeEntriesQuery()*** and, depending on a new cell number, submit an other schedule table.

IxAtmdAcc will check that the number of cells in the schedule table is compatible with the current transmit level. If the

Obsolete or invalid connection Id will be silently discarded.

This function is not reentrant for the same port.

This functions doesn't use system resources and can be used inside an interrupt context.

This function is used as a response to the hardware requesting more cells to transmit.

See also:

ixAtmdAccPortTxScheduledModeEnable

ixAtmdAccPortTxFreeEntriesQuery

ixAtmdAccPortTxCallbackRegister

ixAtmdAccPortEnable

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

scheduleTablePtr **IxAtmScheduleTable*** [in] − pointer to a scheduler update table. The content of this table is not modified by this function. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the schedule table process is complete and cells are transmitted to the hardware

◊ IX_ATMDACC_WARNING : Traffic will be dropped: the schedule table exceed the hardware capacity If this error is ignored, further traffic and schedule will work correctly. Overscheduling does not occur when the schedule table does not contain more entries that the number of free entries returned by ***ixAtmdAccPortTxFreeEntriesQuery()***. However, Disconnect attempts just after this error will fail permanently with the error code *IX_ATMDACC_RESOURCES_STILL_ALLOCATED*, and it is necessary to disable the port to make ***ixAtmdAccTxVcTryDisconnect()*** successful.

◊ IX_FAIL a wrong parameter is supplied, or the format of the schedule table is invalid, or the port is not Enabled, or an internal severe error occured. No cells is transmitted to the hardware

Note:

− If the failure is linked to an overschedule of data cells the result is an inconsistency in the output traffic (one or many cells may be missing and the traffic contract is not respected).

ixAtmdAccPortTxScheduledModeDisable (**IxAtmLogicalPort** *port*)

Put the port into UnScheduled Mode.

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

Returns:

◊ IX_SUCCESS scheduler unregistration is complete and the port is now in unscheduled mode.

◊ IX_FAIL failed (wrong parameters)

```
ixAtmdAccPortTxScheduledModeEnable ( IxAtmLogicalPort                    port,
                                     IxAtmdAccTxVcDemandUpdateCallback   vcDemandUpdateCallback,
                                     IxAtmdAccTxVcDemandClearCallback    vcDemandClearCallback,
                                     IxAtmdAccTxSchVcIdGetCallback       vcIdGetCallback
                                   )
```

Put the port into Scheduled Mode.

This function puts the specified port into scheduled mode of transmission which means an external s/w entity controls the transmission of cells on this port. This faciltates traffic shaping on the port.

Any buffers submitted on a VC for this port will be queued in IxAtmdAcc. The transmission of these buffers to and by the hardware will be driven by a transmit schedule submitted regulary in calls to ***ixAtmdAccPortTxProcess()*** by traffic shaping entity.

The transmit schedule is expected to be dynamic in nature based on the demand in cells for each VC on the port. Hence the callback parameters provided to this function allow IxAtmdAcc to inform the shaping entity of demand changes for each VC on the port.

By default a port is in Unscheduled Mode so if this function is not called, transmission of data is done without sheduling rules, on a first−come, first−out basis.

Once a port is put in scheduled mode it cannot be reverted to un−scheduled mode.

Note:

− This function should be called before any VCs have be connected on a port. Otherwise this function call will return failure.

− This function uses internal locks and should not be called from an interrupt context

See also:

IxAtmdAccTxVcDemandUpdateCallback

IxAtmdAccTxVcDemandClearCallback

IxAtmdAccTxSchVcIdGetCallback

ixAtmdAccPortTxProcess

Parameters:

port **IxAtmLogicalPort** [in] − logical PHY port [*IX_UTOPIA_PORT_0* .. *IX_UTOPIA_MAX_PORTS* − 1]

vcDemandUpdateCallback **IxAtmdAccTxVcDemandUpdateCallback** [in] − callback function used to update the number of outstanding cells for transmission. This

parameter cannot be a null pointer.

vcDemandClearCallback **IxAtmdAccTxVcDemandClearCallback** [in] − callback function used to remove all clear the number of outstanding cells for a VC. This parameter cannot be a null pointer.

vcIdGetCallback **IxAtmdAccTxSchVcIdGetCallback** [in] − callback function used to exchange vc Identifiers between IxAtmdAcc and the entity supplying the transmit schedule. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS scheduler registration is complete and the port is now in scheduled mode.

◊ IX_FAIL failed (wrong parameters, or traffic is already enabled on this port, possibly without ATM shaping)

```
ixAtmdAccRxDispatch ( IxAtmRxQueueId  rxQueueId,
                      unsigned int    numberOfPdusToProcess,
                      unsigned int *  numberOfPdusProcessedPtr
                    )
```

Control function which executes Rx processing for a particular Rx stream.

The *IxAtmdAccRxDispatch()* function is used to process received Pdus available from one of the two incoming Rx streams. When this function is invoked, the incoming traffic (up to the number of PDUs passed as a parameter) will be transferred to the IxAtmdAcc users through the callback ***IxAtmdAccRxVcRxCallback()***, as registered during the ***ixAtmdAccRxVcConnect()*** call.

The user receive callbacks will be executed in the context of this function.

Failing to use this function on a regular basis when there is traffic will block incoming traffic and can result in Pdus being dropped by the hardware.

This should be used to control when received pdus are handed off from the hardware to Aal users from a particluar stream. The function can be used from a timer context, or can be registered as a callback in response to an Rx stream threshold event, or can be used inside an active polling mechanism which is under user control.

Note:

− The signature of this function is directly compatible with the callback prototype which can be register with ***ixAtmdAccRxDispatcherRegister()***.

See also:

ixAtmdAccRxDispatcherRegister

IxAtmdAccRxVcRxCallback

ixAtmdAccRxVcFreeEntriesQuery

Parameters:

rxQueueId **IxAtmRxQueueId** [in] − indicates which Rx queue to process.

numberOfPdusToProcess unsigned int [in] − indicates the maxiumum number of PDU to remove from the Rx queue. A value of IX_ATMDACC_ALLPDUS indicates to process all PDUs from the queue. This includes at least the PDUs in the queue when the fuction is invoked. Because of real−time constraints, there is no guarantee thatthe queue will be empty when the function exits. If this parameter is greater than the number of entries of the queues, the function will succeed and the parameter numberOfPdusProcessedPtr will reflect the exact number of PDUs processed.

**numberOfPdusProcessedPtr* unsigned int [out] − indicates the actual number of PDU processed during this call. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the number of PDUs as indicated in numberOfPdusProcessedPtr are removed from the Rx queue and the VC callback are called.

◊ IX_FAIL invalid parameters or some unspecified internal error occured.

ixAtmdAccRxDispatcherRegister (**IxAtmRxQueueId** *queueId*,
IxAtmdAccRxDispatcher *callback*
)

Register a notification callback to be invoked when there is at least one entry on a particular Rx queue.

This function registers a callback to be invoked when there is at least one entry in a particular queue. The registered callback is called every time when the hardware adds one or more pdus to the specified Rx queue.

This function cannot be used when a Rx Vc using this queue is already existing.

Note:

−The callback function can be the API function ***ixAtmdAccRxDispatch()*** : every time the threhold level of the queue is reached, the **ixAtmdAccRxDispatch()** is invoked to remove all entries from the queue.

See also:

ixAtmdAccRxDispatch

IxAtmdAccRxDispatcher

Parameters:

queueId **IxAtmRxQueueId** [in] Rx queue identification

callback **IxAtmdAccRxDispatcher** [in] function triggering the delivery of incoming traffic. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS Successful call to ***ixAtmdAccRxDispatcherRegister()***

◊ IX_FAIL error in the parameters, or there is an already active Rx VC for this queue or

some unspecified internal error occurred.

ixAtmdAccRxDispatcherUnregister (**IxAtmRxQueueId** *atmdQueueId*)

UnRegister a notification callback to be invoked when there is at least one entry on a particular Rx queue.

Parameters:

queueId **IxAtmRxQueueId** [in] Rx queue identification

Returns:

◊ IX_SUCCESS Successful call

◊ IX_FAIL error in the parameters

ixAtmdAccRxLevelQuery (**IxAtmRxQueueId** *rxQueueId*,
unsigned int * *numberOfPdusPtr*
)

Query the number of entries in a particular Rx queue.

This function is used to retrieve the number of pdus received by the hardware and ready for distribution to users.

Parameters:

rxQueueId **IxAtmRxQueueId** [in] − indicates which of two Rx queues to query.

numberOfPdusPtr unsigned int* [out] − Pointer to store the number of available PDUs in the Rx queue. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the value in numberOfPdusPtr specifies the number of incoming pdus waiting in this queue

◊ IX_FAIL an error occurs during processing. The value in numberOfPdusPtr is unspecified.

Note:

− This function is reentrant, doesn't use system resources and can be used from an interrupt context.

ixAtmdAccRxQueueSizeQuery (**IxAtmRxQueueId** *rxQueueId*,
unsigned int * *numberOfPdusPtr*
)

Query the size of a particular Rx queue.

This function is used to retrieve the number of pdus the system is able to queue when reception is complete.

Parameters:

rxQueueId **IxAtmRxQueueId** [in] − indicates which of two Rx queues to query.

numberOfPdusPtr unsigned int* [out] − Pointer to store the number of pdus the system is able to queue in the Rx queue. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the value in numberOfPdusPtr specifies the number of pdus the system is able to queue.

◊ IX_FAIL an error occurs during processing. The value in numberOfPdusPtr is unspecified.

Note:

− This function is reentrant, doesn't use system resources and can be used from an interrupt context.

```
ixAtmdAccTxDoneDispatch ( unsigned int   numberOfPdusToProcess,
                          unsigned int * numberOfPdusProcessedPtr
                        )
```

Process a number of pending transmit done pdus from the hardware.

As a by−product of Atm transmit operation buffers which transmission is complete need to be recycled to users. This function is invoked to service the oustanding list of transmitted buffers and pass them to VC users.

Users are handed back pdus by invoking the free callback registered during the ***ixAtmdAccTxVcConnect()*** call.

There is a single Tx done stream servicing all active Atm Tx ports which can contain a maximum of 64 entries. If this stream fills port transmission will stop so this function must be call sufficently frequently to ensure no disruption to the transmit operation.

This function can be used from a timer context, or can be associated with a TxDone level threshold event (see ***ixAtmdAccTxDoneDispatcherRegister()***), or can be used inside an active polling mechanism under user control.

For ease of use the signature of this function is compatible with the TxDone threshold event callback prototype.

This functions can be used inside an interrupt context.

See also:

ixAtmdAccTxDoneDispatcherRegister

IxAtmdAccTxVcBufferReturnCallback

ixAtmdAccTxDoneLevelQuery

Parameters:

numberOfPdusToProcess unsigned int [in] − maxiumum number of pdus to remove from the Tx Done queue

**numberOfPdusProcessedPtr* unsigned int [out] − number of pdus removed from the Tx Done

queue. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS the number of pdus as indicated in numberOfPdusToProcess are removed from the Tx Done hardware and passed to the user through the Tx Done callback registered during a call to ***ixAtmdAccTxVcConnect()***

◊ IX_FAIL invalid parameters or numberOfPdusProcessedPtr is a null pointer or some unspecified internal error occured.

ixAtmdAccTxDoneDispatcherRegister (unsigned int *numberOfPdus*,
IxAtmdAccTxDoneDispatcher *notificationCallback*
)

Configure the Tx Done stream threshold value and register a callback to handle threshold notifications.

This function sets the threshold level in term of number of pdus at which the supplied notification function should be called.

The higher the threshold value is, the less events will be necessary to process transmitted buffers.

Transmitted buffers recycling implementation is a sytem−wide mechanism and needs to be set prior any traffic is started. If this threshold mechanism is not used, the user is responsible for polling the transmitted buffers thanks to ***ixAtmdAccTxDoneDispatch()*** and ***ixAtmdAccTxDoneLevelQuery()*** functions.

This function should be called during system initialisation outside an interrupt context

See also:

ixAtmdAccTxDoneDispatcherRegister

ixAtmdAccTxDoneDispatch

ixAtmdAccTxDoneLevelQuery

Parameters:

numberOfPdus	unsigned int [in] − The number of TxDone pdus which triggers the callback invocation This number has to be a power of 2, one of the values 0,1,2,4,8,16,32 ... The maximum value cannot be more than half of the txDone queue size (which can be retrieved using ***ixAtmdAccTxDoneQueueSizeQuery()***)
notificationCallback	**IxAtmdAccTxDoneDispatcher** [in] − The function to invoke. (This parameter can be ***ixAtmdAccTxDoneDispatch()***).This parameter ust not be a null pointer.

Returns:

◊ IX_SUCCESS Successful call to ixAtmdAccTxDoneDispatcherRegister

◊ IX_FAIL error in the parameters:

Note:

− The notificationCallback will be called exactly when the threshold level will increase from (numberOfPdus) to (numberOfPdus+1)

− If there is no Tx traffic, there is no guarantee that TxDone Pdus will be released to the user (when txDone level is permanently under the threshold level. One of the preffered way to return resources to the user is to use a mix of txDone notifications, used together with a slow rate timer and an exclusion mechanism protecting from re−entrancy

− The TxDone threshold will only hand back buffers when the threshold level is crossed. Setting this threshold to a great number reduce the interrupt rate and the cpu load, but also increase the number of outstanding mbufs and has a system wide impact when these mbufs are needed by other components.

ixAtmdAccTxDoneDispatcherUnregister (void)

Unregister a callback to handle threshold notifications.

Parameters:

NONE.

Returns:

NONE

ixAtmdAccTxDoneLevelQuery (unsigned int * *numberOfPdusPtr*)

Query the current number of transmit pdus ready for recycling.

This function is used to get the number of transmitted pdus which the hardware is ready to hand back to user.

This function can be used from a timer context, or can be associated with a threshold event, on can be used inside an active polling mechanism

See also:

ixAtmdAccTxDoneDispatch

Parameters:

**numberOfPdusPtr*	unsigned int [out] − Pointer to the number of pdus transmitted at the time of this function call, and ready for recycling This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS numberOfPdusPtr contains the number of pdus ready for recycling at the time of this function call

◊ IX_FAIL wrong parameter (null pointer as parameter).or unspecified rocessing error occurs..The value in numberOfPdusPtr is unspecified.

ixAtmdAccTxDoneQueueSizeQuery (unsigned int * *numberOfPdusPtr*)

Query the TxDone queue size.

This function is used to get the number of pdus which the hardware is able to store after transmission is complete

The returned value can be used to set a threshold and enable a callback to be notified when the number of pdus is going over the threshold.

See also:

ixAtmdAccTxDoneDispatcherRegister

Parameters:

**numberOfPdusPtr* unsigned int [out] − Pointer to the number of pdus the system is able to queue after transmission

Returns:

◊ IX_SUCCESS numberOfPdusPtr contains the the number of pdus the system is able to queue after transmission

◊ IX_FAIL wrong parameter (null pointer as parameter).or unspecified rocessing error occurs..The value in numberOfPdusPtr is unspecified.

Note:

− This function is reentrant, doesn't use system resources and can be used from an interrupt context.

ixAtmdAccUtopiaConfigReset (const **IxAtmdAccUtopiaConfig** * *utConfig*)

Reset the configuration structure to the Utopia interface initializes configuration registers and write to NPE−A Load and get response from NPE−A and resets the Utopia interface.

Parameters:

ixAtmdAccNPEConfigPtr* **IxAtmdAccUtopiaConfig [in] − pointer to a structure to download to

Returns:

◊ IX_SUCCESS successful unload

◊ IX_FAIL error in the parameters

ixAtmdAccUtopiaConfigSet (const **IxAtmdAccUtopiaConfig** * *ixAtmdAccUtopiaConfigPtr*)

Send the configuration structure to the Utopia interface.

This function downloads the ***IxAtmdAccUtopiaConfig*** structure to the Utopia and has the following effects

- setup the Utopia interface
- initialise the NPE
- reset the Utopia cell counters and status registers to known values

This action has to be done once at initialisation. A lock is preventing the concurrent use of ***ixAtmdAccUtopiaStatusGet()*** and **ixAtmdAccUtopiaConfigSet()**

Parameters:

ixAtmdAccNPEConfigPtr* **IxAtmdAccUtopiaConfig [in] − pointer to a structure to download to Utopia. This parameter cannot be a null pointer.

Returns:

◊ IX_SUCCESS successful download

◊ IX_FAIL error in the parameters, or configuration is not complete or failed

See also:

ixAtmdAccUtopiaStatusGet

ixAtmdAccUtopiaStatusGet (**IxAtmdAccUtopiaStatus** * *ixAtmdAccUtopiaStatus*)

Get the Utopia interface configuration.

This function reads the Utopia registers and the Cell counts and fills the ***IxAtmdAccUtopiaStatus*** structure

A lock is preventing the concurrent use of ***ixAtmdAccUtopiaStatusGet()*** and **ixAtmdAccUtopiaConfigSet()**

Parameters:

ixAtmdAccUtopiaStatus **IxAtmdAccUtopiaStatus** [out] − pointer to structure to be updated from internal hardware counters. This parameter cannot be a NULL pointer.

Returns:

◊ IX_SUCCESS successful read

◊ IX_FAIL error in the parameters null pointer, or configuration read is not complete or failed

See also:

ixAtmdAccUtopiaConfigSet

Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Control API]

The public API for the IXP400 Atm Driver Control component.

Data Structures

struct **IxAtmdAccUtopiaConfig**
Utopia configuration.

struct **IxAtmdAccUtopiaConfig::UtRxConfig_**
Utopia Rx config Register.

struct **IxAtmdAccUtopiaConfig::UtRxDefineIdle_**
Utopia Rx idle cells config Register.

struct **IxAtmdAccUtopiaConfig::UtRxEnableFields_**
Utopia Rx enable Register.

struct **IxAtmdAccUtopiaConfig::UtRxStatsConfig_**
Utopia Rx stats config Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable0_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable1_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable2_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable3_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable4_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable5_**
Utopia Rx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtSysConfig_**
NPE setup Register.

struct **IxAtmdAccUtopiaConfig::UtTxConfig_**
Utopia Tx Config Register.

struct **IxAtmdAccUtopiaConfig::UtTxDefineIdle_**
Utopia Tx idle cells Register.

struct **IxAtmdAccUtopiaConfig::UtTxEnableFields_**
Utopia Tx ienable fields Register.

struct **IxAtmdAccUtopiaConfig::UtTxStatsConfig_**
Utopia Tx stats Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable0_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable1_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable2_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable3_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable4_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable5_**
Utopia Tx translation table Register.

struct **IxAtmdAccUtopiaStatus**
Utopia status.

struct **IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_**
Utopia Rx Status Register.

struct **IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_**
Utopia Tx Status Register.

Detailed Description

The public API for the IXP400 Atm Driver Control component.

IxAtmdAcc is the low level interface by which AAL PDU get transmitted to,and received from the Utopia bus

This part is related to the UTOPIA configuration.

Intel (R) IXP400 Software ATM Manager (IxAtmm) API

IXP400 ATM Manager component Public API.

Data Structures

struct **IxAtmmPortCfg**
Structure contains port−specific information required to initialize IxAtmm, and specifically, the IXP400 UTOPIA Level−2 device.

struct **IxAtmmVc**
This structure describes the required attributes of a virtual connection.

Defines

#define **IX_ATMM_RET_ALREADY_INITIALIZED**
Component has already been initialized.

#define **IX_ATMM_RET_INVALID_PORT**
Specified port does not exist or is out of range.

#define **IX_ATMM_RET_INVALID_VC_DESCRIPTOR**
The VC description does not adhere to ATM standards.

#define **IX_ATMM_RET_VC_CONFLICT**
The VPI/VCI values supplied are either reserved, or they conflict with a previously registered VC on this port.

#define **IX_ATMM_RET_PORT_CAPACITY_IS_FULL**
The virtual connection cannot be established on the port because the remaining port capacity is not sufficient to support it.

#define **IX_ATMM_RET_NO_SUCH_VC**
No registered VC, as described by the supplied VCI/VPI or VC identifier values, exists on this port.

#define **IX_ATMM_RET_INVALID_VC_ID**
The specified VC identifier is out of range.

#define **IX_ATMM_RET_INVALID_PARAM_PTR**
A pointer parameter was NULL.

#define **IX_ATMM_UTOPIA_SPHY_ADDR**
The phy address when in SPHY mode.

#define **IX_ATMM_THREAD_PRI_HIGH**

The value of high priority thread.

Typedefs

typedef void(* **IxAtmmVcChangeCallback**)(**IxAtmmVcChangeEvent** eventType, **IxAtmLogicalPort** port, const **IxAtmmVc** *vcChanged)

*Callback type used with **ixAtmmVcChangeCallbackRegister** interface Defines a callback type which will be used to notify registered users of registration/deregistration events on a particular port.*

Enumerations

enum **IxAtmmVcDirection** {
IX_ATMM_VC_DIRECTION_TX,
IX_ATMM_VC_DIRECTION_RX,
IX_ATMM_VC_DIRECTION_INVALID
}

*Definition for use in the **IxAtmmVc** structure. Indicates the direction of a VC.*

enum **IxAtmmVcChangeEvent** {
IX_ATMM_VC_CHANGE_EVENT_REGISTER,
IX_ATMM_VC_CHANGE_EVENT_DEREGISTER,
IX_ATMM_VC_CHANGE_EVENT_INVALID
}

*Definition for use with **IxAtmmVcChangeCallback** callback. Indicates that the event type represented by the callback for this VC.*

enum **IxAtmmUtopiaLoopbackMode** {
IX_ATMM_UTOPIA_LOOPBACK_DISABLED,
IX_ATMM_UTOPIA_LOOPBACK_ENABLED,
IX_ATMM_UTOPIA_LOOPBACK_INVALID
}

Definitions for use with interface to indicate that UTOPIA loopback should be enabled or disabled on initialisation.

enum **IxAtmmPhyMode** {
IX_ATMM_MPHY_MODE,
IX_ATMM_SPHY_MODE,
IX_ATMM_PHY_MODE_INVALID
}

*Definitions for use with **ixAtmmUtopiaInit** interface to indicate that UTOPIA multi−phy/single−phy mode is used.*

Functions

PUBLIC
IX_STATUS **ixAtmmInit** (void)

Interface to initialize the IxAtmm software component. Can be called once only.

PUBLIC
IX_STATUS **ixAtmmUninit** (void)

Interface to uninitialize the IxAtmm software component. This function deallocates/destroys the allocated buffer/Mutex done during initialization Should be the last function to be called.

PUBLIC**ixAtmmUtopiaInit** (unsigned numPorts, **IxAtmmPhyMode** phyMode, **IxAtmmPortCfg**
IX_STATUS portCfgs[], **IxAtmmUtopiaLoopbackMode** loopbackMode)

Interface to initialize the UTOPIA Level−2 ATM coprocessor for the specified number of physical ports. The function must be called before the ixAtmmPortInitialize interface can operate successfully.

PUBLIC
IX_STATUS **ixAtmmUtopiaUninit** (void)

Interface to uninitialize the UTOPIA Level−2 ATM coprocessor.

PUBLIC
IX_STATUS **ixAtmmPortInitialize** (**IxAtmLogicalPort** port, unsigned txPortRate, unsigned rxPortRate)

*The interface is called following **ixAtmmUtopiaInit** () and before calls to any other IxAtmm interface. It serves to activate the registered ATM port with IxAtmm.*

PUBLIC
IX_STATUS **ixAtmmPortUninitialize** (**IxAtmLogicalPort** port)

It serves to uninitialise the respective port passed as a parameter. Excutes only if Utopia Intialization is done. Uninitializes the Data Path. Unintiializes the specified port.

PUBLIC
IX_STATUS **ixAtmmPortModify** (**IxAtmLogicalPort** port, unsigned txPortRate, unsigned rxPortRate)

A client may call this interface to change the existing port rate (expressed in bits/second) on an established ATM port.

PUBLIC
IX_STATUS **ixAtmmPortQuery** (**IxAtmLogicalPort** port, unsigned *txPortRate, unsigned *rxPortRate)

The client may call this interface to request details on currently registered transmit and receive rates for an ATM port.

PUBLIC
IX_STATUS **ixAtmmPortEnable** (**IxAtmLogicalPort** port)

The client call this interface to enable transmit for an ATM port. At initialisation, all the ports are disabled.

PUBLIC
IX_STATUS **ixAtmmPortDisable** (**IxAtmLogicalPort** port)

The client call this interface to disable transmit for an ATM port. At initialisation, all the ports are disabled.

PUBLIC **ixAtmmVcRegister** (**IxAtmLogicalPort** port, **IxAtmmVc** *vcToAdd,
IX_STATUS **IxAtmSchedulerVcId** *vcId)
This interface is used to register an ATM Virtual Connection on the specified ATM port.

PUBLIC
IX_STATUS **ixAtmmVcDeregister** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId)
Function called by a client to deregister a VC from the system.

PUBLIC **ixAtmmVcQuery** (**IxAtmLogicalPort** port, unsigned vpi, unsigned vci,
IX_STATUS **IxAtmmVcDirection** direction, **IxAtmSchedulerVcId** *vcId, **IxAtmmVc** *vcDesc)
This interface supplies information about an active VC on a particular port when supplied with the VPI, VCI and direction of that VC.

PUBLIC **ixAtmmVcIdQuery** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId, **IxAtmmVc**
IX_STATUS *vcDesc)
This interface supplies information about an active VC on a particular port when supplied with a vcId for that VC.

PUBLIC
IX_STATUS **ixAtmmVcChangeCallbackRegister** (**IxAtmmVcChangeCallback** callback)
This interface is invoked to supply a function to IxAtmm which will be called to notify the client if a new VC is registered with IxAtmm or an existing VC is removed.

PUBLIC
IX_STATUS **ixAtmmVcChangeCallbackDeregister** (**IxAtmmVcChangeCallback** callback)
This interface is invoked to deregister a previously supplied callback function.

PUBLIC
IX_STATUS **ixAtmmUtopiaStatusShow** (void)
Display utopia status counters.

PUBLIC
IX_STATUS **ixAtmmUtopiaCfgShow** (void)
Display utopia information(config registers and status registers).

Detailed Description

IXP400 ATM Manager component Public API.

Define Documentation

```
#define IX_ATMM_RET_ALREADY_INITIALIZED
```

Component has already been initialized.

Definition at line **40** of file **IxAtmm.h**.

#define IX_ATMM_RET_INVALID_PARAM_PTR

A pointer parameter was NULL.

Definition at line **86** of file **IxAtmm.h**.

#define IX_ATMM_RET_INVALID_PORT

Specified port does not exist or is out of range.

Definition at line **46** of file **IxAtmm.h**.

#define IX_ATMM_RET_INVALID_VC_DESCRIPTOR

The VC description does not adhere to ATM standards.

Definition at line **52** of file **IxAtmm.h**.

#define IX_ATMM_RET_INVALID_VC_ID

The specified VC identifier is out of range.

Definition at line **80** of file **IxAtmm.h**.

#define IX_ATMM_RET_NO_SUCH_VC

No registered VC, as described by the supplied VCI/VPI or VC identifier values, exists on this port.

Definition at line **74** of file **IxAtmm.h**.

#define IX_ATMM_RET_PORT_CAPACITY_IS_FULL

The virtual connection cannot be established on the port because the remaining port capacity is not sufficient to support it.

Definition at line **67** of file **IxAtmm.h**.

#define IX_ATMM_RET_VC_CONFLICT

The VPI/VCI values supplied are either reserved, or they conflict with a previously registered VC on this port.

Definition at line **59** of file **IxAtmm.h**.

#define IX_ATMM_THREAD_PRI_HIGH

The value of high priority thread.

Definition at line **98** of file **IxAtmm.h**.

#define IX_ATMM_UTOPIA_SPHY_ADDR

The phy address when in SPHY mode.

Definition at line **92** of file **IxAtmm.h**.

Typedef Documentation

typedef void(* IxAtmmVcChangeCallback)(**IxAtmmVcChangeEvent** eventType, **IxAtmLogicalPort** port, const **IxAtmmVc*** vcChanged)

Callback type used with **ixAtmmVcChangeCallbackRegister** interface Defines a callback type which will be used to notify registered users of registration/deregistration events on a particular port.

Parameters:

eventType	**IxAtmmVcChangeEvent** [in] − Event indicating whether the VC supplied has been added or removed
port	**IxAtmLogicalPort** [in] − Specifies the port on which the event has occurred
vcChanged	**IxAtmmVc*** [in] − Pointer to a structure which gives details of the VC which has been added or removed on the port

Definition at line **189** of file **IxAtmm.h**.

Enumeration Type Documentation

enum IxAtmmPhyMode

Definitions for use with **ixAtmmUtopiaInit** interface to indicate that UTOPIA multi−phy/single−phy mode is used.

Enumeration values:

IX_ATMM_MPHY_MODE	Atmm phy mode mphy.
IX_ATMM_SPHY_MODE	Atmm phy mode sphy.
IX_ATMM_PHY_MODE_INVALID	Atmm phy mode invalid.

Definition at line **150** of file **IxAtmm.h**.

enum IxAtmmUtopiaLoopbackMode

Definitions for use with interface to indicate that UTOPIA loopback should be enabled or disabled on initialisation.

Enumeration values:

IX_ATMM_UTOPIA_LOOPBACK_DISABLED	Atmm Utopia loopback mode disabled.
IX_ATMM_UTOPIA_LOOPBACK_ENABLED	Atmm Utopia loopback mode enabled.
IX_ATMM_UTOPIA_LOOPBACK_INVALID	Atmm Utopia loopback mode invalid.

Definition at line **126** of file **IxAtmm.h**.

enum IxAtmmVcChangeEvent

Definition for use with **IxAtmmVcChangeCallback** callback. Indicates that the event type represented by the callback for this VC.

Enumeration values:

IX_ATMM_VC_CHANGE_EVENT_REGISTER	Atmm Vc event register.
IX_ATMM_VC_CHANGE_EVENT_DEREGISTER	Atmm Vc event de−register.
IX_ATMM_VC_CHANGE_EVENT_INVALID	Atmm Vc event invalid.

Definition at line **116** of file **IxAtmm.h**.

enum IxAtmmVcDirection

Definition for use in the **IxAtmmVc** structure. Indicates the direction of a VC.

Enumeration values:

IX_ATMM_VC_DIRECTION_TX	Atmm Vc direction transmit.
IX_ATMM_VC_DIRECTION_RX	Atmm Vc direction receive.
IX_ATMM_VC_DIRECTION_INVALID	Atmm Vc direction invalid.

Definition at line **106** of file **IxAtmm.h**.

Function Documentation

ixAtmmInit (void)

Interface to initialize the IxAtmm software component. Can be called once only.

Must be called before any other IxAtmm API is called.

Parameters:

none

Returns:

◊ IX_SUCCESS : IxAtmm has been successfully initialized. Calls to other IxAtmm interfaces may now be performed.

◊ IX_FAIL : IxAtmm has already been initialized.

ixAtmmPortDisable (**IxAtmLogicalPort** *port*)

The client call this interface to disable transmit for an ATM port. At initialisation, all the ports are disabled.

Parameters:

port **IxAtmLogicalPort** [in] − Value identifies the port

Returns:

◊ IX_SUCCESS : Transmission over this port is stopped.

◊ IX_FAIL : The port parameter is not valid, or the port is already disabled

Note:

− When a port is disabled, Rx and Tx VC Connect requests will fail

− This function call does not stop Rx traffic. It is supposed that this function is invoked when a serious problem is detected (e.g. physical layer broken). Then, the Rx traffic is not passing.

− This function is blocking until the hw acknowledge that the transmission is stopped.

− This function uses system resources and should not be used inside an interrupt context.

See also:

ixAtmmPortEnable

ixAtmmPortEnable (**IxAtmLogicalPort** *port*)

The client call this interface to enable transmit for an ATM port. At initialisation, all the ports are disabled.

Parameters:

port **IxAtmLogicalPort** [in] − Value identifies the port

Returns:

◊ IX_SUCCESS : Transmission over this port is started.

◊ IX_FAIL : The port parameter is not valid, or the port is already enabled

Note:

− When a port is disabled, Rx and Tx VC Connect requests will fail

− This function uses system resources and should not be used inside an interrupt context.

See also:

ixAtmmPortDisable

```
ixAtmmPortInitialize ( IxAtmLogicalPort port,
                       unsigned         txPortRate,
                       unsigned         rxPortRate
                     )
```

The interface is called following **ixAtmmUtopiaInit** () and before calls to any other IxAtmm interface. It serves to activate the registered ATM port with IxAtmm.

The transmit and receive port rates are specified in bits per second. This translates to ATM cells per second according to the following formula: CellsPerSecond = portRate / (53*8) The IXP400 device supports only 53 byte cells. The client shall make sure that the off−chip physical layer device has already been initialized.

IxAtmm will configure IxAtmdAcc and IxAtmSch to enable scheduling on the port.

This interface must be called once for each active port in the system. The first time the interface is invoked, it will configure the mechanism by which the handling of transmit, transmit−done and receive are driven with the IxAtmdAcc component.

This function is reentrant.

Note:

The minimum Tx rate that will be accepted is 424 bit/s which equates to 1 cell (53 bytes) per second.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies the port which is to be initialized.

txPortRate unsigned [in] − Value specifies the transmit port rate for this port in bits/second. This value is used by the ATM Scheduler component is evaluating VC access requests for the port.

rxPortRate unsigned [in] − Value specifies the receive port rate for this port in bits/second.

Returns:

◊ IX_SUCCESS : The specificed ATM port has been successfully initialized. IxAtmm is ready to accept VC registrations on this port.

◊ IX_ATMM_RET_ALREADY_INITIALIZED : ixAtmmPortInitialize has already been called successfully on this port. The current call is rejected.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid. The request is rejected.

◊ IX_FAIL : IxAtmm could not initialize the port because the inputs are not understood.

See also:

ixAtmmPortEnable, **ixAtmmPortDisable**

```
ixAtmmPortModify ( IxAtmLogicalPort port,
                   unsigned         txPortRate,
                   unsigned         rxPortRate
                 )
```

A client may call this interface to change the existing port rate (expressed in bits/second) on an established ATM port.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies the port which is to be initialized.

txPortRate unsigned [in] − Value specifies the`` transmit port rate for this port in bits/second. This value is used by the ATM Scheduler component is evaluating VC access requests for the port.

rxPortRate unsigned [in] − Value specifies the receive port rate for this port in bits/second.

Returns:

◊ IX_SUCCESS : The indicated ATM port rates have been successfully modified.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid. The request is rejected.

◊ IX_FAIL : IxAtmm could not update the port because the inputs are not understood, or the interface was called before the port was initialized.

```
ixAtmmPortQuery ( IxAtmLogicalPort port,
                  unsigned *       txPortRate,
                  unsigned *       rxPortRate
                )
```

The client may call this interface to request details on currently registered transmit and receive rates for an ATM port.

Parameters:

port **IxAtmLogicalPort** [in] − Value identifies the port from which the rate details are requested.

**txPortRate* unsigned [out] − Pointer to a value which will be filled with the value of the transmit port rate specified in bits/second.

**rxPortRate* unsigned [out] − Pointer to a value which will be filled with the value of the receive port rate specified in bits/second.

Returns:

◊ IX_SUCCESS : The information requested on the specified port has been successfully supplied in the output.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid. The request is rejected.

◊ IX_ATMM_RET_INVALID_PARAM_PTR : A pointer parameter was NULL.

◊ IX_FAIL : IxAtmm could not update the port because the inputs are not understood, or the interface was called before the port was initialized.

ixAtmmPortUninitialize (**IxAtmLogicalPort** *port*)

It serves to uninitialise the respective port passed as a parameter. Excutes only if Utopia Intialization is done. Uninitializes the Data Path. Unintiializes the specified port.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies the port which is to be uninitialized.

Returns:

◊ IX_SUCCESS : The specificed ATM port has been successfully uninitialized.

◊ IX_FAIL : IxAtmm could not uninitialize the port because the inputs are not understood or because of some other internal error.

ixAtmmUninit (void)

Interface to uninitialize the IxAtmm software component. This function deallocates/destroys the allocated buffer/Mutex done during initialization Should be the last function to be called.

Parameters:

none

Returns:

◊ IX_SUCCESS : IxAtmm has been successfully uninitialized. Calls to other IxAtmm interfaces cannot be performed after this.

◊ IX_FAIL : IxAtmm has already been uninitialized.

ixAtmmUtopiaCfgShow (void)

Display utopia information(config registers and status registers).

Parameters:

none

Returns:

◊ IX_SUCCESS : Show function was successful

◊ IX_FAIL : Internal failure

```
ixAtmmUtopiaInit ( unsigned                    numPorts,
                   IxAtmmPhyMode               phyMode,
                   IxAtmmPortCfg               portCfgs[],
                   IxAtmmUtopiaLoopbackMode    loopbackMode
                 )
```

Interface to initialize the UTOPIA Level−2 ATM coprocessor for the specified number of physical ports. The function must be called before the ixAtmmPortInitialize interface can operate successfully.

Parameters:

numPorts	unsigned [in] − Indicates the total number of logical ports that are active on the device. Up to 12 ports are supported.
phyMode	**IxAtmmPhyMode** [in] − Put the Utopia coprocessor in SPHY or MPHY mode.
portCfgs[]	**IxAtmmPortCfg** [in] − Pointer to an array of elements detailing the UTOPIA specific port characteristics. The length of the array must be equal to the number of ports activated. ATM ports are referred to by the relevant offset in this array in all subsequent IxAtmm interface calls.
loopbackMode	**IxAtmmUtopiaLoopbackMode** [in] − Value must be one of **IX_ATMM_UTOPIA_LOOPBACK_ENABLED** or **IX_ATMM_UTOPIA_LOOPBACK_DISABLED** indicating whether loopback should be enabled on the device. Loopback can only be supported on a single PHY, therefore the numPorts parameter must be 1 if loopback is enabled.

Returns:

- ◊ IX_SUCCESS : Indicates that the UTOPIA device has been successfully initialized for the supplied ports.
- ◊ IX_ATMM_RET_ALREADY_INITIALIZED : The UTOPIA device has already been initialized.
- ◊ IX_FAIL : The supplied parameters are invalid or have been rejected by the UTOPIA−NPE device.

Warning:

This interface may only be called once. Port identifiers are assumed to range from 0 to (numPorts − 1) in all instances. In all subsequent calls to interfaces supplied by IxAtmm, the specified port value is expected to represent the offset in the portCfgs array specified in this interface. i.e. The first port in this array will subsequently be represented as port 0, the second port as port 1, and so on.

```
ixAtmmUtopiaStatusShow ( void )
```

Display utopia status counters.

Parameters:

none

Returns:

- ◊ IX_SUCCESS : Show function was successful

◊ IX_FAIL : Internal failure

ixAtmmUtopiaUninit (void)

Interface to uninitialize the UTOPIA Level−2 ATM coprocessor.

Parameters:

none

Returns:

◊ IX_SUCCESS : Indicates that the UTOPIA device has been successfully uninitialized .

◊ IX_FAIL :Uninitialise of the UTOPIA−NPE device did not work.

ixAtmmVcChangeCallbackDeregister (**IxAtmmVcChangeCallback** *callback*)

This interface is invoked to deregister a previously supplied callback function.

Parameters:

callback **IxAtmmVcChangeCallback** [in] − Callback which complies with the **IxAtmmVcChangeCallback** definition. This function will removed from the table of callbacks.

Returns:

◊ IX_SUCCESS : The specified callback has been deregistered successfully from IxAtmm.

◊ IX_FAIL : Either the supplied callback is invalid, or is not currently registered with IxAtmm.

ixAtmmVcChangeCallbackRegister (**IxAtmmVcChangeCallback** *callback*)

This interface is invoked to supply a function to IxAtmm which will be called to notify the client if a new VC is registered with IxAtmm or an existing VC is removed.

The callback, when invoked, will run within the context of the call to **ixAtmmVcRegister** or **ixAtmmVcDeregister** which caused the change of state.

A maximum of 32 calbacks may be registered in with IxAtmm.

Parameters:

callback **IxAtmmVcChangeCallback** [in] − Callback which complies with the **IxAtmmVcChangeCallback** definition. This function will be invoked by IxAtmm with the appropiate parameters for the relevant VC when any VC has been registered or deregistered with IxAtmm.

Returns:

◊ IX_SUCCESS : The specified callback has been registered successfully with IxAtmm and will be invoked when appropriate.

◊ IX_FAIL : Either the supplied callback is invalid, or IxAtmm has already registered 32 and connot accommodate any further registrations of this type. The request is rejected.

Warning:

The client must not call either the **ixAtmmVcRegister** or **ixAtmmVcDeregister** interfaces from within the supplied callback function.

```
ixAtmmVcDeregister ( IxAtmLogicalPort      port,
                     IxAtmSchedulerVcId    vcId
                   )
```

Function called by a client to deregister a VC from the system.

With the removal of each new VC from a port, a series of registered callback functions are invoked by the IxAtmm component to notify possible external components of the change. The callback functions are registered using the **ixAtmmVcChangeCallbackRegister**.

The IxAtmSch component is notified of the removal of transmit VCs.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies port on which the VC to be removed is currently registered.

vcId **IxAtmSchedulerVcId** [in] − VC identifier value of the VC to be deregistered. This value was supplied to the client when the VC was originally registered. This value can also be queried from the IxAtmm component through the **ixAtmmVcQuery** interface.

Returns:

◊ IX_SUCCESS : The specified VC has been successfully removed from this port.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.

◊ IX_FAIL : There is no registered VC associated with the supplied identifier registered on this port.

```
ixAtmmVcIdQuery ( IxAtmLogicalPort      port,
                  IxAtmSchedulerVcId    vcId,
                  IxAtmmVc *            vcDesc
                )
```

This interface supplies information about an active VC on a particular port when supplied with a vcId for that VC.

Parameters:

port

IxAtmLogicalPort [in] − Identifies port on which the VC to be queried is currently registered.

vcId **IxAtmSchedulerVcId** [in] − Value returned by **ixAtmmVcRegister** which uniquely identifies the requested VC on this port.

vcDesc* **IxAtmmVc [out] − Pointer to an **IxAtmmVc** structure which will be filled with the specific details of the requested VC, if it exists on this port.

Returns:

◊ IX_SUCCESS : The specified VC has been found on this port and the requested details have been returned.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.

◊ IX_ATMM_RET_NO_SUCH_VC : No VC exists on the specified port which matches the supplied identifier. No data is returned.

◊ IX_ATMM_RET_INVALID_PARAM_PTR : A pointer parameter was NULL.

```
ixAtmmVcQuery ( IxAtmLogicalPort       port,
                unsigned               vpi,
                unsigned               vci,
                IxAtmmVcDirection      direction,
                IxAtmSchedulerVcId *   vcId,
                IxAtmmVc *             vcDesc
              )
```

This interface supplies information about an active VC on a particular port when supplied with the VPI, VCI and direction of that VC.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies port on which the VC to be queried is currently registered.

vpi unsigned [in] − ATM VPI value of the requested VC.

vci unsigned [in] − ATM VCI value of the requested VC.

direction **IxAtmmVcDirection** [in] − One of **IX_ATMM_VC_DIRECTION_TX** or **IX_ATMM_VC_DIRECTION_RX** indicating the direction (Tx or Rx) of the requested VC.

vcId* **IxAtmSchedulerVcId [out] − Pointer to an integer value which will be filled with the VC identifier value for the requested VC (as returned by **ixAtmmVcRegister**), if it exists on this port.

vcDesc* **IxAtmmVc [out] − Pointer to an **IxAtmmVc** structure which will be filled with the specific details of the requested VC, if it exists on this port.

Returns:

◊ IX_SUCCESS : The specified VC has been found on this port and the requested details have been returned.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.

◊ IX_ATMM_RET_NO_SUCH_VC : No VC exists on the specified port which matches the search criteria (VPI, VCI, direction) given. No data is returned.

◊ IX_ATMM_RET_INVALID_PARAM_PTR : A pointer parameter was NULL.

```
ixAtmmVcRegister ( IxAtmLogicalPort      port,
                   IxAtmmVc *            vcToAdd,
                   IxAtmSchedulerVcId *  vcId
                 )
```

This interface is used to register an ATM Virtual Connection on the specified ATM port.

Each call to this interface registers a unidirectional virtual connection with the parameters specified. If a bi−directional VC is needed, the function should be called twice (once for each direction, Tx & Rx) where the VPI and VCI and port parameters in each call are identical.

With the addition of each new VC to a port, a series of callback functions are invoked by the IxAtmm component to notify possible external components of the change. The callback functions are registered using the **ixAtmmVcChangeCallbackRegister** interface.

The IxAtmSch component is notified of the registration of transmit VCs.

Parameters:

port **IxAtmLogicalPort** [in] − Identifies port on which the specified VC is to be registered.

vcToAdd* **IxAtmmVc [in] − Pointer to an **IxAtmmVc** structure containing a description of the VC to be registered. The client shall fill the vpi, vci and direction and relevant trafficDesc members of this structure before calling this function.

vcId* **IxAtmSchedulerVcId [out] − Pointer to an integer value which is filled with the per−port unique identifier value for this VC. This identifier will be required when a request is made to deregister or change this VC. VC identifiers for transmit VCs will have a value between 0−43, i.e. 32 data Tx VCs + 12 OAM Tx Port VCs. Receive VCs will have a value between 44−66, i.e. 32 data Rx VCs + 1 OAM Rx VC.

Returns:

◊ IX_SUCCESS : The VC has been successfully registered on this port. The VC is ready for a client to configure IxAtmdAcc for receive and transmit operations on the VC.

◊ IX_ATMM_RET_INVALID_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.

◊ IX_ATMM_RET_INVALID_VC_DESCRIPTOR : The descriptor pointed to by vcToAdd is invalid. The registration request is rejected.

◊ IX_ATMM_RET_VC_CONFLICT : The VC requested conflicts with reserved VPI and/or VCI values or with another VC already activated on this port.

- ◊ IX_ATMM_RET_PORT_CAPACITY_IS_FULL : The VC cannot be registered in the port becuase the port capacity is insufficient to support the requested ATM traffic contract. The registration request is rejected.

- ◊ IX_ATMM_RET_INVALID_PARAM_PTR : A pointer parameter was NULL.

Warning:

IxAtmm has no capability of signaling or negotiating a virtual connection. Negotiation of the admission of the VC to the network is beyond the scope of this function. This is assumed to be performed by the calling client, if appropriate, before or after this function is called.

Intel (R) IXP400 Software ATM Transmit Scheduler (IxAtmSch) API

IXP400 ATM scheduler component Public API.

Defines

#define **IX_ATMSCH_RET_NOT_ADMITTED**

Indicates that CAC function has rejected VC registration due to insufficient line capacity.

#define **IX_ATMSCH_RET_QUEUE_FULL**

Indicates that the VC queue is full, no more demand can be queued at this time.

#define **IX_ATMSCH_RET_QUEUE_EMPTY**

Indicates that all VC queues on this port are empty and therefore there are no cells to be scheduled at this time.

Functions

PUBLIC
IX_STATUS **ixAtmSchInit** (void)

This function is used to initialize the ixAtmSch component. It should be called before any other IxAtmSch API function.

PUBLIC
IX_STATUS **ixAtmSchUninit** (void)

This function is used to uninitialize the ixAtmSch component.

PUBLIC**ixAtmSchPortModelInitialize** (**IxAtmLogicalPort** port, UINT64 portRate, unsigned int
IX_STATUS minCellsToSchedule)

This function shall be called first to initialize an ATM port before any other ixAtmSch API calls may be made for that port.

PUBLIC
IX_STATUS **ixAtmSchPortModelUninitialize** (**IxAtmLogicalPort** port)

This function shall be called to uninitialize an ATM port.

PUBLIC
IX_STATUS **ixAtmSchPortRateModify** (**IxAtmLogicalPort** port, UINT64 portRate)

This function is called to modify the portRate on a previously initialized port, typically in the event that the line condition of the port changes.

PUBLIC**ixAtmSchVcModelSetup** (**IxAtmLogicalPort** port, **IxAtmTrafficDescriptor** *trafficDesc,
IX_STATUS **IxAtmSchedulerVcId** *vcId)

*A client calls this interface to set up an upstream (transmitting) virtual connection model (VC) on the specified ATM port. This function also provides the virtual * connection*

admission control (CAC) service to the client.

PUBLIC IX_STATUS **ixAtmSchVcConnIdSet** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId, **IxAtmConnId** vcUserConnId)
A client calls this interface to set the vcUserConnId for a VC on the specified ATM port. This vcUserConnId will default to IX_ATM_IDLE_CELLS_CONNID if this function is not called for a VC. Hence if the client does not call this function for a VC then only idle cells will be scheduled for this VC.

PUBLIC IX_STATUS **ixAtmSchVcModelRemove** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId)
Interface called by the client to remove a previously established VC on a particular port.

PUBLIC IX_STATUS **ixAtmSchVcQueueUpdate** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId, UINT64 numberOfCells)
The client calls this function to notify IxAtmSch that the user of a VC has submitted cells for transmission.

PUBLIC IX_STATUS **ixAtmSchVcQueueClear** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId)
The client calls this function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero.

PUBLIC IX_STATUS **ixAtmSchTableUpdate** (**IxAtmLogicalPort** port, UINT64 maxCells, **IxAtmScheduleTable** **rettable)
The client calls this function to request an update of the schedule table for a particular ATM port.

PUBLIC void **ixAtmSchShow** (void)
Utility function which will print statistics on the current and accumulated state of VCs and traffic in the ATM scheduler component. Output is sent to the default output device.

PUBLIC void **ixAtmSchStatsClear** (void)
Utility function which will reset all counter statistics in the ATM scheduler to zero.

Detailed Description

IXP400 ATM scheduler component Public API.

Define Documentation

```
#define IX_ATMSCH_RET_NOT_ADMITTED
```

Indicates that CAC function has rejected VC registration due to insufficient line capacity.

Definition at line **52** of file **IxAtmSch.h**.

#define IX_ATMSCH_RET_QUEUE_EMPTY

Indicates that all VC queues on this port are empty and therefore there are no cells to be scheduled at this time.

Definition at line **70** of file **IxAtmSch.h**.

#define IX_ATMSCH_RET_QUEUE_FULL

Indicates that the VC queue is full, no more demand can be queued at this time.

Definition at line **61** of file **IxAtmSch.h**.

Function Documentation

ixAtmSchInit (void)

This function is used to initialize the ixAtmSch component. It should be called before any other IxAtmSch API function.

Parameters:

None

Returns:

- ◊ **IX_SUCCESS :** indicates that
  1. The ATM scheduler component has been successfully initialized.
  2. The scheduler is ready to accept Port modelling requests.
- ◊ **IX_FAIL :** Some internal error has prevented the scheduler component from initialising.

```
ixAtmSchPortModelInitialize ( IxAtmLogicalPort port,
                              UINT64           portRate,
                              unsigned int     minCellsToSchedule
                            )
```

This function shall be called first to initialize an ATM port before any other ixAtmSch API calls may be made for that port.

Parameters:

port **IxAtmLogicalPort** [in] − The specific port to initialize. Valid values range from 0 to IX_UTOPIA_MAX_PORTS − 1, representing a maximum of IX_UTOPIA_MAX_PORTS possible ports.

portRate UINT64 [in] − Value indicating the upstream capacity of the indicated port. The value should be supplied in units of ATM (53 bytes) cells per second. A port rate of 800Kbits/s is the equivalent of 1886 cells per second

minCellsToSchedule unsigned int [in] − This parameter specifies the minimum number of cells

which the scheduler will put in a schedule table for this port. This value sets the worst case CDVT for VCs on this port i.e. CDVT = 1*minCellsToSchedule/portRate.

Returns:

- ◊ **IX_SUCCESS :** indicates that
  1. The ATM scheduler has been successfully initialized.
  2. The requested port model has been established.
  3. The scheduler is ready to accept VC modelling requests on the ATM port.
- ◊ **IX_FAIL :** indicates the requested port could not be initialized.

ixAtmSchPortModelUninitialize (**IxAtmLogicalPort** *port*)

This function shall be called to uninitialize an ATM port.

Parameters:

port **IxAtmLogicalPort** [in] − The specific port to uninitialize. Valid values range from 0 to IX_UTOPIA_MAX_PORTS − 1, representing a maximum of IX_UTOPIA_MAX_PORTS possible ports.

Returns:

- ◊ **IX_SUCCESS :** indicates that
  1. The ATM scheduler has been successfully uninitialized.
- ◊ **IX_FAIL :** indicates the requested port could not be uninitialized.

```
ixAtmSchPortRateModify ( IxAtmLogicalPort port,
                         UINT64          portRate
                       )
```

This function is called to modify the portRate on a previously initialized port, typically in the event that the line condition of the port changes.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port which is to be modified.

portRate UINT64 [in] − Value indicating the new upstream capacity for this port in cells/second. A port rate of 800Kbits/s is the equivalent of 1886 cells per second

Returns:

- ◊ **IX_SUCCESS :** The port rate has been successfully modified.
- ◊ **IX_FAIL :** The port rate could not be modified, either because the input data was invalid, or the new port rate is insufficient to support established ATM VC contracts on this port.

Warning:

The IxAtmSch component will validate the supplied port rate is sufficient to support all established VC contracts on the port. If the new port rate is insufficient to support all established contracts then the request to modify the port rate will be rejected. In this event, the user is expected to remove established contracts using the ixAtmSchVcModelRemove interface and then retry this interface.

See also:

ixAtmSchVcModelRemove()

ixAtmSchShow (void)

Utility function which will print statistics on the current and accumulated state of VCs and traffic in the ATM scheduler component. Output is sent to the default output device.

Parameters:

none

Returns:

none

ixAtmSchStatsClear (void)

Utility function which will reset all counter statistics in the ATM scheduler to zero.

Parameters:

none

Returns:

none

```
ixAtmSchTableUpdate ( IxAtmLogicalPort       port,
                      UINT64                 maxCells,
                      IxAtmScheduleTable **  rettable
                    )
```

The client calls this function to request an update of the schedule table for a particular ATM port.

This is called when the client decides it needs a new sequence of cells to send (probably because the transmit queue is near to empty for this ATM port). The scheduler will use its stored information on the cells submitted for transmit (i.e. data supplied via **ixAtmSchVcQueueUpdate** function) with the traffic descriptor information of all established VCs on the ATM port to decide the sequence of cells to be sent and fill the schedule table for a period of time into the future.

IxAtmSch will guarantee a minimum of minCellsToSchedule if there is at least one cell ready to send. If there are no cells then IX_ATMSCH_RET_QUEUE_EMPTY is returned.

This implementation of ixAtmSchTableUpdate uses no operating system or external facilities, either directly or indirectly. This allows clients to call this function form within an FIQ interrupt handler.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port for which requested schedule table is to be generated.

maxCells

UINT64 [in] − Specifies the maximum number of cells that must be scheduled in the supplied table during any call to the interface.

***table* **IxAtmScheduleTable** [out] − A pointer to an area of storage is returned which contains the generated schedule table. The client should not modify the contents of this table.

Returns:

- ◊ **IX_SUCCESS :** The schedule table has been published. Currently there is at least one VC queue that is nonempty.
- ◊ **IX_ATMSCH_RET_QUEUE_EMPTY :** Currently all VC queues on this port are empty. The schedule table returned is set to NULL. The client is not expected to invoke this function again until more cells have been submitted on this port through the **ixAtmSchVcQueueUpdate** function.
- ◊ **IX_FAIL :** The input are invalid. No action is taken.

Warning:

IxAtmSch assumes that the calling software ensures that calls to ixAtmSchVcQueueUpdate, ixAtmSchVcQueueClear and ixAtmSchTableUpdate are both self and mutually exclusive for the same port.

Subsequent calls to this function for the same port will overwrite the contents of previously supplied schedule tables. The client must be completely finished with the previously supplied schedule table before calling this function again for the same port.

See also:

ixAtmSchVcQueueUpdate(), **ixAtmSchVcQueueClear()**, **ixAtmSchTableUpdate()**.

ixAtmSchUninit (void)

This function is used to uninitialize the ixAtmSch component.

Parameters:

None

Returns:

- ◊ **IX_SUCCESS :** indicates that
  1. The ATM scheduler component has been successfully uninitialized.
- ◊ **IX_FAIL :** Some internal error has prevented the scheduler component from uninitialising.

```
ixAtmSchVcConnIdSet ( IxAtmLogicalPort      port,
                      IxAtmSchedulerVcId    vcId,
                      IxAtmConnId           vcUserConnId
                    )
```

A client calls this interface to set the vcUserConnId for a VC on the specified ATM port. This vcUserConnId will default to IX_ATM_IDLE_CELLS_CONNID if this function is not called for a VC. Hence if the client does not call this function for a VC then only idle cells will be scheduled for this VC.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the upstream VC is has been established.

vcId **IxAtmSchedulerVcId** [in] − This is the unique identifier for this virtual connection. A valid identification is a non−negative number and is all ports.

vcUserConnId **IxAtmConnId** [in] − The connId is used to refer to a VC in schedule table entries. It is treated as the Id by which the scheduler client knows the VC. It is used in any communicatations from the Scheduler to the scheduler user e.g. schedule table entries.

Returns:

◊ **IX_SUCCESS :** The id has successfully been set.
◊ **IX_FAIL :**Input data are invalid. connId id is not established.

```
ixAtmSchVcModelRemove ( IxAtmLogicalPort      port,
                        IxAtmSchedulerVcId    vcId
                      )
```

Interface called by the client to remove a previously established VC on a particular port.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the VC to be removed is established.

vcId **IxAtmSchedulerVcId** [in] − Identifies the VC to be removed. This is the value returned by the **ixAtmSchVcModelSetup** call which established the relevant VC.

Returns:

◊ **IX_SUCCESS :** The VC has been successfully removed from this port. It is no longer modelled on this port.
◊ **IX_FAIL :**Input data are invalid. The VC is still being modeled by the traffic shaper.

See also:

ixAtmSchVcModelSetup()

```
ixAtmSchVcModelSetup ( IxAtmLogicalPort           port,
                       IxAtmTrafficDescriptor *   trafficDesc,
                       IxAtmSchedulerVcId *       vcId
                     )
```

A client calls this interface to set up an upstream (transmitting) virtual connection model (VC) on the specified ATM port. This function also provides the virtual * connection admission control (CAC) service to the client.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the upstream VC is to be established.

trafficDesc* **IxAtmTrafficDescriptor [in] − Pointer to a structure describing the requested traffic contract of the VC to be established. This structure contains the typical ATM traffic

descriptor values (e.g. PCR, SCR, MBS, CDVT, etc.) defined by the ATM standard.

vcId* **IxAtmSchedulerVcId [out] − This value will be filled with the port−unique identifier for this virtual connection. A valid identification is a non−negative number.

Returns:

- ◊ **IX_SUCCESS :** The VC has been successfully established on this port. The client may begin to submit demand on this VC.
- ◊ **IX_ATMSCH_RET_NOT_ADMITTED :** The VC cannot be established on this port because there is insufficient upstream capacity available to support the requested traffic contract descriptor
- ◊ **IX_FAIL :**Input data are invalid. VC has not been established.

```
ixAtmSchVcQueueClear ( IxAtmLogicalPort      port,
                       IxAtmSchedulerVcId    vcId
                     )
```

The client calls this function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero.

This interface is structurally compatible with the IxAtmdAccSchQueueClear callback type definition required for IXP400 ATM scheduler interoperability.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the VC to be cleared is established.

vcId **IxAtmSchedulerVcId** [in] − Identifies the VC to be cleared. This is the value returned by the **ixAtmSchVcModelSetup** call which established the relevant VC.

Returns:

- ◊ **IX_SUCCESS :** The VC queue has been successfully cleared.
- ◊ **IX_FAIL :** The input are invalid. No VC queue is modified.

Warning:

IxAtmSch assumes that the calling software ensures that calls to ixAtmSchVcQueueUpdate, ixAtmSchVcQueueClear and ixAtmSchTableUpdate are both self and mutually exclusive for the same port.

See also:

ixAtmSchVcQueueUpdate(), **ixAtmSchVcQueueClear()**, **ixAtmSchTableUpdate()**.

```
ixAtmSchVcQueueUpdate ( IxAtmLogicalPort      port,
                        IxAtmSchedulerVcId    vcId,
                        UINT64                numberOfCells
                      )
```

The client calls this function to notify IxAtmSch that the user of a VC has submitted cells for transmission.

This information is stored, aggregated from a number of calls to ixAtmSchVcQueueUpdate and eventually used in the call to ixAtmSchTableUpdate.

Normally IxAtmSch will update the VC queue by adding the number of cells to the current queue length. However, if IxAtmSch determines that the user has over−submitted for the VC and exceeded its transmission quota the queue request can be rejected. The user should resubmit the request later when the queue has been depleted.

This implementation of ixAtmSchVcQueueUpdate uses no operating system or external facilities, either directly or indirectly. This allows clients to call this function form within an interrupt handler.

This interface is structurally compatible with the IxAtmdAccSchQueueUpdate callback type definition required for IXP400 ATM scheduler interoperability.

Parameters:

port **IxAtmLogicalPort** [in] − Specifies the ATM port on which the VC to be updated is established.

vcId **IxAtmSchedulerVcId** [in] − Identifies the VC to be updated. This is the value returned by the **ixAtmSchVcModelSetup** call which established the relevant VC.

numberOfCells UINT64 [in] − Indicates how many ATM cells should be added to the queue for this VC.

Returns:

- ◊ **IX_SUCCESS :** The VC queue has been successfully updated.
- ◊ **IX_ATMSCH_RET_QUEUE_FULL :** The VC queue has reached a preset limit. This indicates the client has over−submitted and exceeded its transmission quota. The request is rejected. The VC queue is not updated. The VC user is advised to resubmit the request later.
- ◊ **IX_FAIL :** The input are invalid. No VC queue is updated.

Warning:

IxAtmSch assumes that the calling software ensures that calls to ixAtmSchVcQueueUpdate, ixAtmSchVcQueueClear and ixAtmSchTableUpdate are both self and mutually exclusive for the same port.

See also:

ixAtmSchVcQueueUpdate(), **ixAtmSchVcQueueClear()**, **ixAtmSchTableUpdate()**.

Intel (R) IXP400 Software ATM Types (IxAtmTypes)

The common set of types used in many Atm components.

Data Structures

struct **IxAtmScheduleTable**
This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.

struct **IxAtmScheduleTableEntry**
ATM Schedule Table entry.

struct **IxAtmTrafficDescriptor**
Structure describing an ATM traffic contract for a Virtual Connection (VC).

Defines

#define **IX_ATM_CELL_PAYLOAD_SIZE**
Size of a ATM cell payload.

#define **IX_ATM_CELL_SIZE**
Size of a ATM cell, including header.

#define **IX_ATM_CELL_SIZE_NO_HEC**
Size of a ATM cell, excluding HEC byte.

#define **IX_ATM_OAM_CELL_SIZE_NO_HEC**
Size of a OAM cell, excluding HEC byte.

#define **IX_ATM_AAL0_48_CELL_PAYLOAD_SIZE**
Size of a AAL0 48 Cell payload.

#define **IX_ATM_AAL5_CELL_PAYLOAD_SIZE**
Size of a AAL5 Cell payload.

#define **IX_ATM_AAL0_52_CELL_SIZE_NO_HEC**
Size of a AAL0 52 Cell, excluding HEC byte.

#define **IX_ATM_MAX_VPI**
Maximum value of an ATM VPI.

#define **IX_ATM_MAX_VCI**
Maximum value of an ATM VCI.

#define **IX_ATM_MAX_NUM_AAL_VCS**

Maximum number of active AAL5/AAL0 VCs in the system.

#define **IX_ATM_MAX_NUM_VC**
Maximum number of active AAL5/AAL0 VCs in the system The use of this macro is depreciated, it is retained for backward compatiblity. For current software release and beyond the define IX_ATM_MAX_NUM_AAL_VC should be used.

#define **IX_ATM_MAX_NUM_OAM_TX_VCS**
Maximum number of active OAM Tx VCs in the system, 1 OAM VC per port.

#define **IX_ATM_MAX_NUM_OAM_RX_VCS**
Maximum number of active OAM Rx VCs in the system, 1 OAM VC shared accross all ports.

#define **IX_ATM_MAX_NUM_AAL_OAM_TX_VCS**
Maximum number of active AAL5/AAL0/OAM Tx VCs in the system.

#define **IX_ATM_MAX_NUM_AAL_OAM_RX_VCS**
Maximum number of active AAL5/AAL0/OAM Rx VCs in the system.

#define **IX_ATM_IDLE_CELLS_CONNID**
VC Id used to indicate idle cells in the returned schedule table.

#define **IX_ATM_CELL_HEADER_VCI_GET**(cellHeader)
get the VCI field from a cell header

#define **IX_ATM_CELL_HEADER_VPI_GET**(cellHeader)
get the VPI field from a cell header

#define **IX_ATM_CELL_HEADER_PTI_GET**(cellHeader)
get the PTI field from a cell header

Typedefs

typedef unsigned int **IxAtmCellHeader**
ATM Cell Header, does not contain 4 byte HEC, added by NPE−A.

typedef unsigned int **IxAtmConnId**
ATM VC data connection identifier.

typedef int **IxAtmSchedulerVcId**
ATM VC scheduling connection identifier.

typedef unsigned int **IxAtmNpeRxVcId**

ATM Rx VC identifier used by the ATM Npe.

Enumerations

enum **IxAtmLogicalPort** {
IX_UTOPIA_PORT_0,
IX_UTOPIA_MAX_PORTS
}
Logical Port Definitions :.

enum **IxAtmServiceCategory** {
IX_ATM_CBR,
IX_ATM_RTVBR,
IX_ATM_VBR,
IX_ATM_UBR,
IX_ATM_ABR
}
Enumerated type representing available ATM service categories. For more informatoin on these categories, see "Traffic Management Specification" v4.1, published by the ATM Forum − http://www.atmforum.com.

enum **IxAtmRxQueueId** {
IX_ATM_RX_A,
IX_ATM_RX_B,
IX_ATM_MAX_RX_STREAMS
}
Rx Queue Type for RX traffic.

Detailed Description

The common set of types used in many Atm components.

Define Documentation

```
#define IX_ATM_AAL0_48_CELL_PAYLOAD_SIZE
```

Size of a AAL0 48 Cell payload.

Definition at line **92** of file **IxAtmTypes.h**.

```
#define IX_ATM_AAL0_52_CELL_SIZE_NO_HEC
```

Size of a AAL0 52 Cell, excluding HEC byte.

Definition at line **104** of file **IxAtmTypes.h**.

#define IX_ATM_AAL5_CELL_PAYLOAD_SIZE

Size of a AAL5 Cell payload.

Definition at line **98** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_HEADER_PTI_GET (cellHeader)

get the PTI field from a cell header

Definition at line **187** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_HEADER_VCI_GET (cellHeader)

get the VCI field from a cell header

Definition at line **173** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_HEADER_VPI_GET (cellHeader)

get the VPI field from a cell header

Definition at line **180** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_PAYLOAD_SIZE

Size of a ATM cell payload.

Definition at line **68** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_SIZE

Size of a ATM cell, including header.

Definition at line **74** of file **IxAtmTypes.h**.

#define IX_ATM_CELL_SIZE_NO_HEC

Size of a ATM cell, excluding HEC byte.

Definition at line **80** of file **IxAtmTypes.h**.

#define IX_ATM_IDLE_CELLS_CONNID

VC Id used to indicate idle cells in the returned schedule table.

Definition at line **166** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_AAL_OAM_RX_VCS

Maximum number of active AAL5/AAL0/OAM Rx VCs in the system.

Definition at line **160** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_AAL_OAM_TX_VCS

Maximum number of active AAL5/AAL0/OAM Tx VCs in the system.

Definition at line **154** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_AAL_VCS

Maximum number of active AAL5/AAL0 VCs in the system.

Definition at line **123** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_OAM_RX_VCS

Maximum number of active OAM Rx VCs in the system, 1 OAM VC shared accross all ports.

Definition at line **148** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_OAM_TX_VCS

Maximum number of active OAM Tx VCs in the system, 1 OAM VC per port.

Definition at line **141** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_NUM_VC

Maximum number of active AAL5/AAL0 VCs in the system The use of this macro is depreciated, it is

retained for backward compatiblity. For current software release and beyond the define IX_ATM_MAX_NUM_AAL_VC should be used.

Definition at line **132** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_VCI

Maximum value of an ATM VCI.

Definition at line **117** of file **IxAtmTypes.h**.

#define IX_ATM_MAX_VPI

Maximum value of an ATM VPI.

Definition at line **111** of file **IxAtmTypes.h**.

#define IX_ATM_OAM_CELL_SIZE_NO_HEC

Size of a OAM cell, excluding HEC byte.

Definition at line **86** of file **IxAtmTypes.h**.

Typedef Documentation

IxAtmCellHeader

ATM Cell Header, does not contain 4 byte HEC, added by NPE−A.

Definition at line **195** of file **IxAtmTypes.h**.

IxAtmConnId

ATM VC data connection identifier.

This is is generated by IxAtmdAcc when a successful connection is made on a VC. The is the ID by which IxAtmdAcc knows an active VC and should be used in IxAtmdAcc API calls to reference a specific VC.

Definition at line **283** of file **IxAtmTypes.h**.

IxAtmNpeRxVcId

ATM Rx VC identifier used by the ATM Npe.

This Id is generated by IxAtmdAcc when a successful data connection is made on a Rx VC.

Definition at line **307** of file **IxAtmTypes.h**.

IxAtmSchedulerVcId

ATM VC scheduling connection identifier.

This id is generated and used by ATM Tx controller, generally the traffic shaper (e.g. IxAtmSch). The IxAtmdAcc component will request one of these Ids whenever a data connection on a Tx VC is requested. This ID will be used in callbacks to the ATM Transmission Ctrl s/w (e.g. IxAtmm) to reference a particular VC.

Definition at line **297** of file **IxAtmTypes.h**.

Enumeration Type Documentation

enum IxAtmLogicalPort

Logical Port Definitions :.

Only 1 port is available in SPHY configuration 4 ports are enabled in ATMMPORT configuration 12 ports are enabled in MPHY configuration

Enumeration values:

IX_UTOPIA_PORT_0	Port 0.
IX_UTOPIA_MAX_PORTS	Not a port − just a definition for the maximum possible ports.

Definition at line **37** of file **IxAtmTypes.h**.

enum IxAtmRxQueueId

Rx Queue Type for RX traffic.

IxAtmRxQueueId defines the queues involved for receiving data.

There are two queues to facilitate prioritisation handling and processing the 2 queues with different algorithms and constraints

e.g. : one queue can carry voice (or time−critical traffic), the other queue can carry non−voice traffic

Enumeration values:

IX_ATM_RX_A	Rx queue A.
IX_ATM_RX_B	Rx queue B.

IX_ATM_MAX_RX_STREAMS Maximum number of Rx streams.

Definition at line **232** of file **IxAtmTypes.h**.

enum IxAtmServiceCategory

Enumerated type representing available ATM service categories. For more informatoin on these categories, see "Traffic Management Specification" v4.1, published by the ATM Forum − http://www.atmforum.com.

Enumeration values:

IX_ATM_CBR	Constant Bit Rate.
IX_ATM_RTVBR	Real Time Variable Bit Rate.
IX_ATM_VBR	Variable Bit Rate.
IX_ATM_UBR	Unspecified Bit Rate.
IX_ATM_ABR	Available Bit Rate (not supported).

Definition at line **206** of file **IxAtmTypes.h**.

Intel (R) IXP400 Software Security (IxCryptoAcc) API

IXP400 Security component Public API.

Data Structures

struct **IxCryptoAccAuthCtx**
Structure storing authentication configuration parameters required to perform security functionality.

struct **IxCryptoAccCipherCtx**
Structure storing cipher configuration parameters required to perform security functionality.

struct **IxCryptoAccCtx**
Structure storing configuration parameters required to perform security functionality.

struct **IxCryptoAccPkeEauBnAddSubMulOperands**
*Structure storing input operands for large number addition (Carry | R = A + B) or subtraction operation (Borrow | R = A − B) or multiplication (R = A * B).*

struct **IxCryptoAccPkeEauBnModOperands**
Structure storing input operands for large number modular reduction operation (R = A mod N).

union **IxCryptoAccPkeEauInOperands**
*Union storing input operands required for all EAU operations. These input operands will be supplied to **ixCryptoAccPkeEauPerform** interface to perform EAU functionalities.*

struct **IxCryptoAccPkeEauModExpOperands**
Structure storing input operands for large number modular exponential operation (C = M^e mod N).

struct **IxCryptoAccPkeEauOperand**
*Structure storing operand / result data pointer and length for EAU functionality performing through **ixCryptoAccPkeEauPerform** interface.*

Defines

#define **IX_CRYPTO_ACC_MAX_CIPHER_KEY_LENGTH**
Max length (byte) of cipher key for DES (64 bit), 3DES (192 bit), AES (128, 192 & 256 bit).

#define **IX_CRYPTO_ACC_MAX_CIPHER_IV_LENGTH**
Max IV length in bytes.

#define **IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH**
Max length (byte) of authentication key for SHA1 and MD5.

#define **IX_CRYPTO_ACC_MAX_AUTH_IV_LENGTH**
Max length (byte) of initial chaining variable for SHA1 (160 bit) and MD5 (128 bit).

#define **IX_CRYPTO_ACC_MAX_QUEUE_DEPTH**
Max queue depth supported by the Queue Manager.

#define **IX_CRYPTO_ACC_MAX_ACTIVE_SA_TUNNELS**
Maximum active tunnels supported could be changed by the client based on the application's requirements.

#define **IX_CRYPTO_ACC_DES_KEY_64**
DES key length in bytes.

#define **IX_CRYPTO_ACC_DES_BLOCK_64**
DES cipher block length in bytes.

#define **IX_CRYPTO_ACC_DES_IV_64**
DES initialization vector length in bytes.

#define **IX_CRYPTO_ACC_3DES_KEY_192**
3DES key length in bytes

#define **IX_CRYPTO_ACC_AES_KEY_128**
AES−128 key length in bytes.

#define **IX_CRYPTO_ACC_AES_KEY_192**
AES−192 key length in bytes.

#define **IX_CRYPTO_ACC_AES_KEY_256**
AES−256 key length in bytes.

#define **IX_CRYPTO_ACC_AES_BLOCK_128**
AES cipher block length in bytes.

#define **IX_CRYPTO_ACC_AES_CBC_IV_128**
AES initialization vector length in bytes for CBC mode.

#define **IX_CRYPTO_ACC_AES_CTR_IV_128**
AES initialization vector length in bytes for CTR mode.

#define **IX_CRYPTO_ACC_CCM_AAD_LEN_384**
48 Bytes of additional authenticated data for CBC−MAC computation

#define **IX_CRYPTO_ACC_AES_CCM_IV_512**
CCM Initialization vector has 2 components to it.

#define **IX_CRYPTO_ACC_ARC4_KEY_128**
Key length for ARC4 algorithms.

#define **IX_CRYPTO_ACC_ARC4_BLOCK_8**
ARC4 algorithm block size.

#define **IX_CRYPTO_ACC_SHA1_KEY_160**
SHA1 key length in bytes.

#define **IX_CRYPTO_ACC_SHA1_DIGEST_160**
SHA1 message digest length in bytes.

#define **IX_CRYPTO_ACC_MD5_KEY_128**
MD5 key length in bytes.

#define **IX_CRYPTO_ACC_MD5_DIGEST_128**
MD5 message digest length in bytes.

#define **IX_CRYPTO_ACC_CCM_DIGEST_64**
CCM digest length in bytes.

#define **IX_CRYPTO_ACC_WEP_CRC_DIGEST_32**
Digest length of WEP ICV in bytes.

#define **ixCryptoAccHashPerform**
*alias function name to **ixCryptoAccHashKeyGenerate***

Typedefs

typedef **IxCryptoAccPkeEauOperand** **IxCryptoAccPkeEauOpResult**
Structure storing result of EAU operation.

typedef void(* **IxCryptoAccRegisterCompleteCallback**)(UINT32 cryptoCtxId, IX_OSAL_MBUF *pMbuf, **IxCryptoAccStatus** status)
Cryptographic Context registration complete callback notification.

typedef void(* **IxCryptoAccHashKeyGenCompleteCallback**)(UINT32 hashKeyId, IX_OSAL_MBUF *pMbufHashKey, **IxCryptoAccStatus** status)
Hash key generation complete callback notification.

typedef void(* **IxCryptoAccPerformCompleteCallback**)(UINT32 cryptoCtxId, IX_OSAL_MBUF *pSrcMbuf, IX_OSAL_MBUF *pDestMbuf, **IxCryptoAccStatus** status)
Hardware accelerator service request complete callback notification.

typedef void(* **IxCryptoAccPkeHashPerformCompleteCallback**)(UINT8 *pDigest, **IxCryptoAccStatus** status)
PKE Crypto Engine SHA engine hashing perform complete callback notification.

typedef void(* **IxCryptoAccPkeEauPerformCompleteCallback**)(**IxCryptoAccPkeEauOperation** operation, **IxCryptoAccPkeEauOpResult** *pResult, BOOL carryOrBorrow, **IxCryptoAccStatus** status)
EAU perform complete callback notification.

Enumerations

enum **IxCryptoAccCfg** {
IX_CRYPTO_ACC_CFG_WEP_XSCALE_ACC_EN,
IX_CRYPTO_ACC_CFG_CRYPTO_NPE_ACC_EN,
IX_CRYPTO_ACC_CFG_WEP_NPE_ACC_EN,
IX_CRYPTO_ACC_CFG_CRYPTO_WEP_NPE_ACC_EN,
IX_CRYPTO_ACC_CFG_TYPE
}
Possible configuration definitions.

enum **IxCryptoAccOperation** {
IX_CRYPTO_ACC_OP_ENCRYPT,
IX_CRYPTO_ACC_OP_DECRYPT,
IX_CRYPTO_ACC_OP_AUTH_CALC,
IX_CRYPTO_ACC_OP_AUTH_CHECK,
IX_CRYPTO_ACC_OP_ENCRYPT_AUTH,
IX_CRYPTO_ACC_OP_AUTH_DECRYPT,
IX_CRYPTO_ACC_OP_TYPE_OF_OPERATION
}
Cryptographic Operation Definitions.

enum **IxCryptoAccCipherAlgo** {
IX_CRYPTO_ACC_CIPHER_NULL,
IX_CRYPTO_ACC_CIPHER_DES,
IX_CRYPTO_ACC_CIPHER_3DES,
IX_CRYPTO_ACC_CIPHER_AES,
IX_CRYPTO_ACC_CIPHER_ARC4,
IX_CRYPTO_ACC_CIPHER_ALGO_TYPE
}
Cipher Algorithm Definitions.

enum **IxCryptoAccCipherMode** {
IX_CRYPTO_ACC_MODE_NULL,
IX_CRYPTO_ACC_MODE_ECB,
IX_CRYPTO_ACC_MODE_CBC,
IX_CRYPTO_ACC_MODE_CTR,
IX_CRYPTO_ACC_MODE_CCM,

IX_CRYPTO_ACC_MODE_TYPE
}
Cipher Mode Definitions.

enum **IxCryptoAccAuthAlgo** {
IX_CRYPTO_ACC_AUTH_NULL,
IX_CRYPTO_ACC_AUTH_SHA1,
IX_CRYPTO_ACC_AUTH_MD5,
IX_CRYPTO_ACC_AUTH_CBC_MAC,
IX_CRYPTO_ACC_AUTH_WEP_CRC,
IX_CRYPTO_ACC_AUTH_TYPE
}
Authentication Algorithm Definitions.

enum **IxCryptoAccPkeEauOperation** {
IX_CRYPTO_ACC_OP_EAU_MOD_EXP,
IX_CRYPTO_ACC_OP_EAU_BN_MUL,
IX_CRYPTO_ACC_OP_EAU_BN_ADD,
IX_CRYPTO_ACC_OP_EAU_BN_MOD,
IX_CRYPTO_ACC_OP_EAU_BN_SUB,
IX_CRYPTO_ACC_OP_TYPE_OF_EAU_OPERATION
}
EAU (Exponentiation Acceleration Unit) Operation Definitions.

enum **IxCryptoAccStatus** {
IX_CRYPTO_ACC_STATUS_SUCCESS,
IX_CRYPTO_ACC_STATUS_FAIL,
IX_CRYPTO_ACC_STATUS_WAIT,
IX_CRYPTO_ACC_STATUS_RETRY,
IX_CRYPTO_ACC_STATUS_QUEUE_FULL,
IX_CRYPTO_ACC_STATUS_OPERATION_NOT_SUPPORTED,
IX_CRYPTO_ACC_STATUS_CIPHER_ALGO_NOT_SUPPORTED,
IX_CRYPTO_ACC_STATUS_CIPHER_MODE_NOT_SUPPORTED,
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_KEY_LEN,
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_IV_LEN,
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_BLOCK_LEN,
IX_CRYPTO_ACC_STATUS_AUTH_ALGO_NOT_SUPPORTED,
IX_CRYPTO_ACC_STATUS_AUTH_INVALID_DIGEST_LEN,
IX_CRYPTO_ACC_STATUS_AUTH_INVALID_KEY_LEN,
IX_CRYPTO_ACC_STATUS_AUTH_FAIL,
IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID,
IX_CRYPTO_ACC_STATUS_EXCEED_MAX_TUNNELS,
IX_CRYPTO_ACC_STATUS_AUTH_INVALID_AAD_LEN,
IX_CRYPTO_ACC_STATUS_NULL_PTR_ERR,
IX_CRYPTO_ACC_STATUS_OUT_OF_RANGE_ERR
}
Status Definitions.

Functions

PUBLIC **IxCryptoAccStatus ixCryptoAccConfig** (**IxCryptoAccCfg** compCfg)
Selects which interfaces need to be initialized when crypto−access init is called.

PUBLIC **IxCryptoAccStatus ixCryptoAccInit** (void)
Initialise the Security Access component.

PUBLIC **IxCryptoAccStatus ixCryptoAccUninit** (void)
Uninitialise the Security Access component.

PUBLIC **IxCryptoAccStatus ixCryptoAccCtxRegister** (**IxCryptoAccCtx** *pAccCtx, IX_OSAL_MBUF *pMbufPrimaryChainVar, IX_OSAL_MBUF *pMbufSecondaryChainVar, **IxCryptoAccRegisterCompleteCallback** registerCallbackFn, **IxCryptoAccPerformCompleteCallback** performCallbackFn, UINT32 *pCryptoCtxId)
Crypto context registration. Cryptographic Context ID (cryptoCtxId) for the registered crypto context obtained from this registration request will be used in perform service requests.

PUBLIC **IxCryptoAccStatus ixCryptoAccCtxUnregister** (UINT32 cryptoCtxId)
Unregister the crypto context from Cryptographic Context Database.

PUBLIC **IxCryptoAccStatus ixCryptoAccAuthCryptPerform** (UINT32 cryptoCtxId, IX_OSAL_MBUF *pSrcMbuf, IX_OSAL_MBUF *pDestMbuf, UINT16 authStartOffset, UINT16 authDataLen, UINT16 cryptStartOffset, UINT16 cryptDataLen, UINT16 icvOffset, UINT8 *pIV)
Perform Authentication and Decryption/Encryption functionalities.

PUBLIC **IxCryptoAccStatus ixCryptoAccNpeWepPerform** (UINT32 cryptoCtxId, IX_OSAL_MBUF *pSrcMbuf, IX_OSAL_MBUF *pDestMbuf, UINT16 startOffset, UINT16 dataLen, UINT16 icvOffset, UINT8 *pKey)
Function to invoke ARC4 and WEP ICV computations on NPE.

PUBLIC **IxCryptoAccStatus ixCryptoAccXScaleWepPerform** (UINT32 cryptoCtxId, IX_OSAL_MBUF *pSrcMbuf, IX_OSAL_MBUF *pDestMbuf, UINT16 startOffset, UINT16 dataLen, UINT16 icvOffset, UINT8 *pKey)
Function to support ARC4 and WEP ICV computations on Intel XScale(R) Processor.

PUBLIC **IxCryptoAccStatus ixCryptoAccCtxCipherKeyUpdate** (UINT32 cryptoCtxId, UINT8 *pCipherKey)
Change keys for the specified registered context.

PUBLIC void **ixCryptoAccShow** (void)
API for printing statistics and status.

PUBLIC void **ixCryptoAccShowWithId** (UINT32 cryptoCtxId)
API for printing statistic and status.

PUBLIC **IxCryptoAccStatus ixCryptoAccCryptoServiceStop** (void)

Function to stop the crypto services.

PUBLIC **IxCryptoAccStatus ixCryptoAccHashKeyGenerate** (**IxCryptoAccAuthAlgo** hashAlgo, IX_OSAL_MBUF *pMbufHashKey, **IxCryptoAccHashKeyGenCompleteCallback** hashKeyCallbackFn, UINT16 hashKeyStartOffset, UINT16 hashKeyLen, UINT16 hashKeyDestOffset, UINT32 *pHashKeyId)

This function is used to generate authentication key needed in HMAC authentication if the authentication key is greater than 64 bytes. New authentication key of L bytes size will be generated in this function (L = 20 for SHA1, L = 16 for MD5). Please refer to RFC2104 for more details on the key size. The authentication key is padded (extended) so that its length (in bits) is congruent to 448, modulo 512 (please refer to RFC1321 & RFC3174). Authentication key is padded inside cryptoAcc on the mbuf which holds the authentication key. Therefore, mbuf allocated to store authentication key need to take into account the space needed for padding process. Padding space (extra memory allocation) needed is between the range of 9 − 72 bytes, it varies based on the authentication key size specified by client. Minimum of 9 bytes are needed, as 8 bytes is for total length of authentication key (hash data), and 1 bytes is for start of padding indication. The extra padding space is derived from the padding scheme stated in RFC1321 & RFC3174.

PUBLIC **IxCryptoAccStatus ixCryptoAccPkeHashPerform** (**IxCryptoAccAuthAlgo** hashAlgo, UINT8 *pHashData, UINT32 hashDataLen, **IxCryptoAccPkeHashPerformCompleteCallback** hashPerformCallbackFn, UINT8 *pDigest)

This function is used to hash the input data and return the digest generated via callback function supplied by client. L bytes size of digest will be generated (L = 20 for SHA1). The hash data is padded inside the function so that its length (in bits) is congruent to 448, modulo 512 (please refer to RFC1321). The length of data in bytes to be hashed must be greater than 0.

PUBLIC **IxCryptoAccStatus ixCryptoAccPkePseudoRandomNumberGet** (UINT32 pseudoRandomNumberLen, UINT32 *pPseudoRandomNumber)

This function will return the pseudo−random number generated in hardware to the client via the data pointer supplied. The length of pseudo−random number requested must be in words count (pseudoRandomNumberLen >= 1 word). If bigger pseudo−random number is requested, then this function will take longer time to complete.

PUBLIC **IxCryptoAccStatus ixCryptoAccPkeEauExpConfig** (BOOL enableSE, BOOL enableFE)

*This function is used to configure short exponent setting (SE − skipping leading 0s while performing modular exponential) and fast exponent setting (FE − skipping all 0s while performing modular exponential) for EAU modular exponential operation. By default, these 2 settings are DISABLED to prevent timing attacks on keys and also ensure best protection on keys. These 2 setting will have direct impact on function call to **ixCryptoAccPkeEauPerform**. Client may change it accordingly to achieve different level of protection and also performance.*

PUBLIC **IxCryptoAccStatus ixCryptoAccPkeEauPerform** (**IxCryptoAccPkeEauOperation** operation,
IxCryptoAccPkeEauInOperands *pOpr,
IxCryptoAccPkeEauPerformCompleteCallback eauPerformCallbackFn,
IxCryptoAccPkeEauOpResult *pResult)
This function is used to performed large number exponential arithmetic operation requested by client by using EAU (Exponentiation Acceleration Unit) on PKE (Public Key Exchange) Crypto Engine. The result computed will be returned via callback function supplied by client. All the operand size must be specified in words (32−bit), the output data pointer must be big enough to hold the result.

Detailed Description

IXP400 Security component Public API.

Define Documentation

#define IX_CRYPTO_ACC_3DES_KEY_192

3DES key length in bytes

Definition at line **106** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_BLOCK_128

AES cipher block length in bytes.

Definition at line **123** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_CBC_IV_128

AES initialization vector length in bytes for CBC mode.

Definition at line **126** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_CCM_IV_512

CCM Initialization vector has 2 components to it.

First is the IV for the CTR mode encryption and second is the additional authentication data for the CBC−MAC generation.

Definition at line **143** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_CTR_IV_128

AES initialization vector length in bytes for CTR mode.

Definition at line **130** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_KEY_128

AES−128 key length in bytes.

Definition at line **114** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_KEY_192

AES−192 key length in bytes.

Definition at line **117** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_AES_KEY_256

AES−256 key length in bytes.

Definition at line **120** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_ARC4_BLOCK_8

ARC4 algorithm block size.

Definition at line **160** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_ARC4_KEY_128

Key length for ARC4 algorithms.

Definition at line **157** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_CCM_AAD_LEN_384

48 Bytes of additional authenticated data for CBC−MAC computation

Currently only this value is supported.

Definition at line **135** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_CCM_DIGEST_64

CCM digest length in bytes.

Currently only this value (of 8 bytes)is supported for CCM digest length.

Definition at line **192** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_DES_BLOCK_64

DES cipher block length in bytes.

Definition at line **99** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_DES_IV_64

DES initialization vector length in bytes.

Definition at line **102** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_DES_KEY_64

DES key length in bytes.

Definition at line **96** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_ACTIVE_SA_TUNNELS

Maximum active tunnels supported could be changed by the client based on the application's requirements.

Number of active tunnels will not impact the overall performance but will have an impact on the memory needed to keep the crypto context information. Overall memory requirement depends on the number of tunnels.

Definition at line **75** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_AUTH_IV_LENGTH

Max length (byte) of initial chaining variable for SHA1 (160 bit) and MD5 (128 bit).

Definition at line **65** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH

Max length (byte) of authentication key for SHA1 and MD5.

Key size > 64 bytes need to be hashed to produce shorter key by calling API **ixCryptoAccHashKeyGenerate()**. (SHA1 : 20 bytes, MD5 : 16 bytes by default according to RFC2104. If L <= key size <= 64 bytes (L=16 for MD5 & 20 for SHA1) authentication key can be used for registration directly. If key size < L, client will need to pad the key with 0s to length L before calling **ixCryptoAccCtxRegister()** API.

Definition at line **44** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_CIPHER_IV_LENGTH

Max IV length in bytes.

Definition at line **41** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_CIPHER_KEY_LENGTH

Max length (byte) of cipher key for DES (64 bit), 3DES (192 bit), AES (128, 192 & 256 bit).

Definition at line **35** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MAX_QUEUE_DEPTH

Max queue depth supported by the Queue Manager.

Definition at line **71** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MD5_DIGEST_128

MD5 message digest length in bytes.

Definition at line **184** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_MD5_KEY_128

MD5 key length in bytes.

Definition at line **181** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_SHA1_DIGEST_160

SHA1 message digest length in bytes.

Definition at line **171** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_SHA1_KEY_160

SHA1 key length in bytes.

Definition at line **168** of file **IxCryptoAcc.h**.

#define IX_CRYPTO_ACC_WEP_CRC_DIGEST_32

Digest length of WEP ICV in bytes.

Definition at line **204** of file **IxCryptoAcc.h**.

#define ixCryptoAccHashPerform

alias function name to **ixCryptoAccHashKeyGenerate**

Definition at line **1692** of file **IxCryptoAcc.h**.

Typedef Documentation

typedef void(* IxCryptoAccHashKeyGenCompleteCallback)(UINT32 hashKeyId, IX_OSAL_MBUF *pMbufHashKey, **IxCryptoAccStatus** status)

Hash key generation complete callback notification.

This function is called to notify a client that the hash key has been generated. This function will return the status through the associated hashKeyId once the key is calculated by Network Processor Engine (NPE).

hashKeyId becomes invalid once the notification callback is called.

Parameters:

hashKeyId	UINT32 [in] − This hashKeyId is provided when client sends in request to hash the key.
**pMbufHashKey*	[in] − Pointer to the mbuf that contains original key and generated hash key. Client will need to copy the generated hash key from the mbuf into crypto context and used it as hash key for crypto registration request. This mbuf will be freed by client.
status	**IxCryptoAccStatus** [in] − Status (IxCryptoAccStatus) reporting to client.

Note:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − hash key generation is successful.

◊ IX_CRYPTO_ACC_STATUS_FAIL − hash key generation failed.

Definition at line **715** of file **IxCryptoAcc.h**.

typedef void(* IxCryptoAccPerformCompleteCallback)(UINT32 cryptoCtxId, IX_OSAL_MBUF *pSrcMbuf, IX_OSAL_MBUF *pDestMbuf, **IxCryptoAccStatus** status)

Hardware accelerator service request complete callback notification.

This function is called to notify a client that the cryptographic transaction has been completed. The cryptoCtxId and status of completed operation are returned to the client through this callback function to indicate operation which crypto context has been completed. The CryptoCtxId is obtained via ixCryptoAccCtxRegister, and this ID is unique to a particular IPSec tunnel.

Parameters:

cryptoCtxId UINT32 [in] − This crypto context ID is provided when client sends in request via ixCryptoAccCtxRegister API to register crypto context. cryptoCtxId points to a struct consists of cryptographic parameters required by the Network Processor Engine (NPE) in CCD database.

**pSrcMbuf* [in] − Pointer to the source mbuf which contains the data to be processed. It is also the output mbuf which contains the processed data if UseDifferentSrcAndDestMbufs is FALSE.

**pDestMbuf* [in] − Only used if UseDifferentSrcAndDestMbufs is TRUE. Pointer to the output mbuf which contains processed data.

status **IxCryptoAccStatus** [in] − Status reporting to the client via IxCryptoAccStatus.

Note:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Operation is completed successfully.

◊ IX_CRYPTO_ACC_STATUS_FAIL − Operation failed.

◊ IX_CRYPTO_ACC_STATUS_AUTH_FAIL − Authentication is unsuccessful. Note that when authentication fails, the content of the destination mbuf (which is the same as the source mbuf if a normal in−place operation is performed) will be undetermined.

Definition at line **756** of file **IxCryptoAcc.h**.

typedef **IxCryptoAccPkeEauOperand** IxCryptoAccPkeEauOpResult

Structure storing result of EAU operation.

Definition at line **586** of file **IxCryptoAcc.h**.

typedef void(* IxCryptoAccPkeEauPerformCompleteCallback)(**IxCryptoAccPkeEauOperation** operation, **IxCryptoAccPkeEauOpResult** *pResult, BOOL carryOrBorrow, **IxCryptoAccStatus** status)

EAU perform complete callback notification.

This function is called to notify a client that the result from EAU operation is ready. This function will return the status, result computed (IxCryptoAccPkeEauOpResult) and carry/borrow flag as a result of add/sub operation.

Parameters:

operation	**IxCryptoAccPkeEauOperation** [in] − EAU operation performed.
**pResult*	**IxCryptoAccPkeEauOpResult** [in] − pointer to a structure which store EAU operation result.
carryOrBorrow	BOOL [in] − carry/borrow bit status as resulted from large number addition/subtraction operation.
status	**IxCryptoAccStatus** [in] − Status (IxCryptoAccStatus) reporting to client.

Note:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − EAU operation is successful.

◊ IX_CRYPTO_ACC_STATUS_FAIL − EAU operation failed.

Definition at line **810** of file **IxCryptoAcc.h**.

typedef void(* IxCryptoAccPkeHashPerformCompleteCallback)(UINT8 *pDigest, **IxCryptoAccStatus** status)

PKE Crypto Engine SHA engine hashing perform complete callback notification.

This function is called to notify a client that the digest from hashing on AHB−SHA engine in PKE Crypto Engine has been generated. This function will return the status and digest calculated.

Parameters:

**pDigest*	UINT8 [in] − Pointer to the flat buffer that contains computed digest.
status	**IxCryptoAccStatus** [in] − Status (IxCryptoAccStatus) reporting to client.

Note:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − hash digest generation is successful.

◊ IX_CRYPTO_ACC_STATUS_FAIL − hash digest generation failed.

Definition at line **782** of file **IxCryptoAcc.h**.

typedef void(* IxCryptoAccRegisterCompleteCallback)(UINT32 cryptoCtxId, IX_OSAL_MBUF *pMbuf, **IxCryptoAccStatus** status)

Cryptographic Context registration complete callback notification.

This function is called to notify a client that the Crypto Context has been registered. This function returns status through the associated cryptoCtxId once the initial values needed are calculated by Network Processor Engine (NPE) and stored in Cryptographic Context Database (CCD).

The CryptoCtxId is valid until ixCryptoAccCtxUnregister is invoked.

If the callback function returns the IX_CRYPTO_ACC_STATUS_WAIT status, it indicates that registration is not complete yet, but the mbuf pointer needs to be freed by client. Client needs to wait for the next completion indication. Registration complete successfully only if status IX_CRYPTO_ACC_STATUS_SUCCESS is received.

Parameters:

cryptoCtxId UINT32 [in] − This crypto context ID is provided when client sends in request via ixCryptoAccCtxRegister API to register crypto context. cryptoCtxId points to a struct consists of cryptographic parameters required by the Network Processor Engine (NPE) in CCD database.

**pMbuf* [in] − Pointer to the mbuf (to be freed by client). The client should free any mbuf that is not NULL.

status **IxCryptoAccStatus** [in] − Status (IxCryptoAccStatus) reporting to client.

Note:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − registration is successful.

◊ IX_CRYPTO_ACC_STATUS_FAIL − registration failed.

◊ IX_CRYPTO_ACC_STATUS_WAIT − registration is not complete yet, wait for next completion indication. NPE is busy in calculating the initial variables needed for the registration.

Definition at line **683** of file **IxCryptoAcc.h**.

Enumeration Type Documentation

enum IxCryptoAccAuthAlgo

Authentication Algorithm Definitions.

Note:

Only two authentication algorithms are supported, SHA1 and MD5 for IPSEC application, while WEP_CRC is only for WEP application.

Only SHA1 is supported in AHB−SHA engine in PKE Crypto Engine

Enumeration values:

IX_CRYPTO_ACC_AUTH_NULL NULL authentication.

IX_CRYPTO_ACC_AUTH_SHA1 SHA1 algorithm.

IX_CRYPTO_ACC_AUTH_MD5	MD5 algorithm.
IX_CRYPTO_ACC_AUTH_CBC_MAC	CBC MAC algorithm only applicable along with CCM cipher mode.
IX_CRYPTO_ACC_AUTH_WEP_CRC	WEP CRC algorithm.
IX_CRYPTO_ACC_AUTH_TYPE	Maximum value for types of authentication algorithm.

Definition at line **345** of file **IxCryptoAcc.h**.

enum IxCryptoAccCfg

Possible configuration definitions.

Note:

Enums to define various possible configurations.

PKE acceleration support (by using EAU, RNG and AHB−SHA engines) is enabled by default if the silicone variants support that.

Enumeration values:

IX_CRYPTO_ACC_CFG_WEP_XSCALE_ACC_EN	Enable access to WEP Intel XScale(R) Processor processing only. Access to WEP−NPE and hardware accelerators are disabled
IX_CRYPTO_ACC_CFG_CRYPTO_NPE_ACC_EN	Enable access to Hardware accelerators (WEP Intel XScale(R) Processor is also enabled). This is the default configuration for the cryptoAcc component.
IX_CRYPTO_ACC_CFG_WEP_NPE_ACC_EN	Enable access to WEP NPE and Intel XScale(R) Processor.
IX_CRYPTO_ACC_CFG_CRYPTO_WEP_NPE_ACC_EN	Enable access to the Hardware accelerators, WEP − NPE and Intel XScale(R) Processor.
IX_CRYPTO_ACC_CFG_TYPE	Maximum value for types of configurations.

Definition at line **223** of file **IxCryptoAcc.h**.

enum IxCryptoAccCipherAlgo

Cipher Algorithm Definitions.

Note:

3DES and AES will be supported if not violating import/export rules.

Enumeration values:

IX_CRYPTO_ACC_CIPHER_NULL	NULL encryption.
IX_CRYPTO_ACC_CIPHER_DES	DES algorithm.
IX_CRYPTO_ACC_CIPHER_3DES	Triple DES algorithm.
IX_CRYPTO_ACC_CIPHER_AES	AES algorithm.
IX_CRYPTO_ACC_CIPHER_ARC4	ARC4 Algorithm.
IX_CRYPTO_ACC_CIPHER_ALGO_TYPE	Maximum value for types of cipher algorithm.

Definition at line **294** of file **IxCryptoAcc.h**.

enum IxCryptoAccCipherMode

Cipher Mode Definitions.

Note:

CFB and OFB are not supported.

Enumeration values:

IX_CRYPTO_ACC_MODE_NULL	NULL cipher mode.
IX_CRYPTO_ACC_MODE_ECB	ECB mode of operation.
IX_CRYPTO_ACC_MODE_CBC	CBC mode of operation.
IX_CRYPTO_ACC_MODE_CTR	CTR mode of operation, only applicable to AES.
IX_CRYPTO_ACC_MODE_CCM	CCM mode of operation, only applicable with AES cipher Algo.
IX_CRYPTO_ACC_MODE_TYPE	Maximum value for types of operation.

Definition at line **316** of file **IxCryptoAcc.h**.

enum IxCryptoAccOperation

Cryptographic Operation Definitions.

Enumeration values:

IX_CRYPTO_ACC_OP_ENCRYPT	Encrypt operation.
IX_CRYPTO_ACC_OP_DECRYPT	Decrypt operation.
IX_CRYPTO_ACC_OP_AUTH_CALC	Authentication calculation operation.
IX_CRYPTO_ACC_OP_AUTH_CHECK	Authentication verification operation.
IX_CRYPTO_ACC_OP_ENCRYPT_AUTH	Encryption followed by authentication calculation operation.
IX_CRYPTO_ACC_OP_AUTH_DECRYPT	Authentication verification followed by decryption.
IX_CRYPTO_ACC_OP_TYPE_OF_OPERATION	Maximum value for types of operation.

Definition at line **258** of file **IxCryptoAcc.h**.

enum IxCryptoAccPkeEauOperation

EAU (Exponentiation Acceleration Unit) Operation Definitions.

Enumeration values:

IX_CRYPTO_ACC_OP_EAU_MOD_EXP	Modular exponential operation.
IX_CRYPTO_ACC_OP_EAU_BN_MUL	Large number multiplication operation.
IX_CRYPTO_ACC_OP_EAU_BN_ADD	Large number addition operation.
IX_CRYPTO_ACC_OP_EAU_BN_MOD	Large number modular reduction operation.
IX_CRYPTO_ACC_OP_EAU_BN_SUB	Large number subtraction operation.
IX_CRYPTO_ACC_OP_TYPE_OF_EAU_OPERATION	Maximum value for types of EAU operation.

Definition at line **367** of file **IxCryptoAcc.h**.

enum IxCryptoAccStatus

Status Definitions.

Note:

These status will be used by the APIs to return to the client.

Enumeration values:

IX_CRYPTO_ACC_STATUS_SUCCESS	Success status.
IX_CRYPTO_ACC_STATUS_FAIL	Fail status.
IX_CRYPTO_ACC_STATUS_WAIT	Wait status.
IX_CRYPTO_ACC_STATUS_RETRY	Retry status.
IX_CRYPTO_ACC_STATUS_QUEUE_FULL	Queue full.
IX_CRYPTO_ACC_STATUS_OPERATION_NOT_SUPPORTED	Invalid operation.
IX_CRYPTO_ACC_STATUS_CIPHER_ALGO_NOT_SUPPORTED	Invalid cipher algorithm.
IX_CRYPTO_ACC_STATUS_CIPHER_MODE_NOT_SUPPORTED	Invalid cipher mode of operation.
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_KEY_LEN	Invalid cipher key length.
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_IV_LEN	Invalid IV length.
IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_BLOCK_LEN	Invalid cipher block length.
IX_CRYPTO_ACC_STATUS_AUTH_ALGO_NOT_SUPPORTED	Invalid authentication algorithm.
IX_CRYPTO_ACC_STATUS_AUTH_INVALID_DIGEST_LEN	Invalid message digest length.
IX_CRYPTO_ACC_STATUS_AUTH_INVALID_KEY_LEN	Invalid authentication key length.
IX_CRYPTO_ACC_STATUS_AUTH_FAIL	Authentication verification failed.
IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID	Invalid crypto context ID.
IX_CRYPTO_ACC_STATUS_EXCEED_MAX_TUNNELS	Exceed maximum number of crypto contexts allocation.

IX_CRYPTO_ACC_STATUS_AUTH_INVALID_AAD_LEN	Invalid additional authentication length.
IX_CRYPTO_ACC_STATUS_NULL_PTR_ERR	buffer pointer or callback function pointer is NULL
IX_CRYPTO_ACC_STATUS_OUT_OF_RANGE_ERR	Operand size or buffer size is less than minimum size allowed or greater than maximum size allowed.

Definition at line **398** of file **IxCryptoAcc.h**.

Function Documentation

```
IxCryptoAccStatus ixCryptoAccAuthCryptPerform ( UINT32          cryptoCtxId,
                                                IX_OSAL_MBUF *  pSrcMbuf,
                                                IX_OSAL_MBUF *  pDestMbuf,
                                                UINT16          authStartOffset,
                                                UINT16          authDataLen,
                                                UINT16          cryptStartOffset,
                                                UINT16          cryptDataLen,
                                                UINT16          icvOffset,
                                                UINT8 *         pIV
                                              )
```

Perform Authentication and Decryption/Encryption functionalities.

This function is called for authentication and decryption/encryption functionalities service request. For a combined encryption and authentication request with cipher algorithms as AES or DES and authentication algorithm as MD5 and SHA1 the following restriction must be met: That the crypted data must be a subset of the authenticated data. The boundary relationship of [(authStartOffset + authDataLen) >= (cryptDataLen + cryptStartOffset) >= (cryptStartOffset >= authStartOffset] MUST BE SATISFIED. There should not be any chained mbuf boundary within an ICV field.

For performing ARC4 and WEP ICV computations call the ixCryptoAccNpeWepPerform or ixCryptoAccXScaleWepPerform functions. The ARC4 and WEP ICV operations are not supported by this function.

A valid cryptoCtxId must be obtained via **ixCryptoAccCtxRegister** API in order to proceed to perform the functionalities above.

Parameters:

cryptoCtxId	UINT32 [in] − is the crypto context pointer to be supplied by the client. This cryptoCtxId is obtained via ixCryptoAccCtxRegister. The cryptoCtxId must be a valid Id.
**pSrcMbuf*	[in] − is a pointer to mbuf which contains data to be processed. This mbuf structure is allocated by client. Result of this request will be stored in the same mbuf and overwritten the original data if UseDifferentSrcAndDestMbufs flag in **IxCryptoAccCtx** is set to FALSE (in−place operation). Otherwise, if

UseDifferentSrcAndDestMbufs flag is set to TRUE, the result will be written into destination mbuf (non in−place operation) and the original data in this mbuf will remain unchanged. The same pointer is then returned to the client via registered IxCryptoAccPerformCompleteCallback callback function.

**pDestMbuf* [in] − only used if UseDifferentSrcAndDestMbufs is TRUE. This is the buffer where the result is written to. This mbuf structure is allocated by client. The length of mbuf MUST be big enough to hold the result of operation. The result of operation COULD NOT span into two or more different mbufs, thus the mbuf supplied must be at least the length of expected result. The same pointer is then returned to the client via registered IxCryptoAccPerformCompleteCallback callback function. The data is written back at the 'authStartOffset',if it's a authentication only or a combined request. If it's a crypt only request the data is written starting from 'cryptStartOffset'.

authStartOffset UINT16 [in] − supplied by the client to indicate the start of the payload to be authenticated. Ignored when performing request with Cipher Mode set to CCM.

authDataLen UINT16 [in] − supplied by the client to indicate the length of the payload to be authenticated in Bytes. The maximum data length must not exceed 65471 bytes.Ignored when performing request with Cipher Mode set to CCM.

cryptStartOffset UINT16 [in] − supplied by the client to indicate the start of the payload to be decrypted/encrypted.

cryptDataLen UINT16 [in] − supplied by the client to indicate the length of the payload to be decrypted/encrypted in Bytes. The payload to be decrypted/encrypted must have the length that is multiple of cipher block length in size.

icvOffset UINT16 [in] − supplied by the client to indicate the start of the ICV (Integrity Check Value) used for the authentication. This ICV field should not be split across multiple mbufs in a chained mbuf.

**pIV* UINT8 [in] − Initialization Vector supplied by the client to be used for the decryption/encryption processes. For CCM mode of operation, this parameter points to the CTR−IV followed by the 16 bytes of initial block (called − B0, in RFC 3610 and CCM Initial block in 802.11i spec), followed by additional authentication data (lengths − defined in respective standards).

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Operation requested is successfully enqueued to hardware accelerator for processing.

◊ IX_CRYPTO_ACC_STATUS_FAIL − Cryptographic process failed for some unspecified internal reasons.

◊ IX_CRYPTO_ACC_STATUS_QUEUE_FULL − Cryptographic queue is full.

◊ IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_BLOCK_LEN − Invalid plaintext / ciphertext block size passed in by the client.

◊ IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID − Crypto context is not registered.

Note:

◊ Client shall handle IP mutable fields.

◊ Client shall pad the IP datagram to be a multiple of cipher block size, using ESP trailer for encryption (RFC2406, explicit padding)
◊ NPE shall pad the IP datagram to be a multiple of hashing block size, specified by the authentication algorithm (RFC2402, implicit padding)
◊ For authentication generation operation, client needs to clear the field which hold the authentication data (ICV) to zeroes. While for the authentication verification operation, client needs to supply the authentication data at the ICV field for NPE to verify the ICV value. If ICV is embedded in the payload, client DOES NOT need to move the ICV to the front / the back of the payload and clear the original ICV field. cryptoAcc access component will handle this.
◊ Client shall construct the AES CTR counter block (4 bytes NONCE + 8 bytes IV + 4 bytes counter). CTR counter block implementation should be based on internet draft / RFC for IPSEC AES CTR mode recommendation, which can be found at IETF website, www.ietf.org. Counter block will be passed in as IV in this case.
◊ Depending upon the operating system and task switching mechanism, the operation might "seem" to complete before this function returns to the caller. However, the return status with value success only implies that the request was succesfully enqueued to the hardware accelerator. The call of performCallBackFn indicates the completion of operation by the hardware engine.

Assumption:

- Different clients should NOT hold the same crypto context. All the requests for same crypto context should be initiated from same client (thread).
- This function is reentrant, and the client could submit multiple requests for the same crypto context.

- Reentrant : yes
- ISR Callable : yes

IxCryptoAccStatus ixCryptoAccConfig (**IxCryptoAccCfg** *compCfg*)

Selects which interfaces need to be initialized when crypto−access init is called.

This function will set the configuration bits for initializing the interfaces to the Hardware Accelerator engines on NPE−C and WEP Encryption engine on the WAN−NPE. If this function is not called then, only the Hardware Accelerator engine is initialized by default in the crypto−access init function. This function should be called prior to call ixCyptoAccInit function.

```
Consideration on using WEP encryption engine on Intel XScale(R) Processor & WAN−NPE
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
1. Enabling WEP encryption engine on Intel XScale(R) Processor
    Pros:
      i.  Higher throughput compared to WEP running on WAN−NPE.
      ii. Still can process WEP request on Intel XScale(R) Processor if WEP engine
          on WAN−NPE is not available due to hardware is not present or some
          interfaces(e.g. ATM/HSS/DMA) on the NPE that is being used.
    Cons:
      i.  Lesser memory headroom and CPU bandwidth on Intel XScale(R) Processor
          for application stack.
    Note: By default WEP encryption engine on Intel XScale(R) Processor is always
          enabled for all options.
```

```
2. Enabling WEP encryption engine on WAN−NPE
    Pros:
      i.  Can process WEP request via WEP engines on both Intel XScale(R)
          Processor and WAN−NPE.
    Cons:
      i.  Whole WAN−NPE is dedicated for WEP processing, no other interfaces
          (such as ATM, HSS, DMA) can be enabled for this case.
      ii. Lower throughput rate compared to WEP on Intel XScale(R) Processor
          due to NPE core running at lower frequency compared to
          Intel XScale(R) Processor.
    Note: By default WEP encryption on WAN NPE is disabled.
```

Note:

PKE acceleration support (by using EAU, RNG and AHB−SHA engines) is enabled by default if the network processor on the system has PKE bridge. Client does not need to enable it explicitly.

Parameters:

compCfg **IxCryptoAccCfg** [in] − Indicates the appropriate interfaces to be initialized for cryptoAcc access component.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Setting the configuration was successful.

◊ IX_CRYPTO_ACC_STATUS_FAIL− The configuration setting was invalid.
◊ Reentrant : no
◊ ISR Callable : yes

IxCryptoAccStatus ixCryptoAccCryptoServiceStop (void)

Function to stop the crypto services.

Warning:

THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED. It will be removed in future release. See **ixCryptoAccUninit** for more information.

This function is called to stop the Hardware Accelerator services and (if enabled) WEP engine services. All the requests pending in queues will be completed before the services are shutdown. Any new requests issued after this call will be rejected. All the Crypto Contexts will be unregistered in this function call.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Operation requested is successful with all the pending requests are completed and CCD is cleared.

◊ IX_CRYPTO_ACC_STATUS_FAIL − Cryptographic stop request failed for some unspecified internal reasons.
◊ Reentrant : no
◊ ISR Callable : no

IxCryptoAccStatus ixCryptoAccCtxCipherKeyUpdate (UINT32 *cryptoCtxId*,
UINT8 * *pCipherKey*
)

Change keys for the specified registered context.

This function is called to change the key value of a previously registered context. Key change for a registered context is only supported for CCM cipher mode. This is done in order to quickly change keys for CCM mode, without going through the process of context deregistration and registration. Changes to the key lengths are not allowed for a registered context. This function should only be used if one is invoking cryptographic operations using CCM as cipher mode. For contexts registered with other modes the the client should unregister and re−register a context for the particular security association in order to change keys and other parameters. The client should make sure that there are no pending requests on the "cryptoCtxtId" for the key change to happen successfully. If there are pending requests on this context the result of those operations are undefined.

Parameters:

cryptoCtxId UINT32 [in] − The previously registered context id. This context should be registered for CCM cipher mode. Keys updates for other types of registered cipher modes are not supported at this time.

**pCipherKey* UINT8 [in] − The new key value. The number of bytes expected in keys is the "cipherKeyLen" used during the registration of the context.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS : Keys were changed successfully.

◊ IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID: cryptoCtxtId is not valid.

◊ IX_CRYPTO_ACC_STATUS_OPERATION_NOT_SUPPORTED: Returned when the context is not registered for doing operations on NPE C or if the operation was anything other than CCM.

◊ IX_CRYPTO_ACC_STATUS_FAIL: If the component has not been initialized.
◊ Reentrant : yes
◊ ISR Callable : yes

IxCryptoAccStatus
ixCryptoAccCtxRegister (**IxCryptoAccCtx** * *pAccCtx*,
IX_OSAL_MBUF * *pMbufPrimaryChainVar*,
IX_OSAL_MBUF * *pMbufSecondaryChainVar*,
IxCryptoAccRegisterCompleteCallback *registerCallbackFn*,
IxCryptoAccPerformCompleteCallback *performCallbackFn*,
UINT32 * *pCryptoCtxId*
)

Crypto context registration. Cryptographic Context ID (cryptoCtxId) for the registered crypto context obtained from this registration request will be used in perform service requests.

This function is used to register all the required information (eg. key, cipher algorithm, etc) for hardware accelerator services with IxCryptoAcc component. Those information will be stored in CCD database. All the information will be converted into a cryptographic parameters structure to be shared with NPE. The structure will be associated with a unique crypto context ID (cryptoCtxId). cryptoCtxId is passed back to client for future reference in callback function and also for hardware accelerator service requests. Besides, two empty mbuf are required to be passed in through interface for the use of access component and NPE to compute the primary and secondary initial chaining variables. pMbufPrimaryChainVar and pMbufSecondaryChainVar mbuf pointers must be NULL if authentication or combined service is not selected. When initializing authentication context for WEP CRC (i.e when auth algo is IX_CRYPTO_ACC_AUTH_WEP_CRC), variables authKeyLen and keys should be ignored. The authDigestLen should be set to IX_CRYPTO_ACC_WEP_CRC_DIGEST_32.

Client should send in perform service request based on the crypto context registered. 2 different applications are supported by calling different perform functions as below:

1. If the crypto context is registered for crypto hardware accelerator service (DES, 3DES, AES, MD5, SHA1), client could only call the crypto hw accelerator perform function listed below: ixCryptoAccAuthCryptPerform ()

2. If the crypto context is registered for WEP services (WEP_ARC4, WEP_CRC), client could only call the WEP perform functions listed below: ixCryptoAccNpeWepPerform () ixCryptoAccXScaleWepPerform ()

Note:

3 scenarios of crypto context registration depending on authentication key size (in bytes) (only applicable to HMAC−SHA1 and HMAC−MD5): 1. If (L <= key size <= 64), then call this API (ixCryptoAccCtxRegister) directly. L = 16 for MD5 and L = 20 for SHA1. 2. If (key size > 64), then authentication key needs to be hashed to become shorter key first by calling another API ixCryptoAccHashKeyGenerate (). Please follow steps below for this case

- ◊ Call ixCryptoAccHashKeyGenerate () to hash the authentication key
- ◊ Wait for the callback from ixCryptoAccHashKeyGenerate (). Copy generated authentication key from mbuf into **IxCryptoAccCtx** as authentication key.
- ◊ Call this API (ixCryptoAccCtxRegister)to register the context. 3. If (key size < L), client MUST pad the authentication key with 0s to become L bytes of key before calling this API (ixCryptoAccCtxRegister).

Context registration concept is not applicable to PKE acceleration operations (such as EAU perform, Pseudo−random Number Get, PKE SHA engine hashing).

Parameters:

**pAccCtx*	**IxCryptoAccCtx** [in] − is a pointer to hardware accelerator context. Information required in hardware accelerator context, such as key and algorithm configuration can be extracted from Security Association Database (SAD). Important Note: When the crypt algorithm is ARC4, the keys supplied during the registration process are ignored. The caller should pass the per packet keys in the ixCryptoAccXScaleWepPerform function's pIV parameter.

**pMbufPrimaryChainVar*	[in] − a pointer to an empty mbuf(must not be chained) for the use of access component to compute primary chaining variables for SHA1/MD5. This mbuf structure is allocated by the client (minimum size of the cluster required is 64 bytes), only if SHA1/MD5 is selected, and the mbuf pointer must be NULL if SHA1/MD5 is not selected. After the NPE complete the computation, the mbuf is returned separately through the client's registered callback.
**pMbufSecondaryChainVar*	[in] − a pointer to an empty mbuf(must not be chained) for the use of access component to compute secondary chaining variables for SHA1/MD5. This mbuf structure is allocated by the client (minimum size of the cluster required is 64 bytes), only if SHA1/MD5 is selected, and the mbuf pointer must be NULL if SHA1/MD5 is not selected. After the NPE complete the computation, the mbuf is returned separately through the client's registered callback.
registerCallbackFn	**IxCryptoAccRegisterCompleteCallback** [in] − callback function pointer to return crypto context registration status to client when the registration is complete. This cannot be NULL.
performCallbackFn	**IxCryptoAccPerformCompleteCallback** [in] − callback function pointer to return the processed buffer to the client with respect to the unique CryptoCtxId. This cannot be NULL, however this function will not be called if its the requested to be executed by the engine running on Intel XScale(R) Processor.
**pCryptoCtxId*	UINT32 [inout] − Crypto Context ID returned by access component for the crypto context registered.

Returns:

- ◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Registration parameters are valid
- ◊ IX_CRYPTO_ACC_STATUS_FAIL − Registration failed for some unspecified internal reasons.
- ◊ IX_CRYPTO_ACC_STATUS_OPERATION_NOT_SUPPORTED − Invalid operation requested by the client.
- ◊ IX_CRYPTO_ACC_STATUS_CIPHER_ALGO_NOT_SUPPORTED − Invalid cipher algorithm requested by the client.
- ◊ IX_CRYPTO_ACC_STATUS_CIPHER_MODE_NOT_SUPPORTED − Invalid cipher mode requested by the client.
- ◊ IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_KEY_LEN − Invalid cipher key length passed in by the client.
- ◊ IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_BLOCK_LEN − Invalid cipher block

length passed in by the client

◊ IX_CRYPTO_ACC_STATUS_CIPHER_INVALID_IV_LEN − Invalid IV length passed in by the client.

◊ IX_CRYPTO_ACC_STATUS_AUTH_ALGO_NOT_SUPPORTED − Invalid authentication algorithm requested by the client.

◊ IX_CRYPTO_ACC_STATUS_AUTH_INVALID_DIGEST_LEN − Invalid authentication digest length.

◊ IX_CRYPTO_ACC_STATUS_AUTH_INVALID_KEY_LEN − Invalid authentication key length.

◊ IX_CRYPTO_ACC_STATUS_EXCEED_MAX_TUNNELS − Exceed maximum tunnels permitted.

◊ IX_CRYPTO_ACC_STATUS_QUEUE_FULL − Queue full status returned and the registration request will be rejected.
◊ Reentrant : yes
◊ ISR Callable : no

IxCryptoAccStatus ixCryptoAccCtxUnregister (UINT32 *cryptoCtxId*)

Unregister the crypto context from Cryptographic Context Database.

This function is for freeing the particular crypto context (reference through CryptoCtxId) from the Cryptographic Context Database.

Parameters:

cryptoCtxId UINT32 [in] − is pointer to crypto context that is required to be freed, which will be supplied by the client. The CryptoCtxId must be a valid Id.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − successfully unregister the crypto context.

◊ IX_CRYPTO_ACC_STATUS_FAIL − Unregistration failed for some unspecified internal reasons, e.g. uninitialized.

◊ IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID − invalid crypto context.

◊ IX_CRYPTO_ACC_STATUS_RETRY − retry the unregister operation as there are still some ixCryptoAccAuthCryptPerform requests associated with the cryptoCtxId pending on the NPE queue.
◊ Reentrant : yes
◊ ISR Callable : no

Assumption:

- It is client's responsibility to ensure no pending requests for the crypto context before it proceeds to

unregister the crypto context.

```
IxCryptoAccStatus
ixCryptoAccHashKeyGenerate
                                ( IxCryptoAccAuthAlgo                    hashAlgo,
                                  IX_OSAL_MBUF *                         pMbufHashKey,
                                  IxCryptoAccHashKeyGenCompleteCallback  hashKeyCallbackFn,
                                  UINT16                                 hashKeyStartOffset,
                                  UINT16                                 hashKeyLen,
                                  UINT16                                 hashKeyDestOffset,
                                  UINT32 *                               pHashKeyId
                                )
```

This function is used to generate authentication key needed in HMAC authentication if the authentication key is greater than 64 bytes. New authentication key of L bytes size will be generated in this function (L = 20 for SHA1, L = 16 for MD5). Please refer to RFC2104 for more details on the key size. The authentication key is padded (extended) so that its length (in bits) is congruent to 448, modulo 512 (please refer to RFC1321 & RFC3174). Authentication key is padded inside cryptoAcc on the mbuf which holds the authentication key. Therefore, mbuf allocated to store authentication key need to take into account the space needed for padding process. Padding space (extra memory allocation) needed is between the range of 9 − 72 bytes, it varies based on the authentication key size specified by client. Minimum of 9 bytes are needed, as 8 bytes is for total length of authentication key (hash data), and 1 bytes is for start of padding indication. The extra padding space is derived from the padding scheme stated in RFC1321 & RFC3174.

This function has been extended to become a generic hashing function other than above mentioned functionality. It could be used to hash any key size or data that are greater than 0 and less than 65399 bytes. **In order to map to extention of functioality, this function has been aliased to a name ixCryptoAccHashPerform.** The hashing operation is carried out by NPE. The request will be sent to NPE for further processing.

Note:

1. This API will need to be called if the authentication key > 64 bytes. If key size is >=L and <= 64 bytes, authentication key is used directly in Crypto context for registration, no further hashing is needed. 2. This function must be called prior the Crypto Context registration if the key size > 64. Result from this function will be used as authentication key in the Crypto Context registration. 3. It is client's responsibility to ensure the mbuf is big enough to hold the result (generated authentication key) and for padding process stated. No checking on mbuf length will be done.

Parameters:

hashAlgo	**IxCryptoAccAuthAlgo** [in] − is hashing algorithm to be used in generating the hash key. hashAlgo used to generate the key must be the same algorithm that will be used in crypto register and crypto perform services.
**pMbufHashKey*	[in] − a pointer to an mbuf that contains the authentication key to be hashed. This mbuf structure is allocated by the client and the mbuf pointer must not be NULL. After the NPE complete the computation, the mbuf is returned separately through the client's registered callback. The authentication key generated will be stored in this mbuf, pointed by hashKeyDestOffset. This must NOT be a chained mbuf.
hashKeyCallbackFn	**IxCryptoAccHashKeyGenCompleteCallback** [in] − callback function pointer to be called when the hash key generation operation is completed. This

	cannot be NULL.
hashKeyStartOffset	UINT16 [in] − offset to the mbuf mdata which contain the original hash key.
hashKeyLen	UINT16 [in] − key size
hashKeyDestOffset	UINT16 [in] − offset to the mbuf mdata to store the generated result (authentication key)
**pHashKeyId*	UINT32 [inout] − Hash Key ID returned by access component to identify hash key generation.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Hash key generation parameters are valid

◊ IX_CRYPTO_ACC_STATUS_FAIL − Hash key generation failed.

◊ IX_CRYPTO_ACC_STATUS_AUTH_ALGO_NOT_SUPPORTED − Invalid hash algorithm requested by the client.

◊ IX_CRYPTO_ACC_STATUS_AUTH_INVALID_KEY_LEN − Invalid hash key length (0 == hashKeyLen or 65399 < hashKeyLen)

◊ IX_CRYPTO_ACC_STATUS_QUEUE_FULL − Queue full status returned and the authentication key generation request will be rejected.
◊ Reentrant : yes
◊ ISR Callable : no

IxCryptoAccStatus ixCryptoAccInit (void)

Initialise the Security Access component.

This function will initialize all the internal resources of IxCryptoAcc access component depending upon the configuration set by the ixCryptoAccConfig function. When Hardware Accelerator engine is specified, it will check whether the specified AES, DES osr hash coprocessor is present and issue a warning if not. When WEP engine is specified, it checks whether AAL coprocessor is present or not. This function will check whether PKE (public key exchange) Crypto Unit is present or not and issue warning if not. If PKE Crypto Engine does not exist, the subsequent call to PKE Crypto Engine related APIs will be rejected.

This function should be called before any request processing functions are invoked on the component. This function is called only once.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − successfully initialize the component; a warning is issued if coprocessor is not present.

◊ IX_CRYPTO_ACC_STATUS_FAIL − Initialization failed for some unspecified internal reasons.
◊ Reentrant : no
◊ ISR Callable : no

```
IxCryptoAccStatus ixCryptoAccNpeWepPerform ( UINT32            cryptoCtxId,
                                             IX_OSAL_MBUF *    pSrcMbuf,
                                             IX_OSAL_MBUF *    pDestMbuf,
                                             UINT16            startOffset,
                                             UINT16            dataLen,
                                             UINT16            icvOffset,
                                             UINT8 *           pKey
                                           )
```

Function to invoke ARC4 and WEP ICV computations on NPE.

This function submits the request to the NPE to perform ARC4 and WEP ICV operations. If the request was not submitted to the NPE, the function returns failure and the client has the ownership of the buffers. If the function returns success, then request is in the request queue or is being executed by the NPE and the client should not assume the ownership of the buffers. When the processing of the request completes, the performCallBackFn is called with the buffer pointers submitted initially and the client can then assume the ownership of the buffers. When WEP ICV verification operation is requested, the computed ICV is written out at the ICV location for in−place mode of operation.The supplied ICV is overwritten as a result. For non−inplace mode of operation, the ICV is written out at the destination mbuf and the original ICV from the source is left intact. This behavior is true irrespective whether it's WEP ICV verify or a combined

- ARC4 decrypt and WEP ICV request.

A valid cryptoCtxId must be obtained via **ixCryptoAccCtxRegister** API in order to proceed to perform the functionalities above.

Parameters:

cryptoCtxId	UINT32 [in] − is the crypto context id to be supplied by the client. This is obtained by calling ixCryptoAccCtxRegister function.
**pSrcMbuf*	[in] − is a pointer to mbuf which contains data to be processed. This mbuf structure is allocated by client. Result of this request will be stored in the same mbuf and overwritten the original data if UseDifferentSrcAndDestMbufs flag in **IxCryptoAccCtx** is set to FALSE (in−place operation). Otherwise, if UseDifferentSrcAndDestMbufs flag is set to TRUE, the result will be written into destination mbuf (non in−place operation) and the original data in this mbuf will remain unchanged.
**pDestMbuf*	[in] − only used if UseDifferentSrcAndDestMbufs is TRUE. This is the buffer where the result is written to. This mbuf structure is allocated by client. The length of mbuf MUST be big enough to hold the result of operation. The result of operation COULD NOT span into two or more different mbufs, thus the mbuf supplied must be at least the length of expected result. The data is written back starting at startOffset in the pDestMbuf.
startOffset	UINT16 [in] − supplied by the client to indicate the start of the payload to be decrypted/encrypted or authenticated.
dataLen	UINT16 [in] − supplied by the client to indicate the length of the payload to be decrypted/encrypted in number of bytes.
icvOffset	UINT16 [in] − supplied by the client to indicate the start of the ICV (Integrity Check Value) used for the authentication. This ICV field should not be split across multiple mbufs in a chained mbuf.
**pKey*	

UINT8 [in] − Pointer to IX_CRYPTO_ACC_ARC4_KEY_128 bytes of per packet ARC4 keys. This pointer can be NULL if the request is WEP IV gen or verify only.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − The parameters check passed and the request was submitted to the NPE.

◊ IX_CRYPTO_ACC_STATUS_FAIL − The request failed because there was a parameter inconsistency or an invalid valid operation was requested.

Note:

◊ Depending upon the operating system and task switching mechanism, the operation might "seem" to complete before this function returns to the caller. However, the return status with value success only implies that the request was succesfully enqueued to the hardware accelerator. The call of performCallBackFn indicates the completion of operation by the hardware engine.

- Reentrant : yes
- ISR Callable : yes

IxCryptoAccStatus ixCryptoAccPkeEauExpConfig (BOOL *enableSE*,
BOOL *enableFE*
)

This function is used to configure short exponent setting (SE − skipping leading 0s while performing modular exponential) and fast exponent setting (FE − skipping all 0s while performing modular exponential) for EAU modular exponential operation. By default, these 2 settings are DISABLED to prevent timing attacks on keys and also ensure best protection on keys. These 2 setting will have direct impact on function call to **ixCryptoAccPkeEauPerform**. Client may change it accordingly to achieve different level of protection and also performance.

Note:

EAU exponential configuration is global setting for cryptoAcc component. All the function call to **ixCryptoAccPkeEauPerform** will follow these setting when performing modular exponential operation.

Parameters:

enableSE BOOL [in] If TRUE, then SE is enabled.

enableFE BOOL [in] If TRUE, then FE is enabled.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − EAU exponential configuration is set successfully

◊ IX_CRYPTO_ACC_STATUS_FAIL − Configuration failed.
◊ Reentrant : no
◊ ISR Callable : yes

IxCryptoAccStatus ixCryptoAccPkeEauPerform	(**IxCryptoAccPkeEauOperation**	*operation*,
	IxCryptoAccPkeEauInOperands *	*pOpr*,
	IxCryptoAccPkeEauPerformCompleteCallback	*eauPerformCallbackFn*,
	IxCryptoAccPkeEauOpResult *	*pResult*
	)	

This function is used to performed large number exponential arithmetic operation requested by client by using EAU (Exponentiation Acceleration Unit) on PKE (Public Key Exchange) Crypto Engine. The result computed will be returned via callback function supplied by client. All the operand size must be specified in words (32−bit), the output data pointer must be big enough to hold the result.

```
 Large Number Computation operation supported on EAU
 ---------------------------------------------------------------------------
 1. Modular Exponential ( C = M^e MOD N )

3 words <= N <= 64 words (2048 bits)
1 word <= {M, e} <= 64 words with M < N and e < N in value
MSB and LSB of N = 1
sizeof(C) >= sizeof(N)

 2. Large Number Multiplication ( R = A * B )

1 word <= {A, B} <= 64 words (2048−bit)
sizeof(R) >= sizeof(A) + sizeof(B)

 3. Large Number Addition ( Carry | R = A + B )

1 word <= {A, B} <= 64 words (2048−bit)
sizeof(R) >= sizeof(A) or sizeof(B); whichever is greater
Carry bit might be set in operation of addition, carry bit status is
        passed to the client in callback

 4. Modular Reduction ( R = A MOD N )

3 word <= N <= 64 words (2048−bit)
1 word <= A <= 128 words (4096−bit) with sizeof(A) <= 2 * sizeof(N)
MSB of N = 1
sizeof(R) >= sizeof(N)

 5. Large Number Subtraction ( Borrow | R = A − B )

1 word <= {A, B} <= 64 words (2048−bit)
sizeof(R) >= sizeof(A) or sizeof(B); whichever is greater
Borrow bit might be set in operation of subtraction, borrow bit status
        is passed to the client in callback
```

Note:

1. All the result of exponential (C), add/sub/mul/mod (R) will be retured in callback via *pResult. 2. This API is designed to offload the large number operation needed for PKE. 3. Only one request can be processed by the EAU co−processor on PKE bridge at one time, queuing system was not implemented. 4. This API is only targeted for network processors Intel (R) IXP4XX Product Line of Network Processors which support it. 5. By default, short exponent (skipping leading 0s in modular exponential) and fast exponent (skipping all 0s in modular exponential) are disabled. This

is to prevent timing attacks on keys and also ensure best protection. Client may choose to enabled SE & FE by calling **ixCryptoAccPkeEauExpConfig** function. 6. It is client's responsibility to ensure that the result pointer buffer is cleared to zero for the case of buffer size is greater than the actual result returned.

Parameters:

operation	**IxCryptoAccPkeEauOperation** [in] − EAU operation requested.
**pOpr*	**IxCryptoAccPkeEauInOperands** [in] − a structure pointer which contains input operands (data pointers and data length). length. This data structure is allocated by the client and the data pointers inside the structure must not be NULL depending on EAU operation requested. The data length of operand also must be within the range supported.
eauPerformCallbackFn	**IxCryptoAccPkeEauPerformCompleteCallback** [in] − callback function pointer to be called when the EAU operation is completed. This cannot be NULL.
**pResult*	**IxCryptoAccPkeEauOpResult** [out] − a structure pointer which store EAU operation result. Size of output data pointer must be big enough to hold the result computed. This structure must remain valid until the EAU operation requested is complete and client's callback is called. This pointer will be passed back to the client in callback to return the result.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − EAU operation start successfully

◊ IX_CRYPTO_ACC_STATUS_FAIL − EAU operation request failed.

◊ IX_CRYPTO_ACC_STATUS_OUT_OF_RANGE_ERR − Invalid operand length / range

◊ IX_CRYPTO_ACC_STATUS_NULL_PTR_ERR − NULL pointer error, input or output data pointers are NULL.

◊ IX_CRYPTO_ACC_STATUS_OPERATION_NOT_SUPPORTED − invalid operation requested.

◊ IX_CRYPTO_ACC_STATUS_RETRY − coprocessor is busy and the request is rejected, resend the request after a random interval.
◊ Reentrant : yes
◊ ISR Callable : yes

```
IxCryptoAccStatus ( IxCryptoAccAuthAlgo hashAlgo,
ixCryptoAccPkeHashPerform
                    UINT8 * pHashData,
                    UINT32 hashDataLen,
                    IxCryptoAccPkeHashPerformCompleteCallback hashPerformCallbackFn,
                    UINT8 * pDigest
                  )
```

This function is used to hash the input data and return the digest generated via callback function supplied by client. L bytes size of digest will be generated (L = 20 for SHA1). The hash data is padded inside the function so that its length (in bits) is congruent to 448, modulo 512 (please refer to RFC1321). The length of data in bytes to be hashed must be greater than 0.

Note:

1. This API will use the hash engine on PKE (Public Key Exchange) bridge and not NPE. 2. The hash algorithm supported is SHA−1 ONLY. 3. This API is designed to offload the hashing operation needed for PKE. 4. This API uses more Intel XScale(R) Processor bandwidth compared to NPE hashing API **ixCryptoAccHashPerform** as Intel XScale(R) Processor will act as the master to the SHA engine on PKE Crypto Engine to kick off hashing operation. This involves memory accessing (read/write) from Intel XScale(R) Processor to SHA engine. 5. Client can decide to either hashing data by using SHA engine on PKE Crypto Engine or NPE hashing co−processor depend on their needs and interface. The input data is a flat buffer for this API and is IX_OSAL_MBUF for **ixCryptoAccHashPerform** on NPE. 6. ixCryptoAccPkeHashPerform will ensure that the hashing request is processed immediately if PKE SHA engine is not busy. However, hashing request to NPE will be enqueued into AQM hardware queues. It might take longer time to have hashing done if the queue is full with other bulk encryption / authentication requests. 7. Only one request can be processed by the SHA engine on PKE Crypto Engine at one time, queuing system was not implemented. 8. This API is only targeted for network processors Intel (R) IXP4XX Product Line of Network Processors which support it.

Parameters:

hashAlgo	**IxCryptoAccAuthAlgo** [in] − hashing algorithm to be used. Only SHA−1 is supported now. This parameter was introduced for future expansion since only 1 hash algorithm is supported now. Other hashing algo will be rejected.
**pHashData*	UINT8 [in] − a pointer to a flat buffer that contains the data to be hashed. This buffer is allocated by the client and the buffer pointer must not be NULL. This buffer should hold the data to be hashed until Intel XScale(R) Processor completed the hashing operation and the client's callback is called. The digest is returned separately through the client's callback.
hashDataLen	UINT32 [in] − hash data size in bytes
hashPerformCallbackFn	**IxCryptoAccPkeHashPerformCompleteCallback** [in] − callback function pointer to be called when the hashing operation is completed. This cannot be NULL.
**pDigest*	UINT8 [out] − a pointer to a flat buffer which will be used to hold the digest computated. This buffer is allocated by the client and the buffer pointer must not be NULL.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Hashing operation start successfully

◊ IX_CRYPTO_ACC_STATUS_FAIL − Hashing operation request failed

◊ IX_CRYPTO_ACC_STATUS_AUTH_ALGO_NOT_SUPPORTED − Invalid hash algorithm requested by the client.

◊ IX_CRYPTO_ACC_STATUS_OUT_OF_RANGE_ERR − Invalid hash data length (0 == hashDataLen or 65399 < hashDataLen)

◊ IX_CRYPTO_ACC_STATUS_NULL_PTR_ERR − NULL pointer error, either pHashData, pDigest or callback function pointer is NULL.

◊ IX_CRYPTO_ACC_STATUS_RETRY − coprocessor is busy and the request is rejected, resend the request after a random interval.
◊ Reentrant : yes
◊ ISR Callable : yes

IxCryptoAccStatus ixCryptoAccPkePseudoRandomNumberGet (UINT32 *pseudoRandomNumberLen*, UINT32 * *pPseudoRandomNumber*)

This function will return the pseudo−random number generated in hardware to the client via the data pointer supplied. The length of pseudo−random number requested must be in words count (pseudoRandomNumberLen >= 1 word). If bigger pseudo−random number is requested, then this function will take longer time to complete.

Note:

1. Client may randomize the output of RNG by calling **ixCryptoAccPkeHashPerform**. Output of this API is used as seed (hash data) to ixCryptoAccPkeHashPerform API. 2. The random number generator (RNG) resided in PKE (Public Key Exchange) Crypto Engine and not NPE. 3. This is synchronous function call, the pseudo−random number is ready when this function return. 4. This API is designed to generate pseudo−random number needed for PKE. 5. Only one request can be processed at one time, queuing system was not implemented. 6. This API is only targeted for network processors Intel (R) IXP4XX Product Line of Network Processors which support it.

Parameters:

pseudoRandomNumberLen UINT32 [in] − pseudo−random number length in words

**pPseudoRandomNumber* UINT32 [out] − a pointer to a flat buffer which will be used to hold the random number generated. This buffer is allocated by the client and the buffer pointer must not be NULL.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − Pseudo random number get operation complete successfully.

◊ IX_CRYPTO_ACC_STATUS_FAIL − operation failed

◊ IX_CRYPTO_ACC_STATUS_NULL_PTR_ERR − pPseudoRandomNumber is NULL

◊ IX_CRYPTO_ACC_STATUS_OUT_OF_RANGE_ERR − pseudoRandomNumberLen <= 0

◊ IX_CRYPTO_ACC_STATUS_RETRY − coprocessor is busy and the request is rejected, resend the request after a random interval.
◊ Reentrant : yes
◊ ISR Callable : yes

void ixCryptoAccShow (void)

API for printing statistics and status.

This function is called by the client to print statistics, such as number of packets returned with operation fail, number of packets encrypted/ decrypted/authenticated while the function also print the current status of the queue, whether the queue is empty or full or current queue length.

This function also print out statistics of large number arithmetic operation counter and counter of hashing operation done on PKE Crypto Engine if PKE Crypto Engine is present.

Returns:

None

- Reentrant : yes
- ISR Callable : no

void ixCryptoAccShowWithId (UINT32 *cryptoCtxId*)

API for printing statistic and status.

This function prints all the statistics provided by ixCryptoAccShow. In addition the function will print out the contents of the Crypto Context corresponding to the CryptoCtxId supplied through the API

Parameters:

cryptoCtxId UINT32 [in] − Crypto Context ID which has been registered

Returns:

None

- Reentrant : yes
- ISR Callable : no

IxCryptoAccStatus ixCryptoAccUninit (void)

Uninitialise the Security Access component.

Note:

This function should be used to replace the ixCryptoAccCryptoServiceStop function.

Parameters:

none

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − successfully uninitialize the component.

◊ IX_CRYPTO_ACC_STATUS_FAIL − fail to uninitialize the component.
◊ Reentrant : no
◊ ISR Callable : no

```
IxCryptoAccStatus ixCryptoAccXScaleWepPerform (
    UINT32              cryptoCtxId,
    IX_OSAL_MBUF *      pSrcMbuf,
    IX_OSAL_MBUF *      pDestMbuf,
    UINT16              startOffset,
    UINT16              dataLen,
    UINT16              icvOffset,
    UINT8 *             pKey
)
```

Function to support ARC4 and WEP ICV computations on Intel XScale(R) Processor.

This function executes ARC4 and WEP ICV algorithms on Intel XScale(R) Processor. The request is not forwarded to the NPE. The operation is executed on Intel XScale(R) Processor synchronously and therefore the perform done call back function would not be called on its return. The caller can assume the ownership of the buffers when the function returns. Restrictions: This function does not support non−in−place mode of operation. When WEP ICV verification operation is requested, the computed ICV is written out at the ICV location for in−place mode of operation.The supplied ICV is overwritten as a result. This behavior is true irrespective whether it's WEP ICV verify or a combined

- ARC4 decrypt and WEP ICV request.

A valid cryptoCtxId must be obtained via **ixCryptoAccCtxRegister** API in order to proceed to perform the functionalities above.

Parameters:

cryptoCtxId UINT32 [in] − is the crypto context id to be supplied by the client. This is obtained by calling ixCryptoAccCtxRegister function.

pSrcMbuf* [in] − is a pointer to mbuf which contains data to be processed. This mbuf structure is allocated by client. Result of this request will be stored in the same mbuf and overwritten the original data if UseDifferentSrcAndDestMbufs flag in **IxCryptoAccCtx is set to FALSE (in−place operation). Otherwise, if UseDifferentSrcAndDestMbufs flag is set to TRUE, the result will be written into destination mbuf (non in−place operation) and the original data in this mbuf will remain unchanged.

**pDestMbuf* [in] − only used if UseDifferentSrcAndDestMbufs is TRUE. This is the buffer where the result is written to. This mbuf structure is allocated by client. The length of mbuf MUST be big enough to hold the result of operation. The result of operation COULD NOT span into two or more different mbufs, thus the mbuf supplied must be at least the length of expected result. The data is written back starting at startOffset in the pDestMbuf. Non−in−place operations are not supported therefore this parameter is ignored for now.

startOffset UINT16 [in] − supplied by the client to indicate the start of the payload to be decrypted/encrypted or authenticated.

dataLen

	UINT16 [in] − supplied by the client to indicate the length of the payload to be decrypted/encrypted in number of bytes.
icvOffset	UINT16 [in] − supplied by the client to indicate the start of the ICV (Integrity Check Value) used for the authentication. This ICV field should not be split across multiple mbufs in a chained mbuf.
**pKey*	UINT8 [in] − Pointer to IX_CRYPTO_ACC_ARC4_KEY_128 bytes of per packet ARC4 keys. This pointer can be NULL if the request is WEP IV gen or verify only.

Note:

This function operates synchronously and will not return status until the operation is completed.

Returns:

◊ IX_CRYPTO_ACC_STATUS_SUCCESS − The specified operation was performed successfully.If the operation type was WEP ICV verify, then this value means the verification was success as well.

◊ IX_CRYPTO_ACC_STATUS_FAIL − The request failed because there was a parameter inconsistency or an invalid valid operation was requested.

◊ IX_CRYPTO_ACC_STATUS_AUTH_FAIL− The specified operation was performed, but the WEP ICV verification failed.

◊ IX_CRYPTO_ACC_STATUS_CRYPTO_CTX_NOT_VALID − Crypto context is invalid.

Note:

Statistics collection for Intel XScale(R) Processor WEP perform will only be enabled in DEBUG mode. Interrupt locking mechanism is used in protecting the critical section for statistic collection.

- Reentrant : yes
- ISR Callable : yes

Intel (R) IXP400 Software DMA Types (IxDmaTypes)

The common set of types used in the DMA component.

Enumerations

enum **IxDmaReturnStatus** {
IX_DMA_SUCCESS,
IX_DMA_FAIL,
IX_DMA_INVALID_TRANSFER_WIDTH,
IX_DMA_INVALID_TRANSFER_LENGTH,
IX_DMA_INVALID_TRANSFER_MODE,
IX_DMA_INVALID_ADDRESS_MODE,
IX_DMA_REQUEST_FIFO_FULL
}
Dma return status definitions.

enum **IxDmaTransferMode** {
IX_DMA_COPY_CLEAR,
IX_DMA_COPY,
IX_DMA_COPY_BYTE_SWAP,
IX_DMA_COPY_REVERSE,
IX_DMA_TRANSFER_MODE_INVALID
}
Dma transfer mode definitions.

enum **IxDmaAddressingMode** {
IX_DMA_INC_SRC_INC_DST,
IX_DMA_INC_SRC_FIX_DST,
IX_DMA_FIX_SRC_INC_DST,
IX_DMA_FIX_SRC_FIX_DST,
IX_DMA_ADDRESSING_MODE_INVALID
}
Dma addressing mode definitions.

enum **IxDmaTransferWidth** {
IX_DMA_32_SRC_32_DST,
IX_DMA_32_SRC_16_DST,
IX_DMA_32_SRC_8_DST,
IX_DMA_16_SRC_32_DST,
IX_DMA_16_SRC_16_DST,
IX_DMA_16_SRC_8_DST,
IX_DMA_8_SRC_32_DST,
IX_DMA_8_SRC_16_DST,
IX_DMA_8_SRC_8_DST,
IX_DMA_8_SRC_BURST_DST,
IX_DMA_16_SRC_BURST_DST,
IX_DMA_32_SRC_BURST_DST,

IX_DMA_BURST_SRC_8_DST,
IX_DMA_BURST_SRC_16_DST,
IX_DMA_BURST_SRC_32_DST,
IX_DMA_BURST_SRC_BURST_DST,
IX_DMA_TRANSFER_WIDTH_INVALID
}
Dma transfer width definitions Fixed addresses (either source or destination) do not support burst transfer width.

enum **IxDmaNpeId** {
IX_DMA_NPEID_NPEA,
IX_DMA_NPEID_NPEB,
IX_DMA_NPEID_NPEC,
IX_DMA_NPEID_MAX
}
NpeId numbers to identify NPE A, B or C.

Detailed Description

The common set of types used in the DMA component.

Enumeration Type Documentation

enum IxDmaAddressingMode

Dma addressing mode definitions.

Note:
Fixed source address to fixed destination address addressing mode is not supported.

Enumeration values:

IX_DMA_INC_SRC_INC_DST	Incremental source address to incremental destination address.
IX_DMA_INC_SRC_FIX_DST	Incremental source address to incremental destination address.
IX_DMA_FIX_SRC_INC_DST	Incremental source address to incremental destination address.
IX_DMA_FIX_SRC_FIX_DST	Incremental source address to incremental destination address.
IX_DMA_ADDRESSING_MODE_INVALID	Invalid Addressing Mode.

Definition at line **65** of file **IxDmaAcc.h**.

enum IxDmaNpeId

NpeId numbers to identify NPE A, B or C.

Warning:

This enum is deprecated and will be removed in future IXP400 Software release. Please use IxNpeDlNpeId enum instead.

Enumeration values:

IX_DMA_NPEID_NPEA	Identifies NPE A.
IX_DMA_NPEID_NPEB	Identifies NPE B.
IX_DMA_NPEID_NPEC	Identifies NPE C.
IX_DMA_NPEID_MAX	Total Number of NPEs.

Definition at line **109** of file **IxDmaAcc.h**.

enum IxDmaReturnStatus

Dma return status definitions.

Enumeration values:

IX_DMA_SUCCESS	DMA Transfer Success.
IX_DMA_FAIL	DMA Transfer Fail.
IX_DMA_INVALID_TRANSFER_WIDTH	Invalid transfer width.
IX_DMA_INVALID_TRANSFER_LENGTH	Invalid transfer length.
IX_DMA_INVALID_TRANSFER_MODE	Invalid transfer mode.
IX_DMA_INVALID_ADDRESS_MODE	Invalid address mode.
IX_DMA_REQUEST_FIFO_FULL	DMA request queue is full.

Definition at line **33** of file **IxDmaAcc.h**.

enum IxDmaTransferMode

Dma transfer mode definitions.

Note:

Copy and byte swap, and copy and reverse modes only support multiples of word data length.

Enumeration values:

IX_DMA_COPY_CLEAR	copy and clear source
IX_DMA_COPY	copy

IX_DMA_COPY_BYTE_SWAP	copy and byte swap (endian)
IX_DMA_COPY_REVERSE	copy and reverse
IX_DMA_TRANSFER_MODE_INVALID	Invalid transfer mode.

Definition at line **50** of file **IxDmaAcc.h**.

enum IxDmaTransferWidth

Dma transfer width definitions Fixed addresses (either source or destination) do not support burst transfer width.

Enumeration values:

IX_DMA_32_SRC_32_DST	32−bit src to 32−bit dst
IX_DMA_32_SRC_16_DST	32−bit src to 16−bit dst
IX_DMA_32_SRC_8_DST	32−bit src to 8−bit dst
IX_DMA_16_SRC_32_DST	16−bit src to 32−bit dst
IX_DMA_16_SRC_16_DST	16−bit src to 16−bit dst
IX_DMA_16_SRC_8_DST	16−bit src to 8−bit dst
IX_DMA_8_SRC_32_DST	8−bit src to 32−bit dst
IX_DMA_8_SRC_16_DST	8−bit src to 16−bit dst
IX_DMA_8_SRC_8_DST	8−bit src to 8−bit dst
IX_DMA_8_SRC_BURST_DST	8−bit src to burst dst − Not supported for fixed destination address
IX_DMA_16_SRC_BURST_DST	16−bit src to burst dst − Not supported for fixed destination address
IX_DMA_32_SRC_BURST_DST	32−bit src to burst dst − Not supported for fixed destination address
IX_DMA_BURST_SRC_8_DST	burst src to 8−bit dst − Not supported for fixed source address
IX_DMA_BURST_SRC_16_DST	burst src to 16−bit dst − Not supported for fixed source address
IX_DMA_BURST_SRC_32_DST	burst src to 32−bit dst − Not supported for fixed source address
IX_DMA_BURST_SRC_BURST_DST	burst src to burst dst − Not supported for fixed source and destination address
IX_DMA_TRANSFER_WIDTH_INVALID	Invalid transfer width.

Definition at line **80** of file **IxDmaAcc.h**.

Intel (R) IXP400 Software DMA Access Driver (IxDmaAcc) API

The public API for the Intel (R) IXP400 Software IxDmaAcc component.

Defines

#define **IX_DMA_REQUEST_FULL**
DMA request queue is full This constant is a return value used to tell the user that the IxDmaAcc queue is full.

Typedefs

typedef UINT32 **IxDmaAccRequestId**
DMA Request Id type.

typedef void(* **IxDmaAccDmaCompleteCallback**)(**IxDmaReturnStatus** status)
DMA completion notification This function is called to notify a client that the DMA has been completed.

Functions

PUBLIC IX_STATUS **ixDmaAccInit** (**IxNpeDlNpeId** npeId)
Initialise the DMA Access component This function will initialise the DMA Access component internals.

PUBLIC IX_STATUS **ixDmaAccUninit** (**IxNpeDlNpeId** npeId)
Uninitialise the DMA Access component This function will uninitialise the DMA Access component internals.

PUBLIC **IxDmaReturnStatus** **ixDmaAccDmaTransfer** (**IxDmaAccDmaCompleteCallback** callback, UINT32 SourceAddr, UINT32 DestinationAddr, UINT16 TransferLength, **IxDmaTransferMode** TransferMode, **IxDmaAddressingMode** AddressingMode, **IxDmaTransferWidth** TransferWidth)
Perform DMA transfer This function will perform DMA transfer between devices within the Intel (R) IXP400 Software memory map.

PUBLIC IX_STATUS **ixDmaAccShow** (void)
Display some component information for debug purposes Show some internal operation information relating to the DMA service. At a minimum the following will show. −the number of the DMA pend (in queue).

Detailed Description

The public API for the Intel (R) IXP400 Software IxDmaAcc component.

Define Documentation

#define IX_DMA_REQUEST_FULL

DMA request queue is full This constant is a return value used to tell the user that the IxDmaAcc queue is full.

Definition at line **140** of file **IxDmaAcc.h**.

Typedef Documentation

typedef void(* IxDmaAccDmaCompleteCallback)(**IxDmaReturnStatus** status)

DMA completion notification This function is called to notify a client that the DMA has been completed.

Parameters:

status **IxDmaReturnStatus** [out] − reporting to client

Definition at line **149** of file **IxDmaAcc.h**.

typedef UINT32 IxDmaAccRequestId

DMA Request Id type.

Definition at line **130** of file **IxDmaAcc.h**.

Function Documentation

```
ixDmaAccDmaTransfer ( IxDmaAccDmaCompleteCallback callback,
                      UINT32                      SourceAddr,
                      UINT32                      DestinationAddr,
                      UINT16                      TransferLength,
                      IxDmaTransferMode           TransferMode,
                      IxDmaAddressingMode         AddressingMode,
                      IxDmaTransferWidth          TransferWidth
                    )
```

Perform DMA transfer This function will perform DMA transfer between devices within the Intel (R) IXP400 Software memory map.

Note:

The following are restrictions for IxDmaAccDmaTransfer:

- ◊ The function is non re−entrant.
- ◊ The function assumes host devices are operating in big−endian mode.
- ◊ Fixed address does not suport burst transfer width
- ◊ Fixed source address to fixed destinatiom address mode is not suported
- ◊ The incrementing source address for expansion bus will not support a burst transfer width and copy and clear mode

Parameters:

callback	**IxDmaAccDmaCompleteCallback** [in] − function pointer to be stored and called when the DMA transfer is completed. This cannot be NULL.
SourceAddr	UINT32 [in] − Starting address of DMA source. Must be a valid Intel (R) IXP400 Software memory map address.
DestinationAddr	UINT32 [in] − Starting address of DMA destination. Must be a valid Intel (R) IXP400 Software memory map address.
TransferLength	UINT16 [in] − The size of DMA data transfer. The range must be from 1−64Kbyte
TransferMode	**IxDmaTransferMode** [in] − The DMA transfer mode
AddressingMode	**IxDmaAddressingMode** [in] − The DMA addressing mode
TransferWidth	**IxDmaTransferWidth** [in] − The DMA transfer width

Returns:

- ◊ IX_DMA_SUCCESS Notification that the DMA request is succesful
- ◊ IX_DMA_FAIL IxDmaAcc not yet initialised or some internal error has occured
- ◊ IX_DMA_INVALID_TRANSFER_WIDTH Transfer width is nit valid
- ◊ IX_DMA_INVALID_TRANSFER_LENGTH Transfer length outside of valid range
- ◊ IX_DMA_INVALID_TRANSFER_MODE Transfer Mode not valid
- ◊ IX_DMA_REQUEST_FIFO_FULL IxDmaAcc request queue is full

ixDmaAccInit (**IxNpeDlNpeId** *npeId*)

Initialise the DMA Access component This function will initialise the DMA Access component internals.

Parameters:

npeId **IxNpeDlNpeId** [in] − NPE to use for Dma Transfer

Returns:

- ◊ IX_SUCCESS succesfully initialised the component
- ◊ IX_FAIL Initialisation failed for some unspecified internal reason.

ixDmaAccShow (void)

Display some component information for debug purposes Show some internal operation information relating to the DMA service. At a minimum the following will show. −the number of the DMA pend (in queue).

Parameters:

None

Returns:

◊ None

ixDmaAccUninit (**IxNpeDlNpeId** *npeId*)

Uninitialise the DMA Access component This function will uninitialise the DMA Access component internals.

Parameters:

npeId **IxNpeDlNpeId** [in] − NPE to use for Dma Transfer

Returns:

◊ IX_SUCCESS succesfully uninitialised the component

◊ IX_FAIL Unnitialisation failed for some unspecified internal reason.

Intel (R) IXP400 Soft−Error Handler Driver (ixErrHdlAcc) API

The public API for the IXP400 Soft−Error Handler Driver (ixErrHdlAcc).

Data Structures

struct **IxErrHdlAccRecoveryStatistics**
Debug Statistical Data Structure.

Defines

#define **IX_ERRHDLACC_NPEA_ERROR_MASK_BIT**
Definition of the NPE A Error Mask Bit.

#define **IX_ERRHDLACC_NPEB_ERROR_MASK_BIT**
Definition of the NPE B Error Mask Bit.

#define **IX_ERRHDLACC_NPEC_ERROR_MASK_BIT**
Definition of the NPE C Error Mask Bit.

#define **IX_ERRHDLACC_AQM_SRAM_PARITY_ERROR_MASK_BIT**
Definition of the AQM Error Mask Bit.

Typedefs

typedef void(* **IxErrHdlAccRecoveryDoneCallback**)(**IxErrHdlAccErrorEventType**)
Definition of the Recovery Done Call−back type Function pointer with IxErrHdlAccErrorEventType type as the argument.

typedef
IX_STATUS(* **IxErrHdlAccFuncHandler**)(void)
Definition of the Event handler type function pointer.

Enumerations

enum **IxErrHdlAccErrorEventType** {
IX_ERRHDLACC_NPEA_ERROR,
IX_ERRHDLACC_NPEB_ERROR,
IX_ERRHDLACC_NPEC_ERROR,
IX_ERRHDLACC_AQM_ERROR,
IX_ERRHDLACC_NO_ERROR
}

Definition of the Soft−Error Event Types.

enum **IxErrHdlAccNPEId** {
IX_ERRHDLACC_NPEA,
IX_ERRHDLACC_NPEB,
IX_ERRHDLACC_NPEC
}
Definition of the NPE ID.

enum **IxErrHdlAccEnableEvent** {
IX_ERRHDLACC_EVT_DISABLE,
IX_ERRHDLACC_EVT_ENABLE
}
Definition of the Soft−Error Event Enable/Disable.

Functions

PUBLIC
IX_STATUS **ixErrHdlAccEnableConfigGet** (UINT32 *configMask)
Gets the configuration word enable setup.

PUBLIC
IX_STATUS **ixErrHdlAccEnableConfigSet** (UINT32 configMask)
Configures the ixErrHdlAcc module's configuration, to enable or disable a maximum total of 32 Soft Error Events types. −Reentrant − no −ISR Callable − yes.

PUBLIC
IX_STATUS **ixErrHdlAccNPEReset** (**IxErrHdlAccNPEId** npeID)
Reset the NPE via the Expansion bus controller FUSE register. The API sets the FUSE Bit that resets the NPE. It waits for the NPE to reset and then, turns off the FUSE bit to re−enable the NPE.

PUBLIC
IX_STATUS **ixErrHdlAccInit** (void)
Initializes the ixErrHdlAcc component.

PUBLIC
IX_STATUS **ixErrHdlAccUnload** (void)
Unloads the ixErrHdlAccModule, destroying the semaphore object and killing all threads/tasks.

PUBLIC**ixErrHdlAccErrorHandlerGet** (**IxErrHdlAccErrorEventType** event,
IX_STATUS **IxErrHdlAccFuncHandler** *handler)
API to get the event handler API to be called within the interrupt error ISR or task level to initiate a soft−error handling of the error.

PUBLIC
IX_STATUS **ixErrHdlAccStatusSet** (UINT32 status)

Sets the current Error condition status results that indicates if there was an error in handling a soft−error.

PUBLIC
IX_STATUS **ixErrHdlAccStatusGet** (UINT32 *status)
Gets the current Error condition status results that indicates if there was an error in handling a soft−error.

PUBLIC void **ixErrHdlAccStatisticsShow** (void)
Displays the Debug/Measurement statistics recorded by the module.

PUBLIC
IX_STATUS **ixErrHdlAccStatisticsGet** (**IxErrHdlAccRecoveryStatistics** *statistics)
API to retrieve the statistics recorded by the module.

PUBLIC void **ixErrHdlAccStatisticsClear** (void)
Resets and Clears the Debug/Measurement statistics recorded by the module.

PUBLIC
IX_STATUS **ixErrHdlAccCallbackRegister** (**IxErrHdlAccRecoveryDoneCallback** callback)
API to register a Completion done that is called after every completion of an error handling.

Detailed Description

The public API for the IXP400 Soft−Error Handler Driver (ixErrHdlAcc).

Define Documentation

```
#define IX_ERRHDLACC_AQM_SRAM_PARITY_ERROR_MASK_BIT
```

Definition of the AQM Error Mask Bit.

Definition at line **59** of file **IxErrHdlAcc.h**.

```
#define IX_ERRHDLACC_NPEA_ERROR_MASK_BIT
```

Definition of the NPE A Error Mask Bit.

Definition at line **41** of file **IxErrHdlAcc.h**.

```
#define IX_ERRHDLACC_NPEB_ERROR_MASK_BIT
```

Definition of the NPE B Error Mask Bit.

Definition at line **47** of file **IxErrHdlAcc.h**.

#define IX_ERRHDLACC_NPEC_ERROR_MASK_BIT

Definition of the NPE C Error Mask Bit.

Definition at line **53** of file **IxErrHdlAcc.h**.

Typedef Documentation

IxErrHdlAccFuncHandler

Definition of the Event handler type function pointer.

Definition at line **96** of file **IxErrHdlAcc.h**.

IxErrHdlAccRecoveryDoneCallback

Definition of the Recovery Done Call−back type Function pointer with IxErrHdlAccErrorEventType type as the argument.

Definition at line **89** of file **IxErrHdlAcc.h**.

Enumeration Type Documentation

enum IxErrHdlAccEnableEvent

Definition of the Soft−Error Event Enable/Disable.

Definition at line **78** of file **IxErrHdlAcc.h**.

enum IxErrHdlAccErrorEventType

Definition of the Soft−Error Event Types.

Definition at line **29** of file **IxErrHdlAcc.h**.

enum IxErrHdlAccNPEId

Definition of the NPE ID.

Definition at line **67** of file **IxErrHdlAcc.h**.

Function Documentation

IX_STATUS ixErrHdlAccCallbackRegister (**IxErrHdlAccRecoveryDoneCallback** *callback*)

API to register a Completion done that is called after every completion of an error handling.

- Reentrant − no
- ISR Callable − no

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

callback IxErrHdlAccRecoveryDoneCallback [in] Call−back function to be called by the module upon completion of a recovery task. See **IxErrHdlAccRecoveryDoneCallback**

Returns:

IX_STATUS

◊ IX_SUCCESS: Operation success

◊ IX_FAIL : An uninitialized ixErrHdlAcc module or a NULL pointer provided to the 1st parameter argument of the function.

IX_STATUS ixErrHdlAccEnableConfigGet (UINT32 * *configMask*)

Gets the configuration word enable setup.

Note:

This API gets the soft−error configuration setup in the module that was set by the call to the **ixErrHdlAccEnableConfigSet** API.

- Reentrant − no
- ISR Callable − yes

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

configMask UINT32* [in] Data UINT32 pointer to store the configuration Mask whereby, any bits set or clear indicates an enable or disable of a particular Error event type.

Returns:

IX_STATUS

◊ IX_SUCCESS: Operation success
◊ IX_FAIL : An uninitialized ixErrHdlAcc module or a NULL pointer provided to the 1st parameter argument of the function.

See also:

ixErrHdlAccEnableConfigSet

IX_STATUS ixErrHdlAccEnableConfigSet (UINT32 *configMask*)

Configures the ixErrHdlAcc module's configuration, to enable or disable a maximum total of 32 Soft Error Events types. −Reentrant − no −ISR Callable − yes.

Note:

An example to enable NPE A , NPE B and NPE C Error handling.
ixErrHdlAccEnableConfigSet(IX_ERRHDLACC_NPEA_ERROR_MASK_BIT
|IX_ERRHDLACC_NPEB_ERROR_MASK_BIT
|IX_ERRHDLACC_NPEC_ERROR_MASK_BIT);

Precondition:

ixErrHdlAcc must be initialized by calling the ixErrHdlAccInit prior to using this API.

Parameters:

configMask UINT32 [in] − a 32 bit Configuration word where a bit N set will enable the event N a bit N clear will disable the event N N=0, 1, 2, ...31.

Returns:

IX_STATUS
◊ IX_SUCCESS: Operation success
◊ IX_FAIL : A fail if the module is not initialized.

See also:

ixErrHdlAccEnableConfigGet

IX_STATUS ixErrHdlAccErrorHandlerGet (**IxErrHdlAccErrorEventType** *event*,
IxErrHdlAccFuncHandler * *handler*
)

API to get the event handler API to be called within the interrupt error ISR or task level to initiate a soft−error handling of the error.

Note:

For an NPE Error, the event handlers can be called to reset the NPE irespective of whether there is an NPE parity error or an AHB error. This API can be used to restore the NPE in a need to need basis.

- Reentrant − no
- ISR Callable − yes

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

event IxErrHdlAccErrorEventType [in] An Event ID defined as: **IxErrHdlAccErrorEventType**

handler IxErrHdlAccFuncHandler* [in] A pointer to a Event Error handler defined as **IxErrHdlAccFuncHandler**

Returns:

IX_STATUS

- ◊ IX_SUCCESS: Operation success
- ◊ IX_FAIL : An uninitialized ixErrHdlAcc module or a NULL pointer provided to the 2nd parameter argument of the function or an invalid Event ID (1st Parameter)

IX_STATUS ixErrHdlAccInit (void)

Initializes the ixErrHdlAcc component.

- Reentrant − no
- ISR Callable − no

Precondition:

The ixErrHdlAcc module should not already been loaded or initialized.

Returns:

IX_STATUS

- ◊ IX_SUCCESS: Module has initialized or already initialized.
- ◊ IX_FAIL : Failure to create thread, create a semaphore object & initializing of the module which is already initialized.

See also:

ixErrHdlAccUnload

IX_STATUS ixErrHdlAccNPEReset (**IxErrHdlAccNPEId** *npeID*)

Reset the NPE via the Expansion bus controller FUSE register. The API sets the FUSE Bit that resets the NPE. It waits for the NPE to reset and then, turns off the FUSE bit to re−enable the NPE.

- Reentrant − yes
- ISR Callable − yes

Precondition:

npeID must be a valid NPE ID where, 0 − NPEA, 1 − NPEB & 2 − NPEC

Parameters:

npeID IxErrHdlAccNPEId [in] − NPE ID for NPEA, NPEB and NPEC. See **IxErrHdlAccNPEId**

Returns:

IX_STATUS

- ◊ IX_SUCCESS: Operation success
- ◊ IX_FAIL : Invalid NPEID, Failure to Reset the NPE due to time−out while waiting for the NPE to reset and for the NPE FUSE Bit to turn off.

---

void ixErrHdlAccStatisticsClear (void)

Resets and Clears the Debug/Measurement statistics recorded by the module.

- Reentrant − no
- ISR Callable − no

Returns:

NONE

See also:

ixErrHdlAccStatisticsGet

ixErrHdlAccStatisticsShow

IX_STATUS ixErrHdlAccStatisticsGet (**IxErrHdlAccRecoveryStatistics** * *statistics*)

API to retrieve the statistics recorded by the module.

Note:

See **IxErrHdlAccRecoveryStatistics** for details

- Reentrant − no
- ISR Callable − no

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

statistics **IxErrHdlAccRecoveryStatistics** [in]

Returns:

IX_STATUS

- ◊ IX_SUCCESS
- ◊ IX_FAIL : An uninitialized ixErrHdlAcc module or a NULL pointer provided to the 1st parameter argument of the function.

See also:

ixErrHdlAccStatisticsClear

ixErrHdlAccStatisticsShow

---

void ixErrHdlAccStatisticsShow (void)

Displays the Debug/Measurement statistics recorded by the module.

- Reentrant − no
- ISR Callable − no

Returns:

NONE

See also:

ixErrHdlAccStatisticsClear

ixErrHdlAccStatisticsGet

IX_STATUS ixErrHdlAccStatusGet (UINT32 * *status*)

Gets the current Error condition status results that indicates if there was an error in handling a soft−error.

Note:

Usage guide: This API can be used to get the Event Error WORD condition where, Any bit N set in the status WORD (32bit) shall indicate an error. Currently only bit N = 0, 1, 2 & 3 is valid.

Usage Example:

This example gets the current status and checks for NPE A Error.
ixErrHdlAccStatusGet(&status);
if(status&IX_ERRHDLACC_NPEA_ERROR_MASK_BIT)
{< NPE A Error> }

- Reentrant − no
- ISR Callable − yes

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

status UINT32 [in] Pointer to a UINT32 (WORD) to write the current Event Status condition WORD

Returns:

IX_STATUS

◊ IX_SUCCESS: Operation success
◊ IX_FAIL : An uninitialized ixErrHdlAcc module or a NULL pointer provided to the 1st parameter.

See also:

ixErrHdlAccStatusSet

IX_STATUS ixErrHdlAccStatusSet (UINT32 *status*)

Sets the current Error condition status results that indicates if there was an error in handling a soft−error.

Warning:

This function must be used only in conditions whereby, the client performs it's own error handling in addition to the ones provided by the ixErrHdlAcc component which may result in recovery failure. This is particularly applicable for customized NPE firmware done by customers. The API then must be called to indicate an error in the hardware component impacted.

Note:

Usage guide: This API can be used to set the Event Error WORD condition where, any bit N set in the status WORD (32bit) shall indicate an error. Currently only bit N = 0, 1, 2 & 3 is valid.
An Example,
This enables NPE A and NPE C Error handling.
ixErrHdlAccStatusSet(IX_ERRHDLACC_NPEA_ERROR_MASK_BIT
|IX_ERRHDLACC_NPEC_ERROR_MASK_BIT);

- Reentrant − no
- ISR Callable − yes

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Parameters:

status UINT32 [in] UINT32 (WORD) to write the current Event Status condition WORD

Returns:

IX_STATUS
◊ IX_SUCCESS: Operation success
◊ IX_FAIL : An uninitialized ixErrHdlAcc module

See also:

ixErrHdlAccStatusGet

IX_STATUS ixErrHdlAccUnload (void)

Unloads the ixErrHdlAccModule, destroying the semaphore object and killing all threads/tasks.

- Reentrant − no
- ISR Callable − no

Precondition:

The ixErrHdlAcc module should have already been intialized prior to calling this API.

Returns:

IX_STATUS

- ◊ IX_SUCCESS : Module has unloaded or already unloaded.
- ◊ IX_FAIL : Failure to terminate thread, destroy a semaphore objects and to de−allocate any allocated memory.

See also:

ixErrHdlAccInit

Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API

ethAcc is a library that does provides access to the internal Intel IXP4XX Product Line of Network Processors 10/100Bt Ethernet MACs.

Data Structures

struct **IxEthAccMacAddr**
Definition of the IEEE 802.3 Ethernet MAC address structure.

struct **IxEthAccNe**
Definition of service−specific informations.

struct **IxEthAccNe**
Definition of service−specific informations.

struct **IxEthAccNe**
Definition of service−specific informations.

struct **IxEthAccPortInfo**
a local copy of port information.

struct **IxEthAccPortInfo**
a local copy of port information.

struct **IxEthEthObjStats**
This struct defines the statistics returned by this component.

Defines

#define **IX_ETH_ACC_NUMBER_OF_PORTS**
Definition of the number of ports available in a specific Intel IXP4XX product line This does not reflect the maximum value of Ethernet port ID This macro will be deprecated in future release.

#define **IX_ETHACC_NUMBER_OF_PORTS**
Definition of the number of ports available in a specific Intel IXP4XX product line The value is only determinable during run−time This does not reflect the maximum value of Ethernet port ID.

#define **ixEthAccUnload**

#define **IX_IEEE803_MAC_ADDRESS_SIZE**
Definition of the size of the MAC address.

#define **IX_ETH_ACC_NUM_TX_PRIORITIES**

Definition of the number of transmit priorities.

#define **IX_ETHACC_NE_PORT_UNKNOWN**
Contents of the field IX_ETHACC_NE_DESTPORTID when no destination port can be found by the NPE for this frame.

#define **IX_ETHACC_NE_PADLENGTH**(mBufPtr)
The location of the number of padding bytes in the Mbuf header.

#define **IX_ETHACC_NE_DESTMAC**(mBufPtr)
The location of the destination MAC address in the Mbuf header.

#define **IX_ETHACC_NE_SOURCEMAC**(mBufPtr)
The location of the source MAC address in the Mbuf header.

#define **IX_ETHACC_NE_VLANTCI**(mBufPtr)
The VLAN Tag Control Information associated with this frame.

#define **IX_ETHACC_NE_SOURCEPORTID**(mBufPtr)
The port where this frame came from.

#define **IX_ETHACC_NE_DESTPORTID**(mBufPtr)
The destination port where this frame should be sent.

#define **IX_ETHACC_NE_QOS**(mBufPtr)
QualityOfService class (QoS) for this received frame.

#define **IX_ETHACC_NE_FLAGS**(mBufPtr)
Bit Mask of the different flags associated with a frame.

#define **IX_ETHACC_NE_BCASTMASK**
This mask defines if a received frame is a broadcast frame.

#define **IX_ETHACC_NE_MCASTMASK**
This mask defines if a received frame is a multicast frame.

#define **IX_ETHACC_NE_IPMASK**
This mask defines if a received frame is a IP frame.

#define **IX_ETHACC_NE_IPV6MASK**
This mask defines if a received frame is a IP frame.

#define **IX_ETHACC_NE_LINKMASK**
This mask is the link layer protocol indicator.

#define **IX_ETHACC_NE_STMASK**
This mask defines if a received frame is a Spanning Tree frame.

#define **IX_ETHACC_NE_FILTERMASK**
This bit indicates whether a frame has been filtered by the Rx service.

#define **IX_ETHACC_NE_PORTOVERMASK**
This mask defines the rule to transmit a frame.

#define **IX_ETHACC_NE_TAGMODEMASK**
This mask defines the tagging rules to apply to a transmit frame.

#define **IX_ETHACC_NE_TAGOVERMASK**
This mask defines the rule to transmit a frame.

#define **IX_ETHACC_NE_LOCALMACMASK**
This mask defines the rule to receive a frame to local mac.

#define **IX_ETHACC_NE_VLANMASK**
This mask defines if a received frame is VLAN tagged.

#define **IX_ETHACC_NE_VLANENABLEMASK**
This mask defines if a frame is a VLAN frame or not.

#define **IX_ETHACC_NE_NEWSRCMASK**
This mask defines if a received frame has been learned.

#define **IX_ETHACC_RX_MBUF_MIN_SIZE**
This defines the recommanded minimum size of MBUF's submitted to the frame receive service.

#define **IXP400_ETH_ACC_MII_MAX_ADDR**
This defines the highest MII address of any attached PHYs.

#define **IXP425_ETH_ACC_MII_MAX_ADDR**
This defines the highest MII address of any attached PHYs.

#define **ixEthRxPriorityPoll**
Following definition providing backward compatibility to ***ixEthRxPriorityPoll()*** *function that will be deprecated in future release.*

#define **ixEthRxMultiBufferPriorityPoll**
Following definition providing backward compatibility to ***ixEthRxMultiBufferPriorityPoll()*** *function that will be deprecated in future release.*

#define **IxEthAccTxSchedulerDiscipline**
Deprecated definition for the port transmit scheduling discipline.

#define **ixEthAccMiiPhyScan**(phyPresent)
: deprecated API entry point. This definition ensures backward compatibility

#define **ixEthAccMiiPhyConfig**(phyAddr, speed100, fullDuplex, autonegotiate)
: deprecated API entry point. This definition ensures backward compatibility

#define **ixEthAccMiiPhyReset**(phyAddr)
: deprecated API entry point. This definition ensures backward compatibility

#define **ixEthAccMiiLinkStatus**(phyAddr, linkUp, speed100, fullDuplex, autoneg)
: deprecated API entry point. This definition ensures backward compatibility

#define **ixEthAccMiiShow**(phyAddr)
: deprecated API entry point. This definition ensures backward compatibility

Typedefs

typedef enum IxEthNpePortId **IxEthAccPortId**
Definition of the Intel IXP400 Software Mac Ethernet device.

typedef void(* **IxEthAccPortTxDoneCallback**)(UINT32 callbackTag, IX_OSAL_MBUF *buffer)
Function prototype for Ethernet Tx Buffer Done callback. Registered via ixEthAccTxBufferDoneCallbackRegister.

typedef void(* **IxEthAccPortRxCallback**)(UINT32 callbackTag, IX_OSAL_MBUF *buffer, UINT32 reserved)
Function prototype for Ethernet Frame Rx callback. Registered via ixEthAccPortRxCallbackRegister.

typedef void(* **IxEthAccPortMultiBufferRxCallback**)(UINT32 callbackTag, IX_OSAL_MBUF **buffer)
Function prototype for Ethernet Frame Rx callback. Registered via ixEthAccPortMultiBufferRxCallbackRegister.

Enumerations

enum **IxEthAccStatus** {
IX_ETH_ACC_SUCCESS,
IX_ETH_ACC_FAIL,
IX_ETH_ACC_INVALID_PORT,
IX_ETH_ACC_PORT_UNINITIALIZED,
IX_ETH_ACC_MAC_UNINITIALIZED,
IX_ETH_ACC_INVALID_ARG,
IX_ETH_TX_Q_FULL,
IX_ETH_ACC_NO_SUCH_ADDR
}
Definition of the Ethernet Access status.

enum **IxEthAccTxPriority** {
IX_ETH_ACC_TX_PRIORITY_0,
IX_ETH_ACC_TX_PRIORITY_1,
IX_ETH_ACC_TX_PRIORITY_2,
IX_ETH_ACC_TX_PRIORITY_3,
IX_ETH_ACC_TX_PRIORITY_4,
IX_ETH_ACC_TX_PRIORITY_5,
IX_ETH_ACC_TX_PRIORITY_6,

IX_ETH_ACC_TX_PRIORITY_7,
IX_ETH_ACC_TX_DEFAULT_PRIORITY
}
Definition of the relative priority used to transmit a frame.

enum **IxEthAccRxFrameType** {
IX_ETHACC_RX_LLCTYPE,
IX_ETHACC_RX_ETHTYPE,
IX_ETHACC_RX_STATYPE,
IX_ETHACC_RX_APTYPE
}
Identify the type of a frame.

enum **IxEthAccDuplexMode** {
IX_ETH_ACC_FULL_DUPLEX,
IX_ETH_ACC_HALF_DUPLEX
}
Definition to provision the duplex mode of the MAC.

enum **IxEthAccSchedulerDiscipline** {
FIFO_NO_PRIORITY,
FIFO_PRIORITY
}
Definition for the port scheduling discipline.

Functions

PUBLIC
IxEthAccStatus ixEthAccInit (void)
Initializes the Intel (R) IXP400 Software Ethernet Access Service.

PUBLIC
IxEthAccStatus ixEthAccUninit (void)
Un−Initializes the Intel (R) IXP400 Software Ethernet Access Service.

PUBLIC
IxEthAccStatus ixEthAccPortInit (**IxEthAccPortId** portId)
Initializes an NPE/Ethernet MAC Port.

PUBLIC
IxEthAccStatus ixEthAccMiiPortIdPhyAddrSet (**IxEthAccPortId** portId, UINT32 phyAddr)
PUBLIC **ixEthAccPortTxFrameSubmit** (**IxEthAccPortId** portId, IX_OSAL_MBUF
IxEthAccStatus *buffer, **IxEthAccTxPriority** priority)
PUBLIC**ixEthAccPortTxDoneCallbackRegister** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccPortTxDoneCallback txCallbackFn, UINT32 callbackTag)
Register a callback function to allow the transmitted buffers to return to the user.

UINT32 **ixEthAccRxPriorityPoll** (UINT32 reserved, UINT32 maxQEntries)
UINT32 **ixEthAccRxMultiBufferPriorityPoll** (UINT32 reserved, UINT32 maxQEntries)

PUBLIC **ixEthAccPortRxCallbackRegister** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccPortRxCallback rxCallbackFn, UINT32 callbackTag)
Register a callback function to allow the reception of frames.

PUBLIC **ixEthAccPortMultiBufferRxCallbackRegister** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccPortMultiBufferRxCallback rxCallbackFn, UINT32 callbackTag)
Register a callback function to allow the reception of frames.

PUBLIC **ixEthAccPortRxFreeReplenish** (**IxEthAccPortId** portId, IX_OSAL_MBUF
IxEthAccStatus *buffer)
This function provides buffers for the Ethernet receive path.

PUBLIC
IxEthAccStatus ixEthAccPortEnable (**IxEthAccPortId** portId)
This enables an Ethernet port for both Tx and Rx.

PUBLIC
IxEthAccStatus ixEthAccPortDisable (**IxEthAccPortId** portId)
This disables an Ethernet port for both Tx and Rx.

PUBLIC
IxEthAccStatus ixEthAccPortEnabledQuery (**IxEthAccPortId** portId, BOOL *enabled)
Get the enabled state of a port.

PUBLIC
IxEthAccStatus ixEthAccPortPromiscuousModeClear (**IxEthAccPortId** portId)
Put the Ethernet MAC device in non−promiscuous mode.

PUBLIC
IxEthAccStatus ixEthAccPortPromiscuousModeSet (**IxEthAccPortId** portId)
Put the MAC device in promiscuous mode.

PUBLIC **ixEthAccPortUnicastMacAddressSet** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccMacAddr *macAddr)
Configure unicast MAC address for a particular port.

PUBLIC **ixEthAccPortUnicastMacAddressGet** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccMacAddr *macAddr)
PUBLIC **ixEthAccPortMulticastAddressJoin** (**IxEthAccPortId** portId, **IxEthAccMacAddr**
IxEthAccStatus *macAddr)
PUBLIC
IxEthAccStatus ixEthAccPortMulticastAddressJoinAll (**IxEthAccPortId** portId)
Filter all frames with multicast dest.

PUBLIC **ixEthAccPortMulticastAddressLeave** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccMacAddr *macAddr)
PUBLIC
IxEthAccStatus ixEthAccPortMulticastAddressLeaveAll (**IxEthAccPortId** portId)
This function unconfigures the multicast filtering settings.

ixEthAccPortUnicastAddressShow (**IxEthAccPortId** portId)

PUBLIC
IxEthAccStatus

Displays unicast MAC address.

PUBLIC void **ixEthAccPortMulticastAddressShow** (**IxEthAccPortId** portId)

Displays multicast MAC address.

PUBLIC**ixEthAccPortDuplexModeSet** (**IxEthAccPortId** portId, **IxEthAccDuplexMode**
IxEthAccStatus mode)

Set the duplex mode for the MAC.

PUBLIC**ixEthAccPortDuplexModeGet** (**IxEthAccPortId** portId, **IxEthAccDuplexMode**
IxEthAccStatus *mode)

Get the duplex mode for the MAC.

PUBLIC
IxEthAccStatus ixEthAccPortTxFrameAppendPaddingEnable (**IxEthAccPortId** portId)

Enable padding bytes to be appended to runt frames submitted to this port.

PUBLIC
IxEthAccStatus ixEthAccPortTxFrameAppendPaddingDisable (**IxEthAccPortId** portId)

Disable padding bytes to be appended to runt frames submitted to this port.

PUBLIC
IxEthAccStatus ixEthAccPortTxFrameAppendFCSEnable (**IxEthAccPortId** portId)

Enable the appending of Ethernet FCS to all frames submitted to this port.

PUBLIC
IxEthAccStatus ixEthAccPortTxFrameAppendFCSDisable (**IxEthAccPortId** portId)

Disable the appending of Ethernet FCS to all frames submitted to this port.

PUBLIC
IxEthAccStatus ixEthAccPortRxFrameAppendFCSEnable (**IxEthAccPortId** portId)

Forward frames with FCS included in the receive buffer.

PUBLIC
IxEthAccStatus ixEthAccPortRxFrameAppendFCSDisable (**IxEthAccPortId** portId)

Do not forward the FCS portion of the received Ethernet frame to the user. The FCS is striped from the receive buffer. The received frame length does not include the FCS size (4 bytes). Frame FCS validity checks are still carried out on all received frames.

PUBLIC**ixEthAccTxSchedulingDisciplineSet** (**IxEthAccPortId** portId,
IxEthAccStatus IxEthAccSchedulerDiscipline sched)

Set the port scheduling to one of IxEthAccSchedulerDiscipline.

PUBLIC
IxEthAccStatus ixEthAccRxSchedulingDisciplineSet (**IxEthAccSchedulerDiscipline** sched)

Set the Rx scheduling to one of IxEthAccSchedulerDiscipline.

ixEthAccPortNpeLoopbackEnable (**IxEthAccPortId** portId)

PUBLIC
IxEthAccStatus
PUBLIC
IxEthAccStatus ixEthAccPortNpeLoopbackDisable (**IxEthAccPortId** portId)
Disable NPE loopback.

PUBLIC
IxEthAccStatus ixEthAccPortTxEnable (**IxEthAccPortId** portId)
Enable Tx on the port.

PUBLIC
IxEthAccStatus ixEthAccPortTxDisable (**IxEthAccPortId** portId)
Disable Tx on the port.

PUBLIC
IxEthAccStatus ixEthAccPortRxEnable (**IxEthAccPortId** portId)
Enable Rx on the port.

PUBLIC
IxEthAccStatus ixEthAccPortRxDisable (**IxEthAccPortId** portId)
Disable Rx on the port.

PUBLIC
IxEthAccStatus ixEthAccPortMacReset (**IxEthAccPortId** portId)
Reset MAC core on the port.

PUBLIC void **ixEthAccQMgrRxNotificationDisable** (void)
Disable queue interrupts for all Rx queues.

PUBLIC void **ixEthAccQMgrRxNotificationEnable** (void)
Enable queue interrupts for all Rx queues.

PUBLIC void **ixEthAccQMgrRxQEntryGet** (UINT32 *numEntries)
Get the total number of receive buffers in all rx queues.

PUBLIC
IxEthAccStatus ixEthAccStopRequest (void)
Request Ethernet access layer to stop Tx and Rx Services for all the Ethernet ports.

PUBLIC
IxEthAccStatus ixEthAccStartRequest (void)
Request Ethernet access layer to restart its currently stopped Tx and Rx Services for all the Ethernet ports.

PUBLIC BOOL **ixEthAccStopDoneCheck** (void)
Checks whether the Ethernet traffic service is stopped completely.

PUBLIC
IxEthAccStatus ixEthAccQMStatusUpdate (**IxEthAccPortId** portId)
To retrigger the update of queue condition (interrupt and status flag) in order to restore these states that are lost during soft−error handling. This ensures Eth−NPE

to continue its services.

PUBLIC
IxEthAccStatus ixEthAccMibIIStatsGet (**IxEthAccPortId** portId, **IxEthEthObjStats** *retStats)
Returns the statistics maintained for a port.

PUBLIC**ixEthAccMibIIStatsGetClear** (**IxEthAccPortId** portId, **IxEthEthObjStats**
IxEthAccStatus *retStats)
Returns and clears the statistics maintained for a port.

PUBLIC
IxEthAccStatus ixEthAccMibIIStatsClear (**IxEthAccPortId** portId)
Clears the statistics maintained for a port.

PUBLIC
IxEthAccStatus ixEthAccMacInit (**IxEthAccPortId** portId)
Initializes the Ethernet MAC settings.

PUBLIC
IxEthAccStatus ixEthAccMacUninit (**IxEthAccPortId** portId)
Un−Initializes the Ethernet MAC settings.

PUBLIC
IxEthAccStatus ixEthAccMacStateRestore (**IxEthAccPortId** portId)
This function reconfigures the MAC Registers according to the settings in use prior to the occurrence of a soft−error in the NPE.

PUBLIC void **ixEthAccStatsShow** (**IxEthAccPortId** portId)
Displays a ports statistics on the standard io console using printf.

PUBLIC
IxEthAccStatus ixEthAccMiiReadRtn (UINT8 phyAddr, UINT8 phyReg, UINT16 *value)
Reads a 16 bit value from a PHY.

PUBLIC
IxEthAccStatus ixEthAccMiiWriteRtn (UINT8 phyAddr, UINT8 phyReg, UINT16 value)
Writes a 16 bit value to a PHY.

PUBLIC
IxEthAccStatus ixEthAccMiiAccessTimeoutSet (UINT32 timeout, UINT32 retryCount)
PUBLIC
IxEthAccStatus ixEthAccMiiStatsShow (UINT32 phyAddr)
Displays detailed information on a specified PHY.

Variables

IxEthNpePortMapping
* **IxEthAccPortInfo**

Detailed Description

ethAcc is a library that does provides access to the internal Intel IXP4XX Product Line of Network Processors 10/100Bt Ethernet MACs.

Define Documentation

#define IX_ETH_ACC_NUM_TX_PRIORITIES

Definition of the number of transmit priorities.

Definition at line **121** of file **IxEthAcc.h**.

#define IX_ETH_ACC_NUMBER_OF_PORTS

Definition of the number of ports available in a specific Intel IXP4XX product line This does not reflect the maximum value of Ethernet port ID This macro will be deprecated in future release.

Definition at line **57** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_BCASTMASK

This mask defines if a received frame is a broadcast frame.

This mask defines if a received frame is a broadcast frame. The BCAST flag is set when the destination MAC address of a frame is broadcast.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **345** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_DESTMAC (mBufPtr)

The location of the destination MAC address in the Mbuf header.

Definition at line **234** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_DESTPORTID (mBufPtr)

The destination port where this frame should be sent.

The destination port where this frame should be sent.

- In the transmit direction, this field contains the destination port and is ignored unless *IX_ETHACC_NE_FLAG_DST* is set.

- In the receive direction, this field contains the port where the destination MAC addresses has been learned. If the destination MAC address is unknown, then this value is set to the reserved value *IX_ETHACC_NE_PORT_UNKNOWN*

Definition at line **293** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_FILTERMASK

This bit indicates whether a frame has been filtered by the Rx service.

This mask applies to *IX_ETHACC_NE_FLAGS*. Certain frames, which should normally be fully filtered by the NPE to due the destination MAC address being on the same segment as the Rx port are still forwarded to the Intel XScale(R) processor (although the payload is invalid) in order to learn the MAC address of the transmitting station, if this is unknown. Normally EthAcc will filter and recycle these frames internally and no frames with the FILTER bit set will be received by the client.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **452** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_FLAGS (mBufPtr)

Bit Mask of the different flags associated with a frame.

The flags are the bit−oring combination of the following different fields :

- IP flag (Rx *IX_ETHACC_NE_IPMASK*)
- Spanning Tree flag (Rx *IX_ETHACC_NE_STMASK*)
- Link layer type (Rx and Tx *IX_ETHACC_NE_LINKMASK*)
- VLAN Tagged Frame (Rx *IX_ETHACC_NE_VLANMASK*)
- New source MAC address (Rx *IX_ETHACC_NE_NEWSRCMASK*)
- Multicast flag (Rx *IX_ETHACC_NE_MCASTMASK*)
- Broadcast flag (Rx *IX_ETHACC_NE_BCASTMASK*)
- Destination port flag (Tx *IX_ETHACC_NE_PORTMASK*)
- Tag/Untag Tx frame (Tx *IX_ETHACC_NE_TAGMODEMASK*)
- Overwrite destination port (Tx *IX_ETHACC_NE_PORTOVERMASK*)
- Filtered frame (Rx *IX_ETHACC_NE_STMASK*)
- VLAN Enabled (Rx and Tx *IX_ETHACC_NE_VLANENABLEMASK*)
- Local MAC (Rx *IX_ETHACC_NE_LOCALMACMASK*)

Definition at line **329** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_IPMASK

This mask defines if a received frame is a IP frame.

This mask applies to *IX_ETHACC_NE_FLAGS* and defines if a received frame is a IPv4 frame. The IP flag is set on Rx direction, depending on the frame contents. The flag is set when the length/type field of a received frame is 0x8000.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **378** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_IPV6MASK

This mask defines if a received frame is a IP frame.

This mask applies to *IX_ETHACC_NE_FLAGS* and defines if a received frame is a IPv6 frame. The IP flag is set on Rx direction, depending on the frame contents. The flag is set when the length/type field of a received frame is 0x86DD.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **395** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_LINKMASK

This mask is the link layer protocol indicator.

This mask applies to *IX_ETHACC_NE_FLAGS*. It reflects the state of a frame as it exits an NPE on the Rx path or enters an NPE on the Tx path. Its values are as follows:

- 0x00 − IEEE802.3 − 8802 (Rx) / IEEE802.3 − 8802 (Tx)
- 0x01 − IEEE802.3 − Ethernet (Rx) / IEEE802.3 − Ethernet (Tx)
- 0x02 − IEEE802.11 AP −> STA (Rx) / IEEE802.11 STA −> AP (Tx)
- 0x03 − IEEE802.11 AP −> AP (Rx) / IEEE802.11 AP−>AP (Tx)

See also:
IX_ETHACC_NE_FLAGS

Definition at line **415** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_LOCALMACMASK

This mask defines the rule to receive a frame to local mac.

This mask defines the rule to receive a frame to local mac. When set, the default receive rules of a port are overriden. When not set, the default rules as set by **Intel (R) IXP400 Software Ethernet Database (IxEthDB) API** should apply.

See also:

IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_LOCALMACMASK

Definition at line **526** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_MCASTMASK

This mask defines if a received frame is a multicast frame.

This mask defines if a received frame is a multicast frame. The MCAST flag is set when the destination MAC address of a frame is multicast.

See also:

IX_ETHACC_NE_FLAGS

Definition at line **361** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_NEWSRCMASK

This mask defines if a received frame has been learned.

This mask defines if the source MAC address of a frame is already known. If the bit is set, the source MAC address was unknown to the NPE at the time the frame was received.

See also:

IX_ETHACC_NE_FLAGS

Definition at line **582** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_PADLENGTH (mBufPtr)

The location of the number of padding bytes in the Mbuf header.

Definition at line **224** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_PORT_UNKNOWN

Contents of the field *IX_ETHACC_NE_DESTPORTID* when no destination port can be found by the NPE for this frame.

Definition at line **214** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_PORTOVERMASK

This mask defines the rule to transmit a frame.

This mask defines the rule to transmit a frame. When set, a frame is transmitted to the destination port as set by the macro *IX_ETHACC_NE_DESTPORTID*. If not set, the destination port is searched using the destination MAC address.

Note:

This flag is meaningful only for multiport Network Engines.

See also:

IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_DESTPORTID

Definition at line **472** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_QOS (mBufPtr)

QualityOfService class (QoS) for this received frame.

Definition at line **303** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_SOURCEMAC (mBufPtr)

The location of the source MAC address in the Mbuf header.

Definition at line **244** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_SOURCEPORTID (mBufPtr)

The port where this frame came from.

The port where this frame came from. This field is set on receive with the port information. This field is ignored on Transmit path.

Definition at line **273** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_STMASK

This mask defines if a received frame is a Spanning Tree frame.

This mask applies to *IX_ETHACC_NE_FLAGS*. On rx direction, it defines if a received if frame is a Spanning Tree frame. Setting this fkag on transmit direction overrides the port settings regarding the VLAN options and

See also:

IX_ETHACC_NE_FLAGS

Definition at line **432** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_TAGMODEMASK

This mask defines the tagging rules to apply to a transmit frame.

This mask defines the tagging rules to apply to a transmit frame regardless of the default setting for a port. When used together with *IX_ETHACC_NE_TAGOVERMASK* and when set, the frame will be tagged prior to transmission. When not set, the frame will be untagged prior to transmission. This is accomplished irrespective of the Egress tagging rules, constituting a per−frame override.

See also:

IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_TAGOVERMASK

Definition at line **492** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_TAGOVERMASK

This mask defines the rule to transmit a frame.

This mask defines the rule to transmit a frame. When set, the default transmit rules of a port are overriden. When not set, the default rules as set by **Intel (R) IXP400 Software Ethernet Database (IxEthDB) API** should apply.

See also:

IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_TAGMODEMASK

Definition at line **509** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_VLANENABLEMASK

This mask defines if a frame is a VLAN frame or not.

When set, frames undergo normal VLAN processing on the Tx path (membership filtering, tagging, tag removal etc). If this flag is not set, the frame is considered to be a regular non−VLAN frame and no VLAN processing will be performed.

Note that VLAN−enabled NPE images will always set this flag in all Rx frames, and images which are not VLAN enabled will clear this flag for all received frames.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **566** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_VLANMASK

This mask defines if a received frame is VLAN tagged.

This mask defines if a received frame is VLAN tagged. When set, the Rx frame is VLAN−tagged and the tag value is available thru *IX_ETHACC_NE_VLANID*. Note that when sending frames which are already tagged this flag should be set, to avoid inserting another VLAN tag.

See also:
IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_VLANID

Definition at line **545** of file **IxEthAcc.h**.

#define IX_ETHACC_NE_VLANTCI (mBufPtr)

The VLAN Tag Control Information associated with this frame.

The VLAN Tag Control Information associated with this frame. On Rx path, this field is extracted from the packet header. On Tx path, the value of this field is inserted in the frame when the port is configured to insert or replace vlan tags in the egress frames.

See also:
IX_ETHACC_NE_FLAGS

Definition at line **261** of file **IxEthAcc.h**.

#define IX_ETHACC_NUMBER_OF_PORTS

Definition of the number of ports available in a specific Intel IXP4XX product line The value is only determinable during run−time This does not reflect the maximum value of Ethernet port ID.

Definition at line **69** of file **IxEthAcc.h**.

#define IX_ETHACC_RX_MBUF_MIN_SIZE

This defines the recommanded minimum size of MBUF's submitted to the frame receive service.

Definition at line **591** of file **IxEthAcc.h**.

```
#define IX_IEEE803_MAC_ADDRESS_SIZE
```

Definition of the size of the MAC address.

Definition at line **99** of file **IxEthAcc.h**.

```
#define ixEthAccMiiLinkStatus ( phyAddr,
                                linkUp,
                                speed100,
                                fullDuplex,
                                autoneg      )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiLinkStatus**

Note:
this feature is board specific

Definition at line **2933** of file **IxEthAcc.h**.

```
#define ixEthAccMiiPhyConfig ( phyAddr,
                               speed100,
                               fullDuplex,
                               autonegotiate  )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyConfig**

Note:
this feature is board specific

Definition at line **2903** of file **IxEthAcc.h**.

```
#define ixEthAccMiiPhyReset ( phyAddr  )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyReset**

Note:
this feature is board specific

Definition at line **2918** of file **IxEthAcc.h**.

#define ixEthAccMiiPhyScan (phyPresent)

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyScan**

Note:
this feature is board specific

Definition at line **2889** of file **IxEthAcc.h**.

#define ixEthAccMiiShow (phyAddr)

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyShow**

Note:
this feature is board specific

Definition at line **2950** of file **IxEthAcc.h**.

#define IxEthAccTxSchedulerDiscipline

Deprecated definition for the port transmit scheduling discipline.

Definition at line **1966** of file **IxEthAcc.h**.

ixEthRxMultiBufferPriorityPoll

Following definition providing backward compatibility to **ixEthRxMultiBufferPriorityPoll()** function that will be deprecated in future release.

Definition at line **1023** of file **IxEthAcc.h**.

#define ixEthRxPriorityPoll

Following definition providing backward compatibility to **ixEthRxPriorityPoll()** function that will be deprecated in future release.

Definition at line **990** of file **IxEthAcc.h**.

#define IXP400_ETH_ACC_MII_MAX_ADDR

This defines the highest MII address of any attached PHYs.

The maximum number for PHY address is 31, add on for range checking. This is maintained to provide backward compatibility. It will be deprecated in future release.

Definition at line **603** of file **IxEthAcc.h**.

#define IXP425_ETH_ACC_MII_MAX_ADDR

This defines the highest MII address of any attached PHYs.

The maximum number for PHY address is 31, add on for range checking. This is maintained to provide backward compatibility. It will be deprecated in future release.

Definition at line **615** of file **IxEthAcc.h**.

Typedef Documentation

typedef enum IxEthNpePortId IxEthAccPortId

Definition of the Intel IXP400 Software Mac Ethernet device.

Definition at line **78** of file **IxEthAcc.h**.

typedef void(* IxEthAccPortMultiBufferRxCallback)(UINT32 callbackTag, IX_OSAL_MBUF **buffer)

Function prototype for Ethernet Frame Rx callback. Registered via *ixEthAccPortMultiBufferRxCallbackRegister*.

It is the responsibility of the user function to free any MBUF's which it receives.

- Reentrant − yes , The user provided function should be reentrant.
- ISR Callable − yes , The user provided function must be callable from an ISR.
  This function dispatches many frames to the user level via the provided function. The invocation shall be made for multiple frames dequeued from the Ethernet QM queue. The user is required to free any MBUF's supplied via this callback. In addition the registered callback must free up MBUF's from the receive free queue when the port is disabled
  If called several times the latest callback shall be registered for a particular port.

Calling Context :

This callback is called in the context of the queue manager dispatch loop *ixQmgrDispatcherLoopRun* within the **Intel (R) IXP400 Software Queue Manager (IxQMgr) API** component. The calling context may be from interrupt or high priority thread. The decision is system specific.

Parameters:

callbackTag − This tag is that provided when the callback was registered for a particular MAC via *ixEthAccPortMultiBufferRxCallbackRegister*. It allows the same callback to be used for multiple MACs.

mbuf − Pointer to an array of Rx mbuf headers. Mbufs may be chained if the frame length is greater than the supplied mbuf length. The end of the array contains a zeroed entry (NULL pointer).

Returns:

void

Note:

The mbufs passed to this callback have the same structure than the buffers passed to *IxEthAccPortRxCallback* interfac.

The usage of this callback is exclusive with the usage of *ixEthAccPortRxCallbackRegister* and *IxEthAccPortRxCallback*

See also:

ixEthAccPortMultiBufferRxCallbackRegister

IxEthAccPortMultiBufferRxCallback

ixEthAccPortRxCallbackRegister

IxEthAccPortRxCallback

Definition at line **979** of file **IxEthAcc.h**.

```
typedef void(* IxEthAccPortRxCallback)(UINT32 callbackTag, IX_OSAL_MBUF *buffer, UINT32 reserved)
```

Function prototype for Ethernet Frame Rx callback. Registered via *ixEthAccPortRxCallbackRegister*.

It is the responsibility of the user function to free any MBUF's which it receives.

- Reentrant − yes , The user provided function should be reentrant.
- ISR Callable − yes , The user provided function must be callable from an ISR.

This function dispatches frames to the user level via the provided function. The invocation shall be made for each frame dequeued from the Ethernet QM queue. The user is required to free any MBUF's supplied via this callback. In addition the registered callback must free up MBUF's from the receive free queue when the port is disabled

If called several times the latest callback shall be registered for a particular port.

Calling Context :

This callback is called in the context of the queue manager dispatch loop *ixQmgrgrDispatcherLoopRun* within the **Intel (R) IXP400 Software Queue Manager (IxQMgr) API** component. The calling context may be from interrupt or high priority thread. The decision is system specific.

Parameters:

callbackTag UINT32 [in] − This tag is that provided when the callback was registered for a particular MAC via *ixEthAccPortRxCallbackRegister*. It allows the same callback to be used for multiple MACs.

mbuf [in] − Pointer to the Rx mbuf header. Mbufs may be chained if the frame length is greater than the supplied mbuf length.

reserved [in] − deprecated parameter The information is passed thru the **IxEthAccNe** header destination port ID field (

See also:

IX_ETHACC_NE_DESTPORTID). For backward compatibility,the value is equal to **IX_ETH_DB_UNKNOWN_PORT** (0xff).

Returns:

void

Note:

Buffers may not be filled up to the length supplied in ***ixEthAccPortRxFreeReplenish()***. The firmware fills them to the previous 64 bytes boundary. The user has to be aware that the length of the received mbufs may be smaller than the length of the supplied mbufs. The mbuf header contains the following modified field

- ◊ *IX_OSAL_MBUF_PKT_LEN* is set in the header of the first mbuf and indicates the total frame size
- ◊ *IX_OSAL_MBUF_MLEN* is set each mbuf header and indicates the payload length
- ◊ *IX_OSAL_MBUF_NEXT_BUFFER_IN_PKT_PTR* contains a pointer to the next mbuf, or NULL at the end of a chain.
- ◊ *IX_OSAL_MBUF_NEXT_PKT_IN_CHAIN_PTR* is modified. Its value is reset to NULL
- ◊ *IX_OSAL_MBUF_FLAGS* contains the bit 4 set for a broadcast packet and the bit 5 set for a multicast packet. Other bits are unmodified.

Definition at line **929** of file **IxEthAcc.h**.

```
typedef void(* IxEthAccPortTxDoneCallback)( UINT32 callbackTag, IX_OSAL_MBUF *buffer )
```

Function prototype for Ethernet Tx Buffer Done callback. Registered via *ixEthAccTxBufferDoneCallbackRegister*.

This function is called once the previously submitted buffer is no longer required by this service. It may be returned upon successful transmission of the frame or during the shutdown of the port prior to the transmission of a queued frame. The calling of this registered function is not a guarantee of successful transmission of the buffer.

- Reentrant − yes , The user provided function should be reentrant.
- ISR Callable − yes , The user provided function must be callable from an ISR.

Calling Context :

This callback is called in the context of the queue manager dispatch loop *ixQmgrgrDispatcherLoopRun* within the **Intel (R) IXP400 Software Queue Manager (IxQMgr) API** component. The calling context may be from interrupt or high priority thread. The decision is system specific.

Parameters:

callbackTag UINT32 [in] − This tag is that provided when the callback was registered for a particular MAC via *ixEthAccPortTxDoneCallbackRegister*. It allows the same callback to be used for multiple MACs.

mbuf [in] − Pointer to the Tx mbuf descriptor.

Returns:

void

Note:

The field IX_OSAL_MBUF_NEXT_PKT_IN_CHAIN_PTR is modified by the access layer and reset to NULL.

Definition at line **826** of file **IxEthAcc.h**.

Enumeration Type Documentation

enum IxEthAccDuplexMode

Definition to provision the duplex mode of the MAC.

Enumeration values:

IX_ETH_ACC_FULL_DUPLEX Full duplex operation of the MAC.

IX_ETH_ACC_HALF_DUPLEX Half duplex operation of the MAC.

Definition at line **166** of file **IxEthAcc.h**.

enum IxEthAccRxFrameType

Identify the type of a frame.

See also:

IX_ETHACC_NE_FLAGS

IX_ETHACC_NE_LINKMASK

Enumeration values:

IX_ETHACC_RX_LLCTYPE	802.3 − 8802, with LLC/SNAP
IX_ETHACC_RX_ETHTYPE	802.3 (Ethernet) without LLC/SNAP
IX_ETHACC_RX_STATYPE	802.11, AP <=> STA
IX_ETHACC_RX_APTYPE	802.11, AP <=> AP

Definition at line **152** of file **IxEthAcc.h**.

enum IxEthAccSchedulerDiscipline

Definition for the port scheduling discipline.

Select the port scheduling discipline on receive and transmit path

- FIFO : No Priority : In this configuration all frames are processed in the access component in the strict order in which the component received them.
- FIFO : Priority : This shall be a very simple priority mechanism. Higher prior−ity frames shall be forwarded before lower priority frames. There shall be no fairness mechanisms applied across different priorities. Higher priority frames could starve lower priority frames indefinitely.

Enumeration values:

FIFO_NO_PRIORITY	frames submitted with no priority
FIFO_PRIORITY	higher prority frames submitted before lower priority

Definition at line **1953** of file **IxEthAcc.h**.

enum IxEthAccStatus

Definition of the Ethernet Access status.

Enumeration values:

IX_ETH_ACC_SUCCESS	return success
IX_ETH_ACC_FAIL	return fail
IX_ETH_ACC_INVALID_PORT	return invalid port
IX_ETH_ACC_PORT_UNINITIALIZED	return uninitialized
IX_ETH_ACC_MAC_UNINITIALIZED	return MAC uninitialized
IX_ETH_ACC_INVALID_ARG	return invalid arg
IX_ETH_TX_Q_FULL	return tx queue is full
IX_ETH_ACC_NO_SUCH_ADDR	return no such address

Definition at line **34** of file **IxEthAcc.h**.

enum IxEthAccTxPriority

Definition of the relative priority used to transmit a frame.

Enumeration values:

IX_ETH_ACC_TX_PRIORITY_0	Lowest Priority submission.
IX_ETH_ACC_TX_PRIORITY_1	submission prority of 1 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_2	submission prority of 2 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_3	submission prority of 3 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_4	submission prority of 4 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_5	submission prority of 5 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_6	submission prority of 6 (0 is lowest)
IX_ETH_ACC_TX_PRIORITY_7	Highest priority submission.
IX_ETH_ACC_TX_DEFAULT_PRIORITY	By default send all packets with lowest priority.

Definition at line **129** of file **IxEthAcc.h**.

Function Documentation

ixEthAccInit (void)

Initializes the Intel (R) IXP400 Software Ethernet Access Service.

- Reentrant − yes
- ISR Callable − no

This should be called once per module initialization. Secord call to this function yields SUCCESS.

Precondition:

The NPE must first be downloaded with the required microcode which supports all required features.

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : Init Done successfully or already initialized
- ◊ *IX_ETH_ACC_FAIL* : Service has failed to initialize − EthDB, Dataplane, MII, MemInit, Mutex Init.

ixEthAccMacInit (**IxEthAccPortId** *portId*)

Initializes the Ethernet MAC settings.

- Reentrant − no
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

IxEthAccStatus ixEthAccMacStateRestore (**IxEthAccPortId** *portId*)

This function reconfigures the MAC Registers according to the settings in use prior to the occurrence of a soft−error in the NPE.

- Reentrant − no
- ISR Callable − no

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE.

Precondition:

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : MAC settings are reupdated
- ◊ *IX_ETH_ACC_FAIL* : MAC settings are not reupdated
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : PORT is not initialized
- ◊ *IX_ETH_ACC_MAC_UNINITIALIZED* : port MAC address is not initialized

---

ixEthAccMacUninit (**IxEthAccPortId** *portId*)

Un−Initializes the Ethernet MAC settings.

- Reentrant − no
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* :
portId is invalid.

---

ixEthAccMibIIStatsClear (**IxEthAccPortId** *portId*)

Clears the statistics maintained for a port.

- Reentrant − yes
- ISR Callable − no

Precondition:

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_FAIL* : Invalid arguments.
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccMibIIStatsGet (**IxEthAccPortId** *portId*,
IxEthEthObjStats * *retStats*
)

Returns the statistics maintained for a port.

- Reentrant − yes
- ISR Callable − no

Precondition:

Parameters:

portId **IxEthAccPortId** [in]
retStats **IxEthEthObjStats** [out]

Note:

Please note the user is responsible for cache coheriency of the retStat buffer. The data is actually populated via the NPE's. As such cache safe memory should be used in the retStats argument.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_FAIL* : Invalid arguments.
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

```
ixEthAccMibIIStatsGetClear ( IxEthAccPortId      portId,
                             IxEthEthObjStats *  retStats
                           )
```

Returns and clears the statistics maintained for a port.

- Reentrant − yes
- ISR Callable − yes

Precondition:

Parameters:

portId **IxEthAccPortId** [in]
retStats **IxEthEthObjStats** [out]

Note:

Please note the user is responsible for cache coheriency of the retStats buffer. The data is actually populated via the NPE's. As such cache safe memory should be used in the retStats argument.

Returns:

IxEthAccStatus
◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_FAIL* : invalid arguments.
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

```
ixEthAccMiiReadRtn ( UINT8      phyAddr,
                     UINT8      phyReg,
                     UINT16 *   value
                   )
```

Reads a 16 bit value from a PHY.

Reads a 16−bit word from a register of a MII−compliant PHY. Reading is performed through the MII management interface. This function returns when the read operation has successfully completed, or when a timeout has elapsed.

- Reentrant − no
- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr UINT8 [in] − the address of the Ethernet PHY (0−31)
phyReg UINT8 [in] − the number of the MII register to read (0−31)
value UINT16 [in] − the value read from the register

Returns:

IxEthAccStatus
◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_FAIL* : failed to read the register.

---

ixEthAccMiiStatsShow (UINT32 *phyAddr*)

Displays detailed information on a specified PHY.

Displays the current values of the first eigth MII registers for a PHY,

- Reentrant − no
- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr UINT32 [in] − the address of the Ethernet PHY (0−31)

Returns:

IxEthAccStatus
◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_FAIL* : invalid arguments.

```
ixEthAccMiiWriteRtn ( UINT8  phyAddr,
                      UINT8  phyReg,
                      UINT16 value
                    )
```

Writes a 16 bit value to a PHY.

Writes a 16−bit word from a register of a MII−compliant PHY. Writing is performed through the MII management interface. This function returns when the write operation has successfully completed, or when a timeout has elapsed.

- Reentrant − no
- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr UINT8 [in] − the address of the Ethernet PHY (0−31)
phyReg UINT8 [in] − the number of the MII register to write (0−31)
value UINT16 [out] − the value to write to the register

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_FAIL* : failed to write register.

---

ixEthAccPortDisable (**IxEthAccPortId** *portId*)

This disables an Ethernet port for both Tx and Rx.

Free MBufs are returned to the user via the registered callback when the port is disabled

- Reentrant − yes
- ISR Callable − no

Precondition:

The port must be enabled with *ixEthAccPortEnable*, otherwise this function has no effect

Parameters:

portId **IxEthAccPortId** [in] − Port id to act upon.

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is not initialized
- ◊ *IX_ETH_ACC_MAC_UNINITIALIZED* : port MAC address is not initialized

---

ixEthAccPortDuplexModeGet (**IxEthAccPortId** *portId*,
IxEthAccDuplexMode * *mode*
)

Get the duplex mode for the MAC.

return the duplex configuration of the IXP400 MAC.

Note:

The configuration should match that provisioned on the PHY. See *ixEthAccDuplexModeSet*

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]
mode* **IxEthAccDuplexMode [out]

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortDuplexModeSet (**IxEthAccPortId** *portId*,
IxEthAccDuplexMode *mode*
)

Set the duplex mode for the MAC.

Configure the IXP400 MAC to either full or half duplex.

Note:

The configuration should match that provisioned on the PHY.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]
mode **IxEthAccDuplexMode** [in]

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortEnable (**IxEthAccPortId** *portId*)

This enables an Ethernet port for both Tx and Rx.

- Reentrant − yes
- ISR Callable − no

Precondition:

The port must first be initialized via *ixEthAccPortInit* and the MAC address must be set using *ixEthAccUnicastMacAddressSet* before enabling it The rx and Tx Done callbacks registration via *ixEthAccPortTxDoneCallbackRegister* amd *ixEthAccPortRxCallbackRegister* has to be done before enabling the traffic.

Parameters:

portId **IxEthAccPortId** [in] − Port id to act upon.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is not initialized
◊ *IX_ETH_ACC_MAC_UNINITIALIZED* : port MAC address is not initialized

---

ixEthAccPortEnabledQuery (**IxEthAccPortId** *portId*,
BOOL * *enabled*
)

Get the enabled state of a port.

- Reentrant − yes
- ISR Callable − yes

Precondition:

The port must first be initialized via *ixEthAccPortInit*

Parameters:

portId **IxEthAccPortId** [in] − Port id to act upon.
enabled BOOL [out] − location to store the state of the port

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid

---

ixEthAccPortInit (**IxEthAccPortId** *portId*)

Initializes an NPE/Ethernet MAC Port.

The NPE/Ethernet port initialisation includes the following steps

- Initialize the NPE/Ethernet MAC hardware.
- Verify NPE downloaded and operational.
- The NPE shall be available for usage once this API returns.
- Verify that the Ethernet port is present before initializing

- Reentrant − no
- ISR Callable − no

This should be called once per mac device. The NPE/MAC shall be in disabled state after init.

Precondition:

The component must be initialized via *ixEthAccInit* The NPE must first be downloaded with the required microcode which supports all required features.

Dependant on Services: (Must be initialized before using this service may be initialized) ixNPEmh − NPE Message handling service. ixQmgr − Queue Manager component.

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*: if the Ethernet port is not present, a warning is issued.
◊ *IX_ETH_ACC_FAIL* : The NPE processor has failed to initialize.
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

IxEthAccStatus ixEthAccPortMacReset (**IxEthAccPortId** *portId*)

Reset MAC core on the port.

This function will perform a MAC core reset (NPE Ethernet coprocessor). This function is inherently unsafe and the NPE recovery is not guaranteed after this function is called. The proper manner of performing port disable and enable (which will reset the MAC as well) is ixEthAccPortEnable/ixEthAccPortDisable.

This function is the recommended usage scenario for hardware failure recovery and should never be used for throttling traffic.

- Reentrant − yes
- ISR Callable − no

Precondition:

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Parameters:

portId **IxEthAccPortId** [in] : ID of the port

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : MAC core reset
- ◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

```
ixEthAccPortMultiBufferRxCallbackRegister (  IxEthAccPortId                    portId,
                                             IxEthAccPortMultiBufferRxCallback rxCallbackFn,
                                             UINT32                            callbackTag
                                          )
```

Register a callback function to allow the reception of frames.

The registered callback function is called once a frame is received by this service. If many frames are already received, the function is called once.

If called several times the latest callback shall be registered for a particular port.

- Reentrant − yes
- ISR Callable − yes

Parameters:

portId **IxEthAccPortId** [in] − Register callback for a particular MAC device.

rxCallbackFn − *IxEthAccMultiBufferRxCallbackFn* − Function to be called when Ethernet frames are availble.

callbackTag − This tag shall be provided to the callback function.

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized
- ◊ *IX_ETH_ACC_INVALID_ARG* : An argument other than portId is invalid.

See also:

ixEthAccPortMultiBufferRxCallbackRegister

IxEthAccPortMultiBufferRxCallback

ixEthAccPortRxCallbackRegister

IxEthAccPortRxCallback

---

ixEthAccPortMulticastAddressJoinAll (**IxEthAccPortId** *portId*)

Filter all frames with multicast dest.

This function clears the MAC address table, and then sets the MAC to forward ALL multicast frames to the NPE. Specifically, it forwards all frames whose destination address has the LSB of the highest byte set (01:00:00:00:00:00). This bit is commonly referred to as the "multicast bit". Broadcast frames will still be forwarded.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** − all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** − rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** − unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** − provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** − all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** − frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortMulticastAddressLeaveAll (**IxEthAccPortId** *portId*)

This function unconfigures the multicast filtering settings.

This function first clears the MAC address table, and then sets the MAC as configured by the promiscuous mode current settings.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** − all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** − rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** − unprovision a new filtering address

- ***ixEthAccPortMulticastAddressJoin()*** − provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** − all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** − frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortMulticastAddressShow (**IxEthAccPortId** *portId*)

Displays multicast MAC address.

Displays multicast address which have been configured using *ixEthAccMulticastAddressJoin*

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

void

IxEthAccStatus ixEthAccPortNpeLoopbackDisable (**IxEthAccPortId** *portId*)

Disable NPE loopback.

This function is used to disable the NPE loopback if previously enabled using ixEthAccNpeLoopbackEnable.

This function is recommended for power−up diagnostic checks and should never be used under normal Ethernet traffic operations.

- Reentrant − yes

- ISR Callable − no

Precondition:

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Parameters:

portId **IxEthAccPortId** [in] : ID of the port

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS* : NPE loopback successfully disabled
◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortPromiscuousModeClear (**IxEthAccPortId** *portId*)

Put the Ethernet MAC device in non−promiscuous mode.

In non−promiscuous mode the MAC filters all frames other than destination MAC address which matches the following criteria:

- Unicast address provisioned via *ixEthAccUnicastMacAddressSet*
- All broadcast frames.
- Multicast addresses provisioned via *ixEthAccMulticastAddressJoin*

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** − all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** − rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** − unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** − provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** − all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** − frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant − yes
- ISR Callable − no

See also:

ixEthAccPortPromiscuousModeSet

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortPromiscuousModeSet (**IxEthAccPortId** *portId*)

Put the MAC device in promiscuous mode.

If the device is in promiscuous mode then all all received frames shall be forwared to the NPE for processing.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** − all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** − rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** − unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** − provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** − all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** − frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant − yes
- ISR Callable − no

See also:

ixEthAccPortPromiscuousModeClear

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortRxCallbackRegister (**IxEthAccPortId** *portId*,
IxEthAccPortRxCallback *rxCallbackFn*,
UINT32 *callbackTag*
)

Register a callback function to allow the reception of frames.

The registered callback function is called once a frame is received by this service.

If called several times the latest callback shall be registered for a particular port.

- Reentrant − yes
- ISR Callable − yes

Parameters:

portId	**IxEthAccPortId** [in] − Register callback for a particular MAC device.
rxCallbackFn	**IxEthAccPortRxCallback** [in] − Function to be called when Ethernet frames are availble.
callbackTag	UINT32 [in] − This tag shall be provided to the callback function.

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized
- ◊ *IX_ETH_ACC_INVALID_ARG* : An argument other than portId is invalid.

IxEthAccStatus ixEthAccPortRxDisable (**IxEthAccPortId** *portId*)

Disable Rx on the port.

This function can be used to disable Rx in the MAC core. Rx can be re−enabled, although this is not guaranteed, by performing a MAC core reset (*ixEthAccPortMacReset*) and calling ixEthAccPortRxEnable. Note that using this function is not recommended, except for shutting down Rx for emergency reasons. For proper port shutdown and re−enabling see ixEthAccPortEnable and ixEthAccPortDisable.

This function is the recommended usage scenario for emergency security shutdown and hardware failure recovery and should never be used for throttling traffic.

- Reentrant − yes
- ISR Callable − no

Precondition:

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Parameters:

portId : ID of the port

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : Rx successfully disabled
- ◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

IxEthAccStatus ixEthAccPortRxEnable (**IxEthAccPortId** *portId*)

Enable Rx on the port.

This function is the complement of ixEthAccPortRxDisable and should be used only after Rx was disabled.

This function is the recommended usage scenario for emergency security shutdown and hardware failure recovery and should never be used for throttling traffic.

- Reentrant − yes
- ISR Callable − no

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Precondition:

Parameters:

portId **IxEthAccPortId** [in] : ID of the port

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : Rx successfully enabled
- ◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortRxFrameAppendFCSDisable (**IxEthAccPortId** *portId*)

Do not forward the FCS portion of the received Ethernet frame to the user. The FCS is striped from the receive buffer. The received frame length does not include the FCS size (4 bytes). Frame FCS validity checks are still carried out on all received frames.

This is the default behavior of the component. Do not change this behaviour while the port is enabled.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortRxFrameAppendFCSEnable (**IxEthAccPortId** *portId*)

Forward frames with FCS included in the receive buffer.

The FCS is not striped from the receive buffer. The received frame length includes the FCS size (4 bytes). ie. A minimum sized Ethernet frame shall have a length of 64bytes.

Frame FCS validity checks are still carried out on all received frames.

This is not the default behavior of the access component.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortRxFreeReplenish (**IxEthAccPortId** *portId*,
IX_OSAL_MBUF * *buffer*
)

This function provides buffers for the Ethernet receive path.

This component does not have a buffer management mechanisms built in. All Rx buffers must be supplied to it via this interface.

- Reentrant − yes
- ISR Callable − yes

Parameters:

portId **IxEthAccPortId** [in] − Provide buffers only to specific Rx MAC.

buffer [in] − Provide an MBUF to the Ethernet receive mechanism. Buffers smaller than IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MIN is not allowed. Buffers smaller than IX_ETHACC_RX_MBUF_MIN_SIZE may result in poor performances and excessive buffer chaining. Buffers larger than this size may be suitable for jumbo frames. Chained packets are not supported and the field IX_OSAL_MBUF_NEXT_PKT_IN_CHAIN_PTR must be NULL.

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_FAIL* : Not able to queue the buffer in the receive service.
- ◊ *IX_ETH_ACC_FAIL* : Buffer size is less than IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MIN
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

Note:

If the buffer replenish operation fails it is the responsibility of the user to free the buffer.

Sufficient buffers must be supplied to the component to maintain receive throughput and avoid rx buffer underflow conditions. To meet this goal, It is expected that the user preload the component with a sufficent number of buffers prior to enabling the NPE Ethernet receive path. The recommended minimum number of buffers is 8.

For maximum performances, the mbuf size should be greater than the maximum frame size (Ethernet header, payload and FCS) + 64. Supplying smaller mbufs to the service results in mbuf chaining and degraded performances. The recommended size is *IX_ETHACC_RX_MBUF_MIN_SIZE*, which is enough to take care of 802.3 frames and "baby jumbo" frames without chaining, and "jumbo" frame within chaining.

Buffers may not be filled up to their length. The firware fills them up to the previous 64 bytes boundary. The user has to be aware that the length of the received mbufs may be smaller than the length of the supplied mbufs.

Warning:

This function checks the parameters if the NDEBUG flag is not defined. Turning on the argument checking (disabled by default) results in a lower EthAcc performance as this function is part of the data path.

IxEthAccStatus ixEthAccPortTxDisable (**IxEthAccPortId** *portId*)

Disable Tx on the port.

This function can be used to disable Tx in the MAC core. Tx can be re−enabled, although this is not guaranteed, by performing a MAC core reset (*ixEthAccPortMacReset*) and calling ixEthAccPortTxEnable. Note that using this function is not recommended, except for shutting down Tx for emergency reasons. For

proper port shutdown and re−enabling see ixEthAccPortEnable and ixEthAccPortDisable.

This function is the recommended usage scenario for emergency security shutdown and hardware failure recovery and should never be used for throttling traffic.

- Reentrant − yes
- ISR Callable − no

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Precondition:

Parameters:

portId : ID of the port

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS* : Tx successfully disabled
◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

```
ixEthAccPortTxDoneCallbackRegister ( IxEthAccPortId                portId,
                                     IxEthAccPortTxDoneCallback    txCallbackFn,
                                     UINT32                        callbackTag
                                   )
```

Register a callback function to allow the transmitted buffers to return to the user.

This function registers the transmit buffer done function callback for a particular port.

The registered callback function is called once the previously submitted buffer is no longer required by this service. It may be returned upon successful transmission of the frame or shutdown of port prior to submission. The calling of this registered function is not a guarantee of successful transmission of the buffer.

If called several times the latest callback shall be registered for a particular port.

- Reentrant − yes
- ISR Callable − yes

Precondition:

The port must be initialized via *ixEthAccPortInit*

Parameters:

portId **IxEthAccPortId** [in] − Register callback for a particular MAC device.

txCallbackFn **IxEthAccPortTxDoneCallback** [in] − Function to be called to return transmit buffers to the user.

callbackTag UINT32 [in] − This tag shall be provided to the callback function.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

◊ *IX_ETH_ACC_INVALID_ARG* : An argument other than portId is invalid.

IxEthAccStatus ixEthAccPortTxEnable (**IxEthAccPortId** *portId*)

Enable Tx on the port.

This function is the complement of ixEthAccPortTxDisable and should be used only after Tx was disabled. A MAC core reset is required before this function is called (see *ixEthAccPortMacReset*).

This function is the recommended usage scenario for emergency security shutdown and hardware failure recovery and should never be used for throttling traffic.

- Reentrant − yes
- ISR Callable − no

Precondition:

Note:

Calling ixEthAccPortDisable followed by ixEthAccPortEnable is guaranteed to restore correct Ethernet Tx/Rx operation.

Parameters:

portId : ID of the port

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS* : Tx successfully enabled

◊ *IX_ETH_ACC_FAIL* : Invalid port or Ethernet service not initialized

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortTxFrameAppendFCSDisable (**IxEthAccPortId** *portId*)

Disable the appending of Ethernet FCS to all frames submitted to this port.

When disabled, the Ethernet FCS is not added to the submitted frames. This is not the default behavior of the access component.

Note:

Since the FCS is not appended to the frame it is expected that the frame submitted to the component includes a valid FCS at the end of the data, although this will not be validated.

The component shall forward the frame to the Ethernet MAC WITHOUT modification.

Do not change this behaviour while the port is enabled.

Note:

Tx FCS append is not disabled while Tx padding is enabled.

- Reentrant − yes
- ISR Callable − no

See also:

ixEthAccPortTxFrameAppendPaddingEnable

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortTxFrameAppendFCSEnable (**IxEthAccPortId** *portId*)

Enable the appending of Ethernet FCS to all frames submitted to this port.

When enabled, the FCS is added to the submitted frames. This is the default behavior of the access component. Do not change this behaviour while the port is enabled.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*
◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortTxFrameAppendPaddingDisable (**IxEthAccPortId** *portId*)

Disable padding bytes to be appended to runt frames submitted to this port.

Disable padding bytes to be appended to runt frames submitted to this port. This is not the default behavior of the access component.

Warning:

Do not change this behaviour while the port is enabled.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortTxFrameAppendPaddingEnable (**IxEthAccPortId** *portId*)

Enable padding bytes to be appended to runt frames submitted to this port.

Enable up to 60 null−bytes padding bytes to be appended to runt frames submitted to this port. This is the default behavior of the access component.

Warning:

Do not change this behaviour while the port is enabled.

Note:

When Tx padding is enabled, Tx FCS generation is turned on

- Reentrant − yes
- ISR Callable − no

See also:

ixEthAccPortTxFrameAppendFCSDusable

Parameters:

portId **IxEthAccPortId** [in]

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.

◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccPortUnicastAddressShow (**IxEthAccPortId** *portId*)

Displays unicast MAC address.

Displays unicast address which is configured using *ixEthAccUnicastMacAddressSet*. This function also displays the MAC filter used to filter multicast frames.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** − all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** − rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** − unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** − provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** − all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** − frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

Returns:

void

---

ixEthAccPortUnicastMacAddressSet (**IxEthAccPortId** *portId*,
IxEthAccMacAddr * *macAddr*
)

Configure unicast MAC address for a particular port.

- Reentrant − yes
- ISR Callable − no

Parameters:

portId **IxEthAccPortId** [in] − Ethernet port id.

macAddr* **IxEthAccMacAddr [in] − Ethernet Mac address.

Returns:

IxEthAccStatus

◊ *IX_ETH_ACC_SUCCESS*

◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

void ixEthAccQMgrRxNotificationDisable (void)

Disable queue interrupts for all Rx queues.

This function will write to the QMgr interrupt enable register, clearing the bits associated with the receive queues. It will then write to the QMgr status interrupt to clear any already pending interrupts associated with the receive queues. This means that no more interrupts can occur for these queues until the conditions are satisfied: the queues are completely drained, then the interrupt i enable register bits for the queues are re−enabled using **ixEthAccQMgrRxNotificationEnable()**.

- Reentrant − no
- ISR Callable − yes

---

void ixEthAccQMgrRxNotificationEnable (void)

Enable queue interrupts for all Rx queues.

This function will write to the QMgr interrupt enable register, adding the bits associated with the receive queues. After calling this function interrupts can occur for these queues given that the queues were properly drained prior to calling the function.

- Reentrant − no
- ISR Callable − yes

IxEthAccStatus ixEthAccQMgrRxQEntryGet (UINT32 * *numEntries*)

Get the total number of receive buffers in all rx queues.

This function will queuery QMgr to get each receive queue's number of entries and store the total in the UINT32 pointed to by the numEntries parameter.

- Reentrant − yes
- ISR Callable − yes

Precondition:

Parameters:

numEntries : pointer to UINT32 where the number of total entries can be written

IxEthAccStatus ixEthAccQMStatusUpdate (**IxEthAccPortId** *portId*)

To retrigger the update of queue condition (interrupt and status flag) in order to restore these states that are lost during soft−error handling. This ensures Eth−NPE to continue its services.

- Reentrant − no
- ISR Callable − no

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE.

It is expected that IRQ is disabled when this function is called.

Parameters:

portId **IxEthAccPortId** [in] : ID of the port

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : Queue conditions are retriggered
- ◊ *IX_ETH_ACC_FAIL* : Queue conditions are not retriggered
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : PORT is not initialized

---

ixEthAccRxSchedulingDisciplineSet (**IxEthAccSchedulerDiscipline** *sched*)

Set the Rx scheduling to one of *IxEthAccSchedulerDiscipline*.

The default behavior of the component is *FIFO_NO_PRIORITY*.

- Reentrant − yes
- ISR Callable − no

Precondition:

Parameters:

sched : *IxEthAccSchedulerDiscipline*

Returns:

IxEthAccStatus

- ◊ *IX_ETH_ACC_SUCCESS* : Set appropriate discipline.
- ◊ *IX_ETH_ACC_FAIL* : Port is busy/priority scheduling not supported for A0.
- ◊ *IX_ETH_ACC_INVALID_ARG* : Invalid/unsupported discipline.

IxEthAccStatus ixEthAccStartRequ est (void)

Request Ethernet access layer to restart its currently stopped Tx and Rx Services for all the Ethernet ports.

Whenever this function is called, the internal counter is decremented by one (**ixEthAccStopRequest** increments this counter). If the counter becomes zero, this function sends an enable signal to QMgr dispatcher for it to procced in serving the queues.

The internal counter is for synchronizing multiple requests to stop and resume Ethernet services.

- Reentrant − yes
- ISR Callable − no

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE. User may use the APIs as follow:
1) Call **ixEthAccStopRequest** to stop Ethernet service.
2) Check that Ethernet service has been stoppoed with **ixEthAccStopDoneCheck**.
3) Proceed with soft−error recovery routine.
4) Call **ixEthAccStartRequest** to restart Ethernet service.

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS* : Service start request is posted successfully.
- ◊ *IX_ETH_ACC_FAIL* : EthAcc service is not initialized.

---

ixEthAccStatsShow (**IxEthAccPortId** *portId*)

Displays a ports statistics on the standard io console using printf.

- Reentrant − no
- ISR Callable − no

Precondition:

Parameters:

portId **IxEthAccPortId** [in]

Returns:

void

BOOL ixEthAccStopDoneCheck (void)

Checks whether the Ethernet traffic service is stopped completely.

- Reentrant − yes
- ISR Callable − no

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE. User may use the APIs as follow:
1) Call **ixEthAccStopRequest** to stop Ethernet service.
2) Check that Ethernet service has been stoppoed with **ixEthAccStopDoneCheck**.
3) Proceed with soft−error recovery routine.
4) Call **ixEthAccStartRequest** to restart Ethernet service.

Returns:

BOOL
- ◊ *TRUE* : Ethernet Tx & Rx services for all ports are stopped
- ◊ *FALSE* : Ethernet Tx & Rx services for all ports are not stopped

---

IxEthAccStatus ixEthAccStopRequest (void)

Request Ethernet access layer to stop Tx and Rx Services for all the Ethernet ports.

This function maybe called multiple times e.g. NPE soft−errors in different NPEs. Whenever this function is called, an internal counter is incremented by one. The internal counter keep tracks on the number of "stop Ethernet−service" requested by user.

If it is the first request, a disable signal is sent to QMgr dispatcher function. On subsequent request, this function only increments the value of the counter without disabling QMgr dispatcher.

The internal counter is for synchronizing the posibility of multiple requests to stop and resume Ethernet services.

- Reentrant − yes
- ISR Callable − yes

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE. User may use the APIs as follow:
1) Call **ixEthAccStopRequest** to stop Ethernet service.
2) Check that Ethernet service has been stoppoed with **ixEthAccStopDoneCheck**.
3) Proceed with soft−error recovery routine.
4) Call **ixEthAccStartRequest** to restart Ethernet service.

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS* : "Stop Ethernet−service" request is posted successfully.
- ◊ *IX_ETH_ACC_FAIL* : EthAcc service is not initialized.

---

ixEthAccTxSchedulingDisciplineSet (**IxEthAccPortId** *portId*,
IxEthAccSchedulerDiscipline *sched*
)

Set the port scheduling to one of *IxEthAccSchedulerDiscipline*.

The default behavior of the component is *FIFO_NO_PRIORITY*.

- Reentrant − yes
- ISR Callable − no

Precondition:

Parameters:

portId **IxEthAccPortId** [in]
sched **IxEthAccSchedulerDiscipline** [in]

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS* : Set appropriate discipline.
- ◊ *IX_ETH_ACC_INVALID_ARG* : Invalid/unsupported discipline.
- ◊ *IX_ETH_ACC_INVALID_PORT* : portId is invalid.
- ◊ *IX_ETH_ACC_PORT_UNINITIALIZED* : portId is un−initialized

---

ixEthAccUninit (void)

Un−Initializes the Intel (R) IXP400 Software Ethernet Access Service.

- Reentrant − no
- ISR Callable − no

This should be called once per module Un−initialization.

Returns:

IxEthAccStatus
- ◊ *IX_ETH_ACC_SUCCESS*
- ◊ *IX_ETH_ACC_FAIL* : Service has failed to Un−initialize.

Intel (R) IXP400 Software Ethernet Database (IxEthDB) API

ethDB is a library that does provides a MAC address database learning/filtering capability

Data Structures

struct **IX_OSAL_ATTRIBUTE_PACKED**
The IEEE 802.3 Ethernet MAC address structure.

struct **IxEthDBWiFiRecData**
The user wi−fi input parameters structure.

Defines

#define **INLINE**
#define **IX_ETH_DB_PRIVATE**
#define **IX_ETH_DB_PUBLIC**
#define **IX_IEEE803_MAC_ADDRESS_SIZE**
The size of the MAC address.

#define **IX_IEEE802_1Q_QOS_PRIORITY_COUNT**
Number of QoS priorities defined by IEEE802.1Q.

#define **IX_ETH_DB_802_1Q_VLAN_MASK**
VLAN mask.

#define **IX_ETH_DB_802_1Q_QOS_MASK**
QoS Mask.

#define **IX_ETH_DB_802_1Q_MAX_VLAN_ID**
Maximum VLAN IDs.

#define **IX_ETH_DB_SET_VLAN_ID**(vlanTag, vlanID)
returns the given 802.1Q tag with the VLAN ID field substituted with the given VLAN ID

#define **IX_ETH_DB_GET_VLAN_ID**(vlanTag)
returns the VLAN ID from the given 802.1Q tag

#define **IX_ETH_DB_GET_QOS_PRIORITY**(vlanTag)
gets the QOS priority from the given 802.1Q tag

#define **IX_ETH_DB_SET_QOS_PRIORITY**(vlanTag, priority)
sets the QOS priority to the given 802.1Q tag

#define **IX_ETH_DB_CHECK_VLAN_TAG**(vlanTag)
checks the VLAN ID of the given 802.1Q tag

#define **IX_ETH_DB_CHECK_VLAN_ID**(vlanId)
checks the VLAN ID

#define **IX_IEEE802_1Q_VLAN_TPID**
returns the VLAN TPID (0x8100)

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_COUNT_PROPERTY**
Property identifying number the supported number of traffic classes.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_0_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 0.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_1_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 1.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_2_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 2.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_3_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 3.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_4_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 4.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_5_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 5.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_6_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 6.

#define **IX_ETH_DB_QOS_TRAFFIC_CLASS_7_RX_QUEUE_PROPERTY**
Rx queue assigned to traffic class 7.

#define **IX_ETH_DB_QOS_QUEUE_CONFIGURATION_COMPLETE**
Queue configuration complete.

#define **IX_ETH_DB_WIFI_MIN_PAD_SIZE**
Minimum pad size.

#define **IX_ETH_DB_WIFI_MAX_PAD_SIZE**
Maximum pad size.

#define **IX_ETH_DB_MAINTENANCE_TIME**
*The **ixEthDBDatabaseMaintenance** must be called by the user at a frequency of IX_ETH_DB_MAINTENANCE_TIME.*

#define **IX_ETH_DB_LEARNING_ENTRY_AGE_TIME**

The define specifies the filtering database age entry time. Static entries older than IX_ETH_DB_LEARNING_ENTRY_AGE_TIME +/− IX_ETH_DB_MAINTENANCE_TIME shall be removed.

Typedefs

typedef UINT32 **IxEthDBVlanId**
VLAN ID type, valid range is 0..4094, 0 signifying no VLAN membership.

typedef UINT32 **IxEthDBVlanTag**
802.1Q VLAN tag, contains 3 bits user priority, 1 bit CFI, 12 bits VLAN ID

typedef UINT32 **IxEthDBPriority**
QoS priority/traffic class type, valid range is 0..7, 0 being the lowest.

typedef UINT8 **IxEthDBPriorityTable** [8]
Priority mapping table; 0..7 QoS priorities used to index, table contains traffic classes.

typedef UINT8 **IxEthDBVlanSet** [512]
A 4096 bit array used to map the complete VLAN ID range.

typedef UINT32 **IxEthDBProperty**
Property ID type.

typedef UINT32 **IxEthDBPortId**
Definition of an IXP400 port.

typedef UINT8 **IxEthDBPortMap** [32]
Port dependency map definition.

Enumerations

enum **IxEthDBStatus** {
IX_ETH_DB_SUCCESS,
IX_ETH_DB_FAIL,
IX_ETH_DB_INVALID_PORT,
IX_ETH_DB_PORT_UNINITIALIZED,
IX_ETH_DB_MAC_UNINITIALIZED,
IX_ETH_DB_INVALID_ARG,
IX_ETH_DB_NO_SUCH_ADDR,
IX_ETH_DB_NOMEM,
IX_ETH_DB_BUSY,
IX_ETH_DB_END,
IX_ETH_DB_INVALID_VLAN,
IX_ETH_DB_INVALID_PRIORITY,
IX_ETH_DB_NO_PERMISSION,
IX_ETH_DB_FEATURE_UNAVAILABLE,

IX_ETH_DB_INVALID_KEY,
IX_ETH_DB_INVALID_RECORD_TYPE
}
Ethernet Database API return values.

enum **IxEthDBFrameFilter** {
IX_ETH_DB_UNTAGGED_FRAMES,
IX_ETH_DB_VLAN_TAGGED_FRAMES,
IX_ETH_DB_PRIORITY_TAGGED_FRAMES,
IX_ETH_DB_ACCEPT_ALL_FRAMES
}
enum **IxEthDBTaggingAction** {
IX_ETH_DB_PASS_THROUGH,
IX_ETH_DB_ADD_TAG,
IX_ETH_DB_REMOVE_TAG,
IX_ETH_DB_ENABLE_VLAN,
IX_ETH_DB_DISABLE_VLAN
}
enum **IxEthDBFirewallMode** {
IX_ETH_DB_FIREWALL_WHITE_LIST,
IX_ETH_DB_FIREWALL_BLACK_LIST
}
enum **IxEthDBRecordType** {
IX_ETH_DB_FILTERING_RECORD,
IX_ETH_DB_FILTERING_VLAN_RECORD,
IX_ETH_DB_WIFI_RECORD,
IX_ETH_DB_FIREWALL_RECORD,
IX_ETH_DB_GATEWAY_RECORD,
IX_ETH_DB_MASK_RECORD,
IX_ETH_DB_MAX_RECORD_TYPE_INDEX,
IX_ETH_DB_NO_RECORD_TYPE,
IX_ETH_DB_ALL_FILTERING_RECORDS,
IX_ETH_DB_MASKED_FIREWALL_RECORD,
IX_ETH_DB_ALL_RECORD_TYPES
}
enum **IxEthDBFeature** {
IX_ETH_DB_LEARNING,
IX_ETH_DB_FILTERING,
IX_ETH_DB_VLAN_QOS,
IX_ETH_DB_FIREWALL,
IX_ETH_DB_SPANNING_TREE_PROTOCOL,
IX_ETH_DB_WIFI_HEADER_CONVERSION,
IX_ETH_DB_ADDRESS_MASKING
}
enum **IxEthDBPropertyType** {
IX_ETH_DB_INTEGER_PROPERTY,
IX_ETH_DB_STRING_PROPERTY,
IX_ETH_DB_MAC_ADDR_PROPERTY,
IX_ETH_DB_BOOL_PROPERTY
}
enum

IxEthDBWiFiRecordType {
IX_ETH_DB_WIFI_AP_TO_STA,
IX_ETH_DB_WIFI_AP_TO_AP,
IX_ETH_DB_WIFI_TO_ETHER,
IX_ETH_DB_WIFI_TO_LOCAL
}
The WI−FI record types enum.

enum **IxEthDBWiFiVlanTag** {
IX_ETH_DB_WIFI_VLAN_NOTAG,
IX_ETH_DB_WIFI_VLAN_TAG
}
The WI−FI VLAN tag/untag enum.

Functions

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBInit (void)
Initializes the Ethernet learning/filtering database.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBUnload (void)
Stops and prepares the EthDB component for unloading.

IX_ETH_DB_PUBLIC
void **ixEthDBPortInit** (**IxEthDBPortId** portID)
Initializes a port.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBPortEnable (**IxEthDBPortId** portID)
Enables a port.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBPortDisable (**IxEthDBPortId** portID)
Disables processing on a port.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBPortAddressSet (**IxEthDBPortId** portID, IxEthDBMacAddr *macAddr)
Sets the port MAC address.

IX_ETH_DB_PUBLIC**ixEthDBFilteringPortMaximumFrameSizeSet** (**IxEthDBPortId** portID, UINT32
IxEthDBStatus maximumFrameSize)
Set the maximum frame size supported on the given port ID.

IX_ETH_DB_PUBLIC**ixEthDBFilteringStaticEntryProvision** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBMacAddr *macAddr)
Populate the Ethernet learning/filtering database with a static MAC address.

IX_ETH_DB_PUBLIC **ixEthDBFilteringDynamicEntryProvision** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBMacAddr *macAddr)

Populate the Ethernet learning/filtering database with a dynamic MAC address.

IX_ETH_DB_PUBLIC
IxEthDBStatus **ixEthDBFilteringEntryDelete** (IxEthDBMacAddr *macAddr)

Removes a MAC address entry from the Ethernet learning/filtering database.

IX_ETH_DB_PUBLIC **ixEthDBFilteringPortSearch** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)

Search the Ethernet learning/filtering database for the given MAC address and port ID.

IX_ETH_DB_PUBLIC **ixEthDBFilteringDatabaseSearch** (**IxEthDBPortId** *portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)

Search the Ethernet learning/filtering database for a MAC address and return the port ID.

IX_ETH_DB_PUBLIC **ixEthDBFilteringPortUpdatingSearch** (**IxEthDBPortId** *portID,
IxEthDBStatus IxEthDBMacAddr *macAddr)

Search the filtering database for a MAC address, return the port ID and reset the record age.

IX_ETH_DB_PUBLIC
IxEthDBStatus **ixEthDBPortAgingDisable** (**IxEthDBPortId** portID)

Disable the aging function for a specific port.

IX_ETH_DB_PUBLIC
IxEthDBStatus **ixEthDBPortAgingEnable** (**IxEthDBPortId** portID)

Enable the aging function for a specific port.

IX_ETH_DB_PUBLIC
void **ixEthDBDatabaseMaintenance** (void)

Performs a maintenance operation on the Ethernet learning/filtering database.

IX_ETH_DB_PUBLIC
IxEthDBStatus **ixEthDBEventProcessorPauseModeSet** (BOOL pauseEnable)

Pauses/resumes Ethernet DB event processor.

IX_ETH_DB_PUBLIC **ixEthDBFilteringDatabaseShowRecords** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBRecordType** recordFilter)

This function displays per port database records, given a record type filter.

IX_ETH_DB_PUBLIC **ixEthDBPortDependencyMapSet** (**IxEthDBPortId** portID, **IxEthDBPortMap**
IxEthDBStatus dependencyPortMap)

Sets the dependency port map for a port.

IX_ETH_DB_PUBLIC **ixEthDBPortDependencyMapGet** (**IxEthDBPortId** portID, **IxEthDBPortMap**
IxEthDBStatus dependencyPortMap)

Retrieves the dependency port map for a port.

IX_ETH_DB_PUBLIC

IxEthDBStatus ixEthDBPortVlanTagSet (**IxEthDBPortId** portID, **IxEthDBVlanTag** vlanTag)

Sets the default 802.1Q VLAN tag for a given port.

IX_ETH_DB_PUBLIC

IxEthDBStatus ixEthDBPortVlanTagGet (**IxEthDBPortId** portID, **IxEthDBVlanTag** *vlanTag)

*Retrieves the default 802.1Q port VLAN tag for a given port (see also **ixEthDBPortVlanTagSet**).*

IX_ETH_DB_PUBLIC

IxEthDBStatus ixEthDBVlanTagSet (IxEthDBMacAddr *macAddr, **IxEthDBVlanTag** vlanTag)

Sets the 802.1Q VLAN tag for a database record.

IX_ETH_DB_PUBLIC

IxEthDBStatus ixEthDBVlanTagGet (IxEthDBMacAddr *macAddr, **IxEthDBVlanTag** *vlanTag)

Retrieves the 802.1Q VLAN tag from a database record given the record MAC address.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipAdd** (**IxEthDBPortId** portID, **IxEthDBVlanId**
IxEthDBStatus vlanID)

Adds a VLAN ID to a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipRangeAdd** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBVlanId vlanIDMin, **IxEthDBVlanId** vlanIDMax)

Adds a VLAN ID range to a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipRemove** (**IxEthDBPortId** portID, **IxEthDBVlanId**
IxEthDBStatus vlanID)

Removes a VLAN ID from a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipRangeRemove** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBVlanId vlanIDMin, **IxEthDBVlanId** vlanIDMax)

Removes a VLAN ID range from a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipSet** (**IxEthDBPortId** portID, **IxEthDBVlanSet**
IxEthDBStatus vlanSet)

Sets a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBPortVlanMembershipGet** (**IxEthDBPortId** portID, **IxEthDBVlanSet**
IxEthDBStatus vlanSet)

Retrieves a port's VLAN membership table.

IX_ETH_DB_PUBLIC **ixEthDBAcceptableFrameTypeSet** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBFrameFilter frameFilter)

Sets a port's acceptable frame type filter.

IX_ETH_DB_PUBLIC **ixEthDBAcceptableFrameTypeGet** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBFrameFilter *frameFilter)

Retrieves a port's acceptable frame type filter.

IX_ETH_DB_PUBLIC **ixEthDBPriorityMappingTableSet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBPriorityTable** priorityTable)

Sets a port's priority mapping table.

IX_ETH_DB_PUBLIC **ixEthDBPriorityMappingTableGet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBPriorityTable** priorityTable)

Retrieves a port's priority mapping table.

IX_ETH_DB_PUBLIC
IxEthDBStatus **ixEthDBPriorityMappingTableUpdate** (**IxEthDBPortId** portID)

*Reloads the last port priority mapping table set by the user. (see also **ixEthDBPriorityMappingTableSet**).*

IX_ETH_DB_PUBLIC **ixEthDBPriorityMappingClassSet** (**IxEthDBPortId** portID, **IxEthDBPriority**
IxEthDBStatus userPriority, **IxEthDBPriority** trafficClass)

Sets one QoS/user priority => traffic class mapping in a port's priority mapping table.

IX_ETH_DB_PUBLIC **ixEthDBPriorityMappingClassGet** (**IxEthDBPortId** portID, **IxEthDBPriority**
IxEthDBStatus userPriority, **IxEthDBPriority** *trafficClass)

Retrieves one QoS/user priority => traffic class mapping in a port's priority mapping table.

IX_ETH_DB_PUBLIC **ixEthDBEgressVlanEntryTaggingEnabledSet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBVlanId** vlanID, BOOL enabled)

Enables or disables Egress VLAN tagging for a port and a given VLAN.

IX_ETH_DB_PUBLIC **ixEthDBEgressVlanEntryTaggingEnabledGet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBVlanId** vlanID, BOOL *enabled)

Retrieves the Egress VLAN tagging enabling status for a port and VLAN ID.

IX_ETH_DB_PUBLIC **ixEthDBEgressVlanRangeTaggingEnabledSet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBVlanId** vlanIDMin, **IxEthDBVlanId** vlanIDMax, BOOL enabled)

Enables or disables Egress VLAN tagging for a port and given VLAN range.

IX_ETH_DB_PUBLIC **ixEthDBEgressVlanTaggingEnabledSet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBVlanSet** vlanSet)

Sets the complete Egress VLAN tagging table for a port.

IX_ETH_DB_PUBLIC **ixEthDBEgressVlanTaggingEnabledGet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBVlanSet** vlanSet)

Retrieves the complete Egress VLAN tagging table from a port.

IX_ETH_DB_PUBLIC **ixEthDBIngressVlanTaggingEnabledSet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBTaggingAction** taggingAction)

Sets the Ingress VLAN tagging behavior for a port.

IX_ETH_DB_PUBLIC **ixEthDBIngressVlanTaggingEnabledGet** (**IxEthDBPortId** portID,
IxEthDBStatus **IxEthDBTaggingAction** *taggingAction)

*Retrieves the Ingress VLAN tagging behavior from a port (see **ixEthDBIngressVlanTaggingEnabledSet**).*

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBVlanPortExtractionEnable (**IxEthDBPortId** portID, BOOL enable)

Enables or disables port ID extraction.

IX_ETH_DB_PUBLIC**ixEthDBFeatureCapabilityGet** (**IxEthDBPortId** portID, **IxEthDBFeature**
IxEthDBStatus *featureSet)

Retrieves the feature capability set for a port.

IX_ETH_DB_PUBLIC**ixEthDBFeatureEnable** (**IxEthDBPortId** portID, **IxEthDBFeature** feature, BOOL
IxEthDBStatus enabled)

Enables or disables one or more EthDB features.

IX_ETH_DB_PUBLIC**ixEthDBFeatureStatusGet** (**IxEthDBPortId** portID, **IxEthDBFeature** feature,
IxEthDBStatus BOOL *present, BOOL *enabled)

Retrieves the availability and status of a feature set.

IX_ETH_DB_PUBLIC**ixEthDBFeaturePropertyGet** (**IxEthDBPortId** portID, **IxEthDBFeature** feature,
IxEthDBStatus **IxEthDBProperty** property, **IxEthDBPropertyType** *type, void *value)

Retrieves the value of a feature property.

IX_ETH_DB_PUBLIC**ixEthDBFeaturePropertySet** (**IxEthDBPortId** portID, **IxEthDBFeature** feature,
IxEthDBStatus **IxEthDBProperty** property, void *value)

Sets the value of a feature property.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBFeatureStatesRestore (**IxEthDBPortId** portId)

Restores the state of EthDB based on latest settings, following the occurrence of an NPE soft−error. State is restored by re−downloading tables for the enabled features (e.g. Header Conversion and Firewall) or sending configuration messages to NPE.

IX_ETH_DB_PUBLIC**ixEthDBDatabaseClear** (**IxEthDBPortId** portID, **IxEthDBRecordType**
IxEthDBStatus recordType)

Deletes a set of record types from the Ethernet Database.

IX_ETH_DB_PUBLIC**ixEthDBWiFiRecordEntryAdd** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr, **IxEthDBWiFiRecData** *wifiRecData)

Adds "APMAC/BSSID/STATIONS" record to the database, for 802.3 => 802.11 frame header conversion.

IX_ETH_DB_PUBLIC**ixEthDBWiFiStationEntryAdd** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)

Adds an "Access Point to Station" record to the database, for 802.3 => 802.11 frame header conversion.

IX_ETH_DB_PUBLIC**ixEthDBWiFiAccessPointEntryAdd** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr, IxEthDBMacAddr *gatewayMacAddr)

Adds an "Access Point to Access Point" record to the database.

IX_ETH_DB_PUBLIC**ixEthDBWiFiEntryRemove** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)

Removes a WiFi station record.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBWiFiConversionTableDownload (**IxEthDBPortId** portID)
Downloads the MAC address table for 802.3 => 802.11 frame header conversion to the NPE.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBWiFiFrameControlSet (**IxEthDBPortId** portID, UINT16 frameControl)
Sets the GlobalFrameControl field.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBWiFiDurationIDSet (**IxEthDBPortId** portID, UINT16 durationID)
Sets the GlobalDurationID field.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBWiFiBSSIDSet (**IxEthDBPortId** portID, IxEthDBMacAddr *bssid)
Sets the BSSID field.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBSpanningTreeBlockingStateSet (**IxEthDBPortId** portID, BOOL blocked)
Sets the STP blocked/unblocked state for a port.

IX_ETH_DB_PUBLIC**ixEthDBSpanningTreeBlockingStateGet** (**IxEthDBPortId** portID, BOOL
IxEthDBStatus *blocked)
Retrieves the STP blocked/unblocked state for a port.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBFirewallModeSet (**IxEthDBPortId** portID, **IxEthDBFirewallMode** mode)
Sets the firewall mode to use white or black listing.

IX_ETH_DB_PUBLIC**ixEthDBFirewallInvalidAddressFilterEnable** (**IxEthDBPortId** portID, BOOL
IxEthDBStatus enable)
Enables or disables invalid MAC address filtering.

IX_ETH_DB_PUBLIC**ixEthDBFirewallEntryAdd** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)
Adds a MAC address to the firewall address list.

IX_ETH_DB_PUBLIC**ixEthDBFirewallEntryRemove** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr)
Removes a MAC address from the firewall address list.

IX_ETH_DB_PUBLIC**ixEthDBFirewallMaskedEntryAdd** (**IxEthDBPortId** portID, IxEthDBMacAddr
IxEthDBStatus *macAddr, IxEthDBMacAddr *addrMask)
Adds a MAC address + mask to the firewall address list.

IX_ETH_DB_PUBLIC**ixEthDBFirewallMaskedEntryRemove** (**IxEthDBPortId** portID,
IxEthDBStatus IxEthDBMacAddr *macAddr, IxEthDBMacAddr *addrMask)
Removes a MAC address + mask from the firewall address list.

IX_ETH_DB_PUBLIC
IxEthDBStatus ixEthDBFirewallTableDownload (**IxEthDBPortId** portID)
Downloads the MAC firewall table to a port.

IX_ETH_DB_PUBLIC**ixEthDBUserFieldSet** (**IxEthDBRecordType** recordType, IxEthDBMacAddr
IxEthDBStatus *macAddr, **IxEthDBPortId** portID, **IxEthDBVlanId** vlanID, void *field)
Adds a user−defined field to a database record.

IX_ETH_DB_PUBLIC**ixEthDBUserFieldGet** (**IxEthDBRecordType** recordType, IxEthDBMacAddr
IxEthDBStatus *macAddr, **IxEthDBPortId** portId, **IxEthDBVlanId** vlanID, void **field)
Retrieves a user−defined field from a database record.

Detailed Description

ethDB is a library that does provides a MAC address database learning/filtering capability

---

Define Documentation

```
#define IX_ETH_DB_802_1Q_MAX_VLAN_ID
```

Maximum VLAN IDs.

Definition at line **85** of file **IxEthDB.h**.

```
#define IX_ETH_DB_802_1Q_QOS_MASK
```

QoS Mask.

Definition at line **83** of file **IxEthDB.h**.

```
#define IX_ETH_DB_802_1Q_VLAN_MASK
```

VLAN mask.

Definition at line **82** of file **IxEthDB.h**.

```
#define IX_ETH_DB_CHECK_VLAN_ID ( vlanId )
```

checks the VLAN ID

Definition at line **129** of file **IxEthDB.h**.

#define IX_ETH_DB_CHECK_VLAN_TAG (vlanTag)

checks the VLAN ID of the given 802.1Q tag

Definition at line **123** of file **IxEthDB.h**.

#define IX_ETH_DB_GET_QOS_PRIORITY (vlanTag)

gets the QOS priority from the given 802.1Q tag

Definition at line **111** of file **IxEthDB.h**.

#define IX_ETH_DB_GET_VLAN_ID (vlanTag)

returns the VLAN ID from the given 802.1Q tag

Definition at line **105** of file **IxEthDB.h**.

#define IX_ETH_DB_LEARNING_ENTRY_AGE_TIME

The define specifies the filtering database age entry time. Static entries older than IX_ETH_DB_LEARNING_ENTRY_AGE_TIME +/− IX_ETH_DB_MAINTENANCE_TIME shall be removed.

Definition at line **644** of file **IxEthDB.h**.

#define IX_ETH_DB_MAINTENANCE_TIME

The **ixEthDBDatabaseMaintenance** must be called by the user at a frequency of IX_ETH_DB_MAINTENANCE_TIME.

Definition at line **633** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_QUEUE_CONFIGURATION_COMPLETE

Queue configuration complete.

Definition at line **222** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_0_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 0.

Definition at line **212** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_1_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 1.

Definition at line **213** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_2_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 2.

Definition at line **214** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_3_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 3.

Definition at line **215** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_4_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 4.

Definition at line **216** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_5_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 5.

Definition at line **217** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_6_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 6.

Definition at line **218** of file **IxEthDB.h**.

#define IX_ETH_DB_QOS_TRAFFIC_CLASS_7_RX_QUEUE_PROPERTY

Rx queue assigned to traffic class 7.

Definition at line **219** of file **IxEthDB.h**.

```
#define IX_ETH_DB_QOS_TRAFFIC_CLASS_COUNT_PROPERTY
```

Property identifying number the supported number of traffic classes.

Definition at line **211** of file **IxEthDB.h**.

```
#define IX_ETH_DB_SET_QOS_PRIORITY ( vlanTag,
                                     priority  )
```

sets the QOS priority to the given 802.1Q tag

Definition at line **117** of file **IxEthDB.h**.

```
#define IX_ETH_DB_SET_VLAN_ID ( vlanTag,
                                vlanID  )
```

returns the given 802.1Q tag with the VLAN ID field substituted with the given VLAN ID

This macro is used to change the VLAN ID in a 802.1Q tag.

Example:

tag = **IX_ETH_DB_SET_VLAN_ID(tag, 32)**

inserts the VLAN ID "32" in the given tag.

Definition at line **99** of file **IxEthDB.h**.

```
#define IX_ETH_DB_WIFI_MAX_PAD_SIZE
```

Maximum pad size.

Definition at line **226** of file **IxEthDB.h**.

```
#define IX_ETH_DB_WIFI_MIN_PAD_SIZE
```

Minimum pad size.

Definition at line **225** of file **IxEthDB.h**.

```
#define IX_IEEE802_1Q_QOS_PRIORITY_COUNT
```

Number of QoS priorities defined by IEEE802.1Q.

Definition at line **41** of file **IxEthDB.h**.

#define IX_IEEE802_1Q_VLAN_TPID

returns the VLAN TPID (0x8100)

Definition at line **135** of file **IxEthDB.h**.

#define IX_IEEE803_MAC_ADDRESS_SIZE

The size of the MAC address.

Definition at line **35** of file **IxEthDB.h**.

Typedef Documentation

typedef UINT32 IxEthDBPortId

Definition of an IXP400 port.

Definition at line **290** of file **IxEthDB.h**.

typedef UINT8 IxEthDBPortMap[32]

Port dependency map definition.

Definition at line **297** of file **IxEthDB.h**.

typedef UINT32 IxEthDBPriority

QoS priority/traffic class type, valid range is 0..7, 0 being the lowest.

Definition at line **74** of file **IxEthDB.h**.

typedef UINT8 IxEthDBPriorityTable[8]

Priority mapping table; 0..7 QoS priorities used to index, table contains traffic classes.

Definition at line **77** of file **IxEthDB.h**.

typedef UINT32 IxEthDBProperty

Property ID type.

Definition at line **200** of file **IxEthDB.h**.

typedef UINT32 IxEthDBVlanId

VLAN ID type, valid range is 0..4094, 0 signifying no VLAN membership.

Definition at line **68** of file **IxEthDB.h**.

typedef UINT8 IxEthDBVlanSet[512]

A 4096 bit array used to map the complete VLAN ID range.

Definition at line **80** of file **IxEthDB.h**.

typedef UINT32 IxEthDBVlanTag

802.1Q VLAN tag, contains 3 bits user priority, 1 bit CFI, 12 bits VLAN ID

Definition at line **71** of file **IxEthDB.h**.

Enumeration Type Documentation

enum IxEthDBFeature

Enumeration values:

IX_ETH_DB_LEARNING	Learning feature; enables EthDB to learn MAC address (filtering) records, including 802.1Q enabled records.
IX_ETH_DB_FILTERING	Filtering feature; enables EthDB to communicate with the NPEs for downloading filtering information in the NPEs; depends on the learning feature.
IX_ETH_DB_VLAN_QOS	VLAN/QoS feature; enables EthDB to configure NPEs to operate in VLAN/QoS aware modes.
IX_ETH_DB_FIREWALL	Firewall feature; enables EthDB to configure NPEs to operate in firewall mode, using white/black address lists.
IX_ETH_DB_SPANNING_TREE_PROTOCOL	Spanning tree protocol feature; enables EthDB to configure the NPEs as STP nodes.

IX_ETH_DB_WIFI_HEADER_CONVERSION	WiFi 802.3 to 802.11 header conversion feature; enables EthDB to handle WiFi conversion data.
IX_ETH_DB_ADDRESS_MASKING	Masking of MAC addresses using an address mask. Currently only usable in conjunction with the IX_ETH_DB_FIREWALL feature

Definition at line **189** of file **IxEthDB.h**.

enum IxEthDBFirewallMode

Enumeration values:

IX_ETH_DB_FIREWALL_WHITE_LIST	Firewall operates in white−list mode (MAC address based admission).
IX_ETH_DB_FIREWALL_BLACK_LIST	Firewall operates in black−list mode (MAC address based blocking).

Definition at line **155** of file **IxEthDB.h**.

enum IxEthDBFrameFilter

Enumeration values:

IX_ETH_DB_UNTAGGED_FRAMES	Accepts untagged frames.
IX_ETH_DB_VLAN_TAGGED_FRAMES	Accepts tagged frames.
IX_ETH_DB_PRIORITY_TAGGED_FRAMES	Accepts tagged frames with VLAN ID set to 0 (no VLAN membership).
IX_ETH_DB_ACCEPT_ALL_FRAMES	Accepts all the frames.

Definition at line **137** of file **IxEthDB.h**.

enum IxEthDBPropertyType

Enumeration values:

IX_ETH_DB_INTEGER_PROPERTY	4 byte unsigned integer type
IX_ETH_DB_STRING_PROPERTY	NULL−terminated string type of maximum 255 characters (including the terminator).
IX_ETH_DB_MAC_ADDR_PROPERTY	6 byte MAC address type
IX_ETH_DB_BOOL_PROPERTY	4 byte boolean type; can contain only TRUE and FALSE values

Definition at line **202** of file **IxEthDB.h**.

enum IxEthDBRecordType

Enumeration values:

IX_ETH_DB_FILTERING_RECORD	Filtering record MAC address | static/dynamic type | age
IX_ETH_DB_FILTERING_VLAN_RECORD	VLAN−enabled filtering record MAC address | static/dynamic type | age | 802.1Q tag
IX_ETH_DB_WIFI_RECORD	WiFi header conversion record MAC address | optional gateway MAC address
IX_ETH_DB_FIREWALL_RECORD	Firewall record MAC address
IX_ETH_DB_GATEWAY_RECORD	*For internal use only*
IX_ETH_DB_MASK_RECORD	*For internal use only*
IX_ETH_DB_MAX_RECORD_TYPE_INDEX	*For internal use only*
IX_ETH_DB_NO_RECORD_TYPE	None of the registered record types.
IX_ETH_DB_ALL_FILTERING_RECORDS	All the filtering records.
IX_ETH_DB_MASKED_FIREWALL_RECORD	Masked firewall records.
IX_ETH_DB_ALL_RECORD_TYPES	All the record types registered within EthDB.

Definition at line **161** of file **IxEthDB.h**.

enum IxEthDBStatus

Ethernet Database API return values.

Enumeration values:

IX_ETH_DB_SUCCESS	Success.
IX_ETH_DB_FAIL	Failure.
IX_ETH_DB_INVALID_PORT	Invalid port.
IX_ETH_DB_PORT_UNINITIALIZED	Port not initialized.
IX_ETH_DB_MAC_UNINITIALIZED	MAC not initialized.
IX_ETH_DB_INVALID_ARG	Invalid argument.
IX_ETH_DB_NO_SUCH_ADDR	Address not found for search or delete operations.
IX_ETH_DB_NOMEM	Learning database memory full.
IX_ETH_DB_BUSY	Learning database cannot complete operation, access temporarily blocked.
IX_ETH_DB_END	Database browser passed the end of the record set.

IX_ETH_DB_INVALID_VLAN	Invalid VLAN ID (valid range is 0..4094, 0 signifies no VLAN membership, used for priority tagged frames).
IX_ETH_DB_INVALID_PRIORITY	Invalid QoS priority/traffic class (valid range for QoS priority is 0..7, valid range for traffic class depends on run−time configuration).
IX_ETH_DB_NO_PERMISSION	No permission for attempted operation.
IX_ETH_DB_FEATURE_UNAVAILABLE	Feature not available (or not enabled).
IX_ETH_DB_INVALID_KEY	Invalid search key.
IX_ETH_DB_INVALID_RECORD_TYPE	Invalid record type.

Definition at line **47** of file **IxEthDB.h**.

enum IxEthDBTaggingAction

Enumeration values:

IX_ETH_DB_PASS_THROUGH	Leave frame as−is.
IX_ETH_DB_ADD_TAG	Add default port VLAN tag.
IX_ETH_DB_REMOVE_TAG	Remove VLAN tag from frame.
IX_ETH_DB_ENABLE_VLAN	VLAN enable bit.
IX_ETH_DB_DISABLE_VLAN	VLAN disable bit.

Definition at line **146** of file **IxEthDB.h**.

enum IxEthDBWiFiRecordType

The WI−FI record types enum.

The type value should be any of the types

Enumeration values:

IX_ETH_DB_WIFI_AP_TO_STA	Ap to Sta record.
IX_ETH_DB_WIFI_AP_TO_AP	Ap to Ap record.
IX_ETH_DB_WIFI_TO_ETHER	To Ether record.
IX_ETH_DB_WIFI_TO_LOCAL	To Local record.

Definition at line **248** of file **IxEthDB.h**.

enum IxEthDBWiFiVlanTag

The WI−FI VLAN tag/untag enum.

The tag value should be either tag or untag

Enumeration values:

IX_ETH_DB_WIFI_VLAN_NOTAG	Tag 802.11 frames.
IX_ETH_DB_WIFI_VLAN_TAG	Untag 802.11 frames.

Definition at line **263** of file **IxEthDB.h**.

Function Documentation

IxEthDBStatus ixEthDBAcceptableFrameTypeGet (**IxEthDBPortId** *portID*,
IxEthDBFrameFilter * *frameFilter*
)

Retrieves a port's acceptable frame type filter.

For a description of the acceptable frame types see **ixEthDBAcceptableFrameTypeSet**

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − port ID to retrieve the acceptable frame type filter from
frameFilter	**IxEthDBFrameFilter** [out] − location to store the acceptable frame type filter

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *frameFilter* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBAcceptableFrameTypeSet (**IxEthDBPortId** *portID*,
IxEthDBFrameFilter *frameFilter*
)

Sets a port's acceptable frame type filter.

The acceptable frame type is one (or a combination) of the following values:

- IX_ETH_DB_ACCEPT_ALL_FRAMES − accepts all the frames
- IX_ETH_DB_UNTAGGED_FRAMES − accepts untagged frames
- IX_ETH_DB_VLAN_TAGGED_FRAMES − accepts tagged frames
- IX_ETH_DB_PRIORITY_TAGGED_FRAMES − accepts tagged frames with VLAN ID set to 0 (no VLAN membership)

Except for using the exact values given above only the following combinations are valid:

- IX_ETH_DB_UNTAGGED_FRAMES | IX_ETH_DB_VLAN_TAGGED_FRAMES
- IX_ETH_DB_UNTAGGED_FRAMES | IX_ETH_DB_PRIORITY_TAGGED_FRAMES

Please note that IX_ETH_DB_UNTAGGED_FRAMES | IX_ETH_DB_VLAN_TAGGED_FRAMES is equivalent to IX_ETH_DB_ACCEPT_ALL_FRAMES.

- Reentrant − no
- ISR Callable − no

Note:

by default the acceptable frame type filter is set to IX_ETH_DB_ACCEPT_ALL_FRAMES

setting the acceptable frame type to PRIORITY_TAGGED_FRAMES is internally accomplished by changing the frame filter to VLAN_TAGGED_FRAMES and setting the VLAN membership list to include only VLAN ID 0; the membership list will need to be restored manually to an appropriate value if the acceptable frame type filter is changed back to ACCEPT_ALL_FRAMES or VLAN_TAGGED_FRAMES; failure to do so will filter all VLAN traffic bar frames tagged with VLAN ID 0

Parameters:

portID	**IxEthDBPortId**	[in] − port ID to set the acceptable frame type filter to
frameFilter	**IxEthDBFrameFilter**	[in] − acceptable frame type filter

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid frame type filter
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBDatabaseClear (**IxEthDBPortId** *portID*,
IxEthDBRecordType *recordType*
)

Deletes a set of record types from the Ethernet Database.

This function deletes all the records of certain types (specified in the recordType filter) associated with a port. Additionally, the IX_ETH_DB_ALL_PORTS value can be used as port ID to indicate that the specified record types should be deleted for all the ports.

The record type filter can be an ORed combination of the following types:

Record types

- IX_ETH_DB_FILTERING_RECORD
  Filtering record

MAC address	static/dynamic type	age

- IX_ETH_DB_FILTERING_VLAN_RECORD
  VLAN−enabled filtering record

MAC address	static/dynamic type	age	802.1Q tag

- IX_ETH_DB_WIFI_RECORD
  WiFi header conversion record

MAC address	optional gateway MAC address

- IX_ETH_DB_FIREWALL_RECORD
  Firewall record

MAC address

- IX_ETH_DB_ALL_RECORD_TYPES

Any combination of the above types is valid e.g.

(IX_ETH_DB_FILTERING_RECORD | IX_ETH_DB_FILTERING_VLAN_RECORD | IX_ETH_DB_FIREWALL_RECORD),

although some might be redundant (it is not an error to do so) e.g.

(IX_ETH_DB_FILTERING_RECORD | IX_ETH_DB_ALL_RECORD_TYPES)

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
recordType **IxEthDBRecordType** [in] − record type filter

Return values:

IX_ETH_DB_SUCCESS operation completed successfully
IX_ETH_DB_INVALID_PORT portID is not a valid port identifier

IX_ETH_DB_INVALID_ARG invalid *recordType* filter

Note:

If the record type filter contains any unrecognized value (hence the IX_ETH_DB_INVALID_ARG error value is returned) no actual records will be deleted.

void ixEthDBDatabaseMaintenance (void)

Performs a maintenance operation on the Ethernet learning/filtering database.

In order to perform a database maintenance this function must be called every **IX_ETH_DB_MAINTENANCE_TIME** seconds. It should be called regardless of whether learning is enabled or not.

- Reentrant − no
- ISR Callable − no

Note:

this function call will be ignored if the learning feature is disabled

IxEthDBStatus ixEthDBEgressVlanEntryTaggingEnabledGet (**IxEthDBPortId** *portID*,
IxEthDBVlanId *vlanID*,
BOOL * *enabled*
)

Retrieves the Egress VLAN tagging enabling status for a port and VLAN ID.

Parameters:

portID [in] − ID of the port to extract the Egress VLAN ID tagging status from

vlanID VLAN [in] − ID whose tagging status is to be extracted

enabled [in] − user−specifed location where the status is copied to; following the successfull execution of this function the value will be TRUE if Egress VLAN tagging is enabled for the given port and VLAN ID, and FALSE otherwise

- Reentrant − no
- ISR Callable − no

See also:

ixEthDBEgressVlanEntryTaggingEnabledGet

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	invalid VLAN ID (out of range)
IX_ETH_DB_INVALID_ARG	invalid *enabled* argument pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for

the port

```
IxEthDBStatus ixEthDBEgressVlanEntryTaggingEnabledSet (
                                                        IxEthDBPortId portID,
                                                        IxEthDBVlanId vlanID,
                                                        BOOL          enabled
                                                      )
```

Enables or disables Egress VLAN tagging for a port and a given VLAN.

This function enables or disables Egress VLAN tagging for the given port and VLAN ID. If the VLAN tagging for a certain VLAN ID is enabled then all the frames to be transmitted on the given port tagged with the same VLAN ID will be transmitted in a tagged format. If tagging is not enabled for the given VLAN ID, the VLAN tag from the frames matching this VLAN ID will be removed (the frames will be untagged).

VLAN ID 4095 is reserved and should never be used with this function. VLAN ID 0 has the special meaning of "No VLAN membership" and it is used in this context to allow the port to send priority−tagged frames or not.

By default, no Egress VLAN tagging is enabled on any port.

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port to enable or disable the VLAN ID Egress tagging on
vlanID	**IxEthDBVlanId** [in] − VLAN ID to be matched against outgoing frames
enabled	BOOL [in] − TRUE to enable Egress VLAN tagging on the port and given VLAN, and FALSE to disable Egress VLAN tagging

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	invalid VLAN ID (out of range)
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

```
IxEthDBStatus ixEthDBEgressVlanRangeTaggingEnabledSet (
                                                        IxEthDBPortId portID,
                                                        IxEthDBVlanId vlanIDMin,
                                                        IxEthDBVlanId vlanIDMax,
                                                        BOOL          enabled
                                                      )
```

Enables or disables Egress VLAN tagging for a port and given VLAN range.

This function is very similar to **ixEthDBEgressVlanEntryTaggingEnabledSet** with the difference that it can manipulate the Egress tagging status on multiple VLAN IDs, defined by a contiguous range. Note that both limits in the range are explicitly included in the execution of this function.

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port to enable or disable the VLAN ID Egress tagging on
vlanIDMin	**IxEthDBVlanId** [in] − start of the VLAN range to be matched against outgoing frames
vlanIDMax	**IxEthDBVlanId** [in] − end of the VLAN range to be matched against outgoing frames
enabled	BOOL [in] − TRUE to enable Egress VLAN tagging on the port and given VLAN range, and FALSE to disable Egress VLAN tagging

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	invalid VLAN ID (out of range), or do not constitute a range
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_ETH_DB_NO_PERMISSION	attempted to explicitly remove the default port VLAN ID from the tagging table
IX_FAIL	unknown OS or NPE communication error

Note:

Specifically removing the default port VLAN ID from the Egress tagging table by setting both vlanIDMin and vlanIDMax to the VLAN ID portion of the PVID is not allowed by this function and will return IX_ETH_DB_NO_PERMISSION. However, this can be circumvented, should the user specifically desire this, by either using a larger range (vlanIDMin < vlanIDMax) or by using ixEthDBEgressVlanEntryTaggingEnabledSet.

IxEthDBStatus ixEthDBEgressVlanTaggingEnabledGet (**IxEthDBPortId** *portID*, **IxEthDBVlanSet** *vlanSet*)

Retrieves the complete Egress VLAN tagging table from a port.

This function copies the 4096 bit table controlling the Egress VLAN tagging into a user specified area. Each bit in the array indicates whether tagging for the corresponding VLAN (the bit position in the array) is enabled (the bit is set) or not (the bit is unset).

Bit 4095 is reserved and should not be set (it will be ignored if set).

See also:

ixEthDBEgressVlanTaggingEnabledSet

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port whose Egress VLAN tagging behavior is retrieved
vlanSet	**IxEthDBVlanSet** [out] − user location to copy the Egress tagging table into; should have room to store 4096 bits (512 bytes)

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *vlanSet* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBEgressVlanTaggingEnabledSet (**IxEthDBPortId** *portID*,
IxEthDBVlanSet *vlanSet*
)

Sets the complete Egress VLAN tagging table for a port.

This function is used to set the VLAN tagging/untagging per VLAN ID for a given port covering the entire VLAN ID range (0..4094). The *vlanSet* parameter is a 4096 bit array, each bit indicating the Egress behavior for the corresponding VLAN ID. If a bit is set then outgoing frames with the corresponding VLAN ID will be transmitted with the VLAN tag, otherwise the frame will be transmitted without the VLAN tag.

Bit 0 has a special significance, indicating tagging or tag removal for priority−tagged frames.

Bit 4095 is reserved and should never be set (it will be ignored if set).

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port whose Egress VLAN tagging behavior is set
vlanSet	**IxEthDBVlanSet** [in] − 4096 bit array controlling per−VLAN tagging and untagging

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *vlanSet* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

Warning:

This function will automatically add the default port VLAN ID to the Egress tagging table every time it is called. The user should manually call ixEthDBEgressVlanEntryTaggingEnabledSet to prevent tagging on the default port VLAN ID if the default behavior is not intended.

IxEthDBStatus ixEthDBEventProcessorPauseModeSet (BOOL *pauseEnable*)

Pauses/resumes Ethernet DB event processor.

This API is provided to pause/resume Ethernet DB event processor without the need to kill and recreate event processor thread.

- Reentrant − no
- ISR Callable − no

Parameters:

pauseEnable BOOL [in] − pause mode: TRUE/FALSE

Return values:

IX_ETH_DB_SUCCESS	Changing the pause mode is successful.
IX_ETH_DB_FAIL	Ethernet event processor is not started. So, changing the pause mode is unsuccessful.

IxEthDBStatus ixEthDBFeatureCapabilityGet (**IxEthDBPortId** *portID*, **IxEthDBFeature** * *featureSet*)

Retrieves the feature capability set for a port.

This function retrieves the feature capability set for a port or the common capabilities shared between all the ports, writing the feature capability set in a user specified location.

The feature capability set will consist of a set formed by OR−ing one or more of the following values:

- IX_ETH_DB_LEARNING − Learning feature; enables EthDB to learn MAC address (filtering) records, including 802.1Q enabled records
- IX_ETH_DB_FILTERING − Filtering feature; enables EthDB to communicate with the NPEs for downloading filtering information in the NPEs; depends on the learning feature
- IX_ETH_DB_VLAN_QOS − VLAN/QoS feature; enables EthDB to configure NPEs to operate in VLAN/QoS aware modes
- IX_ETH_DB_FIREWALL − Firewall feature; enables EthDB to configure NPEs to operate in firewall mode, using white/black address lists
- IX_ETH_DB_SPANNING_TREE_PROTOCOL − Spanning tree protocol feature; enables EthDB to configure the NPEs as STP nodes
- IX_ETH_DB_WIFI_HEADER_CONVERSION − WiFi 802.3 to 802.11 header conversion feature; enables EthDB to handle WiFi conversion data

Note that EthDB provides only the LEARNING feature for non−NPE ports.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to retrieve the capability set for (use IX_ETH_DB_ALL_PORTS to retrieve the common capabilities shared between all the ports)

featureSet **IxEthDBFeature** [out] − location where the capability set will be written to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *featureSet* pointer

```
IxEthDBStatus ixEthDBFeatureEnable (
    IxEthDBPortId   portID,
    IxEthDBFeature  feature,
    BOOL            enabled
)
```

Enables or disables one or more EthDB features.

Selects one or more features (see **ixEthDBFeatureCapabilityGet** for a description of the supported features) to be enabled or disabled on the selected port (or all the ports).

Note that some features are mutually incompatible:

- IX_ETH_DB_FILTERING is incompatible with IX_ETH_DB_WIFI_HEADER_CONVERSION

Also note that some features require other features to be enabled:

- IX_ETH_DB_FILTERING requires IX_ETH_DB_LEARNING

This function will either enable the entire selected feature set for the selected port (or all the ports), in which case it will return IX_ETH_DB_SUCCESS, or in case of error it will not enable any feature at all and return an appropriate error message.

The following features are enabled by default (for ports with the respective capability), for compatibility reasons with previous versions of CSR:

- IX_ETH_DB_LEARNING
- IX_ETH_DB_FILTERING

All other features are disabled by default and require manual enabling using ixEthDBFeatureEnable.

Default settings for VLAN, QoS, Firewall and WiFi header conversion features:

VLAN

When the VLAN/QoS feature is enabled for a port for the first time the default VLAN behavior of the port is set to be as **permissive** (it will accept all the frames) and **non−interferential** (it will not change any frames) as possible:

- the port VLAN ID (VID) is set to 0
- the Ingress acceptable frame filter is set to accept all frames
- the VLAN port membership is set to the complete VLAN range (0 − 4094)
- the Ingress tagging mode is set to pass−through (will not change frames)
- the Egress tagging mode is to send tagged frames in the entire VLAN range (0 − 4094)

Note that further disabling and re−enabling the VLAN feature for a given port will not reset the port VLAN behavior to the settings listed above. Any VLAN settings made by the user are kept.

QoS

The following default priority mapping table will be used (as per IEEE 802.1Q and IEEE 802.1D):

QoS traffic classes

QoS priority	Default traffic class	Traffic type
0	**2**	**Best effort, default class for unexpedited traffic**
1	**0**	**Background traffic**
2	**1**	**Spare bandwidth**
3	**3**	**Excellent effort**
4	**4**	**Controlled load**
5	**5**	**Video traffic**
6	**6**	**Voice traffic**
7	**7**	**Network control**

Firewall

The port firewall is configured by default in black−list mode, and the firewall address table is empty. This means the firewall will not filter any frames until the feature is configured and the firewall table is downloaded to the NPE.

Spanning Tree

The port is set to STP unblocked mode, therefore it will accept all frames until re−configured.

WiFi header conversion

The WiFi header conversion database is empty, therefore no actual header conversion will take place until this feature is configured and the conversion table downloaded to the NPE.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to enable or disable the features on (use IX_ETH_DB_ALL_PORTS for all the ports)

feature **IxEthDBFeature** [in] − feature or feature set to enable or disable

enabled BOOL [in] − TRUE to enable the feature and FALSE to disable it

Note:

Certain features, from a functional point of view, cannot be disabled as such at NPE level; when such features are set to *disabled* using the EthDB API they will be configured in such a way to determine a behavior equivalent to the feature being disabled. As well as this, disabled features cannot be configured or accessed via the EthDB API (except for getting their status).

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_NO_PERMISSION	attempted to enable mutually exclusive features, or a feature that depends on another feature which is not present or enabled
IX_ETH_DB_FEATURE_UNAVAILABLE	at least one of the features selected is unavailable
IX_FAIL	unknown OS or NPE communication error

```
IxEthDBStatus ixEthDBFeaturePropertyGet (
    IxEthDBPortId           portID,
    IxEthDBFeature          feature,
    IxEthDBProperty         property,
    IxEthDBPropertyType *   type,
    void *                  value
)
```

Retrieves the value of a feature property.

The EthDB features usually contain feature−specific properties describing or controlling how the feature operates. W
(e.g. the firewall operating mode) have their own API, secondary properties can be retrieved using this function.

Properties can be read−only or read−write. ixEthDBFeaturePropertyGet operates with both types of features.

Properties have types associated with them. A descriptor indicating the property type is returned in the *type* argumen

The currently supported properties and their corresponding features are as follows:

Properties for IX_ETH_DB_VLAN_QOS

Property identifier	**Property type**
IX_ETH_DB_QOS_TRAFFIC_CLASS_COUNT_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_0_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_1_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**

IX_ETH_DB_QOS_TRAFFIC_CLASS_2_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_3_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_4_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_5_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_6_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**
IX_ETH_DB_QOS_TRAFFIC_CLASS_7_RX_QUEUE_PROPERTY	**IX_ETH_DB_INTEGER_PROPER**

See also:

ixEthDBFeaturePropertySet

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
feature	**IxEthDBFeature** [in] − EthDB feature for which the property is retrieved
property	**IxEthDBProperty** [in] − property identifier
type	**IxEthDBPropertyType** [out] − location where the property type will be stored
value	void [out] − location where the property value will be stored

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_INVALID_ARG	invalid property identifier, *type* or *value* pointer arguments
IX_ETH_DB_FAIL	incorrect property value or unknown error

IxEthDBStatus ixEthDBFeaturePropertySet (**IxEthDBPortId** *portID*,
IxEthDBFeature *feature*,
IxEthDBProperty *property*,
void * *value*
)

Sets the value of a feature property.

Unlike **ixEthDBFeaturePropertyGet**, this function operates only with read−write properties

The currently supported properties and their corresponding features are as follows:

- IX_ETH_DB_QOS_QUEUE_CONFIGURATION_COMPLETE (for IX_ETH_DB_VLAN_QOS): freezes the availability of traffic classes to the number of traffic classes currently in use

Note that this function creates deep copies of the property values; once the function is invoked the client can free or reuse the memory area containing the original property value.

Copy behavior for different property types is defined as follows:

- IX_ETH_DB_INTEGER_PROPERTY − 4 bytes are copied from the source location
- IX_ETH_DB_STRING_PROPERTY − the source string will be copied up to the NULL " string terminator, maximum of 255 characters
- IX_ETH_DB_MAC_ADDR_PROPERTY − 6 bytes are copied from the source location
- IX_ETH_DB_BOOL_PROPERTY − 4 bytes are copied from the source location; the only allowed values are TRUE (1L) and false (0L)

See also:

ixEthDBFeaturePropertySet

Warning:

IX_ETH_DB_QOS_QUEUE_CONFIGURATION_COMPLETE is provided for EthAcc internal use; do not attempt to set this property directly

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
feature	**IxEthDBFeature** [in] − EthDB feature for which the property is set
property	**IxEthDBProperty** [in] − property identifier
value	void [in] − location where the property value is to be copied from

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_INVALID_ARG	invalid property identifier, *value* pointer, or invalid property value

IxEthDBStatus ixEthDBFeatureStatesRestore (**IxEthDBPortId** *portId*)

Restores the state of EthDB based on latest settings, following the occurrence of an NPE soft−error. State is restored by re−downloading tables for the enabled features (e.g. Header Conversion and Firewall) or sending configuration messages to NPE.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_FAIL	operatoin fails
IX_ETH_DB_INVALID_PORT	portId is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE.

This API will not restore learning table in Ethernet NPE. As the mechanis, to restore the original learning table is not trivial, after soft−error is handled it is expected that Ethernet NPE to learn the new source address again.

```
IxEthDBStatus ixEthDBFeatureStatusGet (
    IxEthDBPortId   portID,
    IxEthDBFeature  feature,
    BOOL *          present,
    BOOL *          enabled
)
```

Retrieves the availability and status of a feature set.

This function returns the availability and status for a feature set. Note that if more than one feature is selected (e.g. IX_ETH_DB_LEARNING | IX_ETH_DB_FILTERING) the "present" and "enabled" return values will be set to TRUE only if all the features in the feature set are present and enabled (not only some).

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
feature	**IxEthDBFeature** [in] − identifier of the feature to retrieve the status for
present	BOOL [out] − location where a boolean flag indicating whether this feature is present will be written to
enabled	BOOL [out] − location where a boolean flag indicating whether this feature is enabled will be written to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	either *present* or *enabled* pointer argument is invalid

IxEthDBStatus ixEthDBFilteringDatabaseSearch (**IxEthDBPortId** * *portID*,
IxEthDBMacAddr * *macAddr*
)

Search the Ethernet learning/filtering database for a MAC address and return the port ID.

Searches the database for a MAC address. The function returns the portID for the MAC address record, if found. If no match is found the function returns IX_ETH_DB_NO_SUCH_ADDR. The portID is only valid if the function finds a match.

- Reentrant − yes
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − port ID the address belongs to (populated only on a successful search)
macAddr	[in] − MAC address to search for

Return values:

IX_ETH_DB_SUCCESS	the record exists in the database
IX_ETH_DB_NO_SUCH_ADDR	the record was not found in the database
IX_ETH_DB_INVALID_ARG	invalid macAddr or portID pointer argument(s)

IxEthDBStatus ixEthDBFilteringDatabaseShowRecords (**IxEthDBPortId** *portID*,
IxEthDBRecordType *recordFilter*
)

This function displays per port database records, given a record type filter.

The supported record type filters are:

- IX_ETH_DB_FILTERING_RECORD − displays the non−VLAN filtering records (MAC address, age, static/dynamic)
- IX_ETH_DB_FILTERING_VLAN_RECORD − displays the VLAN filtering records (MAC address, age, static/dynamic, VLAN ID, CFI, QoS class)
- IX_ETH_DB_FILTERING_RECORD | IX_ETH_DB_FILTERING_VLAN_RECORD − displays the previous two types of records
- IX_ETH_DB_WIFI_RECORD − displays the WiFi header conversion records (MAC address, optional gateway MAC address) and WiFi header conversion parameters (BSSID, Duration/ID)
- IX_ETH_DB_FIREWALL_RECORD − displays the firewall MAC address table and firewall operating mode (white list/black list)
- IX_ETH_DB_ALL_RECORD_TYPES − displays all the record types
- IX_ETH_DB_NO_RECORD_TYPE − displays only the port status (no records are displayed)

Additionally, the status of each port will be displayed, containg the following information: type, capabilities, enabled status, aging enabled status, group membership and maximum frame size.

The port ID can either be an actual port or IX_ETH_DB_ALL_PORTS, in which case the requested information will be displayed for all the ports (grouped by port)

- Reentrant − no
- ISR Callable − no

Parameters:

portID	ID of the port to display information on (use IX_ETH_DB_ALL_PORTS for all the ports)
recordFilter	record type filter

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port ID is not initialized

IxEthDBStatus ixEthDBFilteringDynamicEntryProvision (**IxEthDBPortId** *portID*, IxEthDBMacAddr * *macAddr*)

Populate the Ethernet learning/filtering database with a dynamic MAC address.

Populates the Ethernet learning/filtering database with a dynamic MAC address. This entry will be subject to normal aging function, if aging is enabled on its port. If there is an entry (static or dynamic) with the same MAC address on any port this entry will take precedence. Any other entry with the same MAC address will be removed.

- Reentrant − yes
- ISR Callable − yes

Parameters:

portID	**IxEthDBPortId** [in] − port ID to add the dynamic address to
macAddr	[in] − static MAC address to add

Return values:

IX_ETH_DB_SUCCESS	the add was successful
IX_ETH_DB_FAIL	failed to populate the database entry
IX_ETH_DB_BUSY	failed due to a temporary busy condition (i.e. lack of CPU cycles), try again later
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	learning feature is disabled

IxEthDBStatus ixEthDBFilteringEntryDelete (IxEthDBMacAddr * *macAddr*)

Removes a MAC address entry from the Ethernet learning/filtering database.

Parameters:

macAddr	IxEthDBMacAddr [in] − MAC address to remove

- Reentrant − yes
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	the removal was successful
IX_ETH_DB_NO_SUCH_ADDR	failed to remove the address (not in the database)
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_BUSY	failed due to a temporary busy condition (i.e. lack of CPU cycles), try again later

IxEthDBStatus ixEthDBFilteringPortMaximumFrameSizeSet (**IxEthDBPortId** *portID*,
UINT32 *maximumFrameSize*
)

Set the maximum frame size supported on the given port ID.

This functions set the maximum frame size supported on a specific port ID

- Reentrant − yes
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − port ID to configure
maximumFrameSize	UINT32 [in] − maximum frame size to configure

Return values:

IX_ETH_DB_SUCCESS	the port is configured
IX_ETH_DB_PORT_UNINITIALIZED	the port has not been initialized
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_INVALID_ARG	size parameter is out of range
IX_ETH_DB_NO_PERMISSION	selected port is not an Ethernet NPE
IX_FAIL	unknown OS or NPE communication error

Note:

This maximum frame size is used to filter the frames based on their destination addresses and the capabilities of the destination port. The mximum value that can be set for a NPE port is 16320. (IX_ETHNPE_ACC_FRAME_LENGTH_MAX)

IxEthDBStatus ixEthDBFilteringPortSearch (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Search the Ethernet learning/filtering database for the given MAC address and port ID.

This functions searches the database for a specific port ID and MAC address. Both the port ID and the MAC address have to match in order for the record to be reported as found.

- Reentrant − yes
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − port ID to search for
macAddr [in] − MAC address to search for

Return values:

IX_ETH_DB_SUCCESS	the record exists in the database
IX_ETH_DB_INVALID_ARG	invalid macAddr pointer argument
IX_ETH_DB_NO_SUCH_ADDR	the record was not found in the database
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port ID is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	learning feature is disabled

IxEthDBStatus ixEthDBFilteringPortUpdatingSearch (**IxEthDBPortId** * *portID*,
IxEthDBMacAddr * *macAddr*
)

Search the filtering database for a MAC address, return the port ID and reset the record age.

Searches the database for a MAC address. The function returns the portID for the MAC address record and resets the entry age to 0, if found. If no match is found the function returns IX_ETH_DB_NO_SUCH_ADDR. The portID is only valid if the function finds a match.

- Reentrant − yes
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	the MAC address was found
IX_ETH_DB_NO_SUCH_ADDR	the MAC address was not found
IX_ETH_DB_INVALID_ARG	invalid macAddr or portID pointer argument(s)

IxEthDBStatus ixEthDBFilteringStaticEntryProvision (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Populate the Ethernet learning/filtering database with a static MAC address.

Populates the Ethernet learning/filtering database with a static MAC address. The entry will not be subject to aging. If there is an entry (static or dynamic) with the corresponding MAC address on any port this entry will take precedence. Any other entry with the same MAC address will be removed.

- Reentrant − yes
- ISR Callable − yes

Parameters:

portID **IxEthDBPortId** [in] − port ID to add the static address to
macAddr [in] − static MAC address to add

Return values:

IX_ETH_DB_SUCCESS	the add was successful
IX_ETH_DB_FAIL	failed to populate the database entry
IX_ETH_DB_BUSY	failed due to a temporary busy condition (i.e. lack of CPU cycles), try again later
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	learning feature is disabled

IxEthDBStatus ixEthDBFirewallEntryAdd (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Adds a MAC address to the firewall address list.

Note that adding the same MAC address twice will not return an error but will not actually accomplish anything.

The firewall MAC address list has a limited number of entries; once the maximum number of entries has been reached this function will failed to add more addresses, returning IX_ETH_DB_NOMEM.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
macAddr [in] − MAC address to be added

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_FAIL	maximum number of records reached
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled

IxEthDBStatus ixEthDBFirewallEntryRemove (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Removes a MAC address from the firewall address list.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
macAddr [in] − MAC address to be removed

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_NO_SUCH_ADDR	address not found
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled

```
ixEthDBFirewallInvalidAddressFilterEnable ( IxEthDBPortId portID,
                                            BOOL          enable
                                          )
```

Enables or disables invalid MAC address filtering.

According to IEEE802 it is illegal for a source address to be a multicast or broadcast address. If this feature is enabled the NPE inspects the source MAC addresses of incoming frames and discards them if invalid addresses are detected.

By default this service is enabled, if the firewall feature is supported by the NPE image.

Parameters:

portID ID of the port
enable TRUE to enable invalid MAC address filtering and FALSE to disable it

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

```
IxEthDBStatus ixEthDBFirewallMaskedEntryAdd ( IxEthDBPortId      portID,
                                              IxEthDBMacAddr *   macAddr,
                                              IxEthDBMacAddr *   addrMask
                                            )
```

Adds a MAC address + mask to the firewall address list.

Note that adding the same MAC address twice will not return an error but will not actually accomplish anything.

The firewall MAC address list has a limited number of entries; once the maximum number of entries has been reached this function will failed to add more addresses, returning IX_ETH_DB_NOMEM.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
macAddr	[in] − MAC address to be added
addrMask	[in] − address mask to be added

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_FAIL	maximum number of records reached
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled

IxEthDBStatus ixEthDBFirewallMaskedEntryRemove (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*,
IxEthDBMacAddr * *addrMask*
)

Removes a MAC address + mask from the firewall address list.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
macAddr	[in] − MAC address to be removed
addrMask	[in] − address mask to be added

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_NO_SUCH_ADDR	address not found
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled

IxEthDBStatus ixEthDBFirewallModeSet (**IxEthDBPortId** *portID*,
IxEthDBFirewallMode *mode*
)

Sets the firewall mode to use white or black listing.

When enabled, the NPE MAC address based firewall support operates in two modes:

- white−list mode (MAC address based admission)
  - *mode* set to IX_ETH_DB_FIREWALL_WHITE_LIST
  - only packets originating from MAC addresses contained in the firewall address list are allowed on the Rx path
- black−list mode (MAC address based blocking)
  - *mode* set to IX_ETH_DB_FIREWALL_BLACK_LIST
  - packets originating from MAC addresses contained in the firewall address list are discarded

Parameters:

portID **IxEthDBPortId** [in] − ID of the port

mode **IxEthDBFirewallMode** [in] − firewall mode (IX_ETH_DB_FIREWALL_WHITE_LIST or IX_ETH_DB_FIREWALL_BLACK_LIST)

Note:

by default the firewall operates in black−list mode with an empty address list, hence it doesn't filter any packets

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	Firewall feature not enabled
IX_ETH_DB_INVALID_ARGUMENT	*mode* argument is not a valid firewall configuration mode
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBFirewallTableDownload (**IxEthDBPortId** *portID*)

Downloads the MAC firewall table to a port.

Parameters:

portID ID of the port

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	

	Firewall feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBIngressVlanTaggingEnabledGet (**IxEthDBPortId** *portID*, **IxEthDBTaggingAction** * *taggingAction*)

Retrieves the Ingress VLAN tagging behavior from a port (see **ixEthDBIngressVlanTaggingEnabledSet**).

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port whose Ingress VLAN tagging behavior is set
taggingAction	**IxEthDBTaggingAction** [out] − location where the tagging behavior for the port is written to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *taggingAction* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBIngressVlanTaggingEnabledSet (**IxEthDBPortId** *portID*, **IxEthDBTaggingAction** *taggingAction*)

Sets the Ingress VLAN tagging behavior for a port.

A port's Ingress tagging behavior is controlled by the taggingAction parameter, which can take one of the following values:

- IX_ETH_DB_PASS_THROUGH − leaves the frame unchanged (does not add or remove the VLAN tag)
- IX_ETH_DB_ADD_TAG − adds the VLAN tag if not present, using the default port VID
- IX_ETH_DB_REMOVE_TAG − removes the VLAN tag if present

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port whose Ingress VLAN tagging behavior is set
taggingAction	**IxEthDBTaggingAction** [in] − tagging behavior for the port

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized

IX_ETH_DB_INVALID_ARG	invalid *taggingAction* argument
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBInit (void)

Initializes the Ethernet learning/filtering database.

Note:

calling this function multiple times does not constitute an error; redundant calls will be ignored, returning IX_ETH_DB_SUCCESS

Return values:

IX_ETH_DB_SUCCESS	initialization was successful
IX_ETH_DB_FAIL	initialization failed (OS error)

IxEthDBStatus ixEthDBPortAddressSet (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Sets the port MAC address.

This function is to be called from the Ethernet Access component top−level ixEthDBUnicastAddressSet(). Event processing cannot be enabled for a port until its MAC address has been set.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port whose MAC address is set
macAddr	[in] − port MAC address

Return values:

IX_ETH_DB_SUCCESS	MAC address was set successfully
IX_ETH_DB_FAIL	MAC address was not set due to a message handler failure
IX_ETH_DB_INVALID_PORT	if the port is not an Ethernet NPE

See also:

IxEthDBPortDefs.h for port definitions

IxEthDBStatus ixEthDBPortAgingDisable (**IxEthDBPortId** *portID*)

Disable the aging function for a specific port.

Parameters:

portID	**IxEthDBPortId** [in] − port ID to disable aging on

- Reentrant − yes

- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	aging disabled successfully
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port ID is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	learning feature is disabled

IxEthDBStatus ixEthDBPortAgingEnable (**IxEthDBPortId** *portID*)

Enable the aging function for a specific port.

Enables the aging of dynamic MAC address entries stored in the learning/filtering database

Note:

The aging function relies on the **ixEthDBDatabaseMaintenance** being called with a period of **IX_ETH_DB_MAINTENANCE_TIME** seconds.

- Reentrant − yes
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − port ID to enable aging on

Return values:

IX_ETH_DB_SUCCESS	aging enabled successfully
IX_ETH_DB_INVALID_PORT	portID is invalid
IX_ETH_DB_PORT_UNINITIALIZED	port ID is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	learning feature is disabled

IxEthDBStatus ixEthDBPortDependencyMapGet (**IxEthDBPortId** *portID*, **IxEthDBPortMap** *dependencyPortMap*)

Retrieves the dependency port map for a port.

Parameters:

portID	ID of the port to set the dependency map to
dependencyPortMap	location where the port dependency map is to be copied

This function will copy the port dependency map to a user specified location.

- Reentrant − no
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *dependencyPortMap* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	Filtering is not available or not enabled for the port

IxEthDBStatus ixEthDBPortDependencyMapSet (**IxEthDBPortId** *portID*, **IxEthDBPortMap** *dependencyPortMap*)

Sets the dependency port map for a port.

Parameters:

portID	ID of the port to set the dependency map to
dependencyPortMap	new dependency map (as bitmap, each bit set indicates a port being included)

This function is used to share filtering information between ports. By adding a port into another port's dependency map the target port filtering data will import the filtering data from the port it depends on. Any changes to filtering data for a port − such as adding, updating or removing records − will trigger updates in the filtering information for all the ports depending on on the updated port.

For example, if ports 2 and 3 are set in the port 0 dependency map the filtering information for port 0 will also include the filtering information from ports 2 and 3. Adding a record to port 2 will also trigger an update not only on port 2 but also on port 0.

The dependency map is a 256 bit array where each bit corresponds to a port corresponding to the bit offset (bit 0 − port 0, bit 1 − port 1 etc). Setting a bit to 1 indicates that the corresponding port is the port map. For example, a dependency port map of 0x14 consists in the ports with IDs 2 and 4. Note that the last bit (offset 255) is reserved and should never be set (it will be automatically cleared by the function).

By default, each port has a dependency port map consisting only of itself, i.e.

```
IxEthDBPortMap portMap;

// clear all ports from port map
ixOsalMemSet(portMap, 0, sizeof (portMap));

// include portID in port map
portMap[portID / 8] = 1 << (portID % 8);
```

- Reentrant − no
- ISR Callable − no

Note:

Setting dependency maps is useful for NPE ports, which benefit from automatic updates of filtering information. Setting dependency maps for user−defined ports is not an error but will have no actual effect.

Including a port in its own dependency map is not compulsory, however note that in this case updating the port will not trigger an update on the port itself, which might not be the intended behavior

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *dependencyPortMap* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	Filtering is not available or not enabled for the port

IxEthDBStatus ixEthDBPortDisable (**IxEthDBPortId** *portID*)

Disables processing on a port.

This function is called automatically from the Ethernet Access component **ixEthAccPortDisable()** routine for Ethernet NPE ports and should be manually called for any user−defined port (any port that is not one of the Ethernet NPEs).

Note:

Calling **ixEthAccPortDisable()** will disable the respective Ethernet NPE. After Ethernet NPEs are disabled they are stopped therefore when re−enabled they need to be reset, downloaded with microcode and started. For learning to restart working the user needs to call again ixEthAccPortUnicastMacAddressSet or ixEthDBUnicastAddressSet with the respective port MAC address. Residual MAC addresses learnt before the port was disabled are deleted as soon as the port is disabled. This only applies to dynamic (learnt) entries, static entries do not dissapear when the port is disabled.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to disable processing on

Return values:

IX_ETH_DB_SUCCESS	if disabling is successful
IX_ETH_DB_FAIL	if the disabling was not successful due to a message handler error
IX_ETH_DB_INVALID_PORT	if portID is invalid

Note:

calling this function multiple times after the first time completed successfully does not constitute an error; redundant calls will be ignored and return IX_ETH_DB_SUCCESS

IxEthDBStatus ixEthDBPortEnable (**IxEthDBPortId** *portID*)

Enables a port.

This function is called automatically from the Ethernet Access component **ixEthAccPortEnable()** routine for Ethernet NPE ports and should be manually called for any user−defined port (any port that is not one of the Ethernet NPEs).

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to enable processing on

Return values:

IX_ETH_DB_SUCCESS	if enabling is successful
IX_ETH_DB_FAIL	if the enabling was not successful due to a message handler error
IX_ETH_DB_MAC_UNINITIALIZED	the MAC address of this port was not initialized (only for Ethernet NPEs)
IX_ETH_DB_INVALID_PORT	if portID is invalid

Precondition:

ixEthDBPortAddressSet needs to be called prior to enabling the port events for Ethernet NPEs

See also:

ixEthDBPortAddressSet

IxEthDBPortDefs.h for port definitions

Note:

calling this function multiple times does not constitute an error; redundant calls will be ignored

void ixEthDBPortInit (**IxEthDBPortId** *portID*)

Initializes a port.

This function is called automatically by the Ethernet Access **ixEthAccPortInit()** routine for Ethernet NPE ports and should be manually called for any user−defined port (any port that is not one of the two Ethernet NPEs).

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to be initialized

See also:

IxEthDBPortDefs.h for port definitions

Note:

calling this function multiple times does not constitute an error; redundant calls will be ignored

IxEthDBStatus ixEthDBPortVlanMembershipAdd (**IxEthDBPortId** *portID*,
IxEthDBVlanId *vlanID*
)

Adds a VLAN ID to a port's VLAN membership table.

Adding a VLAN ID to a port's VLAN membership table will cause frames tagged with the specified VLAN ID to be accepted by the frame filter, if Ingress VLAN membership filtering is enabled.

- Reentrant − no
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to add the VLAN ID membership to
vlanID **IxEthDBVlanId** [in] − VLAN ID to be added to the port membership table

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	vlanID is not a valid VLAN ID
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

Note:

A port's default VLAN ID is always in its own membership table, hence there is no need to explicitly add it using this function (although it is not an error to do so)

IxEthDBStatus ixEthDBPortVlanMembershipGet (**IxEthDBPortId** *portID*, **IxEthDBVlanSet** *vlanSet*)

Retrieves a port's VLAN membership table.

Retrieves the complete VLAN membership table from a port, containing all the possible 4096 VLAN IDs. The table format is an array containing 4096 bits (512 bytes), where each bit indicates whether the VLAN at that bit index is in the port's membership list (if set) or not (unset).

The bit at index 0, indicating VLAN ID 0, indicates no VLAN membership and therefore no other bit will be set if bit 0 is set.

The bit at index 4095 is reserved and will not be set (it will be ignored if set).

The bit referencing the same VLAN ID as the default port VLAN ID will always be set, as the membership list must contain at least the default port VLAN ID.

- Reentrant − no
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − port ID to retrieve the VLAN membership table from
vlanSet

IxEthDBVlanSet [out] − pointer a location where the VLAN membership table will be written to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *vlanSet* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBPortVlanMembershipRangeAdd (**IxEthDBPortId** *portID*,
IxEthDBVlanId *vlanIDMin*,
IxEthDBVlanId *vlanIDMax*
)

Adds a VLAN ID range to a port's VLAN membership table.

All the VLAN IDs in the specified range will be added to the port VLAN membership table, including the range start and end VLAN IDs. Tagged frames with VLAN IDs in the specified range will be accepted by the frame filter, if Ingress VLAN membership filtering is enabled.

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − port ID to add the VLAN membership range into
vlanIDMin	**IxEthDBVlanId** [in] − start of the VLAN ID range
vlanIDMax	**IxEthDBVlanId** [in] − end of the VLAN ID range

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	the specified VLAN IDs are invalid or do not constitute a range
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

Note:

Is is valid to use the same VLAN ID for both vlanIDMin and vlanIDMax, in which case this function will behave as **ixEthDBPortVlanMembershipAdd**

A port's default VLAN ID is always in its own membership table, hence there is no need to explicitly add it using this function (although it is not an error to do so)

IxEthDBStatus ixEthDBPortVlanMembershipRangeRemove (**IxEthDBPortId** *portID*,
IxEthDBVlanId *vlanIDMin*,
IxEthDBVlanId *vlanIDMax*
)

Removes a VLAN ID range from a port's VLAN membership table.

All the VLAN IDs in the specified range will be removed from the port VLAN membership table, including the range start and end VLAN IDs. Tagged frames with VLAN IDs in the range will be discarded by the frame filter, if Ingress membership filtering is enabled.

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port to remove the VLAN membership range from
vlanIDMin	**IxEthDBVlanId** [in] − start of the VLAN ID range
vlanIDMax	**IxEthDBVlanId** [in] − end of the VLAN ID range

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_VLAN	the specified VLAN IDs are invalid or do not constitute a range
IX_ETH_DB_NO_PERMISSION	attempted to remove the default VLAN ID from the port membership table (both vlanIDMin and vlanIDMax were set to the default port VLAN ID)
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

Note:

Is is valid to use the same VLAN ID for both vlanIDMin and vlanIDMax, in which case function will behave as **ixEthDBPortVlanMembershipRemove**

If the given range overlaps the default port VLAN ID this function will remove all the VLAN IDs in the range except for the port VLAN ID from its own membership table. This situation will be silently dealt with (no error message will be returned) as long as the range contains more than one value (i.e. at least one other value, apart from the default port VLAN ID). If the function is called with the vlanIDMin and vlanIDMax parameters both set to the port default VLAN ID, the function will infer that an attempt was specifically made to remove the default port VLAN ID from the port membership table, in which case the return value will be IX_ETH_DB_NO_PERMISSION.

IxEthDBStatus ixEthDBPortVlanMembershipRemove (**IxEthDBPortId** *portID*,
IxEthDBVlanId *vlanID*
)

Removes a VLAN ID from a port's VLAN membership table.

Frames tagged with a VLAN ID which is not in a port's VLAN membership table will be discarded by the frame filter, if Ingress membership filtering is enabled.

- Reentrant − no
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to remove the VLAN ID membership from
vlanID **IxEthDBVlanId** [in] − VLAN ID to be removed from the port membership table

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_INVALID_VLAN	vlanID is not a valid VLAN ID
IX_ETH_DB_NO_PERMISSION	attempted to remove the default VLAN ID from the port membership table (vlanID was set to the default port VLAN ID)
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

Note:

A port's default VLAN ID cannot be removed from the port's membership table; attempting it will return IX_ETH_DB_NO_PERMISSION

IxEthDBStatus ixEthDBPortVlanMembershipSet (**IxEthDBPortId** *portID*, **IxEthDBVlanSet** *vlanSet*)

Sets a port's VLAN membership table.

Sets a port's VLAN membership table from a complete VLAN table containing all the possible 4096 VLAN IDs. The table format is an array containing 4096 bits (512 bytes), where each bit indicates whether the VLAN at that bit index is in the port's membership list (if set) or not (unset).

The bit at index 0, indicating VLAN ID 0, indicates no VLAN membership and therefore no other bit must be set if bit 0 is set.

The bit at index 4095 is reserved and should never be set (it will be ignored if set).

The bit referencing the same VLAN ID as the default port VLAN ID should always be set, as the membership list must contain at least the default port VLAN ID.

- Reentrant − no
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − port ID to set the VLAN membership table to
vlanSet **IxEthDBVlanSet** [in] − pointer to the VLAN membership table

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *vlanSet* pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBPortVlanTagGet (**IxEthDBPortId** *portID*,
IxEthDBVlanTag * *vlanTag*
)

Retrieves the default 802.1Q port VLAN tag for a given port (see also **ixEthDBPortVlanTagSet**).

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to retrieve the default VLAN tag from
vlanTag **IxEthDBVlanTag** [out] − location to write the default port 802.1Q VLAN tag to

- Reentrant − no
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid vlanTag pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBPortVlanTagSet (**IxEthDBPortId** *portID*,
IxEthDBVlanTag *vlanTag*
)

Sets the default 802.1Q VLAN tag for a given port.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port to set the default VLAN tag to
vlanTag **IxEthDBVlanTag** [in] − default 802.1Q VLAN tag

The tag format has 16 bits and it is defined in the IEEE802.1Q specification. This tag will be used for tagging untagged frames (if enabled) and classifying unexpedited traffic into an internal traffic class (using the user priority field).

802.1Q tag format

3 bits	1 bit	12 bits
user priority	CFI	VID

User Priority : Defines user priority, giving eight (2^3) priority levels. IEEE 802.1P defines the operation for these 3 user priority bits

CFI : Canonical Format Indicator is always set to zero for Ethernet switches. CFI is used for compatibility reason between Ethernet type network and Token Ring type network. If a frame received at an Ethernet port has a CFI set to 1, then that frame should not be forwarded as it is to an untagged port.

VID : VLAN ID is the identification of the VLAN, which is basically used by the standard 802.1Q. It has 12 bits and allow the id entification of 4096 (2^12) VLANs. Of the 4096 possible VIDs, a VID of 0 is used to identify priority frames and value 4095 (FFF) is reserved, so the maximum possible VLAN configurations are 4,094.

- Reentrant − no
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_ETH_DB_INVALID_VLAN	*vlanTag* argument does not parse to a valid 802.1Q VLAN tag

Note:

a VLAN ID value of 0 indicates that the port is not part of any VLAN

the value of the cannonical frame indicator (CFI) field is ignored, the field being used only in frame tagging operations

IxEthDBStatus ixEthDBPriorityMappingClassGet (**IxEthDBPortId** *portID*,
IxEthDBPriority *userPriority*,
IxEthDBPriority * *trafficClass*
)

Retrieves one QoS/user priority => traffic class mapping in a port's priority mapping table.

This function retrieves the internal traffic class associated with a QoS (user) priority from a given port's priority mapping table. Use this function when not all the QoS priority mappings are required (see also **ixEthDBPriorityMappingTableGet**)

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port to set the mapping to
userPriority	**IxEthDBPriority** [in] − user (QoS) priority, between 0 and 7 (0 being the lowest)
trafficClass	**IxEthDBPriority** [out] − location to write the corresponding internal traffic class to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_PRIORITY	invalid userPriority value (out of range)
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_ETH_DB_INVALID_ARG	invalid *trafficClass* pointer argument

IxEthDBStatus ixEthDBPriorityMappingClassSet (**IxEthDBPortId** *portID*,
IxEthDBPriority *userPriority*,
IxEthDBPriority *trafficClass*
)

Sets one QoS/user priority => traffic class mapping in a port's priority mapping table.

This function establishes a mapping between a user (QoS) priority and an internal traffic class. The mapping will be saved in the port's priority mapping table. Use this function when not all the QoS priorities need remapping (see also **ixEthDBPriorityMappingTableSet**)

- Reentrant − no
- ISR Callable − no

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port to set the mapping to
userPriority	**IxEthDBPriority** [in] − user (QoS) priority, between 0 and 7 (0 being the lowest)
trafficClass	**IxEthDBPriority** [in] − internal traffic class, between 0 and 3 (0 being the lowest)

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_PRIORITY	*userPriority* out of range or *trafficClass* is beyond the number of currently available traffic classes
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IX_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBPriorityMappingTableGet (**IxEthDBPortId** *portID*, **IxEthDBPriorityTable** *priorityTable*)

Retrieves a port's priority mapping table.

The priority mapping table for the given port will be copied in the location specified by the caller using "priorityTable"

- Reentrant − no
- ISR Callable − no

Parameters:

portID	ID **IxEthDBPortId** [in] − of the port to retrieve the priority mapping table from
priorityTable	**IxEthDBPriorityTable** [out] − pointer to a user specified location where the table will be copied to

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid priorityTable pointer
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port

IxEthDBStatus ixEthDBPriorityMappingTableSet (**IxEthDBPortId** *portID*, **IxEthDBPriorityTable** *priorityTable*)

Sets a port's priority mapping table.

The priority mapping table is an 8x2 table mapping a QoS (user) priority into an internal traffic class. There are 8 valid QoS priorities (0..7, 0 being the lowest) which can be mapped into one of the 8 available traffic classes (0..7, 0 being the lowest). If a custom priority mapping table is not specified using this function the following default priority table will be used (as per IEEE 802.1Q and IEEE 802.1D):

QoS traffic classes

QoS priority	Default traffic class	Traffic type
0	**2**	**Best effort, default class for unexpedited traffic**
1	**0**	**Background traffic**
2	**1**	**Spare bandwidth**
3	**3**	**Excellent effort**

4	**4**	**Controlled load**
5	**5**	**Video traffic**
6	**6**	**Voice traffic**
7	**7**	**Network control**

- Reentrant − no
- ISR Callable − no

Parameters:

portID **IxEthDBPortId** [in] − port ID of the port to set the priority mapping table to
priorityTable **IxEthDBPriorityTable** [in] − location of the user priority table

Note:

The provided table will be copied into internal data structures in EthDB and can be deallocated by the called after this function has completed its execution, if so desired

Warning:

The number of available traffic classes differs depending on the NPE images and queue configuration. Check **IxEthDBQoS.h** for up−to−date information on the availability of traffic classes. Note that specifiying a traffic class in the priority map which exceeds the system availability will produce an IX_ETH_DB_INVALID_PRIORITY return error code and no priority will be remapped.

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *priorityTable* pointer
IX_ETH_DB_INVALID_PRIORITY	at least one priority value exceeds the current number of available traffic classes
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBPriorityMappingTableUpdate (**IxEthDBPortId** *portID*)

Reloads the last port priority mapping table set by the user. (see also **ixEthDBPriorityMappingTableSet**).

Parameters:

portID **IxEthDBPortId** [in] − ID of the port

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_FAIL	operatoin fails

Note:

The function is useful for error−handling module to handle soft−error in Ethernet NPE.
As this function is needed in non VLAN/QoS case to reconfigure Ethernet Rx queue, it does not checks for VLAN/QoS feature is enabled.

IxEthDBStatus ixEthDBSpanningTreeBlockingStateGet (**IxEthDBPortId** *portID*,
BOOL * *blocked*
)

Retrieves the STP blocked/unblocked state for a port.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
blocked BOOL * [in] − set to TRUE if the port is STP blocked, FALSE otherwise

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *blocked* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	Spanning Tree Protocol feature not enabled

IxEthDBStatus ixEthDBSpanningTreeBlockingStateSet (**IxEthDBPortId** *portID*,
BOOL *blocked*
)

Sets the STP blocked/unblocked state for a port.

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
blocked BOOL [in] − TRUE to set the port as STP blocked, FALSE to set it as unblocked

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	Spanning Tree Protocol feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBUnload (void)

Stops and prepares the EthDB component for unloading.

Return values:

IX_ETH_DB_SUCCESS	de−initialization was successful
IX_ETH_DB_BUSY	de−initialization failed, ports must be disabled first
IX_ETH_DB_FAIL	de−initialization failed (OS error)

IxEthDBStatus ixEthDBUserFieldGet (**IxEthDBRecordType** *recordType*,
IxEthDBMacAddr * *macAddr*,
IxEthDBPortId *portId*,
IxEthDBVlanId *vlanID*,
void ** *field*
)

Retrieves a user−defined field from a database record.

The database record is identified using a combination of the given parameters, depending on the record type. All the record types require the record MAC address.

- IX_ETH_DB_FILTERING_RECORD requires only the MAC address
- IX_ETH_DB_VLAN_FILTERING_RECORD requires the MAC address and the VLAN ID
- IX_ETH_DB_WIFI_RECORD requires the MAC address and the portID
- IX_ETH_DB_FIREWALL_RECORD requires the MAC address and the portID

Please note that if a parameter is not required it is completely ignored (it does not undergo parameter checking).

If no user−defined field was registered with the specified record then **NULL** will be written at the location specified by *field*.

Parameters:

recordType	type of record (can be IX_ETH_DB_FILTERING_RECORD, IX_ETH_DB_FILTERING_VLAN_RECORD, IX_ETH_DB_WIFI_RECORD or IX_ETH_DB_FIREWALL_RECORD)
portID	ID of the port (required only for WIFI and FIREWALL records)
macAddr	MAC address of the record
vlanID	VLAN ID of the record (required only for FILTERING_VLAN records)
field	location to write the user defined field into

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID was required but it is not a valid port identifier
IX_ETH_DB_INVALID_ARG	invalid *macAddr* or *field* pointer arguments
IX_ETH_DB_NO_SUCH_ADDR	record not found

```
IxEthDBStatus ixEthDBUserFieldSet ( IxEthDBRecordType recordType,
                                    IxEthDBMacAddr * macAddr,
                                    IxEthDBPortId portID,
                                    IxEthDBVlanId vlanID,
                                    void * field
                                  )
```

Adds a user−defined field to a database record.

This function associates a user−defined field to a database record. The user−defined field is passed as a *(void *)* parameter, hence it can be used for any purpose (such as identifying a structure). Retrieving the user−defined field from a record is done using **ixEthDBUserFieldGet**. Note that EthDB never uses the user−defined field for any internal operation and it is not aware of the significance of its contents. The field is only stored as a pointer.

The database record is identified using a combination of the given parameters, depending on the record type. All the record types require the record MAC address.

- IX_ETH_DB_FILTERING_RECORD requires only the MAC address
- IX_ETH_DB_VLAN_FILTERING_RECORD requires the MAC address and the VLAN ID
- IX_ETH_DB_WIFI_RECORD requires the MAC address and the portID
- IX_ETH_DB_FIREWALL_RECORD requires the MAC address and the portID

Please note that if a parameter is not required it is completely ignored (it does not undergo parameter checking). The user−defined field can be cleared using a **NULL** *field* parameter.

Parameters:

recordType	**IxEthDBRecordType** [in] − type of record (can be IX_ETH_DB_FILTERING_RECORD, IX_ETH_DB_FILTERING_VLAN_RECORD, IX_ETH_DB_WIFI_RECORD or IX_ETH_DB_FIREWALL_RECORD)
portID	**IxEthDBPortId** [in] − ID of the port (required only for WIFI and FIREWALL records)
macAddr	* [in] − MAC address of the record
vlanID	**IxEthDBVlanId** [in] − VLAN ID of the record (required only for FILTERING_VLAN records)
field	void * [in] − user defined field

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID was required but it is not a valid port identifier
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_NO_SUCH_ADDR	record not found

```
IxEthDBStatus ixEthDBVlanPortExtractionEnable ( IxEthDBPortId portID,
                                                BOOL enable
                                              )
```

Enables or disables port ID extraction.

This feature can be used in the situation when a multi−port device (e.g. a switch) is connected to an IXP4XX port and the device can provide incoming frame port identification by tagging the TPID field in the Ethernet frame. Enabling port extraction will instruct the NPE to copy the TPID field from the frame and place it in the *ixp_ne_src_port* of the *ixp_buf* header. In addition, the NPE restores the TPID field to 0.

If the frame is not tagged the NPE will fill the *ixp_ne_src_port* with the port ID of the MII interface the frame was received from.

The TPID field is the least significant byte of the type/length field, which is normally set to 0x8100 for 802.1Q−tagged frames.

This feature is disabled by default.

Parameters:

portID ID of the port to configure port ID extraction on
enable TRUE to enable port ID extraction and FALSE to disable it

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	VLAN/QoS feature is not available or not enabled for the port
IX_FAIL	unknown OS or NPE communication error

```
ixEthDBVlanTagGet ( IxEthDBMacAddr *  macAddr,
                    IxEthDBVlanTag *  vlanTag
                  )
```

Retrieves the 802.1Q VLAN tag from a database record given the record MAC address.

Parameters:

macAddr MAC address
vlanTag location to write the record 802.1Q VLAN tag to

Note:

VLAN tags can be retrieved only from IX_ETH_DB_FILTERING_VLAN_RECORD type records

This function is used together with ixEthDBVlanTagSet to provide MAC−based VLAN classification support. Please note that the bridging application must contain specific code to make use of this feature (see **ixEthDBVlanTagSet**).

- Reentrant − no
- ISR Callable − no

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully

IX_ETH_DB_INVALID_ARG	invalid *macAddr* or *vlanTag* pointer
IX_ETH_DB_NO_SUCH_ADDR	a filtering record with the specified MAC address was not found

IxEthDBStatus ixEthDBVlanTagSet (IxEthDBMacAddr * *macAddr*,
IxEthDBVlanTag *vlanTag*
)

Sets the 802.1Q VLAN tag for a database record.

Parameters:

macAddr	MAC address
vlanTag	802.1Q VLAN tag

This function is used together with **ixEthDBVlanTagGet** to provide MAC−based VLAN classification support. Please note that the bridging application must contain specific code to make use of this feature (see below).

VLAN tags can be set only in IX_ETH_DB_FILTERING_RECORD or IX_ETH_DB_FILTERING_VLAN_RECORD type records. If to an IX_ETH_DB_FILTERING_RECORD type record is added a VLAN tag the record type is automatically changed to IX_ETH_DB_FILTERING_VLAN_RECORD. Once this has occurred the record type will never revert to a non−VLAN type (unless deleted and re−added).

Record types used for different purposes (such as IX_ETH_DB_WIFI_RECORD) will be ignored by this function.

After using this function to associate a VLAN ID with a MAC address the VLAN ID can be extracted knowing the MAC address using **ixEthDBVlanTagGet**. This mechanism can be used to implement MAC−based VLAN classification if a bridging application searches for the VLAN tag when receiving a frame based on the source MAC address (contained in the *ixp_ne_src_mac* field of the buffer header). If found in the database, the application can instruct the NPE to tag the frame by writing the VLAN tag in the *ixp_ne_vlan_tci* field of the buffer header. This way the NPE will inspect the Egress tagging rule associated with the given VLAN ID on the Tx port and tag the frame if Egress tagging on the VLAN is allowed. Additionally, Egress tagging can be forced by setting the *ixp_ne_tx_flags.tag_over* and *ixp_ne_tx_flags.tag_mode* flags in the buffer header.

- Reentrant − no
- ISR Callable − no

Note:

this function will **not** add a filtering record, it can only be used to update an existing one

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer
IX_ETH_DB_NO_SUCH_ADDR	a filtering record with the specified MAC address was not found
IX_ETH_DB_INVALID_VLAN	*vlanTag* argument does not parse to a valid 802.1Q VLAN tag

```
IxEthDBStatus ixEthDBWiFiAccessPointEntryAdd ( IxEthDBPortId      portID,
                                               IxEthDBMacAddr *  macAddr,
                                               IxEthDBMacAddr *  gatewayMacAddr
                                             )
```

Adds an "Access Point to Access Point" record to the database.

See also:

ixEthDBWiFiStationEntryAdd

Note that adding the same MAC address twice will simply overwrite the previously defined gateway MAC address value in the same record, if the record was previously of the "Access Point to Access Point" type.

Re−adding a MAC address as "Access Point to Access Point", which was previously added as "Access Point to Station" will migrate the record type to "Access Point to Access Point" and record the gateway MAC address.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
macAddr	[in] − MAC address to add
gatewayMacAddr	[in] − MAC address of the gateway Access Point

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_INVALID_ARG	invalid *macAddr* or *gatewayMacAddr* or *bssid* pointer argument
IX_ETH_DB_FAIL	maximum number of records reached
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later

```
IxEthDBStatus ixEthDBWiFiBSSIDSet ( IxEthDBPortId      portID,
                                    IxEthDBMacAddr *  bssid
                                  )
```

Sets the BSSID field.

The BSSID field is a 6−byte value which identifies the infrastructure of the service set managed by the Access Point having the IXP400 as its processor. The value is written in the *BSSID* field of the 802.11 frame header. The BSSID value is the MAC address of the Access Point.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
bssid	[in] − pointer to 6 bytes containing the BSSID

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully

IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_ARG	invalid *bssid* pointer argument
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBWiFiConversionTableDownload (**IxEthDBPortId** *portID*)

Downloads the MAC address table for 802.3 => 802.11 frame header conversion to the NPE.

Note that the frame conversion MAC address table must be individually downloaded to each NPE for which the frame header conversion feature is enabled (i.e. it is not possible to specify IX_ETH_DB_ALL_PORTS).

Parameters:

portID **IxEthDBPortId** [in] − ID of the port

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBWiFiDurationIDSet (**IxEthDBPortId** *portID*,
UINT16 *durationID*
)

Sets the GlobalDurationID field.

The GlobalDurationID field is a 2−byte value inserted in the *Duration/ID* field for all 802.3 to 802.11 frame header conversions

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
durationID	UINT16 [in] − GlobalDurationID field

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBWiFiEntryRemove (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Removes a WiFi station record.

This function removes both types of WiFi records ("Access Point to Station" and "Access Point to Access Point").

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
macAddr [in] − MAC address to remove

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port is not initialized
IX_ETH_DB_INVALID_ARG	invalid *macAddr* pointer argument
IX_ETH_DB_NO_SUCH_ADDR	specified address was not found in the database
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later

IxEthDBStatus ixEthDBWiFiFrameControlSet (**IxEthDBPortId** *portID*,
UINT16 *frameControl*
)

Sets the GlobalFrameControl field.

The GlobalFrameControl field is a 2−byte value inserted in the *Frame Control* field for all 802.3 to 802.11 frame header conversions

Parameters:

portID **IxEthDBPortId** [in] − ID of the port
frameControl UINT16 [in] − GlobalFrameControl value

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_PORT_UNINITIALIZED	port not initialized
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_FAIL	unknown OS or NPE communication error

IxEthDBStatus ixEthDBWiFiRecordEntryAdd (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*,
IxEthDBWiFiRecData * *wifiRecData*
)

Adds "APMAC/BSSID/STATIONS" record to the database, for 802.3 => 802.11 frame header conversion.

Frame header conversion is controlled by the set of MAC addresses and bssids , for the backward compatibility the old APIs are still usable to add gateway MAC address and station entry **ixEthDBWiFiAccessPointEntryAdd** and **ixEthDBWiFiStationEntryAdd**. Conversion arguments are added using **ixEthDBWiFiFrameControlSet**, **ixEthDBWiFiDurationIDSet** and **ixEthDBWiFiRecordEntryAdd**.

Note that adding the same MAC address twice will not return an error (but will not accomplish anything either)

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
macAddr	[in] − MAC address to add
wifiRecData	pointer to the wifi specific data (flags, gw etc.)

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled
IX_ETH_DB_INVALID_ARG	invalid *macAddr* , *wifiRecData* , *recType* , *logPort*, *vlanFlag* or *padLen* arguments
IX_ETH_DB_FAIL	maximum number of records reached
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later

IxEthDBStatus ixEthDBWiFiStationEntryAdd (**IxEthDBPortId** *portID*,
IxEthDBMacAddr * *macAddr*
)

Adds an "Access Point to Station" record to the database, for 802.3 => 802.11 frame header conversion.

Frame header conversion is controlled by the set of MAC addresses added using **ixEthDBWiFiStationEntryAdd** and **ixEthDBWiFiAccessPointEntryAdd**. Conversion arguments are added using **ixEthDBWiFiFrameControlSet**, **ixEthDBWiFiDurationIDSet** and **ixEthDBWiFiBSSIDSet**.

Note that adding the same MAC address twice will not return an error (but will not accomplish anything either), while re−adding a record previously added as an "Access Point to Access Point" will migrate the record to the "Access Point to Station" type.

Parameters:

portID	**IxEthDBPortId** [in] − ID of the port
macAddr	[in] − MAC address to add

Return values:

IX_ETH_DB_SUCCESS	operation completed successfully
IX_ETH_DB_INVALID_PORT	portID is not a valid port identifier
IX_ETH_DB_FEATURE_UNAVAILABLE	WiFi feature not enabled

IX_ETH_DB_INVALID_ARG	invalid *macAddr* or *bssid* pointer argument
IX_ETH_DB_FAIL	maximum number of records reached
IX_ETH_DB_BUSY	lock condition or transaction in progress, try again later

Intel (R) IXP400 Software Ethernet Database Port Definitions (IxEthDBPortDefs)

IXP400 Public definition of the ports and port capabilities.

Data Structures

struct **IxEthDBPortDefinition**
Port Definition − a structure contains the Port type.

Defines

#define **IX_ETH_DB_NUMBER_OF_PORTS**
number of supported ports

#define **IX_ETH_DB_UNKNOWN_PORT**
definition of an unknown port

#define **IX_ETH_DB_ALL_PORTS**
Special port ID indicating all the ports.

#define **IX_ETH_DB_PORTS_ASSERTION**
catch invalid port definitions (<2) with a compile−time assertion resulting in a duplicate case error.

#define **IX_ETH_DB_CHECK_PORT**(portID)
safety checks to verify whether the port is invalid or uninitialized

#define **IX_ETH_DB_CHECK_PORT_ALL**(portID)
safety checks to verify whether the port is invalid or uninitialized; tolerates the use of IX_ETH_DB_ALL_PORTS

#define **IX_ETH_DB_QUEUE_UNAVAILABLE**
alias to indicate a queue (traffic class) is not available

#define **IX_IEEE802_1Q_QOS_PRIORITY_COUNT**
number of QoS priorities, according to IEEE 802.1Q

#define **IX_ETH_DB_NPE_A_FUNCTIONALITY_ID_INDEX**
value used to index the NPE A functionality ID in the traffic class definition table

#define **IX_ETH_DB_TRAFFIC_CLASS_COUNT_INDEX**
value used to index the traffic class count in the traffic class definition table

#define **IX_ETH_DB_QUEUE_ASSIGNMENT_INDEX**

value used to index the queue assignment index in the traffic class definition table

Enumerations

enum **IxEthDBPortType** {
IX_ETH_GENERIC,
IX_ETH_NPE
}
Port types − currently only Ethernet NPEs are recognized as specific types All other (user−defined) ports must be specified as IX_ETH_GENERIC.

Variables

IxEthDBPortDefinition ixEthDBPortDefinitions [IX_ETH_DB_NUMBER_OF_PORTS]
Port definitions structure, indexed on the port ID.

Detailed Description

IXP400 Public definition of the ports and port capabilities.

Define Documentation

```
#define IX_ETH_DB_ALL_PORTS
```

Special port ID indicating all the ports.

Note:
This port ID can be used only by a subset of the EthDB API; each function specifically mentions whether this is a valid parameter as the port ID

Definition at line **71** of file **IxEthDBPortDefs.h**.

```
#define IX_ETH_DB_CHECK_PORT ( portID )
```

safety checks to verify whether the port is invalid or uninitialized

Definition at line **84** of file **IxEthDBPortDefs.h**.

```
#define IX_ETH_DB_CHECK_PORT_ALL ( portID )
```

safety checks to verify whether the port is invalid or uninitialized; tolerates the use of IX_ETH_DB_ALL_PORTS

Definition at line **102** of file **IxEthDBPortDefs.h**.

#define IX_ETH_DB_NPE_A_FUNCTIONALITY_ID_INDEX

value used to index the NPE A functionality ID in the traffic class definition table

Definition at line **51** of file **IxEthDBQoS.h**.

#define IX_ETH_DB_NUMBER_OF_PORTS

number of supported ports

Definition at line **44** of file **IxEthDBPortDefs.h**.

#define IX_ETH_DB_PORTS_ASSERTION

catch invalid port definitions (<2) with a compile−time assertion resulting in a duplicate case error.

Definition at line **78** of file **IxEthDBPortDefs.h**.

#define IX_ETH_DB_QUEUE_ASSIGNMENT_INDEX

value used to index the queue assignment index in the traffic class definition table

Definition at line **61** of file **IxEthDBQoS.h**.

#define IX_ETH_DB_QUEUE_UNAVAILABLE

alias to indicate a queue (traffic class) is not available

Definition at line **25** of file **IxEthDBQoS.h**.

#define IX_ETH_DB_TRAFFIC_CLASS_COUNT_INDEX

value used to index the traffic class count in the traffic class definition table

Definition at line **56** of file **IxEthDBQoS.h**.

#define IX_ETH_DB_UNKNOWN_PORT

definition of an unknown port

Definition at line **63** of file **IxEthDBPortDefs.h**.

#define IX_IEEE802_1Q_QOS_PRIORITY_COUNT

number of QoS priorities, according to IEEE 802.1Q

Definition at line **32** of file **IxEthDBQoS.h**.

Enumeration Type Documentation

enum IxEthDBPortType

Port types − currently only Ethernet NPEs are recognized as specific types All other (user−defined) ports must be specified as IX_ETH_GENERIC.

Enumeration values:

IX_ETH_GENERIC	generic ethernet port
IX_ETH_NPE	specific Ethernet NPE

Definition at line **25** of file **IxEthDBPortDefs.h**.

Variable Documentation

IxEthDBPortDefinition ixEthDBPortDefinitions[IX_ETH_DB_NUMBER_OF_PORTS]

Port definitions structure, indexed on the port ID.

Warning:

Ports 0 and 1 are used by the Ethernet access component therefore it is essential to be left untouched. Port 2 can be Ethernet Port or WAN port. Port 3 here (WAN) is given as an example port. The NPE firmware also assumes the NPE B to be the port 0, NPE C to be the port 1, and NPE A to be port 2.

Note:

that only 32 ports (0..31) are supported by EthDB

Definition at line **57** of file **IxEthDBPortDefs.h**.

Intel (R) IXP400 Software Ethernet Phy Access (IxEthMii) API

ethMii is a library that does provides access to the Ethernet PHYs

Functions

PUBLIC
IX_STATUS **ixEthMiiPhyScan** (BOOL phyPresent[], UINT32 maxPhyCount)
Scan the MDIO bus for PHYs This function scans PHY addresses 0 through 31, and sets phyPresent[n] to TRUE if a phy is discovered at address n.

PUBLIC**ixEthMiiPhyConfig** (UINT32 phyAddr, BOOL speed100, BOOL fullDuplex, BOOL
IX_STATUS autonegotiate)
Configure a PHY Configure a PHY's speed, duplex and autonegotiation status.

PUBLIC
IX_STATUS **ixEthMiiPhyLoopbackEnable** (UINT32 phyAddr)
Enable PHY Loopback in a specific Eth MII port.

PUBLIC
IX_STATUS **ixEthMiiPhyLoopbackDisable** (UINT32 phyAddr)
Disable PHY Loopback in a specific Eth MII port.

PUBLIC
IX_STATUS **ixEthMiiPhyReset** (UINT32 phyAddr)
Reset a PHY Reset a PHY.

PUBLIC**ixEthMiiLinkStatus** (UINT32 phyAddr, BOOL *linkUp, BOOL *speed100, BOOL
IX_STATUS *fullDuplex, BOOL *autoneg)
Retrieve the current status of a PHY Retrieve the link, speed, duplex and autonegotiation status of a PHY.

PUBLIC
IX_STATUS **ixEthMiiPhyShow** (UINT32 phyAddr)
Display information on a specified PHY Display link status, speed, duplex and Auto Negotiation status.

Detailed Description

ethMii is a library that does provides access to the Ethernet PHYs

---

Function Documentation

```
ixEthMiiLinkStatus ( UINT32  phyAddr,
                     BOOL *  linkUp,
                     BOOL *  speed100,
                     BOOL *  fullDuplex,
                     BOOL *  autoneg
                   )
```

Retrieve the current status of a PHY Retrieve the link, speed, duplex and autonegotiation status of a PHY.

* − Reentrant − no

- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr	UINT32 [in] − the address of the Ethernet PHY (0−31)
linkUp	BOOL [out] − set to TRUE if the link is up
speed100	BOOL [out] − set to TRUE indicates 100Mbit/s, FALSE indicates 10Mbit/s
fullDuplex	BOOL [out] − set to TRUE indicates Full Duplex, FALSE indicates Half Duplex
autoneg	BOOL [out] − set to TRUE indicates autonegotiation is enabled, FALSE indicates autonegotiation is disabled

Returns:

IX_STATUS

- ◊ IX_SUCCESS
- ◊ IX_FAIL : invalid arguments.

```
ixEthMiiPhyConfig ( UINT32 phyAddr,
                    BOOL   speed100,
                    BOOL   fullDuplex,
                    BOOL   autonegotiate
                  )
```

Configure a PHY Configure a PHY's speed, duplex and autonegotiation status.

* − Reentrant − no

- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr	UINT32 [in]
speed100	BOOL [in] − set to TRUE for 100Mbit/s operation, FALSE for 10Mbit/s
fullDuplex	BOOL [in] − set to TRUE for Full Duplex, FALSE for Half Duplex
autonegotiate	BOOL [in] − set to TRUE to enable autonegotiation

Returns:

IX_STATUS

- ◊ IX_SUCCESS
- ◊ IX_FAIL : invalid arguments.

---

ixEthMiiPhyLoopbackDisable (UINT32 *phyAddr*)

Disable PHY Loopback in a specific Eth MII port.

* − Reentrant − no

- ISR Callable − no

Parameters:

phyAddr UINT32 [in] − the address of the Ethernet PHY (0−31)

Returns:

IX_STATUS

- ◊ IX_SUCCESS
- ◊ IX_FAIL : invalid arguments.

---

ixEthMiiPhyLoopbackEnable (UINT32 *phyAddr*)

Enable PHY Loopback in a specific Eth MII port.

Note:

When PHY Loopback is enabled, frames sent out to the PHY from the IXP400 will be looped back to the IXP400. They will not be transmitted out on the wire.

- Reentrant − no
- ISR Callable − no

Parameters:

phyAddr UINT32 [in] − the address of the Ethernet PHY (0−31)

Returns:

IX_STATUS

◊ IX_SUCCESS
◊ IX_FAIL : invalid arguments.

---

ixEthMiiPhyReset (UINT32 *phyAddr*)

Reset a PHY Reset a PHY.

* − Reentrant − no

- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr UINT32 [in] − the address of the Ethernet PHY (0−31)

Returns:

IX_STATUS
◊ IX_SUCCESS
◊ IX_FAIL : invalid arguments.

---

ixEthMiiPhyScan (BOOL *phyPresent*[],
UINT32 *maxPhyCount*
)

Scan the MDIO bus for PHYs This function scans PHY addresses 0 through 31, and sets phyPresent[n] to TRUE if a phy is discovered at address n.

* − Reentrant − no

- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyPresent BOOL [in] − boolean array of IXP400_ETH_ACC_MII_MAX_ADDR entries
maxPhyCount UINT32 [in] − number of PHYs to search for (the scan will stop when the indicated number of PHYs is found).

Returns:

IX_STATUS
◊ IX_ETH_ACC_SUCCESS
◊ IX_ETH_ACC_FAIL : invalid arguments.

ixEthMiiPhyShow (UINT32 *phyAddr*)

Display information on a specified PHY Display link status, speed, duplex and Auto Negotiation status.

* − Reentrant − no

- ISR Callable − no

Precondition:

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

Parameters:

phyAddr UINT32 [in] − the address of the Ethernet PHY (0−31)

Returns:

IX_STATUS

◊ IX_SUCCESS

◊ IX_FAIL : invalid arguments.

Intel (R) IXP400 Software Feature Control (featureCtrl) Public API

The Public API for the IXP400 Feature Control.

Modules

Intel (R) IXP400 Software Configuration for featureCtrl Component

This section describes software configuration in access component.

Defines

#define **IX_FEATURE_CTRL_COMPONENT_DISABLED**
Hardware Component is disabled/unavailable. Return status by ***ixFeatureCtrlComponentCheck()****.*

#define **IX_FEATURE_CTRL_COMPONENT_ENABLED**
Hardware Component is available. Return status by ***ixFeatureCtrlComponentCheck()****.*

#define **IX_FEATURE_CTRL_SILICON_TYPE_A0**
This is the value of A0 Silicon in product ID.

#define **IX_FEATURE_CTRL_SILICON_TYPE_B0**
This is the value of B0 Silicon in product ID.

#define **IX_FEATURE_CTRL_SILICON_STEPPING_MASK**
This is the mask of silicon stepping in product ID.

#define **IX_FEATURE_CTRL_DEVICE_TYPE_MASK**
This is the mask of silicon stepping in product ID.

#define **IX_FEATURE_CTRL_DEVICE_TYPE_OFFSET**
This is the mask of silicon stepping in product ID.

#define **IX_FEATURE_CTRL_XSCALE_FREQ_533**
This is the value of 533MHz Intel XScale(R) Processor in product ID.

#define **IX_FEATURE_CTRL_XSCALE_FREQ_400**
This is the value of 400MHz Intel XScale(R) Processor in product ID.

#define **IX_FEATURE_CTRL_XSCALE_FREQ_266**
This is the value of 266MHz Intel XScale(R) Processor in product ID.

#define **IX_FEATURE_CTRL_XSCALE_FREQ_MASK**
This is the mask of Intel XScale(R) Processor in product ID.

#define **IX_FEATURECTRL_REG_UTOPIA_32PHY**
Maximum UTOPIA PHY available is 32.

#define **IX_FEATURECTRL_REG_UTOPIA_16PHY**
Maximum UTOPIA PHY available is 16.

#define **IX_FEATURECTRL_REG_UTOPIA_8PHY**
Maximum UTOPIA PHY available to is 8.

#define **IX_FEATURECTRL_REG_UTOPIA_4PHY**
Maximum UTOPIA PHY available to is 4.

#define **IX_FEATURECTRL_REG_XSCALE_533FREQ**
Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 533 MHz.

#define **IX_FEATURECTRL_REG_XSCALE_667FREQ**
Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 667 MHz.

#define **IX_FEATURECTRL_REG_XSCALE_400FREQ**
Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 400 MHz.

#define **IX_FEATURECTRL_REG_XSCALE_266FREQ**
Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 266 MHz.

#define **IX_FEATURECTRL_COMPONENT_NOT_AVAILABLE**
Component selected is not available for device.

#define **IX_FEATURECTRL_COMPONENT_ALWAYS_AVAILABLE**
Component selected is not available for device.

#define **IX_FEATURECTRL_ECC**
This value indicates the bit location for ECC in IXP43X Feature Control register. The bit location for ECC is the same as IX_FEATURECTRL_ECC_TIMESYNC bit in IXP46X Feature Control register.

Typedefs

typedef UINT32 **IxFeatureCtrlReg**
Feature Control Register that contains hardware components' availability information.

typedef UINT32 **IxFeatureCtrlProductId**

Product ID of Silicon that contains Silicon Stepping and Maximum Intel XScale(R) Processor Frequency information.

Functions

IxFeatureCtrlDeviceId ixFeatureCtrlDeviceRead (void)
This function gets the type of device that the software is currently running on.

IxFeatureCtrlBuildDevice ixFeatureCtrlSoftwareBuildGet (void)
This function refers to the value set by the compiler flag to determine the type of device the software is built for.

PUBLIC
IxFeatureCtrlReg ixFeatureCtrlHwCapabilityRead (void)
This function reads out the hardware capability of a silicon type as defined in feature control register.This value is different from that returned by ixFeatureCtrlRead() because this function returns the actual hardware component availability.

PUBLIC IX_STATUS **ixFeatureCtrlComponentCheck** (**IxFeatureCtrlComponentType** componentType)
This function will check the availability of hardware component specified as componentType value.

PUBLIC
IxFeatureCtrlProductId ixFeatureCtrlProductIdRead (void)
This function will return the device product ID.

PUBLIC IX_STATUS **ixFeatureCtrlSwConfigurationCheck** (**IxFeatureCtrlSwConfig** swConfigType)
This function checks whether the specified software configuration is enabled or disabled.

PUBLIC void **ixFeatureCtrlSwConfigurationWrite** (**IxFeatureCtrlSwConfig** swConfigType, BOOL enabled)
This function enable/disable the specified software configuration.

PUBLIC void **ixFeatureCtrlSwVersionShow** (void)
This function shows the current software release information for the device.

Detailed Description

The Public API for the IXP400 Feature Control.

Define Documentation

#define IX_FEATURE_CTRL_COMPONENT_DISABLED

Hardware Component is disabled/unavailable. Return status by **ixFeatureCtrlComponentCheck()**.

Definition at line **62** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_COMPONENT_ENABLED

Hardware Component is available. Return status by **ixFeatureCtrlComponentCheck()**.

Definition at line **72** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_DEVICE_TYPE_MASK

This is the mask of silicon stepping in product ID.

Definition at line **173** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_DEVICE_TYPE_OFFSET

This is the mask of silicon stepping in product ID.

Definition at line **187** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_SILICON_STEPPING_MASK

This is the mask of silicon stepping in product ID.

Definition at line **161** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_SILICON_TYPE_A0

This is the value of A0 Silicon in product ID.

Definition at line **130** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_SILICON_TYPE_B0

This is the value of B0 Silicon in product ID.

Definition at line **139** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_XSCALE_FREQ_266

This is the value of 266MHz Intel XScale(R) Processor in product ID.

Definition at line **215** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_XSCALE_FREQ_400

This is the value of 400MHz Intel XScale(R) Processor in product ID.

Definition at line **206** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_XSCALE_FREQ_533

This is the value of 533MHz Intel XScale(R) Processor in product ID.

Definition at line **197** of file **IxFeatureCtrl.h**.

#define IX_FEATURE_CTRL_XSCALE_FREQ_MASK

This is the mask of Intel XScale(R) Processor in product ID.

Definition at line **224** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_COMPONENT_ALWAYS_AVAILABLE

Component selected is not available for device.

Definition at line **378** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_COMPONENT_NOT_AVAILABLE

Component selected is not available for device.

Definition at line **368** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_ECC

This value indicates the bit location for ECC in IXP43X Feature Control register. The bit location for ECC

is the same as IX_FEATURECTRL_ECC_TIMESYNC bit in IXP46X Feature Control register.

Definition at line **565** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_UTOPIA_16PHY

Maximum UTOPIA PHY available is 16.

Definition at line **244** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_UTOPIA_32PHY

Maximum UTOPIA PHY available is 32.

Definition at line **234** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_UTOPIA_4PHY

Maximum UTOPIA PHY available to is 4.

Definition at line **264** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_UTOPIA_8PHY

Maximum UTOPIA PHY available to is 8.

Definition at line **254** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_XSCALE_266FREQ

Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 266 MHz.

Definition at line **308** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_XSCALE_400FREQ

Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 400 MHz.

Definition at line **297** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_XSCALE_533FREQ

Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 533 MHz.

Definition at line **275** of file **IxFeatureCtrl.h**.

#define IX_FEATURECTRL_REG_XSCALE_667FREQ

Maximum frequency available to Intel (R) IXP46X Product Line of Network Processors is 667 MHz.

Definition at line **286** of file **IxFeatureCtrl.h**.

Typedef Documentation

IxFeatureCtrlProductId

Product ID of Silicon that contains Silicon Stepping and Maximum Intel XScale(R) Processor Frequency information.

Definition at line **619** of file **IxFeatureCtrl.h**.

IxFeatureCtrlReg

Feature Control Register that contains hardware components' availability information.

Definition at line **609** of file **IxFeatureCtrl.h**.

Function Documentation

IX_STATUS ixFeatureCtrlComponentCheck (**IxFeatureCtrlComponentType** *componentType*)

This function will check the availability of hardware component specified as componentType value.

Usage Example:

- if(IX_FEATURE_CTRL_COMPONENT_DISABLED != ixFeatureCtrlComponentCheck(IX_FEATURECTRL_NPEA))
- if(IX_FEATURE_CTRL_COMPONENT_ENABLED == ixFeatureCtrlComponentCheck(IX_FEATURECTRL_NPEB))

This function is typically called during component initialization time.

Parameters:

componentType **IxFeatureCtrlComponentType** [in] − the type of a component as defined above as IX_FEATURECTRL_XXX

Returns:

◊ IX_FEATURE_CTRL_COMPONENT_ENABLED if component is available
◊ IX_FEATURE_CTRL_COMPONENT_DISABLED if component is unavailable

IxFeatureDeviceId ixFeatureCtrlDeviceRead (void)

This function gets the type of device that the software is currently running on.

This function reads the feature Ctrl register specifically to obtain the device id. The definitions of the avilable IDs are as above.

Returns:

◊ IxFeatureCtrlDeviceId − the type of device currently running

IxFeatureCtrlReg ixFeatureCtrlHwCapabilityRead (void)

This function reads out the hardware capability of a silicon type as defined in feature control register.This value is different from that returned by ixFeatureCtrlRead() because this function returns the actual hardware component availability.

The bit location of each hardware component is defined above. A value of '1' in bit means the hardware component is not available. A value of '0' means the hardware component is available.

Returns:

◊ IxFeatureCtrlReg − the hardware capability of the device.

Warning:

◊ This function must not be called when the device is running as the result is undefined.

IxFeatureCtrlProductId ixFeatureCtrlProductIdRead (void)

This function will return the device product ID.

Returns:

◊ IxFeatureCtrlProductId − the value of product ID.

IxFeatureCtrlBuildDevice ixFeatureCtrlSoftwareBuildGet (void)

This function refers to the value set by the compiler flag to determine the type of device the software is built for.

The function reads the compiler flag to determine the device the software is built for. When the user executes build in the command line, a compile time flag (__ixp42X/__ixp46X) is set. This API reads this flag and returns the software build type to the calling client.

Returns:

◊ IxFeatureCtrlBuildDevice − the type of device software is built for.

IX_STATUS ixFeatureCtrlSwConfigurationCheck (**IxFeatureCtrlSwConfig** *swConfigType*)

This function checks whether the specified software configuration is enabled or disabled.

Usage Example:

- if(IX_FEATURE_CTRL_SWCONFIG_DISABLED != ixFeatureCtrlSwConfigurationCheck(IX_FEATURECTRL_ETH_LEARNING))
- if(IX_FEATURE_CTRL_SWCONFIG_ENABLED == ixFeatureCtrlSwConfigurationCheck(IX_FEATURECTRL_ETH_LEARNING))

This function is typically called during access component initialization time.

Parameters:

swConfigType **Intel (R) IXP400 Software Configuration for featureCtrl Component** [in] − the type of a software configuration defined in IxFeatureCtrlSwConfig enumeration.

Returns:

◊ IX_FEATURE_CTRL_SWCONFIG_ENABLED if software configuration is enabled.
◊ IX_FEATURE_CTRL_SWCONFIG_DISABLED if software configuration is disabled.

void ixFeatureCtrlSwConfigurationWrite (**IxFeatureCtrlSwConfig** *swConfigType*, BOOL *enabled*)

This function enable/disable the specified software configuration.

Usage Example:

- ixFeatureCtrlSwConfigurationWrite(IX_FEATURECTRL_ETH_LEARNING, TRUE) is used to enable Ethernet Learning Feature
- ixFeatureCtrlSwConfigurationWrite(IX_FEATURECTRL_ETH_LEARNING, FALSE) is used to disable Ethernet Learning Feature

Parameters:

swConfigType IxFeatureCtrlSwConfig [in] − the type of a software configuration defined in IxFeatureCtrlSwConfig enumeration.

enabled BOOL [in] − To enable(TRUE) / disable (FALSE) the specified software configuration.

Returns:

none

void ixFeatureCtrlSwVersionShow (void)

This function shows the current software release information for the device.

Returns:

none

Intel (R) IXP400 Software Configuration for featureCtrl Component

[Intel (R) IXP400 Software Feature Control (featureCtrl) Public API]

This section describes software configuration in access component.

Defines

#define **IX_FEATURE_CTRL_SWCONFIG_DISABLED**

Software configuration is disabled.

#define **IX_FEATURE_CTRL_SWCONFIG_ENABLED**

Software configuration is enabled.

Enumerations

enum **IxFeatureCtrlBuildDevice** {
IX_FEATURE_CTRL_SW_BUILD_IXP42X,
IX_FEATURE_CTRL_SW_BUILD_IXP46X,
IX_FEATURE_CTRL_SW_BUILD_IXP43X,
IX_FEATURE_CTRL_SW_BUILD_MAX
}

Indicates software build type.

enum **IxFeatureCtrlSwConfig** {
IX_FEATURECTRL_ETH_LEARNING,
IX_FEATURECTRL_ORIGB0_DISPATCHER,
IX_FEATURECTRL_SWCONFIG_MAX
}

Enumeration for software configuration in access components.

enum **IxFeatureCtrlComponentType** {
IX_FEATURECTRL_RCOMP,
IX_FEATURECTRL_USB,
IX_FEATURECTRL_HASH,
IX_FEATURECTRL_AES,
IX_FEATURECTRL_DES,
IX_FEATURECTRL_HDLC,
IX_FEATURECTRL_AAL,
IX_FEATURECTRL_HSS,
IX_FEATURECTRL_UTOPIA,
IX_FEATURECTRL_ETH0,
IX_FEATURECTRL_ETH1,
IX_FEATURECTRL_NPEA,
IX_FEATURECTRL_NPEB,

IX_FEATURECTRL_NPEC,
IX_FEATURECTRL_PCI,
IX_FEATURECTRL_ECC_TIMESYNC,
IX_FEATURECTRL_UTOPIA_PHY_LIMIT,
IX_FEATURECTRL_UTOPIA_PHY_LIMIT_BIT2,
IX_FEATURECTRL_USB_HOST_CONTROLLER,
IX_FEATURECTRL_NPEA_ETH,
IX_FEATURECTRL_NPEB_ETH,
IX_FEATURECTRL_RSA,
IX_FEATURECTRL_XSCALE_MAX_FREQ,
IX_FEATURECTRL_XSCALE_MAX_FREQ_BIT2,
IX_FEATURECTRL_CRYSTAL_OSC_BYPASS,
IX_FEATURECTRL_MAX_COMPONENTS
}
Enumeration for components available.

enum **IxFeatureCtrlDeviceId** {
IX_FEATURE_CTRL_DEVICE_TYPE_IXP42X,
IX_FEATURE_CTRL_DEVICE_TYPE_IXP46X,
IX_FEATURE_CTRL_DEVICE_TYPE_IXP43X,
IX_FEATURE_CTRL_DEVICE_TYPE_MAX
}
Enumeration for device type.

Detailed Description

This section describes software configuration in access component.

The configuration can be changed at run−time. **ixFeatureCtrlSwConfigurationCheck()** will be used across applicable access layer components to check the configuration. **ixFeatureCtrlSwConfigurationWrite()** is used to write the software configuration.

Note:
All software configurations are default to be enabled.

Define Documentation

```
#define IX_FEATURE_CTRL_SWCONFIG_DISABLED
```

Software configuration is disabled.

Definition at line **402** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_SWCONFIG_ENABLED
```

Software configuration is enabled.

Definition at line **412** of file **IxFeatureCtrl.h**.

Enumeration Type Documentation

enum IxFeatureCtrlBuildDevice

Indicates software build type.

Enumeration values:

IX_FEATURE_CTRL_SW_BUILD_IXP42X	Build type is IXP42X.
IX_FEATURE_CTRL_SW_BUILD_IXP46X	Build type is IXP46X.
IX_FEATURE_CTRL_SW_BUILD_IXP43X	Build type is IXP43X.
IX_FEATURE_CTRL_SW_BUILD_MAX	Max build types.

Definition at line **426** of file **IxFeatureCtrl.h**.

enum IxFeatureCtrlComponentType

Enumeration for components available.

Enumeration values:

IX_FEATURECTRL_RCOMP	bit location for RComp Circuitry
IX_FEATURECTRL_USB	bit location for USB Controller
IX_FEATURECTRL_HASH	bit location for Hashing Coprocessor
IX_FEATURECTRL_AES	bit location for AES Coprocessor
IX_FEATURECTRL_DES	bit location for DES Coprocessor
IX_FEATURECTRL_HDLC	bit location for HDLC Coprocessor
IX_FEATURECTRL_AAL	bit location for AAL Coprocessor
IX_FEATURECTRL_HSS	bit location for HSS Coprocessor
IX_FEATURECTRL_UTOPIA	bit location for UTOPIA Coprocessor
IX_FEATURECTRL_ETH0	bit location for Ethernet 0 Coprocessor
IX_FEATURECTRL_ETH1	bit location for Ethernet 1 Coprocessor / In IXP43X refers to NPE C Ethernet Disable
IX_FEATURECTRL_NPEA	bit location for NPE A
IX_FEATURECTRL_NPEB	bit location for NPE B
IX_FEATURECTRL_NPEC	bit location for NPE C
IX_FEATURECTRL_PCI	bit location for PCI Controller
IX_FEATURECTRL_ECC_TIMESYNC	bit location for TimeSync Coprocessor (Applicable only to IXP46X)

IX_FEATURECTRL_UTOPIA_PHY_LIMIT	bit location for Utopia PHY Limit Status
IX_FEATURECTRL_UTOPIA_PHY_LIMIT_BIT2	2nd bit of PHY limit status
IX_FEATURECTRL_USB_HOST_CONTROLLER	bit location for USB host controller
IX_FEATURECTRL_NPEA_ETH	bit location for NPE−A Ethernet Disable
IX_FEATURECTRL_NPEB_ETH	bit location for NPE−B Ethernet 1−3 Coprocessors Disable
IX_FEATURECTRL_RSA	bit location for RSA Crypto block Coprocessors Disable
IX_FEATURECTRL_XSCALE_MAX_FREQ	bit location for Intel XScale(R) processor max frequency
IX_FEATURECTRL_XSCALE_MAX_FREQ_BIT2	2nd Intel XScale(R) processor max freq bit NOT TO BE USED
IX_FEATURECTRL_CRYSTAL_OSC_BYPASS	bit location for Crystal Oscillator Bypass

Definition at line **525** of file **IxFeatureCtrl.h**.

enum IxFeatureCtrlDeviceId

Enumeration for device type.

Warning:

This enum is closely related to the npe image. Its format should comply with formats used in the npe image ImageID. This is indicated by the first nibble of the image ID. This should also be in sync with the with what is defined in CP15. Current available formats are

- IXP42X − 0000
- IXP46X − 0001
- IXP43X − 0004

Enumeration values:

IX_FEATURE_CTRL_DEVICE_TYPE_IXP42X	Device type is IXP42X.
IX_FEATURE_CTRL_DEVICE_TYPE_IXP46X	Device type is IXP46X.
IX_FEATURE_CTRL_DEVICE_TYPE_IXP43X	Device type is IXP43X.
IX_FEATURE_CTRL_DEVICE_TYPE_MAX	Max devices.

Definition at line **584** of file **IxFeatureCtrl.h**.

enum IxFeatureCtrlSwConfig

Enumeration for software configuration in access components.

Entry for new run−time software configuration should be added here.

Enumeration values:

IX_FEATURECTRL_ETH_LEARNING	EthDB Learning Feature.
IX_FEATURECTRL_ORIGB0_DISPATCHER	IXP42X B0 Silicon and IXP46X processor dispatcher without livelock prevention functionality

	Feature.
IX_FEATURECTRL_SWCONFIG_MAX	Maximum boudary for IxFeatureCtrlSwConfig.

Definition at line **443** of file **IxFeatureCtrl.h**.

Intel (R) IXP400 Software HSS Access (IxHssAcc) API

The public API for the IXP400 HssAccess component.

Data Structures

struct **IxHssAccConfigParams**
Structure containing HSS configuration parameters.

struct **IxHssAccHdlcMode**
This structure contains 56Kbps, HDLC−mode configuration parameters.

struct **IxHssAccPktHdlcFraming**
This structure contains information required by the NPE to configure the HDLC co−processor.

struct **IxHssAccPortConfig**
Structure containing HSS port configuration parameters.

Defines

#define **IX_HSSACC_TSLOTS_PER_HSS_PORT**
The max number of TDM timeslots supported per HSS port − 4E1's = 32x4 = 128.

#define **IX_HSSACC_CHAN_TSLOTSWITCH_NUM_GCT_ENTRY_MAX**
The max number of entries supported per gain control table for a timeslot switching voice channel.

#define **IX_HSSACC_CHAN_TSLOTSWITCH_NUM_CHAN_MIN**
The minimum number of TDM timeslots that must be setup as channelised timeslots before timeslot switching service can be enabled. This is to maximize NPE resource utilization in order to guarantee high voice quality and minimum latency. Client can choose to ignore the unwanted channelised timeslots.

#define **IX_HSSACC_CHAN_TSLOTSWITCH_NUM_BYPASS_MAX**
The max number of timeslot switching (bypass) channels supported per HSS port at any one time. In current release, only HSS port 0 supports this feature.

#define **IX_HSSACC_CHAN_TSLOTSWITCH_NUM_HSS_PORT_MAX**
The max number of HSS ports that support timeslot switching (bypass) feature. In current release, only HSS port 0 supports this feature.

#define **IX_HSSACC_PARAM_ERR**
HssAccess function return value for a parameter error.

#define **IX_HSSACC_RESOURCE_ERR**
HssAccess function return value for a resource error.

#define **IX_HSSACC_PKT_DISCONNECTING**
Indicates that a disconnect call is progressing and will disconnect soon.

#define **IX_HSSACC_Q_WRITE_OVERFLOW**
Indicates that an attempt to Tx or to replenish an RxFree Q failed due to Q overflow.

#define **IX_HSSACC_NO_ERROR**
HSS port no error present.

#define **IX_HSSACC_TX_FRM_SYNC_ERR**
HSS port Tx Frame Sync error.

#define **IX_HSSACC_TX_OVER_RUN_ERR**
HSS port Tx over−run error.

#define **IX_HSSACC_CHANNELISED_SW_TX_ERR**
NPE software error in channelised Tx.

#define **IX_HSSACC_PACKETISED_SW_TX_ERR**
NPE software error in packetised Tx.

#define **IX_HSSACC_RX_FRM_SYNC_ERR**
HSS port Rx Frame Sync error.

#define **IX_HSSACC_RX_OVER_RUN_ERR**
HSS port Rx over−run error.

#define **IX_HSSACC_CHANNELISED_SW_RX_ERR**
NPE software error in channelised Rx.

#define **IX_HSSACC_PACKETISED_SW_RX_ERR**
NPE software error in packetised Tx.

#define **IX_HSSACC_PKT_MIN_RX_MBUF_SIZE**
Minimum size of the Rx mbuf in bytes which the client must supply to the component.

Typedefs

typedef
UINT32 **IxHssAccPktUserId**
The client supplied value which will be supplied as a parameter with a given callback.

typedef
void(* **IxHssAccLastErrorCallback**)(unsigned lastHssError, unsigned servicePort)
Prototype of the clients function to accept notification of the last error.

typedef **IxHssAccPktRxCallback**)(IX_OSAL_MBUF *buffer, unsigned numHssErrs,
void(* **IxHssAccPktStatus** pktStatus, **IxHssAccPktUserId** rxUserId)

Prototype of the clients function to accept notification of packetised Rx.

typedef
void(* **IxHssAccPktRxFreeLowCallback**)(**IxHssAccPktUserId** rxFreeLowUserId)

Prototype of the clients function to accept notification of requirement of more Rx Free buffers.

typedef **IxHssAccPktTxDoneCallback**)(IX_OSAL_MBUF *buffer, unsigned numHssErrs,
void(* **IxHssAccPktStatus** pktStatus, **IxHssAccPktUserId** txDoneUserId)

Prototype of the clients function to accept notification of completion with Tx buffers.

typedef **IxHssAccChanRxCallback**)(**IxHssAccHssPort** hssPortId, unsigned rxOffset, unsigned
void(* txOffset, unsigned numHssErrs)

Prototype of the clients function to accept notification of channelised Rx.

Enumerations

enum **IxHssAccHssPort** {
IX_HSSACC_HSS_PORT_0,
IX_HSSACC_HSS_PORT_1,
IX_HSSACC_HSS_PORT_MAX
}

The HSS port ID − There are two identical ports (0−1).

enum **IxHssAccHdlcPort** {
IX_HSSACC_HDLC_PORT_0,
IX_HSSACC_HDLC_PORT_1,
IX_HSSACC_HDLC_PORT_2,
IX_HSSACC_HDLC_PORT_3,
IX_HSSACC_HDLC_PORT_MAX
}

The HDLC port ID − There are four identical HDLC ports (0−3) per HSS port and they correspond to the 4 E1/T1 trunks.

enum **IxHssAccTdmSlotUsage** {
IX_HSSACC_TDMMAP_UNASSIGNED,
IX_HSSACC_TDMMAP_HDLC,
IX_HSSACC_TDMMAP_VOICE56K,
IX_HSSACC_TDMMAP_VOICE64K,
IX_HSSACC_TDMMAP_MAX
}

The HSS TDM stream timeslot assignment types.

enum **IxHssAccFrmSyncType** {
IX_HSSACC_FRM_SYNC_ACTIVE_LOW,
IX_HSSACC_FRM_SYNC_ACTIVE_HIGH,
IX_HSSACC_FRM_SYNC_FALLINGEDGE,
IX_HSSACC_FRM_SYNC_RISINGEDGE,

IX_HSSACC_FRM_SYNC_TYPE_MAX
}
The HSS frame sync pulse type.

enum **IxHssAccFrmSyncEnable** {
IX_HSSACC_FRM_SYNC_INPUT,
IX_HSSACC_FRM_SYNC_INVALID_VALUE,
IX_HSSACC_FRM_SYNC_OUTPUT_FALLING,
IX_HSSACC_FRM_SYNC_OUTPUT_RISING,
IX_HSSACC_FRM_SYNC_ENABLE_MAX
}
The IxHssAccFrmSyncEnable determines how the frame sync pulse is used.

enum **IxHssAccClkEdge** {
IX_HSSACC_CLK_EDGE_FALLING,
IX_HSSACC_CLK_EDGE_RISING,
IX_HSSACC_CLK_EDGE_MAX
}
IxHssAccClkEdge is used to determine the clk edge to use for framing and data.

enum **IxHssAccClkDir** {
IX_HSSACC_SYNC_CLK_DIR_INPUT,
IX_HSSACC_SYNC_CLK_DIR_OUTPUT,
IX_HSSACC_SYNC_CLK_DIR_MAX
}
The HSS clock direction.

enum **IxHssAccFrmPulseUsage** {
IX_HSSACC_FRM_PULSE_ENABLED,
IX_HSSACC_FRM_PULSE_DISABLED,
IX_HSSACC_FRM_PULSE_MAX
}
The HSS frame pulse usage.

enum **IxHssAccDataRate** {
IX_HSSACC_CLK_RATE,
IX_HSSACC_HALF_CLK_RATE,
IX_HSSACC_DATA_RATE_MAX
}
The HSS Data rate in relation to the clock.

enum **IxHssAccDataPolarity** {
IX_HSSACC_DATA_POLARITY_SAME,
IX_HSSACC_DATA_POLARITY_INVERT,
IX_HSSACC_DATA_POLARITY_MAX
}
The HSS data polarity type.

enum **IxHssAccBitEndian** {
IX_HSSACC_LSB_ENDIAN,
IX_HSSACC_MSB_ENDIAN,

IX_HSSACC_ENDIAN_MAX
}
HSS Data endianness.

enum **IxHssAccDrainMode** {
IX_HSSACC_TX_PINS_NORMAL,
IX_HSSACC_TX_PINS_OPEN_DRAIN,
IX_HSSACC_TX_PINS_MAX
}
Tx pin open drain mode.

enum **IxHssAccSOFType** {
IX_HSSACC_SOF_FBIT,
IX_HSSACC_SOF_DATA,
IX_HSSACC_SOF_MAX
}
HSS start of frame types.

enum **IxHssAccDataEnable** {
IX_HSSACC_DE_TRI_STATE,
IX_HSSACC_DE_DATA,
IX_HSSACC_DE_MAX
}
IxHssAccDataEnable is used to determine whether or not to drive the data pins.

enum **IxHssAccTxSigType** {
IX_HSSACC_TXSIG_LOW,
IX_HSSACC_TXSIG_HIGH,
IX_HSSACC_TXSIG_HIGH_IMP,
IX_HSSACC_TXSIG_MAX
}
IxHssAccTxSigType is used to determine how to drive the data pins.

enum **IxHssAccFbType** {
IX_HSSACC_FB_FIFO,
IX_HSSACC_FB_HIGH_IMP,
IX_HSSACC_FB_MAX
}
IxHssAccFbType determines how to drive the Fbit.

enum **IxHssAcc56kEndianness** {
IX_HSSACC_56KE_BIT_7_UNUSED,
IX_HSSACC_56KE_BIT_0_UNUSED,
IX_HSSACC_56KE_MAX
}
56k data endianness when using the 56k type

enum **IxHssAcc56kSel** {
IX_HSSACC_56KS_32_8_DATA,
IX_HSSACC_56KS_56K_DATA,
IX_HSSACC_56KS_MAX

}
56k data transmission type when using the 56k type

enum **IxHssAccClkSpeed** {
IX_HSSACC_CLK_SPEED_512KHZ,
IX_HSSACC_CLK_SPEED_1536KHZ,
IX_HSSACC_CLK_SPEED_1544KHZ,
IX_HSSACC_CLK_SPEED_2048KHZ,
IX_HSSACC_CLK_SPEED_4096KHZ,
IX_HSSACC_CLK_SPEED_8192KHZ,
IX_HSSACC_CLK_SPEED_MAX
}
IxHssAccClkSpeed represents the HSS clock speeds available.

enum **IxHssAccPktStatus** {
IX_HSSACC_PKT_OK,
IX_HSSACC_STOP_SHUTDOWN_ERROR,
IX_HSSACC_HDLC_ALN_ERROR,
IX_HSSACC_HDLC_FCS_ERROR,
IX_HSSACC_RXFREE_Q_EMPTY_ERROR,
IX_HSSACC_HDLC_MAX_FRAME_SIZE_EXCEEDED,
IX_HSSACC_HDLC_ABORT_ERROR,
IX_HSSACC_DISCONNECT_IN_PROGRESS
}
Indicates the status of packets passed to the client.

enum **IxHssAccPktCrcType** {
IX_HSSACC_PKT_16_BIT_CRC,
IX_HSSACC_PKT_32_BIT_CRC
}
HDLC CRC type.

enum **IxHssAccPktHdlcIdleType** {
IX_HSSACC_HDLC_IDLE_ONES,
IX_HSSACC_HDLC_IDLE_FLAGS
}
HDLC idle transmission type.

Functions

PUBLIC **ixHssAccPortInit** (**IxHssAccHssPort** hssPortId, **IxHssAccConfigParams** *configParams,
IX_STATUS **IxHssAccTdmSlotUsage** *tdmMap, **IxHssAccLastErrorCallback** lastHssErrorCallback)
Initialise a HSS port. No channelised or packetised connections should exist in the HssAccess layer while this interface is being called.

PUBLIC
IX_STATUS **ixHssAccLastErrorRetrievalInitiate** (**IxHssAccHssPort** hssPortId)
Initiate the retrieval of the last HSS error. The HSS port should be configured before attempting to call this interface.

PUBLIC
IX_STATUS **ixHssAccInit** (void)

This function is responsible for initialising resources for use by the packetised and channelised clients. It should be called after HSS NPE image has been downloaded into NPE−A and before any other HssAccess interface is called. No other HssAccPacketised interface should be called while this interface is being processed.

PUBLIC
IX_STATUS **ixHssAccUninit** (void)

This function is responsible for un−initialize of resources for use by the packetised and channelised clients. It internally calls the uninitialize functions of sub components. This function should be the last function to be called before exiting HssAcc component.

PUBLIC**ixHssAccPktPortConnect** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId,
IX_STATUS BOOL hdlcFraming, **IxHssAccHdlcMode** hdlcMode, BOOL hdlcBitInvert, unsigned blockSizeInWords, UINT32 rawIdleBlockPattern, **IxHssAccPktHdlcFraming** hdlcTxFraming, **IxHssAccPktHdlcFraming** hdlcRxFraming, unsigned frmFlagStart, **IxHssAccPktRxCallback** rxCallback, **IxHssAccPktUserId** rxUserId, **IxHssAccPktRxFreeLowCallback** rxFreeLowCallback, **IxHssAccPktUserId** rxFreeLowUserId, **IxHssAccPktTxDoneCallback** txDoneCallback, **IxHssAccPktUserId** txDoneUserId)

This function is responsible for connecting a client to one of the 4 available HDLC ports. The HSS port should be configured before attempting a connect. No other HssAccPacketised interface should be called while this connect is being processed.

PUBLIC
IX_STATUS **ixHssAccPktPortEnable** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId)

This function is responsible for enabling a packetised service for the specified HSS/HDLC port combination. It enables the Rx flow. The client must have already connected to a packetised service and is responsible for ensuring an adequate amount of Rx mbufs have been supplied to the access component before enabling the packetised service. This function must be called on a given port before any call to ixHssAccPktPortTx on the same port. No other HssAccPacketised interface should be called while this interface is being processed.

PUBLIC
IX_STATUS **ixHssAccPktPortDisable** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId)

This function is responsible for disabling a packetised service for the specified HSS/HDLC port combination. It disables the Rx flow. The client must have already connected to and enabled a packetised service for the specified HDLC port. This disable interface can be called before a disconnect, but is not required to.

PUBLIC
IX_STATUS **ixHssAccPktPortDisconnect** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId)

This function is responsible for disconnecting a client from one of the 4 available HDLC ports. It is not required that the Rx Flow has been disabled before calling this function. If the Rx Flow has not been disabled, the disconnect will disable it before proceeding with the disconnect. No other HssAccPacketised interface should be called while this interface is being processed.

PUBLIC **ixHssAccPktPortIsDisconnectComplete** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort**
BOOL hdlcPortId)

This function is called to check if a given HSS/HDLC port combination is in a connected state or not. This function may be called at any time to determine a ports state. No other HssAccPacketised interface should be called while this interface is being processed.

PUBLIC **ixHssAccPktPortRxFreeReplenish** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort**
IX_STATUS hdlcPortId, IX_OSAL_MBUF *buffer)

Function which the client calls at regular intervals to provide mbufs to the access component for Rx. A connection should exist for the specified hssPortId/hdlcPortId combination before attempting to call this interface. Also, the connection should not be in a disconnecting state.

PUBLIC **ixHssAccPktPortTx** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId,
IX_STATUS IX_OSAL_MBUF *buffer)

Function which the client calls when it wants to transmit packetised data. An enabled connection should exist on the specified hssPortId/hdlcPortId combination before attempting to call this interface. No other HssAccPacketised interface should be called while this interface is being processed.

PUBLIC **ixHssAccChanConnect** (**IxHssAccHssPort** hssPortId, unsigned bytesPerTSTrigger, UINT8
IX_STATUS *rxCircular, unsigned numRxBytesPerTS, UINT32 *txPtrList, unsigned numTxPtrLists,
unsigned numTxBytesPerBlk, **IxHssAccChanRxCallback** rxCallback)

This function allows the client to connect to the Tx/Rx NPE Channelised Service. There can only be one client per HSS port. The client is responsible for ensuring that the HSS port is configured appropriately before its connect request. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC
IX_STATUS **ixHssAccChanPortEnable** (**IxHssAccHssPort** hssPortId)

This function is responsible for enabling a channelised service for the specified HSS port. It enables the NPE Rx flow. The client must have already connected to a channelised service before enabling the channelised service. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC
IX_STATUS **ixHssAccChanPortDisable** (**IxHssAccHssPort** hssPortId)

This function is responsible for disabling a channelised service for the specified HSS port. It disables the NPE Rx flow. The client must have already connected to and enabled a channelised service for the specified HSS port. This disable interface can be called before a disconnect, but is not required to. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC
IX_STATUS **ixHssAccChanDisconnect** (**IxHssAccHssPort** hssPortId)

This function allows the client to Disconnect from a channelised service. If the NPE Rx Flow has not been disabled, the disconnect will disable it before proceeding with other disconnect functionality. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC **ixHssAccChanStatusQuery** (**IxHssAccHssPort** hssPortId, BOOL *dataRecvd, unsigned
IX_STATUS *rxOffset, unsigned *txOffset, unsigned *numHssErrs)

This function is called by the client to query whether or not channelised data has been received. If there is, hssChanAcc will return the details in the output parameters. An enabled connection should exist on the specified hssPortId before attempting to call this interface. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC IX_STATUS **ixHssAccChanTslotSwitchEnable** (**IxHssAccHssPort** hssPortId, UINT32 srcTimeslot, UINT32 destTimeslot, UINT32 *tsSwitchHandle)

This function is responsible for enabling timeslot switching (bypass) channel between two voice channels for the specified HSS port. The voice channels must have already been configured as channelised timeslot for the specified HSS port. In current release, only up to two timeslot switching channels can be enabled at any one time. In order to minimize bypass delay and ensure better voice quality, this function requires at least 8 TDM timeslots on the specified HSS port to be setup as channelised timeslots. In timeslot switching mode, data received on srcTimeslot is transmitted onto a partner timeslot (i.e. destTimeslot) at NPE level. A copy of the received data on srcTimeslot is also sent to client via HssAccess component. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC IX_STATUS **ixHssAccChanTslotSwitchDisable** (**IxHssAccHssPort** hssPortId, UINT32 tsSwitchHandle)

This function is responsible for disabling timeslot switching (bypass) channel, that is associated with tsSwitchHandle, for the specified HSS port. The bypass channel to be disabled must have already been enabled for the specified HSS port. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC IX_STATUS **ixHssAccChanTslotSwitchGctDownload** (**IxHssAccHssPort** hssPortId, UINT8 *gainCtrlTable, UINT32 tsSwitchHandle)

This function is responsible for downloading a gain control table (256 bytes) to NPE for the specified timeslot switching (bypass) channel, associated with tsSwitchHandle, on the specified HSS port. The bypass voice channel must have already been enabled for the specified HSS port before this API can be called to download the gain control table to NPE. No other HssAccChannelised interface should be called while this interface is being processed.

PUBLIC void **ixHssAccShow** (void)

This function will display the current state of the IxHssAcc component. The output is sent to stdout.

PUBLIC void **ixHssAccStatsInit** (void)

This function will reset the IxHssAcc statistics.

Detailed Description

The public API for the IXP400 HssAccess component.

IxHssAcc is the access layer to the HSS packetised and channelised services

Design Notes

- When a packet−pipe is configured for 56Kbps RAW mode, byte alignment of the transmitted data is not preserved. All raw data that is transmitted will be received in proper order by the receiver, but the first bit of the packet may be seen at any offset within a byte; all subsequent bytes will have the same offset for the duration of the packet. The same offset also applies to all subsequent packets received on the packet−pipe too. (Similar results will occur for data received from remote end.) While this behavior will also occur for 56Kbps HDLC mode, the HDLC encoding/decoding will preserve the original byte alignment at the receiver end.

56Kbps Packetised Service Bandwidth Limitation

- IxHssAcc supports 56Kbps packetised service at a maximum aggregate rate for all HSS ports/HDLC channels of 12.288Mbps[1] in each direction, i.e. it supports 56Kbps packetised service on up to 8 T1 trunks. It does not support 56Kbps packetised service on 8 E1 trunks (i.e. 4 trunks per HSS port) unless those trunks are running 'fractional E1' with maximum aggregate rate of 12.288 Mbps in each direction.
  [1] 12.288Mbps = 1.536Mbp * 8 T1

Define Documentation

#define IX_HSSACC_CHAN_TSLOTSWITCH_NUM_BYPASS_MAX

The max number of timeslot switching (bypass) channels supported per HSS port at any one time. In current release, only HSS port 0 supports this feature.

Definition at line **94** of file **IxHssAcc.h**.

#define IX_HSSACC_CHAN_TSLOTSWITCH_NUM_CHAN_MIN

The minimum number of TDM timeslots that must be setup as channelised timeslots before timeslot switching service can be enabled. This is to maximize NPE resource utilization in order to guarantee high voice quality and minimum latency. Client can choose to ignore the unwanted channelised timeslots.

Definition at line **85** of file **IxHssAcc.h**.

#define IX_HSSACC_CHAN_TSLOTSWITCH_NUM_GCT_ENTRY_MAX

The max number of entries supported per gain control table for a timeslot switching voice channel.

Definition at line **74** of file **IxHssAcc.h**.

#define IX_HSSACC_CHAN_TSLOTSWITCH_NUM_HSS_PORT_MAX

The max number of HSS ports that support timeslot switching (bypass) feature. In current release, only HSS port 0 supports this feature.

Definition at line **102** of file **IxHssAcc.h**.

#define IX_HSSACC_CHANNELISED_SW_RX_ERR

NPE software error in channelised Rx.

Definition at line **196** of file **IxHssAcc.h**.

#define IX_HSSACC_CHANNELISED_SW_TX_ERR

NPE software error in channelised Tx.

Definition at line **168** of file **IxHssAcc.h**.

#define IX_HSSACC_NO_ERROR

HSS port no error present.

Definition at line **147** of file **IxHssAcc.h**.

#define IX_HSSACC_PACKETISED_SW_RX_ERR

NPE software error in packetised Tx.

Definition at line **203** of file **IxHssAcc.h**.

#define IX_HSSACC_PACKETISED_SW_TX_ERR

NPE software error in packetised Tx.

Definition at line **175** of file **IxHssAcc.h**.

#define IX_HSSACC_PARAM_ERR

HssAccess function return value for a parameter error.

Definition at line **114** of file **IxHssAcc.h**.

#define IX_HSSACC_PKT_DISCONNECTING

Indicates that a disconnect call is progressing and will disconnect soon.

Definition at line **129** of file **IxHssAcc.h**.

#define IX_HSSACC_PKT_MIN_RX_MBUF_SIZE

Minimum size of the Rx mbuf in bytes which the client must supply to the component.

Definition at line **215** of file **IxHssAcc.h**.

#define IX_HSSACC_Q_WRITE_OVERFLOW

Indicates that an attempt to Tx or to replenish an RxFree Q failed due to Q overflow.

Definition at line **137** of file **IxHssAcc.h**.

#define IX_HSSACC_RESOURCE_ERR

HssAccess function return value for a resource error.

Definition at line **121** of file **IxHssAcc.h**.

#define IX_HSSACC_RX_FRM_SYNC_ERR

HSS port Rx Frame Sync error.

Definition at line **182** of file **IxHssAcc.h**.

#define IX_HSSACC_RX_OVER_RUN_ERR

HSS port Rx over−run error.

Definition at line **189** of file **IxHssAcc.h**.

#define IX_HSSACC_TSLOTS_PER_HSS_PORT

The max number of TDM timeslots supported per HSS port − 4E1's = 32x4 = 128.

Definition at line **66** of file **IxHssAcc.h**.

#define IX_HSSACC_TX_FRM_SYNC_ERR

HSS port Tx Frame Sync error.

Definition at line **154** of file **IxHssAcc.h**.

#define IX_HSSACC_TX_OVER_RUN_ERR

HSS port Tx over−run error.

Definition at line **161** of file **IxHssAcc.h**.

Typedef Documentation

IxHssAccChanRxCallback

Prototype of the clients function to accept notification of channelised Rx.

This callback, if defined by the client in the connect, will get called in the context of an IRQ. The IRQ will be triggered when the hssSyncQMQ is not empty. The queued entry will be dequeued and this function will be executed.

Parameters:

hssPortId	**IxHssAccHssPort** − The HSS port Id. There are two identical ports (0−1).
txOffset	unsigned [in] − an offset indicating from where within the txPtrList the NPE is currently transmitting from.
rxOffset	unsigned [in] − an offset indicating where within the receive buffers the NPE has just written the received data to.
numHssErrs	unsigned [in] − This is the number of hssErrors the Npe has received

Returns:

void

Definition at line **732** of file **IxHssAcc.h**.

IxHssAccLastErrorCallback

Prototype of the clients function to accept notification of the last error.

This function is registered through the config. The client will initiate the last error retrieval. The HssAccess component will send a message to the NPE through the NPE Message Handler. When a response to the read is received, the NPE Message Handler will callback the HssAccess component which will execute this function in the same IxNpeMh context. The client will be passed the last error and the related service port (packetised 0−3, channelised 0)

Parameters:

lastHssError	unsigned [in] − The last Hss error registered that has been registered.
servicePort	unsigned [in] − This is the service port number. (packetised 0−3, channelised 0)

Returns:

void

Definition at line **641** of file **IxHssAcc.h**.

IxHssAccPktRxCallback

Prototype of the clients function to accept notification of packetised Rx.

This function is registered through the ixHssAccPktPortConnect. hssPktAcc will pass received data in the form of mbufs to the client. The mbuf passed back to the client could contain a chain of buffers, depending on the packet size received.

Parameters:

**buffer*	[in] − This is the mbuf which contains the payload received.
numHssErrs	unsigned [in] − This is the number of hssErrors the Npe has received
pktStatus	**IxHssAccPktStatus** [in] − This is the status of the mbuf that has been received.
rxUserId	**IxHssAccPktUserId** [in] − This is the client supplied value passed in at ixHssAccPktPortConnect time which is now returned to the client.

Returns:

void

Definition at line **665** of file **IxHssAcc.h**.

IxHssAccPktRxFreeLowCallback

Prototype of the clients function to accept notification of requirement of more Rx Free buffers.

The client can choose to register a callback of this type when calling a connecting. This function is registered through the ixHssAccPktPortConnect. If defined, the access layer will provide the trigger for this callback. The callback will be responsible for supplying mbufs to the access layer for use on the receive path from the HSS using ixHssPktAccFreeBufReplenish.

Returns:

void

Definition at line **684** of file **IxHssAcc.h**.

IxHssAccPktTxDoneCallback

Prototype of the clients function to accept notification of completion with Tx buffers.

This function is registered through the ixHssAccPktPortConnect. It enables the hssPktAcc to pass buffers back to the client when transmission is complete.

Parameters:

**buffer*	[in] − This is the mbuf which contained the payload that was for Tx.
numHssErrs	unsigned [in] − This is the number of hssErrors the Npe has received
pktStatus	**IxHssAccPktStatus** [in] − This is the status of the mbuf that has been transmitted.
txDoneUserId	**IxHssAccPktUserId** [in] − This is the client supplied value passed in at ixHssAccPktPortConnect time which is now returned to the client.

Returns:

void

Definition at line **706** of file **IxHssAcc.h**.

UINT32 IxHssAccPktUserId

The client supplied value which will be supplied as a parameter with a given callback.

This value will be passed into the ixHssAccPktPortConnect function once each with given callbacks. This value will then be passed back to the client as one of the parameters to each of these callbacks, when these callbacks are called.

Definition at line **618** of file **IxHssAcc.h**.

Enumeration Type Documentation

enum IxHssAcc56kEndianness

56k data endianness when using the 56k type

Enumeration values:

IX_HSSACC_56KE_BIT_7_UNUSED	High bit is unused.
IX_HSSACC_56KE_BIT_0_UNUSED	Low bit is unused.
IX_HSSACC_56KE_MAX	Delimiter for error checks.

Definition at line **437** of file **IxHssAcc.h**.

enum IxHssAcc56kSel

56k data transmission type when using the 56k type

Enumeration values:

IX_HSSACC_56KS_32_8_DATA	32/8 bit data
IX_HSSACC_56KS_56K_DATA	56K data
IX_HSSACC_56KS_MAX	Delimiter for

error checks.

Definition at line **449** of file **IxHssAcc.h**.

enum IxHssAccBitEndian

HSS Data endianness.

Enumeration values:

IX_HSSACC_LSB_ENDIAN	Tx/Rx Least Significant Bit first.
IX_HSSACC_MSB_ENDIAN	Tx/Rx Most Significant Bit first.
IX_HSSACC_ENDIAN_MAX	Delimiter for the purposes of error checks.

Definition at line **360** of file **IxHssAcc.h**.

enum IxHssAccClkDir

The HSS clock direction.

Enumeration values:

IX_HSSACC_SYNC_CLK_DIR_INPUT	Clock is an input.
IX_HSSACC_SYNC_CLK_DIR_OUTPUT	Clock is an output.
IX_HSSACC_SYNC_CLK_DIR_MAX	Delimiter for error checks.

Definition at line **310** of file **IxHssAcc.h**.

enum IxHssAccClkEdge

IxHssAccClkEdge is used to determine the clk edge to use for framing and data.

Enumeration values:

IX_HSSACC_CLK_EDGE_FALLING	Clock sampled off a falling edge.
IX_HSSACC_CLK_EDGE_RISING	Clock sampled off a rising edge.
IX_HSSACC_CLK_EDGE_MAX	Delimiter for error checks.

Definition at line **298** of file **IxHssAcc.h**.

enum IxHssAccClkSpeed

IxHssAccClkSpeed represents the HSS clock speeds available.

Enumeration values:

IX_HSSACC_CLK_SPEED_512KHZ	512KHz
IX_HSSACC_CLK_SPEED_1536KHZ	1.536MHz
IX_HSSACC_CLK_SPEED_1544KHZ	1.544MHz
IX_HSSACC_CLK_SPEED_2048KHZ	2.048MHz
IX_HSSACC_CLK_SPEED_4096KHZ	4.096MHz
IX_HSSACC_CLK_SPEED_8192KHZ	8.192MHz
IX_HSSACC_CLK_SPEED_MAX	Delimiter for error checking.

Definition at line **462** of file **IxHssAcc.h**.

enum IxHssAccDataEnable

IxHssAccDataEnable is used to determine whether or not to drive the data pins.

Enumeration values:

IX_HSSACC_DE_TRI_STATE	TRI−State the data pins.
IX_HSSACC_DE_DATA	Push data out the data pins.
IX_HSSACC_DE_MAX	Delimiter for error checks.

Definition at line **398** of file **IxHssAcc.h**.

enum IxHssAccDataPolarity

The HSS data polarity type.

Enumeration values:

IX_HSSACC_DATA_POLARITY_SAME	Don't invert data between NPE and HSS FIFOs.
IX_HSSACC_DATA_POLARITY_INVERT	Invert data between NPE and HSS FIFOs.
IX_HSSACC_DATA_POLARITY_MAX	Delimiter for error checks.

Definition at line **346** of file **IxHssAcc.h**.

enum IxHssAccDataRate

The HSS Data rate in relation to the clock.

Enumeration values:

IX_HSSACC_CLK_RATE	Data rate is at the configured clk speed.
IX_HSSACC_HALF_CLK_RATE	

	Data rate is half the configured clk speed.
IX_HSSACC_DATA_RATE_MAX	Delimiter for error checks.

Definition at line **334** of file **IxHssAcc.h**.

enum IxHssAccDrainMode

Tx pin open drain mode.

Enumeration values:

IX_HSSACC_TX_PINS_NORMAL	Normal mode.
IX_HSSACC_TX_PINS_OPEN_DRAIN	Open Drain mode.
IX_HSSACC_TX_PINS_MAX	Delimiter for error checks.

Definition at line **373** of file **IxHssAcc.h**.

enum IxHssAccFbType

IxHssAccFbType determines how to drive the Fbit.

Warning:

This will only be used for T1 @ 1.544MHz

Enumeration values:

IX_HSSACC_FB_FIFO	Fbit is dictated in FIFO.
IX_HSSACC_FB_HIGH_IMP	Fbit is high impedance.
IX_HSSACC_FB_MAX	Delimiter for error checks.

Definition at line **425** of file **IxHssAcc.h**.

enum IxHssAccFrmPulseUsage

The HSS frame pulse usage.

Enumeration values:

IX_HSSACC_FRM_PULSE_ENABLED	Generate/Receive frame pulses.
IX_HSSACC_FRM_PULSE_DISABLED	Disregard frame pulses.

IX_HSSACC_FRM_PULSE_MAX	Delimiter for error checks.

Definition at line **322** of file **IxHssAcc.h**.

enum IxHssAccFrmSyncEnable

The IxHssAccFrmSyncEnable determines how the frame sync pulse is used.

Enumeration values:

IX_HSSACC_FRM_SYNC_INPUT	Frame sync is sampled as an input.
IX_HSSACC_FRM_SYNC_INVALID_VALUE	1 is not used
IX_HSSACC_FRM_SYNC_OUTPUT_FALLING	Frame sync is an output generated off a falling clock edge.
IX_HSSACC_FRM_SYNC_OUTPUT_RISING	Frame sync is an output generated off a rising clock edge.
IX_HSSACC_FRM_SYNC_ENABLE_MAX	Delimiter for error checks.

Definition at line **281** of file **IxHssAcc.h**.

enum IxHssAccFrmSyncType

The HSS frame sync pulse type.

Enumeration values:

IX_HSSACC_FRM_SYNC_ACTIVE_LOW	Frame sync is sampled low.
IX_HSSACC_FRM_SYNC_ACTIVE_HIGH	sampled high
IX_HSSACC_FRM_SYNC_FALLINGEDGE	sampled on a falling edge
IX_HSSACC_FRM_SYNC_RISINGEDGE	sampled on a rising edge
IX_HSSACC_FRM_SYNC_TYPE_MAX	Delimiter for error checks.

Definition at line **267** of file **IxHssAcc.h**.

enum IxHssAccHdlcPort

The HDLC port ID − There are four identical HDLC ports (0−3) per HSS port and they correspond to the 4 E1/T1 trunks.

Enumeration values:

IX_HSSACC_HDLC_PORT_0	HDLC Port 0.
IX_HSSACC_HDLC_PORT_1	HDLC Port 1.
IX_HSSACC_HDLC_PORT_2	HDLC Port 2.

IX_HSSACC_HDLC_PORT_3	HDLC Port 3.
IX_HSSACC_HDLC_PORT_MAX	Delimiter for error checks.

Definition at line **239** of file **IxHssAcc.h**.

enum IxHssAccHssPort

The HSS port ID − There are two identical ports (0−1).

Enumeration values:

IX_HSSACC_HSS_PORT_0	HSS Port 0.
IX_HSSACC_HSS_PORT_1	HSS Port 1.
IX_HSSACC_HSS_PORT_MAX	Delimiter for error checks.

Definition at line **226** of file **IxHssAcc.h**.

enum IxHssAccPktCrcType

HDLC CRC type.

Enumeration values:

IX_HSSACC_PKT_16_BIT_CRC	16 bit CRC is being used
IX_HSSACC_PKT_32_BIT_CRC	32 bit CRC is being used

Definition at line **502** of file **IxHssAcc.h**.

enum IxHssAccPktHdlcIdleType

HDLC idle transmission type.

Enumeration values:

IX_HSSACC_HDLC_IDLE_ONES	idle Tx/Rx will be a succession of ones
IX_HSSACC_HDLC_IDLE_FLAGS	idle Tx/Rx will be repeated flags

Definition at line **513** of file **IxHssAcc.h**.

enum IxHssAccPktStatus

Indicates the status of packets passed to the client.

Enumeration values:

IX_HSSACC_PKT_OK	Error free.
IX_HSSACC_STOP_SHUTDOWN_ERROR	Errored due to stop or shutdown occurrance.
IX_HSSACC_HDLC_ALN_ERROR	HDLC alignment error.
IX_HSSACC_HDLC_FCS_ERROR	HDLC Frame Check Sum error.
IX_HSSACC_RXFREE_Q_EMPTY_ERROR	RxFree Q became empty while receiving this packet.
IX_HSSACC_HDLC_MAX_FRAME_SIZE_EXCEEDED	HDLC frame size received is greater than max specified at connect.
IX_HSSACC_HDLC_ABORT_ERROR	HDLC frame received is invalid due to an abort sequence received.
IX_HSSACC_DISCONNECT_IN_PROGRESS	Packet returned because a disconnect is in progress.

Definition at line **478** of file **IxHssAcc.h**.

enum IxHssAccSOFType

HSS start of frame types.

Enumeration values:

IX_HSSACC_SOF_FBIT	Framing bit transmitted and expected on Rx.
IX_HSSACC_SOF_DATA	Framing bit not transmitted nor expected on Rx.
IX_HSSACC_SOF_MAX	Delimiter for error checks.

Definition at line **385** of file **IxHssAcc.h**.

enum IxHssAccTdmSlotUsage

The HSS TDM stream timeslot assignment types.

Enumeration values:

IX_HSSACC_TDMMAP_UNASSIGNED	Unassigned.
IX_HSSACC_TDMMAP_HDLC	HDLC − packetised.
IX_HSSACC_TDMMAP_VOICE56K	Voice56K − channelised.
IX_HSSACC_TDMMAP_VOICE64K	Voice64K − channelised.
IX_HSSACC_TDMMAP_MAX	Delimiter for error checks.

Definition at line **253** of file **IxHssAcc.h**.

enum IxHssAccTxSigType

IxHssAccTxSigType is used to determine how to drive the data pins.

Enumeration values:

IX_HSSACC_TXSIG_LOW	Drive the data pins low.
IX_HSSACC_TXSIG_HIGH	Drive the data pins high.
IX_HSSACC_TXSIG_HIGH_IMP	Drive the data pins with high impedance.
IX_HSSACC_TXSIG_MAX	Delimiter for error checks.

Definition at line **410** of file **IxHssAcc.h**.

Function Documentation

```
IX_STATUS ixHssAccChanConnect ( IxHssAccHssPort          hssPortId,
                                unsigned                 bytesPerTSTrigger,
                                UINT8 *                  rxCircular,
                                unsigned                 numRxBytesPerTS,
                                UINT32 *                 txPtrList,
                                unsigned                 numTxPtrLists,
                                unsigned                 numTxBytesPerBlk,
                                IxHssAccChanRxCallback   rxCallback
                              )
```

This function allows the client to connect to the Tx/Rx NPE Channelised Service. There can only be one client per HSS port. The client is responsible for ensuring that the HSS port is configured appropriately before its connect request. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
bytesPerTSTrigger	unsigned [in] − The NPE will trigger the access component after bytesPerTSTrigger have been received for all trunk timeslots. This figure is a multiple of 8 e.g. 8 for 1ms trigger, 16 for 2ms trigger.
**rxCircular*	UINT8 [in] − A pointer to memory allocated by the client to be filled by data received. The buffer at this address is part of a pool of buffers to be accessed in a circular fashion. This address will be written to by the NPE. Therefore, it needs to be a physical address.
numRxBytesPerTS	unsigned [in] − The number of bytes allocated per timeslot within the receive memory. This figure will depend on the latency of the system. It needs to be deep enough for data to be read by the client before the NPE re−writes over that memory e.g. if the client samples at a rate of 40bytes per timeslot, numRxBytesPerTS may need to be 40bytes * 3. This would give the client 3 * 5ms of time before received data is over−written.

txPtrList UINT32 [in] − The address of an area of contiguous memory allocated by the client to be populated with pointers to data for transmission. Each pointer list contains a pointer per active channel. The txPtrs will point to data to be transmitted by the NPE. Therefore, they must point to physical addresses.

numTxPtrLists unsigned [in] − The number of pointer lists in txPtrList. This figure is dependent on jitter.

numTxBytesPerBlk unsigned [in] − The size of the Tx data, in bytes, that each pointer within the PtrList points to.

rxCallback **IxHssAccChanRxCallback** [in] − A client function pointer to be called back to handle the actual Tx/Rx of channelised data. If this is not NULL, an ISR will call this function. If this pointer is NULL, it implies that the client will use a polling mechanism to detect when the Tx and Rx of channelised data is to occur. The client will use hssChanAccStatus for this.

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

IX_STATUS ixHssAccChanDisconnect (**IxHssAccHssPort** *hssPortId*)

This function allows the client to Disconnect from a channelised service. If the NPE Rx Flow has not been disabled, the disconnect will disable it before proceeding with other disconnect functionality. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

IX_STATUS ixHssAccChanPortDisable (**IxHssAccHssPort** *hssPortId*)

This function is responsible for disabling a channelised service for the specified HSS port. It disables the NPE Rx flow. The client must have already connected to and enabled a channelised service for the specified HSS port. This disable interface can be called before a disconnect, but is not required to. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

IX_STATUS ixHssAccChanPortEnable (**IxHssAccHssPort** *hssPortId*)

This function is responsible for enabling a channelised service for the specified HSS port. It enables the NPE Rx flow. The client must have already connected to a channelised service before enabling the channelised service. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccChanStatusQuery ( IxHssAccHssPort hssPortId,
                                    BOOL *          dataRecvd,
                                    unsigned *      rxOffset,
                                    unsigned *      txOffset,
                                    unsigned *      numHssErrs
                                  )
```

This function is called by the client to query whether or not channelised data has been received. If there is, hssChanAcc will return the details in the output parameters. An enabled connection should exist on the specified hssPortId before attempting to call this interface. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

**dataRecvd* BOOL [out] − This BOOL indicates to the client whether or not the access component has read any data for the client. If FALSE, the other output parameters will not have been written to.

**rxOffset* unsigned [out] − An offset to indicate to the client where within the receive buffers the NPE has just written the received data to.

**txOffset* unsigned [out] − An offset to indicate to the client from where within the txPtrList the NPE is currently transmitting from

**numHssErrs* unsigned [out] − The total number of HSS port errors since initial port configuration

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully

◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccChanTslotSwitchDisable ( IxHssAccHssPort hssPortId,
                                           UINT32          tsSwitchHandle
                                         )
```

This function is responsible for disabling timeslot switching (bypass) channel, that is associated with tsSwitchHandle, for the specified HSS port. The bypass channel to be disabled must have already been enabled for the specified HSS port. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

IxHssAccHssPort hssPortId (in) − The HSS port Id. There are two identical ports (0−1). Only port 0 will be supported.

UINT32 tsSwitchHandle (in) − The handle that hooks to the bypass channel. This handle is the parameter returned to client by ixHssAccChanTslotSwitchEnable during timeslot switching channel enabling operation.

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccChanTslotSwitchEnable ( IxHssAccHssPort hssPortId,
                                          UINT32          srcTimeslot,
                                          UINT32          destTimeslot,
                                          UINT32 *        tsSwitchHandle
                                        )
```

This function is responsible for enabling timeslot switching (bypass) channel between two voice channels for the specified HSS port. The voice channels must have already been configured as channelised timeslot for the specified HSS port. In current release, only up to two timeslot switching channels can be enabled at any one time. In order to minimize bypass delay and ensure better voice quality, this function requires at least 8 TDM timeslots on the specified HSS port to be setup as channelised timeslots. In timeslot switching mode, data received on srcTimeslot is transmitted onto a partner timeslot (i.e. destTimeslot) at NPE level. A copy of the received data on srcTimeslot is also sent to client via HssAccess component. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

IxHssAccHssPort hssPortId (in) − The HSS port Id. There are two identical ports (0−1). Only port 0 will be supported.

UINT32 srcTimeslot (in) − The voice channel Id whose its receive side will be used in the bypass (0−127).

UINT32 destTimeslot (in) − The voice channel Id whose its transmit side will be used in the bypass (0−127).

UINT32

*tsSwitchHandle (out) − The handle returned to client, that hooks to the bypass channel established between srcTimeslot and destTimeslot. This handle will be the mean by which client disables or downloads gain control table to NPE for the bypass channel that associates with this handle. Client must ignore the value returned through this handle if bypass channel fails to setup.

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccChanTslotSwitchGctDownload ( IxHssAccHssPort hssPortId,
                                               UINT8 *        gainCtrlTable,
                                               UINT32         tsSwitchHandle
                                             )
```

This function is responsible for downloading a gain control table (256 bytes) to NPE for the specified timeslot switching (bypass) channel, associated with tsSwitchHandle, on the specified HSS port. The bypass voice channel must have already been enabled for the specified HSS port before this API can be called to download the gain control table to NPE. No other HssAccChannelised interface should be called while this interface is being processed.

Parameters:

IxHssAccHssPort hssPortId (in) − The HSS port Id. There are two identical ports (0−1). Only port 0 will be supported.

UINT8 *gainCtrlTable (in) − A pointer to an array of size IX_HSSACC_ENTRIES_PER_GAIN_CTRL_TABLE, defining each entry for a gain control table for the specified bypass voice channel.

UINT32 tsSwitchHandle (in) − The handle that hooks to the bypass channel. This handle is the parameter returned to client by ixHssAccChanTslotSwitchEnable during timeslot switching channel enabling operation.

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccInit ( void )
```

This function is responsible for initialising resources for use by the packetised and channelised clients. It should be called after HSS NPE image has been downloaded into NPE−A and before any other HssAccess interface is called. No other HssAccPacketised interface should be called while this interface is being processed.

Returns:

- ◊ IX_SUCCESS The function executed successfully

◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_RESOURCE_ERR The function did not execute successfully due to a resource error

IX_STATUS ixHssAccLastErrorRetrievalInitiate (**IxHssAccHssPort** *hssPortId*)

Initiate the retrieval of the last HSS error. The HSS port should be configured before attempting to call this interface.

Parameters:

hssPortId **IxHssAccHssPort** [in] − the HSS port ID

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
ixHssAccPktPortConnect ( IxHssAccHssPort                hssPortId,
                         IxHssAccHdlcPort               hdlcPortId,
                         BOOL                           hdlcFraming,
                         IxHssAccHdlcMode               hdlcMode,
                         BOOL                           hdlcBitInvert,
                         unsigned                       blockSizeInWords,
                         UINT32                         rawIdleBlockPattern,
                         IxHssAccPktHdlcFraming         hdlcTxFraming,
                         IxHssAccPktHdlcFraming         hdlcRxFraming,
                         unsigned                       frmFlagStart,
                         IxHssAccPktRxCallback          rxCallback,
                         IxHssAccPktUserId              rxUserId,
                         IxHssAccPktRxFreeLowCallback   rxFreeLowCallback,
                         IxHssAccPktUserId              rxFreeLowUserId,
                         IxHssAccPktTxDoneCallback      txDoneCallback,
                         IxHssAccPktUserId              txDoneUserId
                       )
```

This function is responsible for connecting a client to one of the 4 available HDLC ports. The HSS port should be configured before attempting a connect. No other HssAccPacketised interface should be called while this connect is being processed.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
hdlcPortId	**IxHssAccHdlcPort** [in] − This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3
hdlcFraming	BOOL [in] − This value determines whether the service will use HDLC data or the debug, raw data type i.e. no HDLC processing
hdlcMode	

	IxHssAccHdlcMode [in] − This structure contains 56Kbps, HDLC−mode configuration parameters
hdlcBitInvert	BOOL [in] − This value determines whether bit inversion will occur between HDLC and HSS co−processors i.e. post−HDLC processing for transmit and pre−HDLC processing for receive, for the specified HDLC Termination Point
blockSizeInWords	unsigned [in] − The max Tx/Rx block size
rawIdleBlockPattern	UINT32 [in] − Tx idle pattern in raw mode
hdlcTxFraming	**IxHssAccPktHdlcFraming** [in] − This structure contains the following information required by the NPE to configure the HDLC co−processor for Tx
hdlcRxFraming	**IxHssAccPktHdlcFraming** [in] − This structure contains the following information required by the NPE to configure the HDLC co−processor for Rx
frmFlagStart	unsigned − Number of flags to precede to transmitted flags (0−2).
rxCallback	**IxHssAccPktRxCallback** [in] − Pointer to the clients packet receive function.
rxUserId	**IxHssAccPktUserId** [in] − The client supplied Rx value to be passed back as an argument to the supplied rxCallback
rxFreeLowCallback	**IxHssAccPktRxFreeLowCallback** [in] − Pointer to the clients Rx free buffer request function. If NULL, assume client will trigger independently.
rxFreeLowUserId	**IxHssAccPktUserId** [in] − The client supplied RxFreeLow value to be passed back as an argument to the supplied rxFreeLowCallback
txDoneCallback	**IxHssAccPktTxDoneCallback** [in] − Pointer to the clients Tx done callback function
txDoneUserId	**IxHssAccPktUserId** [in] − The client supplied txDone value to be passed back as an argument to the supplied txDoneCallback

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error
- ◊ IX_HSSACC_RESOURCE_ERR The function did not execute successfully due to a resource error

```
IX_STATUS ixHssAccPktPortDisable ( IxHssAccHssPort   hssPortId,
                                   IxHssAccHdlcPort  hdlcPortId
                                 )
```

This function is responsible for disabling a packetised service for the specified HSS/HDLC port combination. It disables the Rx flow. The client must have already connected to and enabled a packetised service for the specified HDLC port. This disable interface can be called before a disconnect, but is not required to.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
hdlcPortId	**IxHssAccHdlcPort** [in] − The port id (0,1,2,3) to disable the service on.

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully

◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccPktPortDisconnect ( IxHssAccHssPort   hssPortId,
                                      IxHssAccHdlcPort  hdlcPortId
                                    )
```

This function is responsible for disconnecting a client from one of the 4 available HDLC ports. It is not required that the Rx Flow has been disabled before calling this function. If the Rx Flow has not been disabled, the disconnect will disable it before proceeding with the disconnect. No other HssAccPacketised interface should be called while this interface is being processed.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
hdlcPortId	**IxHssAccHdlcPort** [in] − This is the number of the HDLC port to disconnect and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PKT_DISCONNECTING The function has initiated the disconnecting procedure but it has not completed yet.

```
IX_STATUS ixHssAccPktPortEnable ( IxHssAccHssPort   hssPortId,
                                  IxHssAccHdlcPort  hdlcPortId
                                )
```

This function is responsible for enabling a packetised service for the specified HSS/HDLC port combination. It enables the Rx flow. The client must have already connected to a packetised service and is responsible for ensuring an adequate amount of Rx mbufs have been supplied to the access component before enabling the packetised service. This function must be called on a given port before any call to ixHssAccPktPortTx on the same port. No other HssAccPacketised interface should be called while this interface is being processed.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
hdlcPortId	**IxHssAccHdlcPort** [in] − The port id (0,1,2,3) to enable the service on.

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully
◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

```
BOOL ixHssAccPktPortIsDisconnectComplete ( IxHssAccHssPort   hssPortId,
                                           IxHssAccHdlcPort hdlcPortId
                                         )
```

This function is called to check if a given HSS/HDLC port combination is in a connected state or not. This function may be called at any time to determine a ports state. No other HssAccPacketised interface should be called while this interface is being processed.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

hdlcPortId **IxHssAccHdlcPort** [in] − This is the number of the HDLC port to disconnect and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3

Returns:

- ◊ TRUE The state of this HSS/HDLC port combination is disconnected, so if a disconnect was called, it is now completed.
- ◊ FALSE The state of this HSS/HDLC port combination is connected, so if a disconnect was called, it is not yet completed.

```
IX_STATUS ixHssAccPktPortRxFreeReplenish ( IxHssAccHssPort   hssPortId,
                                           IxHssAccHdlcPort hdlcPortId,
                                           IX_OSAL_MBUF *   buffer
                                         )
```

Function which the client calls at regular intervals to provide mbufs to the access component for Rx. A connection should exist for the specified hssPortId/hdlcPortId combination before attempting to call this interface. Also, the connection should not be in a disconnecting state.

Parameters:

hssPortId **IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).

hdlcPortId **IxHssAccHdlcPort** [in] − This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3

**buffer* [in] − A pointer to a free mbuf to filled with payload.

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error
- ◊ IX_HSSACC_RESOURCE_ERR The function did not execute successfully due to a resource error
- ◊ IX_HSSACC_Q_WRITE_OVERFLOW The function did not succeed due to a Q overflow

```
IX_STATUS ixHssAccPktPortTx ( IxHssAccHssPort   hssPortId,
                              IxHssAccHdlcPort hdlcPortId,
                              IX_OSAL_MBUF *   buffer
                            )
```

Function which the client calls when it wants to transmit packetised data. An enabled connection should exist on the specified hssPortId/hdlcPortId combination before attempting to call this interface. No other HssAccPacketised interface should be called while this interface is being processed.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
hdlcPortId	**IxHssAccHdlcPort** [in] − This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3
**buffer*	[in] − A pointer to a chain of mbufs which the client has filled with the payload

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully
- ◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error
- ◊ IX_HSSACC_RESOURCE_ERR The function did not execute successfully due to a resource error. See note.
- ◊ IX_HSSACC_Q_WRITE_OVERFLOW The function did not succeed due to a Q overflow

Note:

IX_HSSACC_RESOURCE_ERR is returned when a free descriptor cannot be obtained to send the chain of mbufs to the NPE. This is a normal scenario. HssAcc has a pool of descriptors and this error means that they are currently all in use. The recommended approach to this is to retry until a descriptor becomes free and the packet is successfully transmitted. Alternatively, the user could wait until the next IxHssAccPktTxDoneCallback callback is triggered, and then retry, as it is this event that causes a transmit descriptor to be freed.

```
IX_STATUS ixHssAccPortInit ( IxHssAccHssPort              hssPortId,
                             IxHssAccConfigParams *       configParams,
                             IxHssAccTdmSlotUsage *       tdmMap,
                             IxHssAccLastErrorCallback    lastHssErrorCallback
                           )
```

Initialise a HSS port. No channelised or packetised connections should exist in the HssAccess layer while this interface is being called.

Parameters:

hssPortId	**IxHssAccHssPort** [in] − The HSS port Id. There are two identical ports (0−1).
**configParams*	**IxHssAccConfigParams** [in] − A pointer to the HSS configuration structure
**tdmMap*	**IxHssAccTdmSlotUsage** [in] − A pointer to an array of size IX_HSSACC_TSLOTS_PER_HSS_PORT, defining the slot usage over the HSS port
lastHssErrorCallback	**IxHssAccLastErrorCallback** [in] − Client callback to report last error

Returns:

- ◊ IX_SUCCESS The function executed successfully
- ◊ IX_FAIL The function did not execute successfully

◊ IX_HSSACC_PARAM_ERR The function did not execute successfully due to a parameter error

void ixHssAccShow (void)

This function will display the current state of the IxHssAcc component. The output is sent to stdout.

Returns:

void

void ixHssAccStatsInit (void)

This function will reset the IxHssAcc statistics.

Returns:

void

IX_STATUS ixHssAccUninit (void)

This function is responsible for un−initialize of resources for use by the packetised and channelised clients. It internally calls the uninitialize functions of sub components. This function should be the last function to be called before exiting HssAcc component.

Returns:

◊ IX_SUCCESS The function executed successfully
◊ IX_FAIL The function did not execute successfully

Intel (R) IXP400 Software I2C Driver(IxI2cDrv) API

IXP400 I2C Driver Public API.

Data Structures

struct **IxI2cInitVars**
contains all the variables required to initialize the I2C unit

struct **IxI2cStatsCounters**
contains results of counters and their overflow

Defines

#define **IX_I2C_US_POLL_FOR_XFER_STATUS**
The interval of micro/mili seconds the Intel (R) IXP4XX Product Line of Network Processors will wait before it polls for status from the ixI2cIntrXferStatus; Every 20us is 1 byte @ 400Kbps and 4 bytes @ 100Kbps. This is dependent on delay type selected through the API ixI2cDrvDelayTypeSelect.

#define **IX_I2C_NUM_OF_TRIES_TO_CALL_CALLBACK_FUNC**
The number of tries that will be attempted to call a callback function if the callback does not or is unable to resolve the issue it is called to resolve.

#define **IX_I2C_NUM_TO_POLL_IDBR_RX_FULL**
Number of tries slave will poll the IDBR Rx full bit before it gives up.

#define **IX_I2C_NUM_TO_POLL_IDBR_TX_EMPTY**
Number of tries slave will poll the IDBR Tx empty bit before it gives up.

Typedefs

typedef void(* **IxI2cMasterReadCallbackP**)(**IxI2cMasterStatus**, **IxI2cXferMode**, char *, UINT32)
The pointer to the function that will be called when the master has completed its receive. The parameter that is passed will provide the status of the read (success, arb loss, or bus error), the transfer mode (normal or repeated start, the buffer pointer and number of bytes transferred.

typedef void(* **IxI2cMasterWriteCallbackP**)(**IxI2cMasterStatus**, **IxI2cXferMode**, char *, UINT32)
The pointer to the function that will be called when the master has completed its transmit. The parameter that is passed will provide the status of the write (success, arb loss, or buss error), the transfer mode (normal or repeated start), the buffer pointer and number of bytes transferred.

typedef void(* **IxI2cSlaveReadCallbackP**)(**IX_I2C_STATUS**, char *, UINT32, UINT32)

The pointer to the function that will be called when a slave address detected in interrupt mode for a read. The parameters that is passed will provide the read status, buffer pointer, buffer size, and the bytes received. When a start of a read is initiated there will be no buffer allocated and this callback will be called to request for a buffer. While receiving, if the buffer gets filled, this callback will be called to request for a new buffer while sending the filled buffer's pointer and size, and data size received. When the receive is complete, this callback will be called to process the data and free the memory by passing the buffer's pointer and size, and data size received.

typedef void(* **IxI2cSlaveWriteCallbackP**)(**IX_I2C_STATUS**, char *, UINT32, UINT32)
The pointer to the function that will be called when a slave address detected in interrupt mode for a write. The parameters that is passed will provide the write status, buffer pointer, buffer size, and the bytes received. When a start of a write is initiated there will be no buffer allocated and this callback will be called to request for a buffer and to fill it with data. While transmitting, if the data in the buffer empties, this callback will be called to request for more data to be filled in the same or new buffer. When the transmit is complete, this callback will be called to free the memory or other actions to be taken.

typedef void(* **IxI2cGenCallCallbackP**)(**IX_I2C_STATUS**, char *, UINT32, UINT32)
The pointer to the function that will be called when a general call detected in interrupt mode for a read. The parameters that is passed will provide the read status, buffer pointer, buffer size, and the bytes received. When a start of a read is initiated there will be no buffer allocated and this callback will be called to request for a buffer. While receiving, if the buffer gets filled, this callback will be called to request for a new buffer while sending the filled buffer's pointer and size, and data size received. When the receive is complete, this callback will be called to process the data and free the memory by passing the buffer's pointer and size, and data size received.

Enumerations

enum **IxI2cMasterStatus** {
IX_I2C_MASTER_XFER_COMPLETE,
IX_I2C_MASTER_XFER_BUS_ERROR,
IX_I2C_MASTER_XFER_ARB_LOSS
}
The master status − transfer complete, bus error or arbitration loss.

enum **IX_I2C_STATUS** {
IX_I2C_SUCCESS,
IX_I2C_FAIL,
IX_I2C_NOT_SUPPORTED,
IX_I2C_NULL_POINTER,
IX_I2C_INVALID_SPEED_MODE_ENUM_VALUE,
IX_I2C_INVALID_FLOW_MODE_ENUM_VALUE,
IX_I2C_SLAVE_ADDR_CB_MISSING,
IX_I2C_GEN_CALL_CB_MISSING,
IX_I2C_INVALID_SLAVE_ADDR,
IX_I2C_INT_BIND_FAIL,
IX_I2C_INT_UNBIND_FAIL,

IX_I2C_NOT_INIT,
IX_I2C_MASTER_BUS_BUSY,
IX_I2C_MASTER_ARB_LOSS,
IX_I2C_MASTER_XFER_ERROR,
IX_I2C_MASTER_BUS_ERROR,
IX_I2C_MASTER_NO_BUFFER,
IX_I2C_MASTER_INVALID_XFER_MODE,
IX_I2C_SLAVE_ADDR_NOT_DETECTED,
IX_I2C_GEN_CALL_ADDR_DETECTED,
IX_I2C_SLAVE_READ_DETECTED,
IX_I2C_SLAVE_WRITE_DETECTED,
IX_I2C_SLAVE_NO_BUFFER,
IX_I2C_DATA_SIZE_ZERO,
IX_I2C_SLAVE_WRITE_BUFFER_EMPTY,
IX_I2C_SLAVE_WRITE_ERROR,
IX_I2C_SLAVE_OR_GEN_READ_BUFFER_FULL,
IX_I2C_SLAVE_OR_GEN_READ_ERROR
}
The status that can be returned in a I2C driver initialization.

enum **IxI2cSpeedMode** {
IX_I2C_NORMAL_MODE,
IX_I2C_FAST_MODE
}
Type of speed modes supported by the I2C hardware.

enum **IxI2cXferMode** {
IX_I2C_NORMAL,
IX_I2C_REPEATED_START
}
Used for indicating it is a repeated start or normal transfer.

enum **IxI2cFlowMode** {
IX_I2C_POLL_MODE,
IX_I2C_INTERRUPT_MODE
}
Used for indicating it is a poll or interrupt mode.

enum **IxI2cDelayMode** {
IX_I2C_LOOP_DELAY,
IX_I2C_SCHED_DELAY
}
Used for selecting looping delay or OS scheduler delay.

Functions

PUBLIC
IX_I2C_STATUS ixI2cDrvInit (**IxI2cInitVars** *InitVarsSelected)

Initializes the I2C Driver.

PUBLIC
IX_I2C_STATUS ixI2cDrvUninit (void)
Disables the I2C hardware.

PUBLIC
IX_I2C_STATUS ixI2cDrvSlaveAddrSet (UINT8 SlaveAddrSet)
Sets the I2C Slave Address.

PUBLIC
IX_I2C_STATUS ixI2cDrvBusScan (void)
scans the I2C bus for slave devices

PUBLIC**ixI2cDrvWriteTransfer** (UINT8 SlaveAddr, char *bufP, UINT32 dataSize,
IX_I2C_STATUS IxI2cXferMode XferModeSelect)
PUBLIC**ixI2cDrvReadTransfer** (UINT8 SlaveAddr, char *bufP, UINT32 dataSize,
IX_I2C_STATUS IxI2cXferMode XferModeSelect)
Initiates a transfer to receive bytes of data from a slave device through the I2C bus.

PUBLIC
IX_I2C_STATUS ixI2cDrvSlaveAddrAndGenCallDetectedCheck (void)
Checks the I2C Status Register to determine if a slave address is detected.

PUBLIC**ixI2cDrvSlaveOrGenDataReceive** (char *bufP, const UINT32 bufSize, UINT32
IX_I2C_STATUS *dataSizeRcvd)
Performs the slave receive or general call receive data transfer.

PUBLIC**ixI2cDrvSlaveDataTransmit** (char *bufP, const UINT32 dataSize, UINT32
IX_I2C_STATUS *dataSizeXmtd)
Performs the slave write data transfer.

PUBLIC void **ixI2cDrvSlaveOrGenCallBufReplenish** (char *bufP, UINT32 bufSize)
Replenishes the buffer which stores buffer info for both slave and general call.

PUBLIC
IX_I2C_STATUS ixI2cDrvStatsGet (**IxI2cStatsCounters** *I2cStats)
Returns the I2C Statistics through the pointer passed in.

PUBLIC void **ixI2cDrvStatsReset** (void)
Reset I2C statistics counters.

PUBLIC
IX_I2C_STATUS ixI2cDrvShow (void)
Displays the I2C status register and the statistics counter.

PUBLIC void **ixI2cDrvDelayTypeSelect** (**IxI2cDelayMode** delayTypeSelect)
Sets the delay type of either looping delay or OS scheduler delay according to the argument provided.

Detailed Description

IXP400 I2C Driver Public API.

Define Documentation

#define IX_I2C_NUM_OF_TRIES_TO_CALL_CALLBACK_FUNC

The number of tries that will be attempted to call a callback function if the callback does not or is unable to resolve the issue it is called to resolve.

Definition at line **44** of file **IxI2cDrv.h**.

#define IX_I2C_NUM_TO_POLL_IDBR_RX_FULL

Number of tries slave will poll the IDBR Rx full bit before it gives up.

Definition at line **52** of file **IxI2cDrv.h**.

#define IX_I2C_NUM_TO_POLL_IDBR_TX_EMPTY

Number of tries slave will poll the IDBR Tx empty bit before it gives up.

Definition at line **59** of file **IxI2cDrv.h**.

#define IX_I2C_US_POLL_FOR_XFER_STATUS

The interval of micro/mili seconds the Intel (R) IXP4XX Product Line of Network Processors will wait before it polls for status from the ixI2cIntrXferStatus; Every 20us is 1 byte @ 400Kbps and 4 bytes @ 100Kbps. This is dependent on delay type selected through the API ixI2cDrvDelayTypeSelect.

Definition at line **36** of file **IxI2cDrv.h**.

Typedef Documentation

typedef void(* IxI2cGenCallCallbackP)(**IX_I2C_STATUS**, char*, UINT32, UINT32)

The pointer to the function that will be called when a general call detected in interrupt mode for a read. The parameters that is passed will provide the read status, buffer pointer, buffer size, and the bytes received. When a start of a read is initiated there will be no buffer allocated and this callback will be called to request

for a buffer. While receiving, if the buffer gets filled, this callback will be called to request for a new buffer while sending the filled buffer's pointer and size, and data size received. When the receive is complete, this callback will be called to process the data and free the memory by passing the buffer's pointer and size, and data size received.

Definition at line **242** of file **IxI2cDrv.h**.

typedef void(* IxI2cMasterReadCallbackP)(**IxI2cMasterStatus**, **IxI2cXferMode**, char*, UINT32)

The pointer to the function that will be called when the master has completed its receive. The parameter that is passed will provide the status of the read (success, arb loss, or bus error), the transfer mode (normal or repeated start, the buffer pointer and number of bytes transferred.

Definition at line **180** of file **IxI2cDrv.h**.

typedef void(* IxI2cMasterWriteCallbackP)(**IxI2cMasterStatus**, **IxI2cXferMode**, char*, UINT32)

The pointer to the function that will be called when the master has completed its transmit. The parameter that is passed will provide the status of the write (success, arb loss, or buss error), the transfer mode (normal or repeated start), the buffer pointer and number of bytes transferred.

Definition at line **191** of file **IxI2cDrv.h**.

typedef void(* IxI2cSlaveReadCallbackP)(**IX_I2C_STATUS**, char*, UINT32, UINT32)

The pointer to the function that will be called when a slave address detected in interrupt mode for a read. The parameters that is passed will provide the read status, buffer pointer, buffer size, and the bytes received. When a start of a read is initiated there will be no buffer allocated and this callback will be called to request for a buffer. While receiving, if the buffer gets filled, this callback will be called to request for a new buffer while sending the filled buffer's pointer and size, and data size received. When the receive is complete, this callback will be called to process the data and free the memory by passing the buffer's pointer and size, and data size received.

Definition at line **208** of file **IxI2cDrv.h**.

typedef void(* IxI2cSlaveWriteCallbackP)(**IX_I2C_STATUS**, char*, UINT32, UINT32)

The pointer to the function that will be called when a slave address detected in interrupt mode for a write. The parameters that is passed will provide the write status, buffer pointer, buffer size, and the bytes received. When a start of a write is initiated there will be no buffer allocated and this callback will be called to request for a buffer and to fill it with data. While transmitting, if the data in the buffer empties, this callback will be called to request for more data to be filled in the same or new buffer. When the transmit is complete, this callback will be called to free the memory or other actions to be taken.

Definition at line **225** of file **IxI2cDrv.h**.

Enumeration Type Documentation

enum IX_I2C_STATUS

The status that can be returned in a I2C driver initialization.

Enumeration values:

IX_I2C_SUCCESS	Success status.
IX_I2C_FAIL	Fail status.
IX_I2C_NOT_SUPPORTED	hardware does not have dedicated I2C hardware
IX_I2C_NULL_POINTER	parameter passed in is NULL
IX_I2C_INVALID_SPEED_MODE_ENUM_VALUE	speed mode selected is invalid
IX_I2C_INVALID_FLOW_MODE_ENUM_VALUE	flow mode selected is invalid
IX_I2C_SLAVE_ADDR_CB_MISSING	slave callback is NULL
IX_I2C_GEN_CALL_CB_MISSING	general callback is NULL
IX_I2C_INVALID_SLAVE_ADDR	invalid slave address specified
IX_I2C_INT_BIND_FAIL	interrupt bind fail
IX_I2C_INT_UNBIND_FAIL	interrupt unbind fail
IX_I2C_NOT_INIT	I2C is not initialized yet.
IX_I2C_MASTER_BUS_BUSY	master detected a I2C bus busy
IX_I2C_MASTER_ARB_LOSS	master experienced arbitration loss
IX_I2C_MASTER_XFER_ERROR	master experienced a transfer error
IX_I2C_MASTER_BUS_ERROR	master detected a I2C bus error
IX_I2C_MASTER_NO_BUFFER	no buffer provided for master transfer
IX_I2C_MASTER_INVALID_XFER_MODE	xfer mode selected is invalid
IX_I2C_SLAVE_ADDR_NOT_DETECTED	polled slave addr not detected
IX_I2C_GEN_CALL_ADDR_DETECTED	polling detected general call
IX_I2C_SLAVE_READ_DETECTED	polling detected slave read request
IX_I2C_SLAVE_WRITE_DETECTED	polling detected slave

	write request
IX_I2C_SLAVE_NO_BUFFER	no buffer provided for slave transfers
IX_I2C_DATA_SIZE_ZERO	data size transfer is zero − invalid
IX_I2C_SLAVE_WRITE_BUFFER_EMPTY	slave buffer is used till empty
IX_I2C_SLAVE_WRITE_ERROR	slave write experienced an error
IX_I2C_SLAVE_OR_GEN_READ_BUFFER_FULL	slave buffer is filled up
IX_I2C_SLAVE_OR_GEN_READ_ERROR	slave read experienced an error

Definition at line **87** of file **IxI2cDrv.h**.

enum IxI2cDelayMode

Used for selecting looping delay or OS scheduler delay.

Enumeration values:

IX_I2C_LOOP_DELAY	delay in microseconds
IX_I2C_SCHED_DELAY	delay in miliseconds

Definition at line **165** of file **IxI2cDrv.h**.

enum IxI2cFlowMode

Used for indicating it is a poll or interrupt mode.

Definition at line **152** of file **IxI2cDrv.h**.

enum IxI2cMasterStatus

The master status − transfer complete, bus error or arbitration loss.

Definition at line **72** of file **IxI2cDrv.h**.

enum IxI2cSpeedMode

Type of speed modes supported by the I2C hardware.

Definition at line **126** of file **IxI2cDrv.h**.

enum IxI2cXferMode

Used for indicating it is a repeated start or normal transfer.

Definition at line **139** of file **IxI2cDrv.h**.

Function Documentation

ixI2cDrvBusScan (void)

scans the I2C bus for slave devices

Parameters:

- None

Global Data :

- None.

This API will prompt all slave addresses for a reply except its own

Returns:

- ◊ IX_I2C_SUCCESS − found at least one slave device
- ◊ IX_I2C_FAIL − Fail to find even one slave device
- ◊ IX_I2C_BUS_BUSY − The I2C bus is busy (held by another I2C master)
- ◊ IX_I2C_ARB_LOSS − The I2C bus was loss to another I2C master
- ◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : yes
- ISR Callable : yes

ixI2cDrvDelayTypeSelect (**IxI2cDelayMode** *delayTypeSelect*)

Sets the delay type of either looping delay or OS scheduler delay according to the argument provided.

Parameters:

- "IxI2cDelayMode [in] delayTypeSelect" − the I2C delay type selected

Global Data :

- None.

This API will set the delay type used by the I2C Driver to either looping delay or OS scheduler delay.

Returns:

◊ None

- Reentrant : yes
- ISR Callable : no

ixI2cDrvInit (**IxI2cInitVars** * *InitVarsSelected*)

Initializes the I2C Driver.

Parameters:

*IxI2cInitVars [in] *InitVarsSelected* − struct containing required variables for initialization

Global Data :

- None.

This API will check if the hardware supports this I2C driver and the validity of the parameters passed in. It will continue to process the parameters passed in by setting the speed of the I2C unit (100kbps or 400kbps), setting the flow to either interrupt or poll mode, setting the address of the I2C unit, enabling/disabling the respond to General Calls, enabling/disabling the respond to Slave Address and SCL line driving. If it is interrupt mode, then it will register the callback routines for master, slavetransfer and general call receive.

Returns:

◊ IX_I2C_SUCCESS − Successfully initialize and enable the I2C hardware.
◊ IX_I2C_NOT_SUPPORTED − The hardware does not support or have a dedicated I2C unit to support this driver
◊ IX_I2C_NULL_POINTER − The parameter passed in is a NULL pointed
◊ IX_I2C_INVALID_SPEED_MODE_ENUM_VALUE − The speed mode selected in the InitVarsSelected is invalid
◊ IX_I2C_INVALID_FLOW_MODE_ENUM_VALUE − The flow mode selected in the InitVarsSelected is invalid
◊ IX_I2C_INVALID_SLAVE_ADDR − The address 0x0 is reserved for general call.
◊ IX_I2C_SLAVE_ADDR_CB_MISSING − interrupt mode is selected but slave address callback pointer is NULL
◊ IX_I2C_GEN_CALL_CB_MISSING − interrupt mode is selected but general call callback pointer is NULL
◊ IX_I2C_INT_BIND_FAIL − The ISR for the I2C failed to bind
◊ IX_I2C_INT_UNBIND_FAIL − The ISR for the I2C failed to unbind

- Reentrant : yes
- ISR Callable : yes

```
ixI2cDrvReadTransfer ( UINT8           SlaveAddr,
                       char *          bufP,
                       UINT32          dataSize,
                       IxI2cXferMode   XferModeSelect
                     )
```

Initiates a transfer to receive bytes of data from a slave device through the I2C bus.

Parameters:

UINT8 [in] SlaveAddr	− The slave address to request data from.
*char [out] *bufP*	− The pointer to the buffer to store the requested data.
UINT32 [in] dataSize	− The number of bytes requested.
IxI2cXferMode [in] XferModeSelect	− the transfer mode selected, either repeated start (ends w/o stop) or normal (start and stop)

Global Data :

- None.

This API will try to obtain master control of the I2C bus and receive the number of bytes, specified by dataSize, from the user specified address into the receive buffer. It will use either interrupt or poll mode depending on the mode selected.

If in interrupt mode and IxI2cMasterReadCallbackP is not NULL, the driver will initiate the transfer and return immediately. The function pointed to by IxI2cMasterReadCallbackP will be called in the interrupt service handlers when the operation is complete.

If in interrupt mode and IxI2cMasterReadCallbackP is NULL, then the driver will wait for the operation to complete, and then return.

And if the repeated start transfer mode is selected, then it will not send a stop signal at the end of all the transfers. *NOTE*: If repeated start transfer mode is selected, it has to end with a normal mode transfer mode else the bus will continue to be held by the Intel (R) IXP4XX Product Line of Network Processors.

Returns:

◊ IX_I2C_SUCCESS − Successfuuly read slave data
◊ IX_I2C_MASTER_BUS_BUSY − The I2C bus is busy (held by another I2C master)
◊ IX_I2C_MASTER_ARB_LOSS − The I2C bus was loss to another I2C master
◊ IX_I2C_MASTER_XFER_ERROR − There was a bus error during transfer
◊ IX_I2C_MASTER_BUS_ERROR − There was a bus error during transfer
◊ IX_I2C_MASTER_NO_BUFFER − buffer pointer is NULL
◊ IX_I2C_MASTER_INVALID_XFER_MODE − Xfer mode selected is invalid
◊ IX_I2C_INVALID_SLAVE_ADDR − invalid slave address (zero) specified
◊ IX_I2C_DATA_SIZE_ZERO − dataSize passed in is zero, which is invalid
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : no
- ISR Callable : no

ixI2cDrvShow (void)

Displays the I2C status register and the statistics counter.

Parameters:

− None

Global Data :

- None.

This API will display the I2C Status register and is useful if any error occurs. It displays the detection of bus error, slave address, general call, address, IDBR receive full, IDBR transmit empty, arbitration loss, slave STOP signal, I2C bus busy, unit busy, ack/nack, read/write mode, and delay type selected. It will also call the ixI2cDrvGetStats and then display the statistics counter.

Returns:

◊ IX_I2C_SUCCESS − successfully displayed statistics and status register
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : yes
- ISR Callable : no

ixI2cDrvSlaveAddrAndGenCallDetectedCheck (void)

Checks the I2C Status Register to determine if a slave address is detected.

Parameters:

− None

Global Data :

- None.

This API is used in polled mode to determine if the I2C unit is requested for a slave or general call transfer. If it is requested for a slave transfer then it will determine if it is a general call (read only), read, or write transfer requested.

Returns:

◊ IX_I2C_SLAVE_ADDR_NOT_DETECTED − The I2C unit is not requested for slave transfer
◊ IX_I2C_GEN_CALL_ADDR_DETECTED − The I2C unit is not requested for slave transfer but for general call
◊ IX_I2C_SLAVE_READ_DETECTED − The I2C unit is requested for a read
◊ IX_I2C_SLAVE_WRITE_DETECTED − The I2C unit is requested for a write
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : no
- ISR Callable : no

ixI2cDrvSlaveAddrSet (UINT8 *SlaveAddrSet*)

Sets the I2C Slave Address.

Parameters:

UINT8 [in] SlaveAddrSet − Slave Address to be inserted into ISAR

Global Data :

- None.

This API will insert the SlaveAddrSet into the ISAR.

Returns:

◊ IX_I2C_SUCCESS − successfuly set the slave addr
◊ IX_I2C_INVALID_SLAVE_ADDR − invalid slave address (zero) specified
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : yes
- ISR Callable : yes

```
ixI2cDrvSlaveDataTransmit ( char *          bufP,
                            const UINT32    dataSize,
                            UINT32 *        dataSizeXmtd
                          )
```

Performs the slave write data transfer.

Parameters:

*char [in] *bufP* − the pointer to the buffer for data to be transmitted "const UINT32 [in] bufSize" − the buffer size allocated "UINT32 [in] *dataSizeRcvd" − the length of data trasnmitted in bytes

Global Data :

- None.

This API is only used in polled mode to perform the slave transmit data. It will continuously transmit the data from bufP until complete or until bufP is empty in which it will return IX_I2C_SLAVE_WRITE_BUFFER_EMPTY. If in interrupt mode, the callback will be used.

Returns:

◊ IX_I2C_SUCCESS − The I2C driver transferred the data successfully.
◊ IX_I2C_SLAVE_WRITE_BUFFER_EMPTY − The I2C driver needs more data to transmit.
◊ IX_I2C_SLAVE_WRITE_ERROR −The I2C driver didn't manage to detect the IDBR Tx empty bit or the slave stop bit.
◊ IX_I2C_DATA_SIZE_ZERO − dataSize passed in is zero, which is invalid
◊ IX_I2C_SLAVE_NO_BUFFER − buffer pointer is NULL
◊ IX_I2C_NULL_POINTER − dataSizeXmtd is NULL
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : no

- ISR Callable : no

```
ixI2cDrvSlaveOrGenCallBufReplenish ( char *     bufP,
                                     UINT32 bufSize
                                   )
```

Replenishes the buffer which stores buffer info for both slave and general call.

Parameters:

*char [in] *bufP* − pointer to the buffer allocated "UINT32 [in] bufSize" − size of the buffer

Global Data :

- None.

This API is only used in interrupt mode for replenishing the same buffer that is used by both slave and generall call by updating the buffer info with new info and clearing previous offsets set by previous transfers.

Returns:

◊ None

- Reentrant : no
- ISR Callable : no

```
ixI2cDrvSlaveOrGenDataReceive ( char *           bufP,
                                const UINT32 bufSize,
                                UINT32 *       dataSizeRcvd
                              )
```

Performs the slave receive or general call receive data transfer.

Parameters:

*char [in] *bufP* − the pointer to the buffer to store data "const UINT32 [in] bufSize" − the buffer size allocated "UINT32 [in] *dataSizeRcvd" − the length of data received in bytes

Global Data :

- None.

This API is only used in polled mode to perform the slave read or general call receive data. It will continuously store the data received into bufP until complete or until bufP is full in which it will return IX_I2C_SLAVE_OR_GEN_READ_BUFFER_FULL. If in interrupt mode, the callback will be used.

Returns:

◊ IX_I2C_SUCCESS − The I2C driver transferred the data successfully.
◊ IX_I2C_SLAVE_OR_GEN_READ_BUFFER_FULL − The I2C driver has ran out of space to store the received data.

◊ IX_I2C_SLAVE_OR_GEN_READ_ERROR − The I2C driver didn't manage to detect the IDBR Rx Full bit
◊ IX_I2C_DATA_SIZE_ZERO − bufSize passed in is zero, which is invalid
◊ IX_I2C_SLAVE_NO_BUFFER − buffer pointer is NULL
◊ IX_I2C_NULL_POINTER − dataSizeRcvd is NULL
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : no
- ISR Callable : no

ixI2cDrvStatsGet (**IxI2cStatsCounters** * *I2cStats*)

Returns the I2C Statistics through the pointer passed in.

Parameters:

− "IxI2cStatsCounters [out] *I2cStats" − I2C statistics counter will be read and written to the location pointed by this pointer.

Global Data :

- None.

This API will return the statistics counters of the I2C driver.

Returns:

◊ IX_I2C_NULL_POINTER − pointer passed in is NULL
◊ IX_I2C_SUCCESS − successfully obtained I2C statistics

- Reentrant : yes
- ISR Callable : no

ixI2cDrvStatsReset (void)

Reset I2C statistics counters.

Parameters:

− None

Global Data :

- None.

This API will reset the statistics counters of the I2C driver.

Returns:

◊ None

- Reentrant : yes

- ISR Callable : no

```
ixI2cDrvUninit ( void )
```

Disables the I2C hardware.

Parameters:

- None

Global Data :

- None.

This API will disable the I2C hardware, unbind interrupt, and unmap memory.

Returns:

- ◊ IX_I2C_SUCCESS − successfully un−initialized I2C
- ◊ IX_I2C_INT_UNBIND_FAIL − failed to unbind the I2C interrupt
- ◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : yes
- ISR Callable : yes

```
ixI2cDrvWriteTransfer ( UINT8          SlaveAddr,
                        char *         bufP,
                        UINT32         dataSize,
                        IxI2cXferMode  XferModeSelect
                      )
```

Parameters:

UINT8 [in] SlaveAddr	− The slave address to request data from.
*char [in] *bufP*	− The pointer to the data to be transmitted.
UINT32 [in] dataSize	− The number of bytes requested.
IxI2cXferMode [in] XferModeSelect	− the transfer mode selected, either repeated start (ends w/o stop) or normal (start and stop)

Global Data :

- None.

This API will try to obtain master control of the I2C bus and transmit the number of bytes, specified by dataSize, to the user specified slave address from the buffer pointer. It will use either interrupt or poll mode depending on the method selected.

If in interrupt mode and IxI2cMasterWriteCallbackP is not NULL, the driver will initiate the transfer and return immediately. The function pointed to by IxI2cMasterWriteCallbackP will be called in the interrupt

service handlers when the operation is complete.

If in interrupt mode and IxI2cMasterWriteCallbackP is NULL, then the driver will wait for the operation to complete, and then return.

And if the repeated start transfer mode is selected, then it will not send a stop signal at the end of all the transfers. *NOTE*: If repeated start transfer mode is selected, it has to end with a normal mode transfer mode else the bus will continue to be held by the Intel (R) IXP4XX Product Line of Network Processors.

Returns:

◊ IX_I2C_SUCCESS − Successfuuly wrote data to slave.
◊ IX_I2C_MASTER_BUS_BUSY − The I2C bus is busy (held by another I2C master)
◊ IX_I2C_MASTER_ARB_LOSS − The I2C bus was loss to another I2C master
◊ IX_I2C_MASTER_XFER_ERROR − There was a transfer error
◊ IX_I2C_MASTER_BUS_ERROR − There was a bus error during transfer
◊ IX_I2C_MASTER_NO_BUFFER − buffer pointer is NULL
◊ IX_I2C_MASTER_INVALID_XFER_MODE − Xfer mode selected is invalid
◊ IX_I2C_DATA_SIZE_ZERO − dataSize passed in is zero, which is invalid
◊ IX_I2C_NOT_INIT − I2C not init yet.

- Reentrant : no
- ISR Callable : no

Intel (R) IXP NPE−Downloader (IxNpeDl) API

The Public API for the IXP NPE Downloader.

Modules

Intel (R) IXP
Image ID
Definition

*Definition of Image ID to be passed to **ixNpeDlNpeInitAndStart()** or **ixNpeDlCustomImageNpeInitAndStart()** as input of type UINT32 which has the following fields format:.*

Defines

#define **IX_NPEDL_PARAM_ERR**

NpeDl function return value for a parameter error.

#define **IX_NPEDL_RESOURCE_ERR**

NpeDl function return value for a resource error.

#define **IX_NPEDL_CRITICAL_NPE_ERR**

NpeDl function return value for a Critical NPE error occuring during download. Assume NPE is left in unstable condition if this value is returned or NPE is hang / halt.

#define **IX_NPEDL_CRITICAL_MICROCODE_ERR**

NpeDl function return value for a Critical Microcode error discovered during download. Assume NPE is left in unstable condition if this value is returned.

#define **IX_NPEDL_DEVICE_ERR**

NpeDl function return value when image being downloaded is not meant for the device in use.

Functions

PUBLIC
IX_STATUS **ixNpeDlNpeInitAndStart** (UINT32 imageId)

Stop, reset, download microcode (firmware) and finally start NPE.

PUBLIC
IX_STATUS **ixNpeDlCustomImageNpeInitAndStart** (UINT32 *imageLibrary, UINT32 imageId)

Stop, reset, download microcode (firmware) and finally start NPE.

PUBLIC
IX_STATUS **ixNpeDlLoadedImageFunctionalityGet** (**IxNpeDlNpeId** npeId, UINT8 *functionalityId)

Gets the functionality of the image last loaded on a particular NPE.

PUBLIC
IX_STATUS **ixNpeDlNpeStopAndReset** (**IxNpeDlNpeId** npeId)
Stops and Resets an NPE.

PUBLIC
IX_STATUS **ixNpeDlNpeExecutionStart** (**IxNpeDlNpeId** npeId)
Starts code execution on a NPE.

PUBLIC
IX_STATUS **ixNpeDlNpeExecutionStop** (**IxNpeDlNpeId** npeId)
Stops code execution on a NPE.

PUBLIC
IX_STATUS **ixNpeDlUnload** (void)
This function will uninitialise the IxNpeDl component.

PUBLIC void **ixNpeDlStatsShow** (void)
This function will display run−time statistics from the IxNpeDl component.

PUBLIC void **ixNpeDlStatsReset** (void)
This function will reset the statistics of the IxNpeDl component.

PUBLIC
UINT32 **ixNpeDlDataMemRead** (UINT32 npeId, UINT32 dataMemAddress)
This function will read 1 WORD from the DMEM of the NPE.

PUBLIC
IX_STATUS **ixNpeDlLoadedImageGet** (**IxNpeDlNpeId** npeId, **IxNpeDlImageId** *imageIdPtr)
Gets the Id of the image currently loaded on a particular NPE.

Detailed Description

The Public API for the IXP NPE Downloader.

Define Documentation

```
#define IX_NPEDL_CRITICAL_MICROCODE_ERR
```

NpeDl function return value for a Critical Microcode error discovered during download. Assume NPE is left in unstable condition if this value is returned.

Definition at line **65** of file **IxNpeDl.h**.

#define IX_NPEDL_CRITICAL_NPE_ERR

NpeDl function return value for a Critical NPE error occuring during download. Assume NPE is left in unstable condition if this value is returned or NPE is hang / halt.

Definition at line **56** of file **IxNpeDl.h**.

#define IX_NPEDL_DEVICE_ERR

NpeDl function return value when image being downloaded is not meant for the device in use.

Definition at line **73** of file **IxNpeDl.h**.

#define IX_NPEDL_PARAM_ERR

NpeDl function return value for a parameter error.

Definition at line **40** of file **IxNpeDl.h**.

#define IX_NPEDL_RESOURCE_ERR

NpeDl function return value for a resource error.

Definition at line **47** of file **IxNpeDl.h**.

Function Documentation

```
PUBLIC IX_STATUS ixNpeDlCustomImageNpeInitAndStart ( UINT32 * imageLibrary,
                                                     UINT32   imageId
                                                   )
```

Stop, reset, download microcode (firmware) and finally start NPE.

Parameters:

imageId UINT32 [in] − Id of the microcode image to download.

This function locates the image specified by the *imageId* parameter from the specified microcode image library which is pointed to by the *imageLibrary* parameter. It then stops and resets the NPE, loads the firmware image onto the NPE, and then restarts the NPE.

This is a facility for users who wish to use an image from an external library of NPE firmware images. To use a standard image from the built−in library, see **ixNpeDlNpeInitAndStart** instead.

Note:

A list of valid image IDs is included in this header file. See #defines with prefix IX_NPEDL_NPEIMAGE_...

Precondition:

- ◊ The Client is responsible for ensuring mutual access to the NPE.
- ◊ The image library supplied must be in the correct format for use by the NPE Downloader (IxNpeDl) component. Details of the library format are contained in the Functional Specification document for IxNpeDl.

Postcondition:

- ◊ The NPE Instruction Pipeline will be cleared if State Information has been downloaded.
- ◊ If the download fails with a critical error, the NPE may be left in an ususable state.

Returns:

- ◊ IX_SUCCESS if the download was successful;
- ◊ IX_NPEDL_PARAM_ERR if a parameter error occured
- ◊ IX_NPEDL_CRITICAL_NPE_ERR if a critical NPE error occured during download
- ◊ IX_NPEDL_CRITICAL_MICROCODE_ERR if a critical microcode error occured during download
- ◊ IX_NPEDL_DEVICE_ERR if the image being loaded is not meant for the device currently running.
- ◊ IX_FAIL if NPE is not available or image is failed to be located. A warning is issued if the NPE is not present.

```
UINT32 ixNpeDlDataMemRead ( UINT32 npeId,
                            UINT32 dataMemAddress
                          )
```

This function will read 1 WORD from the DMEM of the NPE.

Parameters:

npeId	**IxNpeDlNpeId** [in] − Id of the target NPE
dataMemAddress	UINT32 [in] − DMEM Address

Returns:

DMEM Memory contents value in WORD size

```
PUBLIC IX_STATUS ixNpeDlLoadedImageFunctionalityGet ( IxNpeDlNpeId npeId,
                                                      UINT8 *      functionalityId
                                                    )
```

Gets the functionality of the image last loaded on a particular NPE.

Parameters:

npeId	**IxNpeDlNpeId** [in] − Id of the target NPE.
functionalityId	UINT8* [out] − the functionality ID of the image last loaded on the NPE.

This function retrieves the functionality ID of the image most recently downloaded successfully to the specified NPE. If the NPE does not contain a valid image, this function returns a FAIL status.

Warning:

This function is not intended for general use, as a knowledge of how to interpret the functionality ID is required. As such, this function should only be used by other Access Layer components of the IXP Software Release.

Precondition:

Postcondition:

Returns:

◊ IX_SUCCESS if the operation was successful
◊ IX_NPEDL_PARAM_ERR if a parameter error occured
◊ IX_FAIL if the NPE does not have a valid image loaded

```
PUBLIC IX_STATUS ixNpeDlLoadedImageGet ( IxNpeDlNpeId      npeId,
                                         IxNpeDlImageId *  imageIdPtr
                                       )
```

Gets the Id of the image currently loaded on a particular NPE.

Parameters:

npeId **IxNpeDlNpeId** [in] − Id of the target NPE.
imageIdPtr **IxNpeDlImageId*** [out] − Pointer to the where the image id should be stored.

If an image of microcode was previously downloaded successfully to the NPE by NPE Downloader, this function returns in *imageIdPtr* the image Id of that image loaded on the NPE.

Precondition:

◊ The Client has allocated memory to the *imageIdPtr* pointer.

Postcondition:

Returns:

◊ IX_SUCCESS if the operation was successful
◊ IX_NPEDL_PARAM_ERR if a parameter error occured
◊ IX_FAIL if the NPE doesn't currently have a image loaded

```
PUBLIC IX_STATUS ixNpeDlNpeExecutionStart ( IxNpeDlNpeId npeId )
```

Starts code execution on a NPE.

Parameters:

npeId **IxNpeDlNpeId** [in] − Id of the target NPE

Starts execution of code on a particular NPE. A client would typically use this after a download to NPE is performed, to start/restart code execution on the NPE.

Note:

It is no longer necessary to call this function after downloading a new image to the NPE. It is left on the API only to allow greater control of NPE execution if required. Where appropriate, use **ixNpeDlNpeInitAndStart** or **ixNpeDlCustomImageNpeInitAndStart** instead.

Precondition:

- ◊ The Client is responsible for ensuring mutual access to the NPE.
- ◊ Note that this function does not set the NPE's Next Program Counter (NextPC), so it should be set beforehand if required by downloading appropriate State Information (using **ixNpeDlNpeInitAndStart()**).

Postcondition:

Returns:

- ◊ IX_SUCCESS if the operation was successful
- ◊ IX_NPEDL_PARAM_ERR if a parameter error occured
- ◊ IX_FAIL otherwise

PUBLIC IX_STATUS ixNpeDlNpeExecutionStop (**IxNpeDlNpeId** *npeId*)

Stops code execution on a NPE.

Parameters:

npeId **IxNpeDlNpeId** [in] − Id of the target NPE

Stops execution of code on a particular NPE. This would typically be used by a client before a download to NPE is performed, to stop code execution on an NPE, unless **ixNpeDlNpeStopAndReset()** is used instead. Unlike **ixNpeDlNpeStopAndReset()**, this function only halts the NPE and leaves all registers and settings intact. This is useful, for example, between stages of a multi−stage download, to stop the NPE prior to downloading the next image while leaving the current state of the NPE intact..

Precondition:

- ◊ The Client is responsible for ensuring mutual access to the NPE.

Postcondition:

Returns:

- ◊ IX_SUCCESS if the operation was successful
- ◊ IX_NPEDL_PARAM_ERR if a parameter error occured
- ◊ IX_FAIL otherwise

PUBLIC IX_STATUS ixNpeDlNpeInitAndStart (UINT32 *imageId*)

Stop, reset, download microcode (firmware) and finally start NPE.

Parameters:

imageId UINT32 [in] − Id of the microcode image to download.

This function locates the image specified by the *imageId* parameter from the default microcode image library which is included internally by this component. It then stops and resets the NPE, loads the firmware image onto the NPE, and then restarts the NPE.

Note:

A list of valid image IDs is included in this header file. See #defines with prefix IX_NPEDL_NPEIMAGE_...

Precondition:

◊ The Client is responsible for ensuring mutual access to the NPE.

Postcondition:

◊ The NPE Instruction Pipeline will be cleared if State Information has been downloaded.
◊ If the download fails with a critical error, the NPE may be left in an ususable state.

Returns:

◊ IX_SUCCESS if the download was successful;
◊ IX_NPEDL_PARAM_ERR if a parameter error occured
◊ IX_NPEDL_CRITICAL_NPE_ERR if a critical NPE error occured during download
◊ IX_NPEDL_CRITICAL_MICROCODE_ERR if a critical microcode error occured during download
◊ IX_NPEDL_DEVICE_ERR if the image being loaded is not meant for the device currently running.
◊ IX_FAIL if NPE is not available or image is failed to be located. A warning is issued if the NPE is not present.

PUBLIC IX_STATUS ixNpeDlNpeStopAndReset (**IxNpeDlNpeId** *npeId*)

Stops and Resets an NPE.

Parameters:

npeId **IxNpeDlNpeId** [in] − Id of the target NPE.

This function performs a soft NPE reset by writing reset values to particular NPE registers. Note that this does not reset NPE co−processors. This implicitly stops NPE code execution before resetting the NPE.

Note:

It is no longer necessary to call this function before downloading a new image to the NPE. It is left on the API only to allow greater control of NPE execution if required. Where appropriate, use **ixNpeDlNpeInitAndStart** or **ixNpeDlCustomImageNpeInitAndStart** instead.

Precondition:

◊ The Client is responsible for ensuring mutual access to the NPE.

Postcondition:

Returns:

◊ IX_SUCCESS if the operation was successful
◊ IX_NPEDL_PARAM_ERR if a parameter error occured
◊ IX_FAIL otherwise
◊ IX_NPEDL_CRITICAL_NPE_ERR failed to reset NPE due to timeout error. Timeout error could happen if NPE hang

PUBLIC void ixNpeDlStatsReset (void)

This function will reset the statistics of the IxNpeDl component.

Returns:

none

PUBLIC void ixNpeDlStatsShow (void)

This function will display run−time statistics from the IxNpeDl component.

Returns:

none

PUBLIC IX_STATUS ixNpeDlUnload (void)

This function will uninitialise the IxNpeDl component.

This function will uninitialise the IxNpeDl component. It should only be called once, and only if the IxNpeDl component has already been initialised by calling any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

If possible, this function should be called before a soft reboot or unloading a kernel module to perform any clean up operations required for IxNpeDl.

The following actions will be performed by this function:

- Unmapping of any kernel memory mapped by IxNpeDl

Returns:

◊ IX_SUCCESS if the operation was successful
◊ IX_FAIL otherwise

Intel (R) IXP Image ID Definition

[Intel (R) IXP NPE−Downloader (IxNpeDl) API]

Definition of Image ID to be passed to **ixNpeDlNpeInitAndStart()** or **ixNpeDlCustomImageNpeInitAndStart()** as input of type UINT32 which has the following fields format:.

Data Structures

struct **IxNpeDlImageId**
Image Id to identify each image contained in an image library.

Defines

#define **IX_NPEDL_NPEIMAGE_FIELD_MASK**
Mask for NPE Image ID's Field.

#define **IX_NPEDL_NPEIMAGE_NPEID_MASK**
Mask for NPE Image NPE ID's Field.

#define **IX_NPEDL_NPEIMAGE_DEVICEID_MASK**
Mask for NPE Image Device ID's Field.

#define **IX_NPEIMAGEID_FUNCTIONID_OFFSET**
NPE Image function ID's offset (16bits).

#define **IX_FUNCTIONID_FROM_NPEIMAGEID_GET**(imageId)
Macro to extract Functionality ID field from Image ID.

Typedefs

typedef UINT8 **IxNpeDlFunctionalityId**
Used to make up Functionality ID field of Image Id.

typedef UINT8 **IxNpeDlMajor**
Used to make up Major Release field of Image Id.

typedef UINT8 **IxNpeDlMinor**
Used to make up Minor Revision field of Image Id.

typedef UINT8 **IxNpeDlDeviceId**
Used to make up Device Id field of Image Id.

Enumerations

enum **IxNpeDlNpeId** {
IX_NPEDL_NPEID_NPEA,
IX_NPEDL_NPEID_NPEB,
IX_NPEDL_NPEID_NPEC,
IX_NPEDL_NPEID_MAX
}
NpeId numbers to identify the system's NPEs.

Detailed Description

Definition of Image ID to be passed to **ixNpeDlNpeInitAndStart()** or **ixNpeDlCustomImageNpeInitAndStart()** as input of type UINT32 which has the following fields format:.

The following is the structure of the Image ID. The first row shows the fields; the second shows the bit locations.

Device ID	NPE ID	Functionality ID	Major Release Number	Minor Release Number
31	27	23	15	7

Field [Bit Location] − Purpose
−−−

- Device ID [31 − 28] − Specifies the type of device the image is to be downloaded to
- NPE ID [27 − 24] − Specifies the NPE the image is to be downloaded to
- Functionality ID [23 − 16] − Specifies the functionality of the image
- Major Release Number [15 − 8] − Specifies the major version number of the image
- Minor Release Number [7 − 0] − Specifies the minor version number of the image

---

Define Documentation

#define IX_FUNCTIONID_FROM_NPEIMAGEID_GET (imageId)

Macro to extract Functionality ID field from Image ID.

Definition at line **152** of file **IxNpeDl.h**.

#define IX_NPEDL_NPEIMAGE_DEVICEID_MASK

Mask for NPE Image Device ID's Field.

Definition at line **137** of file **IxNpeDl.h**.

#define IX_NPEDL_NPEIMAGE_FIELD_MASK

Mask for NPE Image ID's Field.

Definition at line **121** of file **IxNpeDl.h**.

#define IX_NPEDL_NPEIMAGE_NPEID_MASK

Mask for NPE Image NPE ID's Field.

Definition at line **129** of file **IxNpeDl.h**.

#define IX_NPEIMAGEID_FUNCTIONID_OFFSET

NPE Image function ID's offset (16bits).

Definition at line **145** of file **IxNpeDl.h**.

Typedef Documentation

IxNpeDlDeviceId

Used to make up Device Id field of Image Id.

See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **189** of file **IxNpeDl.h**.

IxNpeDlFunctionalityId

Used to make up Functionality ID field of Image Id.

See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **165** of file **IxNpeDl.h**.

IxNpeDlMajor

Used to make up Major Release field of Image Id.

See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **173** of file **IxNpeDl.h**.

IxNpeDlMinor

Used to make up Minor Revision field of Image Id.

See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **181** of file **IxNpeDl.h**.

Enumeration Type Documentation

enum IxNpeDlNpeId

NpeId numbers to identify the system's NPEs.

Enumeration values:

IX_NPEDL_NPEID_NPEA	Identifies NPE A.
IX_NPEDL_NPEID_NPEB	Identifies NPE B.
IX_NPEDL_NPEID_NPEC	Identifies NPE C.
IX_NPEDL_NPEID_MAX	Total Number of NPEs.

Definition at line **205** of file **IxNpeDl.h**.

Intel (R) IXP Software NPE Message Handler (IxNpeMh) API

The public API for the IXP NPE Message Handler component.

Data Structures

struct **IxNpeMhMessage**
The 2−word message structure to send to and receive from the NPEs.

Defines

#define **IX_NPEMH_MIN_MESSAGE_ID**
minimum valid message ID

#define **IX_NPEMH_MAX_MESSAGE_ID**
maximum valid message ID

#define **IX_NPEMH_SEND_RETRIES_DEFAULT**
default msg send retries

#define **IX_NPEMH_CRITICAL_NPE_ERR**
NpeMH function return value for a Critical NPE error occuring during sending/receiving message. Assume NPE hang / halt if this value is returned.

Typedefs

typedef UINT32 **IxNpeMhMessageId**
message ID

typedef void(* **IxNpeMhCallback**)(**IxNpeMhNpeId**, **IxNpeMhMessage**)
This prototype shows the format of a message callback function.

Enumerations

enum **IxNpeMhNpeId** {
IX_NPEMH_NPEID_NPEA,
IX_NPEMH_NPEID_NPEB,
IX_NPEMH_NPEID_NPEC,
IX_NPEMH_NUM_NPES
}

The ID of a particular NPE.

enum **IxNpeMhNpeInterrupts** {
IX_NPEMH_NPEINTERRUPTS_NO,
IX_NPEMH_NPEINTERRUPTS_YES
}
Indicator specifying whether or not NPE interrupts should drive receiving of messages from the NPEs.

Functions

PUBLIC IX_STATUS **ixNpeMhInitialize** (**IxNpeMhNpeInterrupts** npeInterrupts)
This function will initialise the IxNpeMh component.

PUBLIC IX_STATUS **ixNpeMhUnload** (void)
This function will uninitialise the IxNpeMh component.

PUBLIC IX_STATUS **ixNpeMhUnsolicitedCallbackRegister** (**IxNpeMhNpeId** npeId, **IxNpeMhMessageId** messageId, **IxNpeMhCallback** unsolicitedCallback)
This function will register an unsolicited callback for a particular NPE and message ID.

PUBLIC IX_STATUS **ixNpeMhUnsolicitedCallbackForRangeRegister** (**IxNpeMhNpeId** npeId, **IxNpeMhMessageId** minMessageId, **IxNpeMhMessageId** maxMessageId, **IxNpeMhCallback** unsolicitedCallback)
This function will register an unsolicited callback for a particular NPE and range of message IDs.

PUBLIC IX_STATUS **ixNpeMhMessageSend** (**IxNpeMhNpeId** npeId, **IxNpeMhMessage** message, UINT32 maxSendRetries)
This function will send a message to a particular NPE.

PUBLIC IX_STATUS **ixNpeMhMessageWithResponseSend** (**IxNpeMhNpeId** npeId, **IxNpeMhMessage** message, **IxNpeMhMessageId** solicitedMessageId, **IxNpeMhCallback** solicitedCallback, UINT32 maxSendRetries)
*This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicited a response.*

PUBLIC IX_STATUS **ixNpeMhMessagesReceive** (**IxNpeMhNpeId** npeId)
This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).

PUBLIC IX_STATUS **ixNpeMhShow** (**IxNpeMhNpeId** npeId)
This function will display the current state of the IxNpeMh component.

PUBLIC IX_STATUS **ixNpeMhShowReset** (**IxNpeMhNpeId** npeId)

This function will reset the current state of the IxNpeMh component.

IX_STATUS **ixNpeMhConfigStateRestore** (**IxNpeMhNpeId** npeId)
This function will restore the Interrupt Configuration setting.

Detailed Description

The public API for the IXP NPE Message Handler component.

Define Documentation

#define IX_NPEMH_CRITICAL_NPE_ERR

NpeMH function return value for a Critical NPE error occuring during sending/receiving message. Assume NPE hang / halt if this value is returned.

Definition at line **44** of file **IxNpeMh.h**.

#define IX_NPEMH_MAX_MESSAGE_ID

maximum valid message ID

Definition at line **32** of file **IxNpeMh.h**.

#define IX_NPEMH_MIN_MESSAGE_ID

minimum valid message ID

Definition at line **31** of file **IxNpeMh.h**.

#define IX_NPEMH_SEND_RETRIES_DEFAULT

default msg send retries

Definition at line **34** of file **IxNpeMh.h**.

Typedef Documentation

IxNpeMhCallback

This prototype shows the format of a message callback function.

This prototype shows the format of a message callback function. The message callback will be passed the message to be handled and will also be told from which NPE the message was received. The message callback will either be registered by **ixNpeMhUnsolicitedCallbackRegister()** or passed as a parameter to **ixNpeMhMessageWithResponseSend()**. It will be called from within an ISR triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**). The parameters passed are the ID of the NPE that the message was received from, and the message to be handled.

Re−entrancy: This function is only a prototype, and will be implemented by the client. It does not need to be re−entrant.

Definition at line **111** of file **IxNpeMh.h**.

typedef UINT32 IxNpeMhMessageId

message ID

Definition at line **92** of file **IxNpeMh.h**.

Enumeration Type Documentation

enum IxNpeMhNpeId

The ID of a particular NPE.

Note:

In this context, the Intel (R) IXP42X Product Line :

- ◊ NPE−A has HDLC, HSS, AAL and UTOPIA Coprocessors.
- ◊ NPE−B has Ethernet Coprocessor.
- ◊ NPE−C has Ethernet, AES, DES and HASH Coprocessors.
- ◊ Intel (R) IXP4XX Product Line of Network Processors have different combinations of coprocessors.

Enumeration values:

IX_NPEMH_NPEID_NPEA	Identifies NPE A.
IX_NPEMH_NPEID_NPEB	Identifies NPE B.
IX_NPEMH_NPEID_NPEC	Identifies NPE C.
IX_NPEMH_NUM_NPES	Number of NPEs.

Definition at line **59** of file **IxNpeMh.h**.

enum IxNpeMhNpeInterrupts

Indicator specifying whether or not NPE interrupts should drive receiving of messages from the NPEs.

Enumeration values:

IX_NPEMH_NPEINTERRUPTS_NO	Don't use NPE interrupts.
IX_NPEMH_NPEINTERRUPTS_YES	Do use NPE interrupts.

Definition at line **75** of file **IxNpeMh.h**.

Function Documentation

IX_STATUS ixNpeMhConfigStateRestore (**IxNpeMhNpeId** *npeId*)

This function will restore the Interrupt Configuration setting.

Note:

This API restores the Interrupt Enabling configuration for the OUTFIFO of the NPE used if the ixNPEMH is set to an interrupt mode during intialization. This API must be called once the NPE is reset which may occur during an NPE error.

- ◊ Re−entrant : No
- ◊ ISR Callable : Yes

Parameters:

npeId **IxNpeMhNpeId** [in] − The ID of the NPE to restore the configuration

Returns:

The fucntion returns a status indicating success or failure.

IX_STATUS ixNpeMhInitialize (**IxNpeMhNpeInterrupts** *npeInterrupts*)

This function will initialise the IxNpeMh component.

Parameters:

npeInterrupts **IxNpeMhNpeInterrupts** [in] − This parameter dictates whether or not the IxNpeMh component will service NPE "outFIFO not empty" interrupts to trigger receiving and processing of messages from the NPEs. If not then the client must use **ixNpeMhMessagesReceive()** to control message receiving and processing.

This function will initialise the IxNpeMh component. It should only be called once, prior to using the IxNpeMh component. The following actions will be performed by this function:

1. Initialization of internal data structures (e.g. solicited and unsolicited callback tables).
2. Configuration of the interface with the NPEs (e.g. enabling of NPE "outFIFO not empty" interrupts).
3. Registration of ISRs that will receive and handle messages when the NPEs' "outFIFO not empty" interrupts fire (if npeInterrupts equals IX_NPEMH_NPEINTERRUPTS_YES).

Returns:

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhMessageSend ( IxNpeMhNpeId     npeId,
                               IxNpeMhMessage   message,
                               UINT32           maxSendRetries
                             )
```

This function will send a message to a particular NPE.

Parameters:

npeId	**IxNpeMhNpeId** [in] − The ID of the NPE to send the message to.
message	**IxNpeMhMessage** [in] − The message to send.
maxSendRetries	UINT32 [in] − Max num. of retries to perform if the NPE's inFIFO is full.

This function will send a message to a particular NPE. It will be the client's responsibility to ensure that the message is properly formed. The return status will signify to the client if the message was successfully sent or not.

If the message is sent to the NPE then this function will return a status of success. Note that this will only mean the message has been placed in the NPE's inFIFO. There will be no way of knowing that the NPE has actually read the message, but once in the incoming message queue it will be safe to assume that the NPE will process it.

The inFIFO may fill up sometimes if the Intel XScale(R) processor is sending messages faster than the NPE can handle them. This forces us to retry attempts to send the message until the NPE services the inFIFO. The client should specify a ceiling value for the number of retries suitable to their needs. IX_NPEMH_SEND_RETRIES_DEFAULT can be used as a default value for the *maxSendRetries* parameter for this function. Each retry exceeding this default number will incur a blocking delay of 1 microsecond, to avoid consuming too much AHB bus bandwidth while performing retries.

Note this function **must** only be used for messages. that do not solicit responses. If the message being sent will solicit a response then the **ixNpeMhMessageWithResponseSend()** function **must** be used to ensure that the response is correctly handled.

This function will return timeout status if NPE hang / halt while sending message. The timeout error is not related to the *maxSendRetries* as mentioned above. The timeout error will only occur if the first word of the message has been sent to NPE (not exceeding *maxSendRetries* when sending 1st message word), but the second word of the message can't be written to NPE's inFIFO due to NPE hang / halt after maximum waiting time (IX_NPE_MH_MAX_NUM_OF_RETRIES).

Re−entrancy: This function will be callable from any thread at any time. IxOsal will be used for any necessary resource protection.

Returns:

The function returns a status indicating success, failure or timeout.

```
IX_STATUS ixNpeMhMessagesReceive ( IxNpeMhNpeId npeId )
```

This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).

Parameters:

npeId **IxNpeMhNpeId** [in] − The ID of the NPE to receive and process messages from.

This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).

If the IxNpeMh component is initialised to service NPE "outFIFO not empty" interrupts (see **ixNpeMhInitialize()**) then there is no need to call this function. This function is only provided as an alternative mechanism to control the receiving and processing of messages from the NPEs.

This function will return timeout status if NPE hang / halt while receiving message. The timeout error will only occur if this function has read the first word of the message and can't read second word of the message from NPE's outFIFO after maximum retries (IX_NPE_MH_MAX_NUM_OF_RETRIES).

Note this function cannot be called from within an ISR as it will use resource protection mechanisms.

Re−entrancy: This function will be callable from any thread at any time. IxOsal will be used for any necessary resource protection.

Returns:

The function returns a status indicating success, failure or timeout.

```
IX_STATUS ixNpeMhMessageWithResponseSend ( IxNpeMhNpeId       npeId,
                                           IxNpeMhMessage     message,
                                           IxNpeMhMessageId   solicitedMessageId,
                                           IxNpeMhCallback    solicitedCallback,
                                           UINT32             maxSendRetries
                                         )
```

This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicited a response.

Parameters:

npeId	**IxNpeMhNpeId** [in] − The ID of the NPE to send the message to.
message	**IxNpeMhMessage** [in] − The message to send.
solicitedMessageId	**IxNpeMhMessageId** [in] − The ID of the solicited response message.
solicitedCallback	**IxNpeMhCallback** [in] − The function to use to pass the response message back to the client. A value of NULL will cause the response message to be discarded.
maxSendRetries	UINT32 [in] − Max num. of retries to perform if the NPE's inFIFO is full.

This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicited a response.

The client must specify the ID of the solicited response message to allow the response to be recognised when it is received. The client must also specify a callback function to handle the received response. The IxNpeMh component will not offer the facility to send a message to a NPE and receive a response within the same context.

Note if the client is not interested in the response, specifying a NULL callback will cause the response message to be discarded.

The solicited callback will be stored and called some time later from an ISR that will be triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**) to handle the response message corresponding to the message sent. Response messages will be handled in the order they are received.

The inFIFO may fill up sometimes if the Intel XScale(R) processor is sending messages faster than the NPE can handle them. This forces us to retry attempts to send the message until the NPE services the inFIFO. The client should specify a ceiling value for the number of retries suitable to their needs. IX_NPEMH_SEND_RETRIES_DEFAULT can be used as a default value for the *maxSendRetries* parameter for this function. Each retry exceeding this default number will incur a blocking delay of 1 microsecond, to avoid consuming too much AHB bus bandwidth while performing retries.

This function will return timeout status if NPE hang / halt while sending message. The timeout error is not related to the *maxSendRetries* as mentioned above. The timeout error will only occur if the first word of the message has been sent to NPE (not exceeding *maxSendRetries* when sending 1st message word), but the second word of the message can't be written to NPE's inFIFO due to NPE hang / halt after maximum waiting time (IX_NPE_MH_MAX_NUM_OF_RETRIES).

Re−entrancy: This function will be callable from any thread at any time. IxOsal will be used for any necessary resource protection.

Returns:

The function returns a status indicating success or failure.

IX_STATUS ixNpeMhShow (**IxNpeMhNpeId** *npeId*)

This function will display the current state of the IxNpeMh component.

Re−entrancy: This function will be callable from any thread at any time. However, no resource protection will be used so as not to impact system performance. As this function is only reading statistical information then this is acceptable.

Parameters:

npeId **IxNpeMhNpeId** [in] − The ID of the NPE to display state information for.

Returns:

The function returns a status indicating success or failure.

IX_STATUS ixNpeMhShowReset (**IxNpeMhNpeId** *npeId*)

This function will reset the current state of the IxNpeMh component.

Re−entrancy: This function will be callable from any thread at any time. However, no resource protection will be used so as not to impact system performance. As this function is only writing statistical information then this is acceptable.

Parameters:

npeId **IxNpeMhNpeId** [in] − The ID of the NPE to reset state information for.

Returns:

The function returns a status indicating success or failure.

IX_STATUS ixNpeMhUnload (void)

This function will uninitialise the IxNpeMh component.

This function will uninitialise the IxNpeMh component. It should only be called once, and only if the IxNpeMh component has already been initialised. No other IxNpeMh API functions should be called until **ixNpeMhInitialize** is called again. If possible, this function should be called before a soft reboot or unloading a kernel module to perform any clean up operations required for IxNpeMh.

The following actions will be performed by this function:

1. Unmapping of kernel memory mapped by the function **ixNpeMhInitialize**.

Returns:

The function returns a status indicating success or failure.

IX_STATUS ixNpeMhUnsolicitedCallbackForRangeRegister (**IxNpeMhNpeId** *npeId*, **IxNpeMhMessageId** *minMessageId*, **IxNpeMhMessageId** *maxMessageId*, **IxNpeMhCallback** *unsolicitedCallback*)

This function will register an unsolicited callback for a particular NPE and range of message IDs.

Parameters:

npeId	**IxNpeMhNpeId** [in] − The ID of the NPE whose messages the unsolicited callback will handle.
minMessageId	**IxNpeMhMessageId** [in] − The minimum message ID in the range of message IDs the unsolicited callback will handle.
maxMessageId	**IxNpeMhMessageId** [in] − The maximum message ID in the range of message IDs the unsolicited callback will handle.
unsolicitedCallback	**IxNpeMhCallback** [in] − The unsolicited callback function. A value of NULL will deregister any previously registered callback(s) for this NPE and range of message IDs.

This function will register an unsolicited callback for a particular NPE and range of message IDs. It is a convenience function that is effectively the same as calling **ixNpeMhUnsolicitedCallbackRegister()** for each ID in the specified range. See **ixNpeMhUnsolicitedCallbackRegister()** for more information.

Re−entrancy: This function will be callable from any thread at any time. IxOsal will be used for any necessary resource protection.

Returns:

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhUnsolicitedCallbackRegister ( IxNpeMhNpeId       npeId,
                                               IxNpeMhMessageId   messageId,
                                               IxNpeMhCallback    unsolicitedCallback
                                             )
```

This function will register an unsolicited callback for a particular NPE and message ID.

Parameters:

npeId	**IxNpeMhNpeId** [in] − The ID of the NPE whose messages the unsolicited callback will handle.
messageId	**IxNpeMhMessageId** [in] − The ID of the messages the unsolicited callback will handle.
unsolicitedCallback	**IxNpeMhCallback** [in] − The unsolicited callback function. A value of NULL will deregister any previously registered callback for this NPE and message ID.

This function will register an unsolicited message callback for a particular NPE and message ID.

If an unsolicited callback is already registered for the specified NPE and message ID then the callback will be overwritten. Only one client will be responsible for handling a particular message ID associated with a NPE. Registering a NULL unsolicited callback will deregister any previously registered callback.

The callback function will be called from an ISR that will be triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**) to handle any unsolicited messages of the specific message ID received from the NPE. Unsolicited messages will be handled in the order they are received.

If no unsolicited callback can be found for a received message then it is assumed that the message is solicited.

If more than one client may be interested in a particular unsolicited message then the suggested strategy is to register a callback for the message that can itself distribute the message to multiple clients as necessary.

See also **ixNpeMhUnsolicitedCallbackForRangeRegister()**.

Re−entrancy: This function will be callable from any thread at any time. IxOsal will be used for any necessary resource protection.

Returns:

The function returns a status indicating success or failure.

Intel (R) IXP400 NPE Microcode Image Library

Library containing a set of NPE firmware images, for use with NPE Downloader s/w component.

Defines

#define **IX_NPE_IMAGE_INCLUDE**

Wrap the following Image identifiers with "#if IX_NPE_IMAGE_INCLUDE ... #endif" to include the image in library.

#define **IX_NPE_IMAGE_OMIT**

Wrap the following Image identifiers with "#if IX_NPE_IMAGE_OMIT ... #endif" to OMIT the image from the

#define **IX_NPEDL_NPEIMAGE_NPEA_ATM_MPHY_12_PORT**

NPE Image Id for NPE−A with ATM−Only feature. It supports AAL5, AAL0 and OAM for UTOPIA MPHY, 1 ports, 32 VCs. It also has Fast Path support.

#define **IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_SPHY_1_PORT**

NPE Image Id for NPE−A with HSS−0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 pac For ATM, it supports AAL5, AAL0 and OAM for UTOPIA SPHY, 1 logical port, 32 VCs. It also has Fast Path

#define **IX_NPEDL_NPEIMAGE_NPEA_DMA**

NPE Image Id for NPE−A with DMA−Only feature.

#define **IX_NPEDL_NPEIMAGE_NPEA_HSS_2_PORT**

NPE Image Id for NPE−A with HSS−0 and HSS−1 feature. Each HSS port supports 32 channelized and 4 pac

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH**

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_FIREWALL**

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS**

NPE Image Id for NPE−A with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV**

NPE Image Id for NPE−A with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS, HEADER_CONVERSION.

#define **IX_NPEDL_NPEIMAGE_NPEA_HSS_TSLOT_SWITCH**

NPE Image Id for NPE−A with HSS−0 and timeslot switching feature. It supports 32 channelized and 4 packe HSS−0 as well.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_**

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB**
NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXT**
NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FI VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_MACFILTERLEARN_HSSCHAN_COEXIST**
NPE Image Id for NPE−A with HSS−0 and Ethernet (with MAC filter/learn) feature.

#define **IX_NPEDL_NPEIMAGE_NPEA_ETH_HDRCONV_HSSCHAN_COEXIST**
NPE Image Id for NPE−A with HSS−0 and Ethernet (with header conversion) feature.

#define **IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_MPHY_4_PORT**
NPE Image Id for NPE−A with ATM MPHY 4 Ports and HSS Channelized Service (with HSS Bypass) feature.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH**
NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_FIREWALL**
NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS**
NPE Image Id for NPE−B with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV**
NPE Image Id for NPE−B with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS, HEADER_CONVERSION.

#define **IX_NPEDL_NPEIMAGE_NPEB_DMA**
NPE Image Id for NPE−B with DMA−Only feature.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_**
NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB**
NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXT**
NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FI VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH**

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_FIREWALL**

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS**

NPE Image Id for NPE−C with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV**

NPE Image Id for NPE−C with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS HEADER_CONVERSION.

#define **IX_NPEDL_NPEIMAGE_NPEA_WEP**

NPE Image Id for NPE−A with ARC4 and WEP CRC engines.

#define **IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_ETH_LEARN_FILTER_SPAN_FIREWALL**

NPE Image Id for NPE−C with Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_CCM_ETH**

NPE Image Id for NPE−C with AES and CCM Crypto and Basic Ethernet Rx/Tx.

#define **IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_CCM_EXTSHA_ETH**

NPE Image Id for NPE−C with AES, CCM and all Hashing (incl. SHA−256/384/512) Crypto and Basic Ether Rx/Tx.

#define **IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_ETH_LEARN_FILTER_FIREWALL**

NPE Image Id for NPE−C with AES Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, FIREWALL.

#define **IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_ETH_LEARN_FILTER_SPAN**

NPE Image Id for NPE−C with AES Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE.

#define **IX_NPEDL_NPEIMAGE_NPEC_DMA**

NPE Image Id for NPE−C with DMA−Only feature.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_**

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB**

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXT**

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FI
VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

#define **IX_NPE_MICROCODE_AVAILABLE_VERSIONS_COUNT**

Detailed Description

Library containing a set of NPE firmware images, for use with NPE Downloader s/w component.

Define Documentation

#define IX_NPE_IMAGE_INCLUDE

Wrap the following Image identifiers with "#if IX_NPE_IMAGE_INCLUDE ... #endif" to include the image in the library.

Definition at line **29** of file **IxNpeMicrocode.h**.

#define IX_NPE_IMAGE_OMIT

Wrap the following Image identifiers with "#if IX_NPE_IMAGE_OMIT ... #endif" to OMIT the image from the library.

Definition at line **36** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ATM_MPHY_12_PORT

NPE Image Id for NPE−A with ATM−Only feature. It supports AAL5, AAL0 and OAM for UTOPIA MPHY, 12 logical ports, 32 VCs. It also has Fast Path support.

Definition at line **45** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_DMA

NPE Image Id for NPE−A with DMA−Only feature.

Definition at line **63** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **81** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_HDRCONV_HSSCHAN_COEXIST

NPE Image Id for NPE−A with HSS−0 and Ethernet (with header conversion) feature.

Definition at line **162** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_FIREWALL

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **90** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS

NPE Image Id for NPE−A with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

Definition at line **99** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_EXTMIB

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

Definition at line **126** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_MACFILTERLEARN_HSSCHAN_COEXIST

NPE Image Id for NPE−A with HSS−0 and Ethernet (with MAC filter/learn) feature.

Definition at line **153** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV

NPE Image Id for NPE−A with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS, HEADER_CONVERSION.

Definition at line **108** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **144** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−A with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **135** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_MPHY_4_PORT

NPE Image Id for NPE−A with ATM MPHY 4 Ports and HSS Channelized Service (with HSS Bypass) feature.

Definition at line **171** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_SPHY_1_PORT

NPE Image Id for NPE−A with HSS−0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 packetized. For ATM, it supports AAL5, AAL0 and OAM for UTOPIA SPHY, 1 logical port, 32 VCs. It also has Fast Path support.

Definition at line **54** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_HSS_2_PORT

NPE Image Id for NPE−A with HSS−0 and HSS−1 feature. Each HSS port supports 32 channelized and 4 packetized.

Definition at line **72** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_HSS_TSLOT_SWITCH

NPE Image Id for NPE−A with HSS−0 and timeslot switching feature. It supports 32 channelized and 4 packetized on HSS−0 as well.

Definition at line **117** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEA_WEP

NPE Image Id for NPE−A with ARC4 and WEP CRC engines.

Definition at line **288** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_DMA

NPE Image Id for NPE−B with DMA−Only feature.

Definition at line **216** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH

NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **180** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_FIREWALL

NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **189** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS

NPE Image Id for NPE−B with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

Definition at line **198** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_EXTMIB

NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

Definition at line **225** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV

NPE Image Id for NPE−B with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS, HEADER_CONVERSION.

Definition at line **207** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **243** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEB_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−B with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **234** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_CCM_ETH

NPE Image Id for NPE−C with AES and CCM Crypto and Basic Ethernet Rx/Tx.

Definition at line **306** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_CCM_EXTSHA_ETH

NPE Image Id for NPE−C with AES, CCM and all Hashing (incl. SHA−256/384/512) Crypto and Basic Ethernet Rx/Tx.

Definition at line **315** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_ETH_LEARN_FILTER_FIREWALL

NPE Image Id for NPE−C with AES Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, FIREWALL.

Definition at line **324** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES_ETH_LEARN_FILTER_SPAN

NPE Image Id for NPE−C with AES Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE.

Definition at line **333** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_ETH_LEARN_FILTER_SPAN_FIREWALL

NPE Image Id for NPE−C with Crypto and Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **297** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_DMA

NPE Image Id for NPE−C with DMA−Only feature.

Definition at line **342** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **252** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_FIREWALL

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL.

Definition at line **261** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_FIREWALL_VLAN_QOS

NPE Image Id for NPE−C with Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, FIREWALL, VLAN_QOS.

Definition at line **270** of file **IxNpeMicrocode.h**.

#define

IX_NPEDL_NPEIMAGE_NPEC_ETH_LEARN_FILTER_SPAN_MASK_FIREWALL_VLAN_QOS_EXTMIB

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: MAC_FILTERING, MAC_LEARNING, SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, EXTENDED MIBII.

Definition at line **351** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_FIREWALL_VLAN_QOS_HDR_CONV

NPE Image Id for NPE−C with Ethernet Rx/Tx which includes: SPANNING_TREE, FIREWALL, VLAN_QOS, HEADER_CONVERSION.

Definition at line **279** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_MASK_FIREWALL_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, MASK_BASED_FIREWALL, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **369** of file **IxNpeMicrocode.h**.

#define IX_NPEDL_NPEIMAGE_NPEC_ETH_SPAN_VLAN_QOS_HDR_CONV_EXTMIB

NPE Image Id for NPE−C with Basic Ethernet Rx/Tx which includes: SPANNING_TREE, VLAN_QOS, HEADER_CONVERSION, EXTENDED MIBII.

Definition at line **360** of file **IxNpeMicrocode.h**.

Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API

The public API for the Parity Error Notifier.

Data Structures

struct **IxParityENAccAHBErrorTransaction**
The Master and Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

struct **IxParityENAccAHBErrorTransaction**
The Master and Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

struct **IxParityENAccAHBErrorTransaction**
The Master and Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

struct **IxParityENAccEbcConfig**
Expansion Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccEbcConfig**
Expansion Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccEbcConfig**
Expansion Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccEbcParityErrorStats**
Expansion Bus Controller parity error statistics.

struct **IxParityENAccEbcParityErrorStats**
Expansion Bus Controller parity error statistics.

struct **IxParityENAccEbcParityErrorStats**
Expansion Bus Controller parity error statistics.

struct **IxParityENAccHWParityConfig**
Parity error configuration of the Hardware Blocks.

struct **IxParityENAccHWParityConfig**
Parity error configuration of the Hardware Blocks.

struct **IxParityENAccHWParityConfig**
Parity error configuration of the Hardware Blocks.

struct **IxParityENAccMcuConfig**

MCU pairty detection is to be enabled/disabled.

struct **IxParityENAccMcuConfig**
MCU pairty detection is to be enabled/disabled.

struct **IxParityENAccMcuConfig**
MCU pairty detection is to be enabled/disabled.

struct **IxParityENAccMcuParityErrorStats**
DDR Memory Control Unit parity error statistics.

struct **IxParityENAccMcuParityErrorStats**
DDR Memory Control Unit parity error statistics.

struct **IxParityENAccMcuParityErrorStats**
DDR Memory Control Unit parity error statistics.

struct **IxParityENAccNpeConfig**
NPE parity detection is to be enabled/disabled.

struct **IxParityENAccNpeConfig**
NPE parity detection is to be enabled/disabled.

struct **IxParityENAccNpeConfig**
NPE parity detection is to be enabled/disabled.

struct **IxParityENAccNpeParityErrorStats**
NPE parity error statistics.

struct **IxParityENAccNpeParityErrorStats**
NPE parity error statistics.

struct **IxParityENAccNpeParityErrorStats**
NPE parity error statistics.

struct **IxParityENAccParityErrorContextMessage**
Parity Error Context Message.

struct **IxParityENAccParityErrorContextMessage**
Parity Error Context Message.

struct **IxParityENAccParityErrorContextMessage**
Parity Error Context Message.

struct **IxParityENAccParityErrorStats**
Parity Error Statistics for the all the hardware blocks.

struct **IxParityENAccParityErrorStats**
Parity Error Statistics for the all the hardware blocks.

struct **IxParityENAccParityErrorStats**

Parity Error Statistics for the all the hardware blocks.

struct **IxParityENAccPbcConfig**
PCI Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccPbcConfig**
PCI Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccPbcConfig**
PCI Bus Controller parity detection is to be enabled or disabled.

struct **IxParityENAccPbcParityErrorStats**
PCI Bus Controller parity error statistics.

struct **IxParityENAccPbcParityErrorStats**
PCI Bus Controller parity error statistics.

struct **IxParityENAccPbcParityErrorStats**
PCI Bus Controller parity error statistics.

Typedefs

typedef UINT32 **IxParityENAccParityErrorAddress**
The memory location which has parity error.

typedef UINT32 **IxParityENAccParityErrorData**
The data read from the memory location which has parity error.

typedef void(* **IxParityENAccCallback**)(void)
This prototype shows the format of a callback function.

Enumerations

enum **IxParityENAccStatus** {
IX_PARITYENACC_SUCCESS,
IX_PARITYENACC_INVALID_PARAMETERS,
IX_PARITYENACC_NOT_INITIALISED,
IX_PARITYENACC_ALREADY_INITIALISED,
IX_PARITYENACC_OPERATION_FAILED,
IX_PARITYENACC_NO_PARITY
}
The status as returend from the API.

enum **IxParityENAccParityType** {
IX_PARITYENACC_EVEN_PARITY,
IX_PARITYENACC_ODD_PARITY
}

Odd or Even Parity Type.

enum **IxParityENAccConfigOption** {
IX_PARITYENACC_DISABLE,
IX_PARITYENACC_ENABLE
}
The parity error enable/disable configuration option.

enum **IxParityENAccParityErrorSource** {
IX_PARITYENACC_NPE_A_IMEM,
IX_PARITYENACC_NPE_A_DMEM,
IX_PARITYENACC_NPE_A_EXT,
IX_PARITYENACC_NPE_B_IMEM,
IX_PARITYENACC_NPE_B_DMEM,
IX_PARITYENACC_NPE_B_EXT,
IX_PARITYENACC_NPE_C_IMEM,
IX_PARITYENACC_NPE_C_DMEM,
IX_PARITYENACC_NPE_C_EXT,
IX_PARITYENACC_SWCP,
IX_PARITYENACC_AQM,
IX_PARITYENACC_MCU_SBIT,
IX_PARITYENACC_MCU_MBIT,
IX_PARITYENACC_MCU_OVERFLOW,
IX_PARITYENACC_PBC_INITIATOR,
IX_PARITYENACC_PBC_TARGET,
IX_PARITYENACC_EBC_CS,
IX_PARITYENACC_EBC_EXTMST
}
The source of the parity error notification.

enum **IxParityENAccParityErrorAccess** {
IX_PARITYENACC_READ,
IX_PARITYENACC_WRITE
}
The type of access resulting in parity error.

enum **IxParityENAccParityErrorRequester** {
IX_PARITYENACC_MPI,
IX_PARITYENACC_AHB_BUS
}
The requester interface through which the SDRAM memory access resulted in the parity error.

enum **IxParityENAccAHBErrorMaster** {
IX_PARITYENACC_AHBN_MST_NPE_A,
IX_PARITYENACC_AHBN_MST_NPE_B,
IX_PARITYENACC_AHBN_MST_NPE_C,
IX_PARITYENACC_AHBS_MST_XSCALE,
IX_PARITYENACC_AHBS_MST_PBC,
IX_PARITYENACC_AHBS_MST_EBC,
IX_PARITYENACC_AHBS_MST_AHB_BRIDGE,

IX_PARITYENACC_AHBS_MST_USBH0,
IX_PARITYENACC_AHBS_MST_USBH1
}
The Master on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

enum **IxParityENAccAHBErrorSlave** {
IX_PARITYENACC_AHBN_SLV_MCU,
IX_PARITYENACC_AHBN_SLV_AHB_BRIDGE,
IX_PARITYENACC_AHBS_SLV_MCU,
IX_PARITYENACC_AHBS_SLV_APB_BRIDGE,
IX_PARITYENACC_AHBS_SLV_AQM,
IX_PARITYENACC_AHBS_SLV_RSA,
IX_PARITYENACC_AHBS_SLV_PBC,
IX_PARITYENACC_AHBS_SLV_EBC,
IX_PARITYENACC_AHBS_SLV_USBH0,
IX_PARITYENACC_AHBS_SLV_USBH1
}
The Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

Functions

PUBLIC
IxParityENAccStatus ixParityENAccInit (void)
This function will initialise the IxParityENAcc component.

PUBLIC
IxParityENAccStatus ixParityENAccUnload (void)
This function unloads the IxParityENAcc component.

PUBLIC**ixParityENAccCallbackRegister** (**IxParityENAccCallback**
IxParityENAccStatus parityErrNfyCallBack)
This function will register a new callback with IxParityENAcc component. It can also reregister a new callback replacing the old callback.

PUBLIC**ixParityENAccParityDetectionConfigure** (const
IxParityENAccStatus IxParityENAccHWParityConfig *hwParityConfig)
This interface allows the client application to enable the parity error detection on the underlying hardware block.

PUBLIC**ixParityENAccParityDetectionQuery** (**IxParityENAccHWParityConfig** *const
IxParityENAccStatus hwParityConfig)
This interface allows the client application to determine the status of the parity error detection on the specified hardware blocks.

PUBLIC**ixParityENAccParityErrorContextGet**
IxParityENAccStatus (**IxParityENAccParityErrorContextMessage** *const pecMessage)

This interface allows the client application to determine the status of the parity error context on hardware block for which the current parity error interrupt triggered.

PUBLIC **ixParityENAccParityErrorInterruptClear** (const **IxParityENAccStatus IxParityENAccParityErrorContextMessage** *pecMessage)

This interface helps the client application to clear off the interrupt condition on the hardware block identified in the parity error context message. Please refer to the table below as the operation varies depending on the interrupt source.

PUBLIC **ixParityENAccStatsGet** (**IxParityENAccParityErrorStats** *const **IxParityENAccStatus** ixParityErrorStats)

This interface allows the client application to retrieve parity error statistics for all the hardware blocks.

PUBLIC **IxParityENAccStatus ixParityENAccStatsShow** (void)

This interface allows the client application to print all the parity error statistics.

PUBLIC **IxParityENAccStatus ixParityENAccStatsReset** (void)

This interface allows the client application to reset all the parity error statistics.

PUBLIC **IxParityENAccStatus ixParityENAccParityNPEConfigReUpdate** (UINT32 npeID)

This interface allows the client to restore and update the NPE Error enabling configuration.

PUBLIC **ixParityENAccNPEParityErrorCheck** (UINT32 npeID, **IxParityENAccStatus IxParityENAccParityErrorContextMessage** *const pecMessage)

This interface allows the client to check for any NPE Error status.

Detailed Description

The public API for the Parity Error Notifier.

Typedef Documentation

IxParityENAccCallback

This prototype shows the format of a callback function.

The callback will be used to notify the parity error to the client application. The callback will be registered by **ixParityENAccCallbackRegister**.

It will be called from an ISR when a parity error is detected and thus needs to follow the interrupt callable function conventions.

Definition at line **422** of file **IxParityENAcc.h**.

IxParityENAccParityErrorAddress

The memory location which has parity error.

Definition at line **290** of file **IxParityENAcc.h**.

IxParityENAccParityErrorData

The data read from the memory location which has parity error.

Definition at line **299** of file **IxParityENAcc.h**.

Enumeration Type Documentation

enum IxParityENAccAHBErrorMaster

The Master on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

Enumeration values:

IX_PARITYENACC_AHBN_MST_NPE_A	NPE − A.
IX_PARITYENACC_AHBN_MST_NPE_B	NPE − B.
IX_PARITYENACC_AHBN_MST_NPE_C	NPE − C.
IX_PARITYENACC_AHBS_MST_XSCALE	Intel XScale(R) Processor Bus Interface Unit.
IX_PARITYENACC_AHBS_MST_PBC	PCI Bus Controller.
IX_PARITYENACC_AHBS_MST_EBC	Expansion Bus Controller.
IX_PARITYENACC_AHBS_MST_AHB_BRIDGE	AHB Bridge.
IX_PARITYENACC_AHBS_MST_USBH0	USB Host Controller 0.
IX_PARITYENACC_AHBS_MST_USBH1	USB Host Controller 1 for IXP 43X only.

Definition at line **323** of file **IxParityENAcc.h**.

enum IxParityENAccAHBErrorSlave

The Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

Enumeration values:

IX_PARITYENACC_AHBN_SLV_MCU	Memory Control Unit.
IX_PARITYENACC_AHBN_SLV_AHB_BRIDGE	AHB Bridge.
IX_PARITYENACC_AHBS_SLV_MCU	Intel XScale(R) Processor Bus Interface Unit.
IX_PARITYENACC_AHBS_SLV_APB_BRIDGE	APB Bridge.

IX_PARITYENACC_AHBS_SLV_AQM	AQM.
IX_PARITYENACC_AHBS_SLV_RSA	RSA (Crypto Bus) for IXP 46X only.
IX_PARITYENACC_AHBS_SLV_PBC	PCI Bus Controller.
IX_PARITYENACC_AHBS_SLV_EBC	Expansion Bus Controller.
IX_PARITYENACC_AHBS_SLV_USBH0	USB Host Controller 0.
IX_PARITYENACC_AHBS_SLV_USBH1	USB Host Controller 1 for IXP 43X only.

Definition at line **345** of file **IxParityENAcc.h**.

enum IxParityENAccConfigOption

The parity error enable/disable configuration option.

Enumeration values:

IX_PARITYENACC_DISABLE	Disable parity error detection.
IX_PARITYENACC_ENABLE	Enable parity error detection.

Definition at line **69** of file **IxParityENAcc.h**.

enum IxParityENAccParityErrorAccess

The type of access resulting in parity error.

Enumeration values:

IX_PARITYENACC_READ	Read Access.
IX_PARITYENACC_WRITE	Write Access.

Definition at line **277** of file **IxParityENAcc.h**.

enum IxParityENAccParityErrorRequester

The requester interface through which the SDRAM memory access resulted in the parity error.

Enumeration values:

IX_PARITYENACC_MPI	Direct Memory Port Interface.
IX_PARITYENACC_AHB_BUS	South or North AHB Bus.

Definition at line **309** of file **IxParityENAcc.h**.

enum IxParityENAccParityErrorSource

The source of the parity error notification.

Enumeration values:

IX_PARITYENACC_NPE_A_IMEM	NPE A − Instruction memory.
IX_PARITYENACC_NPE_A_DMEM	NPE A − Data memory.
IX_PARITYENACC_NPE_A_EXT	NPE A − External Entity.
IX_PARITYENACC_NPE_B_IMEM	NPE B − Instruction memory.
IX_PARITYENACC_NPE_B_DMEM	NPE B − Data memory.
IX_PARITYENACC_NPE_B_EXT	NPE B − External Entity.
IX_PARITYENACC_NPE_C_IMEM	NPE C − Instruction memory.
IX_PARITYENACC_NPE_C_DMEM	NPE C − Data memory.
IX_PARITYENACC_NPE_C_EXT	NPE C − External Entity.
IX_PARITYENACC_SWCP	SWCP.
IX_PARITYENACC_AQM	AQM.
IX_PARITYENACC_MCU_SBIT	DDR Memory Controller Unit − Single bit parity.
IX_PARITYENACC_MCU_MBIT	DDR Memory Controller Unit − Multi bit parity.
IX_PARITYENACC_MCU_OVERFLOW	DDR Memory Controller Unit − Parity errors in excess of two.
IX_PARITYENACC_PBC_INITIATOR	PCI Bus Controller as Initiator.
IX_PARITYENACC_PBC_TARGET	PCI Bus Controller as Target.
IX_PARITYENACC_EBC_CS	Expansion Bus Controller − Chip Select.
IX_PARITYENACC_EBC_EXTMST	Expansion Bus Controller − External Master.

Definition at line **248** of file **IxParityENAcc.h**.

enum IxParityENAccParityType

Odd or Even Parity Type.

Enumeration values:

IX_PARITYENACC_EVEN_PARITY	Even Parity.
IX_PARITYENACC_ODD_PARITY	Odd Parity.

Definition at line **56** of file **IxParityENAcc.h**.

enum IxParityENAccStatus

The status as returend from the API.

Enumeration values:

IX_PARITYENACC_SUCCESS	The request is successful.
IX_PARITYENACC_INVALID_PARAMETERS	Invalid or NULL parameters passed.
IX_PARITYENACC_NOT_INITIALISED	Access layer has not been initialised before accessing the APIs.

IX_PARITYENACC_ALREADY_INITIALISED	Access layer has already been initialised.
IX_PARITYENACC_OPERATION_FAILED	Operation did not succeed due to hardware failure.
IX_PARITYENACC_NO_PARITY	No parity condition exits or has already been cleared.

Definition at line **39** of file **IxParityENAcc.h**.

Function Documentation

IxParityENAccStatus (**IxParityENAccCallback** *parityErrNfyCallBack*)
ixParityENAccCallbackRegister

This function will register a new callback with IxParityENAcc component. It can also reregister a new callback replacing the old callback.

Parameters:

parityErrNfyCallBack [in] − This parameter will specify the call−back function supplied by the client application.

This interface registers the user application supplied call−back handler with the parity error handling access component after the init.

The callback function will be called from an ISR that will be triggered by the parity error in the IXP400 silicon.

The following actions will be performed by this function:

1. Check for the prior initialisation of the module before registering or re−registering of the callback. Check for parity error detection disabled before re−registration of the callback.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_PARITYENACC_SUCCESS − The parameters check passed and the registration is successful.
◊ IX_PARITYENACC_INVALID_PARAMETERS − Request failed due to NULL parameter passed.
◊ IX_PARITYENACC_OPERATION_FAILED − The request failed because the parity error detection not yet disabled.
◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccInit (void)

This function will initialise the IxParityENAcc component.

This function will initialise the IxParityENAcc component. It should only be called once, prior to using the IxParityENAcc component.

1. It initialises the internal data structures, registers the ISR that will be triggered when a parity error occurs in the supported Intel (R) IXP4XX Product Line of Network processors.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_PARITYENACC_SUCCESS − Initialization is successful
◊ IX_PARITYENACC_ALREADY_INITIALISED − The access layer has already been initialized
◊ IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware. Refer to error trace/log for details.

IxParityENAccStatus ixParityENAccNPEParityErrorCheck (UINT32 *npeID*, **IxParityENAccParityErrorContextMessage** *const *pecMessage*)

This interface allows the client to check for any NPE Error status.

- Re−entrant : No
- ISR Callable : Yes

Parameters:

npeID UINT32 [in] NPE ID, NPE−A (0) , NPE−B (1) and NPE−C (2)
pecMessage **IxParityENAccParityErrorContextMessage** [out]

Returns:

- IX_PARITYENACC_SUCCESS− Existance of an NPE Parity Error
- IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to NULL parameter is passed
- IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware.
- IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.
- IX_PARITYENACC_NO_PARITY − No parity condition exits or has already been cleared

IxParityENAccStatus ixParityENAccParityDetectionConfigure (const **IxParityENAccHWParityConfig** * *hwParityConfig*)

This interface allows the client application to enable the parity error detection on the underlying hardware block.

Parameters:

hwParityConfig [in] − Hardware blocks for which the parity error detection is to be enabled or disabled.

The client application allocates and provides the reference to the buffer.

It will also verify whether the specific hardware block is functional or not.

NOTE: Failure in enabling or disabling of one or more components result in trace message but still returns IX_PARITYENACC_SUCCESS. Refer to the function **ixParityENAccParityDetectionQuery** on how to verify the failures while enabling/disabling paritys error detection.

It shall be invoked after the Init and CallbackRegister functions but before any other function of the IxParityENAcc layer.

- Re−entrant : No
- ISR Callable : No

Returns:

- ◊ IX_PARITYENACC_SUCCESS − The parameters check passed and the request to enable/disable is successful.
- ◊ IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to NULL parameter supplied.
- ◊ IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccParityDetectionQuery (**IxParityENAccHWParityConfig** *const *hwParityConfig*)

This interface allows the client application to determine the status of the parity error detection on the specified hardware blocks.

Parameters:

hwParityConfig [out] − Hardware blocks whose parity error detection has been enabled or disabled.

The client application allocates and provides the reference to the buffer.

This interface can be used immediately after the interface **ixParityENAccParityDetectionConfigure** to see on which of the hardware blocks the parity error detection has either been enabled or disabled based on the client application request.

- Re−entrant : No
- ISR Callable : No

Returns:

- ◊ IX_PARITYENACC_SUCCESS − The parameters check passed and the request to query on whether the hardware parity error detection is enabled or disabled is successful.
- ◊ IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to NULL parameter or invalid values supplied.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the

initialisation of the access layer.

IxParityENAccStatus (**IxParityENAccParityErrorContextMessage** *pecMessage*)
ixParityENAccParityErrorContextGet *const

This interface allows the client application to determine the status of the parity error context on hardware block for which the current parity error interrupt triggered.

Parameters:

pecMessage [out] − The parity error context information of the parity interrupt currently being process.

The client application allocates and provides the reference to the buffer.

Refer to the data structure **IxParityENAccParityErrorContextMessage** for details.

The routine will will fetch the parity error context in the following priority, if multiple parity errors observed.

```
0 − MCU (Multi−bit and single−bit in that order)
1 − NPE−A
2 − NPE−B
3 − NPE−C
4 − SWCP
5 − QM
6 − PCI
7 − EXP
```

```
NOTE: The information provided in the IxParityENAccAHBErrorTransaction
may be of help for the client application to decide on the course of action
to take. This info is taken from the Performance Monitoring Unit register
which records most recent error observed on the AHB bus. This information
might have been overwritten by some other error by the time it is retrieved.
```

- Re−entrant : No
- ISR Callable : Yes

Returns:

◊ IX_PARITYENACC_SUCCESS−The parameters check passed and the request to get the parity error context information is successful.
◊ IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to NULL parameter is passed
◊ IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware.
◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.
◊ IX_PARITYENACC_NO_PARITY − No parity condition exits or has already been cleared

IxParityENAccStatus (const *pecMessage*)
ixParityENAccParityErrorInterruptClear **IxParityENAccParityErrorContextMessage**

*

This interface helps the client application to clear off the interrupt condition on the hardware block identified in the parity error context message. Please refer to the table below as the operation varies depending on the interrupt source.

Parameters:

pecMessage [in] − The parity error context information of the hardware block whose parity error interrupt condition is to disabled.

The client application allocates and provides the reference to the buffer.

```
*****************************************************************************
Following actions will be taken during the interrupt clear for respective
hardware blocks.

 Parity Source     Actions taken during Interrupt clear
 -------------     ------------------------------------------------------
 NPE-A             Interrupt will be masked off at the interrupt controller
                   so that it will not trigger continuously.
                     Client application has to take appropriate action and
                     re-configure the parity error detection subsequently.
                     The client application will not be notified of further
                     interrupts, until the re-configuration is done using
                     ixParityENAccParityDetectionConfigure.

 NPE-B             Interrupt will be masked off at the interrupt controller
                   so that it will not trigger continuously.
                     Client application has to take appropriate action and
                     re-configure the parity error detection subsequently.
                     The client application will not be notified of further
                     interrupts, until the re-configuration is done using
                     ixParityENAccParityDetectionConfigure.

 NPE-C             Interrupt will be masked off at the interrupt controller
                     Client application has to take appropriate action and
                     re-configure the parity error detection subsequently.
                     The client application will not be notified of further
                     interrupts, until the re-configuration is done using
                     ixParityENAccParityDetectionConfigure.

 SWCP              Interrupt will be masked off at the interrupt controller.
                     Client application has to take appropriate action and
                     re-configure the parity error detection subsequently.
                     The client application will not be notified of further
                     interrupts, until the re-configuration is done using
                     ixParityENAccParityDetectionConfigure.

 AQM               Interrupt will be masked off at the interrupt controller.
                        Client application has to take appropriate action and
                        re-configure the parity error detection subsequently.
                        The client application will not be notified of further
                        interrupts, until the re-configuration is done using
                        ixParityENAccParityDetectionConfigure.

 MCU               Parity interrupt condition is cleared at the SDRAM MCU for
                   the following:
                   1. Single-bit
```

```
                    2. Multi−bit
                    3. Overflow condition i.e., more than two parity conditions
                       occurred
                    Note that single−parity errors do not result in data abort
                    and not all data aborts caused by multi−bit parity error.

  PCI               Interrupt condition is cleared at the PCI bus controller.

  EXP               Parity interrupt condition is cleared at the expansion bus
                    controller for the following:
                    1. External master initiated Inbound write
                    2. Internal master (IXP400) initiated Outbound read
 ***************************************************************************
```

- Re−entrant : No
- ISR Callable : No

Returns:

- ◊ IX_PARITYENACC_SUCCESS−The parameters check passed and the request to clear the parity error interrupt condition is successful.
- ◊ IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to NULL parameters have been passed or contents have been supplied with invalid values.
- ◊ IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccParityNPEConfigReUpdate (UINT32 *npeID*)

This interface allows the client to restore and update the NPE Error enabling configuration.

- Re−entrant : No
- ISR Callable : No

Parameters:

npeID − [in] NPE ID, NPE−A (0) , NPE−B (1) and NPE−C (2)

Returns:

- IX_PARITYENACC_SUCCESS − The request to Restore/Re−update the parity error control enable bits in the NPE is a success.
- IX_PARITYENACC_OPERATION_FAILED − The request failed because the operation didn't succeed on the hardware.
- IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccStatsGet (**IxParityENAccParityErrorStats** *const* *ixParityErrorStats*)

This interface allows the client application to retrieve parity error statistics for all the hardware blocks.

Parameters:

ixParityErrorStats − [out] The statistics for all the hardware blocks.

The client application allocates and provides the reference to the buffer.

- Re−entrant : No
- ISR Callable : Yes

Returns:

- ◊ IX_PARITYENACC_SUCCESS−The parameters check passed and the request to retrieve parity error statistics for the hardware block is successful.
- ◊ IX_PARITYENACC_INVALID_PARAMETERS−The request failed due to a NULL parameter passed.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccStatsReset (void)

This interface allows the client application to reset all the parity error statistics.

- Re−entrant : No
- ISR Callable : No

Returns:

- ◊ IX_PARITYENACC_SUCCESS − The request to reset the parity error statistics is successful.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccStatsShow (void)

This interface allows the client application to print all the parity error statistics.

- Re−entrant : No
- ISR Callable : No

Returns:

- ◊ IX_PARITYENACC_SUCCESS − The request to show the pairty error statistics is successful.
- ◊ IX_PARITYENACC_NOT_INITIALISED − The operation requested prior to the initialisation of the access layer.

IxParityENAccStatus ixParityENAccUnload (void)

This function unloads the IxParityENAcc component.

This function unloads the IxParityENAcc component. It disables all error detections, unbind all IRQs and disables the enabled interrupts. It can be called only after ixParityENAccInit.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_PARITYENACC_SUCCESS − Unload is successful
◊ IX_PARITYENACC_NOT_INITIALISED − The access layer has not been initialized

Intel (R) IXP400 Software Queue Manager (IxQMgr) API

The public API for the IXP400 qmgr component.

Defines

#define **IX_QMGR_QUELOW_GROUP**
Backward compatability for QMgr queue group select identifier, IX_QMGR_QUELOW_GROUP.

#define **IX_QMGR_QUEUPP_GROUP**
Backward compatability for QMgr queue group select identifier, IX_QMGR_QUEUPP_GROUP.

Typedefs

typedef void(* **IxQMgrDispatcherFuncPtr**)(**IxQMgrDispatchGroup** group)
QMgr Dispatcher Loop function pointer.

Enumerations

enum **IxQMgrDispatcherState** {
IX_QMGR_DISPATCHER_LOOP_FREE,
IX_QMGR_DISPATCHER_LOOP_BUSY
}
Queue dispatcher State.

enum **IxQMgrDispatchGroup** {
IX_QMGR_GROUP_Q0_TO_Q31,
IX_QMGR_GROUP_Q32_TO_Q63,
IX_QMGR_GROUP_MAX
}
QMgr queue group select identifiers.

Functions

PUBLIC IX_STATUS **ixQMgrInit** (void)
Initialise the QMgr.

PUBLIC IX_STATUS **ixQMgrUnload** (void)
Uninitialise the QMgr.

PUBLIC void **ixQMgrShow** (void)
Describe queue configuration and statistics for active queues.

PUBLIC void **ixQMgrDispatcherLoopGet** (**IxQMgrDispatcherFuncPtr** *qDispatcherFuncPtr)
Get QMgr DispatcherLoopRun for respective silicon device.

PUBLIC void **ixQMgrDispatcherLoopEnable** (void)
Enables the Queue Manager Dispatcher.

PUBLIC void **ixQMgrDispatcherLoopDisable** (void)
Disables the Queue Manager Dispatcher.

PUBLIC **IxQMgrDispatcherState ixQMgrDispatcherLoopStatusGet** (void)
Gets the Queue dispatcher Loop status.

PUBLIC void **ixQMgrDispatcherInterruptModeSet** (BOOL mode)
Notifies the ixQMgr that the Dispatcher is in Interrupt mode.

Detailed Description

The public API for the IXP400 qmgr component.

IxQMgr is a low level public interface to the Queue Manager

Define Documentation

#define IX_QMGR_QUELOW_GROUP

Backward compatability for QMgr queue group select identifier, IX_QMGR_QUELOW_GROUP.

This #define provides backward compatability for the IX_QMGR_QUELOW_GROUP, which has been replaced by IX_QMGR_GROUP_Q0_TO_Q31.

This #define has been DEPRECATED and will be removed in a future release.

Definition at line **84** of file **IxQMgr.h**.

#define IX_QMGR_QUEUPP_GROUP

Backward compatability for QMgr queue group select identifier, IX_QMGR_QUEUPP_GROUP.

This #define provides backward compatability for the IX_QMGR_QUEUPP_GROUP, which has been replaced by IX_QMGR_GROUP_Q32_TO_Q63.

This #define has been DEPRECATED and will be removed in a future release.

Definition at line **100** of file **IxQMgr.h**.

Typedef Documentation

IxQMgrDispatcherFuncPtr

QMgr Dispatcher Loop function pointer.

This defines the interface for QMgr Dispather functions.

Parameters:

group **IxQMgrDispatchGroup** [in] − the group of the queue of which the dispatcher will run

Definition at line **114** of file **IxQMgr.h**.

Enumeration Type Documentation

enum IxQMgrDispatcherState

Queue dispatcher State.

Indicates the Queue dispatcher state busy or free.

Definition at line **47** of file **IxQMgr.h**.

enum IxQMgrDispatchGroup

QMgr queue group select identifiers.

This enum defines the groups into which the queues are divided. Each group contains 32 queues. Each qmgr interface function that works on a group of queues takes one of these identifiers as a parameter.

Enumeration values:

IX_QMGR_GROUP_Q0_TO_Q31 Queues 0−31.

IX_QMGR_GROUP_Q32_TO_Q63 Queues 32−63.

Definition at line **63** of file **IxQMgr.h**.

Function Documentation

void ixQMgrDispatcherInterruptModeSet (BOOL *mode*)

Notifies the ixQMgr that the Dispatcher is in Interrupt mode.

Note:

This function records whether the Queue Dispatcher is bind to the QM Interrupt notification IRQ.

Parameters:

BOOL mode [in] − TRUE − dispatcher mode is bind to Queue Manager IRQ for queue notifications (0−63). FALSE − dispatcher is in poll mode.

Returns:

None

ixQMgrDispatcherLoopDisable (void)

Disables the Queue Manager Dispatcher.

Note:

API used by the ixErrHdlAcc to disable the queue manager dispatcher to prevent QM Read/Write access.

Returns:

None

ixQMgrDispatcherLoopEnable (void)

Enables the Queue Manager Dispatcher.

Note:

API used by the ixErrHdlAcc to re−enable the queue manager dispatcher.

Returns:

None

ixQMgrDispatcherLoopGet (**IxQMgrDispatcherFuncPtr** * *qDispatcherFuncPtr*)

Get QMgr DispatcherLoopRun for respective silicon device.

This function gets a function pointer to ixQMgrDispatcherLoopRunB0() for IXP42X B0 Silicon. However if live lock prevention is enabled a function pointer to ixQMgrDispatcherLoopRunB0LLP() is given.

Parameters:

qDispatchFuncPtr* **IxQMgrDispatcherFuncPtr [out] − the function pointer of QMgr Dispatcher

IxQMgrDispatcherState ixQMgrDispatcherLoopStatusGet (void)

Gets the Queue dispatcher Loop status.

Returns:

Returns an enum that indicates the dispatcher state (RUN or STOP)

ixQMgrInit (void)

Initialise the QMgr.

This function must be called before any other QMgr function. It sets up internal data structures.

Returns:

◊ IX_SUCCESS, the IxQMgr successfully initialised

◊ IX_FAIL, failed to initialize the Qmgr

ixQMgrShow (void)

Describe queue configuration and statistics for active queues.

This function shows active queues, their configurations and statistics.

Returns:

◊ void

ixQMgrUnload (void)

Uninitialise the QMgr.

This function will perform the tasks required to unload the QMgr component cleanly. This includes unmapping kernel memory and unconfigures the access layer component's queue configuration.

This should be called before a soft reboot or unloading of a kernel module.

Precondition:

It should only be called if **ixQMgrInit** has already been called.

Postcondition:

No QMgr functions should be called until ixQMgrInit is called again.

Returns:

◊ IX_SUCCESS, the IxQMgr successfully uninitialised

◊ IX_FAIL, failed to uninitialize the Qmgr

Intel (R) IXP400 Software Queue Assignments

Queue Assignments used by Ethernet, HSS, ATM and DMA access libraries. Ethernet QoS traffic class definitions are mapped by EthDB in **IxEthDBQoS.h**.

Defines

#define **IX_NPE_MPHYMULTIPORT**

Global compiler switch to select between 3 possible NPE Modes Define this macro to enable MPHY mode.

#define **IX_NPE_A_TXDONE_QUEUE_HIGHWATERMARK**

The NPE reserves the High Watermark for its operation. But it must be set by the Intel XScale(R) processor.

#define **IX_NPE_A_QMQ_ATM_TX_DONE**

Queue ID for ATM Transmit Done queue.

#define **IX_NPE_A_QMQ_ATM_TX0**

Queue ID for ATM transmit Queue in a single phy configuration.

#define **IX_NPE_A_QMQ_ATM_TXID_MIN**

Queue Manager Queue ID for ATM transmit Queue with minimum number of queue.

#define **IX_NPE_A_QMQ_ATM_TXID_MAX**

Queue Manager Queue ID for ATM transmit Queue with maximum number of queue.

#define **IX_NPE_A_QMQ_ATM_RX_HI**

Queue Manager Queue ID for ATM Receive high Queue.

#define **IX_NPE_A_QMQ_ATM_RX_LO**

Queue Manager Queue ID for ATM Receive low Queue.

#define **IX_NPE_A_QMQ_ATM_TX1**

Queue ID for ATM transmit Queue Multiphy from 1 to 11.

#define **IX_NPE_A_QMQ_ATM_FREE_VC0**

Hardware QMgr Queue ID for ATM Free VC Queue.

#define **IX_NPE_A_QMQ_ATM_RXFREE_MIN**

The minimum queue ID for FreeVC queue.

#define **IX_NPE_A_QMQ_OAM_FREE_VC**

OAM Rx Free queue ID.

#define **IX_NPE_A_QMQ_HSS0_CHL_RX_TRIG**

Hardware QMgr Queue ID for HSS Port 0 Channelized Receive trigger.

#define **IX_NPE_A_QMQ_HSS0_PKT_RX**
Hardware QMgr Queue ID for HSS Port 0 Packetized Receive.

#define **IX_NPE_A_QMQ_HSS0_PKT_TX0**
Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 0.

#define **IX_NPE_A_QMQ_HSS0_PKT_TX1**
Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 1.

#define **IX_NPE_A_QMQ_HSS0_PKT_TX2**
Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 2.

#define **IX_NPE_A_QMQ_HSS0_PKT_TX3**
Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 3.

#define **IX_NPE_A_QMQ_HSS0_PKT_RX_FREE0**
Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 0.

#define **IX_NPE_A_QMQ_HSS0_PKT_RX_FREE1**
Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 1.

#define **IX_NPE_A_QMQ_HSS0_PKT_RX_FREE2**
Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 2.

#define **IX_NPE_A_QMQ_HSS0_PKT_RX_FREE3**
Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 3.

#define **IX_NPE_A_QMQ_HSS0_PKT_TX_DONE**
Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit Done queue.

#define **IX_NPE_A_QMQ_HSS1_CHL_RX_TRIG**
Hardware QMgr Queue ID for HSS Port 1 Channelized Receive trigger.

#define **IX_NPE_A_QMQ_HSS1_PKT_RX**
Hardware QMgr Queue ID for HSS Port 1 Packetized Receive.

#define **IX_NPE_A_QMQ_HSS1_PKT_TX0**
Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 0.

#define **IX_NPE_A_QMQ_HSS1_PKT_TX1**
Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 1.

#define **IX_NPE_A_QMQ_HSS1_PKT_TX2**
Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 2.

#define **IX_NPE_A_QMQ_HSS1_PKT_TX3**
Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 3.

#define **IX_NPE_A_QMQ_HSS1_PKT_RX_FREE0**
Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 0.

#define **IX_NPE_A_QMQ_HSS1_PKT_RX_FREE1**
Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 1.

#define **IX_NPE_A_QMQ_HSS1_PKT_RX_FREE2**
Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 2.

#define **IX_NPE_A_QMQ_HSS1_PKT_RX_FREE3**
Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 3.

#define **IX_NPE_A_QMQ_HSS1_PKT_TX_DONE**
Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit Done queue.

#define **IX_ETH_ACC_RX_FREE_NPEA_Q**
Supply Rx Buffers Ethernet Q for NPEA.

#define **IX_ETH_ACC_RX_FREE_NPEB_Q**
Supply Rx Buffers Ethernet Q for NPEB (for single port images only).

#define **IX_ETH_ACC_RX_FREE_NPEC_Q**
Supply Rx Buffers Ethernet Q for NPEC.

#define **IX_ETH_ACC_TX_NPEA_Q**
Submit frame Q for NPEA.

#define **IX_ETH_ACC_TX_NPEB_Q**
Submit frame Q for NPEB.

#define **IX_ETH_ACC_TX_NPEC_Q**
Submit frame Q for NPEC.

#define **IX_ETH_ACC_TX_DONE_Q**
Transmit complete Q for all NPEs.

Detailed Description

Queue Assignments used by Ethernet, HSS, ATM and DMA access libraries. Ethernet QoS traffic class definitions are mapped by EthDB in **IxEthDBQoS.h**.

Define Documentation

```
#define IX_ETH_ACC_RX_FREE_NPEA_Q
```

Supply Rx Buffers Ethernet Q for NPEA.

Definition at line **511** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_RX_FREE_NPEB_Q

Supply Rx Buffers Ethernet Q for NPEB (for single port images only).

Definition at line **522** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_RX_FREE_NPEC_Q

Supply Rx Buffers Ethernet Q for NPEC.

Definition at line **533** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_TX_DONE_Q

Transmit complete Q for all NPEs.

Definition at line **578** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_TX_NPEA_Q

Submit frame Q for NPEA.

Definition at line **544** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_TX_NPEB_Q

Submit frame Q for NPEB.

Definition at line **556** of file **IxQueueAssignments.h**.

#define IX_ETH_ACC_TX_NPEC_Q

Submit frame Q for NPEC.

Definition at line **567** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_FREE_VC0

Hardware QMgr Queue ID for ATM Free VC Queue.

There are 32 Hardware QMgr Queue ID; from IX_NPE_A_QMQ_ATM_FREE_VC1 to IX_NPE_A_QMQ_ATM_FREE_VC30

Definition at line **172** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_RX_HI

Queue Manager Queue ID for ATM Receive high Queue.

Definition at line **152** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_RX_LO

Queue Manager Queue ID for ATM Receive low Queue.

Definition at line **153** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_RXFREE_MIN

The minimum queue ID for FreeVC queue.

Definition at line **213** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_TX0

Queue ID for ATM transmit Queue in a single phy configuration.

Definition at line **88** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_TX1

Queue ID for ATM transmit Queue Multiphy from 1 to 11.

Definition at line **139** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_TX_DONE

Queue ID for ATM Transmit Done queue.

Definition at line **78** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_TXID_MAX

Queue Manager Queue ID for ATM transmit Queue with maximum number of queue.

Definition at line **151** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_ATM_TXID_MIN

Queue Manager Queue ID for ATM transmit Queue with minimum number of queue.

Definition at line **150** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_CHL_RX_TRIG

Hardware QMgr Queue ID for HSS Port 0 Channelized Receive trigger.

Definition at line **250** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_RX

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive.

Definition at line **275** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE0

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 0.

Definition at line **325** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE1

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 1.

Definition at line **335** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE2

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 2.

Definition at line **345** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE3

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 3.

Definition at line **355** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_TX0

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 0.

Definition at line **285** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_TX1

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 1.

Definition at line **295** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_TX2

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 2.

Definition at line **305** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_TX3

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 3.

Definition at line **315** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS0_PKT_TX_DONE

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit Done queue.

Definition at line **365** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_CHL_RX_TRIG

Hardware QMgr Queue ID for HSS Port 1 Channelized Receive trigger.

Definition at line **377** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_RX

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive.

Definition at line **387** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE0

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 0.

Definition at line **437** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE1

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 1.

Definition at line **447** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE2

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 2.

Definition at line **457** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE3

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 3.

Definition at line **467** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_TX0

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 0.

Definition at line **397** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_TX1

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 1.

Definition at line **407** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_TX2

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 2.

Definition at line **417** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_TX3

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 3.

Definition at line **427** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_HSS1_PKT_TX_DONE

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit Done queue.

Definition at line **477** of file **IxQueueAssignments.h**.

#define IX_NPE_A_QMQ_OAM_FREE_VC

OAM Rx Free queue ID.

Definition at line **230** of file **IxQueueAssignments.h**.

#define IX_NPE_A_TXDONE_QUEUE_HIGHWATERMARK

The NPE reserves the High Watermark for its operation. But it must be set by the Intel XScale(R) processor.

Definition at line **68** of file **IxQueueAssignments.h**.

#define IX_NPE_MPHYMULTIPORT

Global compiler switch to select between 3 possible NPE Modes Define this macro to enable MPHY mode.

Default(No Switch) = MultiPHY Utopia2 Define IX_UTOPIAMODE for single Phy Utopia1 Define IX_MPHYSINGLEPORT for single Phy Utopia2

Definition at line **50** of file **IxQueueAssignments.h**.

Intel (R) IXP400 Software SSP Serial Port Access (IxSspAcc) API

Intel (R) IXP400 Software SSP Serial Port Access Public API.

Data Structures

struct **IxSspAccStatsCounters**
contains counters of the SSP statistics

struct **IxSspAccStatsCounters**
contains counters of the SSP statistics

struct **IxSspInitVars**
contains all the variables required to initialize the SSP serial port hardware.

struct **IxSspInitVars**
contains all the variables required to initialize the SSP serial port hardware.

Typedefs

typedef void(* **RxFIFOOverrunHandler**)(void)
SSP Rx FIFO Overrun handler.

typedef void(* **RxFIFOThresholdHandler**)(void)
SSP Rx FIFO Threshold hit or above handler.

typedef void(* **TxFIFOThresholdHandler**)(void)
SSP Tx FIFO Threshold hit or below handler.

Enumerations

enum **IxSspAccDataSize** {
DATA_SIZE_TOO_SMALL,
DATA_SIZE_4,
DATA_SIZE_5,
DATA_SIZE_6,
DATA_SIZE_7,
DATA_SIZE_8,
DATA_SIZE_9,
DATA_SIZE_10,
DATA_SIZE_11,
DATA_SIZE_12,
DATA_SIZE_13,

DATA_SIZE_14,
DATA_SIZE_15,
DATA_SIZE_16,
DATA_SIZE_TOO_BIG
}
The data sizes in bits that are supported by the protocol.

enum **IxSspAccPortStatus** {
SSP_PORT_DISABLE,
SSP_PORT_ENABLE,
INVALID_SSP_PORT_STATUS
}
The status of the SSP port to be set to enable/disable.

enum **IxSspAccFrameFormat** {
SPI_FORMAT,
SSP_FORMAT,
MICROWIRE_FORMAT,
INVALID_FORMAT
}
The frame format that is to be used − SPI, SSP, or Microwire.

enum **IxSspAccClkSource** {
ON_CHIP_CLK,
EXTERNAL_CLK,
INVALID_CLK_SOURCE
}
The source to produce the SSP serial clock.

enum **IxSspAccSpiSclkPhase** {
START_ONE_END_HALF,
START_HALF_END_ONE,
INVALID_SPI_PHASE
}
The SPI SCLK Phase: 0 − SCLK is inactive one cycle at the start of a frame and 1/2 cycle at the end of a frame. 1 − SCLK is inactive 1/2 cycle at the start of a frame and one cycle at the end of a frame.

enum **IxSspAccSpiSclkPolarity** {
SPI_POLARITY_LOW,
SPI_POLARITY_HIGH,
INVALID_SPI_POLARITY
}
The SPI SCLK Polarity can be set to either low or high.

enum **IxSspAccMicrowireCtlWord** {
MICROWIRE_8_BIT,
MICROWIRE_16_BIT,
INVALID_MICROWIRE_CTL_WORD
}

The Microwire control word can be either 8 or 16 bit.

enum **IxSspAccFifoThreshold** {
FIFO_TSHLD_1,
FIFO_TSHLD_2,
FIFO_TSHLD_3,
FIFO_TSHLD_4,
FIFO_TSHLD_5,
FIFO_TSHLD_6,
FIFO_TSHLD_7,
FIFO_TSHLD_8,
FIFO_TSHLD_9,
FIFO_TSHLD_10,
FIFO_TSHLD_11,
FIFO_TSHLD_12,
FIFO_TSHLD_13,
FIFO_TSHLD_14,
FIFO_TSHLD_15,
FIFO_TSHLD_16,
INVALID_FIFO_TSHLD
}
The threshold in frames (each frame is defined by IxSspAccDataSize) that can be set for the FIFO to trigger a threshold exceed when checking with the ExceedThresholdCheck functions or an interrupt when it is enabled.

enum **IX_SSP_STATUS** {
IX_SSP_SUCCESS,
IX_SSP_FAIL,
IX_SSP_RX_FIFO_OVERRUN_HANDLER_MISSING,
IX_SSP_RX_FIFO_HANDLER_MISSING,
IX_SSP_TX_FIFO_HANDLER_MISSING,
IX_SSP_FIFO_NOT_EMPTY_FOR_SETTING_CTL_CMD,
IX_SSP_INVALID_FRAME_FORMAT_ENUM_VALUE,
IX_SSP_INVALID_DATA_SIZE_ENUM_VALUE,
IX_SSP_INVALID_CLOCK_SOURCE_ENUM_VALUE,
IX_SSP_INVALID_TX_FIFO_THRESHOLD_ENUM_VALUE,
IX_SSP_INVALID_RX_FIFO_THRESHOLD_ENUM_VALUE,
IX_SSP_INVALID_SPI_PHASE_ENUM_VALUE,
IX_SSP_INVALID_SPI_POLARITY_ENUM_VALUE,
IX_SSP_INVALID_MICROWIRE_CTL_CMD_ENUM_VALUE,
IX_SSP_INT_UNBIND_FAIL,
IX_SSP_INT_BIND_FAIL,
IX_SSP_RX_FIFO_NOT_EMPTY,
IX_SSP_TX_FIFO_NOT_EMPTY,
IX_SSP_POLL_MODE_BLOCKING,
IX_SSP_TX_FIFO_HIT_BELOW_THRESHOLD,
IX_SSP_TX_FIFO_EXCEED_THRESHOLD,
IX_SSP_RX_FIFO_HIT_ABOVE_THRESHOLD,
IX_SSP_RX_FIFO_BELOW_THRESHOLD,
IX_SSP_BUSY,
IX_SSP_IDLE,

IX_SSP_OVERRUN_OCCURRED,
IX_SSP_NO_OVERRUN,
IX_SSP_NOT_SUPORTED,
IX_SSP_NOT_INIT,
IX_SSP_NULL_POINTER
}
The statuses that can be returned in a SSP Serial Port Access.

Functions

PUBLIC
IX_SSP_STATUS ixSspAccInit (**IxSspInitVars** *initVarsSelected)
Initializes the SSP Access module.

PUBLIC
IX_SSP_STATUS ixSspAccUninit (void)
Un−initializes the SSP Serial Port Access component.

PUBLIC
IX_SSP_STATUS ixSspAccFIFODataSubmit (UINT16 *data, UINT32 amtOfData)
Inserts data into the SSP Serial Port's FIFO.

PUBLIC
IX_SSP_STATUS ixSspAccFIFODataReceive (UINT16 *data, UINT32 amtOfData)
Extract data from the SSP Serial Port's FIFO.

PUBLIC
IX_SSP_STATUS ixSspAccTxFIFOHitOrBelowThresholdCheck (void)
Check if the Tx FIFO threshold has been hit or fallen below.

PUBLIC
IX_SSP_STATUS ixSspAccRxFIFOHitOrAboveThresholdCheck (void)
Check if the Rx FIFO threshold has been hit or exceeded.

PUBLIC
IX_SSP_STATUS ixSspAccSSPPortStatusSet (**IxSspAccPortStatus** portStatusSelected)
Enables/disables the SSP Serial Port hardware.

PUBLIC
IX_SSP_STATUS ixSspAccFrameFormatSelect (**IxSspAccFrameFormat** frameFormatSelected)
Sets the frame format for the SSP Serial Port hardware.

PUBLIC
IX_SSP_STATUS ixSspAccDataSizeSelect (**IxSspAccDataSize** dataSizeSelected)
Sets the data size for transfers.

PUBLIC
IX_SSP_STATUS ixSspAccClockSourceSelect (**IxSspAccClkSource** clkSourceSelected)

Sets the clock source of the SSP Serial Port hardware.

PUBLIC
IX_SSP_STATUS ixSspAccSerialClockRateConfigure (UINT8 serialClockRateSelected)
Sets the on−chip Serial Clock Rate of the SSP Serial Port hardware.

PUBLIC
IX_SSP_STATUS ixSspAccRxFIFOIntEnable (**RxFIFOThresholdHandler** rxFIFOIntrHandler)
Enables service request interrupt whenever the Rx FIFO hits its threshold.

PUBLIC
IX_SSP_STATUS ixSspAccRxFIFOIntDisable (void)
Disables service request interrupt of the Rx FIFO.

PUBLIC
IX_SSP_STATUS ixSspAccTxFIFOIntEnable (**TxFIFOThresholdHandler** txFIFOIntrHandler)
Enables service request interrupt of the Tx FIFO.

PUBLIC
IX_SSP_STATUS ixSspAccTxFIFOIntDisable (void)
Disables service request interrupt of the Tx FIFO.

PUBLIC
IX_SSP_STATUS ixSspAccLoopbackEnable (BOOL loopbackEnable)
Enables/disables the loopback mode.

PUBLIC
IX_SSP_STATUS ixSspAccSpiSclkPolaritySet (**IxSspAccSpiSclkPolarity** spiSclkPolaritySelected)
Sets the SPI SCLK Polarity to Low or High.

PUBLIC
IX_SSP_STATUS ixSspAccSpiSclkPhaseSet (**IxSspAccSpiSclkPhase** spiSclkPhaseSelected)
Sets the SPI SCLK Phase.

PUBLIC**ixSspAccMicrowireControlWordSet** (**IxSspAccMicrowireCtlWord**
IX_SSP_STATUS microwireCtlWordSelected)
Sets the Microwire control word to 8 or 16 bit format.

PUBLIC
IX_SSP_STATUS ixSspAccTxFIFOThresholdSet (**IxSspAccFifoThreshold** txFIFOThresholdSelected)
Sets the Tx FIFO Threshold.

PUBLIC
IX_SSP_STATUS ixSspAccRxFIFOThresholdSet (**IxSspAccFifoThreshold** rxFIFOThresholdSelected)
Sets the Rx FIFO Threshold.

PUBLIC
IX_SSP_STATUS ixSspAccStatsGet (**IxSspAccStatsCounters** *sspStats)
Returns the SSP Statistics through the pointer passed in.

PUBLIC void **ixSspAccStatsReset** (void)

Resets the SSP Statistics.

PUBLIC
IX_SSP_STATUS ixSspAccShow (void)
Display SSP status registers and statistics counters.

PUBLIC
IX_SSP_STATUS ixSspAccSSPBusyCheck (void)
Determine the state of the SSP serial port hardware.

PUBLIC UINT8 **ixSspAccTxFIFOLevelGet** (void)
Obtain the Tx FIFO's level.

PUBLIC UINT8 **ixSspAccRxFIFOLevelGet** (void)
Obtain the Rx FIFO's level.

PUBLIC
IX_SSP_STATUS ixSspAccRxFIFOOverrunCheck (void)
Check if the Rx FIFO has overrun its FIFOs.

Detailed Description

Intel (R) IXP400 Software SSP Serial Port Access Public API.

Typedef Documentation

```
typedef void(* RxFIFOOverrunHandler)(void)
```

SSP Rx FIFO Overrun handler.

This function is called for the client to handle Rx FIFO Overrun that occurs in the SSP hardware

Definition at line **252** of file **IxSspAcc.h**.

```
typedef void(* RxFIFOThresholdHandler)(void)
```

SSP Rx FIFO Threshold hit or above handler.

This function is called for the client to handle Rx FIFO threshold hit or or above that occurs in the SSP hardware

Definition at line **262** of file **IxSspAcc.h**.

typedef void(* TxFIFOThresholdHandler)(void)

SSP Tx FIFO Threshold hit or below handler.

This function is called for the client to handle Tx FIFO threshold hit or or below that occurs in the SSP hardware

Definition at line **272** of file **IxSspAcc.h**.

Enumeration Type Documentation

enum IX_SSP_STATUS

The statuses that can be returned in a SSP Serial Port Access.

Enumeration values:

IX_SSP_SUCCESS	Success status.
IX_SSP_FAIL	Fail status.
IX_SSP_RX_FIFO_OVERRUN_HANDLER_MISSING	Rx FIFO Overrun handler is NULL.
IX_SSP_RX_FIFO_HANDLER_MISSING	Rx FIFO threshold hit or above handler is NULL.
IX_SSP_TX_FIFO_HANDLER_MISSING	Tx FIFO threshold hit or below handler is NULL.
IX_SSP_FIFO_NOT_EMPTY_FOR_SETTING_CTL_CMD	Tx FIFO not empty and therefore microwire control command size setting is not allowed.
IX_SSP_INVALID_FRAME_FORMAT_ENUM_VALUE	frame format selected is invalid
IX_SSP_INVALID_DATA_SIZE_ENUM_VALUE	data size selected is invalid
IX_SSP_INVALID_CLOCK_SOURCE_ENUM_VALUE	source clock selected is invalid
IX_SSP_INVALID_TX_FIFO_THRESHOLD_ENUM_VALUE	Tx FIFO threshold selected is invalid.
IX_SSP_INVALID_RX_FIFO_THRESHOLD_ENUM_VALUE	Rx FIFO threshold selected is invalid.
IX_SSP_INVALID_SPI_PHASE_ENUM_VALUE	SPI phase selected is invalid.
IX_SSP_INVALID_SPI_POLARITY_ENUM_VALUE	SPI polarity selected is invalid.
IX_SSP_INVALID_MICROWIRE_CTL_CMD_ENUM_VALUE	Microwire control command selected is invalid.
IX_SSP_INT_UNBIND_FAIL	Interrupt unbind fail to unbind SSP interrupt.
IX_SSP_INT_BIND_FAIL	Interrupt bind fail during init.
IX_SSP_RX_FIFO_NOT_EMPTY	Rx FIFO not empty while trying to change data size.
IX_SSP_TX_FIFO_NOT_EMPTY	Rx FIFO not empty while trying to change data size or microwire control command size.

IX_SSP_POLL_MODE_BLOCKING	poll mode selected blocks interrupt mode from being selected
IX_SSP_TX_FIFO_HIT_BELOW_THRESHOLD	Tx FIFO level hit or below threshold.
IX_SSP_TX_FIFO_EXCEED_THRESHOLD	Tx FIFO level exceeded threshold.
IX_SSP_RX_FIFO_HIT_ABOVE_THRESHOLD	Rx FIFO level hit or exceeded threshold.
IX_SSP_RX_FIFO_BELOW_THRESHOLD	Rx FIFO level below threshold.
IX_SSP_BUSY	SSP is busy.
IX_SSP_IDLE	SSP is idle.
IX_SSP_OVERRUN_OCCURRED	SSP has experienced an overrun.
IX_SSP_NO_OVERRUN	SSP did not experience an overrun.
IX_SSP_NOT_SUPORTED	hardware does not support SSP
IX_SSP_NOT_INIT	SSP Access not intialized.
IX_SSP_NULL_POINTER	parameter passed in is NULL

Definition at line **181** of file **IxSspAcc.h**.

enum IxSspAccClkSource

The source to produce the SSP serial clock.

Definition at line **90** of file **IxSspAcc.h**.

enum IxSspAccDataSize

The data sizes in bits that are supported by the protocol.

Definition at line **35** of file **IxSspAcc.h**.

enum IxSspAccFifoThreshold

The threshold in frames (each frame is defined by IxSspAccDataSize) that can be set for the FIFO to trigger a threshold exceed when checking with the ExceedThresholdCheck functions or an interrupt when it is enabled.

Definition at line **153** of file **IxSspAcc.h**.

enum IxSspAccFrameFormat

The frame format that is to be used − SPI, SSP, or Microwire.

Definition at line **75** of file **IxSspAcc.h**.

enum IxSspAccMicrowireCtlWord

The Microwire control word can be either 8 or 16 bit.

Definition at line **136** of file **IxSspAcc.h**.

enum IxSspAccPortStatus

The status of the SSP port to be set to enable/disable.

Definition at line **61** of file **IxSspAcc.h**.

enum IxSspAccSpiSclkPhase

The SPI SCLK Phase: 0 − SCLK is inactive one cycle at the start of a frame and 1/2 cycle at the end of a frame. 1 − SCLK is inactive 1/2 cycle at the start of a frame and one cycle at the end of a frame.

Definition at line **108** of file **IxSspAcc.h**.

enum IxSspAccSpiSclkPolarity

The SPI SCLK Polarity can be set to either low or high.

Definition at line **122** of file **IxSspAcc.h**.

Function Documentation

ixSspAccClockSourceSelect (**IxSspAccClkSource** *clkSourceSelected*)

Sets the clock source of the SSP Serial Port hardware.

Parameters:

IxSspAccClkSource [in] clkSourceSelected	− The clock source from either external source on on−chip can be selected as the source

Global Data :

- None.

This API will set the clock source for the transfers via user input.

Returns:

◊ IX_SSP_SUCCESS − clock source set with valid enum value
◊ IX_SSP_INVALID_CLOCK_SOURCE_ENUM_VALUE − invalid enum value

◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccDataSizeSelect (**IxSspAccDataSize** *dataSizeSelected*)

Sets the data size for transfers.

Parameters:

IxSspAccDataSize [in] dataSizeSelected	− The data size between 4 and 16 that can be selected for data transfers

Global Data :

- None.

This API will set the data size for the transfers via user input. It will disallow the change of the data size if either of the Rx/Tx FIFO is not empty to prevent data loss. *NOTE*: The SSP port will be disabled if the FIFOs are found to be empty and if between the check and disabling of the SSP (which clears the FIFOs) data is received into the FIFO, it might be lost. *NOTE*: The FIFOs can be cleared by disabling the SSP Port if necessary to force the data size change.

Returns:

◊ IX_SSP_SUCCESS − data size set with valid enum value
◊ IX_SSP_RX_FIFO_NOT_EMPTY − Rx FIFO not empty, data size change is not allowed.
◊ IX_SSP_TX_FIFO_NOT_EMPTY − Tx FIFO not empty, data size change is not allowed.
◊ IX_SSP_INVALID_DATA_SIZE_ENUM_VALUE − invalid enum value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccFIFODataReceive (UINT16 * *data*,
UINT32 *amtOfData*
)

Extract data from the SSP Serial Port's FIFO.

Parameters:

*UINT16 [in] *data*	− pointer to the location to receive the data into "UINT32 [in] amtOfData" − number of data to be received.

Global Data :

- None.

This API will extract the amount of data specified by "amtOfData" from the FIFO already received by the hardware into the buffer pointed to by "data".

Returns:

◊ IX_SSP_SUCCESS − Data extracted successfully from FIFO
◊ IX_SSP_FAIL − FIFO has no data
◊ IX_SSP_NULL_POINTER − data pointer passed by client is NULL
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

```
ixSspAccFIFODataSubmit ( UINT16 *  data,
                         UINT32    amtOfData
                       )
```

Inserts data into the SSP Serial Port's FIFO.

Parameters:

*UINT16 [in] *data*	− pointer to the location to transmit the data from "UINT32 [in] amtOfData" − number of data to be transmitted.

Global Data :

- None.

This API will insert the amount of data specified by "amtOfData" from buffer pointed to by "data" into the FIFO to be transmitted by the hardware.

Returns:

◊ IX_SSP_SUCCESS − Data inserted successfully into FIFO
◊ IX_SSP_FAIL − FIFO insufficient space
◊ IX_SSP_NULL_POINTER − data pointer passed by client is NULL
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

```
ixSspAccFrameFormatSelect ( IxSspAccFrameFormat  frameFormatSelected  )
```

Sets the frame format for the SSP Serial Port hardware.

Parameters:

IxSspAccFrameFormat [in] frameFormatSelected	− The frame format of SPI, SSP or Microwire can be selected as the format

Global Data :

- None.

This API will set the format for the transfers via user input. *NOTE*: The SSP hardware will be disabled to clear the FIFOs. Then its previous state (enabled/disabled) restored after changing the format.

Returns:

◊ IX_SSP_SUCCESS − frame format set with valid enum value
◊ IX_SSP_INVALID_FRAME_FORMAT_ENUM_VALUE − invalid frame format value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccInit (**IxSspInitVars** * *initVarsSelected*)

Initializes the SSP Access module.

Parameters:

*IxSspAccInitVars [in] *initVarsSelected* − struct containing required variables for initialization

Global Data :

- None.

This API will initialize the SSP Serial Port hardware to the user specified configuration. Then it will enable the SSP Serial Port. *NOTE*: Once interrupt or polling mode is selected, the mode cannot be changed via the interrupt enable/disable function but the init needs to be called again to change it.

Returns:

◊ IX_SSP_SUCCESS − Successfully initialize and enable the SSP serial port.
◊ IX_SSP_RX_FIFO_HANDLER_MISSING − interrupt mode is selected but Rx FIFO handler pointer is NULL
◊ IX_SSP_TX_FIFO_HANDLER_MISSING − interrupt mode is selected but Tx FIFO handler pointer is NULL
◊ IX_SSP_RX_FIFO_OVERRUN_HANDLER_MISSING − interrupt mode is selected but Rx FIFO Overrun handler pointer is NULL
◊ IX_SSP_RX_FIFO_NOT_EMPTY − Rx FIFO not empty, data size change is not allowed.
◊ IX_SSP_TX_FIFO_NOT_EMPTY − Tx FIFO not empty, data size change is not allowed.
◊ IX_SSP_INVALID_FRAME_FORMAT_ENUM_VALUE − frame format selected is invalid
◊ IX_SSP_INVALID_DATA_SIZE_ENUM_VALUE − data size selected is invalid
◊ IX_SSP_INVALID_CLOCK_SOURCE_ENUM_VALUE − clock source selected is invalid
◊ IX_SSP_INVALID_TX_FIFO_THRESHOLD_ENUM_VALUE − Tx FIFO threshold level selected is invalid
◊ IX_SSP_INVALID_RX_FIFO_THRESHOLD_ENUM_VALUE − Rx FIFO threshold level selected is invalid
◊ IX_SSP_INVALID_SPI_PHASE_ENUM_VALUE − SPI phase selected is invalid
◊ IX_SSP_INVALID_SPI_POLARITY_ENUM_VALUE − SPI polarity selected is invalid

◊ IX_SSP_INVALID_MICROWIRE_CTL_CMD_ENUM_VALUE − microwire control command size is invalid
◊ IX_SSP_INT_UNBIND_FAIL − interrupt handler failed to unbind SSP interrupt
◊ IX_SSP_INT_BIND_FAIL − interrupt handler failed to bind to SSP interrupt hardware trigger
◊ IX_SSP_NOT_SUPORTED − hardware does not support SSP
◊ IX_SSP_NULL_POINTER − parameter passed in is NULL

- Reentrant : yes
- ISR Callable : yes

ixSspAccLoopbackEnable (BOOL *loopbackEnable*)

Enables/disables the loopback mode.

Parameters:

BOOL [in] loopbackEnable − True to enable and false to disable.

Global Data :

- None.

This API will set the mode of operation to either loopback or normal mode according to the user input.

Returns:

◊ IX_SSP_SUCCESS − Loopback enabled successfully
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccMicrowireControlWordSet (**IxSspAccMicrowireCtlWord** *microwireCtlWordSelected*)

Sets the Microwire control word to 8 or 16 bit format.

Parameters:

IxSspAccMicrowireCtlWord [in] microwireCtlWordSelected	− Microwire control word format can be either 8 or 16 bit format

Global Data :

- None.

This API is only used for the Microwire frame format and will set the control word to 8 or 16 bit format

Returns:

◊ IX_SSP_SUCCESS − Microwire Control Word set with valid enum value
◊ IX_SSP_TX_FIFO_NOT_EMPTY − Tx FIFO not empty, data size change is not allowed.

◊ IX_SSP_INVALID_MICROWIRE_CTL_CMD_ENUM_VALUE − invalid enum value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOHitOrAboveThresholdCheck (void)

Check if the Rx FIFO threshold has been hit or exceeded.

Parameters:

− None

Global Data :

- None.

This API will return whether the Rx FIFO level is below threshold or not

Returns:

◊ IX_SSP_RX_FIFO_HIT_ABOVE_THRESHOLD − Rx FIFO level hit or exceeded threshold
◊ IX_SSP_RX_FIFO_BELOW_THRESHOLD − Rx FIFO level below threshold
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOIntDisable (void)

Disables service request interrupt of the Rx FIFO.

Parameters:

− None

Global Data :

- None.

This API will disable the service request interrupt of the Rx FIFO.

Returns:

◊ IX_SSP_SUCCESS − Rx FIFO Interrupt disabled successfully
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOIntEnable (**RxFIFOThresholdHandler** *rxFIFOIntrHandler*)

Enables service request interrupt whenever the Rx FIFO hits its threshold.

Parameters:

*void [in] *rxFIFOIntrHandler(UINT32)*	− function pointer to the interrupt handler for the Rx FIFO exceeded.

Global Data :

- None.

This API will enable the service request interrupt for the Rx FIFO

Returns:

◊ IX_SSP_SUCCESS − Rx FIFO level interrupt enabled successfully
◊ IX_SSP_RX_FIFO_HANDLER_MISSING − missing handler for Rx FIFO level interrupt
◊ IX_SSP_POLL_MODE_BLOCKING − poll mode is selected at init, interrupt not allowed to be enabled. Use init to enable interrupt mode.
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOLevelGet (void)

Obtain the Rx FIFO's level.

Parameters:

− None

Global Data :

- None.

This API will return the level of the Rx FIFO

Returns:

◊ 0..16; 0 can also mean SSP not initialized and will need to be init.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOOverrunCheck (void)

Check if the Rx FIFO has overrun its FIFOs.

Parameters:

− None

Global Data :

- None.

This API will return whether the Rx FIFO has overrun its 16 FIFOs

Returns:

◊ IX_SSP_OVERRUN_OCCURRED − Rx FIFO overrun occurred
◊ IX_SSP_NO_OVERRUN − Rx FIFO did not overrun
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccRxFIFOThresholdSet (**IxSspAccFifoThreshold** *rxFIFOThresholdSelected*)

Sets the Rx FIFO Threshold.

Parameters:

IxSspAccFifoThreshold [in] *rxFIFOThresholdSelected*	− Threshold that is set for a Tx FIFO service request to be triggered

Global Data :

- None.

This API will will set the threshold for a Rx FIFO threshold to be triggered

Returns:

◊ IX_SSP_SUCCESS − Rx FIFO Threshold set with valid enum value
◊ IX_SSP_INVALID_RX_FIFO_THRESHOLD_ENUM_VALUE − invalid enum value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccSerialClockRateConfigure (UINT8 *serialClockRateSelected*)

Sets the on−chip Serial Clock Rate of the SSP Serial Port hardware.

Parameters:

UINT8 [in] serialClockRateSelected	− The serial clock rate that can be set is between 7.2Kbps and 1.8432Mbps. The formula used is Bit rate = 3.6864x10^6 / (2 x (SerialClockRateSelected + 1))

Global Data :

- None.

This API will set the serial clock rate for the transfers via user input.

Returns:

◊ IX_SSP_SUCCESS − Serial clock rate configured successfully
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccShow (void)

Display SSP status registers and statistics counters.

Parameters:

− None

Global Data :

- None.

This API will display the status registers of the SSP and the statistics counters.

Returns:

◊ IX_SSP_SUCCESS − SSP show called successfully.
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccSpiSclkPhaseSet (**IxSspAccSpiSclkPhase** *spiSclkPhaseSelected*)

Sets the SPI SCLK Phase.

Parameters:

IxSspAccSpiSclkPhase [in] spiSclkPhaseSelected	− Phase of either the SCLK is inactive one cycle at the start of a frame and 1/2 cycle at the end of a frame, OR the SCLK is inactive 1/2 cycle at the start of a frame and one cycle at the end of a frame.

Global Data :

- IX_SSP_SUCCESS − SPI Sclk phase set with valid enum value
- IX_SSP_INVALID_SPI_PHASE_ENUM_VALUE − invalid SPI phase value
- IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

This API is only used for the SPI frame format and will set the SPI SCLK phase according to user input.

Returns:

◊ None

- Reentrant : yes
- ISR Callable : yes

ixSspAccSpiSclkPolaritySet (**IxSspAccSpiSclkPolarity** *spiSclkPolaritySelected*)

Sets the SPI SCLK Polarity to Low or High.

Parameters:

− "IxSspAccSpiSclkPolarity [in] spiSclkPolaritySelected" − SPI SCLK polarity that can be selected to either high or low

Global Data :

- None.

This API is only used for the SPI frame format and will set the SPI SCLK polarity to either low or high

Returns:

◊ IX_SSP_SUCCESS − SPI Sclk polarity set with valid enum value
◊ IX_SSP_INVALID_SPI_POLARITY_ENUM_VALUE − invalid SPI polarity value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccSSPBusyCheck (void)

Determine the state of the SSP serial port hardware.

Parameters:

− None

Global Data :

- None.

This API will return the state of the SSP serial port hardware − busy or idle

Returns:

◊ IX_SSP_BUSY − SSP is busy
◊ IX_SSP_IDLE − SSP is idle.
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccSSPPortStatusSet (**IxSspAccPortStatus** *portStatusSelected*)

Enables/disables the SSP Serial Port hardware.

Parameters:

IxSspAccPortStatus [in] portStatusSelected	− Set the SSP port to enable or disable

Global Data :

- None.

This API will enable/disable the SSP Serial Port hardware. NOTE: This function is called by init to enable the SSP after setting up the configurations and by uninit to disable the SSP.

Returns:

◊ IX_SSP_SUCCESS − Port status set with valid enum value
◊ IX_SSP_FAIL − invalid enum value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccStatsGet (**IxSspAccStatsCounters** * *sspStats*)

Returns the SSP Statistics through the pointer passed in.

Parameters:

*IxSspAccStatsCounters [in] *sspStats*	− SSP statistics counter will be read and written to the location pointed by this pointer.

Global Data :

- None.

This API will return the statistics counters of the SSP transfers.

Returns:

◊ IX_SSP_SUCCESS − Stats obtained into the pointer provided successfully

◊ IX_SSP_FAIL − client provided pointer is NULL

- Reentrant : yes
- ISR Callable : yes

ixSspAccStatsReset (void)

Resets the SSP Statistics.

Parameters:

− None

Global Data :

- None.

This API will reset the SSP statistics counters.

Returns:

◊ None

- Reentrant : yes
- ISR Callable : yes

ixSspAccTxFIFOHitOrBelowThresholdCheck (void)

Check if the Tx FIFO threshold has been hit or fallen below.

Parameters:

− None

Global Data :

- None.

This API will return whether the Tx FIFO threshold has been exceeded or not

Returns:

◊ IX_SSP_TX_FIFO_HIT_BELOW_THRESHOLD − Tx FIFO level hit or below threshold .
◊ IX_SSP_TX_FIFO_EXCEED_THRESHOLD − Tx FIFO level exceeded threshold.
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccTxFIFOIntDisable (void)

Disables service request interrupt of the Tx FIFO.

Parameters:

- None

Global Data :

- None.

This API will disable the service request interrupt of the Tx FIFO

Returns:

- ◊ IX_SSP_SUCCESS − Tx FIFO Interrupt disabled successfuly.
- ◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccTxFIFOIntEnable (**TxFIFOThresholdHandler** *txFIFOIntrHandler*)

Enables service request interrupt of the Tx FIFO.

Parameters:

*void [in] *txFIFOIntrHandler(UINT32)*	− function pointer to the interrupt handler for the Tx FIFO exceeded.

Global Data :

- None.

This API will enable the service request interrupt of the Tx FIFO.

Returns:

- ◊ IX_SSP_SUCCESS − Tx FIFO level interrupt enabled successfully
- ◊ IX_SSP_TX_FIFO_HANDLER_MISSING − missing handler for Tx FIFO level interrupt
- ◊ IX_SSP_POLL_MODE_BLOCKING − poll mode is selected at init, interrupt not allowed to be enabled. Use init to enable interrupt mode.
- ◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccTxFIFOLevelGet (void)

Obtain the Tx FIFO's level.

Parameters:

- None

Global Data :

- None.

This API will return the level of the Tx FIFO

Returns:

◊ 0..16; 0 can also mean SSP not initialized and will need to be init.

- Reentrant : yes
- ISR Callable : yes

ixSspAccTxFIFOThresholdSet (**IxSspAccFifoThreshold** *txFIFOThresholdSelected*)

Sets the Tx FIFO Threshold.

Parameters:

IxSspAccFifoThreshold [in]	− Threshold that is set for a Tx FIFO service request
txFIFOThresholdSelected	to be triggered

Global Data :

- None.

This API will set the threshold for a Tx FIFO threshold to be triggered

Returns:

◊ IX_SSP_SUCCESS − Tx FIFO Threshold set with valid enum value
◊ IX_SSP_INVALID_TX_FIFO_THRESHOLD_ENUM_VALUE − invalid enum value
◊ IX_SSP_NOT_INIT − SSP not initialized. SSP init needs to be called.

- Reentrant : yes
- ISR Callable : yes

ixSspAccUninit (void)

Un−initializes the SSP Serial Port Access component.

Parameters:

- None

Global Data :

- None.

This API will disable the SSP Serial Port hardware. The client can call the init function again if they wish to enable the SSP.

Returns:

◊ IX_SSP_SUCCESS − successfully uninit SSP component
◊ IX_SSP_INT_UNBIND_FAIL − interrupt handler failed to unbind SSP interrupt

- Reentrant : yes
- ISR Callable : yes

Intel (R) IXP400 Software Timer Control (IxTimerCtrl) API

The public API for the IXP400 Timer Control Component.

Defines

#define **IX_TIMERCTRL_NO_FREE_TIMERS**

Timer schedule return code.

#define **IX_TIMERCTRL_PARAM_ERROR**

Timer schedule return code.

Typedefs

typedef void(* **IxTimerCtrlTimerCallback**)(void *userParam)

A typedef for a pointer to a timer callback function.

Enumerations

enum **IxTimerCtrlPurpose** {
IxTimerCtrlAdslPurpose,
IxTimerCtrlMaxPurpose
}

List used to identify the users of timers.

Functions

PUBLIC **ixTimerCtrlSchedule** (**IxTimerCtrlTimerCallback** func, void *userParam,
IX_STATUS **IxTimerCtrlPurpose** purpose, UINT32 relativeTime, unsigned *timerId)

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms. This function.

PUBLIC **ixTimerCtrlScheduleRepeating** (**IxTimerCtrlTimerCallback** func, void *param,
IX_STATUS **IxTimerCtrlPurpose** purpose, UINT32 interval, unsigned *timerId)

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms.

PUBLIC
IX_STATUS **ixTimerCtrlCancel** (unsigned id)

Cancels a scheduled callback.

ixTimerCtrlInit (void)

PUBLIC
IX_STATUS
Initialise the Timer Control Component.

PUBLIC void **ixTimerCtrlShow** (void)
Display the status of the Timer Control Component.

Detailed Description

The public API for the IXP400 Timer Control Component.

Define Documentation

```
#define IX_TIMERCTRL_NO_FREE_TIMERS
```

Timer schedule return code.

Indicates that the request to start a timer failed because all available timer resources are used.

Definition at line **48** of file **IxTimerCtrl.h**.

```
#define IX_TIMERCTRL_PARAM_ERROR
```

Timer schedule return code.

Indicates that the request to start a timer failed because the client has supplied invalid parameters.

Definition at line **61** of file **IxTimerCtrl.h**.

Typedef Documentation

```
typedef void(* IxTimerCtrlTimerCallback)(void *userParam)
```

A typedef for a pointer to a timer callback function.

void * − This parameter is supplied by the client when the timer is started and passed back to the client in the callback.

Note:
in general timer callback functions should not block or take longer than 100ms. This constraint is required to ensure that higher priority callbacks are not held up. All callbacks are called from the same thread. This thread is a shared resource. The parameter passed is provided when the timer is scheduled.

Definition at line **81** of file **IxTimerCtrl.h**.

Enumeration Type Documentation

enum IxTimerCtrlPurpose

List used to identify the users of timers.

Note:

The order in this list indicates priority. Components appearing higher in the list will be given priority over components lower in the list. When adding components, please insert at an appropriate position for priority (i.e values should be less than IxTimerCtrlMaxPurpose) .

Definition at line **93** of file **IxTimerCtrl.h**.

Function Documentation

ixTimerCtrlCancel (unsigned *id*)

Cancels a scheduled callback.

Parameters:

id unsigned [in] − the id of the callback to be cancelled.

Returns:

◊ IX_SUCCESS − The timer was successfully stopped.
◊ IX_FAIL − The id parameter did not corrrespond to any running timer..

Note:

This function is re−entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accesed by another thread.

ixTimerCtrlInit (void)

Initialise the Timer Control Component.

Returns:

◊ IX_SUCCESS − The timer control component initialized successfully.
◊ IX_FAIL − The timer control component initialization failed, or the component was already initialized.

Note:

This must be done before any other API function is called. This function should be called once only and is not re−entrant.

```
ixTimerCtrlSchedule ( IxTimerCtrlTimerCallback  func,
                      void *                    userParam,
                      IxTimerCtrlPurpose        purpose,
                      UINT32                    relativeTime,
                      unsigned *                timerId
                    )
```

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms. This function.

Parameters:

func	**IxTimerCtrlTimerCallback** [in] − the callback function to be called.
userParam	void [in] − a parameter to send to the callback function, can be NULL.
purpose	**IxTimerCtrlPurpose** [in] − the purpose of the callback, internally this component will decide the priority of callbacks with different purpose.
relativeTime	UINT32 [in] − time relative to now in milliseconds after which the callback will be called. The time must be greater than the duration of one OS tick.
**timerId*	unsigned [out] − An id for the callback scheduled. This id can be used to cancel the callback.

Returns:

- ◊ IX_SUCCESS − The timer was started successfully.
- ◊ IX_TIMERCTRL_NO_FREE_TIMERS − The timer was not started because the maximum number of running timers has been exceeded.
- ◊ IX_TIMERCTRL_PARAM_ERROR − The timer was not started because the client has supplied a NULL callback func, or the requested timeout is less than one OS tick.

Note:

This function is re−entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accesed by another thread.

```
ixTimerCtrlScheduleRepeating ( IxTimerCtrlTimerCallback  func,
                               void *                    param,
                               IxTimerCtrlPurpose        purpose,
                               UINT32                    interval,
                               unsigned *                timerId
                             )
```

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms.

Parameters:

func	**IxTimerCtrlTimerCallback** [in] − the callback function to be called.
userParam	void [in] − a parameter to send to the callback function, can be NULL.
purpose	**IxTimerCtrlPurpose** [in] − the purpose of the callback, internally this component will decide the priority of callbacks with different purpose.
interval	UINT32 [in] − the interval in milliseconds between calls to func.
timerId	unsigned [out] − An id for the callback scheduled. This id can be used to cancel the

callback.

Returns:

- ◊ IX_SUCCESS − The timer was started successfully.
- ◊ IX_TIMERCTRL_NO_FREE_TIMERS − The timer was not started because the maximum number of running timers has been exceeded.
- ◊ IX_TIMERCTRL_PARAM_ERROR − The timer was not started because the client has supplied a NULL callback func, or the requested timeout is less than one OS tick.

Note:

This function is re−entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accesed by another thread.

ixTimerCtrlShow (void)

Display the status of the Timer Control Component.

Returns:

void

Note:

Displays a list of running timers. This function is not re−entrant. This function does not suspend the calling thread.

Intel (R) IXP400 Software Time Sync Access Component API

Public API for IxTimeSyncAcc.

Data Structures

struct **IxTimeSyncAccPtpMsgData**
Struct for data from the PTP message returned when TimeStamp available.

struct **IxTimeSyncAccPtpMsgData**
Struct for data from the PTP message returned when TimeStamp available.

struct **IxTimeSyncAccPtpMsgData**
Struct for data from the PTP message returned when TimeStamp available.

struct **IxTimeSyncAccStats**
Statistics for the PTP messages.

struct **IxTimeSyncAccStats**
Statistics for the PTP messages.

struct **IxTimeSyncAccStats**
Statistics for the PTP messages.

struct **IxTimeSyncAccTimeValue**
Struct to hold 64 bit SystemTime and TimeStamp values.

struct **IxTimeSyncAccTimeValue**
Struct to hold 64 bit SystemTime and TimeStamp values.

struct **IxTimeSyncAccTimeValue**
Struct to hold 64 bit SystemTime and TimeStamp values.

struct **IxTimeSyncAccUuid**
Struct to hold 48 bit UUID values captured in Sync or Delay_Req messages.

struct **IxTimeSyncAccUuid**
Struct to hold 48 bit UUID values captured in Sync or Delay_Req messages.

struct **IxTimeSyncAccUuid**
Struct to hold 48 bit UUID values captured in Sync or Delay_Req messages.

Typedefs

typedef void(* **IxTimeSyncAccTargetTimeCallback**)(**IxTimeSyncAccTimeValue** targetTime)

Callback for use by target time stamp interrupt.

typedef void(* **IxTimeSyncAccAuxTimeCallback**)(**IxTimeSyncAccAuxMode** auxMode, **IxTimeSyncAccTimeValue** auxTime)

Callback for use by auxiliary time interrupts.

Enumerations

enum **IxTimeSyncAccStatus** {
IX_TIMESYNCACC_SUCCESS,
IX_TIMESYNCACC_INVALIDPARAM,
IX_TIMESYNCACC_NOTIMESTAMP,
IX_TIMESYNCACC_INTERRUPTMODEINUSE,
IX_TIMESYNCACC_FAILED
}

The status as returned from the API.

enum **IxTimeSyncAccAuxMode** {
IX_TIMESYNCACC_AUXMODE_MASTER,
IX_TIMESYNCACC_AUXMODE_SLAVE,
IX_TIMESYNCACC_AUXMODE_INVALID
}

Master or Slave Auxiliary Time Stamp (Snap Shot).

enum **IxTimeSyncAcc1588PTPPort** {
IX_TIMESYNCACC_NPE_A_1588PTP_PORT,
IX_TIMESYNCACC_NPE_B_1588PTP_PORT,
IX_TIMESYNCACC_NPE_C_1588PTP_PORT,
IX_TIMESYNCACC_NPE_1588PORT_INVALID
}

IEEE 1588 PTP Communication Port(Channel).

enum **IxTimeSyncAcc1588PTPPortMode** {
IX_TIMESYNCACC_1588PTP_PORT_MASTER,
IX_TIMESYNCACC_1588PTP_PORT_SLAVE,
IX_TIMESYNCACC_1588PTP_PORT_ANYMODE,
IX_TIMESYNCACC_1588PTP_PORT_MODE_INVALID
}

Master or Slave mode for IEEE 1588 PTP Communication Port.

enum **IxTimeSyncAcc1588PTPMsgType** {
IX_TIMESYNCACC_1588PTP_MSGTYPE_SYNC,
IX_TIMESYNCACC_1588PTP_MSGTYPE_DELAYREQ,
IX_TIMESYNCACC_1588PTP_MSGTYPE_UNKNOWN
}

1588 PTP Messages types that can be detected on communication port

Functions

PUBLIC**ixTimeSyncAccPTPPortConfigSet** (**IxTimeSyncAcc1588PTPPort** ptpPort,
IxTimeSyncAccStatus IxTimeSyncAcc1588PTPPortMode ptpPortMode)

Configures the IEEE 1588 message detect on particular PTP port.

PUBLIC**ixTimeSyncAccPTPPortConfigGet** (**IxTimeSyncAcc1588PTPPort** ptpPort,
IxTimeSyncAccStatus IxTimeSyncAcc1588PTPPortMode *ptpPortMode)

Retrieves IEEE 1588 PTP operation mode on particular PTP port.

PUBLIC**ixTimeSyncAccPTPRxPoll** (**IxTimeSyncAcc1588PTPPort** ptpPort,
IxTimeSyncAccStatus IxTimeSyncAccPtpMsgData *ptpMsgData)

Polls the IEEE 1588 message/time stamp detect status on a particular PTP Port on the Receive side.

PUBLIC**ixTimeSyncAccPTPTxPoll** (**IxTimeSyncAcc1588PTPPort** ptpPort,
IxTimeSyncAccStatus IxTimeSyncAccPtpMsgData *ptpMsgData)

Polls the IEEE 1588 message/time stamp detect status on a particular PTP Port on the Transmit side.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccSystemTimeSet (**IxTimeSyncAccTimeValue** systemTime)

Sets the System Time in the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccSystemTimeGet (**IxTimeSyncAccTimeValue** *systemTime)

Gets the System Time from the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccTickRateSet (UINT32 tickRate)

Sets the Tick Rate (Frequency Scaling Value) in the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccTickRateGet (UINT32 *tickRate)

Gets the Tick Rate (Frequency Scaling Value) from the IEEE 1588 hardware assist block.

PUBLIC**ixTimeSyncAccTargetTimeInterruptEnable**
IxTimeSyncAccStatus (**IxTimeSyncAccTargetTimeCallback** targetTimeCallback)

Enables the interrupt to verify the condition where the System Time greater or equal to the Target Time in the IEEE 1588 hardware assist block. If the condition is true an interrupt will be sent to Intel XScale(R) Processor.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccTargetTimeInterruptDisable (void)

*Disables the interrupt for the condition explained in the function description of **ixTimeSyncAccTargetTimeInterruptEnable**.*

PUBLIC **ixTimeSyncAccTargetTimePoll** (BOOL *ttmPollFlag,
IxTimeSyncAccStatus IxTimeSyncAccTimeValue *targetTime)

Poll to verify the condition where the System Time greater or equal to the Target Time in the IEEE 1588 hardware assist block. If the condition is true an event flag is set in the hardware.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccTargetTimeSet (**IxTimeSyncAccTimeValue** targetTime)

Sets the Target Time in the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccTargetTimeGet (**IxTimeSyncAccTimeValue** *targetTime)

Gets the Target Time in the IEEE 1588 hardware assist block.

PUBLIC **ixTimeSyncAccAuxTimeInterruptEnable** (**IxTimeSyncAccAuxMode** auxMode,
IxTimeSyncAccStatus IxTimeSyncAccAuxTimeCallback auxTimeCallback)

Enables the interrupt notification for the given mode of Auxiliary Time Stamp in the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccAuxTimeInterruptDisable (**IxTimeSyncAccAuxMode** auxMode)

Disables the interrupt for the indicated mode of Auxiliary Time Stamp in the IEEE 1588 hardware assist block.

PUBLIC **ixTimeSyncAccAuxTimePoll** (**IxTimeSyncAccAuxMode** auxMode, BOOL
IxTimeSyncAccStatus *auxPollFlag, **IxTimeSyncAccTimeValue** *auxTime)

Poll for the Auxiliary Time Stamp captured for the mode indicated (Master or Slave).

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccReset (void)

Resets the IEEE 1588 hardware assist block.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccStatsGet (**IxTimeSyncAccStats** *timeSyncStats)

Returns the IxTimeSyncAcc Statistics in the client supplied buffer.

PUBLIC void **ixTimeSyncAccStatsReset** (void)

Reset Time Sync statistics.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccShow (void)

Displays the Time Sync current status.

PUBLIC
IxTimeSyncAccStatus ixTimeSyncAccUnInit (void)

This API is used to unintialize the timesyncAcc component from the client application.

Detailed Description

Public API for IxTimeSyncAcc.

Typedef Documentation

IxTimeSyncAccAuxTimeCallback

Callback for use by auxiliary time interrupts.

Definition at line **188** of file **IxTimeSyncAcc.h**.

IxTimeSyncAccTargetTimeCallback

Callback for use by target time stamp interrupt.

Definition at line **179** of file **IxTimeSyncAcc.h**.

Enumeration Type Documentation

enum IxTimeSyncAcc1588PTPMsgType

1588 PTP Messages types that can be detected on communication port

Note that client code can determine this based on master/slave mode in which it is already operating in and this information is made available for the sake of convenience only.

Enumeration values:

IX_TIMESYNCACC_1588PTP_MSGTYPE_SYNC	PTP Sync message sent by Master or received by Slave.
IX_TIMESYNCACC_1588PTP_MSGTYPE_DELAYREQ	PTP Delay_Req message sent by Slave or received by Master.
IX_TIMESYNCACC_1588PTP_MSGTYPE_UNKNOWN	Other PTP and non−PTP message sent or received by both Master and/or Slave.

Definition at line **106** of file **IxTimeSyncAcc.h**.

enum IxTimeSyncAcc1588PTPPort

IEEE 1588 PTP Communication Port(Channel).

Enumeration values:

IX_TIMESYNCACC_NPE_A_1588PTP_PORT	PTP Communication

	Port on NPE−A.
IX_TIMESYNCACC_NPE_B_1588PTP_PORT	PTP Communication Port on NPE−B.
IX_TIMESYNCACC_NPE_C_1588PTP_PORT	PTP Communication Port on NPE−C.
IX_TIMESYNCACC_NPE_1588PORT_INVALID	Invalid PTP Communication Port.

Definition at line **70** of file **IxTimeSyncAcc.h**.

enum IxTimeSyncAcc1588PTPPortMode

Master or Slave mode for IEEE 1588 PTP Communication Port.

Enumeration values:

IX_TIMESYNCACC_1588PTP_PORT_MASTER	PTP Communication Port in Master Mode.
IX_TIMESYNCACC_1588PTP_PORT_SLAVE	PTP Communication Port in Slave Mode.
IX_TIMESYNCACC_1588PTP_PORT_ANYMODE	PTP Communication Port in ANY Mode allows time stamping of all messages including non−1588 PTP.
IX_TIMESYNCACC_1588PTP_PORT_MODE_INVALID	Invalid PTP Port Mode.

Definition at line **85** of file **IxTimeSyncAcc.h**.

enum IxTimeSyncAccAuxMode

Master or Slave Auxiliary Time Stamp (Snap Shot).

Enumeration values:

IX_TIMESYNCACC_AUXMODE_MASTER	Auxiliary Master Mode.
IX_TIMESYNCACC_AUXMODE_SLAVE	Auxiliary Slave Mode.
IX_TIMESYNCACC_AUXMODE_INVALID	Invalid Auxiliary Mode.

Definition at line **56** of file **IxTimeSyncAcc.h**.

enum IxTimeSyncAccStatus

The status as returned from the API.

Enumeration values:

IX_TIMESYNCACC_SUCCESS	Requested operation successful.
IX_TIMESYNCACC_INVALIDPARAM	An invalid parameter was passed.

IX_TIMESYNCACC_NOTIMESTAMP	While polling no time stamp available.
IX_TIMESYNCACC_INTERRUPTMODEINUSE	Polling not allowed while operating in interrupt mode.
IX_TIMESYNCACC_FAILED	Internal error occurred.

Definition at line **40** of file **IxTimeSyncAcc.h**.

Function Documentation

IxTimeSyncAccStatus ixTimeSyncAccAuxTimeInterruptDisable (**IxTimeSyncAccAuxMode** *auxMode*)

Disables the interrupt for the indicated mode of Auxiliary Time Stamp in the IEEE 1588 hardware assist block.

Parameters:

auxMode	[in] − Auxiliary time stamp mode (slave or master) using which the interrupt will be disabled.

This API will disable the Auxiliary Time Stamp Interrupt (Master or Slave)

- Re−entrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccAuxTimeInterruptEnable (**IxTimeSyncAccAuxMode** *auxMode*, **IxTimeSyncAccAuxTimeCallback** *auxTimeCallback*)

Enables the interrupt notification for the given mode of Auxiliary Time Stamp in the IEEE 1588 hardware assist block.

Parameters:

auxMode	[in] − Auxiliary time stamp register (slave or master) to use
auxTimeCallback	[in] − Callback to be invoked when interrupt fires

This API will enable the Auxiliary Master/Slave Time stamp Interrupt.

```
NOTE: 1) An individual callback is to be registered for each Slave and Master
Auxiliary Time Stamp registers. Thus to register for both Master and Slave time
stamp interrupts either the same callback or two separate callbacks the API has
to be invoked twice.
      2) On the Intel (R) IXDP465 Development Platform, the Auxiliary Timestamp
signal for slave mode is tied to GPIO 8 pin. This signal is software routed by
```

```
default to PCI for backwards compatibility with the Intel (R) IXDP425 Development
Platform. This routing must be disabled for the auxiliary slave time stamp
register to work properly. The following commands may be used to accomplish this.
However, refer to the Intel (R) IXDP465 Development Platform Users Guide or the
BSP/LSP documentation for more specific information.

For Linux* (at the Redboot prompt i.e., before loading zImage):
    mfill −b 0x54100000 −1 −l 1 −p 8
    mfill −b 0x54100001 −1 −l 1 −p 0x7f
For VxWorks*, at the prompt:
    intDisable(25)
    ixdp400FpgaIODetach(8)
```

- Re−entrant : no
- ISR Callable : no

Returns:

- ◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
- ◊ IX_TIMESYNCACC_INVALIDPARAM − Null parameter passed for callback or invalid auxiliary snapshot mode
- ◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccAuxTimePoll (**IxTimeSyncAccAuxMode** *auxMode*, BOOL * *auxPollFlag*, **IxTimeSyncAccTimeValue** * *auxTime*)

Poll for the Auxiliary Time Stamp captured for the mode indicated (Master or Slave).

Parameters:

auxMode	[in] − Auxiliary Snapshot Register (Slave or Master) to be checked
auxPollFlag	[out] − TRUE if the time stamp captured in auxiliary snapshot register FALSE if the time stamp not captured in auxiliary snapshot register
auxTime	[out] − Copy the current Auxiliary Snapshot Register value into the client provided buffer

Polls for the Time stamp in the appropriate Auxiliary Snapshot Registers based on the mode specified. Return true and the contents of the Auxiliary snapshot, if it is available else return false.

Please refer to the note #2 of the API **ixTimeSyncAccAuxTimeInterruptEnable** for more information for Auxiliary Slave mode.

- Re−entrant : yes
- ISR Callable : no

Returns:

- ◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
- ◊ IX_TIMESYNCACC_INVALIDPARAM − Null parameter passed for auxPollFlag, callback or invalid auxiliary snapshot mode
- ◊ IX_TIMESYNCACC_FAILED − Internal error occurred

◊ IX_TIMESYNCACC_INTERRUPTMODEINUSE − Interrupt mode in use

IxTimeSyncAccStatus ixTimeSyncAccPTPPortConfigGet	(**IxTimeSyncAcc1588PTPPort**	*ptpPort*,
	IxTimeSyncAcc1588PTPPortMode *	*ptpPortMode*
	)	

Retrieves IEEE 1588 PTP operation mode on particular PTP port.

Parameters:

ptpPort	[in] − PTP port
ptpPortMode	[in]− Mode of operation of PTP port (Master or Slave)

This API will identify the time stamping capability of a PTP port by means of obtaining its mode of operation.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccPTPPortConfigSet	(**IxTimeSyncAcc1588PTPPort**	*ptpPort*,
	IxTimeSyncAcc1588PTPPortMode	*ptpPortMode*
	)	

Configures the IEEE 1588 message detect on particular PTP port.

Parameters:

ptpPort	[in] − PTP port to config
ptpPortMode	[in]− Port to operate in Master or Slave mode

This API will enable the time stamping on a particular PTP port.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccPTPRxPoll (**IxTimeSyncAcc1588PTPPort** *ptpPort*,
IxTimeSyncAccPtpMsgData * *ptpMsgData*
)

Polls the IEEE 1588 message/time stamp detect status on a particular PTP Port on the Receive side.

Parameters:

ptpPort [in] − PTP port to poll
ptpMsgData [out] − Current TimeStamp and other Data

This API will poll for the availability of a time stamp on the received Sync (Slave) or Delay_Req (Master) messages. The client application will provide the buffer.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_NOTIMESTAMP − No time stamp available
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccPTPTxPoll (**IxTimeSyncAcc1588PTPPort** *ptpPort*,
IxTimeSyncAccPtpMsgData * *ptpMsgData*
)

Polls the IEEE 1588 message/time stamp detect status on a particular PTP Port on the Transmit side.

Parameters:

ptpPort [in] − PTP port to poll
ptpMsgData [out] − Current TimeStamp and other Data

This API will poll for the availability of a time stamp on the transmitted Sync (Master) or Delay_Req (Slave) messages. The client application will provide the buffer.

- Re−entrant : No
- ISR Callable : No

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_NOTIMESTAMP − No time stamp available
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccReset (void)

Resets the IEEE 1588 hardware assist block.

Sets the reset bit in the IEEE1588 silicon which fully resets the silicon block

- Reentrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccShow (void)

Displays the Time Sync current status.

This API will display status on the current configuration of the IEEE 1588 hardware assist block, contents of the various time stamp registers, outstanding interrupts and/or events.

Note that this is intended for debug only, and in contrast to the other functions, it does not clear the any of the status bits associated with active timestamps and so is passive in its nature.

- Reentrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccStatsGet (**IxTimeSyncAccStats** * *timeSyncStats*)

Returns the IxTimeSyncAcc Statistics in the client supplied buffer.

Parameters:

timeSyncStats [out] − TimeSync statistics counter values

This API will return the statistics of the received or transmitted messages.

NOTE: 1) These counters are updated only when the client polls for the time stamps or interrupt are enabled. This is because the IxTimeSyncAcc module does not either transmit or receive messages and does only run the code when explicit requests received by client application.

2) These statistics reflect the number of valid PTP messages exchanged in Master and Slave modes but includes all the messages (including valid non−PTP messages) while operating in the Any mode.

- Reentrant : no
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − NULL parameter passed

◊ IX_TIMESYNCACC_FAILED − Internal error occurred

void ixTimeSyncAccStatsReset (void)

Reset Time Sync statistics.

This API will reset the statistics counters of the TimeSync access layer.

- Reentrant : yes
- ISR Callable: no

Returns:

◊ None

IxTimeSyncAccStatus ixTimeSyncAccSystemTimeGet (**IxTimeSyncAccTimeValue** * *systemTime*)

Gets the System Time from the IEEE 1588 hardware assist block.

Parameters:

systemTime [out] − Copy the current System Time into the client application provided buffer

This API will get the SystemTime from IEEE1588 block and return to client

- Re−entrant : no
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccSystemTimeSet (**IxTimeSyncAccTimeValue** *systemTime*)

Sets the System Time in the IEEE 1588 hardware assist block.

Parameters:

systemTime [in] − Value to set System Time

This API will set the SystemTime to given value.

- Re−entrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful

◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTargetTimeGet (**IxTimeSyncAccTimeValue** * *targetTime*)

Gets the Target Time in the IEEE 1588 hardware assist block.

Parameters:

targetTime [out] − Copy current time to client provided buffer

This API will get the Target Time from IEEE 1588 block and return to the client application

- Re−entrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Null parameter passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTargetTimeInterruptDisable (void)

Disables the interrupt for the condition explained in the function description of **ixTimeSyncAccTargetTimeInterruptEnable**.

This API will disable the Target Time interrupt.

NOTE: The client application needs to ensure that the APIs **ixTimeSyncAccTargetTimeInterruptEnable**, **ixTimeSyncAccTargetTimeSet** and **ixTimeSyncAccTargetTimeInterruptDisable** are accessed in mutual exclusive manner with respect to each other.

- Re−entrant : no
- ISR Callable : yes

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTargetTimeInterruptEnable (**IxTimeSyncAccTargetTimeCallback***targetTimeCallback*)

Enables the interrupt to verify the condition where the System Time greater or equal to the Target Time in the IEEE 1588 hardware assist block. If the condition is true an interrupt will be sent to Intel XScale(R) Processor.

Parameters:

targetTimeCallback [in] − Callback to be invoked when interrupt fires

This API will enable the Target Time reached/hit condition interrupt.

NOTE: The client application needs to ensure that the APIs **ixTimeSyncAccTargetTimeInterruptEnable**, **ixTimeSyncAccTargetTimeSet** and **ixTimeSyncAccTargetTimeInterruptDisable** are accessed in mutual exclusive manner with respect to each other.

- Re−entrant : no
- ISR Callable : yes

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Null parameter passed for callback
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTargetTimePoll (BOOL * *ttmPollFlag*,
IxTimeSyncAccTimeValue * *targetTime*
)

Poll to verify the condition where the System Time greater or equal to the Target Time in the IEEE 1588 hardware assist block. If the condition is true an event flag is set in the hardware.

Parameters:

ttmPollFlag [out] − TRUE if the target time reached/hit condition event set FALSE if the target time reached/hit condition event is not set
targetTime [out] − Capture current targetTime into client provided buffer

Poll the target time reached/hit condition status. Return true and the current target time value, if the condition is true else return false.

NOTE: The client application will need to clear the event flag that will be set as long as the condition that the System Time greater or equal to the Target Time is valid, in one of the following ways: 1) Invoke the API to change the target time 2) Change the system timer value

- Re−entrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Null parameter passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred
◊ IX_TIMESYNCACC_INTERRUPTMODEINUSE − Interrupt mode in use

IxTimeSyncAccStatus ixTimeSyncAccTargetTimeSet (**IxTimeSyncAccTimeValue** *targetTime*)

Sets the Target Time in the IEEE 1588 hardware assist block.

Parameters:

targetTime [in] − Value to set Target Time

This API will set the Target Time to a given value.

NOTE: The client application needs to ensure that the APIs **ixTimeSyncAccTargetTimeInterruptEnable**, **ixTimeSyncAccTargetTimeSet** and **ixTimeSyncAccTargetTimeInterruptDisable** are accessed in mutual exclusive manner with respect to each other.

- Reentrant : no
- ISR Callable : yes

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTickRateGet (UINT32 * *tickRate*)

Gets the Tick Rate (Frequency Scaling Value) from the IEEE 1588 hardware assist block.

Parameters:

tickRate [out] − Current Tick Rate value in the IEEE 1588 block

This API will get the TickRate on IEE15588 block. Refer to **ixTimeSyncAccTickRateSet** for notes on usage of this value.

- Reentrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_INVALIDPARAM − Invalid parameters passed
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccTickRateSet (UINT32 *tickRate*)

Sets the Tick Rate (Frequency Scaling Value) in the IEEE 1588 hardware assist block.

Parameters:

tickRate [in] − Value to set Tick Rate

This API will set the Tick Rate (Frequency Scaling Value) in the IEEE 1588 block to the given value. The Accumulator register (not client visible) is incremented by this TickRate value every clock cycle. When the Accumulator overflows, the SystemTime is incremented by one. This TickValue can therefore be used to adjust the system timer.

- Re−entrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful

◊ IX_TIMESYNCACC_FAILED − Internal error occurred

IxTimeSyncAccStatus ixTimeSyncAccUnInit (void)

This API is used to unintialize the timesyncAcc component from the client application.

It performs the following tasks as part of unintialization: => Unmaps the memory of Block and Register level registers starting base address. => Unbinds the timesyncAcc IRQ => Resets the fused out state of all NPEs to false. => Resets the 1588 Hardware Assist block enabled flag to false and also destroys the system time mutex.

- Reentrant : yes
- ISR Callable : no

Returns:

◊ IX_TIMESYNCACC_SUCCESS − Operation is successful
◊ IX_TIMESYNCACC_FAILED − Internal error occurred

Intel (R) IXP400 Software Types (IxTypes)

Basic data types used by the IXP400 project.

Defines

#define **OK**
#define **ERROR**

Typedefs

typedef int(* **FUNCPTR**)(void)
typedef int **STATUS**

Detailed Description

Basic data types used by the IXP400 project.

Intel (R) IXP400 Software UART Access (IxUARTAcc) API

IXP400 UARTAcc Driver Public API.

Modules

Defines for Default Values
Default values which can be used for UART configuration.

Defines for IOCTL Commands
*IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.*

Defines for IOCTL Arguments *POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.*

Data Structures

struct **ixUARTDev** *Device descriptor for the UART.*

struct **ixUARTDev** *Device descriptor for the UART.*

struct **ixUARTStats** *Statistics for the UART.*

struct **ixUARTStats** *Statistics for the UART.*

Enumerations

enum **ixUARTMode** {
INTERRUPT,
POLLED,
LOOPBACK
} *The mode to set to UART to.*

Functions

PUBLIC IX_STATUS **ixUARTInit** (**ixUARTDev** *pUART) *Initialise the UART. This puts the chip in a quiescent state.*

PUBLIC IX_STATUS **ixUARTPollOutput** (**ixUARTDev** *pUART, int outChar) *Transmit a character in*

polled mode.

PUBLIC IX_STATUS **ixUARTPollInput** (**ixUARTDev** *pUART, char *inChar) *Receive a character in polled mode.*

PUBLIC IX_STATUS **ixUARTIoctl** (**ixUARTDev** *pUART, int cmd, void *arg) *Perform I/O control routines on the device.*

Detailed Description

IXP400 UARTAcc Driver Public API.

Enumeration Type Documentation

enum ixUARTMode

The mode to set to UART to.

Enumeration values:

INTERRUPT	Interrupt mode − Not supported yet.
POLLED	Polled mode.
LOOPBACK	Loopback mode.

Definition at line **308** of file **IxUART.h**.

Function Documentation

IX_STATUS ixUARTInit (**ixUARTDev** * *pUART*)

Initialise the UART. This puts the chip in a quiescent state.

Parameters:

pUART **ixUARTDev** [in] − pointer to UART structure describing our device.

Precondition:

The base address for the UART must contain a valid value. Also the baud rate and hardware options must contain sensible values otherwise the defaults will be used as defined in ixUART.h

Postcondition:

UART is initialized and ready to send and receive data.

Note:

This function should only be called once per device.

Return values:

IX_SUCCESS	− UART device successfully initialised.
IX_FAIL	− Critical error, device not initialised.

```
IX_STATUS ixUARTIoctl ( ixUARTDev * pUART,
                        int         cmd,
                        void *      arg
                      )
```

Perform I/O control routines on the device.

Parameters:

pUART	**ixUARTDev** [in] − pointer to UART structure describing our device.
cmd	int [in] − an ioctl request code.
arg	void* [in] − optional argument used to set the device mode, baud rate, and hardware options.

Return values:

IX_SUCCESS	− requested feature was set/read successfully.
IX_FAIL	− error setting/reading the requested feature.

See also:

IoctlCommandDefines

IoctlArgDefines

```
IX_STATUS ixUARTPollInput ( ixUARTDev * pUART,
                            char *      inChar
                          )
```

Receive a character in polled mode.

Parameters:

pUART	**ixUARTDev** [in] − pointer to UART structure describing our device.
**inChar*	char [in] − character read from the device.

Precondition:

UART device must be initialised.

Return values:

IX_SUCCESS	− character was successfully read.
IX_FAIL	− input buffer empty (try again).

```
IX_STATUS ixUARTPollOutput ( ixUARTDev * pUART,
                             int         outChar
                           )
```

Transmit a character in polled mode.

Parameters:

pUART **ixUARTDev** [out] − pointer to UART structure describing our device.

outChar int [out] − character to transmit.

Precondition:

UART device must be initialised.

Return values:

IX_SUCCESS	− character was successfully transmitted.
IX_FAIL	− output buffer is full (try again).

Defines for Default Values

[Intel (R) IXP400 Software UART Access (IxUARTAcc) API]

Default values which can be used for UART configuration.

Defines

#define **IX_UART_DEF_OPTS**
The default hardware options to set the UART to − no flow control, 8 bit word, 1 stop bit, no parity.

#define **IX_UART_DEF_XMIT**
The default UART FIFO size − must be no bigger than 64.

#define **IX_UART_DEF_BAUD**
The default UART baud rate − 9600.

#define **IX_UART_MIN_BAUD**
The minimum UART baud rate − 9600.

#define **IX_UART_MAX_BAUD**
The maximum UART baud rate − 926100.

#define **IX_UART_XTAL**
The UART clock speed.

Detailed Description

Default values which can be used for UART configuration.

See also:
ixUARTDev

Define Documentation

```
#define IX_UART_DEF_BAUD
```

The default UART baud rate − 9600.

Definition at line **72** of file **IxUART.h**.

```
#define IX_UART_DEF_OPTS
```

The default hardware options to set the UART to − no flow control, 8 bit word, 1 stop bit, no parity.

Definition at line **54** of file **IxUART.h**.

```
#define IX_UART_DEF_XMIT
```

The default UART FIFO size − must be no bigger than 64.

Definition at line **63** of file **IxUART.h**.

```
#define IX_UART_MAX_BAUD
```

The maximum UART baud rate − 926100.

Definition at line **90** of file **IxUART.h**.

```
#define IX_UART_MIN_BAUD
```

The minimum UART baud rate − 9600.

Definition at line **81** of file **IxUART.h**.

```
#define IX_UART_XTAL
```

The UART clock speed.

Definition at line **99** of file **IxUART.h**.

Defines for IOCTL Commands
[Intel (R) IXP400 Software UART Access (IxUARTAcc) API]

IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.

Defines

#define **IX_BAUD_SET**
Set the baud rate.

#define **IX_BAUD_GET**
Get the baud rate.

#define **IX_MODE_SET**
Set the UART mode of operation.

#define **IX_MODE_GET**
Get the current UART mode of operation.

#define **IX_OPTS_SET**
Set the UART device options.

#define **IX_OPTS_GET**
Get the UART device options.

#define **IX_STATS_GET**
Get the UART statistics.

Detailed Description

IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.

Define Documentation

```
#define IX_BAUD_GET
```

Get the baud rate.

Definition at line **129** of file **IxUART.h**.

```
#define IX_BAUD_SET
```

Set the baud rate.

Definition at line **120** of file **IxUART.h**.

```
#define IX_MODE_GET
```

Get the current UART mode of operation.

Definition at line **145** of file **IxUART.h**.

```
#define IX_MODE_SET
```

Set the UART mode of operation.

Definition at line **136** of file **IxUART.h**.

```
#define IX_OPTS_GET
```

Get the UART device options.

Definition at line **163** of file **IxUART.h**.

```
#define IX_OPTS_SET
```

Set the UART device options.

Definition at line **154** of file **IxUART.h**.

```
#define IX_STATS_GET
```

Get the UART statistics.

Definition at line **172** of file **IxUART.h**.

Defines for IOCTL Arguments

[Intel (R) IXP400 Software UART Access (IxUARTAcc) API]

POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.

Defines

#define **CLOCAL**
Software flow control.

#define **CREAD**
Enable interrupt receiver.

#define **CSIZE**
Characters size.

#define **CS5**
5 bits

#define **CS6**
6 bits

#define **CS7**
7 bits

#define **CS8**
8 bits

#define **STOPB**
Send two stop bits (else one).

#define **PARENB**
Parity detection enabled (else disabled).

#define **PARODD**
Odd parity (else even).

Detailed Description

POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.

See also:
ixUARTMode

Define Documentation

#define CLOCAL

Software flow control.

Definition at line **197** of file **IxUART.h**.

#define CREAD

Enable interrupt receiver.

Definition at line **209** of file **IxUART.h**.

#define CS5

5 bits

Definition at line **233** of file **IxUART.h**.

#define CS6

6 bits

Definition at line **245** of file **IxUART.h**.

#define CS7

7 bits

Definition at line **257** of file **IxUART.h**.

#define CS8

8 bits

Definition at line **269** of file **IxUART.h**.

#define CSIZE

Characters size.

Definition at line **221** of file **IxUART.h**.

#define PARENB

Parity detection enabled (else disabled).

Definition at line **290** of file **IxUART.h**.

#define PARODD

Odd parity (else even).

Definition at line **302** of file **IxUART.h**.

#define STOPB

Send two stop bits (else one).

Definition at line **278** of file **IxUART.h**.

Intel (R) IXP400 Software Version ID (IxVersionId)

Version Identifiers.

Defines

#define **IX_VERSION_ID**
Version Identifier String.

#define **IX_VERSION_USBRNDIS_ID**
This string will be updated with each customer release of the IXP400 USB Client driver package.

#define **IX_VERSION_I2C_LINUX_ID**
This string will be updated with each customer release of the IXP400 I2C Linux driver package.*

#define **LINUX_ETHERNET_DRIVER_PATCH_ID**
Linux Ethernet Driver Patch Version Identifier String.*

#define **LINUX_ETHERNET_README_ID**
Linux Ethernet Readme version Identifier String.*

#define **LINUX_I2C_DRIVER_README_ID**
Linux I2C driver Readme version Identifier String.*

#define **IX_VERSION_INTERNAL_ID**
Internal Release Identifier String.

#define **IX_VERSION_COMPATIBLE_TORNADO**
Compatible Tornado Version Identifier.

#define **IX_VERSION_COMPATIBLE_LINUX**
Compatible Linux Version Identifier.*

Detailed Description

Version Identifiers.

Define Documentation

```
#define IX_VERSION_COMPATIBLE_LINUX
```

Compatible Linux* Version Identifier.

Definition at line **81** of file **IxVersionId.h**.

#define IX_VERSION_COMPATIBLE_TORNADO

Compatible Tornado Version Identifier.

Definition at line **76** of file **IxVersionId.h**.

#define IX_VERSION_I2C_LINUX_ID

This string will be updated with each customer release of the IXP400 I2C Linux* driver package.

Definition at line **42** of file **IxVersionId.h**.

#define IX_VERSION_ID

Version Identifier String.

This string will be updated with each customer release of the IXP400 Software.

Definition at line **30** of file **IxVersionId.h**.

#define IX_VERSION_INTERNAL_ID

Internal Release Identifier String.

This string will be updated with each internal release (SQA drop) of the IXP400 Software.

Definition at line **71** of file **IxVersionId.h**.

#define IX_VERSION_USBRNDIS_ID

This string will be updated with each customer release of the IXP400 USB Client driver package.

Definition at line **36** of file **IxVersionId.h**.

#define LINUX_ETHERNET_DRIVER_PATCH_ID

Linux* Ethernet Driver Patch Version Identifier String.

This string will be updated with each release of Linux* Ethernet Patch

Definition at line **49** of file **IxVersionId.h**.

#define LINUX_ETHERNET_README_ID

Linux* Ethernet Readme version Identifier String.

This string will be updated with each release of Linux* Ethernet Readme

Definition at line **56** of file **IxVersionId.h**.

#define LINUX_I2C_DRIVER_README_ID

Linux* I2C driver Readme version Identifier String.

This string will be updated with each release of Linux* I2C Driver Readme

Definition at line **63** of file **IxVersionId.h**.

Intel(R) IXP400 Software USB Driver Public API

Intel(R) IXP400 Software USB Driver Public API.

Data Structures

struct **USBDevice**
USBDevice.

struct **USBSetupPacket**
Standard USB Setup packet components, see the USB Specification 1.1.

Defines

#define **IX_USB_MBLK**
Memory buffer.

#define **IX_USB_MBLK_DATA**(buf)
Return pointer to the data in the mbuf.

#define **IX_USB_MBLK_LEN**(buf)
Return pointer to the data length.

#define **IX_USB_MBLK_FREE**(buf)
#define **IX_USB_MBLK_PKT_LEN**(buf)
Return pointer to the total length of all the data in the mbuf chain for this packet.

#define **IX_USB_HAS_GET_ERROR_STRING**
*define to enable **ixUSBErrorStringGet()***

#define **IX_USB_HAS_ENDPOINT_INFO_SHOW**
*define to enable **ixUSBEndpointInfoShow()***

#define **IX_USB_HAS_STATISTICS_SHOW**
*define to enable **ixUSBStatisticsShow()***

#define **IX_USB_STATS_SHOW_PER_ENDPOINT_INFO**
*define to enable per−endpoint information in **ixUSBStatisticsShow()***

#define **IX_USB_HAS_VERBOSE_WARN_TRACE_MACRO**
define to enable verbose warning tracing

#define **IX_USB_HAS_CRITICAL_DATA_LOCKS**
define to enable critical data sections locking

#define **IX_USB_HAS_ASSERT_MACRO**

define to enable assertion macro

#define **IX_USB_HAS_CT_ASSERT_MACRO**
define to enable compile−time assertion macro

#define **IX_USB_HAS_INT_BIND_MACRO**
*define to enable interrupt handler binding for VxWorks**

#define **logMsg**
#define **UDC_REGISTERS_BASE**
Base I/O address.

#define **UDC_IRQ**
IRQ.

#define **NUM_ENDPOINTS**
Number of endpoints.

#define **SETUP_PACKET_SIZE**
SETUP packet size.

#define **CONTROL_FIFO_SIZE**
CONTROL endpoint FIFO depth.

#define **CONTROL_PACKET_SIZE**
CONTROL endpoint packet size.

#define **INTERRUPT_FIFO_SIZE**
INTERRUPT endpoint FIFO depth.

#define **INTERRUPT_PACKET_SIZE**
INTERRUPT endpoint packet size.

#define **BULK_FIFO_SIZE**
BULK endpoint FIFO depth.

#define **BULK_PACKET_SIZE**
BULK endpoint packet size.

#define **ISOCHRONOUS_FIFO_SIZE**
ISOCHRONOUS endpoint FIFO depth.

#define **ISOCHRONOUS_PACKET_SIZE**
ISOCHRONOUS endpoint packet size.

#define **MAX_TRANSFER_SIZE**
Maximum data size for one transaction in bytes (bulk or control).

#define **MAX_QUEUE_SIZE**
*Maximum outgoing queue size per endpoint, in elements Uses MAX_QUEUE_SIZE * (sizeof(void *)) bytes.*

#define **MEM_POOL_SIZE**
Memory pool for data transactions.

#define **TRANSACTION_TIMEOUT_RX**
Maximum acceptable delay in transactions (timestamp ticks), Rx, 0 disables.

#define **TRANSACTION_TIMEOUT_TX**
Maximum acceptable delay in transactions (timestamp ticks), Tx, 0 disables.

#define **IX_USB_ERROR_BASE**
USB error base.

#define **IX_USB_ERROR**
error due to unknown reasons

#define **IX_USB_INVALID_DEVICE**
*invalid **USBDevice** structure passed as parameter or no device present*

#define **IX_USB_NO_PERMISSION**
no permission for attempted operation

#define **IX_USB_REDUNDANT**
redundant operation

#define **IX_USB_SEND_QUEUE_FULL**
send queue full

#define **IX_USB_NO_ENDPOINT**
invalid endpoint

#define **IX_USB_NO_IN_CAPABILITY**
no IN capability on endpoint

#define **IX_USB_NO_OUT_CAPABILITY**
no OUT capability on endpoint

#define **IX_USB_NO_TRANSFER_CAPABILITY**
transfer type incompatible with endpoint

#define **IX_USB_ENDPOINT_STALLED**
endpoint stalled

#define **IX_USB_INVALID_PARMS**
invalid parameter(s)

#define **IX_USB_DEVICE_DISABLED**
device is disabled

#define **IX_USB_NO_STALL_CAPABILITY**

no STALL capability

#define **IX_USB_DEVICE_NOT_CONFIGURED**
*device is not received SET_CONFIGURATION message **

#define **EP_DIRECTION**(x)
Macro used to extract the endpoint direction from an EPDescriptorTable[] entry.

#define **EP_TYPE**(x)
Macro used to extract the endpoint type from an EPDescriptorTable[] entry.

#define **MIN**(a, b)
Compares two values and returns the minimum.

#define **MAX**(a, b)
Compares two values and returns the maximum.

#define **QUEUE_WRAP**(tail)
Ajusts the tail of a queue implemented in a circular buffer by wrapping at the buffer boundary.

#define **SWAP_USB_WORD**(wPtr)
USB byte swapping routine for a little endian platform.

#define **REG_GET**(reg_ptr)
read generic register access via register pointers

#define **REG_SET**(reg_ptr, val)
write generic register access via register pointers

#define **DREG_GET**(reg_ptr)
generic data register read access via register pointers

#define **DREG_SET**(reg_ptr, val)
generic data register write access via register pointers

#define **CONTEXT**(device)
get context from device pointer

#define **REGISTERS**(device)
get registers from device pointer

#define **EP0CONTROL**(device)
get endpoint 0 control data from device pointer

#define **EVENTS**(device)
get event processor from device pointer

#define **COUNTERS**(device)
get device counters

#define **OPERATION**(device)
get device operation

#define **EPSTATUS**(device, endpointNumber)
get endpoint status from device pointer and endpoint number

#define **EPQUEUE**(device, endpointNumber)
get endpoint queue from device pointer and endpoint number

#define **EPCOUNTERS**(device, endpointNumber)
get endpoint counters from device pointer and endpoint number

#define **RETURN_OK**(device)
set IX_SUCCESS on device and return IX_SUCCESS

#define **RETURN_ERROR**(device)
set IX_USB_ERROR on device and return IX_FAIL

#define **RETURN_INVALID_PARMS**(device)
set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define **RETURN_REDUNDANT**(device)
set IX_USB_REDUNDANT on device and return IX_FAIL

#define **RETURN_INVALID_DEVICE**(device)
set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define **RETURN_NO_ENDPOINT**(device)
set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define **RETURN_ENDPOINT_STALLED**(device)
set IX_USB_ENDPOINT_STALLED on device and return IX_FAIL

#define **RETURN_SEND_QUEUE_FULL**(device)
set IX_USB_SEND_QUEUE_FULL on device and return IX_FAIL

#define **RETURN_NO_IN_CAPABILITY**(device)
set IX_USB_NO_IN_CAPABILITY on device and return IX_FAIL

#define **RETURN_NO_STALL_CAPABILITY**(device)
set IX_USB_NO_STALL_CAPABILITY on device and return IX_FAIL

#define **RETURN_NO_PERMISSION**(device)
set IX_USB_NO_PERMISSION on device and return IX_FAIL

#define **CHECK_DEVICE**(device)
sanity checks for device existence

#define **CHECK_DEVICE_ENABLED**(device)
sanity checks for device enable status

#define **CHECK_DEVICE_CONFIGURED**(device)
sanity checks for device cofigure status

#define **CHECK_ENDPOINT**(device, endpointNumber)
sanity check for endpoint existence

#define **CHECK_ENDPOINT_STALL**(device, endpointNumber)
sanity check for endpoint stall

#define **CHECK_EVENT_MASK**(device, eventMask)
sanity check for event masks

#define **CHECK_ENDPOINT_QUEUE**(epData)
sanity check for endpoint queue size

#define **CHECK_ENDPOINT_IN_CAPABILITY**(epData, device)
sanity check for endpoint IN capability

#define **IX_USB_LOG_DEVICE**
#define **IX_USB_HAS_VERBOSE_TRACE_MACRO**
#define **IX_USB_TRACE**(format, a, b, c, d, e, f)
no trace macro

#define **IX_USB_LOCK**
dummy critial data section lock

#define **IX_USB_UNLOCK**(state)
dummy critial data section unlock

#define **IX_USB_IRQ_LOCK**
dummy irq lock

#define **IX_USB_IRQ_UNLOCK**(state)
dummy irq unlock

#define **IX_USB_DRAIN_FIFO**(registers)
#define **USB_CONTEXT_SIZE**
USB context size.

Typedefs

typedef UINT16 **USBEventSet**
typedef void(* **USBEventCallback**)(**USBDevice** *device, USBEventSet events)
typedef void(* **USBSetupCallback**)(**USBDevice** *device, const char *packet)
typedef void(* **USBReceiveCallback**)(**USBDevice** *device, UINT16 sourceEndpoint, IX_USB_MBLK *receiveBuffer)

Enumerations

enum **USBEndpointDirection** {
USB_NO_DATA,
USB_IN,
USB_OUT,
USB_IN_OUT
}
USB endpoint direction.

enum **USBEndpointType** {
USB_CONTROL,
USB_BULK,
USB_INTERRUPT,
USB_ISOCHRONOUS
}
Note: the values are set for compatibility with USBEndpointDirection.

enum **USBEventMap** {
USB_NO_EVENT,
USB_RESET,
USB_SUSPEND,
USB_RESUME,
USB_SOF,
USB_DEVICE_EVENTS,
USB_BUS_EVENTS,
USB_ALL_EVENTS
}
USB Event Map.

enum **USBDeviceFlags** {
ENABLE_RX_SEQ,
ENABLE_TX_SEQ,
ENABLE_BULK_NAK_THROTTLE
}
USB Device Flags.

enum **USBEndpointNumber** {
ENDPOINT_0,
ENDPOINT_1,
ENDPOINT_2,
ENDPOINT_3,
ENDPOINT_4,
ENDPOINT_5,
ENDPOINT_6,
ENDPOINT_7,
ENDPOINT_8,
ENDPOINT_9,
ENDPOINT_10,
ENDPOINT_11,
ENDPOINT_12,

ENDPOINT_13,
ENDPOINT_14,
ENDPOINT_15
}
USB endpoint number.

enum **USBStdRequestType** {
GET_STATUS_REQUEST,
CLEAR_FEATURE_REQUEST,
SET_FEATURE_REQUEST,
SET_ADDRESS_REQUEST,
GET_DESCRIPTOR_REQUEST,
SET_DESCRIPTOR_REQUEST,
GET_CONFIGURATION_REQUEST,
SET_CONFIGURATION_REQUEST,
GET_INTERFACE_REQUEST,
SET_INTERFACE_REQUEST,
SYNCH_FRAME_REQUEST
}
Standard USB request types.

enum **USBStdDescriptorType** {
USB_DEVICE_DESCRIPTOR,
USB_CONFIGURATION_DESCRIPTOR,
USB_STRING_DESCRIPTOR,
USB_INTERFACE_DESCRIPTOR,
USB_ENDPOINT_DESCRIPTOR
}
Standard USB descriptor types.

enum **USBStdFeatureSelector** {
ENDPOINT_STALL,
DEVICE_REMOTE_WAKEUP
}
Standard USB SET/CLEAR_FEATURE feature selector.

enum **USBStdLanguageId** { **USB_ENGLISH_LANGUAGE** }
Standard language IDs used by USB.

enum **USBStdEndpointType** {
USB_CONTROL_ENDPOINT,
USB_ISOCHRONOUS_ENDPOINT,
USB_BULK_ENDPOINT,
USB_INTERRUPT_ENDPOINT
}
Standard USB endpoint types.

enum **USBStdEndpointDirection** {
USB_ENDPOINT_OUT,
USB_ENDPOINT_IN
}

Standard USB directions.

Functions

PUBLIC void **ixUSBDataSendAllow** (**USBDevice** *)
enable to send requests for USB_IN endpoints

PUBLIC void **ixUSBDataSendBlock** (void)

PUBLIC IX_STATUS **ixUSBDriverInit** (**USBDevice** *device)
Initialize driver and USB Device Controller.

PUBLIC IX_STATUS **ixUSBDeviceEnable** (**USBDevice** *device, BOOL enableDevice)
Enable or disable the device.

PUBLIC IX_STATUS **ixUSBEndpointStall** (**USBDevice** *device, UINT16 endpointNumber, BOOL stallFlag)
Enable or disable endpoint stall (or halt feature).

PUBLIC IX_STATUS **ixUSBEndpointClear** (**USBDevice** *device, UINT16 endpointNumber)
Free all Rx/Tx buffers associated with an endpoint.

PUBLIC IX_STATUS **ixUSBSignalResume** (**USBDevice** *device)
Trigger signal resuming on the bus.

PUBLIC IX_STATUS **ixUSBFrameCounterGet** (**USBDevice** *device, UINT16 *counter)
Retrieve the 11−bit frame counter.

PUBLIC IX_STATUS **ixUSBReceiveCallbackRegister** (**USBDevice** *device, USBReceiveCallback callbackFunction)
Register a data receive callback.

PUBLIC IX_STATUS **ixUSBSetupCallbackRegister** (**USBDevice** *device, USBSetupCallback callbackFunction)
Register a setup receive callback.

PUBLIC IX_STATUS **ixUSBBufferSubmit** (**USBDevice** *device, UINT16 destinationEndpoint, IX_USB_MBLK *sendBuffer)
Submit a buffer for transmit.

PUBLIC IX_STATUS **ixUSBBufferCancel** (**USBDevice** *device, UINT16 destinationEndpoint, IX_USB_MBLK *sendBuffer)
Cancel a buffer previously submitted for transmitting.

PUBLIC IX_STATUS **ixUSBEventCallbackRegister** (**USBDevice** *device, USBEventCallback eventCallback, **USBEventMap** eventMap)
Register an event callback.

PUBLIC IX_STATUS **ixUSBIsEndpointStalled** (**USBDevice** *device, UINT16 endpointNumber, BOOL *stallState)

Retrieve an endpoint's stall status.

PUBLIC
IX_USB_MBLK * **ixUSBBufferAlloc** (size_t size)
Allocates a buffer from the USB driver buffer pool.

PUBLIC IX_STATUS **ixUSBStatisticsShow** (**USBDevice** *device)
Display device state and statistics.

PUBLIC const char * **ixUSBErrorStringGet** (UINT32 errorCode)
Convert an error code into a human−readable string error message.

PUBLIC IX_STATUS **ixUSBEndpointInfoShow** (**USBDevice** *device)
Display endpoint information table.

void **ixUSBMblkFree** (IX_USB_MBLK *)
Returns a buffer to the buffer pool.

Detailed Description

Intel(R) IXP400 Software USB Driver Public API.

Define Documentation

```
#define BULK_FIFO_SIZE
```

BULK endpoint FIFO depth.

Definition at line **61** of file **usbdeviceparam.h**.

```
#define BULK_PACKET_SIZE
```

BULK endpoint packet size.

Definition at line **64** of file **usbdeviceparam.h**.

```
#define CHECK_DEVICE ( device )
```

sanity checks for device existence

Definition at line **237** of file **usbmacros.h**.

```
#define CHECK_DEVICE_CONFIGURED ( device )
```

sanity checks for device cofigure status

Definition at line **257** of file **usbmacros.h**.

```
#define CHECK_DEVICE_ENABLED ( device )
```

sanity checks for device enable status

Definition at line **249** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT ( device,
                         endpointNumber )
```

sanity check for endpoint existence

Definition at line **269** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_IN_CAPABILITY ( epData,
                                       device )
```

sanity check for endpoint IN capability

Definition at line **302** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_QUEUE ( epData )
```

sanity check for endpoint queue size

Definition at line **295** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_STALL ( device,
                               endpointNumber )
```

sanity check for endpoint stall

Definition at line **276** of file **usbmacros.h**.

```
#define CHECK_EVENT_MASK ( device,
                           eventMask )
```

sanity check for event masks

Definition at line **288** of file **usbmacros.h**.

```
#define CONTEXT ( device )
```

get context from device pointer

Definition at line **155** of file **usbmacros.h**.

```
#define CONTROL_FIFO_SIZE
```

CONTROL endpoint FIFO depth.

Definition at line **49** of file **usbdeviceparam.h**.

```
#define CONTROL_PACKET_SIZE
```

CONTROL endpoint packet size.

Definition at line **52** of file **usbdeviceparam.h**.

```
#define COUNTERS ( device )
```

get device counters

Definition at line **167** of file **usbmacros.h**.

```
#define DREG_GET ( reg_ptr )
```

generic data register read access via register pointers

Definition at line **128** of file **usbmacros.h**.

```
#define DREG_SET ( reg_ptr,
                   val )
```

generic data register write access via register pointers

Definition at line **130** of file **usbmacros.h**.

```
#define EP0CONTROL ( device )
```

get endpoint 0 control data from device pointer

Definition at line **161** of file **usbmacros.h**.

```
#define EP_DIRECTION ( x )
```

Macro used to extract the endpoint direction from an EPDescriptorTable[] entry.

Parameters:

x int (in) − the endpoint description entry

Returns:

the endpoint direction (USB_IN, USB_OUT or USB_IN_OUT)

Definition at line **44** of file **usbmacros.h**.

```
#define EP_TYPE ( x )
```

Macro used to extract the endpoint type from an EPDescriptorTable[] entry.

Parameters:

x int (in) − the endpoint description entry

Returns:

the endpoint type (USB_CONTROL, USB_BULK, USB_ISOCHRONOUS, USB_INTERRUPT)

Definition at line **56** of file **usbmacros.h**.

```
#define EPCOUNTERS ( device,
                     endpointNumber )
```

get endpoint counters from device pointer and endpoint number

Definition at line **179** of file **usbmacros.h**.

```
#define EPQUEUE ( device,
                  endpointNumber )
```

get endpoint queue from device pointer and endpoint number

Definition at line **176** of file **usbmacros.h**.

```
#define EPSTATUS ( device,
                   endpointNumber )
```

get endpoint status from device pointer and endpoint number

Definition at line **173** of file **usbmacros.h**.

```
#define EVENTS ( device )
```

get event processor from device pointer

Definition at line **164** of file **usbmacros.h**.

```
#define INTERRUPT_FIFO_SIZE
```

INTERRUPT endpoint FIFO depth.

Definition at line **55** of file **usbdeviceparam.h**.

```
#define INTERRUPT_PACKET_SIZE
```

INTERRUPT endpoint packet size.

Definition at line **58** of file **usbdeviceparam.h**.

```
#define ISOCHRONOUS_FIFO_SIZE
```

ISOCHRONOUS endpoint FIFO depth.

Definition at line **74** of file **usbdeviceparam.h**.

```
#define ISOCHRONOUS_PACKET_SIZE
```

ISOCHRONOUS endpoint packet size.

Definition at line **77** of file **usbdeviceparam.h**.

```
#define IX_USB_DEVICE_DISABLED
```

device is disabled

Definition at line **69** of file **usberrors.h**.

#define IX_USB_DEVICE_NOT_CONFIGURED

device is not received SET_CONFIGURATION message *

Definition at line **75** of file **usberrors.h**.

#define IX_USB_ENDPOINT_STALLED

endpoint stalled

Definition at line **63** of file **usberrors.h**.

#define IX_USB_ERROR

error due to unknown reasons

Definition at line **36** of file **usberrors.h**.

#define IX_USB_ERROR_BASE

USB error base.

Definition at line **32** of file **usberrors.h**.

#define IX_USB_HAS_ASSERT_MACRO

define to enable assertion macro

Definition at line **99** of file **usbconfig.h**.

#define IX_USB_HAS_CRITICAL_DATA_LOCKS

define to enable critical data sections locking

Definition at line **94** of file **usbconfig.h**.

#define IX_USB_HAS_CT_ASSERT_MACRO

define to enable compile−time assertion macro

Definition at line **102** of file **usbconfig.h**.

#define IX_USB_HAS_ENDPOINT_INFO_SHOW

define to enable **ixUSBEndpointInfoShow()**

Definition at line **36** of file **usbconfig.h**.

#define IX_USB_HAS_GET_ERROR_STRING

define to enable **ixUSBErrorStringGet()**

Definition at line **33** of file **usbconfig.h**.

#define IX_USB_HAS_INT_BIND_MACRO

define to enable interrupt handler binding for VxWorks*

Definition at line **105** of file **usbconfig.h**.

#define IX_USB_HAS_STATISTICS_SHOW

define to enable **ixUSBStatisticsShow()**

Definition at line **42** of file **usbconfig.h**.

#define IX_USB_HAS_VERBOSE_WARN_TRACE_MACRO

define to enable verbose warning tracing

Definition at line **88** of file **usbconfig.h**.

#define IX_USB_INVALID_DEVICE

invalid **USBDevice** structure passed as parameter or no device present

Definition at line **39** of file **usberrors.h**.

#define IX_USB_INVALID_PARMS

invalid parameter(s)

Definition at line **66** of file **usberrors.h**.

#define IX_USB_IRQ_LOCK

dummy irq lock

Definition at line **558** of file **usbmacros.h**.

#define IX_USB_IRQ_UNLOCK (state)

dummy irq unlock

Definition at line **561** of file **usbmacros.h**.

#define IX_USB_LOCK

dummy critial data section lock

Definition at line **552** of file **usbmacros.h**.

#define IX_USB_MBLK

Memory buffer.

Definition at line **54** of file **usbbasictypes.h**.

#define IX_USB_MBLK_DATA (buf)

Return pointer to the data in the mbuf.

Definition at line **57** of file **usbbasictypes.h**.

#define IX_USB_MBLK_LEN (buf)

Return pointer to the data length.

Definition at line **60** of file **usbbasictypes.h**.

#define IX_USB_MBLK_PKT_LEN (buf)

Return pointer to the total length of all the data in the mbuf chain for this packet.

Definition at line **67** of file **usbbasictypes.h**.

#define IX_USB_NO_ENDPOINT

invalid endpoint

Definition at line **51** of file **usberrors.h**.

#define IX_USB_NO_IN_CAPABILITY

no IN capability on endpoint

Definition at line **54** of file **usberrors.h**.

#define IX_USB_NO_OUT_CAPABILITY

no OUT capability on endpoint

Definition at line **57** of file **usberrors.h**.

#define IX_USB_NO_PERMISSION

no permission for attempted operation

Definition at line **42** of file **usberrors.h**.

#define IX_USB_NO_STALL_CAPABILITY

no STALL capability

Definition at line **72** of file **usberrors.h**.

#define IX_USB_NO_TRANSFER_CAPABILITY

transfer type incompatible with endpoint

Definition at line **60** of file **usberrors.h**.

#define IX_USB_REDUNDANT

redundant operation

Definition at line **45** of file **usberrors.h**.

```
#define IX_USB_SEND_QUEUE_FULL
```

send queue full

Definition at line **48** of file **usberrors.h**.

```
#define IX_USB_STATS_SHOW_PER_ENDPOINT_INFO
```

define to enable per−endpoint information in **ixUSBStatisticsShow()**

Definition at line **64** of file **usbconfig.h**.

```
#define IX_USB_TRACE ( format,
                       a,
                       b,
                       c,
                       d,
                       e,
                       f       )
```

no trace macro

Definition at line **381** of file **usbmacros.h**.

```
#define IX_USB_UNLOCK ( state )
```

dummy critial data section unlock

Definition at line **555** of file **usbmacros.h**.

```
#define MAX ( a,
              b )
```

Compares two values and returns the maximum.

Parameters:

a − first value

b − second value

Returns:

maximum of the two input values

Definition at line **85** of file **usbmacros.h**.

#define MAX_QUEUE_SIZE

Maximum outgoing queue size per endpoint, in elements Uses MAX_QUEUE_SIZE * (sizeof(void *)) bytes.

Definition at line **39** of file **usbdriverparam.h**.

#define MAX_TRANSFER_SIZE

Maximum data size for one transaction in bytes (bulk or control).

Definition at line **33** of file **usbdriverparam.h**.

#define MEM_POOL_SIZE

Memory pool for data transactions.

Definition at line **42** of file **usbdriverparam.h**.

#define MIN (a,
b)

Compares two values and returns the minimum.

Parameters:

a − first value

b − second value

Returns:

minimum of the two input values

Definition at line **70** of file **usbmacros.h**.

#define NUM_ENDPOINTS

Number of endpoints.

Definition at line **43** of file **usbdeviceparam.h**.

#define OPERATION (device)

get device operation

Definition at line **170** of file **usbmacros.h**.

#define QUEUE_WRAP (tail)

Ajusts the tail of a queue implemented in a circular buffer by wrapping at the buffer boundary.

Parameters:

tail int − virtual tail offset

Returns:

the real adjusted tail offset

Definition at line **99** of file **usbmacros.h**.

#define REG_GET (reg_ptr)

read generic register access via register pointers

Definition at line **123** of file **usbmacros.h**.

#define REG_SET (reg_ptr,
val)

write generic register access via register pointers

Definition at line **125** of file **usbmacros.h**.

#define REGISTERS (device)

get registers from device pointer

Definition at line **158** of file **usbmacros.h**.

#define RETURN_ENDPOINT_STALLED (device)

set IX_USB_ENDPOINT_STALLED on device and return IX_FAIL

Definition at line **212** of file **usbmacros.h**.

#define RETURN_ERROR (device)

set IX_USB_ERROR on device and return IX_FAIL

Definition at line **187** of file **usbmacros.h**.

#define RETURN_INVALID_DEVICE (device)

set IX_USB_INVALID_PARAMS on device and return IX_FAIL

Definition at line **202** of file **usbmacros.h**.

#define RETURN_INVALID_PARMS (device)

set IX_USB_INVALID_PARAMS on device and return IX_FAIL

Definition at line **192** of file **usbmacros.h**.

#define RETURN_NO_ENDPOINT (device)

set IX_USB_INVALID_PARAMS on device and return IX_FAIL

Definition at line **207** of file **usbmacros.h**.

#define RETURN_NO_IN_CAPABILITY (device)

set IX_USB_NO_IN_CAPABILITY on device and return IX_FAIL

Definition at line **222** of file **usbmacros.h**.

#define RETURN_NO_PERMISSION (device)

set IX_USB_NO_PERMISSION on device and return IX_FAIL

Definition at line **232** of file **usbmacros.h**.

#define RETURN_NO_STALL_CAPABILITY (device)

set IX_USB_NO_STALL_CAPABILITY on device and return IX_FAIL

Definition at line **227** of file **usbmacros.h**.

#define RETURN_OK (device)

set IX_SUCCESS on device and return IX_SUCCESS

Definition at line **182** of file **usbmacros.h**.

#define RETURN_REDUNDANT (device)

set IX_USB_REDUNDANT on device and return IX_FAIL

Definition at line **197** of file **usbmacros.h**.

#define RETURN_SEND_QUEUE_FULL (device)

set IX_USB_SEND_QUEUE_FULL on device and return IX_FAIL

Definition at line **217** of file **usbmacros.h**.

#define SETUP_PACKET_SIZE

SETUP packet size.

Definition at line **46** of file **usbdeviceparam.h**.

#define SWAP_USB_WORD (wPtr)

USB byte swapping routine for a little endian platform.

Definition at line **114** of file **usbmacros.h**.

#define TRANSACTION_TIMEOUT_RX

Maximum acceptable delay in transactions (timestamp ticks), Rx, 0 disables.

Definition at line **45** of file **usbdriverparam.h**.

#define TRANSACTION_TIMEOUT_TX

Maximum acceptable delay in transactions (timestamp ticks), Tx, 0 disables.

Definition at line **48** of file **usbdriverparam.h**.

#define UDC_IRQ

IRQ.

Definition at line **40** of file **usbdeviceparam.h**.

#define UDC_REGISTERS_BASE

Base I/O address.

Definition at line **37** of file **usbdeviceparam.h**.

#define USB_CONTEXT_SIZE

USB context size.

Definition at line **33** of file **usbtypes.h**.

Enumeration Type Documentation

enum USBDeviceFlags

USB Device Flags.

Definition at line **72** of file **usbconstants.h**.

enum USBEndpointDirection

USB endpoint direction.

Definition at line **33** of file **usbconstants.h**.

enum USBEndpointNumber

USB endpoint number.

Definition at line **82** of file **usbconstants.h**.

enum USBEndpointType

Note: the values are set for compatibility with USBEndpointDirection.

NB: THESE ARE NOT STANDARD USB ENDPOINT TYPES TO BE USED IN DESCRIPTORS, see **usbstd.h**

Definition at line **46** of file **usbconstants.h**.

enum USBEventMap

USB Event Map.

Enumeration values:

USB_SOF Start Of Frame.

Definition at line **57** of file **usbconstants.h**.

enum USBStdDescriptorType

Standard USB descriptor types.

Definition at line **54** of file **usbstd.h**.

enum USBStdEndpointDirection

Standard USB directions.

Definition at line **94** of file **usbstd.h**.

enum USBStdEndpointType

Standard USB endpoint types.

Definition at line **83** of file **usbstd.h**.

enum USBStdFeatureSelector

Standard USB SET/CLEAR_FEATURE feature selector.

Definition at line **66** of file **usbstd.h**.

enum USBStdLanguageId

Standard language IDs used by USB.

Definition at line **75** of file **usbstd.h**.

enum USBStdRequestType

Standard USB request types.

Definition at line **36** of file **usbstd.h**.

Function Documentation

PUBLIC IX_USB_MBLK * ixUSBBufferAlloc (size_t *size*)

Allocates a buffer from the USB driver buffer pool.

Parameters:

size size_t (in) − desired size, in bytes, of the buffer to be allocated

Returns:

a pointer to a newly allocated buffer or NULL if the allocation failed

PUBLIC IX_STATUS ixUSBBufferCancel (**USBDevice** * *device*,
UINT16 *destinationEndpoint*,
IX_USB_MBLK * *sendBuffer*
)

Cancel a buffer previously submitted for transmitting.

Parameters:

device	**USBDevice** * (in) − a structure identifying the device
destinationEndpoint	UINT16 (in) − endpoint originally used for transmitting the data buffer
sendBuffer	IX_USB_MBLK * (in) − sumbitted data buffer

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBBufferSubmit (**USBDevice** * *device*,
UINT16 *destinationEndpoint*,
IX_USB_MBLK * *sendBuffer*
)

Submit a buffer for transmit.

Parameters:

device	**USBDevice** * (in) − a structure identifying the device
destinationEndpoint	UINT16 (in) − endpoint to be used for transmitting the data buffer
sendBuffer	IX_USB_MBLK * (in) − data buffer

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC void ixUSBDataSendAllow (**USBDevice** *)

enable to send requests for USB_IN endpoints

Parameters:

device **USBDevice** * (in) − a structure identifying the device

Returns:

none

PUBLIC IX_STATUS ixUSBDeviceEnable (**USBDevice** * *device*,
BOOL *enableDevice*
)

Enable or disable the device.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
enableDevice BOOL (in) − **true** to enable the device and **false** to disable it

This function enables or disables the device. A disabled device doesn't generate events and cannot send or receive data.
Disabling the device frees and discards all existent Rx/Tx buffers (received buffers that weren't dispatched yet and buffers waiting to be transmitted)

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBDriverInit (**USBDevice** * *device*)

Initialize driver and USB Device Controller.

Parameters:

device **USBDevice** * (inout) − a structure identifying the device

This function initializes the UDC and all the data structures used to interact with the controller.
It is the responsibility of the caller to create the **USBDevice** structure and fill in the correct *baseIOAddress* and *interruptLevel* fields.
After successful initialization the device will be inactive − use **ixUSBDeviceEnable** to activate the device.
Use the *flags* component of the *device* structure to pass in additional flags such as ENABLE_RX_SEQ or ENABLE_TX_SEQ. Changing these flags later will have no effect.
The driver will assign a device number which will be placed in the *deviceIndex* field.
The initialized *device* structure must be used for all interations with the USB controller. The same *device* pointer will be passed in to all the registered client callbacks.
The *deviceIndex* and *deviceContext* should be treated as read−only fields. A check to verify that the USB device is present is performed and a warning is issued if the device is not present.

Warning:

This function is not reentrant.

Returns:

IX_SUCCESS if the initialization was successful; a warning is issued if the specified USB device is not present. IX_FAIL otherwise,in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBEndpointClear  ( USBDevice *  device,
                                       UINT16       endpointNumber
                                     )
```

Free all Rx/Tx buffers associated with an endpoint.

Parameters:

device	**USBDevice** * (in) − a structure identifying the device
endpointNumber	UINT16 (in) − endpoint number

This function discards and frees all Tx/Rx buffers associated with an endpoint. The corresponding endpoint dropped packet counters will also be incremented.

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC void ixUSBEndpointInfoShow  ( USBDevice *  device  )
```

Display endpoint information table.

Parameters:

device	**USBDevice** * (in) − a structure identifying the device

Returns:

none

```
PUBLIC IX_STATUS ixUSBEndpointStall  ( USBDevice *  device,
                                       UINT16       endpointNumber,
                                       BOOL         stallFlag
                                     )
```

Enable or disable endpoint stall (or *halt* feature).

Parameters:

device	**USBDevice** * (in) − a structure identifying the device
endpointNumber	UINT16 (in) − endpoint number

stallFlag BOOL (in) − **true** to set endpoint stall and **false** to clear it

This function clears or sets the endpoint stall (or *halt*) feature.
Both IN and OUT endpoints can be stalled. A stalled endpoint will not send or receive data. Instead, it will send USB STALL packets in response to IN or OUT tokens.
Unstalling endpoints can be done only by using this function with the exception of endpoint 0 which unstalls itself automatically upon receiving a new SETUP packet, as required by the USB 1.1 Specification.
Isochronous endpoints cannot be stalled and attempting to do so will return an IX_USB_NO_STALL_CAPABILITY failure.

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC const char * ixUSBErrorStringGet (UINT32 *errorCode*)

Convert an error code into a human−readable string error message.

Parameters:

errorCode UINT32 (in) − erorr code as defined in **usberrors.h**

Returns:

a `const char *` pointer to the error message

PUBLIC IX_STATUS ixUSBEventCallbackRegister (**USBDevice** * *device*,
USBEventCallback *eventCallback*,
USBEventMap *eventMap*
)

Register an event callback.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
eventCallback USBEventCallback (in) − event callback function
eventMap USBEventMap (in) − event map

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBFrameCounterGet (**USBDevice** * *device*,
UINT16 * *counter*
)

Retrieve the 11−bit frame counter.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
counter UINT16 * (out) − the 11−bit frame counter

This function returns the hardware USB frame counter.
Since the counter is 11−bit wide it rolls over after every 2048 frames.

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBIsEndpointStalled ( USBDevice *  device,
                                          UINT16       endpointNumber,
                                          BOOL *       stallState
                                        )
```

Retrieve an endpoint's stall status.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
endpointNumber UINT16 (in) − endpoint number
stallState BOOL * (out) − stall state; **true** if the endpoint is stalled (*halted*) or **false** otherwise**true**

Returns:

IX_SUCCESS or IX_FAIL if the device pointer is invalid or the endpoint doesn't exist

```
void ixUSBMblkFree ( IX_USB_MBLK *  )
```

Returns a buffer to the buffer pool.

```
PUBLIC IX_STATUS ixUSBReceiveCallbackRegister ( USBDevice *          device,
                                                USBReceiveCallback   callbackFunction
                                              )
```

Register a data receive callback.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
callbackFunction USBReceiveCallback (in) − receive callback function

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBSetupCallbackRegister (**USBDevice** * *device*,
USBSetupCallback *callbackFunction*
)

Register a setup receive callback.

Parameters:

device **USBDevice** * (in) − a structure identifying the device
callbackFunction USBSetupCallback (in) − setup callback function

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBSignalResume (**USBDevice** * *device*)

Trigger signal resuming on the bus.

Parameters:

device **USBDevice** * (in) − a structure identifying the device

This function triggers signal resuming on the bus, waking up the USB host. Is should be used only if the host has enabled the device to do so using the standard SET_FEATURE USB request, otherwise the function will return IX_FAIL and set the *lastError* field to IX_USB_NO_PERMISSION.

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC IX_STATUS ixUSBStatisticsShow (**USBDevice** * *device*)

Display device state and statistics.

Parameters:

device **USBDevice** * (in) − a structure identifying the device

Returns:

IX_SUCCESS if the initialization was successful, IX_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

Intel (R) IXP400 Software Codelets

Intel (R) IXP400 Software Codelets.

Modules

Intel (R) IXP400 ATM Codelet (IxAtmCodelet) API

This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

Intel (R) IXP400 Software DMA Access Codelet (IxDmaAccCodelet) API

Intel (R) IXP400 Software DMA Access component API.

Intel (R) IXP400 Software Ethernet Access Codelet (IxEthAccCodelet) API *Intel (R) IXP400 Software Ethernet Access Codelet API.*

Intel (R) IXP400 Software HSS Access Codelet (IxHssAccCodelet) API *Intel (R) IXP400 Software HSS Access Codelet API. The interface for the HSS Access Codelet.*

Intel (R) IXP400 Software USB RNDIS Codelet (IxUSBRNDIS) API *Intel (R) IXP400 Software codelet USB RNDIS API.*

Intel (R) IXP400 Software Parity Error Notifier Access Codelet *Intel (R) IXP400 Software Parity Error Notifier Access Codelet.*

Intel (R) IXP400 Software Time Sync Access Codelet *Intel (R) IXP400 Software Time Sync Access Codelet.*

Detailed Description

Intel (R) IXP400 Software Codelets.

Intel (R) IXP400 ATM Codelet (IxAtmCodelet) API

[Intel (R) IXP400 Software Codelets]

This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

Defines

#define **IX_ATM_CODELET_SWLOOPBACK_PORT_RATE**
Port rate for Software Loopback.

#define **IX_ATM_CODELET_REMOTELOOPBACK_PORT_RATE**
Port rate for Remote Looopback.

#define **IX_ATMCODELET_START_VPI**
The first VPI value.

#define **IX_ATMCODELET_START_VCI**
The first VCI value.

#define **IX_ATMCODELET_NUM_AAL5_CHANNELS**
32 Channels for AAL5

#define **IX_ATMCODELET_NUM_AAL0_48_CHANNELS**
32 Channels for AAL0 48−bytes

#define **IX_ATMCODELET_NUM_AAL0_52_CHANNELS**
32 Channels for AAL0 52−bytes

#define **IX_ATMCODELET_QMGR_DISPATCHER_PRIORITY**
Recommended priority of queue manager dispatch loop.

#define **IX_ATMCODELET_QMGR_DISPATCHER_THREAD_STACK_SIZE**
Recommended stack size for queue manager dispatcher thread.

#define **IX_ATMCODELET_UTOPIA_LB_THREAD_PRIORITY**
Recommended priority of UTOPIA loopback thread (for AAL0 and AAL5 type cells).

Functions

PUBLIC IX_STATUS **ixAtmCodeletMain** (**IxAtmCodeletMode** modeType, **IxAtmCodeletAalType** aalType)
This function is used as a single point of execution for ATM codelet.

Detailed Description

This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

This codelet also demonstrates an example implementation of OAM F4 Segment, F4 End−To−End (ETE), F5 Segment and F5 ETE loopback that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

Disclaimer Note: For Linux* Platform

- When 'insmod' the ATM codelet object, it will begin to send AAL packets and display the transmit and receive statistics every 15 second
- Unable to 'rmmod'. User will not be able to type anything in the command line due to: a) the return carriage indicating that the task is sending AAL packets, and b) the failure to kill the thread which is used to transmit AAL packets

VxWorks* User Guide
ixAtmCodeletMain() function is used as a single point of execution for Atm Codelet, which allows the user to enter selections for different type of modes and AAL type. The function also allows the user to execute OAM ping either in UTOPIA or Software Loopback mode. In all modes, the transmit and receive statistics will be displayed every 15secs.

```
Usage :
   ixAtmCodeletMain (modeType, aalType)
       modeType:
               0 = Utopia Loopback Mode
               1 = Software Loopback Mode
               2 = Remote Loopback Mode
               3 = F4 & F5 cells OAM Ping in UTOPIA Loopback mode
               4 = F4 & F5 cells OAM Ping in Software Loopback mode
       aalType:
               1 = AAL5
               2 = AAL0_48
               3 = AAL0_52
```

Linux* User Guide
The idea of using the **ixAtmCodeletMain()** as a single point of execution for ATM codelet is similar in VxWorks* User Guide. It also allows user to execute OAM ping. Similarly, all modes will display the transmit and receive statistics every 15secs. This function will be executed when user issue'insmod' in command prompt

```
Usage :
   # insmod ixp400_codelets_atm.o modeType= aalType=
   Where x:
               0 = Utopia Loopback Mode
               1 = Software Loopback Mode
               2 = Remote Loopback Mode
               3 = F4 & F5 cells OAM Ping in UTOPIA Loopback mode
```

```
            4 = F4 & F5 cells OAM Ping in Software Loopback mode
Where y:
            1 = AAL5
            2 = AAL0_48
            3 = AAL0_52
```

Note for VxWorks* and UNIX* Usage:

- IX_ATM_CODELET_SWLOOPBACK_PORT_RATE and IX_ATM_CODELET_REMOTELOOPBACK_PORT_RATE defined in this header file allows the user to change the port rate (in cells/sec) accordingly. The port rate works when using ADSL connection. The default port rate for both loopback is set to 1962cells/sec (~832kbps)
- IX_ATMCODELET_START_VPI and IX_ATMCODELET_START_VCI defined in this header file can be modified by the user. By default, VPI and VCI are set to 1 and 32 respectively
- IX_ATMCODELET_NUM_AAL5_CHANNELS, IX_ATMCODELET_NUM_AAL0_48_CHANNELS, and IX_ATMCODELET_NUM_AAL0_52_CHANNELS can be changed by the user. By default, they are set to 32 channels
- OAM Ping in UTOPIA Loopback mode will perform the following sequence in forever loop: i) Send AAL packets, ii) Display the transmit and receive statistics, and iii) Perform OAM Ping F4 and F5 (ETE and Segment) and display OAM statistics
- OAM Ping in Software Loopback will perform the following sequence in forever loop: i) Display the transmit and receive statistics, and ii) OAM Ping F4 and F5 (ETE and Segement) and display OAM Statistics

ATM Features

- An interface is provided to setup Aal5 or Aal0 (48 or 52 bytes) Transmit and Recieve VCs on one port. Only UBR VCs can be setup. When a channel is setup using this interface both a Transmit and a Recieve VC is created.

- An interface is provided to remove all registered Aal5/Aal0 VCs.

- Both remote and local loopback of Aal5 or Aal0 is provided by this codelet. Local loopback refers to loopback provided by the UTOPIA interface. In local loopback packets generated by the IXP400 are looped back to the Atm driver by the UTOPIA hardware. Software loopback refers to the software looping all packets received on the wire back out onto the wire. Remote loopback refers to where the far end is expected to performa a software loopback, i.e. any packets sent by the codelet are expected to be looped back by the far end into the codelet.

- Both interrupt and polled mode of operation is provided by this codelet.

- An interface is provided to send Aal5 SDUs specifying the sdu size and the number of Aal5 sdus to send.

- An interface is provided to send Aal0 packets specifying the packet size and the number of Aal0 packets to send.

- Both the Transmit port rate and the Recieve port rate can be modified. The Tx port rate is used by IxAtmSch in performing the shaping functions The Rx port rate is not used by any component.

- An interface is provided to allow the querying of the ATM ports and registered VCs.

IXP400 ATM Components used by this codelet

- IxAtmdAcc. This component is the low level interface by which AAL packets get transmitted to, and received from the UTOPIA bus.

IXP400 ATM Codelet components used by this codelet

- IxAtmSch. This component demonstrates an ATM Traffic Shaper implementation. IxAtmdAcc gets scheduling information from this component.

- IxAtmm. This component provides ATM port and VC management facilities. This component also manages the configuration of the UTOPIA co−processor

IxAtmCodelet modes of operation
This codelet can be initialised to operate in one of three configurations.

- IX_ATMCODELET_UTOPIA_LOOPBACK. In this mode the UTOPIA interface will loopback all traffic transmitted.

  Buffer management In this mode a simple buffering mechanism is used; mbufs are allocated from the VxWorks* pool as needed for RxFree replenishing/Tx and are returned to the VxWorks* pool for TxDone/Rx

  Interrupt/Task based processing In this mode of operation the IxQMgrDispatcher is hooked to interupts. This means that TxDone/RxLo will be perfomed from a task level and IxAtmdAcc Tx processing/RxHi and RxFree will be precessed from an interrupt level.

  Sending Aal5 PDUs In this mode of operation an interface is available to send Aal5 PDUs of specified size in bytes.

  Sending Aal0 Packets In this mode of operation an interface is available to send Aal0 packets of specified size in cells.

- IX_ATMCODELET_REMOTE_LOOPBACK. In this mode packets are sent and are expected to be looped back on the remote end.

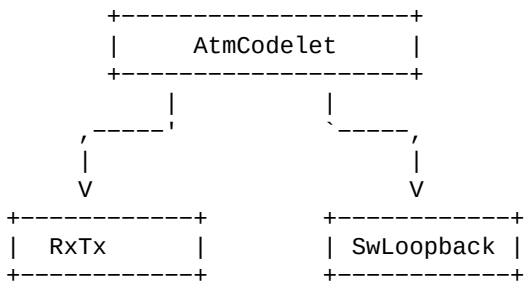
- IX_ATMCODELET_SOFTWARE_LOOPBACK. In this mode of operation all traffic sent are looped back to receive in the IXP400 device.

  Buffer management In this mode a more complex buffering mechanism is used; mbufs are allocated from the VxWorks* pool and stored in a software queue. These mbufs are fetched from this software queue as needed for RxFree replenishing/Tx and are returned to the software queue for TxDone/Rx.

  Interrupt/Task based processing In this mode of operation the IxQMgrDispatcher is called form a task every 1 msec. This means that TxDone/RxLo/IxAtmdAcc Tx processing/RxHi and RxFree will be processed from a task level.

  Sending PDUs/Packets This mode of operation does not provide an interface for sending Aal5 PDUs/Aal0 Packets.

IxAtmCodelet sub−components



AtmCodelet This implements the API functions.

RxTx This sub−component implements the IX_ATMCODELET_UTOPIA_LOOPBACK and IX_ATMCODELET_REMOTE_LOOPBACK modes of operation.

SwLoopback This sub−component implements the IX_ATMCODELET_SOFTWARE_LOOPBACK mode operation.

BufMan This sub−component implements the intefaces used internally for getting and returning VxWorks* mbufs.

UTOPIA Recieve and Transmit PHY addresses In this codelet UTOPIA Phy Addresses are assigned numbers starting at UTOPIA_PHY_ADDR and are incremented for each port used. These addresses _WILL_ need to be changed depending on the hardware setup.

OAM Features

- An interface is provided to configure the access layer to enable OAM traffic.
- OAM Loopback responses are sent automatically on receipt of OAM F4 and F5 parent Segment and ETE loopback cells, i.e. externally initiated OAM Loopbacks.
- Interfaces are provided to initiate an OAM F4/F5 Segment or ETE Loopback, see ITU−610. One loopback can be initiated at a time, this is to keep the codelet simple, this is independant of sending responses to parent loopback cells received.
- An interface is provided to query OAM traffic statistics.

OAM uses the following IXP400 ATM Components

- IxAtmdAcc. This component is the low level interface by which OAM cells get transmitted to, and received from the UTOPIA bus.
- IxAtmm. This component provides ATM port and VC management facilities. This component also contains the configuration of the UTOPIA co−processor.
- AtmUtils. This codelet provides VC setup facilities using IxAtmdAcc.

IMPORTANT!!!
If validation board is used (instead of using IXDP400 development board), then uncomment VALIDATION_PLATFORM_USED flag in component.mk

Define Documentation

#define IX_ATM_CODELET_REMOTELOOPBACK_PORT_RATE

Port rate for Remote Looopback.

By default the port rate for remote loopback is set at PCR = 1962 cells/sec (832kbps)

Definition at line **305** of file **IxAtmCodelet.h**.

#define IX_ATM_CODELET_SWLOOPBACK_PORT_RATE

Port rate for Software Loopback.

By default the port rate for software loopback is set at PCR = 1962 cells/sec (832kbps)

Definition at line **292** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_NUM_AAL0_48_CHANNELS

32 Channels for AAL0 48−bytes

Definition at line **345** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_NUM_AAL0_52_CHANNELS

32 Channels for AAL0 52−bytes

Definition at line **354** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_NUM_AAL5_CHANNELS

32 Channels for AAL5

Definition at line **336** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_QMGR_DISPATCHER_PRIORITY

Recommended priority of queue manager dispatch loop.

Definition at line **364** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_QMGR_DISPATCHER_THREAD_STACK_SIZE

Recommended stack size for queue manager dispatcher thread.

Definition at line **374** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_START_VCI

The first VCI value.

By default the VCI is set to 2

Definition at line **327** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_START_VPI

The first VPI value.

By default the VPI is set to 1

Definition at line **316** of file **IxAtmCodelet.h**.

#define IX_ATMCODELET_UTOPIA_LB_THREAD_PRIORITY

Recommended priority of UTOPIA loopback thread (for AAL0 and AAL5 type cells).

Definition at line **385** of file **IxAtmCodelet.h**.

Function Documentation

ixAtmCodeletMain (**IxAtmCodeletMode** *modeType*,
IxAtmCodeletAalType *aalType*
)

This function is used as a single point of execution for ATM codelet.

Parameters:

IxAtmCodeletMode modeType	The type of mode use, either Utopia, Software or Remote loopback mode. It also consists of OAM Ping in Utopia or Software loopback

	mode.
IxAtmCodeletAalType aalType	The type of AAL: AAL5 or AAL0 with 48− or 52−bytes cell

Returns:

IX_SUCCESS Mode and AAL type successfully setup IX_FAIL Invalid AAL or mode type, or error in setting up the modes

Intel (R) IXP400 Software DMA Access Codelet (IxDmaAccCodelet) API

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software DMA Access component API.

Defines

#define **IX_DMA_CODELET_TRANSFER_LENGTH**

The length of the transfer size if 128 bytes.

Functions

IX_STATUS **ixDmaAccCodeletMain** (void)

This function is the entry point to the Dma Access codelet. It will initialise the Dma codelet which in turn initialises the necessary components.

Detailed Description

Intel (R) IXP400 Software DMA Access component API.

This file contains a main interface of the Dma Access Codelet that initialises the DmaAcc codelet and execute Dma transfer using **ixDmaAccCodeletTestPerform()** function for various DMA transfer mode, addressing mode and transfer width. The block size used in this codelet are 8,1024,16384,32768,65528 bytes. For each Dma configuration, the performance will be measured and the average rate (in Mbps) will be displayed

VxWorks* User Guide

```
   Usage :
      -> ixDmaAccCodeletMain()

 Note:
 1. Once the function is executed, the codelet will display the results


 2. The formulae to calculate the rate is:


    Rate (in Mbps)  = ( (length * 8) / (ticks / 66) )
```

Linux* User Guide

```
  Usage :
     # insmod ixp400_codelets_dmaAcc.o

Note:

1. Once the function is executed, the codelet will display the results

2. The formulae to calculate the rate is:

   Rate (in Mbps)  = ( (length * 8) / (ticks / 66) )
```

DmaAcc Codelet Features

The API **ixDmaAccCodeletTestPerform()** allows the user to perform a Dma transfer of block size 0 to 65535 bytes between two locations in the SRAM. The user can specify any combination of the following modes.

DMA Transfer Modes 1. Copy 2. Copy and Clear Source 3. Copy with Bytes Swap 4. Copy with Bytes Reversed

DMA Addressing Modes 1. Incremental Source to Incremental Destination Addressess 2. Fixed Source to Incremental Destination Addressess 3. Incremental Source to Fixed Destination Addressess

DMA Transfer Widths 1. 32−bit Transfer 2. 16−bit Transfer 3. 8−bit Transfer 4. Burst Transfer

NOTE : The user must initialise the system with ixDmaAccCodeletInit prior to calling the function ixDmaAccCodeletiTestPerform()

Performance will execute PERFORMANCE_NUM_LOOP (i.e. 100 runs) in order to calculate the average rate for each Dma transfer configuration

Define Documentation

```
#define IX_DMA_CODELET_TRANSFER_LENGTH
```

The length of the transfer size if 128 bytes.

It can be changed for Dma transfer. The range is between 1−65535 bytes

Definition at line **113** of file **IxDmaAccCodelet.h**.

Function Documentation

void ixDmaAccCodeletMain (void)

This function is the entry point to the Dma Access codelet. It will initialise the Dma codelet which in turn initialises the necessary components.

Once it has successfully initialise the Dma Codelet, this function will continue to perform valid DMA transfer using IxDmaAccCodeletTestPerform()

Parameters:

none

Returns:

none

Intel (R) IXP400 Software Ethernet Access Codelet (IxEthAccCodelet) API

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software Ethernet Access Codelet API.

Defines

#define **IX_ETHACC_CODELET_NPEA_MAC**
Hard−encoded MAC address for NPEA.

#define **IX_ETHACC_CODELET_NPEB_MAC**
Hard−encoded MAC address for NPEB.

#define **IX_ETHACC_CODELET_NPEC_MAC**
Hard−encoded MAC address for NPEC.

#define **IX_ETHACC_CODELET_RX_MBUF_POOL_SIZE**
Size of receive MBuf pool.

#define **IX_ETHACC_CODELET_TX_MBUF_POOL_SIZE**
Size of transmit MBuf pool.

#define **IX_ETHACC_CODELET_MAX_PORT**
Number of Ethernet Ports supported for this codelet.

#define **IX_ETHACC_CODELET_MBUF_POOL_SIZE**
Size of MBuf pool.

#define **IX_ETHACC_CODELET_PCK_SIZE**
Size of MBuf packet (recommended size for ethAcc component).

#define **IX_ETHACC_CODELET_PCK_LEN**
Length of MBuf payload (in bytes).

#define **IX_ETHACC_CODELET_MBUF_DATA_POOL_SIZE**
Size of MBuf data pool.

#define **IX_ETHACC_CODELET_TXGEN_PCK_LEN**
Size of packets for TxGenRxSink Operation.

#define **IX_ETHACC_CODELET_TXGEN_PCKS**
Number of packets to generate for the TxGenRxSink Operation.

#define **IX_ETHACC_CODELET_RX_FCS_STRIP**
Strip FCS from incoming frames. To undefine, change to #undef.

#define **IX_ETHACC_CODELET_FRAME_SIZE**
Maximum size of a frame.

Enumerations

enum **IxEthAccCodeletOperation** {
IX_ETHACC_CODELET_RX_SINK,
IX_ETHACC_CODELET_SW_LOOPBACK,
IX_ETHACC_CODELET_TXGEN_RXSINK,
IX_ETHACC_CODELET_PHY_LOOPBACK,
IX_ETHACC_CODELET_BRIDGE,
IX_ETHACC_CODELET_BRIDGE_QOS,
IX_ETHACC_CODELET_BRIDGE_FIREWALL,
IX_ETHACC_CODELET_ETH_LEARNING,
IX_ETHACC_CODELET_BRIDGE_WIFI
}

Functions

PUBLIC **ixEthAccCodeletMain** (**IxEthAccCodeletOperation** operationType, **IxEthAccPortId** IX_STATUS inPort, **IxEthAccPortId** outPort, int disableStats)

Detailed Description

Intel (R) IXP400 Software Ethernet Access Codelet API.

This codelet demonstrates both Ethernet Data and Control plane services and Ethernet Management services.

- A) Ethernet Data and Control plane services:
  - ♦ Configuring both ports as a receiver sink from an external source (such as Smartbits).
  - ♦ Configuring Port−1 to automatically transmit frames and receive frames on Port−2. Frames generated and transmitted in Port−1 are loopbacked into Port−2 by using cross cable.
  - ♦ Configuring and performing a software loopback on each of the two ethernet ports.
  - ♦ Configuring both ports to act as a bridge so that frames received on one port are retransmitted on the other.

- B) Ethernet Management services:
  - ♦ Adding and removing static/dynamic entries.
  - ♦ Calling the maintenance interface (shall be run as a separate background task)
  - ♦ Calling the show routine to display the MAC address filtering tables.

Definition
In the context of this codelet, the following definitions are applicable.
Port 1 = ixe0 = Ethernet port associated with NPE−B Ethernet Coprocessor.
Port 2 = ixe1 = Ethernet port associated with NPE−C Ethernet Coprocessor.
Port 3 = ixe2 = Ethernet port associated with NPE−A Ethernet Coprocessor.

Design constraints
This codelet assumes that the underlying Intel(R) IXP4XX Product Line of Network Processors have two Ethernet NPEs. For silicon with single Ethernet NPE, operation will be only functional in the particular Ethernet port that corresponds to the available Ethernet NPE. Particularly, bridge operation will not work as two Ethernet ports are needed in this operation.

Assumptions
This codelet illustrates the use EthAcc APIs. The operations provided may not be working on the best performance as the target of this codelet is just to show the functionality of APIs. In order to get better performance, #undef IX_ETHACC_CODELET_TXGENRXSINK_VERIFY to disable traffic verification.

Please note that this codelet is not optimized for production quality.

For performance testing, please use the operations below:

- Rx Sink Operation.
- TxGenRxSink Operation.
- Bridge Operation with Ethernet frames sent into either one of the Ethernet Ports.

The operations below need special tuning to optimize them. Tuning can be done by either using a lower traffic(frames/second), reducing the value of IX_ETHACC_CODELET_TXGEN_PCKS or #undef IX_ETHACC_CODELET_TXGENRXSINK_VERIFY.

- Software Loopback Operation.
- PHY Loopback Operation.
- Bridge Operation with Ethernet frames sent into both Ethernet Ports.

VxWorks* User Guide
ixEthAccCodeletMain() function is used as a single point of execution for EthAcc Codelet. It allows user to enter selection for different type of supported operations described below:

```
Usage :
   >ixEthAccCodeletMain (operationType,inPort,outPort,disableStats)
   Where operationType:
        1 = To sink received frames as fast as possible for available ports.
        2 = To software loopback received frames to the same port for available ports.
        3 = To generate and transmit frames from outPort, remote loopback by using
            an external cross cable to inPort, and received on inPort (TxGenRxSink).
        4 = To generate frames and perform PHY loopback on the same port for available ports.
        5 = To transmit any frame received on inPort through outPort (Bridge).
        6 = To transmit any 802.1Q-tagged frame received on inPort through outPort,
            using QoS (QoS Bridge)
        7 = To transmit frames received on inPort through outPort, provided
            they meet the MAC address firewall criteria (Firewall Bridge)
        8 = To activate Ethernet MAC learning facility.
        9 = To transmit frames received on inPort through outPort, using
            802.3 <=> 802.11 frame header conversions for frames matching certain
            MAC addresses

   Where inPort and outPort are Ethernet portId:
        0 = NPE-B
        1 = NPE-C
        2 = NPE-A

     Where disableStats:
```

```
        0 = Enable statistic polling task thread from running.
        1 = disable statistic polling task thread.
```

Linux* User Guide

The idea of using the **ixEthAccCodeletMain()** as a single point of execution for EthAcc codelet. The operation selected will be executed when user issue 'insmod' in command prompt.

```
Usage :
   >insmod ixp400_codelets_ethAcc.o operationType= inPort= outPort= disableStats=
   Where x:
        1 = To sink received frames as fast as possible for available ports.
        2 = To software loopback received frames to the same port for available ports.
        3 = To generate and transmit frames from outPort (z), remote loopback by using
            an external cross cable to inPort (y), and received on inPort (y) (TxGenRxSink).
        4 = To generate frames and perform PHY loopback on the same port for available ports.
        5 = To transmit any frame received on inPort (y) through outPort (z) (Bridge).
        6 = To transmit any 802.1Q-tagged frame received on inPort (y) through outPort (z),
            using QoS (QoS Bridge)
        7 = To transmit frames received on inPort (y) through outPort (z), provided
            they meet the MAC address firewall criteria (Firewall Bridge)
        8 = To activate Ethernet MAC learning facility.
        9 = To transmit frames received on inPort (y) through outPort (z), using
            802.3 <=> 802.11 frame header conversions for frames matching certain
            MAC addresses

   Where inPort and outPort are Ethernet portId:
        0 = NPE-B
        1 = NPE-C
        2 = NPE-A

     Where i:
        0 = Enable statistic polling task thread from running.
        1 = Disable statistic polling task thread.
```

WinCE* User Guide

The Ethernet Access Codelet uses serial console to print out menus and accept input from users. Users need to choose and enter which operation to be executed from the menus.

```
Usage :
   Menu: Choose type of test you want to execute.
   Options:
        1 = To sink received frames as fast as possible for available ports.
        2 = To software loopback received frames to the same port for available ports.
        3 = To generate and transmit frames from port 1, remote loopback by using
            an external cross cable to port 2, and received on port 2 (TxGenRxSink).
        4 = To generate frames and perform PHY loopback on the same port for available ports.
        5 = To transmit any frame received on one port through the other one (Bridge).
        6 = To transmit any 802.1Q-tagged frame received on one port through the other
            one, using QoS (QoS Bridge)
        7 = To transmit frames received on one port through the other port, provided
            they meet the MAC address firewall criteria (Firewall Bridge)
        8 = To activate Ethernet MAC learning facility.
        9 = To transmit frames received on one port through the other port, using
```

802.3 <=> 802.11 frame header conversions for frames matching certain MAC addresses
100 = Exit Ethernet Access codelet.

MAC Setup

The default MAC setup will be:

Promiscuous mode enabled (for learning example)

Frame Check Sequence appended for all frames generated on the Intel XScale(R) processor

PHY Setup

This codelet uses two PHYs as defined by IX_ETHACC_CODELET_MAX_PHY
The default PHY setup will be:

100Mbits,

full duplex,

auto−negotiation on.

Jumbo frames

This codelet setup enable Jumbo frame reception
The default setup will be:

frames up to a msdu size of 9018 are supported.

Test Equipment

The test harness will consist of external test equipment capable of generating Ethernet packets (e.g. SmartBits).

The test equipment must be capable of performing at least the following actions to support the scenarios outlined for the Codelet:

Send/receive an Ethernet data stream.

Send/receive frames of different length.

Detect CRC errors.

Append FCS.

Support 100Mbit full duplex mode.

Define Documentation

#define IX_ETHACC_CODELET_FRAME_SIZE

Maximum size of a frame.

This maximum frame size includes different network settings :

- Ethernet frames (up to 1518 bytes),
- BabyJumbo frames (up to nearly 1600 bytes)
- Jumbo frames (9K bytes). Note that different encapsulation types may extend the MTU size of 9000. The NPE firmware compares only the overall ethernet frame size (MSDU), with may be stripped from the FCS at the time of comparison. The frame size is affected by other operations which changes the frame size, like the addition or the removal of VLAN tags.

Definition at line **413** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_MAX_PORT

Number of Ethernet Ports supported for this codelet.

Definition at line **293** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_MBUF_DATA_POOL_SIZE

Size of MBuf data pool.

Definition at line **332** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_MBUF_POOL_SIZE

Size of MBuf pool.

Definition at line **302** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_NPEA_MAC

Hard−encoded MAC address for NPEA.

Definition at line **248** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_NPEB_MAC

Hard−encoded MAC address for NPEB.

Definition at line **257** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_NPEC_MAC

Hard−encoded MAC address for NPEC.

Definition at line **266** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_PCK_LEN

Length of MBuf payload (in bytes).

Definition at line **323** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_PCK_SIZE

Size of MBuf packet (recommended size for ethAcc component).

Definition at line **314** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_RX_FCS_STRIP

Strip FCS from incoming frames. To undefine, change to #undef.

#define IX_ETHACC_CODELET_RX_MBUF_POOL_SIZE

Size of receive MBuf pool.

Definition at line **275** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_TX_MBUF_POOL_SIZE

Size of transmit MBuf pool.

Definition at line **284** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_TXGEN_PCK_LEN

Size of packets for TxGenRxSink Operation.

Definition at line **342** of file **IxEthAccCodelet.h**.

#define IX_ETHACC_CODELET_TXGEN_PCKS

Number of packets to generate for the TxGenRxSink Operation.

Definition at line **351** of file **IxEthAccCodelet.h**.

Enumeration Type Documentation

enum IxEthAccCodeletOperation

Enumeration values:

IX_ETHACC_CODELET_RX_SINK	All frames received (from external device) will be sinked for available ports.
IX_ETHACC_CODELET_SW_LOOPBACK	All frames received are software loopbacked to the same port for available ports.
IX_ETHACC_CODELET_TXGEN_RXSINK	Frames generated and transmitted from port 1, remote loopbacked to port 2 by using cross cable and received on port 2.
IX_ETHACC_CODELET_PHY_LOOPBACK	Frames generated and PHY loopbacked on the same port for available ports.
IX_ETHACC_CODELET_BRIDGE	Frames received on one port will be transmitted through the other port.

IX_ETHACC_CODELET_BRIDGE_QOS	Frames received on one port will be transmitted through the other port, with priority enabled.
IX_ETHACC_CODELET_BRIDGE_FIREWALL	Frames received on one port will be transmitted through the other port if the MAC address match the firewall criteria.
IX_ETHACC_CODELET_ETH_LEARNING	Ethernet Learning Facility where it adds some static and dynamic entries. Dynamic entries are then aged and verified that they no longer appear in the database.
IX_ETHACC_CODELET_BRIDGE_WIFI	Ethernet 802.3 <=> 802.11 header conversion test.

Definition at line **426** of file **IxEthAccCodelet.h**.

Intel (R) IXP400 Software HSS Access Codelet (IxHssAccCodelet) API

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software HSS Access Codelet API. The interface for the HSS Access Codelet.

Defines

#define **IX_HSSACC_CODELET_DURATION_IN_MS**

Functions

PUBLIC **ixHssAccCodeletMain** (IxHssAccCodeletOperation operationType,
IX_STATUS IxHssAccCodeletPortMode portMode, IxHssAccCodeletVerifyMode verifyMode)

Detailed Description

Intel (R) IXP400 Software HSS Access Codelet API. The interface for the HSS Access Codelet.

This module contains the implementation of the HSS Access Codelet.

The following top−level operation is supported:

- Test Packetised and Channelised Services, with the Codelet acting as data source/sink and HSS as loopback. The Codelet will transmit data and verify that data received is the same as that transmitted. Codelet runs for IX_HSS_CODELET_DURATION_IN_MS ms.
- There are four clients of Packetised service per HSS port − 1 client per E1/T1 trunk. Client 0 and 2 are running in Packetised HDLC mode while client 1 and 3 in Packetised RAW mode.

Assumptions
In Channelised service, the codelet transmits traffic continuously. When the codelet runs up to IX_HSS_CODELET_DURATION_IN_MS ms, Tx counter is bigger than Rx counter. This is due to the fact that traffics submitted to NPE (i.e. Tx counter has been increased) are not transmitted out by NPE when HSS service is disabled. These traffics will be dropped and not loopbacked at HSS (Hence, Rx counter not increased).

In Packetised−raw mode service (client 1 and 3), Rx counter will be bigger than Tx counter because in this service, idle packets are received by Intel XScale(R) processor and causes Rx counter to be bigger than Tx counter. As for packetised−HDLC service, idle packets are handled in HDLC coprocessor and not passed to Intel XScale(R) processor (Hence, Rx counter not increased).

Limitations
When executing Packetised service on both HSS ports of 266MHz Intel (R) IXP42X Processor simultaneously, receive traffic verification should fail on client 3 (i.e. Packetised−raw mode) of HSS port 1.

The reason why this issue occured is IXP421 processor does not have enough CPU resources to perform intensive packet verification tasks. However, this does not imply that the same issue will hit customer applications because actual applications will not do any packet verifications in the way that the codelet does. This issue is not seen on a 533MHz Intel (R) IXP42X processor.

VxWorks* User Guide
ixHssAccCodeletMain() function is used as a single point of execution for HssAcc Codelet.

```
Usage :
   >ixHssAccCodeletMain (operationType, portMode, verifyMode)
   Where operationType:
        1 = Packetised Service Only.
        2 = Channelised Service Only.
        3 = Packetised Service and Channelised Services.

   Where portMode:
        1 = HSS Port 0 Only.
        2 = HSS Port 1 Only.
        3 = HSS Port 0 and 1.
          Note :IXP43X supports only HSS Port 0

   Where verifyMode:
        1 = codelet verifies traffic received in hardware loopback mode.
        2 = codelet does not verify traffic received in hardware loopback mode.
```

Linux* User Guide
The idea of using the **ixHssAccCodeletMain()** as a single point of execution for HssAcc codelet. The operation selected will be executed when user issue 'insmod' in command prompt.

```
Usage :
   >insmod ixp400_codelets_hssAcc.o operationType=(a) portMode=(b) verifyMode=(c)
   Where a:
        1 = Packetised Service Only.
        2 = Channelised Service Only.
        3 = Packetised Service and Channelised Services.

   Where b:
        1 = HSS Port 0 Only.
        2 = HSS Port 1 Only.
        3 = HSS Port 0 and 1.
          Note :IXP43X supports only HSS Port 0

   Where c:
        1 = codelet verifies traffic received in hardware loopback mode.
        2 = codelet does not verify traffic received in hardware loopback mode.
```

WinCE* User Guide
The HSS Access Codelet uses serial console to print out menus and accept input from users. Users need to choose and enter which operation to be executed from the menus.

```
Usage :
```

```
Menu 1: Choose type of service you want to execute.
Options:
     1 = Packetised Service Only.
     2 = Channelised Service Only.
     3 = Packetised Service and Channelised Services.
   100 = Exit HSS access codelet.

Menu 2: Choose which port(s) you want to execute.
Where b:
     1 = HSS Port 0 Only.
     2 = HSS Port 1 Only.
     3 = HSS Port 0 and 1.
   100 = Exit HSS access codelet.
       Note :IXP43X supports only HSS Port 0

Menu 3: Choose if codelet should or shouldn't verify traffic.
Where c:
     1 = codelet verifies traffic received in hardware loopback mode.
     2 = codelet does not verify traffic received in hardware loopback mode.
   100 = Exit HSS access codelet.
```

Buffer Management

The packetised service uses mbuf buffers to store data, and chains mbufs together to form large packets. In the transmit direction, mbufs are allocated from a pool on transmit, and returned to the pool on tranmsit done. For receive, mbufs are allocated from a pool when supplying buffers to the free queue, and returned to the pool on receive.

The channelised service operates quite differently. As voice data is very sensitive to latency using mbufs for transferral of the data between Intel XScale(R) processor and NPE is not very appropriate. Instead, circular buffers are used whereby the NPE reads data from a block of SDRAM that the Intel XScale(R) processor writes to, and writes data to a block of SDRAM that Intel XScale(R) processor reads from. On receive, the NPE writes directly into a circular buffer that Intel XScale(R) processor subsequently reads the data from. Each channel has its own circular buffer, and all these buffers are stored contiguously. On transmit, the NPE takes a circular list of pointers from Intel XScale(R) processor and transmits the data referenced by these pointers. Each list of pointers contains a pointer for each channel, and the circular list of pointers contains multiple lists stored contiguously. This is to allow Intel XScale(R) processor to transmit voice samples without having to copy data, as only the pointer to the data blocks needs to be written to SDRAM. The NPE lets Intel XScale(R) processor know, in the form of Tx and Rx offsets, where in the blocks of SDRAM it is currently reading from and writing to. This enables Intel XScale(R) processor to co−ordinate its reading and writing activities to maintain the data flow. The Tx offset lets Intel XScale(R) processor know the list offset into the Tx circular pointer list that the NPE will next use to transmit. The Rx offset lets Intel XScale(R) processor know the byte offset into each channel's Rx circular buffer that the NPE will next receive data into.

Caching

To improve system performance, caching may be enabled for both the channelised and packetised buffers. To allow for this, buffers need to be flushed before transmit, and invalidated after receive. Flushing the buffers before transmit ensures the NPE reads and transmits the correct data. Invalidating the buffers after receive ensures Intel XScale(R) processor reads and processes the correct data. In the case of the Codelet, all data is flushed and invalidated as every byte is being written to on transmit and every byte is verified on receive. In a

real application flushing or invalidating all the data may not be necessary, only the data that the application has written before transmit or will read after receive. Note, regarding the packetised service, the IxHssAcc component itself takes care of flushing and invalidating mbuf headers. The application needs only to concern itself with the mbuf data.

Data Verification Strategy

For both the packetised and channelised service a changing pattern will be transmitted. When the HSS co−processor is performing a loopback the data received is expected to be the same as that transmitted. The data transmitted carries a byte pattern that begins at a known value and is incremented for each byte. An independent byte pattern is transmitted for each channel of the channelised service, and also for each port of the packetised service. When data is received it is expected to match the pattern that was transmitted. For the channelised service the first non−idle byte received is expected to be the beginning of the byte pattern. For the packetised service, RAW mode clients may receive idle data so this is detected and ignored. Only non−idle data is verified.

56Kbps Packetised HDLC feature

This feature is demonstrated in one of the packetised HDLC clients (i.e. client 2). The CAS bit is configured to be in the least signicant bit (LSB) position with bit polarity '1'. Bit inversion is also enabled on this client as well. The data verification strategy remains the same as other packetised clients.

Intel (R) IXP400 Software USB RNDIS Codelet (IxUSBRNDIS) API

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software codelet USB RNDIS API.

Modules

Intel (R) IXP400 Software USB RNDIS End Driver Codelet (IxUSBRNDIS) API

Intel (R) IXP400 Software codelet for RNDIS End API.

Intel (R) IXP400 Software USB RNDIS Vendor Codelet (IxUSBRNDIS)

Intel (R) IXP400 Software codelet for RNDIS Vendor information.

Functions

PUBLIC M_BLK_ID **rndisBuffAlloc** (size_t size)PUBLIC IX_STATUS **ixUSBRNDISSignalEncapsulatedCommand** (void) *Function prototype for signalling encapsulation command.*

PUBLIC IX_STATUS **ixUSBRNDISProcessEncapsulatedCommand** (IX_USB_MBLK *) *Function prototype for processing encapsulation command.*

PUBLIC IX_STATUS **ixUSBRNDISProcessDataPacket** (IX_USB_MBLK *) *Function prototype for processing data packet.*

PUBLIC IX_STATUS **ixUSBRNDISLayerInit** (void *pDrvCtrl) *Function prototype for layer initialization.*

PUBLIC IX_STATUS **ixUSBRNDISInit** (void) *Function prototype for initializing RNDIS.*

PUBLIC void **ixUSBRNDISUnload** (void) *Function prototype for releasing the I/O memory and disconnecting the interrupt.*

PUBLIC IX_USB_MBLK * **ixUSBRNDISCreateMBuf** (UINT8 *buffer, UINT32 len) *Function prototype for creating MBufs.*

PUBLIC IX_STATUS **ixUSBRNDISSendDataPacket** (RNDIS_BUF *packet) *Function prototype for sending data packet. It is the hook for the RNDIS END.*

void **ixUSBRNDISIpHdrDump** (const char *const mData) const char * **ixUSBRNDISIpProtoStrGet**

(const UINT8 ipProto) const char * **ixUSBRNDISEthernetTypeStrGet** (const UINT16 etherType)
void **ixUSBRNDISEthernetHdrDump** (const char *const mData, BOOL *nonIpHdrDetected)

Detailed Description

Intel (R) IXP400 Software codelet USB RNDIS API.

How to use the USB RNDIS codelet:

- build a loadable object and load it into VxWorks*
- start the codelet by typing ixUSBRNDISStart

You should see the "usb" network interface in the output generated by the ifShow command.

Plug the board into the USB port of a Windows 98/ME/2000 machine and selected the driver provided with the codelet when queried for it.

Note:

- the IP and MAC addresses of the END driver (therefore the board side of the link) are defined in ixUSBRNDISEnd.h
- the MAC address of the RNDIS driver (which will be used by Windows as its own MAC address) is defined in ixUSBRNDIS.h

Currently the END MAC address is 00:00:00:00:00:02 and the RNDIS MAC address is 00:00:00:00:00:01. The IP address of the END is 192.168.1.1, therefore you should use a compatible address for the RNDIS controller on the Windows side (such as 192.168.1.2) and set the END IP address as gateway address for the RNDIS network device, or change them to suitable values.

The codelet was tested with Windows 2000 only, and telnet and ftp traffic was passed by routing the PC through the Intel (R) IXP400 Software into a network.

Warning: this codelet is for demonstration purposes only, it should not be considered a fully working application.

Function Documentation

```
PUBLIC IX_USB_MBLK * ixUSBRNDISCreateMBuf  ( UINT8 *  buffer,
                                             UINT32  len
                                           )
```

Function prototype for creating MBufs.

Parameters:

UINT8 *buffer − Pointer to a buffer
UINT32 len − Length of buffer

Returns:

- ◊ IX_USB_MBLK − Successfully create MBUF
- ◊ NULL − Failed to create MBUF

PUBLIC IX_STATUS ixUSBRNDISInit (void)

Function prototype for initializing RNDIS.

Parameters:

None

Returns:

- ◊ IX_SUCCESS − Successfully initalized RNDIS
- ◊ IX_FAIL − Failed to initialize RNDIS

PUBLIC IX_STATUS ixUSBRNDISLayerInit (void * *pDrvCtrl*)

Function prototype for layer initialization.

Parameters:

pDrvCtrl − Pointer to the device to be initialized

Returns:

- ◊ IX_SUCCESS − Successfully initalized
- ◊ IX_FAIL − Failed to initialize

PUBLIC IX_STATUS ixUSBRNDISProcessDataPacket (IX_USB_MBLK *)

Function prototype for processing data packet.

Parameters:

IX_USB_MBLK − data packet to be processed

Returns:

- ◊ IX_SUCCESS − Data packet successfully processed

◊ IX_FAIL − Failed to process data packet

PUBLIC IX_STATUS ixUSBRNDISProcessEncapsulatedCommand (IX_USB_MBLK *)

Function prototype for processing encapsulation command.

Parameters:

IX_USB_MBLK − memory buffer to be encapsulated

Returns:

◊ IX_SUCCESS − Succesfully processing encapsulation command

◊ IX_FAIL − Failed to process encapsulation command for some internal reason

PUBLIC IX_STATUS ixUSBRNDISSendDataPacket (RNDIS_BUF * *packet*)

Function prototype for sending data packet. It is the hook for the RNDIS END.

Parameters:

RNDIS *packet − Pointer to a data packet to be transmitted

Returns:

◊ IX_SUCCESS − Successfully send a data packet

◊ IX_FAIL − Failed to send a data packet

PUBLIC IX_STATUS ixUSBRNDISSignalEncapsulatedCommand (void)

Function prototype for signalling encapsulation command.

Parameters:

None

Returns:

◊ IX_SUCCESS − Succesfully signalling encapsulation command

◊ IX_FAIL − Failed to signal encapsulation command for some internal reason

PUBLIC void ixUSBRNDISUnload (void)

Function prototype for releasing the I/O memory and disconnecting the interrupt.

Parameters:

None

Returns:

None

Intel (R) IXP400 Software USB RNDIS End Driver Codelet (IxUSBRNDIS) API

[Intel (R) IXP400 Software USB RNDIS Codelet (IxUSBRNDIS) API]

Intel (R) IXP400 Software codelet for RNDIS End API.

Defines

#define **RNDIS_END_INET_ADDR**
RNDIS END driver with inet address.

#define **RNDIS_END_MAC_ADDRESS**
RNDIS END driver with MAC address.

Functions

PUBLIC IX_STATUS **ixUSBRNDISStart** (void)
Starts the RNDIS component.

PUBLIC void **rndisInt** (void *pDrvCtrl, int packet)
RNDIS END hook for receiving packets.

Detailed Description

Intel (R) IXP400 Software codelet for RNDIS End API.

Description of the RNDIS End driver

Running the codelet in Linux*.

- insmod lib/linuxbe/ixp400.o
- insmod lib/linuxbe/ixp400_codelets_usb.o
- ifconfig usb0 x.x.x.x (Where x.x.x.x is the desired ip address of the interface)

Running the codelet in VxWorks* big endian.

- Load module that was built
  - ♦ ld < lib/vxbe/codelets_usbTest.out
- Run the codelet
  - ♦ ixUSBRNDISStart
  - ♦ ifAddrSet "usb0", "x.x.x.x" (Where x.x.x.x is the desired ip address of the interface)

Running the codelet in VxWorks* little endian.

- Load module that was built
  - ld < lib/vxle/codelets_usbTest.out
- Run the codelet
  - ixUSBRNDISStart
  - ifAddrSet "usb0", "x.x.x.x" (Where x.x.x.x is the desired ip address of the interface)

The codelet is executed by calling the ixUSBRNDISStart function. No parameters are needed to be passed in. The start function first loads and initializes the device driver. It then goes on to load the device driver into the MUX. The function then goes on to call muxDevStart() passing in the called earlier. muxDevStart start the device that has already been initialized and handles registering the driver's interrupt service routine and anything else necessary to handle receiving and transmitting. The function finally assigns an IP address to the RNDIS interface.

Overview of IxRNDISEnd.c

Description of macros.

In this example driver the macros RNDIS_OUT_SHORT and RNDIS_IN_SHORT are sample macros to read/write data to a mock device. If a device communicates through formatted control blocks in shared memory, the accesses to those control blocks should also be through redefinable macros.

The macros SYS_INT_CONNECT, SYS_INT_DISCONNECT, and SYS_INT_ENABLE allow the driver to be customized for BSPs that use special versions of these routines.

The macro SYS_INT_CONNECT is used to connect the interrupt handler to the appropriate vector. By default it is the routine intConnect().

The macro SYS_INT_DISCONNECT is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro SYS_INT_ENABLE is used to enable the interrupt level for the end device. It is called once during initialization. By default this is the routine sysLanIntEnable(), defined in the module sysLib.o.

The macro SYS_ENET_ADDR_GET is used to get the ethernet address (MAC) for the device. The single argument to this routine is the END_DEVICE pointer. By default this routine copies the ethernet address stored in the global variable sysTemplateEnetAddr into the END_DEVICE structure.

Brief overview of the APIs. ixUSBRNDISStart()

- Initializes and starts the end driver. Called to run the codelet.

END_OBJ* rndisLoad (char* initString, void *unused)

- This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the initString.

rndisParse (END_DEVICE * pDrvCtrl, char * initString)

- Parse the input string. Fill in values in the driver control structure.

rndisMemInit (END_DEVICE * pDrvCtrl)

- Initialize memory for the chip.

rndisStart (END_OBJ * pDrvCtrl)

- Handle controller interrupt.

rndisRecv (END_DEVICE *pDrvCtrl, M_BLK_ID packet)

- Process the next incoming packet.

rndisHandleRcvInt (END_DEVICE *pDrvCtrl, M_BLK_ID packet)

- Task level interrupt service for input packets.

rndisSend (END_DEVICE * pDrvCtrl, M_BLK_ID pMblk)

- The driver send routine.

rndisIoctl (END_DEVICE * pDrvCtrl, int cmd, caddr_t data)

- The driver I/O control routine.

rndisConfig (END_DEVICE *pDrvCtrl)

- Reconfigure the interface under us.

rndisAddrFilterSet (END_DEVICE *pDrvCtrl)

- Set the address filter for multicast addresses.

rndisPollRcv (END_DEVICE * pDrvCtrl, M_BLK_ID pMblk)

- Routine to receive a packet in polled mode.

rndisPollSend (END_DEVICE* pDrvCtrl, M_BLK_ID pMblk)

- Routine to send a packet in polled mode.

rndisMCastAdd (END_DEVICE *pDrvCtrl char* pAddress)

- Add a multicast address for the device.

rndisMCastDel (END_DEVICE *pDrvCtrl, char* pAddress)

- Delete a multicast address for the device.

rndisMCastGet (END_DEVICE *pDrvCtrl, MULTI_TABLE* pTable)

- Get the multicast address list for the device.

rndisStop (END_DEVICE *pDrvCtrl)

- Stop the device.

rndisUnload (END_DEVICE* pDrvCtrl)

- Unload a driver from the system.

rndisPollStart (END_DEVICE * pDrvCtrl)

- Start polled mode operations.

rndisPollStop (END_DEVICE * pDrvCtrl)

- Stop polled mode operations.

rndisReset (END_DEVICE *pDrvCtrl)

- Reset device.

Define Documentation

```
#define RNDIS_END_INET_ADDR
```

RNDIS END driver with inet address.

Definition at line **224** of file **IxUSBRNDISEnd.h**.

```
#define RNDIS_END_MAC_ADDRESS
```

RNDIS END driver with MAC address.

Definition at line **231** of file **IxUSBRNDISEnd.h**.

Function Documentation

```
PUBLIC STATUS ixUSBRNDISStart ( void )
```

Starts the RNDIS component.

Parameters:

None

Returns:

◊ OK − Succesfully start

RNDIS component

◊ ERROR − Failed to start RNDIS component

```
PUBLIC void rndisInt  ( void *  pDrvCtrl,
                        int     packet
                      )
```

RNDIS END hook for receiving packets.

Parameters:

pDrvCtrl − pointer to END device

packet − received packet

Returns:

None

Intel (R) IXP400 Software USB RNDIS Vendor Codelet (IxUSBRNDIS)

[Intel (R) IXP400 Software USB RNDIS Codelet (IxUSBRNDIS) API]

Intel (R) IXP400 Software codelet for RNDIS Vendor information.

Defines

#define **RNDIS_VENDOR_ID**
RNDIS with static Intel Vendor ID.

#define **RNDIS_PRODUCT_ID**
RNDIS with static Product ID.

#define **RNDIS_VENDOR_DESCRIPTION**
RNDIS with vendor description.

#define **RNDIS_MAC_ADDRESS**
RNDIS with MAC address.

Detailed Description

Intel (R) IXP400 Software codelet for RNDIS Vendor information.

Define Documentation

#define RNDIS_MAC_ADDRESS

RNDIS with MAC address.

Definition at line **46** of file **IxUSBRNDISVendor.h**.

#define RNDIS_PRODUCT_ID

RNDIS with static Product ID.

Definition at line **32** of file **IxUSBRNDISVendor.h**.

#define RNDIS_VENDOR_DESCRIPTION

RNDIS with vendor description.

Definition at line **39** of file **IxUSBRNDISVendor.h**.

```
#define RNDIS_VENDOR_ID
```

RNDIS with static Intel Vendor ID.

Definition at line **25** of file **IxUSBRNDISVendor.h**.

Intel (R) IXP400 Software Parity Error Notifier Access Codelet

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software Parity Error Notifier Access Codelet.

Defines

#define **IX_PARITYENACC_CODELET_QWORD_ALIGNED_MASK**
mask value that moves an address to its Q−word aligned location.

#define **IX_PARITYENACC_CODELET_SCAN_SEGMENT_SIZE**
segment size for memory scan. One segment is 256 Kb.

#define **IX_PARITYENACC_CODELET_DISPLAY_INTERVAL**
display interval for memory scan. The progress of scanning will be shown after scanning 32 Mb.

#define **IX_PARITYENACC_CODELET_ECC_TEST_REG_OFFSET**
byte offset for Memory Controller's ECC Test Register from ECC Control Register. ECC Test Register is used to generate bad ECC for ECC testing.

#define **IX_PARITYENACC_CODELET_SINGLE_BIT_ERROR_BIT_0_SYNDROME**
single bit error syndrome value at bit 0.

#define **IX_PARITYENACC_CODELET_MULTI_BIT_ERROR_SYNDROME**
multi bit error syndrome value.

#define **IX_PARITYENACC_CODELET_EXTERNAL_DATA_ABORT**
Fault Status Register's imprecise external data abort value.

#define **IX_PARITYENACC_CODELET_DATA_CACHE_PARITY_ERR**
Fault Status Register's data cache parity error value.

#define **IX_PARITYENACC_CODELET_BIT_MASK_CHECK**(data, mask)
check if the result of bit mask operation on 'data' and 'mask' match the 'mask'.

#define **IX_PARITYENACC_CODELET_REBOOT**()
*map to reboot function. **reboot()** is not fully implemented, and it would cause the board to hang when it is executed. In the mean time, sysToMonitorColdReboot() which reboots the board using the watchdog timer reset functionality, will be used to reboot the board.*

#define **IX_PARITYENACC_CODELET_SDRAM_BASE_ADDR**
SDRAM base address.

#define **IX_PARITYENACC_CODELET_SDRAM_SCAN_START_OFFSET**

memory scan start address offset

Functions

PUBLIC
IX_STATUS **ixParityENAccCodeletMain** (BOOL multiBit, BOOL injectNow)
This is the main function for Parity Error Notifier Access Component (ParityENAcc) Codelet.

PUBLIC void **ixParityENAccCodeletQuit** (void)
ixParityENAccCodeletQuit function terminates SDRAM scan.

Detailed Description

Intel (R) IXP400 Software Parity Error Notifier Access Codelet.

This codelet shows how to integrate Parity Error Notifier to client application. It demonstrates the followings:

```
how to initialize ParityEN.


how to configure ParityEN or modify ParityEN configuration.


how to register callback with ParityEN.


how to register data abort handler with kernel (only for VxWorks*).


how to inject ECC error.


how to spawn a task to initiate SDRAM memory scan.


how to scrub memory to correct single bit ECC error.


how to handle various parity errors reported by ParityENAcc
               e.g. scrub memory to correct single bit ECC error,
                    reboot the board after detecting multi−bit ECC error, etc.


how to determine whether the data abort is due to multi bit ECC
          error initiated when the Intel XScale(R) processor accesses SDRAM.
```

VxWorks* User Guide
(1) **ixParityENAccCodeletMain**
This function is the main function for ParityENAcc codelet. This function will perform the followings:

- initialize ParityENACC
- register callback with ParityENAcc to handle parity error
- configure ParityENAcc
- spawn "shut down" task to reboot the board when reboot is requested
- register callback with VxWorks* kernel to handle data abort
- generate bad ECC on allocated SDRAM memory
- spawn a task to start SDRAM memory scan

```
Usage :
   -> ixParityENAccCodeletMain "multiBit", "injectNow"

      where
      "multiBit" specifies the ECC error type.
              0 - single bit ECC error (DEFAULT)
              1 - multi bit ECC error

      "injectNow" specifies when ECC error will occur.
              0  - do not cause ECC error until the memory is read later (DEFAULT)
              1  - read the memory now to cause ECC error
```

(2) **ixParityENAccCodeletQuit**
This function terminates SDRAM memory scan.

```
Usage :
     -> ixParityENAccCodeletQuit
```

Linux* User Guide
(1) **ixParityENAccCodeletMain**
This function is the main function for ParityENAcc codelet. This function will perform the followings:

- initialize ParityENACC
- register callback with ParityENAcc to handle parity error
- configure ParityENAcc
- spawn "shut down" task to reboot the board when reboot is requested

- generate bad ECC on allocated SDRAM memory
- spawn a task to start SDRAM memory scan

This function will be invoked and executed when the user loads the ParityENACc codelet module using 'insmod' command.

```
Usage :
   prompt> insmod ixp400_codelets_parityENAcc.o multiBit=x injectNow=y

     where x = 0 − single bit ECC error (DEFAULT)
                 1 − multi bit ECC error

     and y = 0 − do not cause ECC error until the memory is read later (DEFAULT)
               1 − read the memory now to cause ECC error
```

(2) **ixParityENAccCodeletQuit**
This function terminates SDRAM memory scan. This function will be executed when user terminates parityENAcc codelet execution using 'rmmod' command.

```
Usage :
     prompt> rmmod ixp400_codelets_parityENAcc
```

Define Documentation

```
#define IX_PARITYENACC_CODELET_BIT_MASK_CHECK ( data,
                                                mask )
```

check if the result of bit mask operation on 'data' and 'mask' match the 'mask'.

Parameters:

data [in] − the data that will be masked and then compared with the mask.

mask [in] − the mask value.

Returns:

BOOL

◊ TRUE − bit mask operation result match the mask

◊ FALSE − bit mask operation result does not match the mask

Definition at line **250** of file **IxParityENAccCodelet.h**.

```
#define IX_PARITYENACC_CODELET_DATA_CACHE_PARITY_ERR
```

Fault Status Register's data cache parity error value.

Definition at line **233** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_DISPLAY_INTERVAL

display interval for memory scan. The progress of scanning will be shown after scanning 32 Mb.

Definition at line **196** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_ECC_TEST_REG_OFFSET

byte offset for Memory Controller's ECC Test Register from ECC Control Register. ECC Test Register is used to generate bad ECC for ECC testing.

Definition at line **205** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_EXTERNAL_DATA_ABORT

Fault Status Register's imprecise external data abort value.

Definition at line **226** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_MULTI_BIT_ERROR_SYNDROME

multi bit error syndrome value.

Definition at line **219** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_QWORD_ALIGNED_MASK

mask value that moves an address to its Q−word aligned location.

Definition at line **179** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_REBOOT ( )

map to reboot function. **reboot()** is not fully implemented, and it would cause the board to hang when it is executed. In the mean time, sysToMonitorColdReboot() which reboots the board using the watchdog timer reset functionality, will be used to reboot the board.

Definition at line **278** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_SCAN_SEGMENT_SIZE

segment size for memory scan. One segment is 256 Kb.

Definition at line **187** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_SDRAM_BASE_ADDR

SDRAM base address.

Definition at line **292** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_SDRAM_SCAN_START_OFFSET

memory scan start address offset

Definition at line **285** of file **IxParityENAccCodelet.h**.

#define IX_PARITYENACC_CODELET_SINGLE_BIT_ERROR_BIT_0_SYNDROME

single bit error syndrome value at bit 0.

Definition at line **212** of file **IxParityENAccCodelet.h**.

Function Documentation

```
ixParityENAccCodeletMain ( BOOL multiBit,
                           BOOL injectNow
                         )
```

This is the main function for Parity Error Notifier Access Component (ParityENAcc) Codelet.

This function will first initialize ParityENAcc. Then, it registers the callback with ParityENAcc to notify the codelet whenever parity error is detected. Next, this function will configure ParityENAcc to enable parity error detection. By default, parity error detection will only enabled on MCU.

Next, this function will spawn "shut down" task. This task will sleep all the time. It will only be invoked when board reboot is requested. This task is mainly used to reboot the board.

In VxWorks*, a data abort handler is hooked up to DATA ABORT exception vector. This data abort handler will be called when data abort occurs. This data abort handler will determine whether the data abort is triggered by multi−bit ECC error. The board will be rebooted after data abort is processed. Linux* does not have the hook up capability.

After that, this function will generate bad ECC on allocated SDRAM memory. The user needs to provide two parameters to specify how the ECC error should be generated. First, the user has to specify the type of ECC error injection − single bit or multi bit. Second, the user has to tell this function when to cause ECC error after bad ECC generation. By default, bad single bit ECC will be injected, and the error will be discovered later when the memory is accessed by the memory scan.

Finally, this function will initiate SDRAM memory scan. Any single bit ECC error found during the scan will be scrubbed and corrected. If multi−bit ECC error is detected during the scan, the address of memory with bad ECC will be printed, then the board will be rebooted.

Parameters:

multiBit BOOL [in] − type of ECC error injection.
- ◊ FALSE : Single bit ECC error.
- ◊ TRUE : Multi−bit ECC error.

injectNow BOOL [in] − preference for when to generate ECC error.
- ◊ FALSE : This function will only generate bad ECC on allocated memory. ECC error will only occur when the memory is read later.
- ◊ TRUE : After generating bad ECC, this function will immediately read the memory to cause ECC error.

Returns:

IX_STATUS
- ◊ IX_SUCCESS − start codelet successfully
- ◊ IX_FAIL − fail to start codelet

ixParityENAccCodeletQuit (void)

ixParityENAccCodeletQuit function terminates SDRAM scan.

Returns:

void

Intel (R) IXP400 Software Time Sync Access Codelet

[Intel (R) IXP400 Software Codelets]

Intel (R) IXP400 Software Time Sync Access Codelet.

Data Structures

struct **IxTimeSyncAccCodeletTSChannelConfig**
This struct is used to store all three Time Sync channels' operating mode: master or slave.

struct **IxTimeSyncAccCodeletTSChannelConfig**
This struct is used to store all three Time Sync channels' operating mode: master or slave.

struct **IxTimeSyncAccCodeletUninitFuncMap**
This struct is used to store each supporting module's unload function's pointer, function parameter, and the state whether the module is initialized.

struct **IxTimeSyncAccCodeletUninitFuncMap**
This struct is used to store each supporting module's unload function's pointer, function parameter, and the state whether the module is initialized.

Defines

#define **IX_TIMESYNCACC_CODELET_ROLLOVER_VALUE**
rollover value

#define **IX_TIMESYNCACC_CODELET_APB_CLOCK_FREQUENCY**
APB clock frequency in MHz (66 MHz).

#define **IX_TIMESYNCACC_CODELET_FSV_DEFAULT**
define default value for frequency scale value or tick rate. With this default tick rate, system time would tick approximately every milli−second.

#define **IX_TIMESYNCACC_CODELET_TARGET_TIME_HIT_INTERVAL**
define default value for target time interval. With default tick rate, default interval is approximately 1 second.

#define **IX_TIMESYNCACC_CODELET_MAX_TS_CHANNELS**
maximum number of time sync channels

#define **IX_TIMESYNCACC_CODELET_MAX_CONFIGURATIONS**
maximum number of supported configurations

#define **IX_TIMESYNCACC_CODELET_PTP_MESSAGE_LEN**

PTP message length in byte.

#define **IX_TIMESYNCACC_CODELET_UDP_PAYLOAD_LEN**
UDP payload size in byte (offset: byte 38 and 39).

#define **IX_TIMESYNCACC_CODELET_UDP_HEADER_LEN**
UDP header size in byte.

#define **IX_TIMESYNCACC_CODELET_UDP_CHECKSUM_LEN**
UDP checksum length in byte.

#define **IX_TIMESYNCACC_CODELET_UDP_FRAME_LEN**
UDP frame size in byte.

#define **IX_TIMESYNCACC_CODELET_IP_DATAGRAM**
frame type field of UDP header (offset: byte 12 and 13) for PTP message. It is set to IP_DATAGRAM.

#define **IX_TIMESYNCACC_CODELET_IP_HEADER_LEN**
IP_HEADER_LEN field of UDP header (offset: byte 14) for PTP message.

#define **IX_TIMESYNCACC_CODELET_IP_DATAGRAM_LEN**
IP_DATAGRAM_LEN field of UDP header (offset: byte 16 and 17) for PTP message.

#define **IX_TIMESYNCACC_CODELET_TIME_TO_LIVE**
TIME_TO_LIVE field of UDP header (offset: byte 22) for PTP message.

#define **IX_TIMESYNCACC_CODELET_UDP_PROTOCOL**
UDP PROTOCOL field of UDP header (offset: byte 23) for PTP message.

#define **IX_TIMESYNCACC_CODELET_PTP_EVENT_PORT**
Destination port number field of UDP header (offset: byte 36 and 37) for PTP message. Event port (319) is used to communicates Sync and Delay_Req messages.

#define **IX_TIMESYNCACC_CODELET_PTP_MESSAGE_TYPE**
PTP_MESSAGE_TYPE field for PTP message (offset: byte 62).

#define **IX_TIMESYNCACC_CODELET_INVALID_PARAM**
value for invalid parameter

#define **IX_TIMESYNCACC_CODELET_PTP_MSG_XMIT_INTERVAL**
time interval for PTP message transmission (2 seconds)

#define **IX_TIMESYNCACC_CODELET_LSB_VALUE**(x)
Get LSB value of x.

#define **IX_TIMESYNCACC_CODELET_MSB_VALUE**(x)
Get MSB value of x (where x is a USHORT type).

Typedefs

typedef void(* **IxTimeSyncAccCodeletUninitFuncPtr**)(**IxNpeDlNpeId**)

Definition of void function pointer with one input parameter. The data type of input parameter is IxNpeDlNpeId.

Enumerations

enum **IxTimeSyncAccCodeletModuleId** {
IX_TIMESYNCACC_CODELET_MBUF_ALLOC,
IX_TIMESYNCACC_CODELET_Q_MGR,
IX_TIMESYNCACC_CODELET_DISPATCHER,
IX_TIMESYNCACC_CODELET_NPE_MH,
IX_TIMESYNCACC_CODELET_NPE_DL,
IX_TIMESYNCACC_CODELET_NPE_A,
IX_TIMESYNCACC_CODELET_NPE_B,
IX_TIMESYNCACC_CODELET_NPE_C,
IX_TIMESYNCACC_CODELET_ETH_ACC,
IX_TIMESYNCACC_CODELET_ETH_PORTS,
IX_TIMESYNCACC_CODELET_TX_PTP
}

Module ID list. These modules are required to be initialized or setup for PTP message transmission from each NPE.

Functions

PUBLIC
IX_STATUS **ixTimeSyncAccCodeletMain** (UINT32 configIndex)

ixTimeSyncAccCodeletMain is the main function for timeSyncAcc codelet.

PUBLIC void **ixTimeSyncAccCodeletUninit** (void)

This function will unload all initialized modules, free all resources, and nicely terminates timeSyncAcc codelet execution.

Detailed Description

Intel (R) IXP400 Software Time Sync Access Codelet.

```
 This codelet shows how to use some of Time Sync Access (timeSyncAcc)
 API functions. It demonstrates the followings:


how to configure Time Sync channel to operate in master or slave mode


how to set frequency scale value (fsv)
```

how to set and get system time

how to set target time

how to setup target time in interrupt mode

how to enable and disable target time interrupt

Basically, Time Sync Access codelet supports three configurations as follow:

```
configuration 0: NPE A − Slave,  NPE B − Slave,  NPE C − Master (default)
configuration 1: NPE A − Slave,  NPE B − Master, NPE C − Slave
configuration 2: NPE A − Master, NPE B − Slave,  NPE C − Slave
```

User provides his/her choice of configuration when executing Time Sync Access codelet. Based on the selected user configuration, each NPE will be configured to operate in the desired operating mode − master or slave. In addition, all NPEs and ethernet ports will be setup to transmit one PTP message using UDP protocol every 2 seconds. PTP Sync message will be transmitted at port where NPE is configured to operate in master mode. Conversely, PTP Delay_Req message will be transmitted from slave mode operating port. The transmission and reception of PTP Sync or Delay_Req messages can be demonstrated if the user connects master operating port to slave operating port. The following table summarizes the activities between the two connecting master port and slave port:

master port	slave port
transmit Sync message	receive Sync message
receive Delay_Req message	transmit Delay_Req message

Time Sync Access codelet sets tick rate to 1000 ticks per second. In other word, the system time will tick every one millisecond. For demonstration purpose, the target time is configured to hit system time every one second. For example, target time is set to one second ahead of current system time. When system time reaches target time, interrupt will occur. Corresponding ISR will be invoked to be processed. Two main things will be performed during ISR. First, the codelet will check if any NPE channel has detected PTP's Sync or Delay_Req message. If yes, the captured system time at RECV/XMIT snapshot registers will be read and printed along with the PTP message information. Then, the codelet will set new target time to a second later. This process will be repeated until the user terminates the Time Sync Access codelet execution.

VxWorks* User Guide

(1) **ixTimeSyncAccCodeletMain**

This function is the main function for timeSyncAcc codelet.
This function will perform the followings:

configure all Time Sync Channels to operate in the mode
specified in the user selected configuration.

set tick rate to get system time to start ticking

setup target time to hit every one second (in interrupt mode)

setup and enable all NPEs and all ethernet components.

spawn a thread to transmit Sync message from master port
and Delay_Req message from slave port every 2 seconds.

Usage :

(a) connect slave port to any master port

(b) run timeSyncAcc codelet:

-> ixTimeSyncAccCodeletMain x

where x =
0 -> NPE A - Slave, NPE B - Slave, NPE C - Master (default)
1 -> NPE A - Slave, NPE B - Master, NPE C - Slave
2 -> NPE A - Master, NPE B - Slave, NPE C - Slave

(2) **ixTimeSyncAccCodeletUninit**

This function will unload all initialized modules, free all resources,
and nicely terminates timeSyncAcc codelet execution.

Usage :
-> ixTimeSyncAccCodeletUninit

Linux* User Guide

(1) **ixTimeSyncAccCodeletMain**

This function is the main function for timeSyncAcc codelet.
This function will perform the followings:

configure all Time Sync Channels to operate in the mode
specified in the user selected configuration.

set tick rate to get system time to start ticking

```
setup target time to hit every one second (in interrupt mode)


setup and enable all NPEs and all ethernet components.


spawn a thread to transmit Sync message from master port
          and Delay_Req message from slave port every 2 seconds.

 This function will be invoked and executed when the user loads the
 timeSyncAcc codelet module using 'insmod' command.

 Usage :

        (a) connect slave port to any master port

        (b) run timeSyncAcc codelet:

            prompt> insmod ixp400_codelets_timeSyncAcc.o config=x

            where x =
                0 -> NPE A - Slave,  NPE B - Slave,  NPE C - Master (default)
                1 -> NPE A - Slave,  NPE B - Master, NPE C - Slave
                2 -> NPE A - Master, NPE B - Slave,  NPE C - Slave


 (2)  ixTimeSyncAccCodeletUninit


 This function will unload all initialized modules, free all resources,
 and nicely terminates timeSyncAcc codelet execution.
 It will be invoked when 'rmmod' command is executed.

 Usage :
        prompt> rmmod ixp400_codelets_timeSyncAcc
```

Define Documentation

#define IX_TIMESYNCACC_CODELET_APB_CLOCK_FREQUENCY

APB clock frequency in MHz (66 MHz).

Definition at line **201** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_FSV_DEFAULT

define default value for frequency scale value or tick rate. With this default tick rate, system time would tick approximately every milli−second.

Definition at line **210** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_INVALID_PARAM

value for invalid parameter

Definition at line **326** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_IP_DATAGRAM

frame type field of UDP header (offset: byte 12 and 13) for PTP message. It is set to IP_DATAGRAM.

Definition at line **276** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_IP_DATAGRAM_LEN

IP_DATAGRAM_LEN field of UDP header (offset: byte 16 and 17) for PTP message.

Definition at line **290** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_IP_HEADER_LEN

IP_HEADER_LEN field of UDP header (offset: byte 14) for PTP message.

Definition at line **283** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_LSB_VALUE (x)

Get LSB value of x.

Definition at line **340** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_MAX_CONFIGURATIONS

maximum number of supported configurations

Definition at line **232** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_MAX_TS_CHANNELS

maximum number of time sync channels

Definition at line **225** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_MSB_VALUE (x)

Get MSB value of x (where x is a USHORT type).

Definition at line **347** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_PTP_EVENT_PORT

Destination port number field of UDP header (offset: byte 36 and 37) for PTP message. Event port (319) is used to communicates Sync and Delay_Req messages.

Definition at line **312** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_PTP_MESSAGE_LEN

PTP message length in byte.

Definition at line **239** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_PTP_MESSAGE_TYPE

PTP_MESSAGE_TYPE field for PTP message (offset: byte 62).

Definition at line **319** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_PTP_MSG_XMIT_INTERVAL

time interval for PTP message transmission (2 seconds)

Definition at line **333** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_ROLLOVER_VALUE

rollover value

Definition at line **194** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_TARGET_TIME_HIT_INTERVAL

define default value for target time interval. With default tick rate, default interval is approximately 1 second.

Definition at line **218** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_TIME_TO_LIVE

TIME_TO_LIVE field of UDP header (offset: byte 22) for PTP message.

Definition at line **297** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_UDP_CHECKSUM_LEN

UDP checksum length in byte.

Definition at line **260** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_UDP_FRAME_LEN

UDP frame size in byte.

Definition at line **267** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_UDP_HEADER_LEN

UDP header size in byte.

Definition at line **253** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_UDP_PAYLOAD_LEN

UDP payload size in byte (offset: byte 38 and 39).

Definition at line **246** of file **IxTimeSyncAccCodelet.h**.

#define IX_TIMESYNCACC_CODELET_UDP_PROTOCOL

UDP PROTOCOL field of UDP header (offset: byte 23) for PTP message.

Definition at line **304** of file **IxTimeSyncAccCodelet.h**.

Typedef Documentation

IxTimeSyncAccCodeletUninitFuncPtr

Definition of void function pointer with one input parameter. The data type of input parameter is IxNpeDlNpeId.

Definition at line **360** of file **IxTimeSyncAccCodelet.h**.

Enumeration Type Documentation

enum IxTimeSyncAccCodeletModuleId

Module ID list. These modules are required to be initialized or setup for PTP message transmission from each NPE.

Enumeration values:

IX_TIMESYNCACC_CODELET_MBUF_ALLOC	mBuf Memory Allocation
IX_TIMESYNCACC_CODELET_Q_MGR	Q Mgr
IX_TIMESYNCACC_CODELET_DISPATCHER	Q Mgr Dispatcher
IX_TIMESYNCACC_CODELET_NPE_MH	NPE Message Handler
IX_TIMESYNCACC_CODELET_NPE_DL	NPE Downloader
IX_TIMESYNCACC_CODELET_NPE_A	NPE A
IX_TIMESYNCACC_CODELET_NPE_B	NPE B
IX_TIMESYNCACC_CODELET_NPE_C	NPE C
IX_TIMESYNCACC_CODELET_ETH_ACC	Ethernet Access Component
IX_TIMESYNCACC_CODELET_ETH_PORTS	Ethernet Port
IX_TIMESYNCACC_CODELET_TX_PTP	PTP Message Transmission Setup

Definition at line **398** of file **IxTimeSyncAccCodelet.h**.

Function Documentation

ixTimeSyncAccCodeletMain (UINT32 *configIndex*)

ixTimeSyncAccCodeletMain is the main function for timeSyncAcc codelet.

This function will perform the followings:

- configure all Time Sync Channels to operate in the mode specified in the user selected configuration.
- set tick rate to get system time to start ticking

- setup target time to hit every one second (in interrupt mode)
- setup and enable all NPEs and all ethernet components.
- spawn a thread to transmit Sync message from master port and Delay_Req message from slave port every 2 seconds.

Parameters:

configIndex UINT32 [in] − choice of configuration
- ◊ 0 −> NPE A − Slave, NPE B − Slave, NPE C − Master (default)
- ◊ 1 −> NPE A − Slave, NPE B − Master, NPE C − Slave
- ◊ 2 −> NPE A − Master, NPE B − Slave, NPE C − Slave

Returns:

IX_STATUS
- ◊ IX_SUCCESS − start codelet successfully
- ◊ IX_FAIL − fail

ixTimeSyncAccCodeletUninit (void)

This function will unload all initialized modules, free all resources, and nicely terminates timeSyncAcc codelet execution.

Returns:

void

ChannelisedStats Struct Reference

ingroup IxHssAccCodeletCom

Data Fields

UINT32 **txSamples**
UINT32 **txBytes**
UINT32 **rxSamples**
UINT32 **rxBytes**
UINT32 **rxIdles**
UINT32 **rxVerifyFails**
UINT32 **connectFails**
UINT32 **portEnableFails**
UINT32 **portDisableFails**
UINT32 **disconnectFails**

Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for channelised statistics

Definition at line **93** of file **IxHssAccCodeletCom.h**.

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

GeneralStats Struct Reference

ingroup IxHssAccCodeletCom

Data Fields

UINT32 **portInitFails**
UINT32 **errorRetrievalFails**
UINT32 **txFrmSyncErrors**
UINT32 **rxFrmSyncErrors**
UINT32 **txOverRunErrors**
UINT32 **rxOverRunErrors**
UINT32 **chanSwTxErrors**
UINT32 **chanSwRxErrors**
UINT32 **pktSwTxErrors**
UINT32 **pktSwRxErrors**
UINT32 **unrecognisedErrors**

Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for general statistics

Definition at line **69** of file **IxHssAccCodeletCom.h**.

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

IX_OSAL_ATTRIBUTE_PACKED Struct Reference

[Intel (R) IXP400 Software Ethernet Database (IxEthDB) API]

The IEEE 802.3 Ethernet MAC address structure.

Data Fields

UINT8 **macAddress** [IX_IEEE803_MAC_ADDRESS_SIZE]

Detailed Description

The IEEE 802.3 Ethernet MAC address structure.

The data should be packed with bytes xx:xx:xx:xx:xx:xx

Note:

The data must be packed in network byte order.

Definition at line **236** of file **IxEthDB.h**.

The documentation for this struct was generated from the following file:

- **IxEthDB.h**

IxAtmCodeletStats Struct Reference

Codelet statistics.

Data Fields

UINT32 **txPdus**
UINT32 **txBytes**
UINT32 **rxPdus**
UINT32 **rxBytes**
UINT32 **txDonePdus**
UINT32 **rxFreeBuffers**
UINT32 **txPdusSubmitFail**
UINT32 **txPdusSubmitBusy**
UINT32 **rxPdusInvalid**

Detailed Description

Codelet statistics.

Definition at line **118** of file **IxAtmCodelet_p.h**.

The documentation for this struct was generated from the following file:

- **IxAtmCodelet_p.h**

IxAtmdAccUtopiaConfig Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia configuration.

Data Fields

IxAtmdAccUtopiaConfig::UtTxConfig_ **utTxConfig**
Tx config Utopia register.

IxAtmdAccUtopiaConfig::UtTxStatsConfig_ **utTxStatsConfig**
Tx stats config Utopia register.

IxAtmdAccUtopiaConfig::UtTxDefineIdle_ **utTxDefineIdle**
Tx idle cell config Utopia register.

IxAtmdAccUtopiaConfig::UtTxEnableFields_ **utTxEnableFields**
Tx enable Utopia register.

IxAtmdAccUtopiaConfig::UtTxTransTable0_ **utTxTransTable0**
Tx translation table.

IxAtmdAccUtopiaConfig::UtTxTransTable1_ **utTxTransTable1**
Tx translation table.

IxAtmdAccUtopiaConfig::UtTxTransTable2_ **utTxTransTable2**
Tx translation table.

IxAtmdAccUtopiaConfig::UtTxTransTable3_ **utTxTransTable3**
Tx translation table.

IxAtmdAccUtopiaConfig::UtTxTransTable4_ **utTxTransTable4**
Tx translation table.

IxAtmdAccUtopiaConfig::UtTxTransTable5_ **utTxTransTable5**
Tx translation table.

IxAtmdAccUtopiaConfig::UtRxConfig_ **utRxConfig**
Rx config Utopia register.

IxAtmdAccUtopiaConfig::UtRxStatsConfig_ **utRxStatsConfig**
Rx stats config Utopia register.

IxAtmdAccUtopiaConfig::UtRxDefineIdle_ **utRxDefineIdle**
Rx idle cell config Utopia register.

IxAtmdAccUtopiaConfig::UtRxEnableFields_ utRxEnableFields
Rx enable Utopia register.

IxAtmdAccUtopiaConfig::UtRxTransTable0_ utRxTransTable0
Rx translation table.

IxAtmdAccUtopiaConfig::UtRxTransTable1_ utRxTransTable1
Rx translation table.

IxAtmdAccUtopiaConfig::UtRxTransTable2_ utRxTransTable2
Rx translation table.

IxAtmdAccUtopiaConfig::UtRxTransTable3_ utRxTransTable3
Rx translation table.

IxAtmdAccUtopiaConfig::UtRxTransTable4_ utRxTransTable4
Rx translation table.

IxAtmdAccUtopiaConfig::UtRxTransTable5_ utRxTransTable5
Rx translation table.

IxAtmdAccUtopiaConfig::UtSysConfig_ utSysConfig
NPE debug config.

Detailed Description

Utopia configuration.

This structure is used to set the Utopia parameters

- contains the values of Utopia registers, to be set during initialisation
- contains debug commands for NPE, to be used during development steps

Note:
− the exact description of all parameters is done in the Utopia reference documents.

Definition at line **979** of file **IxAtmdAccCtrl.h**.

Field Documentation

struct **IxAtmdAccUtopiaConfig::UtRxConfig_** IxAtmdAccUtopiaConfig::utRxConfig

Rx config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtRxDefineIdle_** IxAtmdAccUtopiaConfig::utRxDefineIdle

Rx idle cell config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtRxEnableFields_** IxAtmdAccUtopiaConfig::utRxEnableFields

Rx enable Utopia register.

struct **IxAtmdAccUtopiaConfig::UtRxStatsConfig_** IxAtmdAccUtopiaConfig::utRxStatsConfig

Rx stats config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable0_** IxAtmdAccUtopiaConfig::utRxTransTable0

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable1_** IxAtmdAccUtopiaConfig::utRxTransTable1

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable2_** IxAtmdAccUtopiaConfig::utRxTransTable2

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable3_** IxAtmdAccUtopiaConfig::utRxTransTable3

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable4_** IxAtmdAccUtopiaConfig::utRxTransTable4

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtRxTransTable5_** IxAtmdAccUtopiaConfig::utRxTransTable5

Rx translation table.

struct **IxAtmdAccUtopiaConfig::UtSysConfig_** IxAtmdAccUtopiaConfig::utSysConfig

NPE debug config.

struct **IxAtmdAccUtopiaConfig::UtTxConfig_** IxAtmdAccUtopiaConfig::utTxConfig

Tx config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtTxDefineIdle_** IxAtmdAccUtopiaConfig::utTxDefineIdle

Tx idle cell config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtTxEnableFields_** IxAtmdAccUtopiaConfig::utTxEnableFields

Tx enable Utopia register.

struct **IxAtmdAccUtopiaConfig::UtTxStatsConfig_** IxAtmdAccUtopiaConfig::utTxStatsConfig

Tx stats config Utopia register.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable0_** IxAtmdAccUtopiaConfig::utTxTransTable0

Tx translation table.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable1_** IxAtmdAccUtopiaConfig::utTxTransTable1

Tx translation table.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable2_** IxAtmdAccUtopiaConfig::utTxTransTable2

Tx translation table.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable3_** IxAtmdAccUtopiaConfig::utTxTransTable3

Tx translation table.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable4_** IxAtmdAccUtopiaConfig::utTxTransTable4

Tx translation table.

struct **IxAtmdAccUtopiaConfig::UtTxTransTable5_** IxAtmdAccUtopiaConfig::utTxTransTable5

Tx translation table.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxConfig_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx config Register.

Data Fields

unsigned
int **rxInterface**:1
[31] Utopia Receive Interface

unsigned
int **rxMode**:1
[30] Utopia Receive Mode

unsigned
int **rxOctet**:1
[29] Utopia Receive cell transfer protocol

unsigned
int **rxParity**:1
[28] Utopia Receive Parity Checking enable

unsigned
int **rxEvenParity**:1
[27] Utopia Receive Parity Mode

- *1 − Check for Even Parity*
- *0 − Check for Odd Parity.*

unsigned
int **rxHEC**:1
[26] RxHEC Header Error Check Mode

unsigned
int **rxCOSET**:1
[25] If enabled the HEC is Exclusive−ORÆed with the value 0x55 before being tested with the received HEC

unsigned
int **rxHECpass**:1
[24] Specifies if the incoming cell HEC byte should be transferred after optional processing to the

NPE2 Coprocessor Bus Interface or if it should be discarded

unsigned
int **reserved_1**:1
[23] These bits are always 0

unsigned
int **rxCellSize**:7
[22:16] Receive cell size

unsigned
int **rxHashEnbGFC**:1
[15] Specifies if the VPI field [11:8]/GFC field should be included in the Hash data input or if the bits should be padded with 1Æb0

unsigned
int **rxPreHash**:1
[14] Enable Pre−hash value generation

unsigned
int **reserved_2**:1
[13] These bits are always 0

unsigned
int **rxAddrRange**:5
[12:8] In ATM master, MPHY mode, this register specifies the upper limit of the PHY polling logical range

unsigned
int **reserved_3**:3
[7−5] These bits are always 0 .

unsigned
int **rxPHYAddr**:5
[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Detailed Description

Utopia Rx config Register.

Definition at line **1316** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::reserved_1

[23] These bits are always 0

Definition at line **1359** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::reserved_2

[13] These bits are always 0

Definition at line **1376** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::reserved_3

[7−5] These bits are always 0 .

Definition at line **1383** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxAddrRange

[12:8] In ATM master, MPHY mode, this register specifies the upper limit of the PHY polling logical range

The number of active PHYs are RxAddrRange + 1.

Definition at line **1378** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxCellSize

[22:16] Receive cell size

Configures the receive cell size. Values between 52−64 are valid

Definition at line **1361** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxCOSET

[25] If enabled the HEC is Exclusive−ORÆed with the value 0x55 before being tested with the received HEC

- 1 − Enable HEC ExOR with value 0x55.
- 0 − Use generated HEC value.

Definition at line **1348** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxEvenParity

[27] Utopia Receive Parity Mode

- 1 − Check for Even Parity
- 0 − Check for Odd Parity.

Definition at line **1339** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHashEnbGFC

[15] Specifies if the VPI field [11:8]/GFC field should be included in the Hash data input or if the bits should be padded with 1Æb0

- 1 − VPI [11:8]/GFC field valid and used in Hash residue calculation.
- 0 − VPI [11:8]/GFC field padded with 1Æb0

Definition at line **1364** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHEC

[26] RxHEC Header Error Check Mode

Enables/disables cell header error checking on the received cell header.

- 1 − HEC checking enabled
- 0 − HEC checking disabled

Definition at line **1343** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHECpass

[24] Specifies if the incoming cell HEC byte should be transferred after optional processing to the NPE2 Coprocessor Bus Interface or if it should be discarded

- 1 − HEC maintained 53−byte/UDC cell sent to NPE2.
- 0 − HEC discarded 52−byte/UDC cell sent to NPE2 coprocessor.

Definition at line **1353** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxInterface

[31] Utopia Receive Interface

The following encoding is used to set the Utopia Receive interface as ATM master or PHY slave:

- 1 − PHY

- 0 − ATM

Definition at line **1319** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxMode

[30] Utopia Receive Mode

The following encoding is used to set the Utopia Receive mode to SPHY or MPHY:

- 1 − SPHY
- 0 − MPHY

Definition at line **1324** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxOctet

[29] Utopia Receive cell transfer protocol

Used to set the Utopia cell transfer protocol to Octet−level handshaking. Note this is only applicable in SPHY mode.

- 1 − Octet−handshaking enabled
- 0 − Cell−handshaking enabled

Definition at line **1329** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxParity

[28] Utopia Receive Parity Checking enable

- 1 − Parity checking enabled
- 0 − Parity checking disabled

Definition at line **1335** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxPHYAddr

[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Definition at line **1384** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxPreHash

[14] Enable Pre−hash value generation

Specifies if the incoming cell data should be pre−hashed to allow VPI/VCI header look−up in a hash table.

- 1 − Pre−hashing enabled
- 0 − Pre−hashing disabled

Definition at line **1370** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxDefineIdle_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx idle cells config Register.

Data Fields

unsigned int **vpi**:12
[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- *Note: if VCIdleRxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16
[19:4] ATM VCI [15:0]

unsigned int **pti**:3
[3:1] ATM PTI PTI [2:0]

- *Note: if VCIdleRxPTI is set to 0 the PTI field is ignored in test.*

unsigned int **clp**:1
[0] ATM CLP [0]

- *Note: if VCIdleRxCLP is set to 0 the CLP field is ignored in test.*

Detailed Description

Utopia Rx idle cells config Register.

Definition at line **1418** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle_::clp

[0] ATM CLP [0]

- Note: if VCIdleRxCLP is set to 0 the CLP field is ignored in test.

Definition at line **1429** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle_::pti

[3:1] ATM PTI PTI [2:0]

- Note: if VCIdleRxPTI is set to 0 the PTI field is ignored in test.

Definition at line **1426** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle_::vci

[19:4] ATM VCI [15:0]

Definition at line **1424** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCIdleRxGFC is set to 0 the GFC field is ignored in test

Definition at line **1421** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxEnableFields_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx enable Register.

Data Fields

unsigned
int **defineRxIdleGFC**:1
[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

unsigned
int **defineRxIdlePTI**:1
[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

unsigned
int **defineRxIdleCLP**:1
[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

unsigned
int **phyStatsRxEnb**:1
[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in RxStatsConfig register

unsigned
int **vcStatsRxEnb**:1
[27] This register is used to enable or disable ATM statistics gathering based on a specific VPI/VCI address

unsigned
int **vcStatsRxGFC**:1
[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned
int **vcStatsRxPTI**:1
[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned
int **vcStatsRxCLP**:1

[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned
int **discardHecErr**:1
[23] Discard cells with an invalid HEC

unsigned
int **discardParErr**:1
[22] Discard cells containing parity errors

unsigned
int **discardIdle**:1
[21] Discard Idle Cells based on DefineIdle register values

- *1 − Discard IDLE cells*
- *0 − IDLE cells passed*

unsigned
int **enbHecErrCnt**:1
[20] Enable Receive HEC Error Count

unsigned
int **enbParErrCnt**:1
[19] Enable Parity Error Count

- *1 − Enable count of received cells containing Parity errors*
- *0 − No count is maintained*

unsigned
int **enbIdleCellCnt**:1
[18] Enable Receive Idle Cell Count

unsigned
int **enbSizeErrCnt**:1
[17] Enable Receive Size Error Count

unsigned
int **enbRxCellCnt**:1
[16] Enable Receive Valid Cell Count of non−idle/non−error cells

unsigned
int **reserved_1**:3
[15:13] These bits are always 0

unsigned
int **rxCellOvrInt**:1

[12] Enable CBI Utopia Receive Status Condition if the RxCellCount register overflows

unsigned
int **invalidHecOvrInt**:1
[11] Enable CBI Receive Status Condition if the InvalidHecCount register overflows

unsigned
int **invalidParOvrInt**:1
[10] Enable CBI Receive Status Condition if the InvalidParCount register overflows

- *1 − CBI Receive Condition asserted*

unsigned
int **invalidSizeOvrInt**:1
[9] Enable CBI Receive Status Condition if the InvalidSizeCount register overflows

unsigned
int **rxIdleOvrInt**:1
[8] Enable CBI Receive Status Condition if the RxIdleCount overflows

unsigned
int **reserved_2**:3
[7:5] These bits are always 0

unsigned
int **rxAddrMask**:5
[4:0] This register is used as a mask to allow the user to increase the PHY receive address range

---

Detailed Description

Utopia Rx enable Register.

Definition at line **1438** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::defineRxIdleCLP

[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

- 1 − CLP field is valid.
- 0 − CLP field ignored.

Definition at line **1451** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::defineRxIdleGFC

[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

- 1 − GFC field is valid.
- 0 − GFC field ignored.

Definition at line **1441** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::defineRxIdlePTI

[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

- 1 − PTI field is valid.
- 0 − PTI field ignored.

Definition at line **1446** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardHecErr

[23] Discard cells with an invalid HEC

- 1 − Discard cells with HEC errors
- 0 − Cells with HEC errors are passed

Definition at line **1483** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardIdle

[21] Discard Idle Cells based on DefineIdle register values

- 1 − Discard IDLE cells
- 0 − IDLE cells passed

Definition at line **1491** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardParErr

[22] Discard cells containing parity errors

- 1 − Discard cells with parity errors
- 0 − Cells with parity errors are passed

Definition at line **1487** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::enbHecErrCnt

[20] Enable Receive HEC Error Count

- 1 − Enable count of received cells containing HEC errors
- 0 − No count is maintained.

Definition at line **1495** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::enbIdleCellCnt

[18] Enable Receive Idle Cell Count

- 1 − Enable count of Idle cells received.
- 0 − No count is maintained.

Definition at line **1503** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::enbParErrCnt

[19] Enable Parity Error Count

- 1 − Enable count of received cells containing Parity errors
- 0 − No count is maintained

Definition at line **1499** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::enbRxCellCnt

[16] Enable Receive Valid Cell Count of non−idle/non−error cells

- 1 − Enable count of valid cells received − non−idle/non−error
- 0 − No count is maintained.

Definition at line **1511** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::enbSizeErrCnt

[17] Enable Receive Size Error Count

- 1 − Enable count of received cells of incorrect size

- 0 − No count is maintained.

Definition at line **1507** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::invalidHecOvrInt

[11] Enable CBI Receive Status Condition if the InvalidHecCount register overflows

- 1 − CBI Receive Condition asserted.
- 0 − No CBI Receive Condition asserted

Definition at line **1522** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::invalidParOvrInt

[10] Enable CBI Receive Status Condition if the InvalidParCount register overflows

- 1 − CBI Receive Condition asserted
- 0 − No CBI Receive Condition asserted

Definition at line **1527** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::invalidSizeOvrInt

[9] Enable CBI Receive Status Condition if the InvalidSizeCount register overflows

- 1 − CBI Receive Status Condition asserted.
- ¸0 − No CBI Receive Status asserted

Definition at line **1532** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::phyStatsRxEnb

[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in RxStatsConfig register

- 1 − Enable statistics for specified receive PHY address.
- 0 − Disable statistics for specified receive PHY address.

Definition at line **1456** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::reserved_1

[15:13] These bits are always 0

Definition at line **1515** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::reserved_2

[7:5] These bits are always 0

Definition at line **1541** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxAddrMask

[4:0] This register is used as a mask to allow the user to increase the PHY receive address range

The register should be programmed with the address−range limit, i.e. if set to 0x3 the address range increases to a maximum of 4 addresses.

Definition at line **1543** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxCellOvrInt

[12] Enable CBI Utopia Receive Status Condition if the RxCellCount register overflows

- 1 − CBI Receive Status asserted.
- 0 − No CBI Receive Status asserted.

Definition at line **1517** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxIdleOvrInt

[8] Enable CBI Receive Status Condition if the RxIdleCount overflows

- 1 − CBI Receive Condition asserted.
- 0 − No CBI Receive Condition asserted

Definition at line **1537** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxCLP

[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 − CLP field is valid.
- 0 − CLP field ignored.

Definition at line **1478** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxEnb

[27] This register is used to enable or disable ATM statistics gathering based on a specific VPI/VCI address

- 1 − Enable statistics for specified VPI/VCI address.
- 0 − Disable statistics for specified VPI/VCI address.

Definition at line **1462** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxGFC

[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

GFC is only available at the UNI and uses the first 4−bits of the VPI field.

- 1 − GFC field is valid.
- 0 − GFC field ignored.

Definition at line **1467** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxPTI

[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 − PTI field is valid.
- 0 − PTI field ignored.

Definition at line **1473** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxStatsConfig_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx stats config Register.

Data Fields

unsigned int **vpi**:12
[31:20] ATM VPI VPI [11:0] OR GFC [3:0] and VPI [7:0]

- *Note: if VCStatsRxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16
[19:4] VCI [15:0] or PHY Address [4]

unsigned int **pti**:3
[3:1] PTI [2:0] or or PHY Address [3:1]

- *Note: if VCStatsRxPTI is set to 0 the PTI field is ignored in test*

unsigned int **clp**:1
[0] CLP [0] or PHY Address [0]

- *Note: if VCStatsRxCLP is set to 0 the CLP field is ignored in test*

Detailed Description

Utopia Rx stats config Register.

Definition at line **1394** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig_::clp

[0] CLP [0] or PHY Address [0]

- Note: if VCStatsRxCLP is set to 0 the CLP field is ignored in test
- Note: if VCStatsRxEnb is set to 0 only the PHY port address is used for statistics gathering..

Definition at line **1407** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig_::pti

[3:1] PTI [2:0] or or PHY Address [3:1]

- Note: if VCStatsRxPTI is set to 0 the PTI field is ignored in test
- Note: if VCStatsRxEnb is set to 0 only the PHY port address is used for statistics gathering..

Definition at line **1402** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig_::vci

[19:4] VCI [15:0] or PHY Address [4]

Definition at line **1400** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig_::vpi

[31:20] ATM VPI VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCStatsRxGFC is set to 0 the GFC field is ignored in test

Definition at line **1397** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable0_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy0**:5
[31−27] Rx Mapping value of logical phy 0

unsigned int **phy1**:5
[26−22] Rx Mapping value of logical phy 1

unsigned int **phy2**:5
[21−17] Rx Mapping value of logical phy 2

unsigned int **reserved_1**:1
[16] These bits are always 0

unsigned int **phy3**:5
[15−11] Rx Mapping value of logical phy 3

unsigned int **phy4**:5
[10−6] Rx Mapping value of logical phy 4

unsigned int **phy5**:5
[5−1] Rx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1554** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy0

[31−27] Rx Mapping value of logical phy 0

Definition at line **1557** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy1

[26−22] Rx Mapping value of logical phy 1

Definition at line **1559** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy2

[21−17] Rx Mapping value of logical phy 2

Definition at line **1561** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy3

[15−11] Rx Mapping value of logical phy 3

Definition at line **1565** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy4

[10−6] Rx Mapping value of logical phy 4

Definition at line **1567** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy5

[5−1] Rx Mapping value of logical phy 5

Definition at line **1569** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::reserved_1

[16] These bits are always 0

Definition at line **1563** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::reserved_2

[0] These bits are always 0

Definition at line **1571** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable1_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy6**:5
[31−27] Rx Mapping value of logical phy 6

unsigned int **phy7**:5
[26−22] Rx Mapping value of logical phy 7

unsigned int **phy8**:5
[21−17] Rx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy9**:5
[15−11] Rx Mapping value of logical phy 3

unsigned int **phy10**:5
[10−6] Rx Mapping value of logical phy 4

unsigned int **phy11**:5
[5−1] Rx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1581** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy10

[10−6] Rx Mapping value of logical phy 4

Definition at line **1594** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy11

[5−1] Rx Mapping value of logical phy 5

Definition at line **1596** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy6

[31−27] Rx Mapping value of logical phy 6

Definition at line **1584** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy7

[26−22] Rx Mapping value of logical phy 7

Definition at line **1586** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy8

[21−17] Rx Mapping value of logical phy 8

Definition at line **1588** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy9

[15−11] Rx Mapping value of logical phy 3

Definition at line **1592** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::reserved_1

[16−0] These bits are always 0

Definition at line **1590** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::reserved_2

[0] These bits are always 0

Definition at line **1598** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable2_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy12**:5
[31−27] Rx Mapping value of logical phy 6

unsigned int **phy13**:5
[26−22] Rx Mapping value of logical phy 7

unsigned int **phy14**:5
[21−17] Rx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy15**:5
[15−11] Rx Mapping value of logical phy 3

unsigned int **phy16**:5
[10−6] Rx Mapping value of logical phy 4

unsigned int **phy17**:5
[5−1] Rx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1608** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy12

[31−27] Rx Mapping value of logical phy 6

Definition at line **1611** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy13

[26−22] Rx Mapping value of logical phy 7

Definition at line **1613** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy14

[21−17] Rx Mapping value of logical phy 8

Definition at line **1615** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy15

[15−11] Rx Mapping value of logical phy 3

Definition at line **1619** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy16

[10−6] Rx Mapping value of logical phy 4

Definition at line **1621** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::phy17

[5−1] Rx Mapping value of logical phy 5

Definition at line **1623** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::reserved_1

[16−0] These bits are always 0

Definition at line **1617** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2_::reserved_2

[0] These bits are always 0

Definition at line **1625** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable3_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy18**:5
[31−27] Rx Mapping value of logical phy 6

unsigned int **phy19**:5
[26−22] Rx Mapping value of logical phy 7

unsigned int **phy20**:5
[21−17] Rx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy21**:5
[15−11] Rx Mapping value of logical phy 3

unsigned int **phy22**:5
[10−6] Rx Mapping value of logical phy 4

unsigned int **phy23**:5
[5−1] Rx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1633** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy18

[31−27] Rx Mapping value of logical phy 6

Definition at line **1636** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy19

[26−22] Rx Mapping value of logical phy 7

Definition at line **1638** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy20

[21−17] Rx Mapping value of logical phy 8

Definition at line **1640** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy21

[15−11] Rx Mapping value of logical phy 3

Definition at line **1644** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy22

[10−6] Rx Mapping value of logical phy 4

Definition at line **1646** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::phy23

[5−1] Rx Mapping value of logical phy 5

Definition at line **1648** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::reserved_1

[16−0] These bits are always 0

Definition at line **1642** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3_::reserved_2

[0] These bits are always 0

Definition at line **1650** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable4_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy24**:5
[31−27] Rx Mapping value of logical phy 6

unsigned int **phy25**:5
[26−22] Rx Mapping value of logical phy 7

unsigned int **phy26**:5
[21−17] Rx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy27**:5
[15−11] Rx Mapping value of logical phy 3

unsigned int **phy28**:5
[10−6] Rx Mapping value of logical phy 4

unsigned int **phy29**:5
[5−1] Rx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1658** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy24

[31−27] Rx Mapping value of logical phy 6

Definition at line **1661** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy25

[26−22] Rx Mapping value of logical phy 7

Definition at line **1663** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy26

[21−17] Rx Mapping value of logical phy 8

Definition at line **1665** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy27

[15−11] Rx Mapping value of logical phy 3

Definition at line **1669** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy28

[10−6] Rx Mapping value of logical phy 4

Definition at line **1671** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::phy29

[5−1] Rx Mapping value of logical phy 5

Definition at line **1673** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::reserved_1

[16−0] These bits are always 0

Definition at line **1667** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4_::reserved_2

[0] These bits are always 0

Definition at line **1675** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtRxTransTable5_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

Data Fields

unsigned int **phy30**:5
[31−27] Rx Mapping value of logical phy 6

unsigned int **reserved_1**:27
[26−0] These bits are always 0

Detailed Description

Utopia Rx translation table Register.

Definition at line **1683** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable5_::phy30

[31−27] Rx Mapping value of logical phy 6

Definition at line **1686** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable5_::reserved_1

[26−0] These bits are always 0

Definition at line **1688** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtSysConfig_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

NPE setup Register.

Data Fields

unsigned int **reserved_1**:2
[31−30] These bits are always 0

unsigned int **txEnbFSM**:1
[29] Enables the operation ofthe Utopia Transmit FSM

- *1 − FSM enabled*
- *0 − FSM inactive*

unsigned int **rxEnbFSM**:1
[28] Enables the operation ofthe Utopia Revieve FSM

- *1 − FSM enabled*
- *0 − FSM inactive*

unsigned int **disablePins**:1
[27] Disable Utopia interface I/O pins forcing the signals to an inactive state

unsigned int **tstLoop**:1
[26] Test Loop Back Enable

unsigned int **txReset**:1
[25] Resets the Utopia Coprocessor transmit module to a known state

unsigned int **rxReset**:1
[24] Resets the Utopia Coprocessor receive module to a known state

unsigned int **reserved_2**:24
[23−0] These bits are always 0

Detailed Description

NPE setup Register.

Definition at line **1697** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::disablePins

[27] Disable Utopia interface I/O pins forcing the signals to an inactive state

Note that this bit is set on reset and must be de−asserted

- 0 − Normal data transfer
- 1 − Utopia interface pins are forced inactive

Definition at line **1709** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::reserved_1

[31−30] These bits are always 0

Definition at line **1700** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::reserved_2

[23−0] These bits are always 0

Definition at line **1736** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::rxEnbFSM

[28] Enables the operation ofthe Utopia Revieve FSM

- 1 − FSM enabled
- 0 − FSM inactive

Definition at line **1705** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::rxReset

[24] Resets the Utopia Coprocessor receive module to a known state

- Note: All receive configuration and status registers will be reset to their reset values.
- 0 − Normal operating mode
- 1 − Reset receive modules

Definition at line **1729** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::tstLoop

[26] Test Loop Back Enable

- Note: For loop back to function RxMode and Tx Mode must both be set to single PHY mode.
- 0 − Loop back
- 1 − Normal operating mode

Definition at line **1715** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::txEnbFSM

[29] Enables the operation ofthe Utopia Transmit FSM

- 1 − FSM enabled
- 0 − FSM inactive

Definition at line **1701** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::txReset

[25] Resets the Utopia Coprocessor transmit module to a known state

- Note: All transmit configuration and status registers will be reset to their reset values.
- 0 − Normal operating mode¸
- 1 − Reset transmit modules

Definition at line **1722** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxConfig_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx Config Register.

Data Fields

unsigned
int **reserved_1**:1
[31] These bits are always 0.

unsigned
int **txInterface**:1
[30] Utopia Transmit Interface

unsigned
int **txMode**:1
[29] Utopia Transmit Mode

unsigned
int **txOctet**:1
[28] Utopia Transmit cell transfer protocol

unsigned
int **txParity**:1
[27] Utopia Transmit parity enabled when set

unsigned
int **txEvenParity**:1
[26] Utopia Transmit Parity Mode

- *1 − Even Parity Generated*

unsigned
int **txHEC**:1
[25] Header Error Check Insertion Mode

unsigned
int **txCOSET**:1
[24] If enabled the HEC is Exclusive−ORÆed with the value 0x55 before being presented on the Utopia bus

unsigned
int **reserved_2**:1
[23] These bits are always 0

unsigned
int **txCellSize**:7
[22:16] Transmit expected cell size

unsigned
int **reserved_3**:3
[15:13] These bits are always 0

unsigned
int **txAddrRange**:5
[12:8] When configured as an ATM master in MPHY mode this register specifies the upper limit of the PHY polling logical range

unsigned
int **reserved_4**:3
[7:5] These bits are always 0

unsigned
int **txPHYAddr**:5
[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Detailed Description

Utopia Tx Config Register.

Definition at line **986** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::reserved_1

[31] These bits are always 0.

Definition at line **989** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::reserved_2

[23] These bits are always 0

Definition at line **1027** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::reserved_3

[15:13] These bits are always 0

Definition at line **1032** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::reserved_4

[7:5] These bits are always 0

Definition at line **1037** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txAddrRange

[12:8] When configured as an ATM master in MPHY mode this register specifies the upper limit of the PHY polling logical range

The number of active PHYs are TxAddrRange + 1.

Definition at line **1033** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txCellSize

[22:16] Transmit expected cell size

Configures the cell size for the transmit module: Values between 52−64 are valid.

Definition at line **1029** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txCOSET

[24] If enabled the HEC is Exclusive−ORÆed with the value 0x55 before being presented on the Utopia bus

- 1 − Enable HEC ExOR with value 0x55
- 0 − Use generated HEC value.

Definition at line **1021** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txEvenParity

[26] Utopia Transmit Parity Mode

- 1 − Even Parity Generated

- 0 − Odd Parity Generated.

Definition at line **1012** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txHEC

[25] Header Error Check Insertion Mode

Specifies if the transmit cell header check byte is calculated and inserted when set.

- 1 − Generate HEC.
- 0 − Disable HEC generation.

Definition at line **1016** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txInterface

[30] Utopia Transmit Interface

The following encoding is used to set the Utopia Transmit interface as ATM master or PHY slave:

- 1 − PHY
- 0 − ATM

Definition at line **990** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txMode

[29] Utopia Transmit Mode

The following encoding is used to set the Utopia Transmit mode to SPHY or MPHY:

- 1 − SPHY
- 0 − MPHY

Definition at line **996** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txOctet

[28] Utopia Transmit cell transfer protocol

Used to set the Utopia cell transfer protocol to Octet−level handshaking. Note this is only applicable in SPHY mode.

- 1 − Octet−handshaking enabled

- 0 − Cell−handshaking enabled

Definition at line **1001** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txParity

[27] Utopia Transmit parity enabled when set

TxEvenParity defines the parity format odd/even.

- 1 − Enable Parity generation.
- 0 − ut_op_prty held low.

Definition at line **1007** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txPHYAddr

[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Definition at line **1038** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxDefineIdle_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx idle cells Register.

Data Fields

unsigned int **vpi**:12
[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- *Note: if VCIdleTxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16
[19:4] ATM VCI [15:0]

unsigned int **pti**:3
[3:1] ATM PTI PTI [2:0]

- *Note: if VCIdleTxPTI is set to 0 the PTI field is ignored in test.*

unsigned int **clp**:1
[0] ATM CLP [0]

- *Note: if VCIdleTxCLP is set to 0 the CLP field is ignored in test.*

Detailed Description

Utopia Tx idle cells Register.

Definition at line **1076** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle_::clp

[0] ATM CLP [0]

- Note: if VCIdleTxCLP is set to 0 the CLP field is ignored in test.

Definition at line **1087** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle_::pti

[3:1] ATM PTI PTI [2:0]

- Note: if VCIdleTxPTI is set to 0 the PTI field is ignored in test.

Definition at line **1084** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle_::vci

[19:4] ATM VCI [15:0]

Definition at line **1082** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCIdleTxGFC is set to 0 the GFC field is ignored in test

Definition at line **1079** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxEnableFields_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx ienable fields Register.

Data Fields

unsigned int **defineTxIdleGFC**:1
[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

unsigned int **defineTxIdlePTI**:1
[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

unsigned int **defineTxIdleCLP**:1
[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

unsigned int **phyStatsTxEnb**:1
[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in TxStatsConfig register

unsigned int **vcStatsTxEnb**:1
[27] This register is used to change the ATM statistics−gathering mode from the specified logical PHY address to a specific VPI/VCI address

unsigned int **vcStatsTxGFC**:1
[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned int **vcStatsTxPTI**:1
[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned int **vcStatsTxCLP**:1
[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

unsigned int **reserved_1**:3
[23−21] These bits are always 0

unsigned int **txPollStsInt**:1
[20] Enable the assertion of the ucp_tx_poll_sts condition where there is a change in

polling status

unsigned int **txCellOvrInt**:1
[19] Enable TxCellCount overflow CBI Transmit Status condition assertion

unsigned int **txIdleCellOvrInt**:1
[18] Enable TxIdleCellCount overflow Transmit Status Condition

- *1 − If TxIdleCellCountOvr is set assert the Transmit Status Condition*
- *0 − No CBI Transmit Status condition assertion..*

unsigned int **enbIdleCellCnt**:1
[17] Enable Transmit Idle Cell Count

unsigned int **enbTxCellCnt**:1
[16] Enable Transmit Valid Cell Count of non−idle/non−error cells

- *1 − Enable count of valid cells transmitted− non−idle/non−error*
- *0 − No count is maintained.*

unsigned int **reserved_2**:16
[15:0] These bits are always 0

Detailed Description

Utopia Tx ienable fields Register.

Definition at line **1098** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::defineTxIdleCLP

[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

- 1 − CLP field is valid.
- 0 − CLP field ignored.

Definition at line **1111** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::defineTxIdleGFC

[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

- 1 − GFC field is valid.
- 0 − GFC field ignored.

Definition at line **1101** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::defineTxIdlePTI

[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

- 1 − PTI field is valid
- 0 − PTI field ignored.

Definition at line **1106** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::enbIdleCellCnt

[17] Enable Transmit Idle Cell Count

- 1 − Enable count of Idle cells transmitted.
- 0 − No count is maintained.

Definition at line **1161** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::enbTxCellCnt

[16] Enable Transmit Valid Cell Count of non−idle/non−error cells

- 1 − Enable count of valid cells transmitted− non−idle/non−error
- 0 − No count is maintained.

Definition at line **1165** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::phyStatsTxEnb

[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in TxStatsConfig register

- 1 − Enable statistics for specified transmit PHY address.
- 0 − Disable statistics for specified transmit PHY address.

Definition at line **1116** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::reserved_1

[23−21] These bits are always 0

Definition at line **1145** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::reserved_2

[15:0] These bits are always 0

Definition at line **1169** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txCellOvrInt

[19] Enable TxCellCount overflow CBI Transmit Status condition assertion

- 1 − If TxCellCountOvr is set assert the Transmit Status Condition.
- 0 − No CBI Transmit Status condition assertion

Definition at line **1152** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txIdleCellOvrInt

[18] Enable TxIdleCellCount overflow Transmit Status Condition

- 1 − If TxIdleCellCountOvr is set assert the Transmit Status Condition
- 0 − No CBI Transmit Status condition assertion..

Definition at line **1157** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txPollStsInt

[20] Enable the assertion of the ucp_tx_poll_sts condition where there is a change in polling status

- 1 − ucp_tx_poll_sts asserted whenever there is a change in status
- 0 − ucp_tx_poll_sts asserted if ANY transmit PHY is available

Definition at line **1147** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxCLP

[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 − CLP field is valid
- 0 − CLP field ignored.

Definition at line **1140** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxEnb

[27] This register is used to change the ATM statistics−gathering mode from the specified logical PHY address to a specific VPI/VCI address

- 1 − Enable statistics for specified VPI/VCI address.
- 0 − Disable statistics for specified VPI/VCI address

Definition at line **1122** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxGFC

[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

GFC is only available at the UNI and uses the first 4−bits of the VPI field.

- 1 − GFC field is valid
- 0 − GFC field ignored.

Definition at line **1128** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxPTI

[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 − PTI field is valid
- 0 − PTI field ignored.

Definition at line **1135** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxStatsConfig_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx stats Register.

Data Fields

unsigned int **vpi**:12
[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- *Note: if VCStatsTxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16
[19:4] ATM VCI [15:0] or PHY Address[4]

unsigned int **pti**:3
[3:1] ATM PTI [2:0] or PHY Address[3:1]

- *Note: if VCStatsTxPTI is set to 0 the PTI field is ignored in test*

unsigned int **clp**:1
[0] ATM CLP or PHY Address [0]

- *Note: if VCStatsTxCLP is set to 0 the CLP field is ignored in test*

Detailed Description

Utopia Tx stats Register.

Definition at line **1050** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig_::clp

[0] ATM CLP or PHY Address [0]

- Note: if VCStatsTxCLP is set to 0 the CLP field is ignored in test
- Note: if VCStatsTxEnb is set to 0 only the transmit PHY port address as defined by this register is used for ATM statistics [4:0].

Definition at line **1063** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig_::pti

[3:1] ATM PTI [2:0] or PHY Address[3:1]

- Note: if VCStatsTxPTI is set to 0 the PTI field is ignored in test
- Note: if VCStatsTxEnb is set to 0 only the transmit PHY port address as defined by this register is used for ATM statistics [4:0].

Definition at line **1058** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig_::vci

[19:4] ATM VCI [15:0] or PHY Address[4]

Definition at line **1056** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCStatsTxGFC is set to 0 the GFC field is ignored in test

Definition at line **1053** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable0_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy0**:5
[31−27] Tx Mapping value of logical phy 0

unsigned int **phy1**:5
[26−22] Tx Mapping value of logical phy 1

unsigned int **phy2**:5
[21−17] Tx Mapping value of logical phy 2

unsigned int **reserved_1**:1
[16] These bits are always 0.

unsigned int **phy3**:5
[15−11] Tx Mapping value of logical phy 3

unsigned int **phy4**:5
[10−6] Tx Mapping value of logical phy 4

unsigned int **phy5**:5
[5−1] Tx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1177** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy0

[31−27] Tx Mapping value of logical phy 0

Definition at line **1180** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy1

[26−22] Tx Mapping value of logical phy 1

Definition at line **1182** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy2

[21−17] Tx Mapping value of logical phy 2

Definition at line **1184** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy3

[15−11] Tx Mapping value of logical phy 3

Definition at line **1188** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy4

[10−6] Tx Mapping value of logical phy 4

Definition at line **1190** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy5

[5−1] Tx Mapping value of logical phy 5

Definition at line **1192** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::reserved_1

[16] These bits are always 0.

Definition at line **1186** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::reserved_2

[0] These bits are always 0

Definition at line **1194** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable1_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy6**:5
[31−27] Tx Mapping value of logical phy 6

unsigned int **phy7**:5
[26−22] Tx Mapping value of logical phy 7

unsigned int **phy8**:5
[21−17] Tx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy9**:5
[15−11] Tx Mapping value of logical phy 3

unsigned int **phy10**:5
[10−6] Tx Mapping value of logical phy 4

unsigned int **phy11**:5
[5−1] Tx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1202** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy10

[10−6] Tx Mapping value of logical phy 4

Definition at line **1215** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy11

[5−1] Tx Mapping value of logical phy 5

Definition at line **1217** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy6

[31−27] Tx Mapping value of logical phy 6

Definition at line **1205** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy7

[26−22] Tx Mapping value of logical phy 7

Definition at line **1207** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy8

[21−17] Tx Mapping value of logical phy 8

Definition at line **1209** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy9

[15−11] Tx Mapping value of logical phy 3

Definition at line **1213** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::reserved_1

[16−0] These bits are always 0

Definition at line **1211** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::reserved_2

[0] These bits are always 0

Definition at line **1219** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable2_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy12**:5
[31−27] Tx Mapping value of logical phy 6

unsigned int **phy13**:5
[26−22] Tx Mapping value of logical phy 7

unsigned int **phy14**:5
[21−17] Tx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy15**:5
[15−11] Tx Mapping value of logical phy 3

unsigned int **phy16**:5
[10−6] Tx Mapping value of logical phy 4

unsigned int **phy17**:5
[5−1] Tx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1227** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy12

[31−27] Tx Mapping value of logical phy 6

Definition at line **1230** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy13

[26−22] Tx Mapping value of logical phy 7

Definition at line **1232** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy14

[21−17] Tx Mapping value of logical phy 8

Definition at line **1234** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy15

[15−11] Tx Mapping value of logical phy 3

Definition at line **1238** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy16

[10−6] Tx Mapping value of logical phy 4

Definition at line **1240** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy17

[5−1] Tx Mapping value of logical phy 5

Definition at line **1242** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::reserved_1

[16−0] These bits are always 0

Definition at line **1236** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::reserved_2

[0] These bits are always 0

Definition at line **1244** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable3_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy18**:5
[31−27] Tx Mapping value of logical phy 6

unsigned int **phy19**:5
[26−22] Tx Mapping value of logical phy 7

unsigned int **phy20**:5
[21−17] Tx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy21**:5
[15−11] Tx Mapping value of logical phy 3

unsigned int **phy22**:5
[10−6] Tx Mapping value of logical phy 4

unsigned int **phy23**:5
[5−1] Tx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1252** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy18

[31−27] Tx Mapping value of logical phy 6

Definition at line **1255** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy19

[26−22] Tx Mapping value of logical phy 7

Definition at line **1257** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy20

[21−17] Tx Mapping value of logical phy 8

Definition at line **1259** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy21

[15−11] Tx Mapping value of logical phy 3

Definition at line **1263** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy22

[10−6] Tx Mapping value of logical phy 4

Definition at line **1265** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy23

[5−1] Tx Mapping value of logical phy 5

Definition at line **1267** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::reserved_1

[16−0] These bits are always 0

Definition at line **1261** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::reserved_2

[0] These bits are always 0

Definition at line **1269** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable4_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy24**:5
[31−27] Tx Mapping value of logical phy 6

unsigned int **phy25**:5
[26−22] Tx Mapping value of logical phy 7

unsigned int **phy26**:5
[21−17] Tx Mapping value of logical phy 8

unsigned int **reserved_1**:1
[16−0] These bits are always 0

unsigned int **phy27**:5
[15−11] Tx Mapping value of logical phy 3

unsigned int **phy28**:5
[10−6] Tx Mapping value of logical phy 4

unsigned int **phy29**:5
[5−1] Tx Mapping value of logical phy 5

unsigned int **reserved_2**:1
[0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1277** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy24

[31−27] Tx Mapping value of logical phy 6

Definition at line **1280** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy25

[26−22] Tx Mapping value of logical phy 7

Definition at line **1282** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy26

[21−17] Tx Mapping value of logical phy 8

Definition at line **1284** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy27

[15−11] Tx Mapping value of logical phy 3

Definition at line **1288** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy28

[10−6] Tx Mapping value of logical phy 4

Definition at line **1290** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy29

[5−1] Tx Mapping value of logical phy 5

Definition at line **1292** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::reserved_1

[16−0] These bits are always 0

Definition at line **1286** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::reserved_2

[0] These bits are always 0

Definition at line **1294** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaConfig::UtTxTransTable5_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

Data Fields

unsigned int **phy30**:5
[31−27] Tx Mapping value of logical phy 6

unsigned int **reserved_1**:27
[26−0] These bits are always 0

Detailed Description

Utopia Tx translation table Register.

Definition at line **1302** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable5_::phy30

[31−27] Tx Mapping value of logical phy 6

Definition at line **1305** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable5_::reserved_1

[26−0] These bits are always 0

Definition at line **1307** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaStatus Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia status.

Data Fields

unsigned int **utTxCellCount**
count of cells transmitted

unsigned int **utTxIdleCellCount**
count of idle cells transmitted

IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_ **utTxCellConditionStatus**
Tx cells condition status.

unsigned int **utRxCellCount**
count of cell received

unsigned int **utRxIdleCellCount**
count of idle cell received

unsigned int **utRxInvalidHECount**
count of invalid cell received because of HEC errors

unsigned int **utRxInvalidParCount**
count of invalid cell received because of parity errors

unsigned int **utRxInvalidSizeCount**
count of invalid cell received because of cell size errors

IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_ **utRxCellConditionStatus**
Rx cells condition status.

Detailed Description

Utopia status.

This structure is used to set/get the Utopia status parameters

- contains debug cell counters, to be accessed during a read operation

Note:

− the exact description of all parameters is done in the Utopia reference documents.

Definition at line **1753** of file **IxAtmdAccCtrl.h**.

Field Documentation

struct **IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_**
IxAtmdAccUtopiaStatus::utRxCellConditionStatus

Rx cells condition status.

unsigned int IxAtmdAccUtopiaStatus::utRxCellCount

count of cell received

Definition at line **1794** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::utRxIdleCellCount

count of idle cell received

Definition at line **1795** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::utRxInvalidHECount

count of invalid cell received because of HEC errors

Definition at line **1796** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::utRxInvalidParCount

count of invalid cell received because of parity errors

Definition at line **1799** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::utRxInvalidSizeCount

count of invalid cell received because of cell size errors

Definition at line **1802** of file **IxAtmdAccCtrl.h**.

struct **IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_**
IxAtmdAccUtopiaStatus::utTxCellConditionStatus

Tx cells condition status.

unsigned int IxAtmdAccUtopiaStatus::utTxCellCount

count of cells transmitted

Definition at line **1756** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::utTxIdleCellCount

count of idle cells transmitted

Definition at line **1758** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx Status Register.

Data Fields

unsigned int **reserved_1**:3
[31:29] These bits are always 0.

unsigned int **rxCellCountOvr**:1
[28] This bit is set if the RxCellCount register overflows

unsigned int **invalidHecCountOvr**:1
[27] This bit is set if the InvalidHecCount register overflows.

unsigned int **invalidParCountOvr**:1
[26] This bit is set if the InvalidParCount register overflows.

unsigned int **invalidSizeCountOvr**:1
[25] This bit is set if the InvalidSizeCount register overflows.

unsigned int **rxIdleCountOvr**:1
[24] This bit is set if the RxIdleCount register overflows.

unsigned int **reserved_2**:4
[23:20] These bits are always 0

unsigned int **rxFIFO2Underflow**:1
[19] This bit is set if 64−byte Receive FIFO2 indicates a FIFO underflow error condition

unsigned int **rxFIFO1Underflow**:1
[18] This bit is set if 64−byte Receive FIFO1 indicates a FIFO underflow error condition

unsigned int **rxFIFO2Overflow**:1
[17] This bit is set if 64−byte Receive FIFO2 indicates a FIFO overflow error condition

unsigned int **rxFIFO1Overflow**:1
[16] This bit is set if 64−byte Receive FIFO1 indicates a FIFO overflow error condition

unsigned int **reserved_3**:16
[15:0] These bits are always 0

Detailed Description

Utopia Rx Status Register.

Definition at line **1812** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::invalidHecCountOvr

[27] This bit is set if the InvalidHecCount register overflows.

Definition at line **1817** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::invalidParCountOvr

[26] This bit is set if the InvalidParCount register overflows.

Definition at line **1818** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::invalidSizeCountOvr

[25] This bit is set if the InvalidSizeCount register overflows.

Definition at line **1819** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::reserved_1

[31:29] These bits are always 0.

Definition at line **1815** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::reserved_2

[23:20] These bits are always 0

Definition at line **1821** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::reserved_3

[15:0] These bits are always 0

Definition at line **1834** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxCellCountOvr

[28] This bit is set if the RxCellCount register overflows

Definition at line **1816** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO1Overflow

[16] This bit is set if 64−byte Receive FIFO1 indicates a FIFO overflow error condition

Definition at line **1831** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO1Underflow

[18] This bit is set if 64−byte Receive FIFO1 indicates a FIFO underflow error condition

Definition at line **1825** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO2Overflow

[17] This bit is set if 64−byte Receive FIFO2 indicates a FIFO overflow error condition

Definition at line **1828** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO2Underflow

[19] This bit is set if 64−byte Receive FIFO2 indicates a FIFO underflow error condition

Definition at line **1822** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxIdleCountOvr

[24] This bit is set if the RxIdleCount register overflows.

Definition at line **1820** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_ Struct Reference

[Intel (R) IXP400 Software ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx Status Register.

Data Fields

unsigned int **reserved_1**:2
[31:30] These bits are always 0

unsigned int **txFIFO2Underflow**:1
[29] This bit is set if 64−byte Transmit FIFO2 indicates a FIFO underflow error condition

unsigned int **txFIFO1Underflow**:1
[28] This bit is set if 64−byte Transmit FIFO1 indicates a FIFO underflow error condition

unsigned int **txFIFO2Overflow**:1
[27] This bit is set if 64−byte Transmit FIFO2 indicates a FIFO overflow error condition

unsigned int **txFIFO1Overflow**:1
[26] This bit is set if 64−byte Transmit FIFO1 indicates a FIFO overflow error condition

unsigned int **txIdleCellCountOvr**:1
[25] This bit is set if the TxIdleCellCount register overflows

unsigned int **txCellCountOvr**:1
[24] This bit is set if the TxCellCount register overflows

unsigned int **reserved_2**:24
[23:0] These bits are always 0

Detailed Description

Utopia Tx Status Register.

Definition at line **1765** of file **IxAtmdAccCtrl.h**.

Field Documentation

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::reserved_1

[31:30] These bits are always 0

Definition at line **1768** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::reserved_2

[23:0] These bits are always 0

Definition at line **1791** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txCellCountOvr

[24] This bit is set if the TxCellCount register overflows

Definition at line **1788** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO1Overflow

[26] This bit is set if 64−byte Transmit FIFO1 indicates a FIFO overflow error condition

Definition at line **1781** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO1Underflow

[28] This bit is set if 64−byte Transmit FIFO1 indicates a FIFO underflow error condition

Definition at line **1773** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO2Overflow

[27] This bit is set if 64−byte Transmit FIFO2 indicates a FIFO overflow error condition

Definition at line **1777** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO2Underflow

[29] This bit is set if 64−byte Transmit FIFO2 indicates a FIFO underflow error condition

Definition at line **1769** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txIdleCellCountOvr

[25] This bit is set if the TxIdleCellCount register overflows

Definition at line **1785** of file **IxAtmdAccCtrl.h**.

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

IxAtmmPortCfg Struct Reference

[Intel (R) IXP400 Software ATM Manager (IxAtmm) API]

Structure contains port−specific information required to initialize IxAtmm, and specifically, the IXP400 UTOPIA Level−2 device.

Data Fields

unsigned **reserved_1**:11
[31:21] Should be zero

unsigned **UtopiaTxPhyAddr**:5
[20:16] Address of the transmit (Tx) PHY for this port on the 5−bit UTOPIA Level−2 address bus

unsigned **reserved_2**:11
[15:5] Should be zero

unsigned **UtopiaRxPhyAddr**:5
[4:0] Address of the receive (Rx) PHY for this port on the 5−bit UTOPIA Level−2 address bus

---

Detailed Description

Structure contains port−specific information required to initialize IxAtmm, and specifically, the IXP400 UTOPIA Level−2 device.

Definition at line **161** of file **IxAtmm.h**.

Field Documentation

unsigned IxAtmmPortCfg::reserved_1

[31:21] Should be zero

Definition at line **162** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::reserved_2

[15:5] Should be zero

Definition at line **167** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::UtopiaRxPhyAddr

[4:0] Address of the receive (Rx) PHY for this port on the 5−bit UTOPIA Level−2 address bus

Definition at line **168** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::UtopiaTxPhyAddr

[20:16] Address of the transmit (Tx) PHY for this port on the 5−bit UTOPIA Level−2 address bus

Definition at line **163** of file **IxAtmm.h**.

The documentation for this struct was generated from the following file:

- **IxAtmm.h**

IxAtmmVc Struct Reference

[Intel (R) IXP400 Software ATM Manager (IxAtmm) API]

This structure describes the required attributes of a virtual connection.

Data Fields

unsigned **vpi**
VPI value of this virtual connection.

unsigned **vci**
VCI value of this virtual connection.

IxAtmmVcDirection direction
VC direction.

IxAtmTrafficDescriptor trafficDesc
Traffic descriptor of this virtual connection.

Detailed Description

This structure describes the required attributes of a virtual connection.

Definition at line **136** of file **IxAtmm.h**.

Field Documentation

IxAtmmVcDirection IxAtmmVc::direction

VC direction.

Definition at line **139** of file **IxAtmm.h**.

IxAtmTrafficDescriptor IxAtmmVc::trafficDesc

Traffic descriptor of this virtual connection.

This structure is defined by the **Intel (R) IXP400 Software ATM Transmit Scheduler (IxAtmSch) API** component.

Definition at line **143** of file **IxAtmm.h**.

unsigned IxAtmmVc::vci

VCI value of this virtual connection.

Definition at line **138** of file **IxAtmm.h**.

unsigned IxAtmmVc::vpi

VPI value of this virtual connection.

Definition at line **137** of file **IxAtmm.h**.

The documentation for this struct was generated from the following file:

- **IxAtmm.h**

IxAtmScheduleTable Struct Reference

[Intel (R) IXP400 Software ATM Types (IxAtmTypes)]

This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.

Data Fields

unsigned **tableSize**
Number of entries.

unsigned **totalCellSlots**
Number of cells.

IxAtmScheduleTableEntry * **table**
Pointer to schedule entries.

Detailed Description

This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.

The schedule table consists of a series of entries each of which will schedule one or more cells from a particular registered VC. The total number of cells scheduled and the total number of entries in the table are also indicated.

Definition at line **355** of file **IxAtmTypes.h**.

Field Documentation

IxAtmScheduleTableEntry* IxAtmScheduleTable::table

Pointer to schedule entries.

Pointer to an array containing tableSize entries

Definition at line **368** of file **IxAtmTypes.h**.

unsigned IxAtmScheduleTable::tableSize

Number of entries.

Indicates the total number of entries in the table.

Definition at line **357** of file **IxAtmTypes.h**.

unsigned IxAtmScheduleTable::totalCellSlots

Number of cells.

Indicates the total number of ATM cells which are scheduled by all the entries in the table.

Definition at line **362** of file **IxAtmTypes.h**.

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**

IxAtmScheduleTableEntry Struct Reference

[Intel (R) IXP400 Software ATM Types (IxAtmTypes)]

ATM Schedule Table entry.

Data Fields

IxAtmConnId connId
connection Id

unsigned int **numberOfCells**
number of cells to transmit

Detailed Description

ATM Schedule Table entry.

This IxAtmScheduleTableEntry is used by an ATM scheduler to inform IxAtmdAcc about the data to transmit (in term of cells per VC)

This structure defines

- the number of cells to be transmitted (numberOfCells)
- the VC connection to be used for transmission (connId).

Note:
− When the connection Id value is IX_ATM_IDLE_CELLS_CONNID, the corresponding number of idle cells will be transmitted to the hardware.

Definition at line **323** of file **IxAtmTypes.h**.

Field Documentation

IxAtmConnId IxAtmScheduleTableEntry::connId

connection Id

Identifier of VC from which cells are to be transmitted. When this valus is IX_ATM_IDLE_CELLS_CONNID, this indicates that the system should transmit the specified number of idle cells. Unknown connIds result in the transmission idle cells.

Definition at line **325** of file **IxAtmTypes.h**.

unsigned int IxAtmScheduleTableEntry::numberOfCells

number of cells to transmit

The number of contiguous cells to schedule from this VC at this point. The valid range is from 1 to *IX_ATM_SCHEDULETABLE_MAXCELLS_PER_ENTRY*. This number can swap over mbufs and pdus. OverSchduling results in the transmission of idle cells.

Definition at line **333** of file **IxAtmTypes.h**.

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**

IxAtmTrafficDescriptor Struct Reference

[Intel (R) IXP400 Software ATM Types (IxAtmTypes)]

Structure describing an ATM traffic contract for a Virtual Connection (VC).

Data Fields

IxAtmServiceCategory atmService
ATM service category.

UINT64 **pcr**
Peak Cell Rate − cells per second.

UINT64 **cdvt**
Cell Delay Variation Tolerance − in nanoseconds.

UINT64 **scr**
Sustained Cell Rate − cells per second.

UINT64 **mbs**
Max Burst Size − cells.

UINT64 **mcr**
Minimum Cell Rate − cells per second.

UINT64 **mfs**
Max Frame Size − cells.

Detailed Description

Structure describing an ATM traffic contract for a Virtual Connection (VC).

Structure is used to specify the requested traffic contract for a VC to the IxAtmSch component using the **ixAtmSchVcModelSetup** interface.

These parameters are defined by the ATM forum working group (http://www.atmforum.com).

Note:

Typical values for a voice channel 64 Kbit/s
- ◊ atmService *IX_ATM_RTVBR*
- ◊ pcr 400 (include IP overhead, and AAL5 trailer)
- ◊ cdvt 5000000 (5 ms)
- ◊ scr = pcr

Typical values for a data channel 800 Kbit/s

◊ atmService *IX_ATM_UBR*
◊ pcr 1962 (include IP overhead, and AAL5 trailer)
◊ cdvt 5000000 (5 ms)

Definition at line **262** of file **IxAtmTypes.h**.

Field Documentation

IxAtmServiceCategory IxAtmTrafficDescriptor::atmService

ATM service category.

Definition at line **264** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::cdvt

Cell Delay Variation Tolerance − in nanoseconds.

Definition at line **266** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::mbs

Max Burst Size − cells.

Definition at line **268** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::mcr

Minimum Cell Rate − cells per second.

Definition at line **269** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::mfs

Max Frame Size − cells.

Definition at line **270** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::pcr

Peak Cell Rate − cells per second.

Definition at line **265** of file **IxAtmTypes.h**.

UINT64 IxAtmTrafficDescriptor::scr

Sustained Cell Rate − cells per second.

Definition at line **267** of file **IxAtmTypes.h**.

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**

IxCryptoAccAuthCtx Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing authentication configuration parameters required to perform security functionality.

Data Fields

IxCryptoAccAuthAlgo authAlgo
authentication algorithm − MD5, SHA1, WEP_CRC

UINT32 **authDigestLen**
Digest length in bytes.

UINT32 **authKeyLen**
Hash key length in bytes.

UINT32 **aadLen**
Additional Authentication Data (AAD) length in bytes.

union {
  UINT8 **authKey** [IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]
  UINT8 **sha1Key** [IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]
  UINT8 **md5Key** [IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]
} **key**
Hash key, key is not required for WEP_CRC.

Detailed Description

Structure storing authentication configuration parameters required to perform security functionality.

Structure is used to specify the authentication context required for using the **Intel (R) IXP400 Software Security (IxCryptoAcc) API** interface.

Definition at line **516** of file **IxCryptoAcc.h**.

Field Documentation

UINT32 IxCryptoAccAuthCtx::aadLen

Additional Authentication Data (AAD) length in bytes.

This is the 16 bytes of initial block (called B0, in RFC 3610 and CCM Initial block in 802.11i spec), followed by additional authentication data (lengths− defined in respective standards).

Definition at line **523** of file **IxCryptoAcc.h**.

IxCryptoAccAuthAlgo IxCryptoAccAuthCtx::authAlgo

authentication algorithm − MD5, SHA1, WEP_CRC

Definition at line **518** of file **IxCryptoAcc.h**.

UINT32 IxCryptoAccAuthCtx::authDigestLen

Digest length in bytes.

Definition at line **521** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccAuthCtx::authKey[IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]

default hash key array

Definition at line **532** of file **IxCryptoAcc.h**.

UINT32 IxCryptoAccAuthCtx::authKeyLen

Hash key length in bytes.

Definition at line **522** of file **IxCryptoAcc.h**.

union { ... } IxCryptoAccAuthCtx::key

Hash key, key is not required for WEP_CRC.

UINT8 IxCryptoAccAuthCtx::md5Key[IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]

MD5 key.

Definition at line **536** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccAuthCtx::sha1Key[IX_CRYPTO_ACC_MAX_AUTH_KEY_LENGTH]

SHA1 key.

Definition at line **535** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccCipherCtx Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing cipher configuration parameters required to perform security functionality.

Data Fields

IxCryptoAccCipherAlgo cipherAlgo
Cipher Algorithm − DES, 3DES, AES, ARC4.

IxCryptoAccCipherMode cipherMode
Cipher mode − ECB, CBC, CTR, CCM.

UINT32 **cipherKeyLen**
Cipher key length in bytes.

union {
  UINT8 **cipherKey** [IX_CRYPTO_ACC_MAX_CIPHER_KEY_LENGTH]
  UINT8 **desKey** [IX_CRYPTO_ACC_DES_KEY_64]
  UINT8 **tripleDesKey** [IX_CRYPTO_ACC_3DES_KEY_192]
  UINT8 **aesKey128** [IX_CRYPTO_ACC_AES_KEY_128]
  UINT8 **aesKey192** [IX_CRYPTO_ACC_AES_KEY_192]
  UINT8 **aesKey256** [IX_CRYPTO_ACC_AES_KEY_256]
} **key**
Cipher key, key is not required for ARC4 during registration.

UINT32 **cipherBlockLen**
Cipher block length in bytes.

UINT32 **cipherInitialVectorLen**
Length of IV in bytes.

Detailed Description

Structure storing cipher configuration parameters required to perform security functionality.

Structure is used to specify the cipher context required for using the **Intel (R) IXP400 Software Security (IxCryptoAcc) API** interface.

Note:
SPI (Security Parameter Index) is not needed for crypto context registration as CTR counter block construction is client's responsibility

Definition at line **479** of file **IxCryptoAcc.h**.

Field Documentation

UINT8 IxCryptoAccCipherCtx::aesKey128[IX_CRYPTO_ACC_AES_KEY_128]

AES−128 key.

Definition at line **493** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccCipherCtx::aesKey192[IX_CRYPTO_ACC_AES_KEY_192]

AES−192 key.

Definition at line **494** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccCipherCtx::aesKey256[IX_CRYPTO_ACC_AES_KEY_256]

AES−256 key.

Definition at line **495** of file **IxCryptoAcc.h**.

IxCryptoAccCipherAlgo IxCryptoAccCipherCtx::cipherAlgo

Cipher Algorithm − DES, 3DES, AES, ARC4.

Definition at line **481** of file **IxCryptoAcc.h**.

UINT32 IxCryptoAccCipherCtx::cipherBlockLen

Cipher block length in bytes.

(DES/3DES − 8 bytes, AES − 16 bytes) (ARC4 − 1 byte)

Definition at line **499** of file **IxCryptoAcc.h**.

UINT32 IxCryptoAccCipherCtx::cipherInitialVectorLen

Length of IV in bytes.

Definition at line **504** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccCipherCtx::cipherKey[IX_CRYPTO_ACC_MAX_CIPHER_KEY_LENGTH]

default key array

Definition at line **488** of file **IxCryptoAcc.h**.

UINT32 IxCryptoAccCipherCtx::cipherKeyLen

Cipher key length in bytes.

Definition at line **485** of file **IxCryptoAcc.h**.

IxCryptoAccCipherMode IxCryptoAccCipherCtx::cipherMode

Cipher mode − ECB, CBC, CTR, CCM.

Definition at line **484** of file **IxCryptoAcc.h**.

UINT8 IxCryptoAccCipherCtx::desKey[IX_CRYPTO_ACC_DES_KEY_64]

DES key.

Definition at line **491** of file **IxCryptoAcc.h**.

union { ... } IxCryptoAccCipherCtx::key

Cipher key, key is not required for ARC4 during registration.

UINT8 IxCryptoAccCipherCtx::tripleDesKey[IX_CRYPTO_ACC_3DES_KEY_192]

3DES key

Definition at line **492** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccCtx Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing configuration parameters required to perform security functionality.

Data Fields

IxCryptoAccOperation operation
Types of operation.

IxCryptoAccCipherCtx cipherCtx
Cipher context.

IxCryptoAccAuthCtx authCtx
Authentication context.

BOOL **useDifferentSrcAndDestMbufs**
If TRUE, data is read from srcMbuf, result is written to destMbuf (non in−place operation).

Detailed Description

Structure storing configuration parameters required to perform security functionality.

Structure is used to specify the crypto context (hardware accelerator context) required for using the **Intel (R) IXP400 Software Security (IxCryptoAcc) API** interface.

Definition at line **550** of file **IxCryptoAcc.h**.

Field Documentation

IxCryptoAccAuthCtx IxCryptoAccCtx::authCtx

Authentication context.

Definition at line **554** of file **IxCryptoAcc.h**.

IxCryptoAccCipherCtx IxCryptoAccCtx::cipherCtx

Cipher context.

Definition at line **553** of file **IxCryptoAcc.h**.

IxCryptoAccOperation IxCryptoAccCtx::operation

Types of operation.

Definition at line **552** of file **IxCryptoAcc.h**.

BOOL IxCryptoAccCtx::useDifferentSrcAndDestMbufs

If TRUE, data is read from srcMbuf, result is written to destMbuf (non in−place operation).

If FALSE data is read from srcMbuf, and written back to srcMbuf (in−place operation). Default is FALSE. Note that only the crypted/authenticated data that is copied, not the entire source mbuf.

Definition at line **555** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccPkeEauBnAddSubMulOperands Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing input operands for large number addition (Carry | R = A + B) or subtraction operation (Borrow | R = A − B) or multiplication (R = A * B).

Data Fields

IxCryptoAccPkeEauOperand A

A : 1st operand of addition/subtraction/ multiplication.

IxCryptoAccPkeEauOperand B

B : 2nd operand of addition/subtraction multiplication.

Detailed Description

Structure storing input operands for large number addition (Carry | R = A + B) or subtraction operation (Borrow | R = A − B) or multiplication (R = A * B).

Definition at line **606** of file **IxCryptoAcc.h**.

Field Documentation

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauBnAddSubMulOperands::A

A : 1st operand of addition/subtraction/ multiplication.

Definition at line **608** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauBnAddSubMulOperands::B

B : 2nd operand of addition/subtraction multiplication.

Definition at line **610** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccPkeEauBnModOperands Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing input operands for large number modular reduction operation (R = A mod N).

Data Fields

IxCryptoAccPkeEauOperand A
A : Operand for modular reduction.

IxCryptoAccPkeEauOperand N
N : modulus.

Detailed Description

Structure storing input operands for large number modular reduction operation (R = A mod N).

Definition at line **619** of file **IxCryptoAcc.h**.

Field Documentation

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauBnModOperands::A

A : Operand for modular reduction.

Definition at line **621** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauBnModOperands::N

N : modulus.

Definition at line **622** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccPkeEauInOperands Union Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Union storing input operands required for all EAU operations. These input operands will be supplied to **ixCryptoAccPkeEauPerform** interface to perform EAU functionalities.

Data Fields

IxCryptoAccPkeEauModExpOperands modExpOpr
Modular exponential input operands.

IxCryptoAccPkeEauBnAddSubMulOperands bnAddSubMulOpr
Large number addition or subtraction or multiplication input operands.

IxCryptoAccPkeEauBnModOperands bnModOpr
Modular reduction input operands.

Detailed Description

Union storing input operands required for all EAU operations. These input operands will be supplied to **ixCryptoAccPkeEauPerform** interface to perform EAU functionalities.

Definition at line **631** of file **IxCryptoAcc.h**.

Field Documentation

IxCryptoAccPkeEauBnAddSubMulOperands IxCryptoAccPkeEauInOperands::bnAddSubMulOpr

Large number addition or subtraction or multiplication input operands.

Definition at line **637** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauBnModOperands IxCryptoAccPkeEauInOperands::bnModOpr

Modular reduction input operands.

Definition at line **641** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauModExpOperands IxCryptoAccPkeEauInOperands::modExpOpr

Modular exponential input operands.

Definition at line **633** of file **IxCryptoAcc.h**.

The documentation for this union was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccPkeEauModExpOperands Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing input operands for large number modular exponential operation (C = M^e mod N).

Data Fields

IxCryptoAccPkeEauOperand M
M : Base operand.

IxCryptoAccPkeEauOperand e
e : exponent

IxCryptoAccPkeEauOperand N
N : modulus.

Detailed Description

Structure storing input operands for large number modular exponential operation (C = M^e mod N).

Definition at line **593** of file **IxCryptoAcc.h**.

Field Documentation

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauModExpOperands::e

e : exponent

Definition at line **596** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauModExpOperands::M

M : Base operand.

Definition at line **595** of file **IxCryptoAcc.h**.

IxCryptoAccPkeEauOperand IxCryptoAccPkeEauModExpOperands::N

N : modulus.

Definition at line **597** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxCryptoAccPkeEauOperand Struct Reference

[Intel (R) IXP400 Software Security (IxCryptoAcc) API]

Structure storing operand / result data pointer and length for EAU functionality performing through **ixCryptoAccPkeEauPerform** interface.

Data Fields

UINT32 * **pData**
Data pointer which contains EAU input operand or output result.

UINT32 **dataLen**
data length in words (multiple of 32−bit)

Detailed Description

Structure storing operand / result data pointer and length for EAU functionality performing through **ixCryptoAccPkeEauPerform** interface.

Definition at line **575** of file **IxCryptoAcc.h**.

Field Documentation

UINT32 IxCryptoAccPkeEauOperand::dataLen

data length in words (multiple of 32−bit)

Definition at line **579** of file **IxCryptoAcc.h**.

UINT32* IxCryptoAccPkeEauOperand::pData

Data pointer which contains EAU input operand or output result.

Definition at line **577** of file **IxCryptoAcc.h**.

The documentation for this struct was generated from the following file:

- **IxCryptoAcc.h**

IxErrHdlAccRecoveryStatistics Struct Reference

[Intel (R) IXP400 Soft−Error Handler Driver (ixErrHdlAcc) API]

Debug Statistical Data Structure.

Data Fields

UINT32 **totalNPEAEvent**
UINT32 **totalNPEBEvent**
UINT32 **totalNPECEvent**
UINT32 **totalAQMEvent**
UINT32 **totalNPEARecoveryPass**
UINT32 **totalNPEBRecoveryPass**
UINT32 **totalNPECRecoveryPass**
UINT32 **totalAQMRecoveryPass**

Detailed Description

Debug Statistical Data Structure.

Definition at line **101** of file **IxErrHdlAcc.h**.

The documentation for this struct was generated from the following file:

- **IxErrHdlAcc.h**

IxEthAccMacAddr Struct Reference

[Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API]

Definition of the IEEE 802.3 Ethernet MAC address structure.

Data Fields

UINT8 **macAddress** [IX_IEEE803_MAC_ADDRESS_SIZE]
MAC address.

Detailed Description

Definition of the IEEE 802.3 Ethernet MAC address structure.

The data should be packed with bytes xx:xx:xx:xx:xx:xx

Note:
The data must be packed in network byte order.

Definition at line **110** of file **IxEthAcc.h**.

Field Documentation

UINT8 IxEthAccMacAddr::macAddress[IX_IEEE803_MAC_ADDRESS_SIZE]

MAC address.

Definition at line **112** of file **IxEthAcc.h**.

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

IxEthAccNe Struct Reference

[Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API, Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API, Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API]

Definition of service−specific informations.

Data Fields

UINT32 **ixReserved_next**
reserved for chaining

UINT32 **ixReserved_lengths**
reserved for buffer lengths

UINT32 **ixReserved_data**
reserved for buffer pointer

UINT8 **ixDestinationPortId**
Destination portId for this packet, if known by NPE.

UINT8 **ixSourcePortId**
Source portId for this packet.

UINT16 **ixFlags**
BitField of option for this frame.

UINT8 **ixQoS**
QoS class of the frame.

UINT8 **ixPadLength**
Size of pad field in bytes (min 0 − max 16).

UINT16 **ixVlanTCI**
Vlan TCI.

UINT8 **ixDestMac** [IX_IEEE803_MAC_ADDRESS_SIZE]
Destination MAC address.

UINT8 **ixSourceMac** [IX_IEEE803_MAC_ADDRESS_SIZE]
Source MAC address.

Detailed Description

Definition of service−specific informations.

This structure defines the Ethernet service−specific informations and enable QoS and VLAN features.

Definition at line **181** of file **IxEthAcc.h**.

Field Documentation

UINT8 IxEthAccNe::ixDestinationPortId

Destination portId for this packet, if known by NPE.

Definition at line **186** of file **IxEthAcc.h**.

UINT8 IxEthAccNe::ixDestMac[IX_IEEE803_MAC_ADDRESS_SIZE]

Destination MAC address.

Definition at line **192** of file **IxEthAcc.h**.

UINT16 IxEthAccNe::ixFlags

BitField of option for this frame.

Definition at line **188** of file **IxEthAcc.h**.

UINT8 IxEthAccNe::ixPadLength

Size of pad field in bytes (min 0 − max 16).

Definition at line **190** of file **IxEthAcc.h**.

UINT8 IxEthAccNe::ixQoS

QoS class of the frame.

Definition at line **189** of file **IxEthAcc.h**.

UINT32 IxEthAccNe::ixReserved_data

reserved for buffer pointer

Definition at line **185** of file **IxEthAcc.h**.

UINT32 IxEthAccNe::ixReserved_lengths

reserved for buffer lengths

Definition at line **184** of file **IxEthAcc.h**.

UINT32 IxEthAccNe::ixReserved_next

reserved for chaining

Definition at line **183** of file **IxEthAcc.h**.

UINT8 IxEthAccNe::ixSourceMac[IX_IEEE803_MAC_ADDRESS_SIZE]

Source MAC address.

Definition at line **193** of file **IxEthAcc.h**.

UINT8 IxEthAccNe::ixSourcePortId

Source portId for this packet.

Definition at line **187** of file **IxEthAcc.h**.

UINT16 IxEthAccNe::ixVlanTCI

Vlan TCI.

Definition at line **191** of file **IxEthAcc.h**.

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

IxEthAccPortInfo Struct Reference

[Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API, Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API]

a local copy of port information.

Detailed Description

a local copy of port information.

This structure pointer points to the port mapping information from IxEthNpePortMap[index].

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

IxEthDBPortDefinition Struct Reference

[Intel (R) IXP400 Software Ethernet Database Port Definitions (IxEthDBPortDefs)]

Port Definition − a structure contains the Port type.

Data Fields

IxEthDBPortType type

Detailed Description

Port Definition − a structure contains the Port type.

Definition at line **34** of file **IxEthDBPortDefs.h**.

The documentation for this struct was generated from the following file:

- **IxEthDBPortDefs.h**

IxEthDBWiFiRecData Struct Reference

[Intel (R) IXP400 Software Ethernet Database (IxEthDB) API]

The user wi−fi input parameters structure.

Data Fields

IxEthDBWiFiRecordType	**recType**
IxEthDBWiFiVlanTag	**vlanTagFlag**
UINT32	**logicalPortID**
UINT8	**padLength**
UINT8	**gatewayMacAddr** [IX_IEEE803_MAC_ADDRESS_SIZE]
UINT8	**bssid** [IX_IEEE803_MAC_ADDRESS_SIZE]

Detailed Description

The user wi−fi input parameters structure.

Note:

The data must be packed in network byte order.

Definition at line **275** of file **IxEthDB.h**.

The documentation for this struct was generated from the following file:

- **IxEthDB.h**

IxEthEthObjStats Struct Reference

[Intel (R) IXP400 Software Ethernet Access (IxEthAcc) API]

This struct defines the statistics returned by this component.

Data Fields

UINT32 **dot3StatsAlignmentErrors**
link error count (rx)

UINT32 **dot3StatsFCSErrors**
link error count (rx)

UINT32 **dot3StatsInternalMacReceiveErrors**
link error count (rx)

UINT32 **RxOverrunDiscards**
NPE: discarded frames count (rx).

UINT32 **RxLearnedEntryDiscards**
NPE: discarded frames count(rx).

UINT32 **RxLargeFramesDiscards**
NPE: discarded frames count(rx).

UINT32 **RxSTPBlockedDiscards**
NPE: discarded frames count(rx).

UINT32 **RxVLANTypeFilterDiscards**
NPE: discarded frames count (rx).

UINT32 **RxVLANIdFilterDiscards**
NPE: discarded frames count (rx).

UINT32 **RxInvalidSourceDiscards**
NPE: discarded frames count (rx).

UINT32 **RxBlackListDiscards**
NPE: discarded frames count (rx).

UINT32 **RxWhiteListDiscards**
NPE: discarded frames count (rx).

UINT32 **RxUnderflowEntryDiscards**
NPE: discarded frames count (rx).

UINT32 **dot3StatsSingleCollisionFrames**

link error count (tx)

UINT32 **dot3StatsMultipleCollisionFrames**
link error count (tx)

UINT32 **dot3StatsDeferredTransmissions**
link error count (tx)

UINT32 **dot3StatsLateCollisions**
link error count (tx)

UINT32 **dot3StatsExcessiveCollsions**
link error count (tx)

UINT32 **dot3StatsInternalMacTransmitErrors**
link error count (tx)

UINT32 **dot3StatsCarrierSenseErrors**
link error count (tx)

UINT32 **TxLargeFrameDiscards**
NPE: discarded frames count (tx).

UINT32 **TxVLANIdFilterDiscards**
NPE: discarded frames count (tx).

UINT32 **TxUnderrunDiscards**
NPE: discarded frames count (tx).

UINT32 **MacRecoveryTriggered**
MAC recovery count.

UINT32 **RxValidFramesTotalOctets**
UINT32 **RxUcastPkts**
UINT32 **RxBcastPkts**
UINT32 **RxMcastPkts**
UINT32 **RxPkts64Octets**
UINT32 **RxPkts65to127Octets**
UINT32 **RxPkts128to255Octets**
UINT32 **RxPkts256to511Octets**
UINT32 **RxPkts512to1023Octets**
UINT32 **RxPkts1024to1518Octets**
UINT32 **RxInternalNPEReceiveErrors**
UINT32 **TxInternalNPETransmitErrors**

Detailed Description

This struct defines the statistics returned by this component.

The component returns MIB2 EthObj variables which are obtained from the hardware or maintained by this component.

Definition at line **2505** of file **IxEthAcc.h**.

Field Documentation

UINT32 IxEthEthObjStats::dot3StatsAlignmentErrors

link error count (rx)

Definition at line **2510** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsCarrierSenseErrors

link error count (tx)

Definition at line **2529** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsDeferredTransmissions

link error count (tx)

Definition at line **2525** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsExcessiveCollsions

link error count (tx)

Definition at line **2527** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsFCSErrors

link error count (rx)

Definition at line **2511** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsInternalMacReceiveErrors

link error count (rx)

Definition at line **2512** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsInternalMacTransmitErrors

link error count (tx)

Definition at line **2528** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsLateCollisions

link error count (tx)

Definition at line **2526** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsMultipleCollisionFrames

link error count (tx)

Definition at line **2524** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsSingleCollisionFrames

link error count (tx)

Definition at line **2523** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::MacRecoveryTriggered

MAC recovery count.

Definition at line **2533** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxBlackListDiscards

NPE: discarded frames count (rx).

Definition at line **2520** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxInvalidSourceDiscards

NPE: discarded frames count (rx).

Definition at line **2519** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxLargeFramesDiscards

NPE: discarded frames count(rx).

Definition at line **2515** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxLearnedEntryDiscards

NPE: discarded frames count(rx).

Definition at line **2514** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxOverrunDiscards

NPE: discarded frames count (rx).

Definition at line **2513** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxSTPBlockedDiscards

NPE: discarded frames count(rx).

Definition at line **2516** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxUnderflowEntryDiscards

NPE: discarded frames count (rx).

Definition at line **2522** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxVLANIdFilterDiscards

NPE: discarded frames count (rx).

Definition at line **2518** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxVLANTypeFilterDiscards

NPE: discarded frames count (rx).

Definition at line **2517** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::RxWhiteListDiscards

NPE: discarded frames count (rx).

Definition at line **2521** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::TxLargeFrameDiscards

NPE: discarded frames count (tx).

Definition at line **2530** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::TxUnderrunDiscards

NPE: discarded frames count (tx).

Definition at line **2532** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::TxVLANIdFilterDiscards

NPE: discarded frames count (tx).

Definition at line **2531** of file **IxEthAcc.h**.

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

IxHssAccCodeletStats Struct Reference

ingroup IxHssAccCodeletCom

Data Fields

GeneralStats	**gen**
ChannelisedStats	**chan**
PacketisedStats	**pkt** [IX_HSSACC_HDLC_PORT_MAX]

Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for HSS Access Codelet statistics

Definition at line **145** of file **IxHssAccCodeletCom.h**.

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

IxHssAccConfigParams Struct Reference

[Intel (R) IXP400 Software HSS Access (IxHssAcc) API]

Structure containing HSS configuration parameters.

Data Fields

IxHssAccPortConfig txPortConfig
HSS Tx port configuration.

IxHssAccPortConfig rxPortConfig
HSS Rx port configuration.

unsigned **numChannelised**
The number of channelised timeslots (0−32).

unsigned **hssPktChannelCount**
The number of packetised clients (0 − 4).

UINT8 **channelisedIdlePattern**
The byte to be transmitted on channelised service when there is no client data to Tx.

BOOL **loopback**
The HSS loopback state.

unsigned **packetizedIdlePattern**
The data to be transmitted on packetised service when there is no client data to Tx.

IxHssAccClkSpeed clkSpeed
The HSS clock speed.

Detailed Description

Structure containing HSS configuration parameters.

Definition at line **563** of file **IxHssAcc.h**.

Field Documentation

UINT8 IxHssAccConfigParams::channelisedIdlePattern

The byte to be transmitted on channelised service when there is no client data to Tx.

Definition at line **571** of file **IxHssAcc.h**.

IxHssAccClkSpeed IxHssAccConfigParams::clkSpeed

The HSS clock speed.

Definition at line **578** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::hssPktChannelCount

The number of packetised clients (0 − 4).

Definition at line **569** of file **IxHssAcc.h**.

BOOL IxHssAccConfigParams::loopback

The HSS loopback state.

Definition at line **574** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::numChannelised

The number of channelised timeslots (0−32).

Definition at line **567** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::packetizedIdlePattern

The data to be transmitted on packetised service when there is no client data to Tx.

Definition at line **575** of file **IxHssAcc.h**.

IxHssAccPortConfig IxHssAccConfigParams::rxPortConfig

HSS Rx port configuration.

Definition at line **566** of file **IxHssAcc.h**.

IxHssAccPortConfig IxHssAccConfigParams::txPortConfig

HSS Tx port configuration.

Definition at line **565** of file **IxHssAcc.h**.

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

IxHssAccHdlcMode Struct Reference

[Intel (R) IXP400 Software HSS Access (IxHssAcc) API]

This structure contains 56Kbps, HDLC−mode configuration parameters.

Data Fields

BOOL **hdlc56kMode**
56kbps(TRUE)/64kbps(FALSE) HDLC

IxHssAcc56kEndianness **hdlc56kEndian**
56kbps data endianness

- *ignored if hdlc56kMode is FALSE*

BOOL **hdlc56kUnusedBitPolarity0**
The polarity '0'(TRUE)/'1'(FALSE) of the unused bit while in 56kbps mode

- *ignored if hdlc56kMode is FALSE.*

Detailed Description

This structure contains 56Kbps, HDLC−mode configuration parameters.

Definition at line **585** of file **IxHssAcc.h**.

Field Documentation

IxHssAcc56kEndianness IxHssAccHdlcMode::hdlc56kEndian

56kbps data endianness

- ignored if hdlc56kMode is FALSE

Definition at line **588** of file **IxHssAcc.h**.

BOOL IxHssAccHdlcMode::hdlc56kMode

56kbps(TRUE)/64kbps(FALSE) HDLC

Definition at line **587** of file **IxHssAcc.h**.

BOOL IxHssAccHdlcMode::hdlc56kUnusedBitPolarity0

The polarity '0'(TRUE)/'1'(FALSE) of the unused bit while in 56kbps mode

- ignored if hdlc56kMode is FALSE.

Definition at line **590** of file **IxHssAcc.h**.

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

IxHssAccPktHdlcFraming Struct Reference

[Intel (R) IXP400 Software HSS Access (IxHssAcc) API]

This structure contains information required by the NPE to configure the HDLC co−processor.

Data Fields

IxHssAccPktHdlcIdleType hdlcIdleType
What to transmit when a HDLC port is idle.

IxHssAccBitEndian dataEndian
The HDLC data endianness.

IxHssAccPktCrcType crcType
The CRC type to be used for this HDLC port.

Detailed Description

This structure contains information required by the NPE to configure the HDLC co−processor.

Definition at line **600** of file **IxHssAcc.h**.

Field Documentation

IxHssAccPktCrcType IxHssAccPktHdlcFraming::crcType

The CRC type to be used for this HDLC port.

Definition at line **604** of file **IxHssAcc.h**.

IxHssAccBitEndian IxHssAccPktHdlcFraming::dataEndian

The HDLC data endianness.

Definition at line **603** of file **IxHssAcc.h**.

IxHssAccPktHdlcIdleType IxHssAccPktHdlcFraming::hdlcIdleType

What to transmit when a HDLC port is idle.

Definition at line **602** of file **IxHssAcc.h**.

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

IxHssAccPortConfig Struct Reference

[Intel (R) IXP400 Software HSS Access (IxHssAcc) API]

Structure containing HSS port configuration parameters.

Data Fields

IxHssAccFrmSyncType frmSyncType
frame sync pulse type (Tx/Rx)

IxHssAccFrmSyncEnable frmSyncIO
how the frame sync pulse is used (Tx/Rx)

IxHssAccClkEdge frmSyncClkEdge
frame sync clock edge type (Tx/Rx)

IxHssAccClkEdge dataClkEdge
data clock edge type (Tx/Rx)

IxHssAccClkDir clkDirection
clock direction (Tx/Rx)

IxHssAccFrmPulseUsage frmPulseUsage
whether to use the frame sync pulse or not (Tx/Rx)

IxHssAccDataRate dataRate
data rate in relation to the clock (Tx/Rx)

IxHssAccDataPolarity dataPolarity
data polarity type (Tx/Rx)

IxHssAccBitEndian dataEndianness
data endianness (Tx/Rx)

IxHssAccDrainMode drainMode
Tx pin open drain mode (Tx).

IxHssAccSOFType fBitUsage
start of frame types (Tx/Rx)

IxHssAccDataEnable dataEnable
whether or not to drive the data pins (Tx)

IxHssAccTxSigType voice56kType
how to drive the data pins for voice56k type (Tx)

IxHssAccTxSigType unassignedType

how to drive the data pins for unassigned type (Tx)

IxHssAccFbType fBitType
how to drive the Fbit (Tx)

IxHssAcc56kEndianness voice56kEndian
56k data endianness when using the 56k type (Tx)

IxHssAcc56kSel voice56kSel
56k data transmission type when using the 56k type (Tx)

unsigned **frmOffset**
frame pulse offset in bits wrt the first timeslot (0−1023) (Tx/Rx)

unsigned **maxFrmSize**
frame size in bits (1−1024) (Tx/Rx)

Detailed Description

Structure containing HSS port configuration parameters.

Note: All of these are used for Tx. Only some are specific to Rx.

Definition at line **525** of file **IxHssAcc.h**.

Field Documentation

IxHssAccClkDir IxHssAccPortConfig::clkDirection

clock direction (Tx/Rx)

Definition at line **533** of file **IxHssAcc.h**.

IxHssAccClkEdge IxHssAccPortConfig::dataClkEdge

data clock edge type (Tx/Rx)

Definition at line **532** of file **IxHssAcc.h**.

IxHssAccDataEnable IxHssAccPortConfig::dataEnable

whether or not to drive the data pins (Tx)

Definition at line **542** of file **IxHssAcc.h**.

IxHssAccBitEndian IxHssAccPortConfig::dataEndianness

data endianness (Tx/Rx)

Definition at line **539** of file **IxHssAcc.h**.

IxHssAccDataPolarity IxHssAccPortConfig::dataPolarity

data polarity type (Tx/Rx)

Definition at line **538** of file **IxHssAcc.h**.

IxHssAccDataRate IxHssAccPortConfig::dataRate

data rate in relation to the clock (Tx/Rx)

Definition at line **536** of file **IxHssAcc.h**.

IxHssAccDrainMode IxHssAccPortConfig::drainMode

Tx pin open drain mode (Tx).

Definition at line **540** of file **IxHssAcc.h**.

IxHssAccFbType IxHssAccPortConfig::fBitType

how to drive the Fbit (Tx)

Definition at line **548** of file **IxHssAcc.h**.

IxHssAccSOFType IxHssAccPortConfig::fBitUsage

start of frame types (Tx/Rx)

Definition at line **541** of file **IxHssAcc.h**.

unsigned IxHssAccPortConfig::frmOffset

frame pulse offset in bits wrt the first timeslot (0−1023) (Tx/Rx)

Definition at line **553** of file **IxHssAcc.h**.

IxHssAccFrmPulseUsage IxHssAccPortConfig::frmPulseUsage

whether to use the frame sync pulse or not (Tx/Rx)

Definition at line **534** of file **IxHssAcc.h**.

IxHssAccClkEdge IxHssAccPortConfig::frmSyncClkEdge

frame sync clock edge type (Tx/Rx)

Definition at line **530** of file **IxHssAcc.h**.

IxHssAccFrmSyncEnable IxHssAccPortConfig::frmSyncIO

how the frame sync pulse is used (Tx/Rx)

Definition at line **528** of file **IxHssAcc.h**.

IxHssAccFrmSyncType IxHssAccPortConfig::frmSyncType

frame sync pulse type (Tx/Rx)

Definition at line **527** of file **IxHssAcc.h**.

unsigned IxHssAccPortConfig::maxFrmSize

frame size in bits (1−1024) (Tx/Rx)

Definition at line **555** of file **IxHssAcc.h**.

IxHssAccTxSigType IxHssAccPortConfig::unassignedType

how to drive the data pins for unassigned type (Tx)

Definition at line **546** of file **IxHssAcc.h**.

IxHssAcc56kEndianness IxHssAccPortConfig::voice56kEndian

56k data endianness when using the 56k type (Tx)

Definition at line **549** of file **IxHssAcc.h**.

IxHssAcc56kSel IxHssAccPortConfig::voice56kSel

56k data transmission type when using the 56k type (Tx)

Definition at line **551** of file **IxHssAcc.h**.

IxHssAccTxSigType IxHssAccPortConfig::voice56kType

how to drive the data pins for voice56k type (Tx)

Definition at line **544** of file **IxHssAcc.h**.

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

IxI2cInitVars Struct Reference

[Intel (R) IXP400 Software I2C Driver(IxI2cDrv) API]

contains all the variables required to initialize the I2C unit

Data Fields

IxI2cSpeedMode I2cSpeedSelect
Select either normal (100kbps) or fast mode (400kbps).

IxI2cFlowMode I2cFlowSelect
Select interrupt or poll mode.

IxI2cMasterReadCallbackP MasterReadCBP
The master read callback pointer.

IxI2cMasterWriteCallbackP MasterWriteCBP
The master write callback pointer.

IxI2cSlaveReadCallbackP SlaveReadCBP
The slave read callback pointer.

IxI2cSlaveWriteCallbackP SlaveWriteCBP
The slave write callback pointer.

IxI2cGenCallCallbackP GenCallCBP
The general call callback pointer.

BOOL **I2cGenCallResponseEnable**
Enable/disable the unit to respond to generall calls.

BOOL **I2cSlaveAddrResponseEnable**
Enable/disable the unit to respond to the slave address set in ISAR.

BOOL **SCLEnable**
Enable/disable the unit from driving the SCL line during master mode operation.

UINT8 **I2cHWAddr**
The address the unit will response to as a slave device.

Detailed Description

contains all the variables required to initialize the I2C unit

Structure to be filled and used for calling initialization

Definition at line **253** of file **IxI2cDrv.h**.

Field Documentation

IxI2cGenCallCallbackP IxI2cInitVars::GenCallCBP

The general call callback pointer.

Definition at line **267** of file **IxI2cDrv.h**.

IxI2cFlowMode IxI2cInitVars::I2cFlowSelect

Select interrupt or poll mode.

Definition at line **257** of file **IxI2cDrv.h**.

BOOL IxI2cInitVars::I2cGenCallResponseEnable

Enable/disable the unit to respond to generall calls.

Definition at line **268** of file **IxI2cDrv.h**.

UINT8 IxI2cInitVars::I2cHWAddr

The address the unit will response to as a slave device.

Definition at line **276** of file **IxI2cDrv.h**.

BOOL IxI2cInitVars::I2cSlaveAddrResponseEnable

Enable/disable the unit to respond to the slave address set in ISAR.

Definition at line **270** of file **IxI2cDrv.h**.

IxI2cSpeedMode IxI2cInitVars::I2cSpeedSelect

Select either normal (100kbps) or fast mode (400kbps).

Definition at line **255** of file **IxI2cDrv.h**.

IxI2cMasterReadCallbackP IxI2cInitVars::MasterReadCBP

The master read callback pointer.

Definition at line **259** of file **IxI2cDrv.h**.

IxI2cMasterWriteCallbackP IxI2cInitVars::MasterWriteCBP

The master write callback pointer.

Definition at line **261** of file **IxI2cDrv.h**.

BOOL IxI2cInitVars::SCLEnable

Enable/disable the unit from driving the SCL line during master mode operation.

Definition at line **273** of file **IxI2cDrv.h**.

IxI2cSlaveReadCallbackP IxI2cInitVars::SlaveReadCBP

The slave read callback pointer.

Definition at line **263** of file **IxI2cDrv.h**.

IxI2cSlaveWriteCallbackP IxI2cInitVars::SlaveWriteCBP

The slave write callback pointer.

Definition at line **265** of file **IxI2cDrv.h**.

The documentation for this struct was generated from the following file:

- **IxI2cDrv.h**

IxI2cStatsCounters Struct Reference

[Intel (R) IXP400 Software I2C Driver(IxI2cDrv) API]

contains results of counters and their overflow

Data Fields

UINT32 **ixI2cMasterXmitCounter**
Total bytes transmitted as master.

UINT32 **ixI2cMasterFailedXmitCounter**
Total bytes failed for transmission as master.

UINT32 **ixI2cMasterRcvCounter**
Total bytes received as master.

UINT32 **ixI2cMasterFailedRcvCounter**
Total bytes failed for receival as master.

UINT32 **ixI2cSlaveXmitCounter**
Total bytes transmitted as slave.

UINT32 **ixI2cSlaveFailedXmitCounter**
Total bytes failed for transmission as slave.

UINT32 **ixI2cSlaveRcvCounter**
Total bytes received as slave.

UINT32 **ixI2cSlaveFailedRcvCounter**
Total bytes failed for receival as slave.

UINT32 **ixI2cGenAddrCallSucceedCounter**
Total bytes successfully transmitted for general address.

UINT32 **ixI2cGenAddrCallFailedCounter**
Total bytes failed transmission for general address.

UINT32 **ixI2cArbLossCounter**
Total instances of arbitration loss has occured.

Detailed Description

contains results of counters and their overflow

Structure contains all values of counters and associated overflows.

Definition at line **285** of file **IxI2cDrv.h**.

Field Documentation

UINT32 IxI2cStatsCounters::ixI2cArbLossCounter

Total instances of arbitration loss has occured.

Definition at line **307** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cGenAddrCallFailedCounter

Total bytes failed transmission for general address.

Definition at line **305** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cGenAddrCallSucceedCounter

Total bytes successfully transmitted for general address.

Definition at line **303** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cMasterFailedRcvCounter

Total bytes failed for receival as master.

Definition at line **293** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cMasterFailedXmitCounter

Total bytes failed for transmission as master.

Definition at line **289** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cMasterRcvCounter

Total bytes received as master.

Definition at line **291** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cMasterXmitCounter

Total bytes transmitted as master.

Definition at line **287** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cSlaveFailedRcvCounter

Total bytes failed for receival as slave.

Definition at line **301** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cSlaveFailedXmitCounter

Total bytes failed for transmission as slave.

Definition at line **297** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cSlaveRcvCounter

Total bytes received as slave.

Definition at line **299** of file **IxI2cDrv.h**.

UINT32 IxI2cStatsCounters::ixI2cSlaveXmitCounter

Total bytes transmitted as slave.

Definition at line **295** of file **IxI2cDrv.h**.

The documentation for this struct was generated from the following file:

- **IxI2cDrv.h**

IxNpeDlImageId Struct Reference
[Intel (R) IXP Image ID Definition]

Image Id to identify each image contained in an image library.

Data Fields

IxNpeDlNpeId npeId
NPE ID.

IxNpeDlDeviceId deviceId
Device Id.

IxNpeDlFunctionalityId functionalityId
Build ID indicates functionality of image.

IxNpeDlMajor major
Major Release Number.

IxNpeDlMinor minor
Minor Revision Number.

Detailed Description

Image Id to identify each image contained in an image library.

See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **221** of file **IxNpeDl.h**.

Field Documentation

IxNpeDlDeviceId IxNpeDlImageId::deviceId

Device Id.

Definition at line **224** of file **IxNpeDl.h**.

IxNpeDlFunctionalityId IxNpeDlImageId::functionalityId

Build ID indicates functionality of image.

Definition at line **225** of file **IxNpeDl.h**.

IxNpeDlMajor IxNpeDlImageId::major

Major Release Number.

Definition at line **226** of file **IxNpeDl.h**.

IxNpeDlMinor IxNpeDlImageId::minor

Minor Revision Number.

Definition at line **227** of file **IxNpeDl.h**.

IxNpeDlNpeId IxNpeDlImageId::npeId

NPE ID.

Definition at line **223** of file **IxNpeDl.h**.

The documentation for this struct was generated from the following file:

- **IxNpeDl.h**

IxNpeMhMessage Struct Reference

[Intel (R) IXP Software NPE Message Handler (IxNpeMh) API]

The 2−word message structure to send to and receive from the NPEs.

Data Fields

UINT32 **data** [2]
the actual data of the message

Detailed Description

The 2−word message structure to send to and receive from the NPEs.

Definition at line **86** of file **IxNpeMh.h**.

Field Documentation

UINT32 IxNpeMhMessage::data[2]

the actual data of the message

Definition at line **88** of file **IxNpeMh.h**.

The documentation for this struct was generated from the following file:

- **IxNpeMh.h**

IxOamITU610Cell Struct Reference

OAM ITU610 Cell.

Data Fields

atmCellHeader	**header**
IxOamITU610Payload	**payload**

Detailed Description

OAM ITU610 Cell.

Definition at line **221** of file **IxAtmCodelet_p.h**.

The documentation for this struct was generated from the following file:

- **IxAtmCodelet_p.h**

IxOamITU610GenericPayload Struct Reference

Generic payload isn't a real payload but is used for checking which payload type a received OAM cell is.

Data Fields

UINT8 **oamTypeAndFunction**
UINT8 **reserved** [IX_OAM_ITU610_GENERIC_PAYLOAD_RESERVED_BYTES_LEN]
UINT8 **reservedAndCrc10** [IX_OAM_ITU610_RESERVED_AND_CRC10_LEN]

Detailed Description

Generic payload isn't a real payload but is used for checking which payload type a received OAM cell is.

Definition at line **201** of file **IxAtmCodelet_p.h**.

The documentation for this struct was generated from the following file:

- **IxAtmCodelet_p.h**

IxOamITU610LbPayload Struct Reference

Oam cells payload typedefs.

Data Fields

UINT8 **oamTypeAndFunction**
UINT8 **loopbackIndication**
UINT8 **correlationTag** [IX_OAM_ITU610_LB_CORRELATION_TAG_LEN]
UINT8 **llid** [IX_OAM_ITU610_LOCATION_ID_LEN]
UINT8 **sourceId** [IX_OAM_ITU610_LOCATION_ID_LEN]
UINT8 **reserved** [IX_OAM_ITU610_LB_RESERVED_BYTES_LEN]
UINT8 **reservedAndCrc10** [IX_OAM_ITU610_RESERVED_AND_CRC10_LEN]

Detailed Description

Oam cells payload typedefs.

Definition at line **186** of file **IxAtmCodelet_p.h**.

The documentation for this struct was generated from the following file:

- **IxAtmCodelet_p.h**

IxOamITU610Payload Union Reference

OAM ITU610 Payload.

Data Fields

IxOamITU610LbPayload lbPayload
IxOamITU610GenericPayload genericPayload

Detailed Description

OAM ITU610 Payload.

Definition at line **211** of file **IxAtmCodelet_p.h**.

The documentation for this union was generated from the following file:

- **IxAtmCodelet_p.h**

IxParityENAccAHBErrorTransaction Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

The Master and Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

Data Fields

IxParityENAccAHBErrorMaster ahbErrorMaster
Master on AHB bus.

IxParityENAccAHBErrorSlave ahbErrorSlave
Slave on AHB bus.

Detailed Description

The Master and Slave on the AHB bus interface whose transaction might have resulted in the parity error notification to Intel XScale(R) Processor.

NOTE: This information may be used in the data abort exception handler to differentiate between the Intel XScale(R) Processor and non−Intel XScale(R) Processor access to the SDRAM memory.

Definition at line **372** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccAHBErrorMaster IxParityENAccAHBErrorTransaction::ahbErrorMaster

Master on AHB bus.

Definition at line **374** of file **IxParityENAcc.h**.

IxParityENAccAHBErrorSlave IxParityENAccAHBErrorTransaction::ahbErrorSlave

Slave on AHB bus.

Definition at line **375** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccEbcConfig Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

Expansion Bus Controller parity detection is to be enabled or disabled.

Data Fields

IxParityENAccConfigOption ebcCs0Enabled
Expansion Bus Controller − Chip Select 0.

IxParityENAccConfigOption ebcCs1Enabled
Expansion Bus Controller − Chip Select 1.

IxParityENAccConfigOption ebcCs2Enabled
Expansion Bus Controller − Chip Select 2.

IxParityENAccConfigOption ebcCs3Enabled
Expansion Bus Controller − Chip Select 3.

IxParityENAccConfigOption ebcCs4Enabled
Expansion Bus Controller − Chip Select 4.

IxParityENAccConfigOption ebcCs5Enabled
Expansion Bus Controller − Chip Select 5.

IxParityENAccConfigOption ebcCs6Enabled
Expansion Bus Controller − Chip Select 6.

IxParityENAccConfigOption ebcCs7Enabled
Expansion Bus Controller − Chip Select 7.

IxParityENAccConfigOption ebcExtMstEnabled
External Master on Expansion bus.

IxParityENAccParityType parityOddEven
Parity − Odd or Even.

Detailed Description

Expansion Bus Controller parity detection is to be enabled or disabled.

Note: All the Chip Select(s) and External Masters will have the same parity

Definition at line **113** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs0Enabled

Expansion Bus Controller − Chip Select 0.

Definition at line **115** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs1Enabled

Expansion Bus Controller − Chip Select 1.

Definition at line **116** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs2Enabled

Expansion Bus Controller − Chip Select 2.

Definition at line **117** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs3Enabled

Expansion Bus Controller − Chip Select 3.

Definition at line **118** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs4Enabled

Expansion Bus Controller − Chip Select 4.

Definition at line **119** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs5Enabled

Expansion Bus Controller − Chip Select 5.

Definition at line **120** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs6Enabled

Expansion Bus Controller − Chip Select 6.

Definition at line **121** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcCs7Enabled

Expansion Bus Controller − Chip Select 7.

Definition at line **122** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccEbcConfig::ebcExtMstEnabled

External Master on Expansion bus.

Definition at line **123** of file **IxParityENAcc.h**.

IxParityENAccParityType IxParityENAccEbcConfig::parityOddEven

Parity − Odd or Even.

Definition at line **124** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccEbcParityErrorStats Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

Expansion Bus Controller parity error statistics.

Data Fields

UINT32 **parityErrorsInbound**
Odd bit parity errors on inbound transfers.

UINT32 **parityErrorsOutbound**
Odd bit parity errors on outbound transfers.

Detailed Description

Expansion Bus Controller parity error statistics.

Definition at line **216** of file **IxParityENAcc.h**.

Field Documentation

UINT32 IxParityENAccEbcParityErrorStats::parityErrorsInbound

Odd bit parity errors on inbound transfers.

Definition at line **218** of file **IxParityENAcc.h**.

UINT32 IxParityENAccEbcParityErrorStats::parityErrorsOutbound

Odd bit parity errors on outbound transfers.

Definition at line **219** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccHWParityConfig Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

Parity error configuration of the Hardware Blocks.

Data Fields

IxParityENAccNpeConfig npeAConfig
NPE A parity detection is to be enabled/disabled.

IxParityENAccNpeConfig npeBConfig
NPE B parity detection is to be enabled/disabled.

IxParityENAccNpeConfig npeCConfig
NPE C parity detection is to be enabled/disabled.

IxParityENAccMcuConfig mcuConfig
MCU pairty detection is to be enabled/disabled.

IxParityENAccConfigOption swcpEnabled
SWCP parity detection is to be enabled.

IxParityENAccConfigOption aqmEnabled
AQM parity detection is to be enabled.

IxParityENAccPbcConfig pbcConfig
PCI Bus Controller parity detection is to be enabled/disabled.

IxParityENAccEbcConfig ebcConfig
Expansion Bus Controller parity detection is to be enabled/disabled.

Detailed Description

Parity error configuration of the Hardware Blocks.

Definition at line **149** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccConfigOption IxParityENAccHWParityConfig::aqmEnabled

AQM parity detection is to be enabled.

Definition at line **156** of file **IxParityENAcc.h**.

IxParityENAccEbcConfig IxParityENAccHWParityConfig::ebcConfig

Expansion Bus Controller parity detection is to be enabled/disabled.

Definition at line **158** of file **IxParityENAcc.h**.

IxParityENAccMcuConfig IxParityENAccHWParityConfig::mcuConfig

MCU pairty detection is to be enabled/disabled.

Definition at line **154** of file **IxParityENAcc.h**.

IxParityENAccNpeConfig IxParityENAccHWParityConfig::npeAConfig

NPE A parity detection is to be enabled/disabled.

Definition at line **151** of file **IxParityENAcc.h**.

IxParityENAccNpeConfig IxParityENAccHWParityConfig::npeBConfig

NPE B parity detection is to be enabled/disabled.

Definition at line **152** of file **IxParityENAcc.h**.

IxParityENAccNpeConfig IxParityENAccHWParityConfig::npeCConfig

NPE C parity detection is to be enabled/disabled.

Definition at line **153** of file **IxParityENAcc.h**.

IxParityENAccPbcConfig IxParityENAccHWParityConfig::pbcConfig

PCI Bus Controller parity detection is to be enabled/disabled.

Definition at line **157** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccHWParityConfig::swcpEnabled

SWCP parity detection is to be enabled.

Definition at line **155** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccMcuConfig Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

MCU pairty detection is to be enabled/disabled.

Data Fields

IxParityENAccConfigOption singlebitDetectEnabled
Single−bit parity error detection.

IxParityENAccConfigOption singlebitCorrectionEnabled
Single−bit parity error correction.

IxParityENAccConfigOption multibitDetectionEnabled
Multi−bit parity error detection.

Detailed Description

MCU pairty detection is to be enabled/disabled.

Definition at line **96** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccConfigOption IxParityENAccMcuConfig::multibitDetectionEnabled

Multi−bit parity error detection.

Definition at line **100** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccMcuConfig::singlebitCorrectionEnabled

Single−bit parity error correction.

Definition at line **99** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccMcuConfig::singlebitDetectEnabled

Single−bit parity error detection.

Definition at line **98** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccMcuParityErrorStats Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

DDR Memory Control Unit parity error statistics.

Data Fields

UINT32 **parityErrorsSingleBit**
Parity errors of the type Single−Bit.

UINT32 **parityErrorsMultiBit**
Parity errors of the type Multi−Bit.

UINT32 **parityErrorsOverflow**
Parity errors when more than two parity errors occured.

Detailed Description

DDR Memory Control Unit parity error statistics.

Note: There could be two outstanding parity errors at any given time whose address details captured. If there is no room for the new interrupt then it would be treated as overflow parity condition.

Definition at line **187** of file **IxParityENAcc.h**.

Field Documentation

UINT32 IxParityENAccMcuParityErrorStats::parityErrorsMultiBit

Parity errors of the type Multi−Bit.

Definition at line **190** of file **IxParityENAcc.h**.

UINT32 IxParityENAccMcuParityErrorStats::parityErrorsOverflow

Parity errors when more than two parity errors occured.

Definition at line **191** of file **IxParityENAcc.h**.

UINT32 IxParityENAccMcuParityErrorStats::parityErrorsSingleBit

Parity errors of the type Single−Bit.

Definition at line **189** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccNpeConfig Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

NPE parity detection is to be enabled/disabled.

Data Fields

IxParityENAccConfigOption ideEnabled
NPE IMem, DMem and External.

IxParityENAccParityType parityOddEven
Parity − Odd or Even.

Detailed Description

NPE parity detection is to be enabled/disabled.

Definition at line **82** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccConfigOption IxParityENAccNpeConfig::ideEnabled

NPE IMem, DMem and External.

Definition at line **84** of file **IxParityENAcc.h**.

IxParityENAccParityType IxParityENAccNpeConfig::parityOddEven

Parity − Odd or Even.

Definition at line **85** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccNpeParityErrorStats Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

NPE parity error statistics.

Data Fields

UINT32 **parityErrorsIMem**
Parity errors in Instruction Memory.

UINT32 **parityErrorsDMem**
Parity errors in Data Memory.

UINT32 **parityErrorsExternal**
Parity errors in NPE External Entities.

Detailed Description

NPE parity error statistics.

Definition at line **169** of file **IxParityENAcc.h**.

Field Documentation

UINT32 IxParityENAccNpeParityErrorStats::parityErrorsDMem

Parity errors in Data Memory.

Definition at line **172** of file **IxParityENAcc.h**.

UINT32 IxParityENAccNpeParityErrorStats::parityErrorsExternal

Parity errors in NPE External Entities.

Definition at line **173** of file **IxParityENAcc.h**.

UINT32 IxParityENAccNpeParityErrorStats::parityErrorsIMem

Parity errors in Instruction Memory.

Definition at line **171** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccParityErrorContextMessage Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

Parity Error Context Message.

Data Fields

IxParityENAccParityErrorSource pecParitySource
Source info of parity error.

IxParityENAccParityErrorAccess pecAccessType
Read or Write Access Read − NPE, SWCP, AQM, DDR MCU, PCI Bus Ctrlr, Exp Bus Ctrlr (Outbound) Write − DDR MCU, PCI Bus Ctrlr, Exp Bus Ctrlr (Inbound i.e., External Master).

IxParityENAccParityErrorAddress pecAddress
Address faulty location Valid only for AQM, DDR MCU, Exp Bus Ctrlr.

IxParityENAccParityErrorData pecData
Data read from the faulty location Valid only for AQM and DDR MCU For DDR MCU it is the bit location of the Single−bit parity.

IxParityENAccParityErrorRequester pecRequester
Requester of SDRAM memory access Valid only for the DDR MCU.

IxParityENAccAHBErrorTransaction ahbErrorTran
Master and Slave information on the last AHB Error Transaction.

Detailed Description

Parity Error Context Message.

Definition at line **385** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccAHBErrorTransaction IxParityENAccParityErrorContextMessage::ahbErrorTran

Master and Slave information on the last AHB Error Transaction.

Definition at line **404** of file **IxParityENAcc.h**.

IxParityENAccParityErrorAccess IxParityENAccParityErrorContextMessage::pecAccessType

Read or Write Access Read − NPE, SWCP, AQM, DDR MCU, PCI Bus Ctrlr, Exp Bus Ctrlr (Outbound) Write − DDR MCU, PCI Bus Ctrlr, Exp Bus Ctrlr (Inbound i.e., External Master).

Definition at line **388** of file **IxParityENAcc.h**.

IxParityENAccParityErrorAddress IxParityENAccParityErrorContextMessage::pecAddress

Address faulty location Valid only for AQM, DDR MCU, Exp Bus Ctrlr.

Definition at line **395** of file **IxParityENAcc.h**.

IxParityENAccParityErrorData IxParityENAccParityErrorContextMessage::pecData

Data read from the faulty location Valid only for AQM and DDR MCU For DDR MCU it is the bit location of the Single−bit parity.

Definition at line **398** of file **IxParityENAcc.h**.

IxParityENAccParityErrorSource IxParityENAccParityErrorContextMessage::pecParitySource

Source info of parity error.

Definition at line **387** of file **IxParityENAcc.h**.

IxParityENAccParityErrorRequester IxParityENAccParityErrorContextMessage::pecRequester

Requester of SDRAM memory access Valid only for the DDR MCU.

Definition at line **402** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccParityErrorStats Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

Parity Error Statistics for the all the hardware blocks.

Data Fields

IxParityENAccNpeParityErrorStats npeStats
NPE parity error statistics.

IxParityENAccMcuParityErrorStats mcuStats
MCU parity error statistics.

IxParityENAccPbcParityErrorStats pbcStats
PBC parity error statistics.

IxParityENAccEbcParityErrorStats ebcStats
EBC parity error statistics.

UINT32 **swcpStats**
SWCP parity error statistics.

UINT32 **aqmStats**
AQM parity error statistics.

Detailed Description

Parity Error Statistics for the all the hardware blocks.

Definition at line **230** of file **IxParityENAcc.h**.

Field Documentation

UINT32 IxParityENAccParityErrorStats::aqmStats

AQM parity error statistics.

Definition at line **237** of file **IxParityENAcc.h**.

IxParityENAccEbcParityErrorStats IxParityENAccParityErrorStats::ebcStats

EBC parity error statistics.

Definition at line **235** of file **IxParityENAcc.h**.

IxParityENAccMcuParityErrorStats IxParityENAccParityErrorStats::mcuStats

MCU parity error statistics.

Definition at line **233** of file **IxParityENAcc.h**.

IxParityENAccNpeParityErrorStats IxParityENAccParityErrorStats::npeStats

NPE parity error statistics.

Definition at line **232** of file **IxParityENAcc.h**.

IxParityENAccPbcParityErrorStats IxParityENAccParityErrorStats::pbcStats

PBC parity error statistics.

Definition at line **234** of file **IxParityENAcc.h**.

UINT32 IxParityENAccParityErrorStats::swcpStats

SWCP parity error statistics.

Definition at line **236** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccPbcConfig Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

PCI Bus Controller parity detection is to be enabled or disabled.

Data Fields

IxParityENAccConfigOption pbcInitiatorEnabled
PCI Bus Controller as PCI Initiator.

IxParityENAccConfigOption pbcTargetEnabled
PCI Bus Controller as PCI Target.

Detailed Description

PCI Bus Controller parity detection is to be enabled or disabled.

Definition at line **135** of file **IxParityENAcc.h**.

Field Documentation

IxParityENAccConfigOption IxParityENAccPbcConfig::pbcInitiatorEnabled

PCI Bus Controller as PCI Initiator.

Definition at line **137** of file **IxParityENAcc.h**.

IxParityENAccConfigOption IxParityENAccPbcConfig::pbcTargetEnabled

PCI Bus Controller as PCI Target.

Definition at line **138** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxParityENAccPbcParityErrorStats Struct Reference

[Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API, Intel (R) IXP400 Software Parity Error Notifier (IxParityENAcc) API]

PCI Bus Controller parity error statistics.

Data Fields

UINT32 **parityErrorsPciInitiator**
Parity errors received as a PCI Initiator.

UINT32 **parityErrorsPciTarget**
Parity errors received as a PCI Target.

Detailed Description

PCI Bus Controller parity error statistics.

Definition at line **202** of file **IxParityENAcc.h**.

Field Documentation

UINT32 IxParityENAccPbcParityErrorStats::parityErrorsPciInitiator

Parity errors received as a PCI Initiator.

Definition at line **204** of file **IxParityENAcc.h**.

UINT32 IxParityENAccPbcParityErrorStats::parityErrorsPciTarget

Parity errors received as a PCI Target.

Definition at line **205** of file **IxParityENAcc.h**.

The documentation for this struct was generated from the following file:

- **IxParityENAcc.h**

IxSspAccStatsCounters Struct Reference

[Intel (R) IXP400 Software SSP Serial Port Access (IxSspAcc) API, Intel (R) IXP400 Software SSP Serial Port Access (IxSspAcc) API]

contains counters of the SSP statistics

Data Fields

UINT32 **ixSspRcvCounter**
Total frames received.

UINT32 **ixSspXmitCounter**
Total frames transmitted.

UINT32 **ixSspOverflowCounter**
Total occurrences of overflow.

Detailed Description

contains counters of the SSP statistics

Structure contains all values of counters and associated overflows.

Definition at line **348** of file **IxSspAcc.h**.

Field Documentation

UINT32 IxSspAccStatsCounters::ixSspOverflowCounter

Total occurrences of overflow.

Definition at line **352** of file **IxSspAcc.h**.

UINT32 IxSspAccStatsCounters::ixSspRcvCounter

Total frames received.

Definition at line **350** of file **IxSspAcc.h**.

UINT32 IxSspAccStatsCounters::ixSspXmitCounter

Total frames transmitted.

Definition at line **351** of file **IxSspAcc.h**.

The documentation for this struct was generated from the following file:

- **IxSspAcc.h**

IxSspInitVars Struct Reference

[Intel (R) IXP400 Software SSP Serial Port Access (IxSspAcc) API, Intel (R) IXP400 Software SSP Serial Port Access (IxSspAcc) API]

contains all the variables required to initialize the SSP serial port hardware.

Data Fields

IxSspAccFrameFormat FrameFormatSelected
Select between SPI, SSP and Microwire.

IxSspAccDataSize DataSizeSelected
Select between 4 and 16.

IxSspAccClkSource ClkSourceSelected
Select clock source to be on−chip or external.

IxSspAccFifoThreshold TxFIFOThresholdSelected
Select Tx FIFO threshold between 1 to 16.

IxSspAccFifoThreshold RxFIFOThresholdSelected
Select Rx FIFO threshold between 1 to 16.

BOOL **RxFIFOIntrEnable**
Enable/disable Rx FIFO threshold interrupt.

BOOL **TxFIFOIntrEnable**
Enable/disable Tx FIFO threshold interrupt.

RxFIFOThresholdHandler RxFIFOThsldHdlr
Pointer to function to handle a Rx FIFO interrupt.

TxFIFOThresholdHandler TxFIFOThsldHdlr
Pointer to function to handle a Tx FIFO interrupt.

RxFIFOOverrunHandler RxFIFOOverrunHdlr
Pointer to function to handle a Rx FIFO overrun interrupt.

BOOL **LoopbackEnable**
Select operation mode to be normal or loopback mode.

IxSspAccSpiSclkPhase SpiSclkPhaseSelected
Select SPI SCLK phase to start with one inactive cycle and end with 1/2 inactive cycle or start with 1/2 inactive cycle and end with one inactive cycle.

IxSspAccSpiSclkPolarity SpiSclkPolaritySelected
Select SPI SCLK idle state to be low or high.

IxSspAccMicrowireCtlWord MicrowireCtlWordSelected
Select Microwire control format to be 8 or 16−bit.

UINT8 **SerialClkRateSelected**
Select between 0 (1.8432Mbps) and 255 (7.2Kbps).

Detailed Description

contains all the variables required to initialize the SSP serial port hardware.

Structure to be filled and used for calling initialization

Definition at line **286** of file **IxSspAcc.h**.

Field Documentation

IxSspAccClkSource IxSspInitVars::ClkSourceSelected

Select clock source to be on−chip or external.

Definition at line **291** of file **IxSspAcc.h**.

IxSspAccDataSize IxSspInitVars::DataSizeSelected

Select between 4 and 16.

Definition at line **290** of file **IxSspAcc.h**.

IxSspAccFrameFormat IxSspInitVars::FrameFormatSelected

Select between SPI, SSP and Microwire.

Definition at line **288** of file **IxSspAcc.h**.

BOOL IxSspInitVars::LoopbackEnable

Select operation mode to be normal or loopback mode.

Definition at line **315** of file **IxSspAcc.h**.

IxSspAccMicrowireCtlWord IxSspInitVars::MicrowireCtlWordSelected

Select Microwire control format to be 8 or 16−bit.

(Only used in Microwire format).

Definition at line **330** of file **IxSspAcc.h**.

BOOL IxSspInitVars::RxFIFOIntrEnable

Enable/disable Rx FIFO threshold interrupt.

Disabling this interrupt will require the use of the polling function RxFIFOExceedThresholdCheck.

Definition at line **299** of file **IxSspAcc.h**.

RxFIFOOverrunHandler IxSspInitVars::RxFIFOOverrunHdlr

Pointer to function to handle a Rx FIFO overrun interrupt.

Definition at line **313** of file **IxSspAcc.h**.

IxSspAccFifoThreshold IxSspInitVars::RxFIFOThresholdSelected

Select Rx FIFO threshold between 1 to 16.

Definition at line **297** of file **IxSspAcc.h**.

RxFIFOThresholdHandler IxSspInitVars::RxFIFOThsldHdlr

Pointer to function to handle a Rx FIFO interrupt.

Definition at line **309** of file **IxSspAcc.h**.

UINT8 IxSspInitVars::SerialClkRateSelected

Select between 0 (1.8432Mbps) and 255 (7.2Kbps).

The formula used is Bit rate = 3.6864x10^6 / (2 x (SerialClkRateSelect + 1))

Definition at line **333** of file **IxSspAcc.h**.

IxSspAccSpiSclkPhase IxSspInitVars::SpiSclkPhaseSelected

Select SPI SCLK phase to start with one inactive cycle and end with 1/2 inactive cycle or start with 1/2 inactive cycle and end with one inactive cycle.

(Only used in SPI format).

Definition at line **318** of file **IxSspAcc.h**.

IxSspAccSpiSclkPolarity IxSspInitVars::SpiSclkPolaritySelected

Select SPI SCLK idle state to be low or high.

(Only used in SPI format).

Definition at line **326** of file **IxSspAcc.h**.

BOOL IxSspInitVars::TxFIFOIntrEnable

Enable/disable Tx FIFO threshold interrupt.

Disabling this interrupt will require the use of the polling function TxFIFOExceedThresholdCheck.

Definition at line **304** of file **IxSspAcc.h**.

IxSspAccFifoThreshold IxSspInitVars::TxFIFOThresholdSelected

Select Tx FIFO threshold between 1 to 16.

Definition at line **294** of file **IxSspAcc.h**.

TxFIFOThresholdHandler IxSspInitVars::TxFIFOThsldHdlr

Pointer to function to handle a Tx FIFO interrupt.

Definition at line **311** of file **IxSspAcc.h**.

The documentation for this struct was generated from the following file:

- **IxSspAcc.h**

IxTimeSyncAccCodeletTSChannelConfig Struct Reference

[Intel (R) IXP400 Software Time Sync Access Codelet, Intel (R) IXP400 Software Time Sync Access Codelet]

This struct is used to store all three Time Sync channels' operating mode: master or slave.

Data Fields

IxTimeSyncAcc1588PTPPortMode tsChannelMode
[IX_TIMESYNCACC_CODELET_MAX_TS_CHANNELS]

Detailed Description

This struct is used to store all three Time Sync channels' operating mode: master or slave.

Definition at line **383** of file **IxTimeSyncAccCodelet.h**.

Field Documentation

IxTimeSyncAcc1588PTPPortMode
IxTimeSyncAccCodeletTSChannelConfig::tsChannelMode[IX_TIMESYNCACC_CODELET_MAX_TS_CHANNEL

channel operating mode

Definition at line **385** of file **IxTimeSyncAccCodelet.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAccCodelet.h**

IxTimeSyncAccCodeletUninitFuncMap Struct Reference

[Intel (R) IXP400 Software Time Sync Access Codelet, Intel (R) IXP400 Software Time Sync Access Codelet]

This struct is used to store each supporting module's unload function's pointer, function parameter, and the state whether the module is initialized.

Data Fields

IxOsalVoidFnPtr **funcPtr**
IxNpeDlNpeId funcParameter
BOOL **initialized**

Detailed Description

This struct is used to store each supporting module's unload function's pointer, function parameter, and the state whether the module is initialized.

Definition at line **369** of file **IxTimeSyncAccCodelet.h**.

Field Documentation

IxNpeDlNpeId IxTimeSyncAccCodeletUninitFuncMap::funcParameter

function's input parameter

Definition at line **372** of file **IxTimeSyncAccCodelet.h**.

IxOsalVoidFnPtr IxTimeSyncAccCodeletUninitFuncMap::funcPtr

function pointer

Definition at line **371** of file **IxTimeSyncAccCodelet.h**.

BOOL IxTimeSyncAccCodeletUninitFuncMap::initialized

initialization state: TRUE − initialized, FALSE − not initialized

Definition at line **373** of file **IxTimeSyncAccCodelet.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAccCodelet.h**

IxTimeSyncAccPtpMsgData Struct Reference

[Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API]

Struct for data from the PTP message returned when TimeStamp available.

Data Fields

IxTimeSyncAcc1588PTPMsgType ptpMsgType
PTP Messages type.

IxTimeSyncAccTimeValue ptpTimeStamp
64 bit TimeStamp value from PTP Message

IxTimeSyncAccUuid ptpUuid
48 bit UUID value from the PTP Message

UINT16 **ptpSequenceNumber**
16 bit Sequence Number from PTP Message

Detailed Description

Struct for data from the PTP message returned when TimeStamp available.

Definition at line **151** of file **IxTimeSyncAcc.h**.

Field Documentation

IxTimeSyncAcc1588PTPMsgType IxTimeSyncAccPtpMsgData::ptpMsgType

PTP Messages type.

Definition at line **153** of file **IxTimeSyncAcc.h**.

UINT16 IxTimeSyncAccPtpMsgData::ptpSequenceNumber

16 bit Sequence Number from PTP Message

Definition at line **156** of file **IxTimeSyncAcc.h**.

IxTimeSyncAccTimeValue IxTimeSyncAccPtpMsgData::ptpTimeStamp

64 bit TimeStamp value from PTP Message

Definition at line **154** of file **IxTimeSyncAcc.h**.

IxTimeSyncAccUuid IxTimeSyncAccPtpMsgData::ptpUuid

48 bit UUID value from the PTP Message

Definition at line **155** of file **IxTimeSyncAcc.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAcc.h**

IxTimeSyncAccStats Struct Reference

[Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API]

Statistics for the PTP messages.

Data Fields

UINT32 **rxMsgs**
Count of timestamps for received PTP Messages.

UINT32 **txMsgs**
Count of timestamps for transmitted PTP Messages.

Detailed Description

Statistics for the PTP messages.

Definition at line **166** of file **IxTimeSyncAcc.h**.

Field Documentation

UINT32 IxTimeSyncAccStats::rxMsgs

Count of timestamps for received PTP Messages.

Definition at line **168** of file **IxTimeSyncAcc.h**.

UINT32 IxTimeSyncAccStats::txMsgs

Count of timestamps for transmitted PTP Messages.

Definition at line **169** of file **IxTimeSyncAcc.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAcc.h**

IxTimeSyncAccTimeValue Struct Reference

[Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API]

Struct to hold 64 bit SystemTime and TimeStamp values.

Data Fields

UINT32 **timeValueLowWord**
Lower 32 bits of the time value.

UINT32 **timeValueHighWord**
Upper 32 bits of the time value.

Detailed Description

Struct to hold 64 bit SystemTime and TimeStamp values.

Definition at line **125** of file **IxTimeSyncAcc.h**.

Field Documentation

UINT32 IxTimeSyncAccTimeValue::timeValueHighWord

Upper 32 bits of the time value.

Definition at line **128** of file **IxTimeSyncAcc.h**.

UINT32 IxTimeSyncAccTimeValue::timeValueLowWord

Lower 32 bits of the time value.

Definition at line **127** of file **IxTimeSyncAcc.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAcc.h**

IxTimeSyncAccUuid Struct Reference

[Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API, Intel (R) IXP400 Software Time Sync Access Component API]

Struct to hold 48 bit UUID values captured in Sync or Delay_Req messages.

Data Fields

UINT32 **uuidValueLowWord**
The lower 32 bits of the UUID.

UINT16 **uuidValueHighHalfword**
The upper 16 bits of the UUID.

Detailed Description

Struct to hold 48 bit UUID values captured in Sync or Delay_Req messages.

Definition at line **138** of file **IxTimeSyncAcc.h**.

Field Documentation

UINT16 IxTimeSyncAccUuid::uuidValueHighHalfword

The upper 16 bits of the UUID.

Definition at line **141** of file **IxTimeSyncAcc.h**.

UINT32 IxTimeSyncAccUuid::uuidValueLowWord

The lower 32 bits of the UUID.

Definition at line **140** of file **IxTimeSyncAcc.h**.

The documentation for this struct was generated from the following file:

- **IxTimeSyncAcc.h**

ixUARTDev Struct Reference

[Intel (R) IXP400 Software UART Access (IxUARTAcc) API, Intel (R) IXP400 Software UART Access (IxUARTAcc) API]

Device descriptor for the UART.

Data Fields

UINT8 * **addr**
device base address

ixUARTMode mode
interrupt, polled or loopback

int **baudRate**
baud rate

int **freq**
UART clock frequency.

int **options**
hardware options

int **fifoSize**
FIFO xmit size.

ixUARTStats stats
device statistics

Detailed Description

Device descriptor for the UART.

Definition at line **333** of file **IxUART.h**.

Field Documentation

UINT8* ixUARTDev::addr

device base address

Definition at line **335** of file **IxUART.h**.

int ixUARTDev::baudRate

baud rate

Definition at line **337** of file **IxUART.h**.

int ixUARTDev::fifoSize

FIFO xmit size.

Definition at line **340** of file **IxUART.h**.

int ixUARTDev::freq

UART clock frequency.

Definition at line **338** of file **IxUART.h**.

ixUARTMode ixUARTDev::mode

interrupt, polled or loopback

Definition at line **336** of file **IxUART.h**.

int ixUARTDev::options

hardware options

Definition at line **339** of file **IxUART.h**.

ixUARTStats ixUARTDev::stats

device statistics

Definition at line **342** of file **IxUART.h**.

The documentation for this struct was generated from the following file:

- **IxUART.h**

ixUARTStats Struct Reference

[Intel (R) IXP400 Software UART Access (IxUARTAcc) API, Intel (R) IXP400 Software UART Access (IxUARTAcc) API]

Statistics for the UART.

Data Fields

UINT32 **rxCount**
UINT32 **txCount**
UINT32 **overrunErr**
UINT32 **parityErr**
UINT32 **framingErr**
UINT32 **breakErr**

Detailed Description

Statistics for the UART.

Definition at line **319** of file **IxUART.h**.

The documentation for this struct was generated from the following file:

- **IxUART.h**

PacketisedStats Struct Reference

ingroup IxHssAccCodeletCom

Data Fields

UINT32 **txPackets**
UINT32 **txBytes**
UINT32 **txNoBuffers**
UINT32 **rxPackets**
UINT32 **rxBytes**
UINT32 **rxNoBuffers**
UINT32 **rxIdles**
UINT32 **rxVerifyFails**
UINT32 **connectFails**
UINT32 **portEnableFails**
UINT32 **txFails**
UINT32 **replenishFails**
UINT32 **portDisableFails**
UINT32 **disconnectFails**
UINT32 **txBufsInUse**
UINT32 **rxBufsInUse**
UINT32 **stopShutdownErrors**
UINT32 **hdlcAlignErrors**
UINT32 **hdlcFcsErrors**
UINT32 **rxQueueEmptyErrors**
UINT32 **hdlcMaxSizeErrors**
UINT32 **hdlcAbortErrors**
UINT32 **disconnectErrors**
UINT32 **unrecognisedErrors**

Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for Packetised statistics

Definition at line **112** of file **IxHssAccCodeletCom.h**.

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

USBDevice Struct Reference

[Intel(R) IXP400 Software USB Driver Public API]

USBDevice.

Data Fields

UINT32 **baseIOAddress**
base I/O device address

UINT32 **interruptLevel**
device IRQ

UINT32 **lastError**
detailed error of last function call

UINT32 **deviceIndex**
USB device index.

UINT32 **flags**
initialization flags

UINT8 **deviceContext** [USB_CONTEXT_SIZE]
used by the driver to identify the device

Detailed Description

USBDevice.

Definition at line **38** of file **usbtypes.h**.

Field Documentation

UINT32 USBDevice::baseIOAddress

base I/O device address

Definition at line **40** of file **usbtypes.h**.

UINT8 USBDevice::deviceContext[USB_CONTEXT_SIZE]

used by the driver to identify the device

Definition at line **45** of file **usbtypes.h**.

UINT32 USBDevice::deviceIndex

USB device index.

Definition at line **43** of file **usbtypes.h**.

UINT32 USBDevice::flags

initialization flags

Definition at line **44** of file **usbtypes.h**.

UINT32 USBDevice::interruptLevel

device IRQ

Definition at line **41** of file **usbtypes.h**.

UINT32 USBDevice::lastError

detailed error of last function call

Definition at line **42** of file **usbtypes.h**.

The documentation for this struct was generated from the following file:

- **usbtypes.h**

USBDeviceCounters Struct Reference

file **usbprivatetypes.h**

Data Fields

UINT32 **frames**
UINT32 **irqCount**
UINT32 **Rx**
UINT32 **Tx**
UINT32 **DRx**
UINT32 **DTx**
UINT32 **bytesRx**
UINT32 **bytesTx**
UINT32 **setup**

Detailed Description

file **usbprivatetypes.h**

author Intel Corporation date 30−OCT−2001

This file contains the private USB Driver data types

Definition at line **22** of file **usbprivatetypes.h**.

The documentation for this struct was generated from the following file:

- **usbprivatetypes.h**

USBSetupPacket Struct Reference

[Intel(R) IXP400 Software USB Driver Public API]

Standard USB Setup packet components, see the USB Specification 1.1.

Data Fields

UCHAR **bmRequestType**
UCHAR **bRequest**
UINT16 **wValue**
UINT16 **wIndex**
UINT16 **wLength**

Detailed Description

Standard USB Setup packet components, see the USB Specification 1.1.

Definition at line **24** of file **usbstd.h**.

The documentation for this struct was generated from the following file:

- **usbstd.h**