User Guide, v2.2, 17 Feb 2006

SSC 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

User Guide, v2.2, 17 Feb 2006

SSC HAL
API User Guide

Microcontrollers

Never stop thinking.

SSC API

Revision History 17 Feb 2006 v2.2

Previous Version: v0.0.0

Page	Subjects (major changes since last revision)
ALL	First release
Back cover	Back cover 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 SSC HAL Introduction

The SSC 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 SSC 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 SSC 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

2.1.1SSC_API_V_MAJ macro

```
#define SSC_API_V_MAJ
```

Defined in: SSC_API.H

SSC_API_V_MAJ is defined to the major version of this API which the SSC HAL supports. This version is defined as 0.1 so the following define will be in SSC_API.H:

#define SSC_API_V_MAJ 0

Application software may check this field to determine if the HAL API version is acceptable. SSC_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

2.1.2SSC_API_V_MIN macro

```
#define SSC_API_V_MIN
```

Defined in: SSC_API.H

SSC_API_V_MIN is defined to the minor version of this API which the SSC HAL supports. This version is defined as 0.1 so the following define will be in SSC_API.H:

#define SSC_API_V_MIN 1

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

2.1.3SSC_DEVICE typedef

This indicates the Device ID

```
typedef SSC_DEVICE IFX_UINT8
```

Defined in: SSC_API.H

2.1.3.1Comments

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

2.1.4SSC_STATUS

```
enum SSC_STATUS {
      SSC_SUCCESS,
      SSC_ERR,
      SSC_ERR_RES,
      SSC_ERR_RES_MEM,
      SSC_ERR_RES_IO,
      SSC_ERR_NOT_SUPPORTED,
      SSC_ERR_NOT_SUPPORTED_HW,
      SSC_ERR_UNKNOWN_DEV,
      SSC_ERR_NOT_INITIALISED,
      SSC_ERR_PHASE,
      SSC_ERR_RECEIVE,
      SSC_ERR_BAUD,
      SSC_ERR_TRANS
};
```

Defined in: SSC_API.H

2.1.4.1Members

SSC_SUCCESS

SSC_SUCCESS indicates that an operation completed successfully.

SSC_ERR

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

SSC_ERR_RES

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

SSC_ERR_RES_MEM

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

SSC_ERR_RES_IO

SSC_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 been reserved) or if it is currently in use as a general purpose I/O line.

SSC_ERR_NOT_SUPPORTED

SSC_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 configured out (see configuring the SSC HAL).

SSC_ERR_NOT_SUPPORTED_HW

SSC_ERR_NOT_SUPPORTED_HW is used to indicate that a requested operation cannot be performed because a required feature is not supported in hardware.

SSC_ERR_UNKNOWN_DEV

SSC_ERR_UNKNOWN_DEV indicates that a device ID passed to an API function was not valid. Device ID checking can be removed from the HAL to reduce code size, see configuring the SSC HAL for details.

SSC_ERR_NOT_INITIALISED

SSC_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 runtime performance and reduce code size, see configuring the SSC HAL for information.

SSC_ERR_PHASE

SSC_ERR_PHASE is returned to indicate that a phase error occurred during SSC_read or SSC_write operation.

SSC_ERR_RECEIVE

SSC_ERR_RECEIVE is returned to indicate that an error occurred during a SSC_read operation. This is usually because data was not read from the peripheral in time and was overwritten by the next incoming data word.

SSC_ERR_BAUD

SSC_ERR_BAUD is returned to indicate that a baud rate error was detected, this error will only occur when the peripheral is configured in slave mode. A baud rate error is detected when the clock signal from the master device differs substantially from that expected.

SSC_ERR_TRANS

SSC_ERR_TRANS is returned to indicate that an error occurred during a SSC_write operation. This usually means that data was not written to the peripheral in time. This error will only occur when the peripheral is configured as a slave device.

2.1.4.2Comments

Many of the following API functions will return an SSC_STATUS data type. This is a typedef name which is defined as an enumeration it can be found in SSC_API.H. SSC_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.

2.1.5SSC_CTRL_CODE

```
enum SSC_CTRL_CODE {
      SSC_CTRL_TRNS_BAUD,
      SSC_CTRL_TRNS_DATA,
      SSC_CTRL_TRNS_SHIFT,
      SSC_CTRL_TRNS_PHASE,
      SSC_CTRL_TRNS_CLOCK,
      SSC_CTRL_TX_HANDLING,
      SSC_CTRL_TRNS_ALL,
      SSC_CTRL_FIFO_GET_RX_DEPTH,
      SSC_CTRL_FIFO_GET_TX_DEPTH,
      SSC_CTRL_FIFO_GET_RX_LEVEL,
      SSC_CTRL_FIFO_GET_TX_LEVEL,
      SSC_CTRL_FIFO_SET_RX_LEVEL,
      SSC_CTRL_FIFO_SET_TX_LEVEL,
      SSC_CTRL_DISABLE,
      SSC_CTRL_ENABLE
};
```

Defined in: SSC_API.H

2.1.5.1Members

SSC_CTRL_TRNS_BAUD

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_TRNS_BAUD may be used during runtime to change the baud rate. Refer to SSC_control_dev for more information. The baud rate is set initially by passing a pointer to a SSC_COM_PARMS structure, with SSC_com_baud set appropriately, to SSC_initialise_dev.

SSC_CTRL_TRNS_DATA

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_TRNS_DATA may be used during runtime to change the number of bits per SSC data frame, this is the number of bits the SSC will shift for a single transfer. Refer to SSC_control_dev and SSC_DATA for more information. The word size is set initially by passing a pointer to a SSC_COM_PARMS structure, with SSC_com_data set appropriately, to SSC_initialise_dev.

SSC_CTRL_TRNS_SHIFT

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_TRNS_SHIFT may be used during runtime to change which bit in the data word will be shifted onto the transmit line first using one of the enumeration constants in SSC_SHIFT_DIR. Please refer to SSC_control_dev and SSC_COM_PARMS for more information. The shift direction is set initially by passing a pointer to a SSC_COM_PARMS structure, with SSC_com_shift set appropriately, to SSC_initialise_dev.

SSC_CTRL_TRNS_PHASE

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_TRNS_PHASE may be used during runtime to change the clock edge on which data will be shifted, and on which edge it will be latched, using one of the enumeration constants in SSC_PHASE. Refer to SSC_control_dev and for more information. The phase is set initially by passing a pointer to a SSC_COM_PARMS structure, with SSC_com_phase set appropriately, to SSC_initialise_dev.

SSC_CTRL_TRNS_CLOCK

This enumeration constant is used with the SSC_control_dev API function.

SSC_CTRL_TRNS_CLOCK may be used during runtime to set the idle level for the clock line using one of the enumeration constants in SSC_CLOCK_IDLE. This is the level at which the clock line will be held when no transfer is underway and the SSC is in master mode. Refer to SSC_control_dev for more information. The idle level for the clock line is set initially by passing a pointer to a SSC_COM_PARMS structure, with SSC_com_clock set appropriately, to SSC_initialise_dev.

SSC_CTRL_TX_HANDLING

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_TX_HANDLING may be used during runtime to set the scheme to be employed when a transmit line is shared. Please refer to SSC_control_dev and SSC_CFG_TX_OPTIONS for more information.

SSC_CTRL_TRNS_ALL

This enumeration constant is used with the SSC_control_dev API function.SSC_CTRL_TRNS_ALL may be used at runtime to change the baud rate, operating mode, shift direction, phase and the clock idle level in one operation. Please refer to SSC_control_dev, SSC_DATA, SSC_CLOCK_IDLE, SSC_SHIFT_DIR and SSC_PHASE for more information. These communication parameters are set initially by passing a pointer to an initialised SSC_COM_PARMS structure to SSC_initialise_dev.

SSC_CTRL_FIFO_GET_RX_DEPTH

This enumeration constant is used with the SSC_control_dev API function to return the depth of the receive FIFO supported in hardware. Refer to SSC_control_dev for more information.

SSC_CTRL_FIFO_GET_TX_DEPTH

This enumeration constant is used with the SSC_control_dev API function to return the depth of the transmit FIFO supported in hardware. Refer to SSC_control_dev for more information.

SSC_CTRL_FIFO_GET_RX_LEVEL

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

SSC_CTRL_FIFO_GET_TX_LEVEL

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

SSC_CTRL_FIFO_SET_RX_LEVEL

This enumeration constant is used with the SSC_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. Refer to SSC_control_dev for more information.

SSC_CTRL_FIFO_SET_TX_LEVEL

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

SSC_CTRL_DISABLE

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_DISABLE may be used to disable any of the SSC peripherals controlled by the HAL, any I/O pins previously allocated will all be set to inputs, it is the users responsibility to ensure that any external SSC devices are unaffected by this. Refer to SSC_control_dev for more information.

SSC_CTRL_ENABLE

This enumeration constant is used with the SSC_control_dev API function. SSC_CTRL_ENABLE may be used to enable any of the SSC peripherals controlled by the HAL which have previously been disabled using SSC_CTRL_DISABLE, any I/O pins previously allocated will all be set to inputs/outputs as relevant. Refer to SSC_control_dev for more information.

2.1.5.2Comments

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

2.1.6SSC_MODE

```
enum SSC_MODE {
      SSC_MASTER,
      SSC_SLAVE
};
```

Defined in: SSC_API.H

2.1.6.1Members

SSC_MASTER

Peripheral will generate a clock to read or write data.

SSC_SLAVE

Peripheral will wait for a clock to be generated by another master device in order to read or write data.

2.1.6.2Comments

SSC_MODE is used with SSC_ctrl_trns_all to specify the operating mode of the SSC device.

2.1.7SSC_DATA

```
typedef IFX_UINT8 SSC_DATA;
```

Defined in: SSC_API.H

2.1.7.1Comments

SSC_DATA is used with SSC_ctrl_trns_all or SSC_ctrl_trns_data to set the width of a data word (the number of bits which the SSC will try to shift for each transfer). The lower limit is defined, in this API, as 2 bits and the upper as 16 bits. typedef IFX_UINT8 SSC_DATA;

2.1.8SSC_SHIFT_DIR

```
enum SSC_SHIFT_DIR {
      SSC_MSB_FIRST,
      SSC_LSB_FIRST
};
```

Defined in: SSC_API.H

2.1.8.1 Members

SSC_MSB_FIRST

Transmit/receive most significant bit first.

SSC_LSB_FIRST

Transmit/receive least significant bit first.

2.1.8.2 Comments

SSC_SHIFT_DIR may be used with SSC_ctrl_trns_all or SSC_ctrl_trns_shift to set which bit of the data word the SSC peripheral should shift out first.

2.1.9 SSC_CLOCK_IDLE

```
enum SSC_CLOCK_IDLE {
      SSC_CLOCK_IDLE_LOW,
      SSC_CLOCK_IDLE_HIGH
};
```

Defined in: SSC_API.H

2.1.9.1 Members

SSC_CLOCK_IDLE_LOW

Idle clock line is low, leading clock edge is low-to-high transition.

SSC_CLOCK_IDLE_HIGH

Idle clock line is high, leading clock edge is high-to-low transition.

2.1.9.2 Comments

SSC_CLOCK_IDLE may be used with SSC_ctrl_trns_all or SSC_ctrl_trns_clock to set the level at which the clock line should remain when no data transfer is underway. This will be ignored if the SSC is configured as a slave device.

2.1.10 SSC_PHASE

```
enum SSC_PHASE {
      SSC_LATCH_RISING,
      SSC_LATCH_FALLING
};
```

Defined in: SSC_API.H

2.1.10.1 Members

SSC_LATCH_RISING

Shifts transmit data on the leading clock edge, latch on trailing edge.

SSC_LATCH_FALLING

Latch receive data on leading clock edge, shift on trailing edge.

2.1.10.2 Comments

SSC_PHASE may be used with SSC_ctrl_trns_all or SSC_ctrl_trns_phase to set the clock edge on which data should be shifted and on which edge it should be latched.

2.1.11 SSC_COM_PARMS Structure

```
typedef struct {
      SSC_MODE SSC_mode;
      SSC_DATA SSC_com_data;
      IFX_UINT8 SSC_slave_num;
      SSC_CLOCK_IDLE SSC_com_clock;
      SSC_PHASE SSC_com_phase;
      SSC_SHIFT_DIR SSC_com_shift;
      IFX_UINT32 SSC_com_baud;
} SSC_COM_PARMS;
```

Defined in: SSC_API.H

2.1.11.1 Members

SSC_mode

Mode of device is currently operating either in slave/master.

SSC_com_data

Number of data bits that the device expects per data frame.

SSC_slave_num

Slave select input (slave mode[1])/output (master mode[1 - 8]) number, if value is zero slave select functionality not provided

SSC_com_clock

Level of clock line will remain at idle state.

SSC_com_phase

Clock edge the SSC shifts data out and on which edge it latches data in.

SSC_com_shift

Bit of the data word SSC shifts out first.

SSC_com_baud

Baud rate of device.

2.1.11.2Comments

SSC_COM_PARMS is a typedef name which is defined as a structure, it can be found in SSC_API.H. The SSC_COM_PARMS structure is used to specify complete communication settings for the SSC devices controlled by the HAL. It is used with the SSC_ctrl_trns_all and SSC_status_dev API functions.

2.1.12SSC_OSLAVE_OPT Structure

```
typedef struct {
      IFX_UINT16 SSC_lead_dly:2;
      IFX_UINT16 SSC_trl_dly:2;
      IFX_UINT16 SSC_inact_dly:2;
      IFX_UINT16 SSC_delay:1;
      IFX_UINT16 SSC_slv_idl_lvl:1;
      IFX_UINT16 SSC_slv_num:7;
} SSC_OSLAVE_OPT;
```

Defined in: SSC_API.H

2.1.12.1Members

SSC_lead_dly:2

Number of leading delay cycles, ranges from 0 - 3.

SSC_trl_dly:2

Number of trail delay cycles, ranges from 0 - 3.

SSC_inact_dly:2

Number of inact delay cycles, ranges from 0 - 3.

SSC_delay:1

Specifies the mode of device 0 --> normal, 1 --> delay.

SSC_slv_idl_lvl:1

Defines the logic level of the slave mode transmit signal MRST when the SSC is deselected

SSC_slv_num:7

Slave select number output (master)/input (slave)

2.1.12.2Comments

This structure is used to configure the slave select output line at run time. This feature is fully hardware dependent feature.

2.1.13SSC_TRANSFER Structure

```
typedef struct {
      void * SSC_buffer;
      IFX_UINT32 SSC_buffer_size;
      IFX_UINT32 SSC_return_num;
      SYS_TRANS_MODE SSC_transfer_mode;
      void (*SSC_trans_ucb)(struct SSC_transfer *, SSC_STATUS);
      IFX_UINT32 SSC_slave_device;
} SSC_TRANSFER;
```

Defined in: SSC_API.H

2.1.13.1Members

SSC_buffer

Address of the data buffer which is to be used for transmit data or filled with data read from the selected SSC device.

SSC_buffer_size

Size of data buffer used to read or transmit.

SSC_return_num

Actual number of data frames read or transmitted.

SSC_transfer_mode

Set with one of the constants defined in the SYS_TRNS_MODE enum.

void (*SSC_trans_ucb)(struct SSC_transfer *, SSC_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.

SSC_slave_device

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

2.1.13.2Comments

SSC_TRANSFER is used by the SSC_read and SSC_write functions to provide information regarding the data transfer that is to be performed.

2.1.14SSC_STAT_INF Structure

```
typedef struct {
      SSC_COM_PARMS SSC_com_parms;
      IFX_UINT32 SSC_receive_err;
      IFX_UINT32 SSC_transmit_err;
      IFX_UINT32 SSC_successful;
      IFX_UINT8 SSC_phase_err;
      IFX_UINT8 SSC_baud_err;
      IFX_UINT8 SSC_tx_fifo_lev;
      IFX_UINT8 SSC_rx_fifo_lev;
      SSC_DATA SSC_current_progress;
} SSC_STAT_INF;
```

Defined in: SSC_API.H

2.1.14.1Members

SSC_com_parms

Current configuration of the selected SSC device.

SSC_receive_err

Current value of the internal receive error counter, after this call this counter will be reset to 0. SSC_receive_err will only be present if status logging is enabled in the SSC HAL, see configuring the SSC HAL for details.

SSC_transmit_err

Current value of the internal transmit error counter, after this call this counter will be reset to 0. SSC_transmit_err will only be present if status logging is enabled in the SSC HAL, see configuring the SSC HAL for details.

SSC_successful

Number of words sent of received without error, after this call the counter will be reset to 0. SSC_successful will only be present if status logging is enabled in the SSC HAL, see configuring the SSC HAL for details.

SSC_phase_err

Current value of the internal phase error counter, after this call this counter will be reset to 0. SSC_phase_err will only be present if status logging is enabled in the SSC HAL, see configuring the SSC HAL for details.

SSC_baud_err

Current value of the internal baud error counter, after this call this counter will be reset to 0. SSC_baud_err will only be present if status logging is enabled in the SSC HAL, see configuring the SSC HAL for details.

SSC_tx_fifo_lev

Number of words currently in the transmit FIFO or 0 if there is no transmit FIFO on the device.

SSC_rx_fifo_lev

Number of words currently in the receive FIFO or 0 if there is no receive FIFO on the device.

SSC_current_progress

Set to the number of bits that have been shifted in the current transaction.

2.1.14.2Comments

It is used by the SSC_status_dev API function to return information about any of the SSC peripherals controlled by the HAL.

2.2 API Functions

2.2.1SSC_initialise_dev

SSC_STATUS SSC_initialise_dev(SSC_DEVICE *SSC_device*, **SSC_COM_PARMS *** *SSC_setup***)**

SSC driver initialization function, this function initialises the internal data structures of the HAL related to the device selected by SSC_device, allocates any required system resources and configures the peripheral according to the SSC_COM_PARMS structure. The SSC_COM_PARMS structure must be initialised by the user before calling SSC_initialise_dev. This function must be called successfully before any of the other API functions are used and if SSC_terminate_dev is called then SSC_initialise_dev must be called again before using the other API functions.

Defined in: SSC_API.H

2.2.1.1Return Value

SSC status

SSC_SUCCESS
Initialization is successful.

SSC_ERR_NOT_SUPPORTED_HW
FIFO levels are not with in the device supported limits.

SSC_ERR_BAUA
Not able to get the required baud rate within user configured tolerance level.

SSC_ERR_RES_IO
System HAL is not able to reserve required ports required by SSC.

2.2.1.2Parameters

SSC_device
SSC hardware module identification number (e.g. 0-->SSC0, 1-->SSC1).

SSC_setup
Driver initialization configuration parameters.

2.2.2SSC_terminate_dev

SSC_STATUS SSC_terminate_dev(SSC_DEVICE *SSC_device*)

SSC driver termination function, this function sets the peripheral, selected by the SSC_device parameter, into a disabled state and frees any system resources previously allocated in SSC_initialise. After this function has been called SSC_initialise_dev must be called successfully before any of the other API functions are used.

Defined in: SSC_API.H

2.2.2.1Return Value

SSC status

SSC_SUCCESS
Termination of device is successful.

SSC_ERR_RES_IO
Error occurred while returning ports to System HAL.

2.2.2.2Parameters

SSC_device

SSC hardware module identification number.

2.2.3SSC_abort

SSC_STATUS SSC_abort(SSC_DEVICE *SSC_device***)**

SSC driver abort function cancels all currently queued data transfers and stops any transfers currently being processed on the peripheral module selected by SSC_device. SSC_initialise_dev need not be called after this function before the other API functions can be used, this function merely clears all current and pending transfers it does not terminate the HAL. New transfers may be requested using SSC_read and/or SSC_write immediately after this function returns. All aborted transfers will return an SSC_ERR error code. This function may be used to clear all requests before changing modes etc.

Defined in: SSC_API.H

2.2.3.1Return Value

SSC status

SSC_SUCCESS

Abort of device is successful.

2.2.3.2Parameters

SSC_device

SSC hardware module identification number.

2.2.4SSC_status_dev

SSC_STATUS SSC_status_dev(SSC_DEVICE *SSC_device***, SSC_STAT_INF *** *SSC_stat_inf***)**

SSC driver status function, return the present driver configuration parameters and statistics information. Configuration parameters include number of data bits per frame, phase, polarity, baud rate and FIFO levels. Statistics include number of data frames received/transmitted success fully without errors, phase errors, transfer, receive and baud rate errors. After returning from this function all statistics counters will re reset to zero.

Defined in: SSC_API.H

2.2.4.1 Return Value

SSC status

SSC_SUCCESS

Status of device success fully read and returned to application.

2.2.4.2 Parameters

SSC_device

SSC hardware module identification number.

SSC_stat_inf

Data structure to write the current status of the device.

2.2.5 SSC_control_dev

SSC_STATUS SSC_control_dev(SSC_DEVICE *SSC_device*, **SSC_CTRL_CODE** *SSC_ctrl_code*, **void** * *SSC_ctrl_arg*)

SSC driver runtime configuration control function, SSC_control_dev may be used as a single entry point for all the control functions, such as set baud rate, which are provided in the SSC HAL API.

The device to configure must be selected by using the SSC_device parameter.

The function to call is specified by the SSC_ctrl_code. One of the enumeration constants from SSC_CTRL_CODE must be used to specify the operation to perform.

The SSC_ctrl_arg parameter is used to pass arguments to the configuration function, it is a void pointer because its actual use depends upon the function requested.

The direct call equivalents are also described, it is quicker to use the function directly without going through SSC_control_dev.

Defined in: SSC_API.H

2.2.5.1 Return Value

SSC status

SSC_SUCCESS

Setting the new configuration parameters is successful.

SSC_ERR

The provided SSC_ctrl_code does not match with any of the values defined in SSC_CTRL_CODE.

2.2.5.2 Parameters

SSC_device

SSC hardware module identification number.

SSC_ctrl_code

Operation to perform is specified by one of the enum in SSC_CTRL_CODE.

SSC_ctrl_arg

New configuration parameters.

2.2.6 SSC_ctrl_trns_baud

SSC_STATUS SSC_ctrl_trns_baud(SSC_DEVICE *SSC_device*, **IFX_UINT32** *SSC_ctrl_baud***)**

SSC driver run time baud rate control function used to select the baud rate for the chosen device. The argument should be treated as an IFX_UINT32 data type and should be set to specify the required baud rate.

E.g. IFX_UINT32 SSC_baud_rate = 32000; SSC_ctrl_trns_baud(first_dev, SSC_baud_rate);

Defined in: SSC_API.H

2.2.6.1 Return Value

SSC status

SSC_SUCCESS

Setting the new Baud rate is successful.

SSC_ERR

New baud rate is not supported by hardware or not able to get with in user specified tolerance level.

2.2.6.2 Parameters

SSC_device

SSC hardware module identification number.

SSC_ctrl_baud

New baud rate to be programmed.

2.2.7SSC_ctrl_trns_data

SSC_STATUS SSC_ctrl_trns_data(SSC_DEVICE *SSC_device*, **SSC_DATA** *SSC_ctrl_data***)**

SSC driver run time data control function, used to select the number of data bits per frame. The argument should be treated as an SSC_DATA data type and should be set between 2 and 16 (inclusive) to specify the required number of data bits.

E.g. SSC_DATA SSC_ctrl_data = 7; SSC_ctrl_trns_data(first_dev, SSC_ctrl_data);

Defined in: SSC_API.H

2.2.7.1Return Value

SSC status

SSC_SUCCESS
Setting number of bits per data frame is successful.

SSC_ERR
New configuration is not supported by hardware.

2.2.7.2Parameters

SSC_device
SSC hardware module identification number.

SSC_ctrl_data
New configuration specifies number of bits per frame.

2.2.8SSC_ctrl_trns_clock

SSC_STATUS SSC_ctrl_trns_clock(SSC_DEVICE *SSC_device*, **SSC_CLOCK_IDLE** *SSC_ctrl_clock***)**

SSC driver run time clock control function, used to select the level the clock line should remain at while the SSC is idle. The argument should be treated as an SSC_CLOCK_IDLE enumeration constant and should be set using one of the enumeration constants defined in SSC_CLOCK_IDLE.

E.g. SSC_CLOCK_IDLE SSC_ctrl_clk = SSC_CLOCK_IDLE_HIGH; SSC_ctrl_trns_clock(first_dev, SSC_ctrl_clk);

Defined in: SSC_API.H

2.2.8.1Return Value

SSC status

SSC_SUCCESS
Setting new idle state of clock is successful.

2.2.8.2Parameters

SSC_device
SSC hardware module identification number.

SSC_ctrl_clock
New configuration parameters.

2.2.9SSC_ctrl_trns_phase

SSC_STATUS SSC_ctrl_trns_phase(SSC_DEVICE *SSC_device*, **SSC_PHASE** *SSC_ctrl_phase***)**

SSC driver run time phase control function, used to select the clock edge that should be used to shift data and which edge should be used to latch data. The argument should be treated as an SSC_PHASE data type. The SSC_PHASE enumeration constant should be set using one of the enumeration constants defined in SSC_PHASE.

Defined in: SSC_API.H

2.2.9.1Return Value

SSC status

SSC_SUCCESS
Setting new phase of clock is successful.

2.2.9.2Parameters

SSC_device
SSC hardware module identification number.

SSC_ctrl_phase
New configuration parameters.

2.2.10SSC_ctrl_trns_shift

SSC_STATUS SSC_ctrl_trns_shift(SSC_DEVICE *SSC_device*, **SSC_SHIFT_DIR** *SSC_ctrl_shift*)

SSC driver run time data shift (LSB/MSB) control function, used to select which bit is shifted out by the SSC first. The argument should be treated as an SSC_SHIFT_DIR data type. The SSC_SHIFT_DIR enumeration constant should be set using one of the enumeration constants defined in SSC_SHIFT_DIR.

Defined in: SSC_API.H

2.2.10.1Return Value

SSC status

SSC_SUCCESS
Setting new configuration is successful.

2.2.10.2Parameters

SSC_device
SSC hardware module identification number.

SSC_ctrl_shift
New configuration parameters.

2.2.11SSC_ctrl_trns_all

SSC_STATUS SSC_ctrl_trns_all(SSC_DEVICE *SSC_device*, **SSC_COM_PARMS *** *SSC_ctrl_all*)

SSC driver run time configuration control function used to configure all the SSC communication settings and allows the operating mode to be changed. The argument should be treated as a SSC_COM_PARMS data type. The SSC_COM_PARMS structure should be initialised to set the desired communication parameters.

Defined in: SSC_API.H

2.2.11.1Return Value

SSC status

SSC_SUCCESS
Setting new configuration is successful.

2.2.11.2ParametersSetting new configuration is successful.

SSC_device
SSC hardware module identification number.

SSC_ctrl_all
New configuration parameters.

2.2.11.3Comments

Slave select configuration do not handled in this routine, user has to configure separately by calling SSC_ctrl_slv_oslct API.

2.2.12SSC_ctrl_fifo_get_rx_depth

SSC_STATUS SSC_ctrl_fifo_get_rx_depth(SSC_DEVICE *SSC_device*, **IFX_UINT8 *** *rx_fifo_depth***)**

SSC driver run time receive FIFO depth read control function, used to return the depth of the receive FIFO available on a SSC peripheral controlled by the HAL. For the purposes of this function the argument should be treated as an IFX_UINT8 pointer. If there is no receive FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned and the IFX_UINT8 pointed to will be set to 0. If a receive FIFO is available then the IFX_UINT8 will be set to the depth of the receive FIFO and SSC_SUCCESS will be returned from SSC_control_dev.

Defined in: SSC_API.H

2.2.12.1Return Value

SSC status

SSC_SUCCESS
Successfully return the receive FIFO depth.

SSC_ERR_NOT_SUPPORTED_HW
Receive FIFO not supported by hardware.

2.2.12.2Parameters

SSC_device
SSC hardware module identification number.

rx_fifo_depth
Pointer to read RX FIFO depth.

2.2.13SSC_ctrl_fifo_get_tx_depth

SSC_STATUS SSC_ctrl_fifo_get_tx_depth(SSC_DEVICE *SSC_device*, **IFX_UINT8 *** *tx_fifo_depth***)**

SSC driver run time transmit FIFO depth read control function, used to return the depth of the transmit FIFO available on a SSC peripheral controlled by the HAL. For the purposes of this function the argument should be treated as an IFX_UINT8 pointer. If there is no transmit FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned and the IFX_UINT8 pointed to will be set to 0. If a transmit FIFO is available then the IFX_UINT8 will be set to the depth of the transmit FIFO and SSC_SUCCESS will be returned from SSC_control_dev.

Defined in: SSC_API.H

2.2.13.1Return Value

SSC status

SSC_SUCCESS
Successfully return the transmit FIFO depth.

SSC_ERR_NOT_SUPPORTED_HW
Transmit FIFO not supported by hardware.

2.2.13.2Parameters

SSC_device
SSC hardware module identification number.

tx_fifo_depth
Pointer to read RX FIFO depth.

2.2.14SSC_ctrl_fifo_get_rx_level

SSC_STATUS SSC_ctrl_fifo_get_rx_level(SSC_DEVICE *SSC_device*, **IFX_UINT8 *** *rx_fifo_lvl***)**

SSC driver run time receive FIFO level read control function, used to return the number of data frames to be in receive FIFO available for receive on a SSC peripheral controlled by the HAL. For the purposes of this function the argument should be treated as an IFX_UINT8 pointer. If there is no receive FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned and the IFX_UINT8 pointed to will be set to 0. If a receive FIFO is available then the IFX_UINT8 will be set to level of the receive FIFO and SSC_SUCCESS will be returned from SSC_control_dev.

Defined in: SSC_API.H

2.2.14.1Return Value

SSC status

SSC_SUCCESS
Successfully return the number of data frames in receive FIFO.

SSC_ERR_NOT_SUPPORTED_HW
Transmit FIFO not supported by hardware.

2.2.14.2Parameters

SSC_device
SSC hardware module identification number.

rx_fifo_lvl
Pointer to read RX FIFO level.

2.2.15SSC_ctrl_fifo_get_tx_level

SSC_STATUS SSC_ctrl_fifo_get_tx_level(SSC_DEVICE *SSC_device*, **IFX_UINT8 *** *tx_fifo_lvl***)**

SSC driver run time transmit FIFO level read control function, used to return the number of data frames to be in transmit FIFO available for transmit on a SSC peripheral controlled by the HAL. For the purposes of this function the argument should be treated as an IFX_UINT8 pointer. If there is no transmit FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned and the IFX_UINT8 pointed to will be set to 0. If a transmit FIFO is available then the IFX_UINT8 will be set to level of the transmit FIFO and SSC_SUCCESS will be returned from SSC_control_dev.

Defined in: SSC_API.H

2.2.15.1Return Value

SSC status

SSC_SUCCESS
Successfully return the number of data frames in transmit FIFO.

SSC_ERR_NOT_SUPPORTED_HW
Transmit FIFO not supported by hardware.

2.2.15.2Parameters

SSC_device
SSC hardware module identification number.

tx_fifo_lvl
Pointer to read TX FIFO level.

2.2.16SSC_ctrl_fifo_set_rx_level

SSC_STATUS SSC_ctrl_fifo_set_rx_level(SSC_DEVICE *SSC_device*, **IFX_UINT8** *SSC_fifo_rx_lev_set***)**

SSC driver run time receive FIFO level write control function, this function may be used to set the filling level at which the receive FIFO will generate an interrupt. For the purposes of this function the argument should be treated as an IFX_UINT8 variable. If there is no receive FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned.

Defined in: SSC_API.H

2.2.16.1Return Value

SSC status

SSC_SUCCESS
Successfully programmed receive FIFO interrupt trigger level.

SSC_ERR_NOT_SUPPORTED_HW
Receive FIFO not supported by hardware or disabled by software.

2.2.16.2Parameters

SSC_device
SSC hardware module identification number.

SSC_fifo_rx_lev_set
Specifies the programmable RX FIFO interrupt trigger level

2.2.17SSC_ctrl_fifo_set_tx_level

SSC_STATUS SSC_ctrl_fifo_set_tx_level(SSC_DEVICE *SSC_device*, **IFX_UINT8** *SSC_fifo_tx_lev_set***)**

SSC driver run time transmit FIFO level write control function, this function may be used to set the filling level at which the transmit FIFO will generate an interrupt. For the purposes of this function the argument should be treated as an IFX_UINT8 variable. If there is no transmit FIFO then SSC_ERR_NOT_SUPPORTED_HW will be returned.

Defined in: SSC_API.H

2.2.17.1Return Value

SSC status

SSC_SUCCESS
: Successfully programmed receive FIFO interrupt trigger level.

SSC_ERR_NOT_SUPPORTED_HW
: Receive FIFO not supported by hardware or disabled by software.

2.2.17.2Parameters

SSC_device
: SSC hardware module identification number.

SSC_fifo_tx_lev_set
: Specifies the programmable TX FIFO interrupt trigger level

2.2.18SSC_ctrl_disable

SSC_STATUS SSC_ctrl_disable(SSC_DEVICE *SSC_device***)**

SSC driver run time disable control function used to disable the peripheral without terminating it. The result of calling this function is that all the GPIO pins the SSC 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: SSC_API.H

2.2.18.1Return Value

SSC status

SSC_SUCCESS
Successfully disabled the device.

2.2.18.2Parameters

SSC_device
SSC hardware module identification number.

2.2.19SSC_ctrl_enable

SSC_STATUS SSC_ctrl_enable(SSC_DEVICE *SSC_device*)

SSC driver run time enable control function must be called after SSC_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: SSC_API.H

2.2.19.1Return Value

SSC status

SSC_SUCCESS
Successfully enabled the device.

2.2.19.2Parameters

SSC_device
SSC hardware module identification number.

2.2.20SSC_ctrl_slv_oslct

SSC_STATUS SSC_ctrl_slv_oslct(SSC_DEVICE *SSC_device*, **SSC_OSLAVE_OPT** *SSC_slave_cfg***)**

SSC driver run time slave select output control function, this function is used to configure slave select output line. Configuration includes lead, trail, inact delays and delayed/ normal mode. This feature is fully hardware dependent. This function is used to swap the slave select lines and program delay cycles. SSC controller configures slave select line as input or output in salve or master mode respectively. These features are fully hardware dependent. It will free the pre-allocated port line and return to system HAL and reserve the new requested port line.

Defined in: SSC_API.H

2.2.20.1Return Value

SSC status

SSC_SUCCESS
Successfully configured the device.

2.2.20.2Parameters

SSC_device
SSC hardware module identification number.

SSC_slave_cfg
Slave select output line configuration parameters.

2.2.21SSC_read

SSC_STATUS SSC_read(SSC_DEVICE *SSC_device*, **SSC_TRANSFER *** *SSC_transfer***)**

SSC driver read function, the behavior of the SSC_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 pendinglist and then return SSC_SUCCESS provided the number of pending read requestsare less than SSC_CFG_REQUEST_QUEUE_WR.If no user call back function was supplied then the SSC_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: SSC_API.H

2.2.21.1Return Value

SSC status

SSC_SUCCESS
Reading data from device is successful.

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

SSC_ERR_NOT_SUPPORTED_HW
Requested transfer mode is not supported by hardware.

2.2.21.2Parameters

SSC_device
SSC hardware module identification number.

SSC_transfer
Read request configuration parameter values.

2.2.22SSC_write

SSC_STATUS SSC_write(SSC_DEVICE *SSC_device*, **SSC_TRANSFER *** *SSC_transfer*)

SSC driver write function, the behavior of the SSC_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 SSC_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 SSC_write function will return immediately with SSC_SUCCESS provided the number of pending write requests are less than SSC_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: SSC_API.H

2.2.22.1Return Value

SSC status

SSC_SUCCESS
: Writing data to device is successful.

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

SSC_ERR_NOT_SUPPORTED_HW
: Requested transfer mode is not supported by hardware.

2.2.22.2Parameters

SSC_device
: SSC hardware module identification number.

SSC_transfer
: Write request configuration parameter values.

3 Configure/Optimise SSC HAL

Although the SSC 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 SSC 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 SSC_CFG.H file:

–SSC device number checking

–Initialisation check on API calls

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

–Initial interrupts numbers/priorities settings

–SSC 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 SSC driver HAL configuration parameters

3.1.1SSC_CFG_DEV_CHK macro

```
#define SSC_CFG_DEV_CHK
```

Defined in: SSC_CFG.H

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

0
: Disable device number check

1
: Enable device number check

3.1.2SSC_CFG_INIT_CHK macro

```
#define SSC_CFG_INIT_CHK
```

Defined in: SSC_CFG.H

The following define selects whether certain SSC 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 initialisation check

1
: Enable initialisation check

3.1.3SSC_CFG_STAT_LOG macro

```
#define SSC_CFG_STAT_LOG
```

Defined in: SSC_CFG.H

The following define selects whether the SSC 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
: To disable statistics logging use

1
: To enable statistics logging use

3.1.4SSC_CFG_PCP_SUP macro

```
#define SSC_CFG_PCP_SUP
```

Defined in: SSC_CFG.H

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

Note: Hardware does not support this feature.

0
: To disable PCP support use

1
: To enable PCP support use

3.1.5SSC_CFG_DMA_SUP macro

```
#define SSC_CFG_DMA_SUP
```

Defined in: SSC_CFG.H

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

Note: Present version of software does not support this feature.

0

To disable DMA support use

1

To enable DMA support use

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

3.2.1SSC_CFG_FUNC_TERMINATE macro

```
#define SSC_CFG_FUNC_TERMINATE
```

Defined in: SSC_CFG.H

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

1

Include the function code in driver code

0

Exclude the function code from driver code

3.2.2SSC_CFG_FUNC_READ macro

```
#define SSC_CFG_FUNC_READ
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.3SSC_CFG_FUNC_WRITE macro

```
#define SSC_CFG_FUNC_WRITE
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.4SSC_CFG_FUNC_ABORT macro

```
#define SSC_CFG_FUNC_ABORT
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.5SSC_CFG_FUNC_STATUS macro

```
#define SSC_CFG_FUNC_STATUS
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.6SSC_CFG_FUNC_CONTROL macro

```
#define SSC_CFG_FUNC_CONTROL
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.7SSC_CFG_FUNC_CTRL_BAUD macro

```
#define SSC_CFG_FUNC_CTRL_BAUD
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.8 SSC_CFG_FUNC_CTRL_DATA macro

```
#define SSC_CFG_FUNC_CTRL_DATA
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.9 SSC_CFG_FUNC_CTRL_CLOCK macro

```
#define SSC_CFG_FUNC_CTRL_CLOCK
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.10 SSC_CFG_FUNC_CTRL_PHASE macro

```
#define SSC_CFG_FUNC_CTRL_PHASE
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.11 SSC_CFG_FUNC_CTRL_SHIFT macro

```
#define SSC_CFG_FUNC_CTRL_SHIFT
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.12 SSC_CFG_FUNC_CTRL_ALL macro

```
#define SSC_CFG_FUNC_CTRL_ALL
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.13 SSC_CFG_FUNC_CTRL_FIFO_GET_RX_DEPTH macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_GET_RX_DEPTH
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.14SSC_CFG_FUNC_CTRL_FIFO_GET_TX_DEPTH macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_GET_TX_DEPTH
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.15SSC_CFG_FUNC_CTRL_FIFO_GET_RX_LEVEL macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_GET_RX_LEVEL
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.16SSC_CFG_FUNC_CTRL_FIFO_GET_TX_LEVEL macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_GET_TX_LEVEL
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.17SSC_CFG_FUNC_CTRL_FIFO_SET_RX_LEVEL macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_SET_RX_LEVEL
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.18SSC_CFG_FUNC_CTRL_FIFO_SET_TX_LEVEL macro

```
#define SSC_CFG_FUNC_CTRL_FIFO_SET_TX_LEVEL
```

Defined in: SSC_CFG.H

This controls whether or not the SSC_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

3.2.19SSC_CFG_FUNC_CTRL_ENABLE macro

```
#define SSC_CFG_FUNC_CTRL_ENABLE
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.20SSC_CFG_FUNC_CTRL_DISABLE macro

```
#define SSC_CFG_FUNC_CTRL_DISABLE
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.21SSC_CFG_FUNC_SLVE_SLCT macro

```
#define SSC_CFG_FUNC_SLVE_SLCT
```

Defined in: SSC_CFG.H

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

1
: Include the function code in driver code

0
: Exclude the function code from driver code

3.2.22SSC_CFG_DUMMY_DATA macro

```
#define SSC_CFG_DUMMY_DATA
```

Defined in: SSC_CFG.H

The following define should be set to the value which is to be sent out, where requested number of read frames are more than transmitted frames.

Note: If the value of this macro is -1, then this feature will be disabled.

3.2.23 SSC_CFG_SLAVE_DUMMY_DAT macro

```
#define SSC_CFG_SLAVE_DUMMY_DAT
```

Defined in: SSC_CFG.H

The following define decides whether slave should send dummy data when there is no data to be transmitted, provided SSC_CFG_DUMMY_DATA (data value to be used to transmit) feature is enabled.

0
: Slave does not send dummy data

1
: Slave sends dummy data (SSC_CFG_DUMMY_DATA)

3.2.24 SSC_CFG_TX_IDLE_LEV macro

```
#define SSC_CFG_TX_IDLE_LEV
```

Defined in: SSC_CFG.H

The following define specifies the level at which the transmit line will be held at when the device is not selected. This option may be used with devices which have a slave select ability.

0
: To hold the transmit line at logic level 0.

1
: To hold the transmit line at logic level 1.

3.2.25 SSC_CFG_DELAY macro

```
#define SSC_CFG_DELAY
```

Defined in: SSC_CFG.H

This define selects delay for the selected device. This feature is fully hardware dependent, software ignore this feature if hardware does not support.

0
: Device operates in normal mode without delays.

1
: Device operates in delayed mode.

3.2.26SSC_CFG_TRAIL_DELAY_CYCLS macro

```
#define SSC_CFG_TRAIL_DELAY_CYCLS
```

Defined in: SSC_CFG.H

This define specifies the number of trailing delay cycles and this feature is fully hardware dependent. Software ignores this feature if hardware does not support this feature.

Value ranges from 0 to 3

3.2.27SSC_CFG_LEAD_DELAY_CYCLS macro

```
#define SSC_CFG_LEAD_DELAY_CYCLS
```

Defined in: SSC_CFG.H

This define specifies the number of leading delay cycles and this feature is fully hardware dependent. Software ignores this feature if hardware does not support this feature.

Value ranges from 0 to 3

3.2.28SSC_CFG_INACT_DELAY_CYCLS macro

```
#define SSC_CFG_INACT_DELAY_CYCLS
```

Defined in: SSC_CFG.H

This define specifies the number of delay cycles when device is in inactive state and this feature is fully hardware dependent. Software ignores this feature if hardware does not support this feature.

Value ranges from 0 to 3

3.2.29SSC_CFG_SLV_IDLE_LVL macro

```
#define SSC_CFG_SLV_IDLE_LVL
```

Defined in: SSC_CFG.H

This defines the logic level of the slave mode transmit signal MRST when the SSC is deselected

0
: MRST is 0, when SSC deselected in slave mode.

1
: MRST is 1, when SSC deselected in slave mode.

3.2.30 SSC_CFG_RX_FIFO_LEVEL macro

```
#define SSC_CFG_RX_FIFO_LEVEL
```

Defined in: SSC_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 SSC_CFG_RX_FIFO_LEVEL

3.2.30.1 Comments

The value should be in range of 0 to 8.

Note: Present hardware (TC1130 - A) does not support FIFOs, so user should configure this value to zero.

3.2.31 SSC_CFG_TX_FIFO_LEVEL macro

```
#define SSC_CFG_TX_FIFO_LEVEL
```

Defined in: SSC_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 SSC_CFG_TX_FIFO_LEVEL.

3.2.31.1 Comments

The value should be in range of 0 to 8.

Note: Present hardware (Tc1130 - A) does not support FIFOs, so user should configure this value to zero.

3.2.32 SSC_CFG_REQUEST_QUEUE_WR macro

```
#define SSC_CFG_REQUEST_QUEUE_WR
```

Defined in: SSC_CFG.H

The following define selects how many write requests should be buffered by the SSC HAL. If this define is set to 0 then the application program must wait after using SSC_write, for the transfer to complete before it may start the next write request.

If this define is set to a non zero value then the SSC_write API function will be able to queue this number of write requests.

Note: Provide non negative value.

3.2.33 SSC_CFG_REQUEST_QUEUE_RD macro

```
#define SSC_CFG_REQUEST_QUEUE_RD
```

Defined in: SSC_CFG.H

The following define selects how many read requests should be buffered by the SSC HAL. If this define is set to 0 then the application program must wait after using SSC_read, for the transfer to complete before it may start the next read request.

If this define is set to a non zero value then the SSC_read API function will be able to queue this number of read requests.

Note: Provide non negative value.

3.2.34 SSC_CFG_TX_CHK macro

```
#define SSC_CFG_TX_CHK
```

Defined in: SSC_CFG.H

This is used to enable or to disable transmit error checking.

0
: Disable transmit error check.

1
: Enable transmit error check.

3.2.35 SSC_CFG_RX_CHK macro

```
#define SSC_CFG_RX_CHK
```

Defined in: SSC_CFG.H

This is used to enable or to disable receive error checking.

0
: Disable receive error check.

1
: Enable receive error check.

3.2.36SSC_CFG_PHASE_CHK macro

```
#define SSC_CFG_PHASE_CHK
```

Defined in: SSC_CFG.H

This is used to enable or to disable phase error checking.

0
: Disable phase error check.

1
: Enable phase error check.

3.2.37SSC_CFG_BR_CHK macro

```
#define SSC_CFG_BR_CHK
```

Defined in: SSC_CFG.H

This is used to enable or to disable baud rate error checking.

0
: Disable baud rate error check.

1
: Enable baud rate error check.

3.2.38SSC_CFG_BAUD_TOL macro

```
#define SSC_CFG_BAUD_TOL
```

Defined in: SSC_CFG.H

This value used to compare the resultant baud rate deviation from the required baud rate. Tolerance will be calculated using the following formula.

(calculate_baudrate - required_baudrate) / required_baudrate.

3.2.39SSC_CFG_RMC_VAL macro

```
#define SSC_CFG_RMC_VAL
```

Defined in: SSC_CFG.H

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

The value should be in range of 1 - 255, if hardware supports RMC value (e.g. TC1765).Note: This value should be always zero, if hardware does not support

RMC value (e.g. TC1130).

3.2.40SSC_CFG_LPBACK macro

```
#define SSC_CFG_LPBACK
```

Defined in: SSC_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

4 Application Examples

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

4.1 Communication between SSC0 and SSC1 modules

```
-------------------------------------------------------------------------
Test description

A). This test is carried out between SSC0 and SSC1 module.

B). Connect the pins in the following way
   SSC0 RX - SSC1 TX
   SSC1 RX - SSC0 TX
   If SSC0 and SSC1 on different board connect grounds also.

Alternative to this is:

a. The SSC0 and SSC1 port pins are connected to x802 bus.

b. Connect SSC0 and SSC1 by using T2LA (Triboard to Logic Analyser) board.

c. In T2LA board SSC0 pins connected to X3-9, x3-11, x3-13 and SSC1 pins
connected to X1-9,
   X1-11, X1-13.

C). Include the files in project space as explained in point 2. above.

D). Compile, build and download the image to target board.

E). Wait for some time (Say 2 minutes), stop and observe the follwing
variables for successful
   test.
buffer_rx1="abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefgh
ijklmnopqrst"
buffer_rx2="5555555555555555555555555555555555555555555555555555555555555
555555555555"
buffer_rx3="abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefgh
ijklmnopqrst"
buffer_rx4="123456789012345678901234567890123456789012345678901234567890
123456789012"

flag1 = 4, flag2 = 4.

F). Observe the status of devices by using stat1, stat2 variables.
```

```
/*
All required includes are done in SSC_API.H, this
allows the end user to easily find what is used by
the HAL.
*/
#include "COMPILER.H"
#include "SSC_CFG.H"
#include "SSC_API.H"
#include "SSC_IDL.H"
#include "SSC_IIL.H"
#include "SYS_API.H"
#include "SYS_CFG.H"
#include <stdio.h> /* =========== for debugging only ========= */

/*Flags used for statistics purpose*/
IFX_UINT8 flag1 = 0, flag2 = 0, flag3 = 0;

/*Transmit user callback function*/
void ssc_hal_ucb_tx( SSC_TRANSFER *trans_struct, SSC_STATUS stat )
{
  if(stat == SSC_SUCCESS)
  {
    flag1++;
  }
}
/*Receive user callback function*/
void ssc_hal_ucb_rx( SSC_TRANSFER *trans_struct, SSC_STATUS stat )
{
  if(stat == SSC_SUCCESS)
  {
    flag2++;
  }
}

/*Buffers used for transmit and receive data*/
unsigned                      char                       buffer_tx1[120]=
"55555555555555555555555555555555555555555555555555555555555555555555555
5";
unsigned                char                  buffer_tx2[120]               =
"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs
t";
unsigned char buffer_rx1[120] = {0};
unsigned char buffer_rx2[120] = {0};

unsigned                char                  buffer_tx3[120]               =
"12345678901234567890123456789012345678901234567890123456789012345678901
2";
```

```
unsigned char buffer_tx4[120] =
"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs
t";
unsigned char buffer_rx3[120] = {0};
unsigned char buffer_rx4[120] = {0};

/*Transfer structurs used for communication*/
SSC_TRANSFER transfer_tx1,transfer_tx2, transfer_rx1, transfer_rx2;
SSC_TRANSFER transfer_tx3,transfer_tx4, transfer_rx3, transfer_rx4;

/*Structurs used for status*/
SSC_STAT_INF stat1, stat2;

int main()
{
SSC_COM_PARMS parms1, parms2;
IFX_VUINT32 i = 0xffffff;

// MASTER SSC0
parms1.SSC_com_baud = 38400; /*Baud rate*/
parms1.SSC_com_clock = SSC_CLOCK_IDLE_LOW;
parms1.SSC_com_data = 8;    // data width
parms1.SSC_com_phase = SSC_LATCH_FALLING;
parms1.SSC_com_shift = SSC_LSB_FIRST;
parms1.SSC_mode = SSC_MASTER; /*SSC0 acts as master*/
parms1.SSC_slave_num = 0;

// SLAVE  SSC1
parms2.SSC_com_baud = 38400;
parms2.SSC_com_clock = SSC_CLOCK_IDLE_LOW;
parms2.SSC_com_data = 8;         // Data width
parms2.SSC_com_phase = SSC_LATCH_FALLING;
parms2.SSC_com_shift = SSC_LSB_FIRST;
parms2.SSC_mode = SSC_SLAVE; /*SSC1 acts as slave*/
parms2.SSC_slave_num = 0;

/*Initialise tranfer structure with parameters*/
transfer_tx1.SSC_buffer = (IFX_UINT32 *) &buffer_tx1;
transfer_tx1.SSC_buffer_size = 72;
transfer_tx1.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_tx1.SSC_trans_ucb = ssc_hal_ucb_tx; /*Unblocked request*/

transfer_tx2.SSC_buffer = (IFX_UINT32 *) &buffer_tx2;
transfer_tx2.SSC_buffer_size = 72;
transfer_tx2.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_tx2.SSC_trans_ucb = ssc_hal_ucb_tx;
```

```
transfer_rx1.SSC_buffer = (IFX_UINT32 *) &buffer_rx1;
transfer_rx1.SSC_buffer_size = 72;
transfer_rx1.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_rx1.SSC_trans_ucb = ssc_hal_ucb_rx;

transfer_rx2.SSC_buffer = (IFX_UINT32 *) &buffer_rx2;
transfer_rx2.SSC_buffer_size = 72;
transfer_rx2.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_rx2.SSC_trans_ucb = ssc_hal_ucb_rx;

if(SSC_initialise_dev(0, &parms1) != SSC_SUCCESS)
{
  printf("Fail0\n");
  return 0;
}

if(SSC_initialise_dev(1, &parms2) != SSC_SUCCESS)
{
  printf("Fail1\n");
  return 0;
}


SSC_read(1, &transfer_rx2);
SSC_write(1, &transfer_tx2);
SSC_read(0, &transfer_rx1);
SSC_write(0, &transfer_tx1);

#if 1
    i = 0xffffff;
    /*Sufficeint delay to complete all transfers, flag1 = 4, flag2 = 4*/
#else
    i = 0xff;
    /*Insufficeint delay to complete all transfers, flag1 != 4, flag2 !=
4*/
#endif

for(;i>0;i--)
{
}
SSC_abort(0);
SSC_abort(1);

i = 0xffff;
for(;i>0;i--)
{
}
```

```
transfer_tx3.SSC_buffer = (IFX_UINT32 *) &buffer_tx3;
transfer_tx3.SSC_buffer_size = 72;
transfer_tx3.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_tx3.SSC_trans_ucb = ssc_hal_ucb_tx;

transfer_tx4.SSC_buffer = (IFX_UINT32 *) &buffer_tx4;
transfer_tx4.SSC_buffer_size = 72;
transfer_tx4.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_tx4.SSC_trans_ucb = ssc_hal_ucb_tx;

transfer_rx3.SSC_buffer = (IFX_UINT32 *) &buffer_rx3;
transfer_rx3.SSC_buffer_size = 72;
transfer_rx3.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_rx3.SSC_trans_ucb = ssc_hal_ucb_rx;

transfer_rx4.SSC_buffer = (IFX_UINT32 *) &buffer_rx4;
transfer_rx4.SSC_buffer_size = 72;
transfer_rx4.SSC_transfer_mode = SYS_TRNS_MCU_INT;
transfer_rx4.SSC_trans_ucb = ssc_hal_ucb_rx;

SSC_read(1, &transfer_rx4);
SSC_write(1, &transfer_tx4);
SSC_read(0, &transfer_rx3);
SSC_write(0, &transfer_tx3);

i = 0xfffff;
for(;i>0;i--)
{
}

SSC_status_dev(0, &stat1);
SSC_status_dev(1, &stat2);
while(1)
{
}
return 0;
}
```

4.2 Port configuration for SSC

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_SSC0_MRST.

#define SYS_GPIO_SSC0_MRST 2, 2, 0, 1, 0, -1, -1, -1

The define value will be configured as mentioned below

Column	*Control field*
1	Port number(2)
2	Control bit in port(2).
3	Direction bit(0)
4	Alternate 0 control field(1).
5	Alternate 1 control field(0).
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 SSC LLD with DAvE genearated 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 Application note to use LLD with TC1130-A hardware

- To remove FIFO support from SSC Low Level Driver

In SSC_IDL.H file

Set the values of "SSC_HW_FIFO_RX" and "SSC_HW_FIFO_TX" to zero.

and

In SSC_CFG.H file

Set the values of "SSC_CFG_TX_FIFO_LEVEL" and "SSC_CFG_RX_FIFO_LEVEL" to zero.

- To remove delay configuration support from SSC Low Level Driver

In SSC_IDL.H file

Set the value of "SSC_HW_DEL_SUP" to zero.

8 Related Documentation

- Infineon Technologies HAL/Device Driver Software Suite Overview
- Ethernet and ASC HAL User Guide

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 SSC 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 SSC_TRANSFER structure passed to the SSC_read and SSC_write API functions. If the transfer operation is not available in the system then SSC_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 SSC 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 range from 1 to 255. Each interrupt should have a unique priority. Lowest priority is 1 and the highest priority is 255.

10.4.1SYS_SSC0_RIR macro

```
#define SYS_SSC0_RIR
```

Defined in: SYS_CFG.H

SSC0 receive interrupt priority.

10.4.2SYS_SSC0_TIR macro

```
#define SYS_SSC0_TIR
```

Defined in: SYS_CFG.H

SSC0 transmit interrupt priority.

10.4.3SYS_SSC0_EIR macro

```
#define SYS_SSC0_EIR
```

Defined in: SYS_CFG.H

SSC0 error interrupt priority.

10.4.4SYS_SSC1_RIR macro

```
#define SYS_SSC1_RIR
```

Defined in: SYS_CFG.H

SSC1 receive interrupt priority.

10.4.5SYS_SSC1_TIR macro

```
#define SYS_SSC1_TIR
```

Defined in: SYS_CFG.H

SSC1 transmit interrupt priority.

10.4.6SYS_SSC1_EIR macro

```
#define SYS_SSC1_EIR
```

Defined in: SYS_CFG.H

SSC1 error interrupt priority.

10.5GPIO Port Configuration Parameters

This section defines the configurable port settings of the peripherals. These macros define the 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: 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_SSC0_MRST macro

```
#define SYS_GPIO_SSC0_MRST
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 master receive slave transmit line.

10.5.2SYS_GPIO_SSC0_MTSR macro

```
#define SYS_GPIO_SSC0_MTSR
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 master transmit slave receive line.

10.5.3SYS_GPIO_SSC0_SCLK macro

```
#define SYS_GPIO_SSC0_SCLK
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 synchronous clock line.

10.5.4SYS_GPIO_SSC0_SLSI macro

```
#define SYS_GPIO_SSC0_SLSI
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave select input line.

10.5.5SYS_GPIO_SSC0_SLSO00 macro

```
#define SYS_GPIO_SSC0_SLSO00
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(0) select line.

10.5.6SYS_GPIO_SSC0_SLSO01 macro

```
#define SYS_GPIO_SSC0_SLSO01
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(1) select line.

10.5.7SYS_GPIO_SSC0_SLSO02 macro

```
#define SYS_GPIO_SSC0_SLSO02
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(2) select line.

10.5.8SYS_GPIO_SSC0_SLSO03 macro

```
#define SYS_GPIO_SSC0_SLSO03
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(3) select line.

10.5.9SYS_GPIO_SSC0_SLSO04 macro

```
#define SYS_GPIO_SSC0_SLSO04
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(4) select line.

10.5.10SYS_GPIO_SSC0_SLSO05 macro

```
#define SYS_GPIO_SSC0_SLSO05
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(5) select line.

10.5.11SYS_GPIO_SSC0_SLSO06 macro

```
#define SYS_GPIO_SSC0_SLSO06
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(6) select line.

10.5.12SYS_GPIO_SSC0_SLSO07 macro

```
#define SYS_GPIO_SSC0_SLSO07
```

Defined in: SYS_CFG.H

Port configuration used for SSC0 slave(7) select line.

10.5.13SYS_GPIO_SSC1_MRST macro

```
#define SYS_GPIO_SSC1_MRST
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 master receive slave transmit line.

10.5.14SYS_GPIO_SSC1_MTSR macro

```
#define SYS_GPIO_SSC1_MTSR
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 master transmit slave receive line.

10.5.15SYS_GPIO_SSC1_SCLK macro

```
#define SYS_GPIO_SSC1_SCLK
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 synchronous clock line.

10.5.16SYS_GPIO_SSC1_SLSI macro

```
#define SYS_GPIO_SSC1_SLSI
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave select input line.

10.5.17 SYS_GPIO_SSC1_SLSO00 macro

```
#define SYS_GPIO_SSC1_SLSO00
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(0) select line.

10.5.18 SYS_GPIO_SSC1_SLSO01 macro

```
#define SYS_GPIO_SSC1_SLSO01
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(1) select line.

10.5.19 SYS_GPIO_SSC1_SLSO02 macro

```
#define SYS_GPIO_SSC1_SLSO02
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(2) select line.

10.5.20 SYS_GPIO_SSC1_SLSO03 macro

```
#define SYS_GPIO_SSC1_SLSO03
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(3) select line.

10.5.21 SYS_GPIO_SSC1_SLSO04 macro

```
#define SYS_GPIO_SSC1_SLSO04
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(4) select line.

10.5.22SYS_GPIO_SSC1_SLSO05 macro

```
#define SYS_GPIO_SSC1_SLSO05
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(5) select line.

10.5.23SYS_GPIO_SSC1_SLSO06 macro

```
#define SYS_GPIO_SSC1_SLSO06
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(6) select line.

10.5.24SYS_GPIO_SSC1_SLSO07 macro

```
#define SYS_GPIO_SSC1_SLSO07
```

Defined in: SYS_CFG.H

Port configuration used for SSC1 slave(7) select line.

l

http://www.infineon.com

Published by Infineon Technologies AP