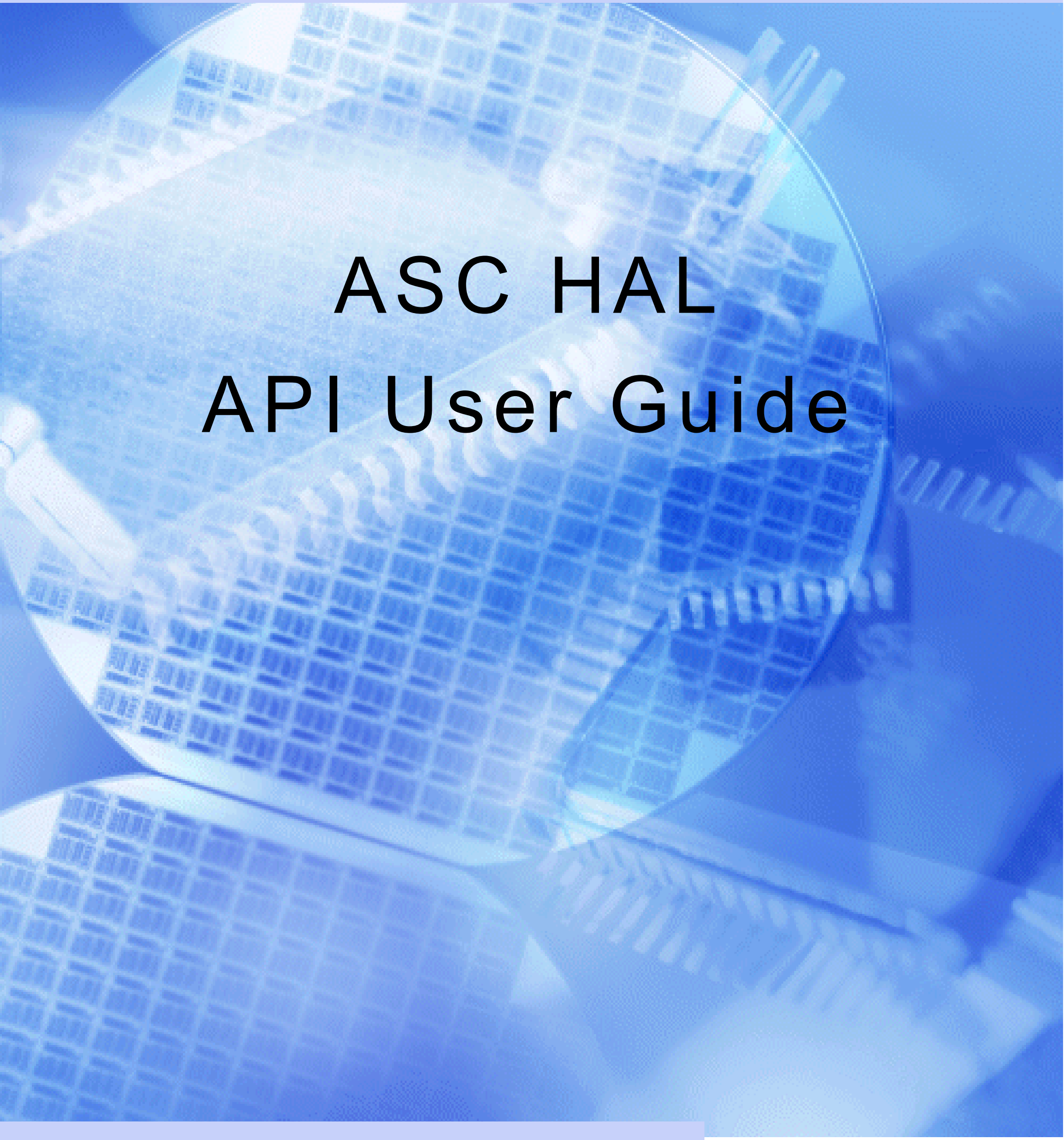
User Guide, v2.2, 17 Feb 2006



ASC HAL
API User Guide

Microcontrollers

Never stop thinking.

Edition 17 Feb 2006

Published by Infineon Technologies India Pvt Ltd,
10 Floor Discoveror Building

ITPL, Whitefield Road,
Bangalore - 560 066

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain

ASC HAL
API User Guide

Microcontrollers

Never stop thinking.

ASC API Revision History 17 Feb 2006	v2.2
Previous Version: v0.0.0	
Page	Subjects (major changes since last revision)
ALL	First release
Cover Page	Back cover page has been modified

Author:

- Mahesh S

Contributors:

- Bhavjit Walha

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing ? Your feedback will help us to continuously improve the quality of our documentation. Please send your feedback (including a reference to this document) to:

ipdoc@infineon.com

1 ASC HAL Introduction

The ASC HAL (hardware abstraction layer) forms one part out of a larger system of software modules designed to buffer application software from hardware specific implementation details. The Infineon Technologies modular HAL approach however goes beyond simply providing a standardised API for a type of device, each HAL is designed to work as part of a larger system. System resources are reserved by the HAL's using a system hardware abstraction layer meaning that runtime conflicts are avoided and different peripherals may use the same resources at different points in the application. If the peripheral clock is changeable then the HAL's will normally support on the fly clock changing allowing the clock speed to be changed for power saving etc. Data transfer requests from the application software can be queued so that the application does not need to wait for one transfer to end before the next can be started.

In cases where these features are not desirable, possibly due to runtime efficiency or code size constraints, they can simply be removed by modifying a single configuration file and recompiling the HAL, meaning there is no unnecessary code overhead.

In summary the modular HAL system provides the following advantages:

–No extra hardware specific code is required

–Pre-tested code modules are available

–Hardware can be exchanged with little or no application software modifications

–Power saving features incorporated in the HAL

–System resource conflicts are automatically avoided

–Data transfer requests can be queued.

–HAL's are highly configurable

–Porting to different hardware is easy

–Standardised API can be used for development

–Application and hardware dependant developments can go in parallel assuming a standard API

2 ASC HAL Application Program Interface

This section defines the interface between the peripheral hardware abstraction layer and the application software. It defines the constants, typedef names, function names and limitations of the HAL. The ASC HAL API utilizes a range of constants and typedef names in order to interact in a logical way with the application program. The first section of this chapter will look at these constants and data types.

Please refer to Appendix A - Infineon IFX types for details on the IFX data types.

2.1 API constants and typedefs

2.1.1ASC_API_V_MAJ macro

```
#define ASC_API_V_MAJ
```

Defined in: ASC_API.H

ASC_API_V_MAJ is defined to the major version of this API which the ASC HAL supports. This version is defined as 0.1 so the following define will be in ASC_API.h:

#define ASC_API_V_MAJ 0

Application software may check this field to determine if the HAL API version is acceptable. ASC_API_V_MAJ will be advanced whenever a change is made to this API which will result in it being incompatible with older versions, this will only be done if the API cannot be extended in a way which maintains backwards compatibility

2.1.2ASC_API_V_MIN macro

```
#define ASC_API_V_MIN
```

Defined in: ASC_API.H

ASC_API_V_MIN is defined to the minor version of this API which the ASC HAL supports. This version is defined as 0.1 so the following define will be in ASC_API.h:

#define ASC_API_V_MIN 1

Application software may check this field to determine if the HAL API version is acceptable. ASC_API_V_MIN will be advanced whenever an extension is made to this API which does not affect backwards compatibility.

2.1.3ASC_DEVICE typedef

This indicates the Device ID

Defined in: ASC_API.H

2.1.3.1Comments

ASC_DEVICE is used in the API wherever a device must be selected. This is required because many ASC peripherals may be implemented in the same system.

2.1.4ASC_STATUS

```
enum ASC_STATUS {
      ASC_SUCCESS,
      ASC_ERR,
      ASC_ERR_RES,
      ASC_ERR_RES_INT,
      ASC_ERR_RES_MEM,
      ASC_ERR_RES_IO,
      ASC_ERR_NOT_SUPPORTED,
      ASC_ERR_NOT_SUPPORTED_HW,
      ASC_ERR_UNKNOWN_DEV,
      ASC_ERR_BUSY,
      ASC_ERR_NOT_INITIALISED,
      ASC_ERR_OVR,
      ASC_ERR_PARITY,
      ASC_ERR_FRAME
};
```

Defined in: ASC_API.H

2.1.4.1Members

ASC_SUCCESS

ASC_SUCCESS indicates that an operation completed successfully.

ASC_ERR

ASC_ERR is used to indicate that an unspecified error was encountered by the HAL.ASC_ERR will only be used as a last resort when the HAL is unable to describe the error using a more specific error code.

ASC_ERR_RES

ASC_ERR_RES is used to indicate that the ASC HAL was unable to reserve a system resource required to carry out the requested operation. This will only be used when the resource is not covered by the other ASC_ERR_RES constants.

ASC_ERR_RES_INT

ASC_ERR_RES_INT is used to indicate that a required interrupt number/ priority is currently unavailable for use by the HAL. This error will be encountered either when an attempt is made to change an interrupt number/ priority during run time or when ASC_initialise_dev is called. If interrupt numbers/priorities cannot be dynamically changed due to hardware limitations then ASC_ERR_NOT_SUPPORTED_HW will be returned upon any attempt to use an incompatible number/priority.

ASC_ERR_RES_MEM

ASC_ERR_RES_MEM is used to indicate that the HAL was unable to allocate enough memory to complete the requested operation.

ASC_ERR_RES_IO

ASC_ERR_RES_IO is used to indicate that one or more physical connection lines are unavailable. This may be because a line is shared with another peripheral (and has beenreserved) or if it is currently in use as a general purpose I/O line.

ASC_ERR_NOT_SUPPORTED

ASC_ERR_NOT_SUPPORTED is used to indicate that a requested operation cannot be performed because it is not supported in software. This may be because a required software module has been compiled out

ASC_ERR_NOT_SUPPORTED_HW

ASC_ERR_NOT_SUPPORTED_HW is used to indicate that a requested operationcannot be performed because a required feature is not supported in hardware.

ASC_ERR_UNKNOWN_DEV

ASC_ERR_UNKNOWN_DEV indicates that a device ID passed to an API function was not valid.

ASC_ERR_BUSY

ASC_ERR_BUSY is returned if the ASC HAL is already busy performing an operationand the request queue is full or disabled. See Configuring the ASC HAL for information about disabling/enabling request queuing in the ASC

HAL.

ASC_ERR_NOT_INITIALISED

ASC_ERR_NOT_INITIALISED is returned if an API function is called before the HAL has been successfully initialised. This checking may be configured out to improve runtimeperformance, see Configuring the ASC HAL for information

ASC_ERR_OVR

ASC_ERR_OVR indicates that a data overrun has occurred during data reception (data has been lost because it was not retrieved from the peripheral in time).

ASC_ERR_PARITY

ASC_ERR_PARITY is used to inform the user that a parity error was detected during data reception.

ASC_ERR_FRAME

ASC_ERR_FRAME indicates that a frame error was detected during data reception.

2.1.4.2Comments

Many of the following API functions will return an ASC_STATUS data type. This is a typedef name which is defined as an enumeration, it can be found in ASC_API.h. ASC_STATUS is used by the HAL to return both the initial status of an operation and to communicate any errors encountered during data transmission back to the application via a user call back function.

2.1.5ASC_CTRL_CODE

```
enum ASC_CTRL_CODE {
      ASC_CTRL_TRNS_BAUD,
      ASC_CTRL_TRNS_DATA,
      ASC_CTRL_TRNS_STOP,
      ASC_CTRL_TRNS_PARITY,
      ASC_CTRL_TRNS_ALL,
      ASC_CTRL_IRDA_CFG,
      ASC_CTRL_FIFO_GET_RX_DEPTH,
      ASC_CTRL_FIFO_GET_TX_DEPTH,
      ASC_CTRL_FIFO_GET_RX_LEVEL,
      ASC_CTRL_FIFO_GET_TX_LEVEL,
      ASC_CTRL_FIFO_SET_RX_LEVEL,
      ASC_CTRL_FIFO_SET_TX_LEVEL,
      ASC_CTRL_FLOW,
      ASC_CTRL_BAUD_DETECT,
      ASC_CTRL_DISABLE,
      ASC_CTRL_ENABLE
};
```

Defined in: ASC_API.H

2.1.5.1Members

ASC_CTRL_TRNS_BAUD

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_TRNS_BAUD may be used during runtime to change the baud rate. Please refer to ASC_control_dev and ASC_COM_PARMS for more information.

ASC_CTRL_TRNS_DATA

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_TRNS_DATA may be used during runtime to change the number of data bits expected per data frame. Please refer to ASC_control_dev, ASC_COM_PARMS and ASC_DATA for more information.

ASC_CTRL_TRNS_STOP

This enumeration constant is used with the ASC_control_dev API function.ASC_CTRL_TRNS_STOP may be used during runtime to change the number of stop bits expected at the end of a data frame. Please refer to ASC_control_dev,ASC_COM_PARMS and ASC_STOP for more information.

ASC_CTRL_TRNS_PARITY

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_TRNS_PARITY may be used during runtime to set the required parity behavior (no parity, even parity, odd parity etc). Please refer to ASC_control_dev, ASC_COM_PARMS and ASC_PARITY for more information.

ASC_CTRL_TRNS_ALL

This enumeration constant is used with the ASC_control_dev API function.ASC_CTRL_TRNS_ALL may be used during runtime to set operating mode, parity, stop bits, data bits and baud rate in one operation. Please refer to ASC_control_dev, ASC_COM_PARMS, ASC_DATA, ASC_PARITY and ASC_STOP for more information.

ASC_CTRL_IRDA_CFG

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_IRDA_CFG may be used during runtime to configure IrDA settings. IrDA may not always be supported in hardware. Please refer to ASC_control_dev for moreinformation.

ASC_CTRL_FIFO_GET_RX_DEPTH

This enumeration constant is used with the ASC_control_dev API function to return the depth of the receive FIFO supported in hardware. Please refer to ASC_control_dev formore information.

ASC_CTRL_FIFO_GET_TX_DEPTH

This enumeration constant is used with the ASC_control_dev API function to return the depth of the transmit FIFO supported in hardware. Please refer to ASC_control_dev formore information.

ASC_CTRL_FIFO_GET_RX_LEVEL

This enumeration constant is used with the ASC_control_dev API function to return the current filling level of the receive FIFO at which an interrupt will be generated. Please refer to ASC_control_dev for more information.

ASC_CTRL_FIFO_GET_TX_LEVEL

This enumeration constant is used with the ASC_control_dev API function to return the current filling level of the transmit FIFO at which an interrupt will be generated. Please refer to ASC_control_dev for more information.

ASC_CTRL_FIFO_SET_RX_LEVEL

This enumeration constant is used with the ASC_control_dev API function to set the filling level of the receive FIFO at which an interrupt will be generated. If this value is high there will be less interrupt overhead but the risk of losing data will be greater. Please refer to ASC_control_dev for more information.

ASC_CTRL_FIFO_SET_TX_LEVEL

This enumeration constant is used with the ASC_control_dev API function to set the filling level of the transmit FIFO at which an interrupt will be generated. If this value islow there will be less interrupt overhead but the risk of losing data will be greater. Please refer to ASC_control_dev for more information.

ASC_CTRL_FLOW

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_FLOW may be used to configure hardware and software flow control. Please refer to ASC_control_dev for more information. Flow control support can be removed from the HAL, please refer to Configuring the ASC HAL for more information.

ASC_CTRL_BAUD_DETECT

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_BAUD_DETECT is used to request baud rate auto-detection in hardware, if there is no support for this then ASC_ERR_NOT_SUPPORTED_HW will be returned. Please refer to ASC_control_dev for more information.

ASC_CTRL_DISABLE

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_DISABLE may be used to disable an ASC peripheral, any IO pins previously allocated will all be set to inputs. Please refer to ASC_control_dev for moreinformation.

ASC_CTRL_ENABLE

This enumeration constant is used with the ASC_control_dev API function. ASC_CTRL_ENABLE may be used to enable an ASC peripheral, any IO pins previously allocated will all be set to inputs/outputs as required. Please refer to ASC_control_dev for more information.

2.1.5.2Comments

This is a typedef name which is defined as an enumeration. ASC_CTRL_CODE defines a number of enumeration constants which are used to request a specific operation from the ASC_control_dev API function.

2.1.6ASC_PARITY

```
enum ASC_PARITY {
      ASC_PARITY_NONE,
      ASC_PARITY_ODD,
      ASC_PARITY_EVEN,
      ASC_PARITY_STICKY_1,
      ASC_PARITY_STICKY_0
};
```

Defined in: ASC_API.H

2.1.6.1Members

ASC_PARITY_NONE

ASC_PARITY_NONE is used to specify no parity bit.

ASC_PARITY_ODD

ASC_PARITY_ODD is used to specify odd parity.

ASC_PARITY_EVEN

ASC_PARITY_EVEN is used to specify even parity.

ASC_PARITY_STICKY_1

ASC_PARITY_STICKY_1is used to specify that the parity bit is always set.

ASC_PARITY_STICKY_0

ASC_PARITY_STICKY_0 is used to specify that the parity bit is always clear.

2.1.6.2Comments

ASC_PARITY is used to specify a list of parity options supported in the API. Not all of these options will be available on all peripherals.

2.1.7ASC_MODE

```
enum ASC_MODE {
      ASC_ASYNC,
      ASC_ASYNC_MASTER,
      ASC_ASYNC_SLAVE,
      ASC_SYNC_R,
      ASC_SYNC_T,
      ASC_IRDA
};
```

Defined in: ASC_API.H

2.1.7.1Members

ASC_ASYNC
ASC_ASYNC is used to specify asynchronous mode where only two peripherals are connected.

ASC_ASYNC_MASTER
ASC_ASYNC_MASTER is used to specify asynchronous mode where many slaves exist and the peripheral controlled by the HAL is a master device.

ASC_ASYNC_SLAVE
ASC_ASYNC_SLAVE is used to specify asynchronous mode where many slaves exist and the peripheral controlled by the HAL is a slave device.

ASC_SYNC_R
ASC_SYNC_R is used to specify synchronous receive mode.

ASC_SYNC_T
ASC_SYNC_T is used to specify synchronous transmit mode.

ASC_IRDA
ASC_IRDA is used to specify IrDA mode.

2.1.7.2Comments

ASC_MODE is a typedef name which is defined as an enumeration in ASC_API.h. ASC_MODE is used to specify the operating mode of the ASC device. Not all of these options will be available on all peripherals

2.1.8 ASC_STOP

```
enum ASC_STOP {
      ASC_STOP_1_5,
      ASC_STOP_2,
      ASC_STOP_1
};
```

Defined in: ASC_API.H

2.1.8.1 Members

ASC_STOP_1_5

ASC_STOP_1_5 used to specify 1.5 stop bit, hardware does not support this mode.

ASC_STOP_2

ASC_STOP_2 used to specify 2 stop bits.

ASC_STOP_1

ASC_STOP_1 used to specify 1 stop bit.

2.1.8.2 Comments

ASC_STOP is a typedef name which is defined as an enumeration in ASC_API.h. ASC_STOP is used to specify a list of stop bit options supported in the API. Not all of these options will be available on all peripherals and some may require that other communication settings be configured in a certain way.

2.1.9 ASC_DATA

```
enum ASC_DATA {
      ASC_DATA_5,
      ASC_DATA_6,
      ASC_DATA_7,
      ASC_DATA_8,
      ASC_DATA_9
};
```

Defined in: ASC_API.H

2.1.9.1 Members

ASC_DATA_5

ASC_DATA_5 is used to specify 5 data bits per frame, hardware does not support this mode.

ASC_DATA_6
ASC_DATA_6 is used to specify 6 data bits per frame, hardware does not support this mode.

ASC_DATA_7
ASC_DATA_7 is used to specify 7 data bits per frame

ASC_DATA_8
ASC_DATA_8 is used to specify 8 data bits per frame

ASC_DATA_9
ASC_DATA_9 is used to specify 9 data bits per frame

2.1.9.2Comments

ASC_DATA is a typedef name which is defined as an enumeration in ASC_API.h.ASC_DATA is used to specify a list of data bit options supported in the API. Not all of these options will be available on all peripherals.

2.1.10ASC_COM_PARMS Structure

```
typedef struct {
      ASC_MODE ASC_mode;
      ASC_PARITY ASC_com_parity;
      ASC_STOP ASC_com_stop;
      IFX_UINT32 ASC_com_baud;
} ASC_COM_PARMS;
```

Defined in: ASC_API.H

2.1.10.1Members

ASC_mode
Mode of the module

ASC_com_parity
Parity of module

ASC_com_stop
Number of stop bits

ASC_com_baud
Baud rate of module

2.1.10.2Comments

ASC_COM_PARMS is a typedef name which is defined as a structure. The ASC_COM_PARMS structure is used to specify complete communication settings for an ASC device. It is used with the ASC_control_dev API function.

2.1.11ASC_TRANSFER Structure

```
typedef struct {
      void * ASC_buffer;
      IFX_UINT32 ASC_buffer_size;
      IFX_UINT32 ASC_return_num;
      SYS_TRANS_MODE ASC_transfer_mode;
       void(*ASC_trans_ucb)(struct ASC_transfer *, ASC_STATUS);
      IFX_UINT32 ASC_slave_device;
} ASC_TRANSFER;
```

Defined in: ASC_API.H

2.1.11.1Members

ASC_buffer

Address of the data buffer which should be pre-initialised with the data to be written.

ASC_buffer_size

The size of the data buffer, when this number of items has been sent by the peripheral the transfer is deemed complete and the application software notified.

ASC_return_num

The number of data frames read/written from/to ASC_buffer

ASC_transfer_mode

Should be set using one of the constants which are defined in the SYS_TRNS_MODE enum

void(*ASC_trans_ucb)(struct ASC_transfer *, ASC_STATUS)

Address of the user call back function to call when the transfer is complete. This may be set to 0 if no user call back function is to be invoked.

ASC_slave_device

Address of the slave to write data to if slave addressing is supported in hardware.

2.1.11.2 Comments

ASC_TRANSFER is used by the ASC_read and ASC_write functions to provide information regarding the data transfer that is to be performed.

2.1.12 ASC_FLOW_TYPE

```
enum ASC_FLOW_TYPE {
      ASC_FLOW_OFF,
      ASC_FLOW_SOFT,
      ASC_FLOW_HW
};
```

Defined in: ASC_API.H

2.1.12.1 Members

ASC_FLOW_OFF

Flow control will not be handled by either software or hardware

ASC_FLOW_SOFT

Flow control will be handled by software

ASC_FLOW_HW

Flow control will be handled by hardware

2.1.12.2 Comments

Select the type of flow control to use on a selected ASC device.

2.1.13ASC_FLOW_CTRL_SETUP Structure

```
typedef struct {
      ASC_FLOW_TYPE ASC_flow_type;
      IFX_UINT16 ASC_soft_xon;
      IFX_UINT16 ASC_soft_xoff;
} ASC_FLOW_CTRL_SETUP;
```

Defined in: ASC_API.H

2.1.13.1Members

ASC_flow_type

Set to one of the constants which are defined in ASC_FLOW_TYPE.

ASC_soft_xon

If software flow control is enabled then ASC_soft_xon should be set to the XON character

ASC_soft_xoff

If software flow control is enabled then ASC_soft_xoff should be set to the XOFF character

2.1.13.2Comments

Used to enable, disable and set up software and hardware flow control on an ASC device.

2.1.14ASC_STAT_INF Structure

```
typedef struct {
      ASC_COM_PARMS ASC_com_parms;
      IFX_UINT8 ASC_rx_fifo_lev;
      IFX_UINT8 ASC_tx_fifo_lev;
      IFX_UINT32 ASC_successful;
      IFX_UINT32 ASC_frame_errs;
      IFX_UINT32 ASC_parity_errs;
      IFX_UINT32 ASC_ovr_errs;
} ASC_STAT_INF;
```

Defined in: ASC_API.H

2.1.14.1Members

ASC_com_parms

Configuration parameters of an ASC device.

ASC_rx_fifo_lev
Users configured receive FIFO level. It will be included in statistics if hardware supports receive FIFO.

ASC_tx_fifo_lev
Users configured transmit FIFO level. It will be included in statistics if hardware supports transmit FIFO.

ASC_successful
Number of frames received successfully without errors.

ASC_frame_errs
Counter for frames received with frame error

ASC_parity_errs
Number of frames received with parity error

ASC_ovr_errs
Counter for frames received with over run error

2.1.14.2Comments

It is used by the ASC_status_dev API function to return configuration information about an ASC device, which includes the statistics, provided the value of ASC_CFG_STAT_LOG is 1.

2.2 API Functions

2.2.1ASC_initialise_dev

ASC_STATUS ASC_initialise_dev(ASC_DEVICE *ASC_device*, **ASC_COM_PARMS** * *ASC_setup***)**

ASC driver initialization function, this function initialises the internal data structures of the HAL related to the device selected by ASC_device, allocates any required system resources and configures the peripheral according to the ASC_COM_PARMS structure. The ASC_COM_PARMSstructure must be initialised by the user before calling ASC_initialise_dev. This functionmust be called successfully before any of the other API functions are used and if ASC_terminate_dev is called then ASC_initialise_dev must be called again before using the other API functions. Initialisation of one HAL should run to completion (successfully or otherwise) before the next HAL is initialised. For this reason ASC_initialise should not be called from an ISR or user callback function.

Defined in: ASC_API.H

2.2.1.1Return Value

ASC status

ASC_SUCCESS
: Initialization is success.

ASC_ERR_NOT_SUPPORTED_HW
: Require baud rate and FIFO levels are not with in the device supported limits.

ASC_ERR_NOT_SUPPORTED
: Not able to get the required baud rate below the user configured tolerance level.

2.2.1.2Parameters

ASC_device
: ASC hardware module identification number.

ASC_setup
: Driver initialization configuration parameters.

2.2.2ASC_terminate_dev

ASC_STATUS ASC_terminate_dev(ASC_DEVICE *ASC_device*)

ASC driver termination function, this function sets the peripheral, selected by the ASC_device parameter, into a disabled state and frees any system resources previously allocated in ASC_initialise. After this function has been called ASC_initialise_dev must be called successfully before any ofthe other API functions are used.
ASC_terminate_dev should not be called from an ISR or user callback function.

Defined in: ASC_API.H

2.2.2.1Return Value

ASC status

ASC_SUCCESS
Termination of device is success.

2.2.2.2Parameters

ASC_device
ASC hardware module identification number.

2.2.3ASC_abort

ASC_STATUS ASC_abort(ASC_DEVICE *ASC_device*)

ASC driver abort function cancels all currently queued data transfers and stops any transfers currently beingprocessed on the peripheral module selected by ASC_device. ASC_initialise_dev need not be called after this function before the other API functions can be used, this functionmerely clears all current and pending transfers it does not terminate the HAL. New transfers may be requested using ASC_read and/or ASC_write immediately after this function returns. All aborted transfers will return an ASC_ERR error code. This functionmay be used to clear all requests before changing modes etc.

Defined in: ASC_API.H

2.2.3.1Return Value

ASC status

ASC_SUCCESS
Abort of device is success.

2.2.3.2Parameters

ASC_device

ASC hardware module identification number.

2.2.4ASC_status_dev

ASC_STATUS ASC_status_dev(ASC_DEVICE *ASC_device*, **ASC_STAT_INF *** *ASC_stat_inf***)**

ASC driver status function, return the present driver configuration parameters and statistics information.

Defined in: ASC_API.H

2.2.4.1Parameters

ASC_device

ASC hardware module identification number.

ASC_stat_inf

Users provide data structure to write the current status of the device.

ASC_SUCCESS

Status of device success fully read and returned to application.

2.2.5ASC_control_dev

ASC_STATUS ASC_control_dev(ASC_DEVICE *ASC_device*, **ASC_CTRL_CODE** *ASC_ctrl_code*, **void *** *ASC_ctrl_arg***)**

ASC driver runtime configuration control function, ASC_control_dev may be used as a single entry point for all the control functions. The user would call the desired control function and provide new configuration parameters through ASC_CTRL_CODE and ASC_ctrl_arg parameters respectively.

Defined in: ASC_API.H

2.2.5.1Return Value

ASC status

ASC_SUCCESS

Setting the new configuration parameters is success.

ASC_ERR

The provided ASC_ctrl_code does not match with any of the values defined in ASC_CTRL_CODE.

2.2.5.2Parameters

ASC_device

ASC hardware module identification number.

ASC_ctrl_code

Function to call to specify the operation to perform.

ASC_ctrl_arg

New configuration parameters

2.2.6ASC_read

ASC_STATUS ASC_read(ASC_DEVICE *ASC_device*, **ASC_TRANSFER *** *ASC_transfer*)

ASC driver read function, the behavior of the ASC_read function depends upon the chosen transfer mode (SYS_TRNS_MCU_INT etc...) and whether or not a user call back function has been provided. If user call back function provided then request will be add it to the tail end of pending list and then return ASC_SUCCESS provided the number of pending read requestsare less than ASC_CFG_REQUEST_QUEUE_WR. If no user call back function was supplied then the ASC_read API function will not return until the requested transfer has completed.The data will be received in the user specified transfer mode.

Defined in: ASC_API.H

2.2.6.1Return Value

ASC status

ASC_SUCCESS

Reading data from device is success.

ASC_ERR_RES

The number of pending requests crosses the user configured

ASC_CFG_REQUEST_QUEUE_RD level or the input parameters do not match.

ASC_ERR_NOT_SUPPORTED_HW

Requested transfer mode is not supported by hardware.

2.2.6.2Parameters

ASC_device

ASC hardware module identification number.

ASC_transfer

Read request configuration parameter values.

2.2.7ASC_write

ASC_STATUS ASC_write(ASC_DEVICE *ASC_device*, **ASC_TRANSFER *** *ASC_transfer*)

ASC driver write function, the behavior of the ASC_write function depends upon the chosen transfer mode (SYS_TRNS_MCU_INT etc...) and whether or not a user call back function has been provided. If no user call back function was supplied then the ASC_write API function will not return until the requested transfer has completed. If a user call back function is provided and interrupt mode is requested, then the ASC_write function will return immediately with ASC_SUCCESS provided the number of pending write requestsare less than ASC_CFG_REQUEST_QUEUE_WR.Once the transfer has completed the user call back function will be invoked and the status of the operation passed to it as an argument.

Defined in: ASC_API.H

2.2.7.1Return Value

ASC status

ASC_SUCCESS

Writing data to device is success.

ASC_ERR_RES

The number of pending requests crosses the user configured ASC_CFG_REQUEST_QUEUE_WR level or the input parameters do not match.

ASC_ERR_NOT_SUPPORTED_HW
Requested transfer mode is not supported by hardware.

2.2.7.2Parameters

ASC_device
ASC hardware module identification number.

ASC_transfer
Write request configuration parameter values.

2.2.8ASC_ctrl_trns_baud

ASC_STATUS ASC_ctrl_trns_baud(ASC_DEVICE *ASC_device*, IFX_UINT32 *ASC_ctrl_baud*)

ASC driver runtime baud rate configuration control function, ASC_ctrl_trns_baud is used to select the baud rate for the chosen device.

Defined in: ASC_API.H

2.2.8.1Return Value

ASC status

ASC_SUCCESS
Setting the baud rate of device is success.

ASC_ERR_NOT_SUPPORTED_HW
The requested baud rate is crosses the hard ware supported limits.

ASC_ERR_NOT_SUPPORTED
Not able to get the requested baud rate with in the user specified tolerance level.

2.2.8.2Parameters

ASC_device
ASC hardware module identification number.

ASC_ctrl_baud
Application specified baud rate

2.2.9ASC_ctrl_trns_data

ASC_STATUS ASC_ctrl_trns_data(ASC_DEVICE *ASC_device*, **ASC_DATA** *ASC_ctrl_data*)

ASC driver data bits run time configuration control function. ASC_ctrl_trns_data is used to select the number of data bits per frame. Hardware supports 7, 8 and 9 data bits.

Defined in: ASC_API.H

2.2.9.1Return Value

ASC status

ASC_SUCCESS
The configuration of number of data bits is success.

ASC_ERR_NOT_SUPPORTED_HW
The present mode of device is not supporting new configuration.

2.2.9.2Parameters

ASC_device
ASC hardware module identification number.

ASC_ctrl_data
Specify the required number of data bits using one of the enumeration constants in ASC_DATA.

Implemented but not checkedComments

2.2.10ASC_ctrl_trns_stop

ASC_STATUS ASC_ctrl_trns_stop(ASC_DEVICE *ASC_device*, **ASC_STOP** *ASC_ctrl_stop*)

ASC driver stop bits run time configuration control function. ASC_ctrl_trns_stop is used to select the number of stop bits expected at the end of a frame. Hardware supports either 1 or 2 stop bits.

Defined in: ASC_API.H

2.2.10.1Return Value

ASC status

ASC_SUCCESS
: Parity bit has been set successfully.

ASC_ERR_NOT_SUPPORTED
: The present mode of device is not supporting new configuration.

ASC_ERR_NOT_SUPPORTED_HW
: Hardware is not supporting new configuration

2.2.10.2Parameters

ASC_device
: ASC hardware module identification number.

ASC_ctrl_stop
: Set to specify the required number of stop bits using one of the enumeration constants in ASC_STOP.

2.2.11ASC_ctrl_trns_parity

ASC_STATUS ASC_ctrl_trns_parity(ASC_DEVICE ***ASC_device*****, ASC_PARITY** ***ASC_ctrl_parity*****)**

ASC driver parity bit run time configuration control function. ASC_ctrl_trns_parity is used to choose the parity options for a device.

Defined in: ASC_API.H

2.2.11.1Return Value

ASC status

ASC_SUCCESS
: Changed the parity bit of device success fully.

ASC_ERR_NOT_SUPPORTED
: The present mode of device is not supporting new configuration.

ASC_ERR_NOT_SUPPORTED_HW
: Hardware is not supporting new configuration

2.2.11.2 Parameters

ASC_device

ASC hardware module identification number.

ASC_ctrl_parity

Set to the userspecified parity

2.2.12 ASC_ctrl_trns_all

ASC_STATUS ASC_ctrl_trns_all(ASC_DEVICE *ASC_device*, **ASC_COM_PARMS *** *ASC_ctrl_all*)

ASC driver run time configuration control function used to configure all the standard communication settings and allows the ASC operating mode to be changed. The argument should be treated as an ASC_COM_PARMS pointer for the purpose of this function, the ASC_COM_PARMS structure, which is pointed to, should be initialised to set the desired communication parameters.

Defined in: ASC_API.H

2.2.12.1 Return Value

ASC status

ASC_SUCCESS

Device is programmed with new configuration values successfully.

ASC_ERR_NOT_SUPPORTED

Incompatible new configuration parameters or baud rate is not supported by hardware.

ASC_ERR_NOT_SUPPORTED_HW

Hardware is not supporting new configuration

2.2.12.2 Parameters

ASC_device

ASC hardware module identification number.

ASC_ctrl_all

New configuration parameters.

2.2.13ASC_ctrl_irda_cfg

ASC_STATUS ASC_ctrl_irda_cfg(ASC_DEVICE *ASC_device*, **ASC_COM_PARMS *** *ASC_ctrl_irda***)**

ASC driver run time IrDA configuration control function, ASC_ctrl_irda_cfg may be used to configure IrDA when this support is available in hardware.ASC_baud will be used to specify the required pulse width in fixed mode (in nano seconds) or baud rate in variable pulse width (3/16 of bit time) mode.ASC_com_stop used to specify the selected mode (fixed or variable). ASC_STOP_1 used for variable pulse width and ASC_STOP_2 for fixed pulse width.ASC_com_parity used for RXD input inverted mode. ASC_PARITY_ODD or ASC_PARITY_EVEN for RXD input inverted mode and ASC_PARITY_NONE for RXD input non invert mode. Set ASC_mode to ASC_IRDA value.

Defined in: ASC_API.H

2.2.13.1Return Value

ASC status

ASC_SUCCESS
: Device is configured for IrDA successfully.

ASC_ERR
: Selected mode is not IrDA.

ASC_ERR_NOT_SUPPORTED_HW
: Selected baud rate or pulse width is not supported by h/w.

2.2.13.2Parameters

ASC_device
: ASC hardware module identification number.

ASC_ctrl_irda
: IrDA configuration parameters.

2.2.13.3Comments

Implemented but not checked.

2.2.14 ASC_ctrl_baud_detect

ASC_STATUS ASC_ctrl_baud_detect(ASC_DEVICE *ASC_device*, **IFX_UINT32** *ASC_ctrl_autobaud_hint***)**

ASC driver run time baud rate detect control function, this function is used to attempt to automatically detect the baud rate of an asynchronous serial connection.

Defined in: ASC_API.H

2.2.14.1 Return Value

ASC status

ASC_SUCCESS
Baud rate is detected and programmed successfully.

2.2.14.2 Parameters

ASC_device
ASC hardware module identification number.

ASC_ctrl_autobaud_hint
Hint baud rate.

2.2.14.3 Comments

Hardware not supported.

2.2.15 ASC_ctrl_fifo_get_rx_depth

IFX_UINT8 ASC_ctrl_fifo_get_rx_depth(ASC_DEVICE *ASC_device***)**

ASC driver run time receive FIFO depth read control function, Return the depth of the receive FIFO available on an ASC peripheral controlled by the HAL.

Defined in: ASC_API.H

2.2.15.1 Return Value

ASC status

ASC_SUCCESS
Successfully return the receive FIFO depth.

2.2.15.2Parameters

ASC_device

ASC hardware module identification number.

2.2.16ASC_ctrl_fifo_get_tx_depth

IFX_UINT8 ASC_ctrl_fifo_get_tx_depth(ASC_DEVICE *ASC_device*)

ASC driver run time transmit FIFO depth read control function, Return the depth of the transmit FIFO available on an ASC peripheral controlled by the HAL.

Defined in: ASC_API.H

2.2.16.1Return Value

ASC status

ASC_SUCCESS

Successfully return the receive FIFO depth.

2.2.16.2Parameters

ASC_device

ASC hardware module identification number.

2.2.17ASC_ctrl_fifo_get_rx_level

IFX_UINT8 ASC_ctrl_fifo_get_rx_level(ASC_DEVICE *ASC_device*)

ASC driver run time receive FIFO level read control function, Return the filling level at which the receive FIFO will generate an interrupt.

Defined in: ASC_API.H

2.2.17.1Return Value

ASC status

ASC_SUCCESS

Successfully return the receive FIFO interrupt trigger level.

2.2.17.2Parameters

ASC_device
ASC hardware module identification number.

2.2.18ASC_ctrl_fifo_get_tx_level

IFX_UINT8 ASC_ctrl_fifo_get_tx_level(ASC_DEVICE *ASC_device*)

ASC driver run time transmit FIFO level read control function, Return the filling level at which the transmit FIFO will generate an interrupt.

Defined in: ASC_API.H

2.2.18.1Return Value

ASC status

ASC_SUCCESS
Successfully return the transmit FIFO interrupt trigger level.

2.2.18.2Parameters

ASC_device
ASC hardware module identification number.

2.2.19ASC_ctrl_fifo_set_rx_level

ASC_STATUS ASC_ctrl_fifo_set_rx_level(ASC_DEVICE *ASC_device*, IFX_UINT8 *ASC_fifo_rx_lev_set*)

ASC driver run time receive FIFO level set control function, Set the filling level at which the receive FIFO will generate an interrupt.

Defined in: ASC_API.H

2.2.19.1Return Value

ASC status

ASC_SUCCESS
Successfully program the receive FIFO interrupt trigger level.

ASC_ERR_NOT_SUPPORTED_HW
The requested values is not supported by hardware or hardware does not support receive FIFO.

2.2.19.2Parameters

ASC_device
ASC hardware module identification number.

ASC_fifo_rx_lev_set
Receive FIFO interrupt trigger level

2.2.20ASC_ctrl_fifo_set_tx_level

ASC_STATUS ASC_ctrl_fifo_set_tx_level(ASC_DEVICE *ASC_device*, **IFX_UINT8** *ASC_fifo_tx_lev_set***)**

ASC driver run time transmit FIFO level set control function, Set the filling level at which the transmit FIFO will generate an interrupt.

Defined in: ASC_API.H

2.2.20.1Return Value

ASC status

ASC_SUCCESS
Successfully program the transmit FIFO interrupt trigger level.

ASC_ERR_NOT_SUPPORTED_HW
The requested values is not supported by hardware or hardware does not support transmit FIFO.

2.2.20.2Parameters

ASC_device
ASC hardware module identification number.

ASC_fifo_tx_lev_set
Transmit FIFO interrupt trigger level.

2.2.21ASC_ctrl_flow

ASC_STATUS ASC_ctrl_flow(ASC_DEVICE *ASC_device*, **ASC_FLOW_CTRL_SETUP *** *ASC_ctrl_flow_settings***)**

ASC driver run time flow control configuration function, Used to set the flow control settings of an ASC peripheral.

Defined in: ASC_API.H

2.2.21.1Return Value

ASC status

2.2.21.2Parameters

ASC_device

ASC hardware module identification number.

ASC_ctrl_flow_settings

Pointer to an ASC_FLOW_CTRL_SETUP structure.

2.2.21.3Comments

Not implemented.

2.2.22ASC_ctrl_disable

ASC_STATUS ASC_ctrl_disable(void)

ASC driver run time disable control function, ASC_ctrl_disable may be used to disable the peripheral, without terminating it. The result of calling this function is that all the GPIO pins the ASC HAL has allocated will be set to inputs and the peripheral disconnected. This allows the peripheral to be isolated from the outside world while communication parameters are changed or while GPIO configurations are switched. The behavior of this function may vary in some systems but it should always stop the peripheral sending and receiving data.

Defined in: ASC_API.H

2.2.22.1Return Value

ASC status

ASC_SUCCESS

Successfully disabled ASC module.

2.2.23ASC_ctrl_enable

ASC_STATUS ASC_ctrl_enable(ASC_DEVICE *ASC_device*)

ASC driver run time enable control function, ASC_ctrl_enable must be called after ASC_ctrl_disable before the peripheral will be able to communicate with other connected devices. The peripheral will be reconnected to the outside world and the GPIO lines set according to the configuration the peripheral has been set into. The behavior of this function may vary in some systems but it should always restore the peripheral to the state last configured successfully.

Defined in: ASC_API.H

2.2.23.1Return Value

ASC status

ASC_SUCCESS
Successfully enabled ASC module.

2.2.23.2Parameters

ASC_device
ASC hardware module identification number.

3 Configure/Optimise ASC HAL

Although the ASC HAL can be used immediately without configuring it to suit a particular application it will often be the case that some features written into the HAL will either be unnecessary; degrade performance to an unacceptable level, take up too much memory or only be required for debugging purposes. For this reason the ASC HAL has been designed in such a way that it can be easily configured to remove unused features, a number of optional features may be enabled and/or disabled through the ASC_CFG.h file:

–ASC device number checking

–Initialisation check on API calls

Additionally further options which affect the ASC HAL may be present in the System HAL. Depending on the actual system in question these, or other, options may be available:

–On the fly peripheral clock changing

–Initial interrupts numbers/priorities settings

–ASC physical interface (GPIO) configuration

The System HAL User Guide should be available from the same source as this document, please refer to it for more details on the available settings and features.

3.1 ASC driver HAL configuration parameters

3.1.1ASC_CFG_DEV_CHK macro

```
#define ASC_CFG_DEV_CHK
```

Defined in: ASC_CFG.H

The following define selects whether device ID checking will be performed in the ASC HAL API functions. Disabling this feature will result in less code being generated.

0

Disable the feature

1

Enable the feature

3.1.2ASC_CFG_INIT_CHK macro

```
#define ASC_CFG_INIT_CHK
```

Defined in: ASC_CFG.H

The following define selects whether certain ASC HAL API functions will check if the HAL has been initialised before executing. Disabling this feature will result in less code being generated.

0
: Disable the feature

1
: Enable the feature

3.1.3ASC_CFG_STAT_LOG macro

```
#define ASC_CFG_STAT_LOG
```

Defined in: ASC_CFG.H

The following define selects whether the ASC HAL should maintain counts of successfully received frames and frames with errors on each peripheral it controls. This allows application software to gauge the reliability of the connection.

Disabling this feature will result in smaller code and less data.

0
: Disable the feature

1
: Enable the feature

3.1.4ASC_CFG_PCP_SUP macro

```
#define ASC_CFG_PCP_SUP
```

Defined in: ASC_CFG.H

The following define may be used to include or exclude PCP support from the ASC HAL. Including PCP support results in larger code and more data. For systems which do not have a PCP this setting is ignored.

3.1.4.1Comments

Present version of software is not supporting this feature.

0

Disable the feature

1

Enable the feature

3.1.5ASC_CFG_DMA_SUP macro

```
#define ASC_CFG_DMA_SUP
```

Defined in: ASC_CFG.H

The following define may be used to include or exclude DMA support from the ASC HAL. Including DMA support results in larger code and more data. For systems which do not have a DMA controller this setting is ignored.

3.1.5.1Comments

Present version of software is not supporting this feature.

0

Disable the feature

1

Enable the feature

3.1.6ASC_CFG_FLOW macro

```
#define ASC_CFG_FLOW
```

Defined in: ASC_CFG.H

The following define may be used to disable or enable hardware and software flow control support in the ASC HAL. Disabling flow control support in the HAL results in smaller code and less data.

3.1.6.1Comments

Hardware does not support this feature.

0

Disable the feature

1

Enable the feature

3.2 API Function Exclusion

If certain API functions are not required then they may be removed in order to reduce code size. In the HAL distribution all the API functions will be included by default, in order to remove an API function the relevant define should located and value is changed from 1 to 0. This sections detail defines which are available to exclude API functions from the HAL. Set Macro value as one to include corresponding function code. Setting the value to zero not to include the function code

3.2.1ASC_CFG_FUNC_TERMINATE macro

```
#define ASC_CFG_FUNC_TERMINATE
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_terminate_dev API function is included.

1

Include the function code in driver code

0

Exclude the function code from driver code

3.2.2ASC_CFG_FUNC_READ macro

```
#define ASC_CFG_FUNC_READ
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_read API function is included.

1

Include the function code in driver code

0

Exclude the function code from driver code

3.2.3ASC_CFG_FUNC_WRITE macro

```
#define ASC_CFG_FUNC_WRITE
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_write API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.4ASC_CFG_FUNC_ABORT macro

```
#define ASC_CFG_FUNC_ABORT
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_abort_dev API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.5ASC_CFG_FUNC_STATUS macro

```
#define ASC_CFG_FUNC_STATUS
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_status_dev API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.6ASC_CFG_FUNC_CONTROL macro

```
#define ASC_CFG_FUNC_CONTROL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_control_dev API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.7ASC_CFG_FUNC_CTRL_BAUD macro

```
#define ASC_CFG_FUNC_CTRL_BAUD
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_trns_baud API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.8ASC_CFG_FUNC_CTRL_DATA macro

```
#define ASC_CFG_FUNC_CTRL_DATA
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_trns_data API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.9ASC_CFG_FUNC_CTRL_STOP macro

```
#define ASC_CFG_FUNC_CTRL_STOP
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_trns_stop API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.10ASC_CFG_FUNC_CTRL_PARITY macro

```
#define ASC_CFG_FUNC_CTRL_PARITY
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_trns_parity API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.11ASC_CFG_FUNC_CTRL_ALL macro

```
#define ASC_CFG_FUNC_CTRL_ALL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_trns_all API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.12ASC_CFG_FUNC_CTRL_IRDA macro

```
#define ASC_CFG_FUNC_CTRL_IRDA
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_irda_cfg API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.13ASC_CFG_FUNC_CTRL_AUTOBAUD macro

```
#define ASC_CFG_FUNC_CTRL_AUTOBAUD
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_baud_detect API function is included.

3.2.13.1Comments

Hardware is not supporting this feature.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.14ASC_CFG_FUNC_CTRL_FIFO_GET_RX_DEPTH macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_GET_RX_DEPTH
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_get_rx_depth API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.15ASC_CFG_FUNC_CTRL_FIFO_GET_TX_DEPTH macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_GET_TX_DEPTH
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_get_tx_depth API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.16ASC_CFG_FUNC_CTRL_FIFO_GET_RX_LEVEL macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_GET_RX_LEVEL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_get_rx_level API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.17ASC_CFG_FUNC_CTRL_FIFO_GET_TX_LEVEL macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_GET_TX_LEVEL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_get_tx_level API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.18ASC_CFG_FUNC_CTRL_FIFO_SET_RX_LEVEL macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_SET_RX_LEVEL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_set_rx_level API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.19ASC_CFG_FUNC_CTRL_FIFO_SET_TX_LEVEL macro

```
#define ASC_CFG_FUNC_CTRL_FIFO_SET_TX_LEVEL
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_fifo_set_tx_level API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.20ASC_CFG_FUNC_CTRL_FLOW macro

```
#define ASC_CFG_FUNC_CTRL_FLOW
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_flow API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.21ASC_CFG_FUNC_CTRL_ENABLE macro

```
#define ASC_CFG_FUNC_CTRL_ENABLE
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_enable API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.22ASC_CFG_FUNC_CTRL_DISABLE macro

```
#define ASC_CFG_FUNC_CTRL_DISABLE
```

Defined in: ASC_CFG.H

This controls whether or not the ASC_ctrl_disable API function is included.

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.23ASC_CFG_PEN macro

```
#define ASC_CFG_PEN
```

Defined in: ASC_CFG.H

The following define selects whether the parity bit should be checked when a packet is received by the ASC module.

This define is ignored when the module is operating in IrDA mode.

0
: Disable the parity error interrupt

1
: Enable the parity error interrupt

3.2.24ASC_CFG_OVR_CHK macro

```
#define ASC_CFG_OVR_CHK
```

Defined in: ASC_CFG.H

The following define selects whether the over run check bit should be checked when a packet is received by the ASC module.

0
: Disable the over run error interrupt

1
: Enable the over run error interrupt

3.2.25ASC_CFG_FRAME_CHK macro

```
#define ASC_CFG_FRAME_CHK
```

Defined in: ASC_CFG.H

The following define selects whether the frame error check bit should be checked when a packet is received by the ASC module.

0
: Disable the frame error interrupt

1
: Enable the frame error interrupt

3.2.26ASC_CFG_LPBACK macro

```
#define ASC_CFG_LPBACK
```

Defined in: ASC_CFG.H

The following define select the device to operate in loop back mode

0
: Device operates in non loop back mode

1
: Device operates in loop back mode

3.2.27ASC_CFG_REQUEST_QUEUE_WR macro

```
#define ASC_CFG_REQUEST_QUEUE_WR
```

Defined in: ASC_CFG.H

The following define selects whether read requests should be buffered by the ASC HAL. If this define is set to 0 then the application program must wait, after using ASC_read, for the transfer to complete before it may start the next read request. This setting (0) results in smaller faster code and less data will be used.

If this define is set to a non zero value then the ASC_read API function will be able to queue this number of read requests. This will result in a small amount of extra code and extra data for each request which is to be queued.

3.2.28ASC_CFG_REQUEST_QUEUE_RD macro

```
#define ASC_CFG_REQUEST_QUEUE_RD
```

Defined in: ASC_CFG.H

The following define selects whether write requests should be buffered by the ASC HAL. If this define is set to 0 then the application program must wait, after using ASC_write, for the transfer to complete before it may start the next write request. This setting (0) results in smaller, faster, code and less data will be used.

If this define is set to a non zero value then the ASC_write API function will be able to queue this number of write requests. This will result in a small amount of extra code and extra data for each request which is to be queued.

3.2.29ASC_CFG_RX_FIFO_LEVEL macro

```
#define ASC_CFG_RX_FIFO_LEVEL
```

Defined in: ASC_CFG.H

Defines a receive FIFO interrupt trigger level. A receive interrupt request (RIR) is always generated after the reception of a byte when the filling level of the receive FIFO is equal to or greater than ASC_CFG_RX_FIFO_LEVEL

3.2.29.1Comments

The value should be in range of 0 to 8.

3.2.30ASC_CFG_TX_FIFO_LEVEL macro

```
#define ASC_CFG_TX_FIFO_LEVEL
```

Defined in: ASC_CFG.H

Defines a transmit FIFO interrupt trigger level. A transmit interrupt request (TIR) is always generated after the transfer of a byte when the filling level of the transmit FIFO is equal to or lower than ASC_CFG_TX_FIFO_LEVEL.

3.2.30.1Comments

The value should be in range of 0 to 8.

3.2.31ASC_CFG_BAUD_TOL macro

```
#define ASC_CFG_BAUD_TOL
```

Defined in: ASC_CFG.H

The user defined baud rate tolerance either positive or negative. This tolerance will be calculated by using the following formulae. (calculate_baudrate - required_baudrate) / required_baudrate.

3.2.31.1Comments

All the standard baud rates can be able to set with the tolerance value of 0.0003 when ASC clock frequency at 48MHz.

3.2.32ASC_CFG_RDBUFF_SIZE macro

```
#define ASC_CFG_RDBUFF_SIZE
```

Defined in: ASC_CFG.H

The software read buffer size. Software buffer will store the received data when there are no pending read requests. The stored data will be copied into the read proceeding read requests. Hence this software will buffer try to avoid the receiving data loss.

The buffer size value should be in range of 1 to 4294967295.

3.2.33ASC_CFG_RMC_VAL macro

```
#define ASC_CFG_RMC_VAL
```

Defined in: ASC_CFG.H

This value is used to lower down the ASC clock frequency. If this value is zero then ASC module will be disabled.

The value should be in range of 1 - 255.

4 Application Examples

This section presents a number of simple example applications using the ASC HAL which cover the most commonly used ASC HAL API functions.

4.1 Initialise ASC0 module

```
/*
  This test sample will initialise ASC0 module with following
  configuration.
  Baudrate  - 38400
  Data bits - 8
  Stop bits - 1
  Parity    - None

  Connect RS232-0 of triboard to the hyper terminal of PC.
  To open hyper terminal in windos based PC
  (Start -> programs -> Accessories -> Communications -> Hyper terminal)

  After downloading image to triboard welcome message will be displayed,
  asking user to enter 20 characters on hyper terminal.

 After entering 20 characters, the entered characters will be displayed
  along with Good Bye message.
*/

/*Include the required header files*/
#include "COMPILER.h"
#include "ASC_CFG.h"
#include "ASC_IDL.h"
#include "ASC_IIL.h"
#include "SYS_CFG.h"
#include "ASC_API.h"
#include "SYS_API.h"

/*Buffers used for communication*/
unsigned char buffer_tx1[100]  =
  "\r\nWelcome to ASC LLD ...!  Please enter data [20 characters]\r\n";
unsigned char buffer_tx2[50]  = "";
unsigned char buffer_tx3[50]  = "\r\nEntered data ..\r\n";
unsigned char buffer_tx4[50]  = "\r\nGood bye ...!";

/*Used for transfer requests*/
ASC_TRANSFER transfer_tx1, transfer_tx2, transfer_tx3, transfer_tx4;
```

```
int main()
{
  ASC_COM_PARMS parms;

  SYS_clk_initialise(); /*Initialise the system clock*/

  /* Iniatialization parameters */
  parms.ASC_com_data =   ASC_DATA_8;
  parms.ASC_com_stop = ASC_STOP_1;
  parms.ASC_com_parity = ASC_PARITY_NONE;
  parms.ASC_mode = ASC_ASYNC;
  parms.ASC_com_baud = 38400;

  /* Setting the transfer ( read or write ) characteristics */
  /*Buffer address*/
  transfer_tx1.ASC_buffer = (IFX_UINT8 *) &buffer_tx1[0];
  /*Number of characters*/
  transfer_tx1.ASC_buffer_size = 62;
  /*Mode of transfer*/
  transfer_tx1.ASC_transfer_mode = SYS_TRNS_MCU_INT;
  /*Blocked transfer request*/
  transfer_tx1.ASC_trans_ucb = 0x0;

  transfer_tx2.ASC_buffer = (IFX_UINT8 *) &buffer_tx2[0];
  transfer_tx2.ASC_buffer_size = 20;
  transfer_tx2.ASC_transfer_mode = SYS_TRNS_MCU_INT;
  transfer_tx2.ASC_trans_ucb = 0x0;

  /* Initialize the device */
  if(ASC_initialise_dev(0, &parms ) != ASC_SUCCESS)
  {
    return 0;
  }

  /*Write welcome message*/
  ASC_write(0, &transfer_tx1);
  /*Waiting for user to enter 20 characters*/
  ASC_read(0, &transfer_tx2);
  transfer_tx1.ASC_buffer = (IFX_UINT8 *) &buffer_tx3[0];
  transfer_tx1.ASC_buffer_size = 19;
  /*Write 'Entered data' message*/
  ASC_write(0, &transfer_tx1);
  transfer_tx1.ASC_buffer = (IFX_UINT8 *) &buffer_tx2[0];
  transfer_tx1.ASC_buffer_size = 20;
  /*Write the characters entered by user*/
  ASC_write(0, &transfer_tx1);
  transfer_tx1.ASC_buffer = (IFX_UINT8 *) &buffer_tx4[0];
```

```
  transfer_tx1.ASC_buffer_size = 15;
  /*Write 'Good Bye' message*/
  ASC_write(0, &transfer_tx1);

  return 0;
}
```

4.2 Reading Data With A User Callback Function

```
/*Include the following header files*/
#include "COMPILER.H"
#include "ASC_CFG.H"
#include "ASC_IDL.H"
#include "ASC_IIL.H"
#include "SYS_CFG.H"
#include "ASC_API.H"
#include "ASC_INIT.H"
#include <stdio.h> /*For debugging purpose only*/
/*
  Prototype for user call back. This call back function will be
  used for non block read request.
*/
void asc_hal_ucb( ASC_TRANSFER *trans_struct, ASC_STATUS stat );
/*
  String to recive data
*/
unsigned char buffer_rx[25] = {0};
/*
  Transfer structure used for non blocked write request.
*/
ASC_TRANSFER transfer_nblckd_rx;
IFX_VUINT8 flag = 0;
int main()
{
  ASC_COM_PARMS parms;    /* Parameters for initialization. */
  IFX_UINT8 dev_id = 0;  /*ASC device identification number */

  /* Iniatialization parameters */
  parms.ASC_com_data =   ASC_DATA_8;
  parms.ASC_com_stop = ASC_STOP_1;
  parms.ASC_com_parity = ASC_PARITY_EVEN;
  parms.ASC_mode = ASC_ASYNC;
  parms.ASC_com_baud = 19200;
```

```
 /*Initialize unblocked request transfer structure*/
 transfer_nblckd_rx.ASC_buffer = (IFX_UINT32 *) &buffer_rx[0];
 transfer_nblckd_rx.ASC_buffer_size = 20;
 transfer_nblckd_rx.ASC_transfer_mode = SYS_TRNS_MCU_INT;
 transfer_nblckd_rx.ASC_trans_ucb = asc_hal_ucb;

 SYS_clk_initialise();  /*Initialise system clock, for more details refer
 Appendix B*/

 /*Initialise ASC0 module*/
 if(ASC_initialise_dev(dev_id, &parms ) != ASC_SUCCESS)
 {
  printf("error in initialising ASC module\n");
  return 0;
 }

 ASC_read(dev_id, &transfer_nblckd_rx); /*non blocked read*/

  while(!flag) /*Wait for data*/
  {
  }
  printf("received string %S\n",buffer_rx );

}/*End of main*/


/*User call back function*/
void asc_hal_ucb( ASC_TRANSFER *trans_struct, ASC_STATUS stat )
{
  flag = 1;
}
```

4.3 Port configuration for ASC

Port configurations will be defined in SYS_CFG.H file. For more details about port configurations please refer Appendix B.

Example for port configuration of transmit I/O line.

The macro will be defined as SYS_GPIO_ASC0_TX.

#define SYS_GPIO_ASC0_TX 1, 7, 1, 1, -1, -1, -1, -1

The define value will be configured as mentioned below

Column	*Control field*
1	Port number(1)
2	Control bit in port(7).
3	Direction bit(1)
4	Alternate 0 control field(1).
5	Alternate 1 control field(-1).
6	Open drain control field(-1)
7	Pull up selection control field(-1).
8	Pull up enable control field(-1).

Note:- If the value of any control field is -1, then that particular control field will be ignored.

5 Application note on the disabling of interrupts in the Interrupt Service Routine

From the hardware perspective, an Interrupt Service Routine(ISR) is entered with the interrupt system globally disabled. The Low Level Driver(LLD) does not enable global interupt in the ISR, as the LLD ISRs are kept short. Most LLD ISRs invoke a callback function that was registered by the application. If required, the application may enable global interrupts (by calling ENABLE_GLOBAL_INTERRUPT()) at the beginning of the ISR callback function.

6 Application note to use ASC LLD with DAvE generated drivers

- Replace Main.h file from the LLD release package with the DAvE generated MAIN.H file
- Update the **SYS_DAVE_GEN_SYS_CLC_FREQ** configuration parameter in SYS_CFG.H with the DAvE generated system clock value(The DAvE generated system clock can be observed in MAIN.c file).

7 Related Documentation

– Infineon Technologies HAL/Device Driver Software Suite Overview
– Ethernet HAL User Guide

8 Open Issues

- Half duplex synchronous mode is not working in non loop back mode.

9 Appendix A - Infineon IFX types

To overcome the problem of the size of data types changing between different compilers the HAL software modules use IFX types. These are defined in a file called COMPILER.H which is generated for each compiler that is supported. Table 1 presents these IFX types.

Table 1 Table of IFX Data Types

IFX_UINT8	Unsigned 8 bit integer
IFX_UINT16	Unsigned 16 bit integer
IFX_UINT32	Unsigned 32 bit integer
IFX_SINT8	Signed 8 bit integer
IFX_SINT16	Signed 16 bit integer
IFX_SINT32	Signed 32 bit integer
IFX_VUINT8	Unsigned 8 bit volatile integer
IFX_VUINT16	Unsigned 16 bit volatile integer
IFX_VUINT32	Unsigned 32 bit volatile integer
IFX_VSINT8	Signed 8 bit volatile integer
IFX_VSINT16	Signed 16 bit volatile integer
IFX_VSINT32	Signed 32 bit volatile integer
IFX_SFLOAT	Signed flaot
IFX_STINT8	Signed static 8 bit integer
IFX_STINT16	Signed static 16 bit integer
IFX_STINT32	Signed static 32 bit integer
IFX_STUINT8	Unsigned static 8 bit integer
IFX_STUINT16	Unsigned static 16 bit integer
IFX_STUINT32	Unsigned static 32 bit integer

10 Appendix B - The System HAL

This appendix presents a brief description of the ASC related settings and options available in the System HAL.

This section defines the configurable parameters of the System HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

10.1 Data Transfer Options

Depending upon the system the HAL is operating in there may be several different options available regarding data transfers. Some systems have a DMA controller available, others have a PCP, some have both of these and some have neither. The system HAL provides enumeration constants which can be used to specify the desired data transfer option, this enumeration is given the typedef name SYS_TRANS_MODE.

The data transfer option must be initialised in the ASC_TRANSFER structure passed to the ASC_read and ASC_write API functions. If the transfer operation is not available in the system then ASC_ERR_NOT_SUPPORTED_HW will be returned. **Table 2** presents the possible transfer options.

Table 2 Data Transfer Options

SYS_TRNS_DMA	Use the DMA controller to move the data
SYS_TRNS_PCP	Use the PCP to manage the transfer (requires a additional PCP ASC program module)
SYS_TRNS_MCU_INT	Use the microcontroller unit to manage the transfer using interrupts.
SYS_TRNS_MCU	Use the microcontroller unit to manage the transfer by polling the peripheral.

10.2 SYS HAL Configurable Parameters

This section defines the configurable parameters of the SYS HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

10.3System Clock Frequency

The clock must be operational before the controller can function. This clock is connected to the peripheral clock control registers, so changing the value of this clock frequency will affect all peripherals. The individual peripherals can scale down this frequency according to their requirements, for more details please refer to the corresponding user guide documents.

10.3.1SYS_CFG_USB_DEVICE_ENABLE macro

```
#define SYS_CFG_USB_DEVICE_ENABLE
```

Defined in: SYS_CFG.H

User needs to configure whether the USB device has been used.

1

Equate this macro to 1 if onchip USB device is used.

0

Equate this macro to 0 if usb device is not used (default).

10.3.2SYS_CFG_USB_ONCHIP_CLK macro

```
#define SYS_CFG_USB_ONCHIP_CLK
```

Defined in: SYS_CFG.H

User can configure the USB clock generation logic whether it internal or external. If clock is external, it will be derived from pin P4.0.

1

Equate this macro to 1 for internal clock generation

0

Equate this macro to 0 for external clock generation

10.3.3SYS_CFG_USB_CLK_DIVISOR macro

```
#define SYS_CFG_USB_CLK_DIVISOR
```

Defined in: SYS_CFG.H

User needs to configure the USB clock ratio based upon the USB clock frequency. Since clock frequency can be either 48 MHZ, 96 or 144 MHZ, the ratio can 1, 2 or 3 respectively.

10.3.4SYS_CFG_OSC_FREQ macro

```
#define SYS_CFG_OSC_FREQ
```

Defined in: SYS_CFG.H

User has to configure this with external applied frequency.

10.3.5SYS_CFG_CLK_MODE macro

```
#define SYS_CFG_CLK_MODE
```

Defined in: SYS_CFG.H

User needs to configure this macro to any one of the following clock operation mode.

0
: Direct drive (CPU clock directly derived from external applied frequency, N, P, and K values are not considered).

1
: PLL mode (N, P, K values will be considered to derive CPU clock frequency from external frequency)

2
: VCO bypass/pre-scalar mode (N value not considered to derive CPU clock from external frequency).

10.3.6SYS_CFG_FREQ_SEL macro

```
#define SYS_CFG_FREQ_SEL
```

Defined in: SYS_CFG.H

This define decide the frequency ration between CPU and system, this is independent from the clock mode selection(SYS_CFG_CLK_MODE).

0
: Ratio of fcpu/fsys is 2.

1
: Ratio of fcpu/fsys is 1 i.e. fcpu = fsys.

10.3.7SYS_CFG_KDIV macro

```
#define SYS_CFG_KDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 16, used for both PLL and VCO bypass modes.

10.3.8SYS_CFG_PDIV macro

```
#define SYS_CFG_PDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 8, used for both PLL and VCO bypass modes.

10.3.9SYS_CFG_NDIV macro

```
#define SYS_CFG_NDIV
```

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 128, used only for PLL mode.

10.3.9.1Comments

Advisable value range is 20 to 100.

10.3.10SYS_CFG_FIX_TC1130A_BUG macro

```
#define SYS_CFG_FIX_TC1130A_BUG
```

Defined in: SYS_CFG.H

User can use this definition for software workaround done for TC1130A at system driver and not at module level.

1
: Enbale software work-around for hardware bug fixes.

0
: Disbale software work-around for hardware bug fixes.

10.4Interrupt priorities configuration

The following priorities are used for interrupts. Corresponding to these priorities ISR code will be placed in Interrupt base Vector Table. The user can edit the priorities according to application requirements. These priorities will be static.

Priorities ranges from 1 to 255. Each interrupt should have a unique priority. 1 is the lowest priority and 255 is the highest priority.

The priority changes made by user will be effective only when changes are made before calling the driver initialization function.

10.4.1SYS_ASC0_RIR macro

```
#define SYS_ASC0_RIR
```

Defined in: SYS_CFG.H

Priority used for ASC0 receive interrupt.

10.4.2SYS_ASC0_TIR macro

```
#define SYS_ASC0_TIR
```

Defined in: SYS_CFG.H

Priority used for ASC0 transmit interrupt.

10.4.3SYS_ASC0_TBIR macro

```
#define SYS_ASC0_TBIR
```

Defined in: SYS_CFG.H

Priority used for ASC0 transmit buffer interrupt.

10.4.4SYS_ASC0_EIR macro

```
#define SYS_ASC0_EIR
```

Defined in: SYS_CFG.H

Priority used for ASC0 error interrupt.

10.4.5SYS_ASC1_RIR macro

```
#define SYS_ASC1_RIR
```

Defined in: SYS_CFG.H

Priority used for ASC1 receive interrupt.

10.4.6SYS_ASC1_TIR macro

```
#define SYS_ASC1_TIR
```

Defined in: SYS_CFG.H

Priority used for ASC1 transmit interrupt.

10.4.7SYS_ASC1_TBIR macro

```
#define SYS_ASC1_TBIR
```

Defined in: SYS_CFG.H

Priority used for ASC1 transmit buffer interrupt.

10.4.8SYS_ASC1_EIR macro

```
#define SYS_ASC1_EIR
```

Defined in: SYS_CFG.H

Priority used for ASC1 error interrupt.

10.4.9SYS_ASC2_RIR macro

```
#define SYS_ASC2_RIR
```

Defined in: SYS_CFG.H

Priority used for ASC2 receive interrupt.

10.4.10SYS_ASC2_TIR macro

```
#define SYS_ASC2_TIR
```

Defined in: SYS_CFG.H

Priority used for ASC2 transmit interrupt.

10.4.11SYS_ASC2_TBIR macro

```
#define SYS_ASC2_TBIR
```

Defined in: SYS_CFG.H

Priority used for ASC2 transmit buffer interrupt.

10.4.12SYS_ASC2_EIR macro

```
#define SYS_ASC2_EIR
```

Defined in: SYS_CFG.H

Priority used for ASC2 error interrupt.

10.5GPIO Port Configurable Parameters

This section defines the configurable port settings of the peripherals. These macros define following parameters:

Peripheral Module
- Name of the macro which includes the name of the peripheral and the port line (Transmit/Receive).

Port
- Port Number.

Pin
- Bit Number in the Port.

Dir
- Value of the bit in the Dir register.

Alt0
- Value of the bit in the Altsel0 register.

Alt1
- Value of the bit in the Altsel1 register.

Od
- Value of the bit in the Open Drain register.

Pullsel
- Value of the bit in the Pull up/Pull down selection register.

Pullen
- Value of the bit in the Pull up/Pull down enable register.

Note: The user may use -1, to indicate an unused (or don't care) value.

These macros should be defined has a set of values in above sequence and separated by commas (,).

E.g. #define SYS_GPIO_ASC0_TX 1, 7, 1, 1, -1, -1, -1, -1

10.5.1SYS_GPIO_ASC0_TX macro

```
#define SYS_GPIO_ASC0_TX
```

Defined in: SYS_CFG.H

Port configuration used for ASC0 transmit I/O line.

10.5.2SYS_GPIO_ASC0_RX macro

```
#define SYS_GPIO_ASC0_RX
```

Defined in: SYS_CFG.H

Port configuration used for ASC0 receive I/O line.

10.5.3SYS_GPIO_ASC1_TX macro

```
#define SYS_GPIO_ASC1_TX
```

Defined in: SYS_CFG.H

Port configuration used for ASC1 transmit I/O line.

10.5.4SYS_GPIO_ASC1_RX macro

```
#define SYS_GPIO_ASC1_RX
```

Defined in: SYS_CFG.H

Port configuration used for ASC1 receive I/O line.

10.5.5SYS_GPIO_ASC2_TX macro

```
#define SYS_GPIO_ASC2_TX
```

Defined in: SYS_CFG.H

Port configuration used for ASC2 transmit I/O line.

10.5.6SYS_GPIO_ASC2_RX macro

```
#define SYS_GPIO_ASC2_RX
```

Defined in: SYS_CFG.H

Port configuration used for ASC2 receive I/O line.

I

http://www.infineon.com

Published by Infineon Technologies AP