File List

Here is a list of all files with brief descriptions:

[detail level 1 2 3]

▼ **Afe79xx**	
▼ **Include**	
afe79xxLog.h	This file has function definitions regarding logging. **Version 2.1:**
afe79xxTypes.h	
afeCommonMacros.h	This file contains C Macros for different kinds of operations in the AFE function. **Version 2.1:**
afeParameters.h	
agc.h	
baseFunc.h	
basicFunctions.h	
calibrations.h	
controls.h	
dsaAndNco.h	
hMacro.h	
init.h	
jesd.h	
pap.h	
paramsSetterGetter.h	
serDes.h	
▼ **Src**	
agc.c	This file has AGC related functions. **Version 2.2:**
basicFunctions.c	This file has Basic SPI functions. **Version 2.2:**
calibrations.c	This file has Factory calibration related functions. **Version 2.1:**
controls.c	This file has generic control related functions. **Version 2.2:**
dsaAndNco.c	This file has DSA and NCO related functions. **Version 2.2:**
hMacro.c	This file has Macros related functions. **Version 2.1:**
init.c	
jesd.c	This file has JESD related functions. **Version 2.2:**
pap.c	This file has PAP related functions. **Version 2.1:**
serDes.c	This file has SerDes related functions. **Version 2.2:**
▼ **Afe79xxUser**	
▼ **Src**	
afeParameters.c	This file contains System Parameters used in AFE initialization
baseFunc.c	This file has functions which can be edited by customers to integrate it into their system. **Version 2.1.1:**
▼ **example**	
main.c	

Generated by doxygen 1.8.17

Afe79xx Directory Reference

Directories

directory	**Include**
directory	**Src**

Generated by doxygen 1.8.17

Include Directory Reference

Files

file	**afe79xxLog.h** [code] This file has function definitions regarding logging. **Version 2.1:**
file	**afe79xxTypes.h** [code]
file	**afeCommonMacros.h** [code] This file contains C Macros for different kinds of operations in the AFE function. **Version 2.1:**
file	**afeParameters.h** [code]
file	**agc.h** [code]
file	**baseFunc.h** [code]
file	**basicFunctions.h** [code]
file	**calibrations.h** [code]
file	**controls.h** [code]
file	**dsaAndNco.h** [code]
file	**hMacro.h** [code]
file	**init.h** [code]
file	**jesd.h** [code]
file	**pap.h** [code]
file	**paramsSetterGetter.h** [code]
file	**serDes.h** [code]

Generated by doxygen 1.8.17

afe79xxLog.h File Reference

This file has function definitions regarding logging.

Version 2.1:

More...

Go to the source code of this file.

Macros

#define	**AFE_LOG_LEVEL_ERROR** 0 /* error conditions */
#define	**AFE_LOG_LEVEL_WARNING** 1 /*warning conditions */
#define	**AFE_LOG_LEVEL_INFO** 2 /* informational */
#define	**AFE_LOG_LEVEL_SPILOG** 3 /*SPI-level messages */

#define	AFE_LOG_LEVEL_DEBUG 4 /*debug-level messages */
#define	afeLogErr(fmt, ...) afeLogmsg(AFE_LOG_LEVEL_ERROR, "[%s][%s][%d]ERROR:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
#define	afeLogDbg(fmt, ...) afeLogmsg(AFE_LOG_LEVEL_DEBUG, "[%s][%s][%d]DEBUG:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
#define	afeLogSpiLog(fmt, ...) afeLogmsg(AFE_LOG_LEVEL_SPILOG, "[%s][%s][%d]SPI " fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
#define	afeLogInfo(fmt, ...) afeLogmsg(AFE_LOG_LEVEL_INFO, "[%s][%s][%d]INFO:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)

Detailed Description

This file has function definitions regarding logging.

Version 2.1:

1. Added documentation Level for SPI Log.

Macro Definition Documentation

◆ AFE_LOG_LEVEL_DEBUG

```
#define AFE_LOG_LEVEL_DEBUG   4 /*debug-level messages */
```

◆ AFE_LOG_LEVEL_ERROR

```
#define AFE_LOG_LEVEL_ERROR   0 /* error conditions */
```

◆ AFE_LOG_LEVEL_INFO

```
#define AFE_LOG_LEVEL_INFO   2 /* informational */
```

◆ AFE_LOG_LEVEL_SPILOG

```
#define AFE_LOG_LEVEL_SPILOG   3 /*SPI-level messages */
```

◆ AFE_LOG_LEVEL_WARNING

```
#define AFE_LOG_LEVEL_WARNING   1 /*warning conditions */
```

◆ afeLogDbg

```
#define afeLogDbg (   fmt,
                      ...
                  )      afeLogmsg(AFE_LOG_LEVEL_DEBUG, "[%s][%s][%d]DEBUG:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
```

◆ afeLogErr

```
#define afeLogErr (   fmt,
                      ...
                  )   afeLogmsg(AFE_LOG_LEVEL_ERROR, "[%s][%s][%d]ERROR:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
```

◆ afeLogInfo

```
#define afeLogInfo (   fmt,
                       ...
                   )   afeLogmsg(AFE_LOG_LEVEL_INFO, "[%s][%s][%d]INFO:" fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
```

◆ afeLogSpiLog

```
#define afeLogSpiLog (   fmt,
                         ...
                     )   afeLogmsg(AFE_LOG_LEVEL_SPILOG, "[%s][%s][%d]SPI " fmt "\r\n", __FILE__, __func__, __LINE__, __VA_ARGS__)
```

Generated by doxygen 1.8.17

afe79xxTypes.h File Reference

Go to the source code of this file.

Macros

#define	AFE_NUM_RX_CHANNELS	4
#define	AFE_NUM_RX_CHANNELS_BITWISE	0xf
#define	AFE_NUM_BANDS_PER_RX	2
#define	AFE_NUM_TX_CHANNELS	4
#define	AFE_NUM_TX_CHANNELS_BITWISE	0xf
#define	AFE_NUM_BANDS_PER_TX	2
#define	AFE_NUM_FB_CHANNELS	2
#define	AFE_NUM_FB_CHANNELS_BITWISE	0x3
#define	AFE_NUM_BANDS_PER_FB	1
#define	AFE_NUM_JESD_INSTANCES	2
#define	AFE_NUM_CH_PER_JESD_INSTANCE	2
#define	AFE_NUM_SERDES_LANES	8
#define	jesdToSerdesLaneMapping	
#define	AFE_PAGE_START_ADDR	0x10
#define	AFE_PAGE_END_ADDR	0x19
#define	AFE_MACRO_NO_ERROR	0
#define	AFE_MACRO_ERROR_IN_OPCODE	1
#define	AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED	2
#define	AFE_MACRO_ERROR_IN_OPERAND	4
#define	AFE_MACRO_ERROR_IN_EXECUTION	8
#define	AFE_MACRO_STATUS_REG_ADDR	0xF0
#define	AFE_MACRO_OPCODE_REG_ADDR	0x193

#define	AFE_MACRO_EXTENDED_ERROR_CODE_REG_ADDR	0xF2
#define	AFE_MACRO_RESULT_START_REG_ADDR	0xF8
#define	AFE_MACRO_OPERAND_START_REG_ADDR	0xA0
#define	AFE_MACRO_PAGE_REG_ADDR	0x18
#define	AFE_MACRO_PAGE_SEL_VAL	0x20
#define	AFE_MACRO_OPCODE_SYSTEM_TUNE	0x90
#define	AFE_MACRO_OPCODE_PREPARE_FOR_TUNE	0x35
#define	AFE_MACRO_OPCODE_SYSTEM_TUNE_SELECTIVE	0x36
#define	AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION	0x37
#define	AFE_MACRO_OPCODE_UPDATE_TX_DIG_PARAM	0x50
#define	AFE_MACRO_OPCODE_UPDATE_TX_GAIN	0x51
#define	AFE_MACRO_OPCODE_UPDATE_SYSTEM_RX_CHANNEL_FREQUENCY_CONFIGURATION	0x38
#define	AFE_MACRO_OPCODE_UPDATE_SYSTEM_FB_CHANNEL_FREQUENCY_CONFIGURATION	0x39
#define	AFE_MACRO_OPCODE_APPLY_DSA_GAIN_PHASE_COMPENSATION	0x11
#define	AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION_ALL_BANDS	0x3E
#define	AFE_MACRO_OPCODE_FACTORY_RX_DSA_GAIN_PHASE_CALIBRATION	0x41
#define	AFE_MACRO_OPCODE_FACTORY_TX_DSA_GAIN_PHASE_CALIBRATION	0x42
#define	AFE_MACRO_OPCODE_CONFIG_SIGGEN_FOR_CAL	0x48
#define	AFE_MACRO_OPCODE_AGC_STATE_CONTROL	0x68
#define	AFE_MACRO_OPCODE_AGC_DIG_DET_CONFIG	0x58
#define	AFE_MACRO_OPCODE_AGC_DET_TIME_CONST_CONFIG	0x59
#define	AFE_MACRO_OPCODE_AGC_DIG_DET_ABSOLUTE_NUM_CROSSINGS_CONFIG	0x5B
#define	AFE_MACRO_OPCODE_AGC_DIG_DET_RELATIVE_NUM_CROSSINGS_CONFIG	0x5A
#define	AFE_MACRO_OPCODE_EXT_AGC_CONFIG	0x5C
#define	AFE_MACRO_OPCODE_INT_AGC_CONTROLLER_CONFIG	0x5E
#define	AFE_MACRO_OPCODE_MIN_MAX_DSA_ATTN_CONFIG	0x5F
#define	AFE_MACRO_OPCODE_AGC_EXT_LNA_CONFIG	0x61
#define	AFE_MACRO_OPCODE_AGC_EXT_LNA_GAIN_CONFIG	0x66
#define	AFE_MACRO_OPCODE_AGC_GAIN_STEP_SIZE_CONFIG	0x67
#define	AFE_MACRO_OPCODE_AGC_RF_ANALOG_CONFIG	0x65
#define	AFE_MACRO_OPCODE_ALC_CONFIGURATION	0x69
#define	AFE_MACRO_OPCODE_FLOATING_POINT_CONFIG_ALC	0x6A
#define	AFE_MACRO_OPCODE_COARSE_FINE_MODE_ALC	0x6B
#define	AFE_RX_DSA_MAX_ANA_DSA_DB	25
#define	AFE_TX_DSA_MAX_ANA_DSA_DB	34
#define	AFE_FB_DSA_MAX_ANA_DSA_DB	25
#define	AFE_RX_DSA_MAX_ANA_DSA_INDEX	50
#define	AFE_RX_DSA_MAX_DIG_DSA_INDEX	47
#define	AFE_TX_DSA_MAX_ANA_DSA_INDEX	29
#define	AFE_FB_DSA_MAX_ANA_DSA_INDEX	50
#define	AFE_TX_DSA_MAX_ANA_PLUS_DIG_DSA_DB	39
#define	AFE_AGC_MAX_WIN_LEN	4000000
#define	AFE_AGC_MAX_ABS_NUM_HITS	0xffffff
#define	NULL	(0)

Typedefs

typedef enum **RET_TYPE** **RetType_e**

Enumerations

```
enum  RET_TYPE { RET_OK = 0, RET_EXEC_FAIL }
```

Macro Definition Documentation

◆ AFE_AGC_MAX_ABS_NUM_HITS

```
#define AFE_AGC_MAX_ABS_NUM_HITS   0xffffff
```

◆ AFE_AGC_MAX_WIN_LEN

```
#define AFE_AGC_MAX_WIN_LEN   4000000
```

◆ AFE_FB_DSA_MAX_ANA_DSA_DB

```
#define AFE_FB_DSA_MAX_ANA_DSA_DB   25
```

◆ AFE_FB_DSA_MAX_ANA_DSA_INDEX

```
#define AFE_FB_DSA_MAX_ANA_DSA_INDEX   50
```

◆ AFE_MACRO_ERROR_IN_EXECUTION

```
#define AFE_MACRO_ERROR_IN_EXECUTION   8
```

◆ AFE_MACRO_ERROR_IN_OPCODE

```
#define AFE_MACRO_ERROR_IN_OPCODE   1
```

◆ AFE_MACRO_ERROR_IN_OPERAND

```
#define AFE_MACRO_ERROR_IN_OPERAND   4
```

◆ AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED

```
#define AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED   2
```

◆ AFE_MACRO_EXTENDED_ERROR_CODE_REG_ADDR

```
#define AFE_MACRO_EXTENDED_ERROR_CODE_REG_ADDR   0xF2
```

◆ AFE_MACRO_NO_ERROR

```
#define AFE_MACRO_NO_ERROR   0
```

◆ AFE_MACRO_OPCODE_AGC_DET_TIME_CONST_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_DET_TIME_CONST_CONFIG   0x59
```

◆ AFE_MACRO_OPCODE_AGC_DIG_DET_ABSOLUTE_NUM_CROSSINGS_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_DIG_DET_ABSOLUTE_NUM_CROSSINGS_CONFIG   0x5B
```

◆ AFE_MACRO_OPCODE_AGC_DIG_DET_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_DIG_DET_CONFIG   0x58
```

◆ AFE_MACRO_OPCODE_AGC_DIG_DET_RELATIVE_NUM_CROSSINGS_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_DIG_DET_RELATIVE_NUM_CROSSINGS_CONFIG   0x5A
```

◆ AFE_MACRO_OPCODE_AGC_EXT_LNA_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_EXT_LNA_CONFIG   0x61
```

◆ AFE_MACRO_OPCODE_AGC_EXT_LNA_GAIN_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_EXT_LNA_GAIN_CONFIG   0x66
```

◆ AFE_MACRO_OPCODE_AGC_GAIN_STEP_SIZE_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_GAIN_STEP_SIZE_CONFIG   0x67
```

◆ AFE_MACRO_OPCODE_AGC_RF_ANALOG_CONFIG

```
#define AFE_MACRO_OPCODE_AGC_RF_ANALOG_CONFIG   0x65
```

◆ AFE_MACRO_OPCODE_AGC_STATE_CONTROL

```
#define AFE_MACRO_OPCODE_AGC_STATE_CONTROL   0x68
```

◆ AFE_MACRO_OPCODE_ALC_CONFIGURATION

```
#define AFE_MACRO_OPCODE_ALC_CONFIGURATION   0x69
```

◆ AFE_MACRO_OPCODE_APPLY_DSA_GAIN_PHASE_COMPENSATION

```
#define AFE_MACRO_OPCODE_APPLY_DSA_GAIN_PHASE_COMPENSATION   0x11
```

◆ AFE_MACRO_OPCODE_COARSE_FINE_MODE_ALC

```
#define AFE_MACRO_OPCODE_COARSE_FINE_MODE_ALC   0x6B
```

◆ AFE_MACRO_OPCODE_CONFIG_SIGGEN_FOR_CAL

```
#define AFE_MACRO_OPCODE_CONFIG_SIGGEN_FOR_CAL   0x48
```

◆ AFE_MACRO_OPCODE_EXT_AGC_CONFIG

```
#define AFE_MACRO_OPCODE_EXT_AGC_CONFIG   0x5C
```

◆ AFE_MACRO_OPCODE_FACTORY_RX_DSA_GAIN_PHASE_CALIBRATION

```
#define AFE_MACRO_OPCODE_FACTORY_RX_DSA_GAIN_PHASE_CALIBRATION   0x41
```

◆ AFE_MACRO_OPCODE_FACTORY_TX_DSA_GAIN_PHASE_CALIBRATION

```
#define AFE_MACRO_OPCODE_FACTORY_TX_DSA_GAIN_PHASE_CALIBRATION   0x42
```

◆ AFE_MACRO_OPCODE_FLOATING_POINT_CONFIG_ALC

```
#define AFE_MACRO_OPCODE_FLOATING_POINT_CONFIG_ALC   0x6A
```

◆ AFE_MACRO_OPCODE_INT_AGC_CONTROLLER_CONFIG

```
#define AFE_MACRO_OPCODE_INT_AGC_CONTROLLER_CONFIG   0x5E
```

◆ AFE_MACRO_OPCODE_MIN_MAX_DSA_ATTN_CONFIG

```
#define AFE_MACRO_OPCODE_MIN_MAX_DSA_ATTN_CONFIG   0x5F
```

◆ AFE_MACRO_OPCODE_PREPARE_FOR_TUNE

```
#define AFE_MACRO_OPCODE_PREPARE_FOR_TUNE   0x35
```

◆ AFE_MACRO_OPCODE_REG_ADDR

```
#define AFE_MACRO_OPCODE_REG_ADDR   0x193
```

◆ AFE_MACRO_OPCODE_SYSTEM_TUNE

```
#define AFE_MACRO_OPCODE_SYSTEM_TUNE   0x90
```

◆ AFE_MACRO_OPCODE_SYSTEM_TUNE_SELECTIVE

```
#define AFE_MACRO_OPCODE_SYSTEM_TUNE_SELECTIVE   0x36
```

◆ AFE_MACRO_OPCODE_UPDATE_SYSTEM_FB_CHANNEL_FREQUENCY_CONFIGURATION

```
#define AFE_MACRO_OPCODE_UPDATE_SYSTEM_FB_CHANNEL_FREQUENCY_CONFIGURATION   0x39
```

◆ AFE_MACRO_OPCODE_UPDATE_SYSTEM_RX_CHANNEL_FREQUENCY_CONFIGURATION

```
#define AFE_MACRO_OPCODE_UPDATE_SYSTEM_RX_CHANNEL_FREQUENCY_CONFIGURATION   0x38
```

◆ AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION

```
#define AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION   0x37
```

◆ AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION_ALL_BANDS

```
#define AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION_ALL_BANDS   0x3E
```

◆ AFE_MACRO_OPCODE_UPDATE_TX_DIG_PARAM

```
#define AFE_MACRO_OPCODE_UPDATE_TX_DIG_PARAM   0x50
```

◆ AFE_MACRO_OPCODE_UPDATE_TX_GAIN

```
#define AFE_MACRO_OPCODE_UPDATE_TX_GAIN   0x51
```

◆ AFE_MACRO_OPERAND_START_REG_ADDR

```
#define AFE_MACRO_OPERAND_START_REG_ADDR   0xA0
```

◆ AFE_MACRO_PAGE_REG_ADDR

```
#define AFE_MACRO_PAGE_REG_ADDR   0x18
```

◆ AFE_MACRO_PAGE_SEL_VAL

```
#define AFE_MACRO_PAGE_SEL_VAL   0x20
```

◆ AFE_MACRO_RESULT_START_REG_ADDR

```
#define AFE_MACRO_RESULT_START_REG_ADDR   0xF8
```

◆ AFE_MACRO_STATUS_REG_ADDR

```
#define AFE_MACRO_STATUS_REG_ADDR   0xF0
```

◆ AFE_NUM_BANDS_PER_FB

```
#define AFE_NUM_BANDS_PER_FB   1
```

◆ AFE_NUM_BANDS_PER_RX

```
#define AFE_NUM_BANDS_PER_RX   2
```

◆ AFE_NUM_BANDS_PER_TX

```
#define AFE_NUM_BANDS_PER_TX   2
```

◆ AFE_NUM_CH_PER_JESD_INSTANCE

```
#define AFE_NUM_CH_PER_JESD_INSTANCE   2
```

◆ AFE_NUM_FB_CHANNELS

```
#define AFE_NUM_FB_CHANNELS   2
```

◆ AFE_NUM_FB_CHANNELS_BITWISE

```
#define AFE_NUM_FB_CHANNELS_BITWISE   0x3
```

◆ AFE_NUM_JESD_INSTANCES

```
#define AFE_NUM_JESD_INSTANCES   2
```

◆ AFE_NUM_RX_CHANNELS

```
#define AFE_NUM_RX_CHANNELS   4
```

◆ AFE_NUM_RX_CHANNELS_BITWISE

```
#define AFE_NUM_RX_CHANNELS_BITWISE   0xf
```

◆ AFE_NUM_SERDES_LANES

```
#define AFE_NUM_SERDES_LANES   8
```

◆ AFE_NUM_TX_CHANNELS

```
#define AFE_NUM_TX_CHANNELS   4
```

◆ AFE_NUM_TX_CHANNELS_BITWISE

```
#define AFE_NUM_TX_CHANNELS_BITWISE   0xf
```

◆ AFE_PAGE_END_ADDR

```
#define AFE_PAGE_END_ADDR   0x19
```

◆ AFE_PAGE_START_ADDR

```
#define AFE_PAGE_START_ADDR   0x10
```

◆ AFE_RX_DSA_MAX_ANA_DSA_DB

```
#define AFE_RX_DSA_MAX_ANA_DSA_DB   25
```

◆ AFE_RX_DSA_MAX_ANA_DSA_INDEX

```
#define AFE_RX_DSA_MAX_ANA_DSA_INDEX   50
```

◆ AFE_RX_DSA_MAX_DIG_DSA_INDEX

```
#define AFE_RX_DSA_MAX_DIG_DSA_INDEX   47
```

◆ AFE_TX_DSA_MAX_ANA_DSA_DB

```
#define AFE_TX_DSA_MAX_ANA_DSA_DB   34
```

◆ AFE_TX_DSA_MAX_ANA_DSA_INDEX

```
#define AFE_TX_DSA_MAX_ANA_DSA_INDEX   29
```

◆ AFE_TX_DSA_MAX_ANA_PLUS_DIG_DSA_DB

```
#define AFE_TX_DSA_MAX_ANA_PLUS_DIG_DSA_DB   39
```

◆ jesdToSerdesLaneMapping

```
#define jesdToSerdesLaneMapping
```

Value:

```
    {                              \
        1, 0, 2, 3, 3, 2, 0, 1   \
    }
```

◆ NULL

```
#define NULL   (0)
```

Typedef Documentation

◆ RetType_e

typedef enum **RET_TYPE RetType_e**

Enumeration Type Documentation

◆ RET_TYPE

enum **RET_TYPE**

Enumerator	
RET_OK	
RET_EXEC_FAIL	

Generated by doxygen 1.8.17

afeCommonMacros.h File Reference

This file contains C Macros for different kinds of operations in the AFE function.

Version 2.1:

More...

```
#include "afe79xxTypes.h"
```

Go to the source code of this file.

Macros

#define **NUM_OF_AFE** 2

Number of AFEs controlled by the host. This should be set by the user. More...

#define **ARRAY_SIZE**(arr) (sizeof(arr) / sizeof(arr[0]))

#define **AFE_PARAMS_VALID**(args)

This C Macro has the operation on what to do when the input parameters to AFE function are invalid. It is not recommended to change its contents. More...

#define **AFE_ID_VALIDITY**()

This C Macro has the operation on what to do when the AFE ID(afeId) is invalid. It is not recommended to change its contents. More...

#define **AFE_SPI_EXEC**(args)

#define **AFE_FUNC_EXEC**(args)

#define **AFE_MACRO_READY_POLL_FAIL**(args)

This C Macro has the operation on what to do when the AFE MCU MACRO(not C Macro) Ready poll fails. It is not recommended to change its contents. More...

#define **AFE_MACRO_DONE_POLL_FAIL**(args)

This C Macro has the operation on what to do when the AFE MCU MACRO(not C Macro) Done poll fails. It is not recommended to change its contents. More...

#define **AFE_MACRO_EXEC_ERROR**(args)

Detailed Description

This file contains C Macros for different kinds of operations in the AFE function.

Version 2.1:

1. Added Documentation.
2. Modified the Macro Execution errors for better handling.

Macro Definition Documentation

◆ AFE_FUNC_EXEC

#define AFE_FUNC_EXEC (args)

Value:

```
    if (RET_OK != (args))                                              \
    {                                                                  \
        afeLogErr("AFE Function Execution failed: %s", #args);         \
        errorStatus |= 1;                                              \
        return RET_EXEC_FAIL;                                          \
    }                                                                  \
    else                                                               \
    {                                                                  \
        afeLogDbg("AFE Function Executed successfully: %s ", #args); \
    }
```

This C Macro handles what to do when a sub-function call fails.

If on fail, RET_EXEC_FAIL is returned, then the function execution is stopped and returns RET_EXEC_FAIL.

If only the command "errorStatus |= 1;" is executed, then the main function execution will continue and the main called function will return RET_EXEC_FAIL.

◆ AFE_ID_VALIDITY

#define AFE_ID_VALIDITY ( )

Value:

```
    if (afeId >= NUM_OF_AFE)                         \
    {                                                \
        afeLogErr("%s", "device ID out of bounds"); \
        return RET_EXEC_FAIL;                        \
    }
```

This C Macro has the operation on what to do when the AFE ID(afeId) is invalid. It is not recommended to change its contents.

◆ AFE_MACRO_DONE_POLL_FAIL

#define AFE_MACRO_DONE_POLL_FAIL (args)

Value:

```
    if (RET_OK != (args))                                           \
    {                                                               \
        afeLogErr("AFE MACRO DONE POLL FAILED: %s", #args);         \
        errorStatus |= 1;                                           \
        return RET_EXEC_FAIL;                                       \
    }                                                               \
    else                                                            \
    {                                                               \
        afeLogDbg("AFE MACRO DONE Successfully Passed: %s", #args); \
    }
```

This C Macro has the operation on what to do when the AFE MCU MACRO(not C Macro) Done poll fails. It is not recommended to change its contents.

◆ AFE_MACRO_EXEC_ERROR

#define AFE_MACRO_EXEC_ERROR (args)

Value:

```
    if (AFE_MACRO_NO_ERROR != (args))                                                                            \
    {                                                                                                            \
        if (((AFE_MACRO_ERROR_IN_OPCODE & (args)) != 0) || ((AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED & (args)) != 0)) \
        {                                                                                                        \
            afeLogErr("AFE MACRO 0x%X: ERROR in OPCODE Received", opcode);                                       \
        }                                                                                                        \
        else if ((AFE_MACRO_ERROR_IN_OPERAND & (args)) != 0)                                                     \
        {                                                                                                        \
            afeLogErr("AFE MACRO 0x%X: ERROR in Operand Received", opcode);                                      \
        }                                                                                                        \
        else if ((AFE_MACRO_ERROR_IN_EXECUTION & (args)) != 0)                                                   \
        {                                                                                                        \
            afeLogErr("AFE MACRO 0x%X: ERROR in Execution.", opcode);                                            \
        }                                                                                                        \
        errorStatus |= 1;                                                                                        \
        return RET_EXEC_FAIL;                                                                                    \
    }                                                                                                            \
    else                                                                                                         \
    {                                                                                                            \
        afeLogDbg("AFE MACRO 0x%X: Executed without Error.", opcode);                                            \
    }
```

This C Macro has the operation on what to do when the AFE MCU MACRO(not C Macro) execution fails.

- If on fail, RET_EXEC_FAIL is returned, then the function execution is stopped and returns RET_EXEC_FAIL.

  If only the command "errorStatus |= 1;" is executed, then the main function execution will continue and the main called function will return RET_EXEC_FAIL.

◆ AFE_MACRO_READY_POLL_FAIL

#define AFE_MACRO_READY_POLL_FAIL (args)

Value:

```
    if (RET_OK != (args))                                            \
    {                                                                \
        afeLogErr("AFE MACRO READY POLL FAILED: %s", #args);         \
        errorStatus |= 1;                                            \
        return RET_EXEC_FAIL;                                        \
    }                                                                \
    else                                                             \
    {                                                                \
        afeLogDbg("AFE MACRO READY Successfully Passed: %s", #args); \
    }
```

This C Macro has the operation on what to do when the AFE MCU MACRO(not C Macro) Ready poll fails. It is not recommended to change its contents.

◆ AFE_PARAMS_VALID

#define AFE_PARAMS_VALID (args)

Value:

```
    if (!(args))                                                         \
    {                                                                    \
        afeLogErr("Parameter did not satisfy the condition: %s", #args); \
        return RET_EXEC_FAIL;                                            \
    }
```

This C Macro has the operation on what to do when the input parameters to AFE function are invalid. It is not recommended to change its contents.

◆ AFE_SPI_EXEC

#define AFE_SPI_EXEC (args)

Value:

```
    if (RET_OK != (args))                                            \
    {                                                                \
        afeLogErr("Execution of function(SPI) failed: %s", #args);   \
        errorStatus |= 1;                                            \
        return RET_EXEC_FAIL;                                        \
    }                                                                \
    else                                                             \
    {                                                                \
        afeLogDbg("Executed function(SPI) successfully: %s", #args); \
    }
```

This C Macro has the operation on what to do when an SPI driver function fails.

There are two recommended recovery ways for this

Option 1. Resolve the SPI issue, reperform the last SPI operation and continue with the execution.

Option 2. Resolve the SPI issue, close all the open pages using the function closeAllPages(afeId). Call the failed function again.

◆ ARRAY_SIZE

#define ARRAY_SIZE (arr) (sizeof(arr) / sizeof(arr[0]))

◆ NUM_OF_AFE

#define NUM_OF_AFE 2

Number of AFEs controlled by the host. This should be set by the user.

afeParameters.h File Reference

Go to the source code of this file.

Classes

struct **afeSystemParamsStruct**

This structure contains the System Parameters used in the intialization script of the AFE.

Some of the system parameters, which are static for a use case, like sampling and interface rates, are captured in this structure, systemParams. This is to prevent passing these redundantly for related functions. For some variables this may act as a state variable to capture current state.

An array of structures of size NUM_OF_AFE, one per each AFE, should be defined in /Afe79xxUser/Src/afeParameters.c similar to the sample provided.

This can be generated for each AFE configuration by running AFE.saveCAfeParamsFile() in Latte after generating the initial configuration.

Version 2.1:

More...

Functions

uint8_t **getSystemParam** (uint32_t afeId, struct **afeSystemParamsStruct** *pstParam)

Variables

struct **afeSystemParamsStruct** **systemParams** [**NUM_OF_AFE**]

Function Documentation

◆ getSystemParam()

```
uint8_t getSystemParam ( uint32_t                          afeId,
                         struct afeSystemParamsStruct *    pstParam
                       )
```

Variable Documentation

◆ systemParams

struct **afeSystemParamsStruct** systemParams[**NUM_OF_AFE**]

systemParams is the Array of structures contains the System Parameters used in the intialization script for each AFE.

Some of the system parameters, which are static for a use case, like sampling and interface rates, are captured in this structure, systemParams. This is to prevent passing these redundantly for related functions. For some variables this may act as a state variable to capture current state.

This can be generated for each AFE configuration by running AFE.saveCAfeParamsFile() in Latte after generating the initial configuration.

afeSystemParamsStruct Struct Reference

This structure contains the System Parameters used in the intialization script of the AFE.

Some of the system parameters, which are static for a use case, like sampling and interface rates, are captured in this structure, systemParams. This is to prevent passing these redundantly for related functions. For some variables this may act as a state variable to capture current state.

An array of structures of size NUM_OF_AFE, one per each AFE, should be defined in /Afe79xxUser/Src/afeParameters.c similar to the sample provided.

This can be generated for each AFE configuration by running AFE.saveCAfeParamsFile() in Latte after generating the initial configuration.

Version 2.1:

More...

```
#include <afeParameters.h>
```

Public Attributes

uint32_t	**X** Multiplier Constant. More...
uint8_t	**numTxNCO** Number of TX NCOs. More...
uint8_t	**numRxNCO** Number of RX NCOs. More...
uint8_t	**numFbNCO** Number of FB NCOs. More...
float	**FRef** Input Reference Clock (MHz) More...
float	**FadcRx** RX ADC Sampling clock. (MHz) More...
float	**FadcFb** FB ADC Sampling clock. (MHz) More...
float	**Fdac** DAC Sampling clock. (MHz) More...
uint8_t	**useSpiSysref** When this is 0, the Pin based Sysref will be used by the AFE. When this is set to 1, AFE uses internal Sysref override in AFE and pin sysref is not used. This can be used in cases where there is no need for deterministic latency or phase consistency. More...
uint8_t	**ncoFreqMode** NCO Frequency Mode. 0- FCW mode. 1-1KHz mode. More...
uint8_t	**halfRateModeRx** [2] Enabling Half Rate Mode for RX [AB,CD]. This will make the sampling rate half of FadcRx. More...
uint8_t	**halfRateModeFb** [2] Enabling Half Rate Mode for FB [AB,CD]. This will make the sampling rate half of FadcFb. More...
uint8_t	**halfRateModeTx** [2] Enabling Half Rate Mode for TX [AB,CD]. This will make the sampling rate half of Fdac. More...
uint8_t	**syncLoopBack** 0- Software JESD Sync 1- Hardware JESD Sync loopback More...
uint8_t	**ddcFactorRx** [4] DDC decimation factor for RX [A,B,C,D]. FadcRx/ddcFactorRx for the channel will be output data rate. More...
float	**rxNco** [2][4][2] RX NCO Frequencies in MHz. rxNco[NCO Number][Channel Number][Band Number]. More...
uint8_t	**numBandsRx** [4] Number of bands per channel for RX A, B, C, D. 0-Single Band. 1-Dual Band. More...

uint8_t **ddcFactorFb** [2]

DDC decimation factor for FB [AB, CD]. FadcFb/ddcFactorFb for the channel will be output data rate. More...

float **fbNco** [4][2]

FB NCO Frequencies in MHz. fbNco[Channel Number][NCO Number]. More...

uint8_t **ducFactorTx** [4]

DUC interpolation factor for TX [A,B, C, D]. Fdac/ducFactorTx for the channel will be output data rate. More...

float **txNco** [2][4][2]

TX NCO Frequencies in MHz. txNco[NCO Number][Channel Number][Band Number]. More...

uint8_t **numBandsTx** [4]

Number of bands per channel for TX A, B, C, D. 0-Single Band. 1-Dual Band. More...

uint8_t **enableDacInterleavedMode**

Operates DAC in interleaved mode when this is set to 1. More...

uint8_t **txToFbMode**

uint32_t **chipId**

Chip ID of the Device. More...

uint8_t **chipVersion**

Chip Version of the device. More...

uint8_t **agcMode**

uint8_t **bigStepAttkEn** [4]

Per RX channel Big Step Attack Detector. More...

uint8_t **smallStepAttkEn** [4]

Per RX channel Small Step Attack Detector. More...

uint8_t **powerAttkEn** [4]

Per RX channel Power Attack Detector. More...

uint8_t **bigStepDecEn** [4]

Per RX channel Big Step Decay Detector. More...

uint8_t **smallStepDecEn** [4]

Per RX channel Small Step Decay Detector. More...

uint8_t **powerDecEn** [4]

Per RX channel Power Decay Detector. More...

uint8_t **bigStepAttkThresh** [4]

Per RX channel Big Step Attack Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint8_t **smallStepAttkThresh** [4]

Per RX channel Small Step Attack Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint8_t **powerAttkThresh** [4]

Per RX channel Power Attack Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint8_t **bigStepDecThresh** [4]

Per RX channel Big Step Decay Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint8_t **smallStepDecThresh** [4]

Per RX channel Small Step Decay Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint8_t **powerDecThresh** [4]

Per RX channel Power Decay Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9. More...

uint32_t **bigStepAttkWinLen** [4]

ADC Digital Detector Window Length configuration per RX channel. Window length for the detectors is the corresponding WinLen value *10ns.

bigStepAttkWinLen is the window length of all big step detectors. More...

uint32_t **miscStepAttkWinLen** [4]

miscStepAttkWinLen is Window Length configuration per RX channel used for all other attack detectors. Window length for the detectors is the corresponding WinLen value *10ns. More...

uint32_t **decayWinLen** [4]

decayWinLen is Window Length configuration per RX channel common for all the decay detectors. Window length for the detectors is the corresponding WinLen value *10ns. More...

uint8_t **jesdProtocol**

JESD Protocol. JESD Protocol for ADC/DAC JESD instance [0, 1]. 0- 204B.

2- 204C 64/66.

3- 204B 64/80. More...

uint8_t **spiInUseForPllAccess**

SPI used to access PLL Pages.

1- SPIA.

2-SPIB. More...

Detailed Description

This structure contains the System Parameters used in the intialization script of the AFE.

Some of the system parameters, which are static for a use case, like sampling and interface rates, are captured in this structure, systemParams. This is to prevent passing these redundantly for related functions. For some variables this may act as a state variable to capture current state.

An array of structures of size NUM_OF_AFE, one per each AFE, should be defined in /Afe79xxUser/Src/afeParameters.c similar to the sample provided.

This can be generated for each AFE configuration by running AFE.saveCAfeParamsFile() in Latte after generating the initial configuration.

Version 2.1:

1. Added documentation.

Member Data Documentation

◆ agcMode

uint8_t afeSystemParamsStruct::agcMode

Mode of operation of the AGC.

0- disabled

1- Internal AGC

2- External AGC SPI control

3- External AGC 3-Pin control

4- External AGC 8-Pin control

◆ bigStepAttkEn

uint8_t afeSystemParamsStruct::bigStepAttkEn[4]

Per RX channel Big Step Attack Detector.

◆ bigStepAttkThresh

uint8_t afeSystemParamsStruct::bigStepAttkThresh[4]

Per RX channel Big Step Attack Threshold.
. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ bigStepAttkWinLen

uint32_t afeSystemParamsStruct::bigStepAttkWinLen[4]

ADC Digital Detector Window Length configuration per RX channel. Window length for the detectors is the corresponding WinLen value *10ns. bigStepAttkWinLen is the window length of all big step detectors.

◆ bigStepDecEn

uint8_t afeSystemParamsStruct::bigStepDecEn[4]

Per RX channel Big Step Decay Detector.

◆ bigStepDecThresh

uint8_t afeSystemParamsStruct::bigStepDecThresh[4]

Per RX channel Big Step Decay Threshold.
. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ chipId

uint32_t afeSystemParamsStruct::chipId

Chip ID of the Device.

◆ chipVersion

uint8_t afeSystemParamsStruct::chipVersion

Chip Version of the device.

◆ ddcFactorFb

uint8_t afeSystemParamsStruct::ddcFactorFb[2]

DDC decimation factor for FB [AB, CD]. FadcFb/ddcFactorFb for the channel will be output data rate.

◆ ddcFactorRx

uint8_t afeSystemParamsStruct::ddcFactorRx[4]

DDC decimation factor for RX [A,B,C,D]. FadcRx/ddcFactorRx for the channel will be output data rate.

◆ decayWinLen

uint32_t afeSystemParamsStruct::decayWinLen[4]

decayWinLen is Window Length configuration per RX channel common for all the decay detectors. Window length for the detectors is the corresponding WinLen value *10ns.

◆ ducFactorTx

uint8_t afeSystemParamsStruct::ducFactorTx[4]

DUC interpolation factor for TX [A,B, C, D]. Fdac/ducFactorTx for the channel will be output data rate.

◆ enableDacInterleavedMode

uint8_t afeSystemParamsStruct::enableDacInterleavedMode

Operates DAC in interleaved mode when this is set to 1.

◆ FadcFb

float afeSystemParamsStruct::FadcFb

FB ADC Sampling clock. (MHz)

◆ FadcRx

float afeSystemParamsStruct::FadcRx

RX ADC Sampling clock. (MHz)

◆ fbNco

float afeSystemParamsStruct::fbNco[4][2]

FB NCO Frequencies in MHz. fbNco[Channel Number][NCO Number].

◆ Fdac

float afeSystemParamsStruct::Fdac

DAC Sampling clock. (MHz)

◆ FRef

float afeSystemParamsStruct::FRef

Input Reference Clock (MHz)

◆ halfRateModeFb

uint8_t afeSystemParamsStruct::halfRateModeFb[2]

Enabling Half Rate Mode for FB [AB,CD]. This will make the sampling rate half of FadcFb.

◆ halfRateModeRx

uint8_t afeSystemParamsStruct::halfRateModeRx[2]

Enabling Half Rate Mode for RX [AB,CD]. This will make the sampling rate half of FadcRx.

◆ halfRateModeTx

uint8_t afeSystemParamsStruct::halfRateModeTx[2]

Enabling Half Rate Mode for TX [AB,CD]. This will make the sampling rate half of Fdac.

◆ jesdProtocol

uint8_t afeSystemParamsStruct::jesdProtocol

JESD Protocol. JESD Protocol for ADC/DAC JESD instance [0, 1]. 0- 204B.

2- 204C 64/66.

3- 204B 64/80.

◆ miscStepAttkWinLen

uint32_t afeSystemParamsStruct::miscStepAttkWinLen[4]

miscStepAttkWinLen is Window Length configuration per RX channel used for all other attack detectors. Window length for the detectors is the corresponding WinLen value *10ns.

◆ ncoFreqMode

uint8_t afeSystemParamsStruct::ncoFreqMode

NCO Frequency Mode. 0- FCW mode. 1-1KHz mode.

◆ numBandsRx

uint8_t afeSystemParamsStruct::numBandsRx[4]

Number of bands per channel for RX A, B, C, D. 0-Single Band. 1-Dual Band.

◆ numBandsTx

uint8_t afeSystemParamsStruct::numBandsTx[4]

Number of bands per channel for TX A, B, C, D. 0-Single Band. 1-Dual Band.

◆ numFbNCO

uint8_t afeSystemParamsStruct::numFbNCO

Number of FB NCOs.

◆ numRxNCO

uint8_t afeSystemParamsStruct::numRxNCO

Number of RX NCOs.

◆ numTxNCO

uint8_t afeSystemParamsStruct::numTxNCO

Number of TX NCOs.

◆ powerAttkEn

uint8_t afeSystemParamsStruct::powerAttkEn[4]

Per RX channel Power Attack Detector.

◆ powerAttkThresh

uint8_t afeSystemParamsStruct::powerAttkThresh[4]

Per RX channel Power Attack Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ powerDecEn

uint8_t afeSystemParamsStruct::powerDecEn[4]

Per RX channel Power Decay Detector.

◆ powerDecThresh

uint8_t afeSystemParamsStruct::powerDecThresh[4]

Per RX channel Power Decay Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ rxNco

float afeSystemParamsStruct::rxNco[2][4][2]

RX NCO Frequencies in MHz. rxNco[NCO Number][Channel Number][Band Number].

◆ smallStepAttkEn

uint8_t afeSystemParamsStruct::smallStepAttkEn[4]

Per RX channel Small Step Attack Detector.

◆ smallStepAttkThresh

uint8_t afeSystemParamsStruct::smallStepAttkThresh[4]

Per RX channel Small Step Attack Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ smallStepDecEn

uint8_t afeSystemParamsStruct::smallStepDecEn[4]

Per RX channel Small Step Decay Detector.

◆ smallStepDecThresh

uint8_t afeSystemParamsStruct::smallStepDecThresh[4]

Per RX channel Small Step Decay Threshold.

. This -threshValue/4 is the threshold in dbfs set. That is, to set the a threshold of -2.25dbfs, this value should be 2.25*4=9.

◆ spiInUseForPllAccess

uint8_t afeSystemParamsStruct::spiInUseForPllAccess

SPI used to access PLL Pages.

1- SPIA.

2-SPIB.

◆ syncLoopBack

uint8_t afeSystemParamsStruct::syncLoopBack

0- Software JESD Sync 1- Hardware JESD Sync loopback

◆ txNco

float afeSystemParamsStruct::txNco[2][4][2]

TX NCO Frequencies in MHz. txNco[NCO Number][Channel Number][Band Number].

◆ txToFbMode

uint8_t afeSystemParamsStruct::txToFbMode

Sets the Mux mode for the Pin based FB DSA control.

0 -Single Fb Mode FB AB. Only FBAB DSA should be controlled by pins.

1 -Single Fb Mode FB CD. Only FBCD DSA should be controlled by pins.

2- Dual Fb_Mode. LSB 2 pins Control to FBAB DSA and MSB 2 pins control FBCD DSA.

◆ useSpiSysref

uint8_t afeSystemParamsStruct::useSpiSysref

When this is 0, the Pin based Sysref will be used by the AFE. When this is set to 1, AFE uses internal Sysref override in AFE and pin sysref is not used. This can be used in cases where there is no need for deterministic latency or phase consistency.

◆ X

uint32_t afeSystemParamsStruct::X

Multiplier Constant.

The documentation for this struct was generated from the following file:

- Afe79xx/Include/**afeParameters.h**

Generated by doxygen 1.8.17

agc.h File Reference

Go to the source code of this file.

Functions

uint8_t	**agcStateControlConfig** (uint8_t afeId, uint8_t chNo, uint16_t agcstate) AGC State Control Macro. More...
uint8_t	**agcDigDetConfig** (uint8_t afeId, uint8_t chNo, uint8_t bigStepAttkEn, uint8_t smallStepAttkEn, uint8_t bigStepDecEn, uint8_t smallStepDecEn, uint8_t powerAttkEn, uint8_t powerDecEn, uint8_t bigStepAttkThresh, uint8_t smallStepAttkThresh, uint8_t bigStepDecThresh, uint8_t smallStepDecThresh, uint8_t powerAttkThresh, uint8_t powerDecThresh) ADC Digital Detector Threshold configuration. More...
uint8_t	**agcDigDetTimeConstantConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkWinLen, uint32_t miscStepAttkWinLen, uint32_t decayWinLen) ADC Digital Detector Window Length configuration. More...
uint8_t	**agcDigDetAbsoluteNumCrossingConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkNumHits, uint32_t smallStepAttkNumHits, uint32_t bigStepDecNumHits, uint32_t smallStepDecNumHits) ADC Digital Detector Absolute NumHits configuration. More...
uint8_t	**agcDigDetRelativeNumCrossingConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkNumHits, uint32_t smallStepAttkNumHits, uint32_t bigStepDecNumHits, uint32_t smallStepDecNumHits) ADC Digital Detector Relative NumHits configuration. More...
uint8_t	**externalAgcConfig** (uint8_t afeId, uint8_t chNo, uint16_t pin0sel, uint16_t pin1sel, uint16_t pin2sel, uint16_t pin3sel, uint8_t pkDetPinLsbSel, uint8_t pulseExpansionCount, uint8_t noLsbsToSend) External AGC Configuration. More...
uint8_t	**minMaxDsaAttnConfig** (uint8_t afeId, uint8_t chNo, uint8_t minDsaAttn, uint8_t maxDsaAttn) Internal AGC Min-Max Attenuation Configuration. More...
uint8_t	**agcGainStepSizeConfig** (uint8_t afeId, uint8_t chNo, uint8_t bigStepAttkStepSize, uint8_t smallStepAttkStepSize, uint8_t bigStepDecayStepSize, uint8_t smallStepDecayStepSize) Internal AGC Gain-Step Configuration. More...
uint8_t	**internalAgcConfig** (uint8_t afeId, uint8_t chNo, uint8_t tdd_freeze_agc, uint16_t blank_time_extcomp, uint8_t en_agcfreeze_pin, uint8_t extCompControlEn) Internal AGC Configuration. More...
uint8_t	**rfAnalogDetConfig** (uint8_t afeId, uint8_t chNo, uint8_t rfdeten, uint8_t rfDetMode, uint8_t rfDetNumHitsMode, uint32_t rfdetnumhits, uint8_t rfdetThreshold, uint8_t rfdetstepsize) Analog RF Detector Configuration. More...
uint8_t	**extLnaConfig** (uint8_t afeId, uint8_t chNo, uint8_t singleDualBandMode, uint8_t lnaGainMargin, uint8_t enBandDet, uint8_t tapOffPoint) External LNA Configuration. More...
uint8_t	**extLnaGainConfig** (uint8_t afeId, uint8_t chNo, uint16_t lnaGainB0, uint16_t lnaPhaseB0, uint16_t lnaGainB1, uint16_t lnaPhaseB1) External LNA Fixed Gain Configuration. More...

uint8_t **alcConfig** (uint8_t afeId, uint8_t chNo, uint8_t alcMode, uint8_t totalGainRange, uint8_t minAttnAlc, uint8_t useMinAttnAgc)
ALC Configuration. More...

uint8_t **fltPtConfig** (uint8_t afeId, uint8_t chNo, uint8_t fltPtMode, uint8_t fltPtFmt)
Floating Point Configuration. More...

uint8_t **coarseFineConfig** (uint8_t afeId, uint8_t chNo, uint8_t stepSize, uint8_t nBitIndex, uint8_t indexInvert, uint8_t indexSwapIQ, uint8_t sigBackOff, uint8_t gainChangeIndEn)
Coarse-Fine Mode Configuration. More...

Function Documentation

◆ agcDigDetAbsoluteNumCrossingConfig()

```
uint8_t agcDigDetAbsoluteNumCrossingConfig ( uint8_t   afeId,
                                             uint8_t   chNo,
                                             uint32_t  bigStepAttkNumHits,
                                             uint32_t  smallStepAttkNumHits,
                                             uint32_t  bigStepDecNumHits,
                                             uint32_t  smallStepDecNumHits
                                           )
```

ADC Digital Detector Absolute NumHits configuration.

ADC Digital Detector Absolute NumHits configuration. This represents the exact number of crossings threshold and this may need to be adjusted whenever window length is reconfigured to ensure the NumHits threshold will be lower than the Window Length configuration.
Note that only this function or agcDigDetAbsoluteNumCrossingConfig should be used. Both shouldn't be called.
agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkNumHits	Absolute Number of Threshold crossing hits threshold of big step attack detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
smallStepAttkNumHits	Absolute Number of Threshold crossing hits threshold of Small step attack detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
bigStepDecNumHits	Absolute Number of Threshold crossing hits threshold of big step decay detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
smallStepDecNumHits	Absolute Number of Threshold crossing hits threshold of small step decay detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.

Returns

Returns if the function execution passed or failed.

◆ agcDigDetConfig()

```
uint8_t agcDigDetConfig ( uint8_t  afeId,
                          uint8_t  chNo,
                          uint8_t  bigStepAttkEn,
                          uint8_t  smallStepAttkEn,
                          uint8_t  bigStepDecEn,
                          uint8_t  smallStepDecEn,
                          uint8_t  powerAttkEn,
                          uint8_t  powerDecEn,
                          uint8_t  bigStepAttkThresh,
                          uint8_t  smallStepAttkThresh,
                          uint8_t  bigStepDecThresh,
                          uint8_t  smallStepDecThresh,
                          uint8_t  powerAttkThresh,
                          uint8_t  powerDecThresh
                        )
```

ADC Digital Detector Threshold configuration.

Enables or disables the detectors. agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkEn	0 disables and 1 enables the corresponding detector.
smallStepAttkEn	0 disables and 1 enables the corresponding detector.
bigStepDecEn	0 disables and 1 enables the corresponding detector.
smallStepDecEn	0 disables and 1 enables the corresponding detector.
powerAttkEn	0 disables and 1 enables the corresponding detector.
powerDecEn	0 disables and 1 enables the corresponding detector.
bigStepAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
smallStepAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
bigStepDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
smallStepDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
powerAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
powerDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).

Returns

Returns if the function execution passed or failed.

◆ agcDigDetRelativeNumCrossingConfig()

```
uint8_t agcDigDetRelativeNumCrossingConfig ( uint8_t   afeId,
                                             uint8_t   chNo,
                                             uint32_t  bigStepAttkNumHits,
                                             uint32_t  smallStepAttkNumHits,
                                             uint32_t  bigStepDecNumHits,
                                             uint32_t  smallStepDecNumHits
                                           )
```

ADC Digital Detector Relative NumHits configuration.

ADC Digital Detector Relative NumHits configuration. This specifies the threshold relative to the window length of the corresponding detector. The advantage of this approach is, this will scale automatically when the window length is changed.
Note that only this function or agcDigDetAbsoluteNumCrossingConfig should be used. Both shouldn't be called.
agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkNumHits	Relative Number of Threshold crossing hits threshold of big step attack detectors. The window counter threhold is (value*bigStepAttkWinLen/2^16). Range is 0-0xffff.
smallStepAttkNumHits	Relative Number of Threshold crossing hits threshold of Small step attack detectors. The window counter threhold is (value*smallStepAttkNumHits/2^16).Range is 0-0xffff.
bigStepDecNumHits	Relative Number of Threshold crossing hits threshold of big step decay detectors. The window counter threhold is (value*bigStepDecNumHits/2^16). Range is 0-0xffff.
smallStepDecNumHits	Relative Number of Threshold crossing hits threshold of small step decay detectors. The window counter threhold is (value*smallStepDecNumHits/2^16). Range is 0-0xffff.

Returns

Returns if the function execution passed or failed.

◆ agcDigDetTimeConstantConfig()

```
uint8_t agcDigDetTimeConstantConfig ( uint8_t   afeId,
                                      uint8_t   chNo,
                                      uint32_t  bigStepAttkWinLen,
                                      uint32_t  miscStepAttkWinLen,
                                      uint32_t  decayWinLen
                                    )
```

ADC Digital Detector Window Length configuration.

Configures the Window Length (or time constant) of the detectors. agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkWinLen	Window length of big step attack and RF Analog (also called Customer RF) detectors. Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.
miscStepAttkWinLen	Window length of all other attack detectors. Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.
decayWinLen	Window Length of all the decay detectors.Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.

Returns

Returns if the function execution passed or failed.

◆ agcGainStepSizeConfig()

```
uint8_t agcGainStepSizeConfig ( uint8_t  afeId,
                                uint8_t  chNo,
                                uint8_t  bigStepAttkStepSize,
                                uint8_t  smallStepAttkStepSize,
                                uint8_t  bigStepDecayStepSize,
                                uint8_t  smallStepDecayStepSize
                              )
```

Internal AGC Gain-Step Configuration.

Configures the Step size of the AGC (DSA index by which to change on detector triggering).
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
smallStepAttkStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
bigStepDecayStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
smallStepDecayStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)

Returns

Returns if the function execution passed or failed.

◆ agcStateControlConfig()

```
uint8_t agcStateControlConfig ( uint8_t   afeId,
                                uint8_t   chNo,
                                uint16_t  agcstate
                              )
```

AGC State Control Macro.

Controls the state of the AGC

Parameters

afeId AFE ID

chNo Bit wise channel select

Bit0 for RXA

Bit1 for RXB

Bit2 for RXC

Bit3 for RXD

agcstate Bit wise parameter controlling the state of the AGC. Making a bit 1 does the corresponding operation.

Bit 0: Start Internal AGC with entire configuration redone

Bit 1: Freeze the Internal AGC loop

Bit 2: Unfreeze the Internal AGC loop (takes effect only if the loop is already in freeze)

Bit 3: Disable Internal AGC loop

Bit 4: ALC Block enable

Bit 5: ALC Block disable

Bit 6: External AGC enable

Bit 7: External AGC disable

Bit 8: Restart the Internal AGC. (Step1: Disable Internal AGC, Step2:Enable Internal AGC)

Bit 9: Restart ALC(Step1: Disable ALC, Step2:Enable ALC)

Bit 10: Restart external AGC(Step1: Disable external AGC, Step2:Enable external AGC)

All the bits should not be set together.For example, the enables and disables should not be set together. Invalid combinations include:

1. No other AGC related bit should be enabled when AGC enable is 1.
2. Enable and disable of the ALC should not be set at the same time.
3. Enable and disable of the AGC should not be set at the same time.

Returns

Returns if the function execution passed or failed.

◆ alcConfig()

```
uint8_t alcConfig ( uint8_t  afeId,
                    uint8_t  chNo,
                    uint8_t  alcMode,
                    uint8_t  totalGainRange,
                    uint8_t  minAttnAlc,
                    uint8_t  useMinAttnAgc
                  )
```

ALC Configuration.

Configures ALC. Note that this only informs the MCU of the mode. agcStateControlConfig function should be called with appropriate parameter after this to enable or disable it. agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
alcMode	ALC Mode #0: Floatingpoint #2: coarsefineI #3: coarsefineIQ #4: coarsefineALCpin #5: inputALC
totalGainRange	Total gain range used by ALC for gain compensation. should be <AFE_RX_DSA_MAX_ANA_DSA_DB
minAttnAlc	Minimum Attenuation used by ALC for compensation when useMinAttnAgc = 0. should be <32. Value doesn;t matter when useMinAttnAgc=1
useMinAttnAgc	Configure the Min Attenuation Mode. 0: Use minAttnAlc for minimum attenuation for which compensation is required. 1: Enable ALC to use minimum attenuation from AGC for which compensation is required.

Returns

Returns if the function execution passed or failed.

◆ coarseFineConfig()

```
uint8_t coarseFineConfig ( uint8_t  afeId,
                           uint8_t  chNo,
                           uint8_t  stepSize,
                           uint8_t  nBitIndex,
                           uint8_t  indexInvert,
                           uint8_t  indexSwapIQ,
                           uint8_t  sigBackOff,
                           uint8_t  gainChangeIndEn
                         )
```

Coarse-Fine Mode Configuration.

Configures Coarse-Fine Mode related parameters. This needs to be called only when alcMode in alcConfig is set to coarse-fine mode. Note that this only informs the MCU of the mode.
agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
stepSize	Choose the coarse step size. Appropriate value has to be chosen which can represent the complete attenuation range of operation. 0x00 → 0 dB 0x01 → 1 dB 0x02 → 2 dB 0x03 → 3 dB 0x04 → 4 dB 0x05 → 5 dB 0x06 → 6 dB 0x08 → 8 dB
nBitIndex	Choose the number of bits of coarse index. Supported Values are 0,2,3,4.
indexInvert	Coarse Index Invert. If this value is 0: coarse index is transmitted as is. 1: (15-coarse index) is transmitted
indexSwapIQ	Coarse Index Swap. If to swap coarse index on I and Q. 0: LSB on I, MSB on Q 1: MSB on I, LSB on Q
sigBackOff	This is the signal back-off, the offset attenuation applied. (in dB) This should be less than totalGainRange.
gainChangeIndEn	Applicable only when nBitIndex is 3. If this is set, in the bit-4 indicates if the DSA changed. Otherwise, 0 will be sent.

Returns

Returns if the function execution passed or failed.

◆ externalAgcConfig()

```
uint8_t externalAgcConfig ( uint8_t   afeId,
                            uint8_t   chNo,
                            uint16_t  pin0sel,
                            uint16_t  pin1sel,
                            uint16_t  pin2sel,
                            uint16_t  pin3sel,
                            uint8_t   pkDetPinLsbSel,
                            uint8_t   pulseExpansionCount,
                            uint8_t   noLsbsToSend
                          )
```

External AGC Configuration.

Configures the External AGC.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
pin0sel	Pin0/I_BIT0 configuration. Determines what detectors come out. It can be configured to carry ORed combination of selected bits. Setting a particular bit gets the detector on to the corresponding pin/LSB. Bit 14: OVR Bit Bit 13: Band 0 power detector Bit 12: Band 0 peak detector Bit 11: RF detector Bit 10: Band 1 power detector Bit 9: Band 1 peak detector Bit 8: Reserved Bit 7: Digital big step attack Bit 6: Digital small step attack Bit 5: Digital big step decay Bit 4: Digital small step decay Bit 3: Dig power attack Bit 2: Dig power decay Bit 1: Reserved (0) Bit 0: Reserved (0)
pin1sel	Pin1/I_BIT1 configuration. Determines what detectors come out. Description same as pin1Sel.
pin2sel	Pin2/Q_BIT0 configuration. Determines what detectors come out. Description same as pin2Sel.
pin3sel	Pin3/Q_BIT1 configuration. Determines what detectors come out. Description same as pin30Sel.
pulseExpansionCount	Pulse Expansion Count. This value here is in steps of 10 ns. This pulseExpansionCount*10ns is the pulse width.Supported Range: 0-0xff
pkDetPinLsbSel	Determines whether to send detector data on LSB in External AGC mode. 0: send on Pin 1: send on Pin and LSB
noLsbsToSend	0-Send only on Bits 0 of I and Q. 1- Send on both Bits 0 and 1.

Returns

Returns if the function execution passed or failed.

◆ extLnaConfig()

```
uint8_t extLnaConfig ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  singleDualBandMode,
                       uint8_t  lnaGainMargin,
                       uint8_t  enBandDet,
                       uint8_t  tapOffPoint
                     )
```

External LNA Configuration.

Configures External LNA Configuration.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
singleDualBandMode	Whether to use Single LNA control or dual LNA control in dual-band configuration. 0: Single LNA control, 1: Dual LNA control
lnaGainMargin	LNA gain margin (this value is in dB scale where 1 LSB= 0.5 dB) LNA reenable will happen when Current DSA Attenuation ≤ Maximum DSA Attenuation - LNA Gain - LNA Gain Margin in Single LNA Control Mode. Not Applicable in Dual LNA Control
enBandDet	0: Disable Band Detectors 1: Enable band detectors Applicable only when Dual LNA control is enabled.
tapOffPoint	Band Detector Bandwidth Selection (Applicable only when dual LNA control and band detectors are enabled) 0: Higher bandwidth 1: Output bandwidth

Returns

Returns if the function execution passed or failed.

◆ extLnaGainConfig()

```
uint8_t extLnaGainConfig ( uint8_t   afeId,
                           uint8_t   chNo,
                           uint16_t  lnaGainB0,
                           uint16_t  lnaPhaseB0,
                           uint16_t  lnaGainB1,
                           uint16_t  lnaPhaseB1
                         )
```

External LNA Fixed Gain Configuration.

External LNA Fixed Gain Configuration.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
lnaGainB0	LNA Gain for Band 0 in dB. (1LSB=1/32dB). Supported Range 0-0x7ff.
lnaPhaseB0	LNA Phase for Band 0 in degrees. (1LSB=360/1024 degrees). Supported Range 0-0x3ff.
lnaGainB1	LNA Gain for Band 1 in dB. Valid only in dual band operation with dual LNA control enabled. (1LSB=1/32dB). Supported Range 0-0x7ff.
lnaPhaseB1	LNA Phase for Band 1 in degrees. Valid only in dual band operation with dual LNA control enabled. (1LSB=360/1024 degrees). Supported Range 0-0x3ff.

Returns

Returns if the function execution passed or failed.

◆ fltPtConfig()

```
uint8_t fltPtConfig ( uint8_t  afeId,
                      uint8_t  chNo,
                      uint8_t  fltPtMode,
                      uint8_t  fltPtFmt
                    )
```

Floating Point Configuration.

Configures Floating Point Mode related parameters. This needs to be called only when alcMode in alcConfig is set to floating point mode. Note that this only informs the MCU of the mode.
agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fltPtMode	ALC Floating Point Mode. Sets whether to send MSB of mantissa always in Floating Point mode of ALC. 0: If exponent > 0, do not send MSB 1: Send MSB always
fltPtFmt	Floating Point Format. Number of Mantissa and Exponent bits to be used in floating point mode of ALC 0: 2 bit exponent , 13 bit mantissa and 1 bit sign 1: 3 bit exponent, 12 bit mantissa and 1 bit sign 2: 4 bit exponent, 11 bit mantissa and 1 bit sign

Returns

Returns if the function execution passed or failed.

◆ internalAgcConfig()

```
uint8_t internalAgcConfig ( uint8_t   afeId,
                            uint8_t   chNo,
                            uint8_t   tdd_freeze_agc,
                            uint16_t  blank_time_extcomp,
                            uint8_t   en_agcfreeze_pin,
                            uint8_t   extCompControlEn
                          )
```

Internal AGC Configuration.

Configures the internal AGC related settings.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
tdd_freeze_agc	Whether to reset or freeze the attack detectors during the OFF period of TDD. 0: Reset 1: Freeze
blank_time_extcomp	Blanking Time for all Detectors when external component (LNA or DVGA) gain change. This is interpreted as number of clocks of FadcRx/8. Supported range: 0-0xffff
en_agcfreeze_pin	Enable or Disable pin based AGC freeze. 0: Disable 1: Enable
extCompControlEn	External Component control to enable. 0x00: Neither of the controls are active 0x01: External LNA control is active 0x02: External DVGA control is active Others: invalid

Returns

Returns if the function execution passed or failed.

◆ minMaxDsaAttnConfig()

```
uint8_t minMaxDsaAttnConfig ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  minDsaAttn,
                              uint8_t  maxDsaAttn
                            )
```

Internal AGC Min-Max Attenuation Configuration.

Configures the Minimum and Maximum DSA index between which the internal AGC operates. This is a dynamic Macro and AGC state macro needn't be called after this.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
minDsaAttn	Minimum DSA index. (1LSB = 0.5dB)
maxDsaAttn	Maximum DSA index. (1LSB = 0.5dB)

Returns

Returns if the function execution passed or failed.

◆ rfAnalogDetConfig()

```
uint8_t rfAnalogDetConfig ( uint8_t   afeId,
                            uint8_t   chNo,
                            uint8_t   rfdeten,
                            uint8_t   rfDetMode,
                            uint8_t   rfDetNumHitsMode,
                            uint32_t  rfdetnumhits,
                            uint8_t   rfdetThreshold,
                            uint8_t   rfdetstepsize
                          )
```

Analog RF Detector Configuration.

Analog RF Detector Configuration. agcStateControlConfig function should be called with internal or external AGC enable after this function call to update the configuration.

Parameters

afeId AFE ID

chNo Bit wise channel select
Bit0 for RXA
Bit1 for RXB
Bit2 for RXC
Bit3 for RXD

rfdeten Use RF Analog detector for internal AGC. 0-Disable. 1-Enable

rfDetMode Mode to use the RF Analog Detector Mode in AGC.
0: extAgc: Use RF Analog detector in External AGC.
1: bigStepAtk : Use RF Analog detector as very big step attack in internal AGC.
2: lnaBypass : Use RF Analog detector for external LNA bypass in internal AGC.

rfDetNumHitsMode Mode of input of the rfDetNumHitsMode. 0- Absolute. 1- Relative

rfdetnumhits When rfDetNumHitsMode=0, this is the absolute Number of times signal crosses threshold above which attack is declared. This detector operates at FadcRx rate. Supported Range: <2^32.
When rfDetNumHitsMode=1, this is the relative Number of times signal crosses threshold above which attack is declared. The actual threshold is floor(rfdetnumhits*bigStepAttkWinLen/2^32). Supported Range: <2^32.

rfdetThreshold RF detect Threshold in dBm (for rfDetMode= 0 or 2) and in dbfs (for rfDetMode =1)

rfdetstepsize Step Size of big step attack in dB. Valid only when rfDetMode=1

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

baseFunc.h File Reference

Go to the source code of this file.

Functions

uint8_t **dev_spi_write** (uint8_t afeId, uint16_t addr, uint8_t data)

AFE SPI Write driver function. More...

uint8_t **dev_spi_read** (uint8_t afeId, uint16_t addr, uint8_t *readVal)

AFE SPI read driver function. More...

uint8_t	**wait** (uint32_t wait_s) Wait in Seconds. More...
uint8_t	**waitMs** (uint32_t wait_ms) Wait in milli Seconds. More...
void	**afeLogmsg** (uint32_t level, const char *pcLogFmt,...) Logging function. More...
void	**setAfeLogLvl** (uint32_t level) Set the AFE Log Level. More...
uint32_t	**getAfeLogLvl** () Get the AFE Log Level. More...
uint8_t	**giveSingleSysrefPulse** (uint8_t afeId) AFE single shot Pin Sysref. More...
uint8_t	**giveAfeAdcInput** (uint8_t afeId, uint8_t rxChNo, uint8_t bandNo) Give ADC input for factory Calibration. More...
uint8_t	**connectAfeTxToFb** (uint8_t afeId, uint8_t txChNo, uint8_t fbChNo, uint8_t bandNo) Connect TX Output to FB for factory Calibration. More...

Function Documentation

◆ afeLogmsg()

```
void afeLogmsg ( uint32_t      level,
                 const char *  pcLogFmt,
                               ...
               )
```

Logging function.

The contents of this function should be replaced by host driver function.

Can handle different log levels differently.

Parameters

level This logger level of the caller function.

Returns

Returns the AFE Log Level.

◆ connectAfeTxToFb()

```
uint8_t connectAfeTxToFb ( uint8_t  afeId,
                           uint8_t  txChNo,
                           uint8_t  fbChNo,
                           uint8_t  bandNo
                         )
```

Connect TX Output to FB for factory Calibration.

Connect TX Output to FB for factory Calibration.
The contents of this function contents should be replaced by host driver function.

Parameters

afeId	AFE ID
txChNo	Bit Wise TX Channel Select. Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD
fbChNo	Bit Wise FB Channel Select. Bit0 for FBAB Bit1 for FBCD When this is 1 or 2, connect the TX represented by txChNo to FBAB or FBCD respectively. When this is 3, connect the TX represented by txChNo[1:0] to FBAB and TX represented by txChNo[3:2] to FBCD
bandNo	Bit Wise Band Select. Bit0 for Band 0 Bit1 for Band 1

Returns

Returns if the function execution passed or failed.

◆ dev_spi_read()

```
uint8_t dev_spi_read ( uint8_t   afeId,
                       uint16_t  addr,
                       uint8_t * readVal
                     )
```

AFE SPI read driver function.

AFE SPI read driver function and returns the read value as pointer. The contents of this function should be replaced by host SPI driver function.

Parameters

afeId	AFE ID
addr	Address to be read from.
readVal	Pointer return of the value read.

Returns

Returns if the function execution passed or failed.

◆ dev_spi_write()

```
uint8_t dev_spi_write ( uint8_t  afeId,
                        uint16_t addr,
                        uint8_t  data
                      )
```

AFE SPI Write driver function.

AFE SPI Write driver function. The contents of this function should be replaced by host SPI driver function.

Parameters

afeId	AFE ID
addr	Address to be written to.
data	value to be written.

Returns

Returns if the function execution passed or failed.

◆ getAfeLogLvl()

```
uint32_t getAfeLogLvl ( )
```

Get the AFE Log Level.

Returns the AFE Log Level. There are multiple levels of logging as below.

AFE_LOG_LEVEL_ERROR 0 : Error conditions

AFE_LOG_LEVEL_WARNING 1 : warning conditions

AFE_LOG_LEVEL_INFO 2 : informational

AFE_LOG_LEVEL_SPILOG 3 : SPI-level messages

AFE_LOG_LEVEL_DEBUG 4 : debug-level messages

Returns

Returns the AFE Log Level.

◆ giveAfeAdcInput()

```
uint8_t giveAfeAdcInput ( uint8_t afeId,
                          uint8_t chNo,
                          uint8_t bandNo
                        )
```

Give ADC input for factory Calibration.

Give ADC input for factory Calibration. The contents of this function contents should be replaced by host driver function.

Parameters

afeId	AFE ID
chNo	Channel Number. 0-RXA 1-RXB 2-RXC 3-RXD 4-FBAB 5-FBCD
bandNo	Band Number 0/1.

Returns

Returns if the function execution passed or failed.

◆ giveSingleSysrefPulse()

uint8_t giveSingleSysrefPulse (uint8_t afeId)

AFE single shot Pin Sysref.

AFE single shot pin sysref driver function. The contents of this function should be replaced by host driver function.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ setAfeLogLvl()

void setAfeLogLvl (uint32_t level)

Set the AFE Log Level.

Sets the AFE Log Level. There are multiple levels of logging as below.

AFE_LOG_LEVEL_ERROR 0 : Error conditions

AFE_LOG_LEVEL_WARNING 1 : warning conditions

AFE_LOG_LEVEL_INFO 2 : informational

AFE_LOG_LEVEL_SPILOG 3 : SPI-level messages

AFE_LOG_LEVEL_DEBUG 4 : debug-level messages

Parameters

level Log level.

◆ wait()

uint8_t wait (uint32_t wait_s)

Wait in Seconds.

Wait in Seconds. The contents of this function should be replaced by host driver function.

Parameters

wait_s Wait time in seconds.

Returns

Returns if the function execution passed or failed.

◆ waitMs()

uint8_t waitMs (uint32_t wait_ms)

Wait in milli Seconds.

Wait in milli Seconds. The contents of this function should be replaced by host driver function.

Parameters

wait_ms Wait time in seconds.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

basicFunctions.h File Reference

Go to the source code of this file.

Functions

uint8_t **serdesRawRead** (uint8_t afeId, uint16_t addr, uint16_t *readVal)

SerDes Read. More...

uint8_t **serdesRawWrite** (uint8_t afeId, uint16_t addr, uint16_t data)

SerDes Write. More...

uint8_t **afeSpiWriteWrapper** (uint8_t afeId, uint16_t addr, uint8_t data, uint8_t lsb, uint8_t msb)

SPI Write Wrapper. More...

uint8_t **afeSpiReadWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t *readVal)

SPI Read Wrapper. More...

uint8_t **serdesWriteWrapper** (uint8_t afeId, uint16_t addr, uint16_t data, uint8_t lsb, uint8_t msb)

SerDes Write Wrapper. More...

uint8_t **serdesReadWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint16_t *readVal)

SerDes Read Wrapper. More...

uint8_t **serdesLaneWriteWrapper** (uint8_t afeId, uint16_t addr, uint8_t laneNo, uint16_t data, uint8_t lsb, uint8_t msb)

SerDes Lane Write Wrapper. More...

uint8_t **serdesLaneReadWrapper** (uint8_t afeId, uint16_t addr, uint32_t laneNo, uint8_t lsb, uint8_t msb, uint16_t *readVal)

SerDes Lane Read Wrapper. More...

uint8_t **afeSpiCheckWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t data, uint8_t *pbSame)

AFE SPI Check Wrapper. More...

uint8_t **afeSpiPollWrapper** (uint8_t afeId, uint16_t addr, uint8_t expectedData, uint8_t lsb, uint8_t msb)

AFE SPI Poll Wrapper. More...

uint8_t **afeSpiPollLogWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t expectedData)

AFE SPI Poll Wrapper. More...

uint8_t **requestPllSpiAccess** (uint8_t afeId, uint32_t regType)

Requesting PLL Spi Access. More...

uint8_t **readTopMem** (uint8_t afeId, uint32_t addr, uint64_t *readVal, uint32_t noBytes)

Reads the MCU Memory. More...

uint8_t **closeAllPages** (uint8_t afeId)

Close All Pages. More...

Function Documentation

◆ afeSpiCheckWrapper()

```
uint8_t afeSpiCheckWrapper ( uint8_t   afeId,
                             uint16_t  addr,
                             uint8_t   lsb,
                             uint8_t   msb,
                             uint8_t   data,
                             uint8_t * pbSame
                           )
```

AFE SPI Check Wrapper.

Reads and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb);

Parameters

afeId	AFE ID
addr	SPI address
data	Expected Value.
lsb	lsb of the field.
msb	msb of the field.
pbSame	Pointer return. Returns 0 if the check passes and if check fails.

Returns

Returns if the function execution passed or failed.

◆ afeSpiPollLogWrapper()

```
uint8_t afeSpiPollLogWrapper ( uint8_t   afeId,
                               uint16_t  addr,
                               uint8_t   lsb,
                               uint8_t   msb,
                               uint8_t   expectedData
                             )
```

AFE SPI Poll Wrapper.

Polls and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb); Function definition reordered from afeSpiPollWrapper to suit the log format.

Parameters

afeId	AFE ID
addr	SPI address
expectedData	Expected Value.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed. It returns fail even when the read data didn't match the expected value.

◆ afeSpiPollWrapper()

```
uint8_t afeSpiPollWrapper ( uint8_t   afeId,
                            uint16_t  addr,
                            uint8_t   expectedData,
                            uint8_t   lsb,
                            uint8_t   msb
                          )
```

AFE SPI Poll Wrapper.

Polls and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb);

Parameters

afeId	AFE ID
addr	SPI address
expectedData	Expected Value.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed. It returns fail even when the read data didn't match the expected value.

◆ afeSpiReadWrapper()

```
uint8_t afeSpiReadWrapper ( uint8_t   afeId,
                            uint16_t  addr,
                            uint8_t   lsb,
                            uint8_t   msb,
                            uint8_t * readVal
                          )
```

SPI Read Wrapper.

Reads the value to the specified bits of the register and returns as a pointer.

Parameters

afeId	AFE ID
addr	SPI address
lsb	lsb of the field.
msb	msb of the field.
readVal	pointer of the read value.

Returns

Returns if the function execution passed or failed.

◆ afeSpiWriteWrapper()

```
uint8_t afeSpiWriteWrapper ( uint8_t   afeId,
                             uint16_t  addr,
                             uint8_t   data,
                             uint8_t   lsb,
                             uint8_t   msb
                           )
```

SPI Write Wrapper.

Writes the value to the specified bits of the register.

Parameters

afeId	AFE ID
addr	SPI address
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

◆ closeAllPages()

```
uint8_t closeAllPages ( uint8_t  afeId )
```

Close All Pages.

This function closes all the pages. Need to be called in case of a SPI/function to ensure no open page is present.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ readTopMem()

```
uint8_t readTopMem ( uint8_t     afeId,
                     uint32_t    addr,
                     uint64_t *  readVal,
                     uint32_t    noBytes
                   )
```

Reads the MCU Memory.

This reads the MCU memory and returns the value as a pointer.

Parameters

afeId	AFE ID
addr	Memory Address.
readVal	Value read returned as a pointer.
noBytes	Number of bytes to be read. Supported values: 0<noBytes<=8

Returns

Returns if the function execution passed or failed.

◆ requestPllSpiAccess()

```
uint8_t requestPllSpiAccess ( uint8_t   afeId,
                              uint32_t  regType
                            )
```

Requesting PLL Spi Access.

For access PLL registers, the access to the PLL page should be requested and we should proceed only after it is granted. After the access is complete, the SPI access should be. This function does these operations. This access is independent for SPIA and SPIB.

Parameters

afeId	AFE ID
regType	0-Relinquish SPI access 1- Request Access for SPIA 2- Request Access for SPIB

Returns

Returns if the function execution passed or failed. It returns fail even when the request has not been granted.

◆ serdesLaneReadWrapper()

```
uint8_t serdesLaneReadWrapper ( uint8_t    afeId,
                                uint16_t   addr,
                                uint32_t   laneNo,
                                uint8_t    lsb,
                                uint8_t    msb,
                                uint16_t * readVal
                              )
```

SerDes Lane Read Wrapper.

Reads the value to the specified bits of the SerDes lane register of the corresponding lane by adding the appropriate offset. Returns the value as pointer.

Parameters

afeId	AFE ID
addr	SerDes lane base address
laneNo	SerDes lane number. 0-7 is the supported range.
lsb	lsb of the field.
msb	msb of the field.
readVal	Pointer of the value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesLaneWriteWrapper()

```
uint8_t serdesLaneWriteWrapper ( uint8_t   afeId,
                                 uint16_t  addr,
                                 uint8_t   laneNo,
                                 uint16_t  data,
                                 uint8_t   lsb,
                                 uint8_t   msb
                               )
```

SerDes Lane Write Wrapper.

Writes the value to the specified bits of the SerDes lane register of the corresponding lane by adding the appropriate offset.

Parameters

afeId	AFE ID
addr	SerDes lane base address
laneNo	SerDes lane Number. Values supported are: 0-7.
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

◆ serdesRawRead()

```
uint8_t serdesRawRead ( uint8_t    afeId,
                        uint16_t   addr,
                        uint16_t * readVal
                      )
```

SerDes Read.

SerDes registers are 16-bit wide while SPI is 8-bit. This necessitates a translation between SPI and SerDes. This function reads SerDes registers and returns the read value as a pointer.

Parameters

afeId	AFE ID
addr	SerDes address
readVal	Pointer returning the read value

Returns

Returns if the function execution passed or failed.

◆ serdesRawWrite()

```
uint8_t serdesRawWrite ( uint8_t   afeId,
                         uint16_t  addr,
                         uint16_t  data
                       )
```

SerDes Write.

SerDes registers are 16-bit wide while SPI is 8-bit. This necessitates a translation between SPI and SerDes. This function writes SerDes registers.

Parameters

afeId	AFE ID
addr	SerDes address
data	Value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesReadWrapper()

```
uint8_t serdesReadWrapper ( uint8_t    afeId,
                            uint16_t   addr,
                            uint8_t    lsb,
                            uint8_t    msb,
                            uint16_t * readVal
                          )
```

SerDes Read Wrapper.

Reads the value to the specified bits of the SerDes register and returns as a pointer.

Parameters

afeId	AFE ID
addr	SerDes address
lsb	lsb of the field.
msb	msb of the field.
readVal	Pointer of the value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesWriteWrapper()

```
uint8_t serdesWriteWrapper ( uint8_t  afeId,
                             uint16_t addr,
                             uint16_t data,
                             uint8_t  lsb,
                             uint8_t  msb
                           )
```

SerDes Write Wrapper.

Writes the value to the specified bits of the SerDes register.

Parameters

afeId	AFE ID
addr	SerDes address
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

calibrations.h File Reference

Go to the source code of this file.

Functions

uint8_t	**doRxDsaCalib** (uint8_t afeId, uint8_t rxChainForCalib, uint8_t fbChainForCalib, uint8_t useTxForCalib, uint8_t rxDsaBandCalibMode, uint8_t *readPacket, uint16_t *readPacketSize) Perform ADC DSA Calibration. More...
uint8_t	**doTxDsaCalib** (uint8_t afeId, uint8_t txChainForCalib, uint8_t txDsaCalibMode, uint8_t txDsaBandCalibMode, uint8_t *readPacket, uint16_t *readPacketSize) Perform DAC DSA Calibration. More...
uint8_t	**loadTxDsaPacket** (uint8_t afeId, uint8_t *array, uint8_t arraySize) Load the TX DSA Calibration Packet. More...
uint8_t	**loadRxDsaPacket** (uint8_t afeId, uint8_t *array, uint8_t arraySize) Load the ADC DSA Calibration Packet. More...

Function Documentation

◆ doRxDsaCalib()

```
uint8_t doRxDsaCalib ( uint8_t    afeId,
                       uint8_t    rxChainForCalib,
                       uint8_t    fbChainForCalib,
                       uint8_t    useTxForCalib,
                       uint8_t    rxDsaBandCalibMode,
                       uint8_t *  readPacket,
                       uint16_t * readPacketSize
                     )
```

Perform ADC DSA Calibration.

This function Performs the RX DSA calibration. giveAfeAdcInput function in **baseFunc.c** file contents should be coded by the user as needed. However, in a single band case, if all the channels can be given input at the same time, this function needn't do any operation and all the channels should be given input before calling this function.

Parameters

afeId	AFE ID
rxChainForCalib	Bit Wise RX Channel Select. Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fbChainForCalib	Bit Wise FB Channel Select. Bit0 for FBAB Bit1 for FBCD
useTxForCalib	When Set to 1, TX TDD will be kept on so that TX can be used for the calibration. The data should be still sent from the ASIC/FPGA through JESD.
rxDsaBandCalibMode	Sets the RX DSA Band Calibration Mode. 0 -One Band at a time 1 - both bands together
readPacket	Pointer returns Array of the Read packet. This should be stored in the host memory and be loaded post initialization in normal mode of operation.
readPacketSize	Pointer returns the size of the array.

Returns

Returns if the function execution passed or failed.

◆ doTxDsaCalib()

```
uint8_t doTxDsaCalib ( uint8_t    afeId,
                       uint8_t    txChainForCalib,
                       uint8_t    txDsaCalibMode,
                       uint8_t    txDsaBandCalibMode,
                       uint8_t *  readPacket,
                       uint16_t * readPacketSize
                     )
```

Perform DAC DSA Calibration.

This function Performs the TX DSA calibration. connectAfeTxToFb function in **baseFunc.c** file contents should be coded by the user as needed. However, in a single band case, if all the channels can be given input at the same time, this function needn't do any operation and all the channels should be given input before calling this function.

Parameters

afeId	AFE ID
txChainForCalib	Bit Wise TX Channel Select. Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD
txDsaCalibMode	DSA Calibration Mode. 0 -Single Fb Mode FB AB ; 1 -Single Fb Mode FB CD ; 2- Dual Fb_Mode
txDsaBandCalibMode	Sets the TX DSA Band Calibration Mode. 0 -One Band at a time 1 - both bands together
readPacket	Pointer returns Array of the Read packet. This should be stored in the host memory and be loaded post initialization in normal mode of operation.
readPacketSize	Pointer returns the size of the array.

Returns

Returns if the function execution passed or failed.

◆ loadRxDsaPacket()

```
uint8_t loadRxDsaPacket ( uint8_t   afeId,
                          uint8_t * array,
                          uint8_t   arraySize
                        )
```

Load the ADC DSA Calibration Packet.

This function loads the ADC DSA Calibration Packet

Parameters

afeId	AFE ID
array	Pointer of array of the packet which was stored in host after calibration.
arraySize	Value of the size of the array.

Returns

Returns if the function execution passed or failed.

◆ loadTxDsaPacket()

```
uint8_t loadTxDsaPacket ( uint8_t   afeId,
                          uint8_t * array,
                          uint8_t   arraySize
                        )
```

Load the TX DSA Calibration Packet.

This function loads the TX DSA Calibration Packet

Parameters

afeId	AFE ID
array	Pointer of array of the packet which was stored in host after calibration.
arraySize	Value of the size of the array.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

controls.h File Reference

Go to the source code of this file.

Functions

uint8_t **getChipVersion** (uint8_t afeId)

Reads the Chip version of the AFE. More...

uint8_t **checkSysref** (uint8_t afeId, uint8_t clearSysrefFlag, uint8_t *sysrefReceived)

Check if the Sysref Reached. More...

uint8_t **sendSysref** (uint8_t afeId, uint8_t spiSysref, uint8_t getSpiAccess)

Give a new Sysref to the AFE. More...

uint8_t	**overrideTdd** (uint8_t afeId, uint8_t rx, uint8_t fb, uint8_t tx, uint8_t enableOverride) Override TDD Control Signals and set the SPI override value. More...
uint8_t	**overrideTddPins** (uint8_t afeId, uint8_t rx, uint8_t fb, uint8_t tx) Override TDD Control Signals. More...
uint8_t	**checkPllLockStatus** (uint8_t afeId, uint8_t *pllLockStatus) Checks the PLL Lock Status. More...
uint8_t	**clearPllStickyLockStatus** (uint8_t afeId) Clears the PLL Lock Sticky Status. More...
uint8_t	**readAlarmPinStatus** (uint8_t afeId, uint8_t alarmNo, uint8_t *status) Checks the Alarm Pin Status. More...
uint8_t	**clearSpiAlarms** (uint8_t afeId) Clears the SPI Alarm Status. More...
uint8_t	**readSpiAlarms** (uint8_t afeId, uint8_t *alarmStatus) Checks the SPI Alarm Status. More...
uint8_t	**readTxPower** (uint8_t afeId, uint8_t chNo, uint16_t windowLen, double *powerReadB0, double *powerReadB1, double *combinedRead) Read the TX power. More...
uint8_t	**getRxRmsPower** (uint8_t afeId, uint8_t chNo, double *avg_pwrdb) Read the RX power. More...
uint8_t	**clearAllAlarms** (uint8_t afeId) Clear all the alarms. More...
uint8_t	**overrideAlarmPin** (uint8_t afeId, uint8_t alarmNo, uint8_t overrideSel, uint8_t overrideVal) Override Alarm Pin output and set the SPI override value. More...
uint8_t	**overrideRelDetPin** (uint8_t afeId, uint8_t chNo, uint8_t overrideSel, uint8_t overrideVal) Override RX Reliability Pin output and set the SPI override value. More...
uint8_t	**overrideDigPkDetPin** (uint8_t afeId, uint8_t chNo, uint8_t pinNo, uint8_t overrideSel, uint8_t overrideVal) Override RX Peak Detector Pin output and set the SPI override value. More...
uint8_t	**checkDeviceHealth** (uint8_t afeId, uint16_t *allOk) Checks the Device Health. More...

Function Documentation

◆ checkDeviceHealth()

```
uint8_t checkDeviceHealth ( uint8_t    afeId,
                            uint16_t * allOk
                          )
```

Checks the Device Health.

This function Reads the complete device health and returns as a pointer.

Parameters

afeId AFE ID

allOk Pointer return of the device health status.
If there is no error, allOk will be 0.
If it is non-zero, below is the interpretation
Bit 0: PLL Not Okay
Bit 1: DAC JESD Not Okay
Bit 2: ADC JESD Not Okay
Bit 3: SPI Not Okay
Bit 4: MCU Not Okay
Bit 5: PAP Triggered

Returns

Returns if the function execution passed or failed.

◆ checkPllLockStatus()

```
uint8_t checkPllLockStatus ( uint8_t   afeId,
                             uint8_t * pllLockStatus
                           )
```

Checks the PLL Lock Status.

This function checks the PLL Lock Status and returns it as a pointer.

Parameters

afeId AFE ID

pllLockStatus Pointer Return of the lock statud of the PLL. 3 is ideal good state.
0: LOCK is low and LOCK_LOST is high. PLL is currently not locked but locked some time in the past since the status clear bit was last toggled.
1: LOCK is high and LOCK_LOST is high. PLL is currently locked but lost lock since the status clear bit was last toggled. (since clearPllStickyLockStatus was called)
2: LOCK is low and LOCK_LOST is low. PLL is currently not locked and never locked since the status clear bit was last toggled.
3: LOCK is high and LOCK_LOST is low. PLL is currently locked and didn't lose lock since the status clear bit was last toggled. (since clearPllStickyLockStatus was called).

Returns

Returns if the function execution passed or failed.

◆ checkSysref()

```
uint8_t checkSysref ( uint8_t   afeId,
                      uint8_t   clearSysrefFlag,
                      uint8_t * sysrefReceived
                    )
```

Check if the Sysref Reached.

This function Checks if the Sysref is detected by the AFE.

Parameters

afeId	AFE ID
clearSysrefFlag	Setting this to 1 clear the Sysref flag before reading.
sysrefReceived	Pointer return. Value of 1 means Sysref reached. 0 means Sysref reached. 0 means it didn't reach.

Returns

Returns if the function execution passed or failed.

◆ clearAllAlarms()

```
uint8_t clearAllAlarms ( uint8_t  afeId )
```

Clear all the alarms.

Clears all the AFE alarms

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearPllStickyLockStatus()

```
uint8_t clearPllStickyLockStatus ( uint8_t  afeId )
```

Clears the PLL Lock Sticky Status.

This function clears the PLL Lock sticky Status.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearSpiAlarms()

```
uint8_t clearSpiAlarms ( uint8_t  afeId )
```

Clears the SPI Alarm Status.

This function clears the SPI Alarm Sticky Status. This is important when multiple SPIs are used and not critical when single SPI is being used.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ getChipVersion()

```
uint8_t getChipVersion ( uint8_t  afeId )
```

Reads the Chip version of the AFE.

This function Reads the Chip version, logs it and also updates the same in the System Params (systemParams[afeId].chipVersion).

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ getRxRmsPower()

```
uint8_t getRxRmsPower ( uint8_t   afeId,
                        uint8_t   chNo,
                        double *  avg_pwrdb
                      )
```

Read the RX power.

This function reads the RX Power.
Note that this detector is near the ADC-DDC interface and needs the RX TDD to be ON.
For reading FB power needed in ADC shared case, it should be operated in RX Mode and correponding RX channel should be read.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
avg_pwrdb	Pointer Return of RX Power Read

Returns

Returns if the function execution passed or failed.

◆ overrideAlarmPin()

```
uint8_t overrideAlarmPin ( uint8_t  afeId,
                           uint8_t  alarmNo,
                           uint8_t  overrideSel,
                           uint8_t  overrideVal
                         )
```

Override Alarm Pin output and set the SPI override value.

This function overrides Alarm Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
alarmNo	Select the Alarm number, Alarm Pin Number 0/1.
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideDigPkDetPin()

```
uint8_t overrideDigPkDetPin ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  pinNo,
                              uint8_t  overrideSel,
                              uint8_t  overrideVal
                            )
```

Override RX Peak Detector Pin output and set the SPI override value.

This function RX Peak Detector Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
chNo	Select the RX channel number. 0 for RXA 1 for RXB 2 for RXC 3 for RXD
pinNo	Pin Number to be overridden. Supported values are 0-3.
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideRelDetPin()

```
uint8_t overrideRelDetPin ( uint8_t  afeId,
                            uint8_t  chNo,
                            uint8_t  overrideSel,
                            uint8_t  overrideVal
                          )
```

Override RX Reliability Pin output and set the SPI override value.

This function overrides the RX Reliability Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
chNo	Select the RX channel number. 0 for RXA 1 for RXB 2 for RXC 3 for RXD
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideTdd()

```
uint8_t overrideTdd ( uint8_t  afeId,
                      uint8_t  rx,
                      uint8_t  fb,
                      uint8_t  tx,
                      uint8_t  enableOverride
                    )
```

Override TDD Control Signals and set the SPI override value.

This function overrides SPI TDD Control Signals and set the SPI override value

Parameters

afeId	AFE ID
rx	Override Value of the RX chain. This is Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fb	Override Value of the FB chain. This is Bit wise channel select Bit0 for FBAB Bit1 for FBCD
tx	Override Value of the TX chain. This is Bit wise channel select Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD

enableOverride	Enables the Override.
	if enableOverride=0, it disables the TDD override
	if enableOverride=1, it enables the TDD override && also sets the TDD values
	if enableOverride=2, it only sets the TDD values

Returns

Returns if the function execution passed or failed.

◆ overrideTddPins()

```
uint8_t overrideTddPins ( uint8_t  afeId,
                          uint8_t  rx,
                          uint8_t  fb,
                          uint8_t  tx
                        )
```

Override TDD Control Signals.

This function Set the override values for each of RX,FB,TX TDD pins.

Parameters

afeId AFE ID

rx Override enable Value of the RX chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.

fb Override enable Value of the FB chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.

tx Override enable Value of the TX chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.

Returns

Returns if the function execution passed or failed.

◆ readAlarmPinStatus()

```
uint8_t readAlarmPinStatus ( uint8_t    afeId,
                             uint8_t    alarmNo,
                             uint8_t *  status
                           )
```

Checks the Alarm Pin Status.

This function reads the Alarm Pin Status and returns it as a pointer.

Parameters

afeId AFE ID

alarmNo Choose the Alarm Pin Number (0/1)

status Pointer return Status of the alarm pin. 0 means there is no alarm and 1 means there is alarm.

Returns

Returns if the function execution passed or failed.

◆ readSpiAlarms()

```
uint8_t readSpiAlarms ( uint8_t   afeId,
                        uint8_t * alarmStatus
                      )
```

Checks the SPI Alarm Status.

This function reads the Alarm Status and returns it as a pointer. It also prints the error description.

Parameters

afeId	AFE ID
alarmStatus	Pointer return status of the SPI alarm. 0 means there are no alarms and 1 means there is a alarm.

Returns

Returns if the function execution passed or failed.

◆ readTxPower()

```
uint8_t readTxPower ( uint8_t  afeId,
                      uint8_t  chNo,
                      uint16_t windowLen,
                      double * powerReadB0,
                      double * powerReadB1,
                      double * combinedRead
                    )
```

Read the TX power.

This function reads the TX Power.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
windowLen	Determines the window length for number of samples. 2^(windowLen+5) samples at the interface rate will be used for power measurement. Range of this is 0-0xfff
powerReadB0	Pointer Return of Band 0 Power Read
powerReadB1	Pointer Return of Band 1 Power Read
combinedRead	Pointer Return of Power Read after the combiner

Returns

Returns if the function execution passed or failed.

◆ sendSysref()

```
uint8_t sendSysref ( uint8_t  afeId,
                     uint8_t  spiSysref,
                     uint8_t  getSpiAccess
                   )
```

Give a new Sysref to the AFE.

This function is used to send a new sysref to the AFE. This enables the latch and performs the required operations for AFE to accept a new Sysref.
Note the following:

1. Contents of the giveSingleSysrefPulse function should be replaced by host function to give Pin Sysref to AFE. This is used only in case of a single shot sysref.
2. For Continuous Syref mode, external Pin Sysref should be enabled before this function is called. Note that even in this case, only one pulse edge will be captured by the AFE. In this mode, giveSingleSysrefPulse needn't do any operation.
3. systemParams[afeId].spiInUseForPllAccess should be set before the function call to the appropriate value for selecting SPIA/SPIB. In Normal use-case SPIA is used and hence can be left at the default.
4. The selection between the single shot and continuous sysref mode should be done in Latte during generation of the configuration log.

Parameters

afeId	AFE ID
spiSysref	If this is set to 0, external pin based Sysref is used. If this is set to 1, then the internal override of the Sysref pin will be used. Note that in this case, deterministic latency will not satisfied.
getSpiAccess	Setting this to 1 will take PLL SPI access. This should always be set 1.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

dsaAndNco.h File Reference

Go to the source code of this file.

Functions

uint8_t	**setTxDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the TX Analog DSA. More...
uint8_t	**setFbDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the FB Analog DSA. More...
uint8_t	**setRxDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the RX Analog DSA. More...
uint8_t	**setRxDigGain** (uint8_t afeId, uint8_t chNo, uint8_t bandNo, uint8_t dsaSetting) Set the RX Digital DSA. More...
uint8_t	**setRxDsaMode** (uint8_t afeId, uint8_t topNo, uint8_t mode) Set the RX DSA Mode. More...
uint8_t	**setPinRxDsaSettings** (uint8_t afeId, uint8_t chNo, uint8_t dsaInit, uint8_t dsaStep, uint8_t maxDelay) Configure Settings related to the 4-pin based DSA control mode. More...
uint8_t	**setTxDigGain** (uint8_t afeId, uint8_t chNo, uint8_t bandNo, int16_t dig_gain) Set the TX Digital DSA. More...
uint8_t	**txDsaIdxGainSwap** (uint8_t afeId, uint8_t chNo, uint8_t anaAttn0, uint8_t anaAttn1, int8_t digB0Gain0, int8_t digB0Gain1, int8_t digB1Gain0, int8_t digB1Gain1) Set the TX DSA Gain Swap Attenuation. More...

uint8_t	**updateTxGainParam** (uint8_t afeId, uint8_t mode, uint8_t transitTime, uint8_t maxAnaDsa) Set the TX DSA Update Mode. More...
uint8_t	**updateTxGain** (uint8_t afeId, uint8_t txChainSel, uint8_t gainValidity, uint16_t tx0B0Dsa, uint16_t tx0B1Dsa, uint16_t tx1B0Dsa, uint16_t tx1B1Dsa) Set the TX DSA. More...
uint8_t	**updateTxNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t nco) Set the TX NCO for single band. More...
uint8_t	**updateTxNcoDb** (uint8_t afeId, uint8_t chNo, uint8_t nco, uint32_t band0Nco0, uint32_t band1Nco0, uint32_t band0Nco1, uint32_t band1Nco1) Set the TX NCO for Dual band. More...
uint8_t	**rxNCOSel** (uint8_t afeId, uint8_t chno, uint8_t BandId, uint8_t ovr, uint8_t NCOId) Set the RX NCO Select. More...
uint8_t	**fbNCOSel** (uint8_t afeId, uint8_t topno, uint8_t ovr, uint8_t NCOId) Set the FB NCO Select. More...
uint8_t	**updateRxNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t band, uint8_t nco) Set the RX NCO. More...
uint8_t	**updateFbNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t nco) Set the FB NCO. More...
uint8_t	**readRxNco** (uint8_t afeId, uint8_t chNo, uint8_t band, uint8_t nco, double *ncoFreq) Read the RX NCO. More...
uint8_t	**readTxNco** (uint8_t afeId, uint8_t chNo, uint8_t band, uint8_t nco, int64_t *val) Read the TX NCO. More...
uint8_t	**setFbDsaPerTx** (uint8_t afeId, uint8_t pinNo, uint8_t dsaSetting) Set the FB Analog DSA for pin select mode. More...
uint8_t	**fbDsaPerTxEn** (uint8_t afeId, uint8_t en) Enable the pin select based Mode for FB DSA. More...

Function Documentation

◆ fbDsaPerTxEn()

```
uint8_t fbDsaPerTxEn ( uint8_t  afeId,
                       uint8_t  en
                     )
```

Enable the pin select based Mode for FB DSA.

AFE has a feature to select the FB DSA value from a set of pre-programmed values using pins. This function sets the FB Analog DSA index.

Parameters

afeId	AFE ID
en	en as 1 will enable the feature to set FB DSA per TX based on the GPIO.

Returns

Returns if the function execution passed or failed.

◆ fbNCOSel()

```
uint8_t fbNCOSel ( uint8_t  afeId,
                   uint8_t  topno,
                   uint8_t  ovr,
                   uint8_t  NCOId
                 )
```

Set the FB NCO Select.

This function sets the override to the FB NCO select. This is useful only when more than 1 NCO is used.

Parameters

afeId	AFE ID
topno	Select the FB Channel 0 for FBAB 1 for FBCD
ovr	1 will override the pin. 0 will give control to the pin.
NCOId	NCO number which is to be selected. Supported range is 0 to numFbNco set in the initial configuration.

Returns

Returns if the function execution passed or failed.

◆ readRxNco()

```
uint8_t readRxNco ( uint8_t   afeId,
                    uint8_t   chNo,
                    uint8_t   band,
                    uint8_t   nco,
                    double *  ncoFreq
                  )
```

Read the RX NCO.

This function reads the RX NCO and returns it as a pointer. systemParams[afeId].ncoFreqMode should be matched with the value set in the initial configuration.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.
ncoFreq	Pointer Return. Returns the value of the NCO frequency read in MHz.

Returns

Returns if the function execution passed or failed.

◆ readTxNco()

```
uint8_t readTxNco ( uint8_t   afeId,
                    uint8_t   chNo,
                    uint8_t   band,
                    uint8_t   nco,
                    int64_t * val
                  )
```

Read the TX NCO.

This function reads the RX NCO and returns it as a pointer. systemParams[afeId].ncoFreqMode should be matched with the value set in the initial configuration.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.
val	Pointer Return. Returns the value of the NCO frequency read in MHz.

Returns

Returns if the function execution passed or failed.

◆ rxNCOSel()

```
uint8_t rxNCOSel ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  BandId,
                   uint8_t  ovr,
                   uint8_t  NCOId
                 )
```

Set the RX NCO Select.

This function sets the override to the RX NCO select. This is useful only when more than 1 NCO is used.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
BandId	NCO number. 0-NCO0, 1-NCO1.
ovr	1 will override the pin. 0 will give control to the pin.
NCOId	NCO number which is to be selected. Supported range is 0 to numRxNco set in the initial configuration.

Returns

Returns if the function execution passed or failed.

◆ setFbDsa()

```
uint8_t setFbDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the FB Analog DSA.

Sets the FB Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the FB Channel 0 for FBAB 1 for FBCD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*0.5dB

Returns

Returns if the function execution passed or failed.

◆ setFbDsaPerTx()

```
uint8_t setFbDsaPerTx ( uint8_t  afeId,
                        uint8_t  pinNo,
                        uint8_t  dsaSetting
                      )
```

Set the FB Analog DSA for pin select mode.

AFE has a feature to select the FB DSA value from a set of pre-programmed values using pins. This function sets the FB Analog DSA index. systemParams[afeId].txToFbMode should be set as needed in the initialization.

Parameters

afeId	AFE ID
pinNo	Select the pin value for which to program the DSA. The range of this is 0-3.
dsaSetting	Analog dsaSetting is FB DSA for the corresponding pin value. dsaSetting*0.5 is the attenuation in dB applied when the pin value is pinNo.

Returns

Returns if the function execution passed or failed.

◆ setPinRxDsaSettings()

```
uint8_t setPinRxDsaSettings ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  dsaInit,
                              uint8_t  dsaStep,
                              uint8_t  maxDelay
                            )
```

Configure Settings related to the 4-pin based DSA control mode.

Configure Settings related to the 4-pin based DSA control mode. Effective DSA attenuation is ((pin_value * dsaStep) +dsaInit)*0.5dB.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
dsaInit	Offset of the DSA.
dsaStep	DSA Step Value.
maxDelay	This is the delay after the change of pin change to latch the values. This is to account for the latency variation between pins. This should be the maximum latency variation between the earliest pin and the last pin. This is the common control for 2RX. The unit is in cycles of FadcRx/8 clock. Supported values: 0<=maxDelay<=255.

Returns

Returns if the function execution passed or failed.

◆ setRxDigGain()

```
uint8_t setRxDigGain ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  bandNo,
                       uint8_t  dsaSetting
                     )
```

Set the RX Digital DSA.

Sets the RX Digital DSA.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
bandNo	Select the RX Band. 0-Band0, 1-Band1
dsaSetting	(dsaSetting*0.5-3)dB is the applied DSA gain (if positive) and attenuation (if negative). Range for dsaSetting is 0 to 47.

Returns

Returns if the function execution passed or failed.

◆ setRxDsa()

```
uint8_t setRxDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the RX Analog DSA.

Sets the RX Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*0.5dB

Returns

Returns if the function execution passed or failed.

◆ setRxDsaMode()

```
uint8_t setRxDsaMode ( uint8_t  afeId,
                       uint8_t  topNo,
                       uint8_t  mode
                     )
```

Set the RX DSA Mode.

Sets the RX DSA Control Mode.

Parameters

afeId	AFE ID
topNo	Select the RX Channel 0 for RXAB 1 for RXCD
mode	DSA Control Mode Setting. 1-8-Pin Based DSA Control 2-Internal AGC 3-SPI AGC 4-4-Pin Based DSA Control

Returns

Returns if the function execution passed or failed.

◆ setTxDigGain()

```
uint8_t setTxDigGain ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  bandNo,
                       int16_t  dig_gain
                     )
```

Set the TX Digital DSA.

Sets the TX Digital DSA.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
bandNo	Select the TX Band. 0-Band0, 1-Band1
dig_gain	dig_gain is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain. (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed.

Returns

Returns if the function execution passed or failed.

◆ setTxDsa()

```
uint8_t setTxDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the TX Analog DSA.

Sets the TX Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*1dB

Returns

Returns if the function execution passed or failed.

◆ txDsaIdxGainSwap()

```
uint8_t txDsaIdxGainSwap ( uint8_t  afeId,
                           uint8_t  chNo,
                           uint8_t  anaAttn0,
                           uint8_t  anaAttn1,
                           int8_t   digB0Gain0,
                           int8_t   digB0Gain1,
                           int8_t   digB1Gain0,
                           int8_t   digB1Gain1
                         )
```

Set the TX DSA Gain Swap Attenuation.

Set the TX DSA Gain Swap Attenuation. There are 2 Gain Swap settings possible which can be chosen using the pin.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
anaAttn0	Analog Attenuation for Swap Attenuation 0, from 0 to 29. (1dB steps)
anaAttn1	Analog Attenuation for Swap Attenuation 1, from 0 to 29. (1dB steps)
digB0Gain0	Digital Attenuation*8 for Swap Attenuation 0 for band 0 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB0Gain1	Digital Attenuation*8 for Swap Attenuation 1 for band 0 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB1Gain0	Digital Attenuation*8 for Swap Attenuation 0 for band 1 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB1Gain1	Digital Attenuation*8 for Swap Attenuation 1 for band 1 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed

Returns

Returns if the function execution passed or failed.

◆ updateFbNco()

```
uint8_t updateFbNco ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint32_t  mixer,
                      uint8_t   nco
                    )
```

Set the FB NCO.

This function updates the FB NCO.

Parameters

afeId AFE ID

chNo Select the FB Channel

0 for FBAB

1 for FBCD

mixer Mixer frequency.

Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script.

In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/FadcRx).

nco NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateRxNco()

```
uint8_t updateRxNco ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint32_t  mixer,
                      uint8_t   band,
                      uint8_t   nco
                    )
```

Set the RX NCO.

This function updates the RX NCO.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
mixer	Mixer frequency. Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/FadcRx).
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateTxGain()

```
uint8_t updateTxGain ( uint8_t   afeId,
                       uint8_t   txChainSel,
                       uint8_t   gainValidity,
                       uint16_t  tx0B0Dsa,
                       uint16_t  tx0B1Dsa,
                       uint16_t  tx1B0Dsa,
                       uint16_t  tx1B1Dsa
                     )
```

Set the TX DSA.

This function sets the TX DSA (analog+digital) through Macro.
When the value is less or equal to than the maxAnaDsa setting in updateTxGainParam function, the integer part of the value will be applied to analog and fractional part will be applied to digital.
When the value is more than the maxAnaDsa setting in updateTxGainParam function, maxAnaDsa will be applied to the analog and rest will be applied in digital.
For single band case, set same value as band0 to band1 and apply gain validity accordingly.

Parameters

afeId	AFE ID
txChainSel	Selects if the DSA attenuation needs to be applied to AB or CD channels. 0-AB 1-CD
gainValidity	Selects where all to set the DSA. This is a bit wise field. bit 0- TXA/C Band0 bit 1- TXA/C Band1 bit 2- TXB/D Band0 bit 3- TXB/D Band1
tx0B0Dsa	TXA/C Band 0 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.
tx0B1Dsa	TXA/C Band 1 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.
tx1B0Dsa	TXB/D Band 0 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.
tx1B1Dsa	TXB/D Band 1 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.

Returns

Returns if the function execution passed or failed.

◆ updateTxGainParam()

```
uint8_t updateTxGainParam ( uint8_t  afeId,
                            uint8_t  mode,
                            uint8_t  transitTime,
                            uint8_t  maxAnaDsa
                          )
```

Set the TX DSA Update Mode.

This function sets the Params of applying TX DSA through Macro.

Parameters

afeId	AFE ID
mode	Mode of TX DSA Update 0-oneshot (Immediately update) 1-smoothening (Enable smooth transition of DSA) 2-TDD mode (Set DSA on TX TDD off state)
transitTime	This value/8 us is the time taken for each step in smoothening mode.
maxAnaDsa	This is the maximum analog DSA (in dB) beyond which the digital gain/attenuation will be applied. Maximum value of this is 29

Returns

Returns if the function execution passed or failed.

◆ updateTxNco()

```
uint8_t updateTxNco ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint32_t  mixer,
                      uint8_t   nco
                    )
```

Set the TX NCO for single band.

This function updates the TX NCO and should be used only single band of operation.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
mixer	Mixer frequency. Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
nco	NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateTxNcoDb()

```
uint8_t updateTxNcoDb ( uint8_t   afeId,
                        uint8_t   chNo,
                        uint8_t   nco,
                        uint32_t  band0Nco0,
                        uint32_t  band1Nco0,
                        uint32_t  band0Nco1,
                        uint32_t  band1Nco1
                      )
```

Set the TX NCO for Dual band.

This function updates the TX NCO and should be used only single band of operation.
For all the mixer frequency values, should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script.
In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
In case second NCO is not used, set the band1 parameters to same value as band0.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
nco	NCO number. 0-NCO0, 1-NCO1.
band0Nco0	Band0, NCO0 Mixer frequency.
band1Nco0	Band1, NCO0 Mixer frequency.
band0Nco1	Band0, NCO1 Mixer frequency.
band1Nco1	Band1, NCO1 Mixer frequency.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

hMacro.h File Reference

Go to the source code of this file.

Functions

uint8_t	**splitToByte** (uint64_t val, uint8_t numBytes, uint8_t *splitByteList) Converts a value into byte wise array. More...
uint8_t	**writeOperandList** (uint8_t afeId, uint8_t *operandList, uint8_t numOfOperands) Write the Macro Operands. More...
uint8_t	**readResultRegSpi** (uint8_t afeId, uint8_t regNum, uint32_t *result) Read Macro Result Register. More...
uint8_t	**waitForMacroReady** (uint8_t afeId)

	Poll for Macro Ready. More...
uint8_t	**waitForMacroDone** (uint8_t afeId) Poll for Macro Done. More...
uint8_t	**waitForMacroAck** (uint8_t afeId) Poll for Macro Acknowledgement. More...
uint8_t	**checkForMacroError** (uint8_t afeId, uint8_t *errorReg) Checks if there is a Macro Error. More...
uint8_t	**executeMacro** (uint8_t afeId, uint8_t *byteList, uint8_t numOfOperands, uint8_t opcode) Execute a Macro. More...
uint8_t	**triggerMacro** (uint8_t afeId, uint8_t opcode) Writes Opcode and triggers the Macro. More...
uint8_t	**enableMemAccess** (uint8_t afeId, uint8_t en) Enables MCU Memory Access for SPI. More...
uint8_t	**doSystemTuneSelective** (uint8_t afeId, uint8_t rxChList, uint8_t fbChList, uint8_t txChList, uint8_t sectionEnable) Reconfigures the selected Chains. More...
uint8_t	**updateSystemTxChannelFreqConfig** (uint8_t afeId, uint8_t txChList, uint8_t listNCO, uint32_t txNCO, uint8_t immUpdt, uint8_t reload) Reconfigures the TX NCO info to the MCU. More...
uint8_t	**checkMcuHealth** (uint8_t afeId, uint8_t *healthOk) Checks for MCU Health. More...
uint8_t	**txCalibSiggen** (uint8_t afeId, uint8_t chNo, uint8_t configOption, uint32_t freq0, uint8_t freq0Amp) TX Tone Generator. More...

Function Documentation

◆ checkForMacroError()

```
uint8_t checkForMacroError ( uint8_t   afeId,
                             uint8_t * macroErrorStatus
                           )
```

Checks if there is a Macro Error.

Checks if there is a Macro Error and returns the error status as pointer.

Parameters

afeId	AFE ID
macroErrorStatus	Macro Error Status return as pointer.

Returns

Returns if the function execution passed or failed.

◆ checkMcuHealth()

```
uint8_t checkMcuHealth ( uint8_t   afeId,
                         uint8_t * healthOk
                       )
```

Checks for MCU Health.

Checks for MCU Health and returns the status as a pointer.

Parameters

afeId	AFE ID
healthOk	Return Pointer of the status of the MCU. This value is 1 if the MCU is working properly and 0 if MCU is stuck.

Returns

Returns if the function execution passed or failed.

◆ doSystemTuneSelective()

```
uint8_t doSystemTuneSelective ( uint8_t afeId,
                                uint8_t rxChList,
                                uint8_t fbChList,
                                uint8_t txChList,
                                uint8_t sectionEnable
                              )
```

Reconfigures the selected Chains.

Reconfigures the selected Chains. This function is called in updateTxNco function and is not recommended to be called independently.

Returns

Returns if the function execution passed or failed.

◆ enableMemAccess()

```
uint8_t enableMemAccess ( uint8_t afeId,
                          uint8_t en
                        )
```

Enables MCU Memory Access for SPI.

Enables MCU Memory Access for SPI. Note that this should be relinquished after the access is complete.

Parameters

afeId	AFE ID
en	1 enable MCU memory access for SPI. 0 disable MCU memory access for SPI

Returns

Returns if the function execution passed or failed.

◆ executeMacro()

```
uint8_t executeMacro ( uint8_t   afeId,
                       uint8_t * byteList,
                       uint8_t   numOfOperands,
                       uint8_t   opcode
                     )
```

Execute a Macro.

Executes the Macro by calling other sub functions.

Parameters

afeId	AFE ID
byteList	Byte-wise array of operands to be written.
numOfOperands	Size of operandList.
opcode	Opcode of the Macro.

Returns

Returns if the function execution passed or failed.

◆ readResultRegSpi()

```
uint8_t readResultRegSpi ( uint8_t    afeId,
                           uint8_t    regNum,
                           uint32_t * result
                         )
```

Read Macro Result Register.

Read Macro Result Register

Parameters

afeId	AFE ID
regNum	Result register number.
result	Returns result register as a pointer.

Returns

Returns if the function execution passed or failed.

◆ splitToByte()

```
uint8_t splitToByte ( uint64_t  val,
                      uint8_t   numBytes,
                      uint8_t * splitByteList
                    )
```

Converts a value into byte wise array.

Converts a value into byte wise array.

Parameters

val	Value to be converted
numBytes	Number of Bytes to convert it to.
splitByteList	Pointer return of the resultant array.

Returns

Returns if the function execution passed or failed.

◆ triggerMacro()

```
uint8_t triggerMacro ( uint8_t afeId,
                       uint8_t opcode
                     )
```

Writes Opcode and triggers the Macro.

Writes Opcode and triggers the Macro.

Parameters

afeId	AFE ID
opcode	Opcode of the Macro.

Returns

Returns if the function execution passed or failed.

◆ txCalibSiggen()

```
uint8_t txCalibSiggen ( uint8_t   afeId,
                        uint8_t   chNo,
                        uint8_t   configOption,
                        uint32_t  freq0,
                        uint8_t   freq0Amp
                      )
```

TX Tone Generator.

This function sends a single tone to the TX.

Parameters

afeId	AFE ID
chNo	TX Channel Select. 0 for TXA 1 for TXB 2 for TXC 3 for TXD
configOption	Tone Generation Command 0 → RESERVED 1 → The current mixer configuration will be saved and the mixers will be configured to give the new tone frequency. 2 → The mixers will be configured to give the new tone but the saved configuration will not be modified. 3 → RESERVED 4 → Restore saved configuration. This can be called to restore the last saved mixer configuration.
freq0	RF tone Frequency Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
freq0Amp	Tone Backoff in dB

Returns

Returns if the function execution passed or failed.

◆ updateSystemTxChannelFreqConfig()

```
uint8_t updateSystemTxChannelFreqConfig ( uint8_t   afeId,
                                          uint8_t   txChList,
                                          uint8_t   listNCO,
                                          uint32_t  txNCO,
                                          uint8_t   immUpdt,
                                          uint8_t   reload
                                        )
```

Reconfigures the TX NCO info to the MCU.

Reconfigures the TX NCO info to the MCU. This function is called in updateTxNco function and is not recommended to be called independently.

Returns

Returns if the function execution passed or failed.

◆ waitForMacroAck()

uint8_t waitForMacroAck (uint8_t afeId)

Poll for Macro Acknowledgement.

Polls for Macro Acknowledgement

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_ACK doesn't become 1.

◆ waitForMacroDone()

uint8_t waitForMacroDone (uint8_t afeId)

Poll for Macro Done.

Polls for Macro Done

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_Done doesn't become 1.

◆ waitForMacroReady()

uint8_t waitForMacroReady (uint8_t afeId)

Poll for Macro Ready.

Polls for Macro Ready

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_Ready doesn't become 1.

◆ writeOperandList()

```
uint8_t writeOperandList ( uint8_t   afeId,
                           uint8_t * operandList,
                           uint8_t   numOfOperands
                         )
```

Write the Macro Operands.

Write the Macro Operands.

Parameters

afeId	AFE ID
operandList	Byte-wise array of operands to be written.
numOfOperands	Size of operandList.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

init.h File Reference

Go to the source code of this file.

Functions

int8_t **configAfeFromFileFormat0** (uint8_t afeId, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail)

Bringup function configuration function from log file for format 0 of Latte log. More...

int8_t **configAfeFromFileFormat5** (uint8_t afeId, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail)

Bringup function configuration function from log file for format 5 of Latte log. More...

int8_t **configAfeFromFile** (uint8_t afeId, uint8_t logFormat, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail)

Common Bringup function configuration function. More...

Function Documentation

◆ configAfeFromFile()

```
int8_t configAfeFromFile ( uint8_t  afeId,
                           uint8_t  logFormat,
                           char *   file,
                           uint8_t  breakAtPollFail,
                           uint8_t  breakAtReadCheckFail
                         )
```

Common Bringup function configuration function.

Common Bringup function configuration function from log file. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
logFormat	Choose the format between 0 and 5.
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

◆ configAfeFromFileFormat0()

```
int8_t configAfeFromFileFormat0 ( uint8_t  afeId,
                                  char *   file,
                                  uint8_t  breakAtPollFail,
                                  uint8_t  breakAtReadCheckFail
                                )
```

Bringup function configuration function from log file for format 0 of Latte log.

Bringup function configuration function from log file for format 0 of Latte log. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

◆ configAfeFromFileFormat5()

```
int8_t configAfeFromFileFormat5 ( uint8_t  afeId,
                                  char *   file,
                                  uint8_t  breakAtPollFail,
                                  uint8_t  breakAtReadCheckFail
                                )
```

Bringup function configuration function from log file for format 5 of Latte log.

Bringup function configuration function from log file for format 5 of Latte log. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

Generated by doxygen 1.8.17

jesd.h File Reference

Go to the source code of this file.

Functions

uint8_t	**dacJesdSendData** (uint8_t afeId, uint8_t topno) Send JESD Data from SerDes to DAC. More...
uint8_t	**dacJesdConstantTestPatternValue** (uint8_t afeId, uint8_t topno, uint8_t enable, uint8_t chNo, uint8_t bandNo, uint16_t valueI, uint16_t valueQ) Send Constant Test Pattern to DAC. More...
uint8_t	**dacJesdSendRampTestPattern** (uint8_t afeId, uint8_t topno, uint8_t increment) Send Ramp Test pattern to DAC. More...
uint8_t	**getJesdRxLaneErrors** (uint8_t afeId, uint8_t laneNo, uint8_t *error) Read the DAC JESD Lane Errors. More...
uint8_t	**getJesdRxLaneFifoErrors** (uint8_t afeId, uint8_t laneNo, uint8_t *error) Read the DAC JESD Lane FIFO Errors. More...
uint8_t	**getJesdRxMiscSerdesErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t *errorValue) Read the DAC JESD Miscellaneous Errors. More...
uint8_t	**getJesdRxAlarms** (uint8_t afeId, uint8_t *error) Read all the DAC JESD Errors. More...
uint8_t	**getJesdRxLinkStatus** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for DAC JESD. More...
uint8_t	**getJesdRxLinkStatus204B** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for for DAC JESD204B. More...
uint8_t	**getJesdRxLinkStatus204C** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for for DAC JESD204B. More...

uint8_t	**clearJesdTxAlarms** (uint8_t afeId) Clears ADC JESD JESD204 alarms. More...
uint8_t	**clearJesdRxAlarms** (uint8_t afeId) Clears DAC JESD JESD204 alarms going to the pin. More...
uint8_t	**clearJesdRxAlarmsForPap** (uint8_t afeId) Clears DAC JESD JESD204 alarms going to the PAP block. More...
uint8_t	**jesdRxClearSyncErrorCnt** (uint8_t afeId, uint8_t jesdNo) Clears the Sync Error counter for DAC JESD. More...
uint8_t	**jesdRxGetSyncErrorCnt** (uint8_t afeId, uint8_t jesdNo, uint8_t *linkErrorCount) Reads the Sync Error counter for DAC JESD. More...
uint8_t	**jesdTxGetSyncErrorCnt** (uint8_t afeId, uint8_t jesdLaneNo, uint8_t *linkErrorCount) Reads the Sync Error counter for ADC JESD. More...
uint8_t	**adcRampTestPattern** (uint8_t afeId, uint8_t topno, uint8_t chNo, uint8_t enable, uint8_t rampIncr) Send Ramp Test Pattern from ADC JESD. More...
uint8_t	**toggleSync** (uint8_t afeId, uint8_t overrideValue) Toggles the ADC JESD204B Sync Override. More...
uint8_t	**setJesdTxSyncOverride** (uint8_t afeId, uint8_t syncNo, uint8_t overrideValue, uint8_t syncValue) Overrides the SyncIn of the ADC JESD204B. More...
uint8_t	**setJesdRxSyncOverride** (uint8_t afeId, uint8_t syncNo, uint8_t overrideValue, uint8_t syncValue) Overrides the SyncOut of the DAC JESD204B. More...
uint8_t	**getJesdTxFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t *errors) Reads the ADC JESD Lane FIFO Errors. More...
uint8_t	**jesdRxFullResetToggle** (uint8_t afeId, uint8_t jesdNo) Resets the DAC JESD Block. More...
uint8_t	**jesdTxFullResetToggle** (uint8_t afeId, uint8_t jesdNo) Resets the ADC JESD Block. More...
uint8_t	**adcDacSync** (uint8_t afeId, uint8_t pinSysref) Resets and relinks the AFE JESD. More...
uint8_t	**jesdRxResetStateMachine** (uint8_t afeId, uint8_t linkNo) Resets the DAC JESD State Machine. More...
uint8_t	**getAllLaneReady** (uint8_t afeId, uint8_t jesdNo, uint8_t *rbdOffset) Returns the all lane ready counter. More...
uint8_t	**checkIfRbdIsGood** (uint8_t afeId, uint8_t jesdNo, uint8_t *rbdStatus) Checks if the set RBD value is okay or not. More...
uint8_t	**setGoodRbd** (uint8_t afeId, uint8_t jesdNo) Set Good RBD of DAC JESD. More...
uint8_t	**maskJesdRxLaneErrors** (uint8_t afeId, uint8_t laneNo, uint8_t maskValue) Mask DAC JESD Lane Errors to Pin. More...
uint8_t	**maskJesdRxLaneFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t losMaskValue, uint8_t fifoMaskValue) Mask DAC JESD FIFO Errors to Pin. More...
uint8_t	**maskJesdRxMiscSerdesErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t maskSerdesPllLock) Mask DAC JESD Miscellaneous Errors to Pin. More...
uint8_t	**maskJesdTxFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t maskValue) Mask ADC JESD FIFO Errors to Pin. More...
uint8_t	**maskJesdRxLaneErrorsToPap** (uint8_t afeId, uint8_t laneNo, uint8_t maskValue) Mask DAC JESD Lane Errors to PAP. More...

uint8_t	**maskJesdRxLaneFifoErrorsToPap** (uint8_t afeId, uint8_t jesdNo, uint8_t losMaskValue, uint8_t fifoMaskValue) Mask DAC JESD FIFO Errors to PAP. More...
uint8_t	**maskJesdRxMiscSerdesErrorsToPap** (uint8_t afeId, uint8_t jesdNo, uint8_t maskSerdesPllLock) Mask DAC JESD Miscellaneous Errors to PAP. More...
uint8_t	**setManualRbd** (uint8_t afeId, uint8_t jesdNo, uint8_t value) Sets the RBS valuw. More...

Function Documentation

◆ adcDacSync()

```
uint8_t adcDacSync ( uint8_t  afeId,
                     uint8_t  pinSysref
                   )
```

Resets and relinks the AFE JESD.

This resets all the JESD blocks, gives sysref to the AFE and checks for the DAC link status.
Note the following:

1. Contents of the giveSingleSysrefPulse function should be replaced by host function to give Pin Sysref to AFE. This is used only in case of a single shot sysref.
2. For Continuous Syref mode, external Pin Sysref should be enabled before this function is called. Note that even in this case, only one pulse edge will be captured by the AFE. In this mode, giveSingleSysrefPulse needn't do any operation.
3. systemParams[afeId].spiInUseForPllAccess should be set before the function call to the appropriate value for selecting SPIA/SPIB. In Normal use-case SPIA is used and hence can be left at the default.
4. The selection between the single shot and continuous sysref mode should be done in Latte during generation of the configuration log.
5. systemParams[afeId].syncLoopBack and systemParams[afeId].jesdProtocol should be appropriately set according to what is there in the initialization.
6. This doesn't perform any SerDes operations
7. Any ASIC operations needed for the relink should be done as needed.

Parameters

afeId	AFE ID
pinSysref	Chooses between pinSysref and internal copy of Sysref 0-Uses the internal copy of the Sysref to relink the JESD. 1-Uses Pin sysref for relink. Note that when the pin sysref is made 0, the internal copy of the Sysref is used which is not same as the SPI override of the pin sysref. The internal copy of the sysref will be synchronous to the previous copy of the sysref the AFE received. And since the Sysref frequency is calculated accounts for it, deterministic latency will be achieved even in this mode, assuming the phase of the external sysref is not disturbed.

Returns

Returns if the function execution passed or failed or if the relink is not successful.

◆ adcRampTestPattern()

```
uint8_t adcRampTestPattern ( uint8_t  afeId,
                             uint8_t  topno,
                             uint8_t  chNo,
                             uint8_t  enable,
                             uint8_t  rampIncr
                           )
```

Send Ramp Test Pattern from ADC JESD.

Send Ramp Test Pattern from ADC JESD. This test pattern is near the ADC-JESD interface.

Parameters

afeId	AFE ID
topno	Select the JESD instance. 0-AB. 1-CD.
chNo	0 for RXA/C; 1 for RX B/D; 2 for FB AB/CD
enable	1 to enable the Ramp pattern. 0 to disable.
rampIncr	rampIncr+1 is the increment of the steps.

Returns

Returns if the function execution passed or failed.

◆ checkIfRbdIsGood()

```
uint8_t checkIfRbdIsGood ( uint8_t    afeId,
                           uint8_t    jesdNo,
                           uint8_t *  rbdStatus
                         )
```

Checks if the set RBD value is okay or not.

Checks if the set RBD value is okay or not.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.
rbdStatus	Pointer return. Value will be 1 if the RBD set is good, else returns 0.

Returns

Returns if the function execution passed or failed.

◆ clearJesdRxAlarms()

uint8_t clearJesdRxAlarms (uint8_t afeId)

Clears DAC JESD JESD204 alarms going to the pin.

Clears DAC JESD JESD204 alarms going to the pin of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearJesdRxAlarmsForPap()

uint8_t clearJesdRxAlarmsForPap (uint8_t afeId)

Clears DAC JESD JESD204 alarms going to the PAP block.

Clears DAC JESD JESD204 alarms going to the PAP block of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearJesdTxAlarms()

uint8_t clearJesdTxAlarms (uint8_t afeId)

Clears ADC JESD JESD204 alarms.

Clears ADC JESD JESD204 alarms of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ dacJesdConstantTestPatternValue()

```
uint8_t dacJesdConstantTestPatternValue ( uint8_t   afeId,
                                          uint8_t   topno,
                                          uint8_t   enable,
                                          uint8_t   chNo,
                                          uint8_t   bandNo,
                                          uint16_t  valueI,
                                          uint16_t  valueQ
                                        )
```

Send Constant Test Pattern to DAC.

Send Constant Test Pattern to DAC. This test pattern is near the JESD-DUC interface. The output of the DAC will be a single tone for each band at the mixer frequency.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.
enable	0-Send Data From SERDES. 1- Send Constant Test Pattern. This is common for AB/CD.
chNo	0-A/C, 1-B/D
bandNo	0-Band 0, 1- Band1. In single band case, this should always be 0.
valueI	Value to be sent on I
valueQ	Value to be sent on Q

Returns

Returns if the function execution passed or failed.

◆ dacJesdSendData()

```
uint8_t dacJesdSendData ( uint8_t  afeId,
                          uint8_t  topno
                        )
```

Send JESD Data from SerDes to DAC.

Send JESD Data from SerDes. This should be called to change from test pattern mode to normal data mode.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.

Returns

Returns if the function execution passed or failed.

◆ dacJesdSendRampTestPattern()

```
uint8_t dacJesdSendRampTestPattern ( uint8_t  afeId,
                                     uint8_t  topno,
                                     uint8_t  increment
                                   )
```

Send Ramp Test pattern to DAC.

Send Ramp Test pattern to DAC. This test pattern is near the JESD-DUC interface.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.
increment	increment+1 is the step value of the ramp.

Returns

Returns if the function execution passed or failed.

◆ getAllLaneReady()

```
uint8_t getAllLaneReady ( uint8_t    afeId,
                          uint8_t    jesdNo,
                          uint8_t *  rbdOffset
                        )
```

Returns the all lane ready counter.

This function reads the all lane ready counter which is the offset between the internal LMFC boundary and the multiframe boundary (in JESD204B) or extended multi block boundary (in JESD204C) of the last lane of arrival. This value after an offset (of say, 2) with modulus of 64 should be writen to the RBD register.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.
rbdOffset	Pointer return. Value of the last all lane ready counter.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxAlarms()

```
uint8_t getJesdRxAlarms ( uint8_t    afeId,
                          uint8_t *  error
                        )
```

Read all the DAC JESD Errors.

Reads all the DAC JESD Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId AFE ID

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLaneErrors()

```
uint8_t getJesdRxLaneErrors ( uint8_t    afeId,
                              uint8_t    laneNo,
                              uint8_t *  error
                            )
```

Read the DAC JESD Lane Errors.

Reads the DAC JESD Lane Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId AFE ID

laneNo JESD Lane Number. Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLaneFifoErrors()

```
uint8_t getJesdRxLaneFifoErrors ( uint8_t   afeId,
                                  uint8_t   laneNo,
                                  uint8_t * error
                                )
```

Read the DAC JESD Lane FIFO Errors.

Reads the DAC JESD Lane FIFO Errors, logs their meaning and returns the alarm status as pointer. These are SerDes FIFO errors. If this error is present, either the SerDes is likely seeing some eye based issues .

Parameters

afeId AFE ID

laneNo JESD Lane Number. Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus()

```
uint8_t getJesdRxLinkStatus ( uint8_t    afeId,
                              uint16_t * linkStatus
                            )
```

Read Link Status for DAC JESD.

Reads link status for DAC JESD for all the enabled lanes and returns it. This calls functions getJesdRxLinkStatus204B/getJesdRxLinkStatus204C based on set systemParams[afeId].jesdProtocol.

Parameters

afeId AFE ID

linkStatus Pointer return of the status.

Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes.

=0 Idle state. No change in state.

=1 CGS Passed. Still in K characters mode.

=2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus204B()

```
uint8_t getJesdRxLinkStatus204B ( uint8_t    afeId,
                                  uint16_t *  linkStatus
                                )
```

Read Link Status for for DAC JESD204B.

Reads link status for all the enabled lanes and returns it.

Parameters

afeId	AFE ID
linkStatus	Pointer return of the status. Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes. =0 Idle state. No change in state. =1 In JESD204B: CGS Passed. Still in K characters mode. In JESD204C:Header Aligned but EoEMB lock yet to happen. =2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus204C()

```
uint8_t getJesdRxLinkStatus204C ( uint8_t    afeId,
                                  uint16_t *  linkStatus
                                )
```

Read Link Status for for DAC JESD204B.

Reads link status for all the enabled lanes and returns it.

Parameters

afeId	AFE ID
linkStatus	Pointer return of the status. Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes. =0 Idle state. No change in state. =1 Header Aligned but EoEMB lock yet to happen. =2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxMiscSerdesErrors()

```
uint8_t getJesdRxMiscSerdesErrors ( uint8_t   afeId,
                                    uint8_t   jesdNo,
                                    uint8_t * errorValue
                                  )
```

Read the DAC JESD Miscellaneous Errors.

Reads the DAC JESD Miscellaneous Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId	AFE ID
jesdNo	JESD Instance Number (0/1). Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).
errorValue	Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdTxFifoErrors()

```
uint8_t getJesdTxFifoErrors ( uint8_t   afeId,
                              uint8_t   jesdNo,
                              uint8_t * errors
                            )
```

Reads the ADC JESD Lane FIFO Errors.

Reads the ADC JESD Lane FIFO Errors, logs their meaning and returns the alarm status as pointer

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.
errors	Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ jesdRxClearSyncErrorCnt()

```
uint8_t jesdRxClearSyncErrorCnt ( uint8_t  afeId,
                                  uint8_t  jesdNo
                                )
```

Clears the Sync Error counter for DAC JESD.

Clears the Sync Error counter for DAC JESD.

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.

Returns

Returns if the function execution passed or failed.

◆ jesdRxFullResetToggle()

```
uint8_t jesdRxFullResetToggle ( uint8_t  afeId,
                                uint8_t  jesdNo
                              )
```

Resets the DAC JESD Block.

Resets the DAC JESD Block. Sysref should be given to the complete AFE after doing this. It is recommended to always keep jesdNo as 3.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdRxGetSyncErrorCnt()

```
uint8_t jesdRxGetSyncErrorCnt ( uint8_t    afeId,
                                uint8_t    jesdNo,
                                uint8_t *  linkErrorCount
                              )
```

Reads the Sync Error counter for DAC JESD.

Reads the Sync Error counter for DAC JESD and returns it as a pointer.

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
linkErrorCount	Pointer returns the counter value of the sync error. This denotes the number of times the resync request was given since the last time it is cleared.

Returns

Returns if the function execution passed or failed.

◆ jesdRxResetStateMachine()

```
uint8_t jesdRxResetStateMachine ( uint8_t  afeId,
                                  uint8_t  linkNo
                                )
```

Resets the DAC JESD State Machine.

Resets the DAC JESD State Machine. In this case no Sysref is needed to be given to the AFE. The LMFC boundary will remain aligned to the previous sysref AFE received.

Parameters

afeId	AFE ID
linkNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdTxFullResetToggle()

```
uint8_t jesdTxFullResetToggle ( uint8_t afeId,
                                uint8_t jesdNo
                              )
```

Resets the ADC JESD Block.

Resets the ADC JESD Block. Sysref should be given to the complete AFE after doing this. It is recommended to always keep jesdNo as 3.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdTxGetSyncErrorCnt()

```
uint8_t jesdTxGetSyncErrorCnt ( uint8_t   afeId,
                                uint8_t   jesdLaneNo,
                                uint8_t * linkErrorCount
                              )
```

Reads the Sync Error counter for ADC JESD.

Reads the Sync Error counter for ADC JESD and returns it as a pointer.

Parameters

afeId	AFE ID
jesdLaneNo	0-7 is the lane number pre-lane mux (towards the AFE).
linkErrorCount	Pointer returns the counter value of the sync error. This denotes the number of times the resync request was given since the last time the JESD was reset. There is no clear counter for this.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneErrors()

```
uint8_t maskJesdRxLaneErrors ( uint8_t  afeId,
                               uint8_t  laneNo,
                               uint8_t  maskValue
                             )
```

Mask DAC JESD Lane Errors to Pin.

Mask DAC JESD Lane Errors to Pin.

Parameters

afeId	AFE ID
laneNo	The laneNo is the post-laneMux lane number 0-7 (towards the AFE).

maskValue	The bits made 1 will be masked and not reflect on pin.
	Bit No 7 = "multiframe alignment error";
	Bit No 6 = "frame alignment error";
	Bit No 5 = "link configuration error";
	Bit No 4 = "elastic buffer overflow (bad RBD value)";
	Bit No 3 = "elastic buffer match error. The first no-/K/ does not match 'match_ctrl' and 'match_data' programmed values";
	Bit No 2 = "code synchronization error";
	Bit No 1 = "JESD 204B: 8b/10b not-in-table code error. JESD 204C: sync_header_invalid_err";
	Bit No 0 = "JESD 204B: 8b/10b disparty error. JESD 204C: sync_header_parity_err";

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneErrorsToPap()

```
uint8_t maskJesdRxLaneErrorsToPap ( uint8_t  afeId,
                                    uint8_t  laneNo,
                                    uint8_t  maskValue
                                  )
```

Mask DAC JESD Lane Errors to PAP.

Mask DAC JESD Lane Errors to PAP.

Parameters

afeId	AFE ID
laneNo	The laneNo is the post-laneMux lane number 0-7 (towards the AFE).

maskValue	The bits made 1 will be masked and not reflect on PAP.
	Bit No 7 = "multiframe alignment error";
	Bit No 6 = "frame alignment error";
	Bit No 5 = "link configuration error";
	Bit No 4 = "elastic buffer overflow (bad RBD value)";
	Bit No 3 = "elastic buffer match error. The first no-/K/ does not match 'match_ctrl' and 'match_data' programmed values";
	Bit No 2 = "code synchronization error";
	Bit No 1 = "JESD 204B: 8b/10b not-in-table code error. JESD 204C: sync_header_invalid_err";
	Bit No 0 = "JESD 204B: 8b/10b disparty error. JESD 204C: sync_header_parity_err";

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneFifoErrors()

```
uint8_t maskJesdRxLaneFifoErrors ( uint8_t  afeId,
                                   uint8_t  jesdNo,
                                   uint8_t  losMaskValue,
                                   uint8_t  fifoMaskValue
                                 )
```

Mask DAC JESD FIFO Errors to Pin.

Mask DAC JESD FIFO Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
losMaskValue	The bits made 1 will be masked and not reflect on pin. Bits0-3 for SerDes Rx Lane Loss of lock error for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.
fifoMaskValue	The bits made 1 will be masked and not reflect on pin. Bits0-3 for SerDes Rx Lane FIFO Error for for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneFifoErrorsToPap()

```
uint8_t maskJesdRxLaneFifoErrorsToPap ( uint8_t  afeId,
                                        uint8_t  jesdNo,
                                        uint8_t  losMaskValue,
                                        uint8_t  fifoMaskValue
                                      )
```

Mask DAC JESD FIFO Errors to PAP.

Mask DAC JESD FIFO Errors to PAP

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
losMaskValue	The bits made 1 will be masked and not reflect on PAP. Bits0-3 for SerDes Rx Lane Loss of lock error for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.
fifoMaskValue	The bits made 1 will be masked and not reflect on PAP. Bits0-3 for SerDes Rx Lane FIFO Error for for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxMiscSerdesErrors()

```
uint8_t maskJesdRxMiscSerdesErrors ( uint8_t  afeId,
                                     uint8_t  jesdNo,
                                     uint8_t  maskSerdesPllLock
                                   )
```

Mask DAC JESD Miscellaneous Errors to Pin.

Mask DAC JESD Miscellaneous Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskSerdesPllLock	The bits made 1 will be masked and not reflect on pin. Bit 0 for SRX1-4 and Bit 1 for SRX 5-8. These are Actual SerDes Lane numbers. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxMiscSerdesErrorsToPap()

```
uint8_t maskJesdRxMiscSerdesErrorsToPap ( uint8_t  afeId,
                                          uint8_t  jesdNo,
                                          uint8_t  maskSerdesPllLock
                                        )
```

Mask DAC JESD Miscellaneous Errors to PAP.

Mask DAC JESD Miscellaneous Errors to PAP

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskSerdesPllLock	The bits made 1 will be masked and not reflect on PAP. Bit 0 for SRX1-4 and Bit 1 for SRX 5-8. These are Actual SerDes Lane numbers. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdTxFifoErrors()

```
uint8_t maskJesdTxFifoErrors ( uint8_t  afeId,
                               uint8_t  jesdNo,
                               uint8_t  maskValue
                             )
```

Mask ADC JESD FIFO Errors to Pin.

Mask ADC JESD FIFO Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskValue	The bits made 1 will be masked and not reflect on pin. bit 0 is for lane 0 errors, bit 1 for lane 1 errors bit 2 for lane 2 errors bit 3 for lane 3 errors The lane number is the lane number of this instance pre-lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ setGoodRbd()

```
uint8_t setGoodRbd ( uint8_t  afeId,
                     uint8_t  jesdNo
                   )
```

Set Good RBD of DAC JESD.

This function does the following:

1. Reads the internal counter between the internal LMFC counter to the received LMFC boundary (multi frame boundary in 204B and Extended Multi Block Boundary in 204C).
2. Sets the RBD by giving an offset of 4 to the LMFC Counter.
3. Relinks by calling the adcDacSync function with pinSysref=1. So all the related conditions of adcDacSync are to be satisfied here too.
   Note that this sequence may not ensure deterministic latency across bring-ups and devices. For acheiving it the above 3 steps should be executed using the functions getAllLaneReady, setManualRbd, adcDacSync with external pin sysref. The getAllLaneReady should be done only once and the same value should be loaded each time during the intialization. This can be input as a parameter to Latte.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.

Returns

Returns if the function execution passed or failed.

◆ setJesdRxSyncOverride()

```
uint8_t setJesdRxSyncOverride ( uint8_t  afeId,
                                uint8_t  syncNo,
                                uint8_t  overrideValue,
                                uint8_t  syncValue
                              )
```

Overrides the SyncOut of the DAC JESD204B.

Overrides the SyncOut of the DAC JESD204B.

Parameters

afeId	AFE ID
syncNo	syncNo the sync value. 0-3
overrideValue	0- do not override. 1- override the SyncIn pin
syncValue	Pin state

Returns

Returns if the function execution passed or failed.

◆ setJesdTxSyncOverride()

```
uint8_t setJesdTxSyncOverride ( uint8_t  afeId,
                                uint8_t  syncNo,
                                uint8_t  overrideValue,
                                uint8_t  syncValue
                              )
```

Overrides the SyncIn of the ADC JESD204B.

Overrides the SyncIn of the ADC JESD204B.

Parameters

afeId	AFE ID
syncNo	syncNo the sync value. 0-5
overrideValue	0- do not override. 1- override the SyncIn pin
syncValue	0- Send K characters. 1- Send Data

Returns

Returns if the function execution passed or failed.

◆ setManualRbd()

```
uint8_t setManualRbd ( uint8_t  afeId,
                       uint8_t  jesdNo,
                       uint8_t  value
                     )
```

Sets the RBS valuw.

Mask DAC JESD Miscellaneous Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
value	RBD Value.

Returns

Returns if the function execution passed or failed.

◆ toggleSync()

```
uint8_t toggleSync ( uint8_t  afeId,
                     uint8_t  overrideValue
                   )
```

Toggles the ADC JESD204B Sync Override.

Override the SyncIN override, forces K characters for 100ms and then sends the data. This is to be used only in software sync mode in JESD 204B.

Parameters

afeId	AFE ID
overrideValue	Overrides the Sync Pin to this value during K characters mode. Bits 0-5 refer to syncin numbers 0-5. This can be made 0x3f to send K characters on all the links. To toggle sync of only a particular link, need to set only that bit to 1. For example, to toggle only link 2 using SyncIn2, need to set this value to 0x04.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

pap.h File Reference

Go to the source code of this file.

Functions

uint8_t	**configurePapMaDet** (uint8_t afeId, uint8_t chno, uint8_t maEnable, uint16_t maNumSample, uint16_t maWindowCntr, uint16_t maWindowCntrTh, uint16_t maThreshB0, uint16_t maThreshB1, uint16_t maThreshComb) Configure the Moving Average PAP Detector. More...
uint8_t	**configurePapHpfDet** (uint8_t afeId, uint8_t chno, uint8_t hpfEnable, uint16_t hpfNumSample, uint16_t hpfWindowCntr, uint16_t hpfWindowCntrTh, uint16_t hpfThreshB0, uint16_t hpfThreshB1, uint16_t hpfThreshComb) Configure the High Pass Filter PAP Detector. More...
uint8_t	

	configurePap (uint8_t afeId, uint8_t chno, uint8_t enable, uint8_t multMode, uint8_t rampDownStartVal, uint8_t attnStepSize, uint8_t gainStepSize, uint8_t detectInWaitState, float triggerToRampDown, float waitCounter, float triggerClearToRampUp, float amplUpdateCycles, float alarmPulseGPIO, uint8_t alarmMask, uint8_t alarmChannelMask, uint8_t alarmPinDynamicMode, uint8_t rampStickyMode) Configure the PAP Block. More...
uint8_t	**rampStickyClear** (uint8_t afeId, uint8_t chno) Clear the ramp sticky state. More...
uint8_t	**papAlarmStatus** (uint8_t afeId, uint8_t chno, uint8_t *alarmTriggered) Reads the PAP alarm Status. More...
uint8_t	**clearPapAlarms** (uint8_t afeId, uint8_t chno) Clears the PAP alarm Status. More...
uint8_t	**configLaneErrorsForTxPap** (uint8_t afeId, uint8_t chno, uint8_t laneMask) Map the DAC JESD lane errors to the PAP block. More...

Function Documentation

◆ clearPapAlarms()

```
uint8_t clearPapAlarms ( uint8_t  afeId,
                         uint8_t  chno
                       )
```

Clears the PAP alarm Status.

Clears the PAP alarm sticky status which also goes to GPIO.

Parameters

afeId AFE ID

chno Value 1 = Clear, Value 0 = No Clear

bit [0] :TxA Alarm

bit [1]: TxB Alarm

bit [2]: TxC Alarm

bit [3]: TxD Alarm

Returns

Returns if the function execution passed or failed.

◆ configLaneErrorsForTxPap()

```
uint8_t configLaneErrorsForTxPap ( uint8_t  afeId,
                                   uint8_t  chno,
                                   uint8_t  laneMask
                                 )
```

Map the DAC JESD lane errors to the PAP block.

This function chooses which lanes' errors should go to a particular TX PAP.

Parameters

afeId AFE ID

chno Select the TX Channel

0 for TXA

1 for TXB

2 for TXC

3 for TXD

laneMask This is 8-bit field with bit wise enable for lane errors.

Bit0 is for lane0, Bit1 for lane1, Bit2 for lane 2 and so on.

The errors from lanes with corresponding bits set to 1 will reach the PAP.

The lane numbers are pre-lane mux (towards the AFE).

For example, for TXA PAP to get errors from lanes 0 and 1, laneMask should be 0b00000011.

Registers written are (tx<a/b/c/d>_lane_alarms_to_pap_en) in DAC JESD.s

Returns

Returns if the function execution passed or failed.

◆ configurePap()

```
uint8_t configurePap ( uint8_t  afeId,
                       uint8_t  chno,
                       uint8_t  enable,
                       uint8_t  multMode,
                       uint8_t  rampDownStartVal,
                       uint8_t  attnStepSize,
                       uint8_t  gainStepSize,
                       uint8_t  detectInWaitState,
                       float    triggerToRampDown,
                       float    waitCounter,
                       float    triggerClearToRampUp,
                       float    amplUpdateCycles,
                       float    alarmPulseGPIO,
                       uint8_t  alarmMask,
                       uint8_t  alarmChannelMask,
                       uint8_t  alarmPinDynamicMode,
                       uint8_t  rampStickyMode
                     )
```

Configure the PAP Block.

Configure the PAP Block. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
enable	1 to enable PAP. 0 To disable PAP
multMode	Mode of Ramp up/down. 0:Cosine 1:Linear
rampDownStartVal	This is the starting value for the ramp down. Supported range: 0 to 127. For cosine mode, the start phase in radians is (128-rampDownStartVal)* pi /128. For linear mode, (rampDownStartVal/128) is the start value.
attnStepSize	This is the ramp step size while ramping down (actualSampleStep=attnStepSize*(last good sample)/1024). Supported Range is 0 to 127.
gainStepSize	This is the ramp step size while gaining up (actualSampleStep=GainStepSize*(last good sample)/1024). Supported Range is 0 to 127.
detectInWaitState	This determines if the PAP trigger should be acknowledged in the wait state. 0:Do not detect in wait state 1:detect in wait state
triggerToRampDown	Time from trigger occurance to ramp down time (ns). Supported Range: 0 to floor(65520000.0/Fdac) where Fdac is in MHz.
waitCounter	Wait time counter (ns). Supported Range: 0 to floor(1048560000.0/Fdac) where Fdac is in MHz.
triggerClearToRampUp	Time from end of wait state to Ramp up (ns). Supported Range: 0 to floor(65520000.0/Fdac) where Fdac is in MHz.
amplUpdateCycles	Time for each step during ramp up or down. (ns). Supported Range: 0 to floor(2032000.0/Fdac) where Fdac is in MHz.
alarmPulseGPIO	Pulse width of PAP alarm going to GPIO (ns). Supported Range: 0 to floor(1048560000.0/Fdac) where Fdac is in MHz.
alarmMask	

	Bit wise alarms. Bit value 0 will make corresponding alarm to trigger PAP state machine. BitNo: Alarm 0 : pll_alarm, 1 : serdes_alarm, 2 : fifo_alarm, 3 : ovr_saturation_alarm, 4 : dual band det alarm, 5 : combined band det alarm, 6 : spi trigger
alarmChannelMask	Mask other channels (bit-wise). For each channel the bit-wise description is different. Ch : BitNo 3-2-1-0 TxA : D-C-B-A, TxB : D-C-A-B, TxC : B-A-D-C, TxD : B-A-C-D
alarmPinDynamicMode	Determines if the PAP Pin is sticky or non-sticky. 1:dynamic, 0:sticky
rampStickyMode	Determines if the Ramp up mode is sticky or non-sticky. 0:Automatically come to ramp up mode after wait state. 1: Wait for pap clear bit to be written

Returns

Returns if the function execution passed or failed.

◆ configurePapHpfDet()

```
uint8_t configurePapHpfDet ( uint8_t   afeId,
                             uint8_t   chno,
                             uint8_t   hpfEnable,
                             uint16_t  hpfNumSample,
                             uint16_t  hpfWindowCntr,
                             uint16_t  hpfWindowCntrTh,
                             uint16_t  hpfThreshB0,
                             uint16_t  hpfThreshB1,
                             uint16_t  hpfThreshComb
                           )
```

Configure the High Pass Filter PAP Detector.

Configure the High Pass Filter PAP Detector. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
hpfEnable	0: Disable High Pass Filter based PAP detector. 1:Enable High Pass Filter based PAP detector.
hpfNumSample	Number of samples in a window. Supported values: 1-32; 2-64; 3-128 Samples
hpfWindowCntr	Number of windows. Supported Range: 0 to 2**12-1
hpfWindowCntrTh	Window Counter Threshold. When the number of windows in a set of maWindowCntr windows have filter trigger. This should be lower than maWindowCntr. Supported Range: 0:2**12-1.
hpfThreshB0	(128*val) is the filter threshold for band 0 detector. Supported Range: 0-511
hpfThreshB1	(128*val) is the filter threshold for band 1 detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.
hpfThreshComb	(128*val) is the filter threshold for combined detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.

Returns

Returns if the function execution passed or failed.

◆ configurePapMaDet()

```
uint8_t configurePapMaDet ( uint8_t   afeId,
                            uint8_t   chno,
                            uint8_t   maEnable,
                            uint16_t  maNumSample,
                            uint16_t  maWindowCntr,
                            uint16_t  maWindowCntrTh,
                            uint16_t  maThreshB0,
                            uint16_t  maThreshB1,
                            uint16_t  maThreshComb
                          )
```

Configure the Moving Average PAP Detector.

Configure the Moving Average PAP Detector. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
maEnable	0: Disable Moving Average based PAP detector. 1:Enable Moving Average based PAP detector.
maNumSample	Number of samples in a window. Supported values: 1-32; 2-64; 3-128 Samples
maWindowCntr	Number of windows. Supported Range: 0 to 2**12-1
maWindowCntrTh	Window Counter Threshold. When the number of windows in a set of maWindowCntr windows have power above the power threshold. This should be lower than or equal to maWindowCntr. Supported Range: 0:2**12-1.
maThreshB0	(128*val) is the power threshold for band 0 detector. Supported Range: 0-511
maThreshB1	(128*val) is the power threshold for band 1 detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.
maThreshComb	(128*val) is the power threshold for combined detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.

Returns

Returns if the function execution passed or failed.

◆ papAlarmStatus()

```
uint8_t papAlarmStatus ( uint8_t   afeId,
                         uint8_t   chno,
                         uint8_t * alarmTriggered
                       )
```

Reads the PAP alarm Status.

Reads the PAP alarm Status and returns it as a pointer. This is sticky status and clearPapAlarms needs to be called to clear the status.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
alarmTriggered	Pointer return of the status. If this value is 1, then there was a PAP trigger.

Returns

Returns if the function execution passed or failed.

◆ rampStickyClear()

```
uint8_t rampStickyClear ( uint8_t afeId,
                          uint8_t chno
                        )
```

Clear the ramp sticky state.

In case where rampStickyMode is set, this function should be called to clear the PAP alarm and move to ramp up state.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

paramsSetterGetter.h File Reference

Go to the source code of this file.

Functions

uint8_t	**set_X** (uint8_t afeId, uint32_t X)

uint8_t	**get_X** (uint8_t afeId, uint32_t *X)
uint8_t	**set_numTxNCO** (uint8_t afeId, uint8_t numTxNCO)
uint8_t	**get_numTxNCO** (uint8_t afeId, uint8_t *numTxNCO)
uint8_t	**set_numRxNCO** (uint8_t afeId, uint8_t numRxNCO)
uint8_t	**get_numRxNCO** (uint8_t afeId, uint8_t *numRxNCO)
uint8_t	**set_numFbNCO** (uint8_t afeId, uint8_t numFbNCO)
uint8_t	**get_numFbNCO** (uint8_t afeId, uint8_t *numFbNCO)
uint8_t	**set_FRef** (uint8_t afeId, float FRef)
uint8_t	**get_FRef** (uint8_t afeId, float *FRef)
uint8_t	**set_FadcRx** (uint8_t afeId, float FadcRx)
uint8_t	**get_FadcRx** (uint8_t afeId, float *FadcRx)
uint8_t	**set_FadcFb** (uint8_t afeId, float FadcFb)
uint8_t	**get_FadcFb** (uint8_t afeId, float *FadcFb)
uint8_t	**set_Fdac** (uint8_t afeId, float Fdac)
uint8_t	**get_Fdac** (uint8_t afeId, float *Fdac)
uint8_t	**set_useSpiSysref** (uint8_t afeId, uint8_t useSpiSysref)
uint8_t	**get_useSpiSysref** (uint8_t afeId, uint8_t *useSpiSysref)
uint8_t	**set_ncoFreqMode** (uint8_t afeId, uint8_t ncoFreqMode)
uint8_t	**get_ncoFreqMode** (uint8_t afeId, uint8_t *ncoFreqMode)
uint8_t	**set_halfRateModeRx** (uint8_t afeId, uint8_t *halfRateModeRx)
uint8_t	**get_halfRateModeRx** (uint8_t afeId, uint8_t *halfRateModeRx)
uint8_t	**set_halfRateModeFb** (uint8_t afeId, uint8_t *halfRateModeFb)
uint8_t	**get_halfRateModeFb** (uint8_t afeId, uint8_t *halfRateModeFb)
uint8_t	**set_halfRateModeTx** (uint8_t afeId, uint8_t *halfRateModeTx)
uint8_t	**get_halfRateModeTx** (uint8_t afeId, uint8_t *halfRateModeTx)
uint8_t	**set_syncLoopBack** (uint8_t afeId, uint8_t syncLoopBack)
uint8_t	**get_syncLoopBack** (uint8_t afeId, uint8_t *syncLoopBack)
uint8_t	**set_ddcFactorRx** (uint8_t afeId, uint8_t *ddcFactorRx)
uint8_t	**get_ddcFactorRx** (uint8_t afeId, uint8_t *ddcFactorRx)
uint8_t	**set_numBandsRx** (uint8_t afeId, uint8_t *numBandsRx)
uint8_t	**get_numBandsRx** (uint8_t afeId, uint8_t *numBandsRx)
uint8_t	**set_ddcFactorFb** (uint8_t afeId, uint8_t *ddcFactorFb)
uint8_t	**get_ddcFactorFb** (uint8_t afeId, uint8_t *ddcFactorFb)
uint8_t	**set_ducFactorTx** (uint8_t afeId, uint8_t *ducFactorTx)
uint8_t	**get_ducFactorTx** (uint8_t afeId, uint8_t *ducFactorTx)
uint8_t	**set_numBandsTx** (uint8_t afeId, uint8_t *numBandsTx)
uint8_t	**get_numBandsTx** (uint8_t afeId, uint8_t *numBandsTx)
uint8_t	**set_enableDacInterleavedMode** (uint8_t afeId, uint8_t enableDacInterleavedMode)
uint8_t	**get_enableDacInterleavedMode** (uint8_t afeId, uint8_t *enableDacInterleavedMode)
uint8_t	**set_txToFbMode** (uint8_t afeId, uint8_t txToFbMode)
uint8_t	**get_txToFbMode** (uint8_t afeId, uint8_t *txToFbMode)
uint8_t	**set_chipId** (uint8_t afeId, uint32_t chipId)
uint8_t	**get_chipId** (uint8_t afeId, uint32_t *chipId)
uint8_t	**set_chipVersion** (uint8_t afeId, uint8_t chipVersion)
uint8_t	**get_chipVersion** (uint8_t afeId, uint8_t *chipVersion)
uint8_t	**set_agcMode** (uint8_t afeId, uint8_t agcMode)
uint8_t	**get_agcMode** (uint8_t afeId, uint8_t *agcMode)

uint8_t	**set_bigStepAttkEn** (uint8_t afeId, uint8_t *bigStepAttkEn)
uint8_t	**get_bigStepAttkEn** (uint8_t afeId, uint8_t *bigStepAttkEn)
uint8_t	**set_smallStepAttkEn** (uint8_t afeId, uint8_t *smallStepAttkEn)
uint8_t	**get_smallStepAttkEn** (uint8_t afeId, uint8_t *smallStepAttkEn)
uint8_t	**set_powerAttkEn** (uint8_t afeId, uint8_t *powerAttkEn)
uint8_t	**get_powerAttkEn** (uint8_t afeId, uint8_t *powerAttkEn)
uint8_t	**set_bigStepDecEn** (uint8_t afeId, uint8_t *bigStepDecEn)
uint8_t	**get_bigStepDecEn** (uint8_t afeId, uint8_t *bigStepDecEn)
uint8_t	**set_smallStepDecEn** (uint8_t afeId, uint8_t *smallStepDecEn)
uint8_t	**get_smallStepDecEn** (uint8_t afeId, uint8_t *smallStepDecEn)
uint8_t	**set_powerDecEn** (uint8_t afeId, uint8_t *powerDecEn)
uint8_t	**get_powerDecEn** (uint8_t afeId, uint8_t *powerDecEn)
uint8_t	**set_bigStepAttkThresh** (uint8_t afeId, uint8_t *bigStepAttkThresh)
uint8_t	**get_bigStepAttkThresh** (uint8_t afeId, uint8_t *bigStepAttkThresh)
uint8_t	**set_smallStepAttkThresh** (uint8_t afeId, uint8_t *smallStepAttkThresh)
uint8_t	**get_smallStepAttkThresh** (uint8_t afeId, uint8_t *smallStepAttkThresh)
uint8_t	**set_powerAttkThresh** (uint8_t afeId, uint8_t *powerAttkThresh)
uint8_t	**get_powerAttkThresh** (uint8_t afeId, uint8_t *powerAttkThresh)
uint8_t	**set_bigStepDecThresh** (uint8_t afeId, uint8_t *bigStepDecThresh)
uint8_t	**get_bigStepDecThresh** (uint8_t afeId, uint8_t *bigStepDecThresh)
uint8_t	**set_smallStepDecThresh** (uint8_t afeId, uint8_t *smallStepDecThresh)
uint8_t	**get_smallStepDecThresh** (uint8_t afeId, uint8_t *smallStepDecThresh)
uint8_t	**set_powerDecThresh** (uint8_t afeId, uint8_t *powerDecThresh)
uint8_t	**get_powerDecThresh** (uint8_t afeId, uint8_t *powerDecThresh)
uint8_t	**set_bigStepAttkWinLen** (uint8_t afeId, uint32_t *bigStepAttkWinLen)
uint8_t	**get_bigStepAttkWinLen** (uint8_t afeId, uint32_t *bigStepAttkWinLen)
uint8_t	**set_miscStepAttkWinLen** (uint8_t afeId, uint32_t *miscStepAttkWinLen)
uint8_t	**get_miscStepAttkWinLen** (uint8_t afeId, uint32_t *miscStepAttkWinLen)
uint8_t	**set_decayWinLen** (uint8_t afeId, uint32_t *decayWinLen)
uint8_t	**get_decayWinLen** (uint8_t afeId, uint32_t *decayWinLen)
uint8_t	**set_jesdProtocol** (uint8_t afeId, uint8_t jesdProtocol)
uint8_t	**get_jesdProtocol** (uint8_t afeId, uint8_t *jesdProtocol)
uint8_t	**set_spiInUseForPllAccess** (uint8_t afeId, uint8_t spiInUseForPllAccess)
uint8_t	**get_spiInUseForPllAccess** (uint8_t afeId, uint8_t *spiInUseForPllAccess)

Function Documentation

◆ get_agcMode()

```
uint8_t get_agcMode ( uint8_t    afeId,
                      uint8_t *  agcMode 
                    )
```

◆ get_bigStepAttkEn()

```
uint8_t get_bigStepAttkEn ( uint8_t    afeId,
                            uint8_t *  bigStepAttkEn
                          )
```

◆ get_bigStepAttkThresh()

```
uint8_t get_bigStepAttkThresh ( uint8_t    afeId,
                                uint8_t *  bigStepAttkThresh
                              )
```

◆ get_bigStepAttkWinLen()

```
uint8_t get_bigStepAttkWinLen ( uint8_t     afeId,
                                uint32_t *  bigStepAttkWinLen
                              )
```

◆ get_bigStepDecEn()

```
uint8_t get_bigStepDecEn ( uint8_t    afeId,
                           uint8_t *  bigStepDecEn
                         )
```

◆ get_bigStepDecThresh()

```
uint8_t get_bigStepDecThresh ( uint8_t    afeId,
                               uint8_t *  bigStepDecThresh
                             )
```

◆ get_chipId()

```
uint8_t get_chipId ( uint8_t     afeId,
                     uint32_t *  chipId
                   )
```

◆ get_chipVersion()

```
uint8_t get_chipVersion ( uint8_t    afeId,
                          uint8_t *  chipVersion
                        )
```

◆ get_ddcFactorFb()

```
uint8_t get_ddcFactorFb ( uint8_t    afeId,
                          uint8_t *  ddcFactorFb
                        )
```

◆ get_ddcFactorRx()

```
uint8_t get_ddcFactorRx ( uint8_t    afeId,
                          uint8_t *  ddcFactorRx
                        )
```

◆ get_decayWinLen()

```
uint8_t get_decayWinLen ( uint8_t     afeId,
                          uint32_t *  decayWinLen
                        )
```

◆ get_ducFactorTx()

```
uint8_t get_ducFactorTx ( uint8_t    afeId,
                          uint8_t *  ducFactorTx
                        )
```

◆ get_enableDacInterleavedMode()

```
uint8_t get_enableDacInterleavedMode ( uint8_t    afeId,
                                       uint8_t *  enableDacInterleavedMode
                                     )
```

◆ get_FadcFb()

```
uint8_t get_FadcFb ( uint8_t  afeId,
                     float *  FadcFb
                   )
```

◆ get_FadcRx()

```
uint8_t get_FadcRx ( uint8_t  afeId,
                     float *  FadcRx
                   )
```

◆ get_Fdac()

```
uint8_t get_Fdac ( uint8_t  afeId,
                   float *  Fdac
                 )
```

◆ get_FRef()

```
uint8_t get_FRef ( uint8_t  afeId,
                   float *  FRef
                 )
```

◆ get_halfRateModeFb()

```
uint8_t get_halfRateModeFb ( uint8_t    afeId,
                             uint8_t *  halfRateModeFb
                           )
```

◆ get_halfRateModeRx()

```
uint8_t get_halfRateModeRx ( uint8_t    afeId,
                             uint8_t *  halfRateModeRx
                           )
```

◆ get_halfRateModeTx()

```
uint8_t get_halfRateModeTx ( uint8_t    afeId,
                             uint8_t *  halfRateModeTx
                           )
```

◆ get_jesdProtocol()

```
uint8_t get_jesdProtocol ( uint8_t    afeId,
                           uint8_t *  jesdProtocol
                         )
```

◆ get_miscStepAttkWinLen()

```
uint8_t get_miscStepAttkWinLen ( uint8_t     afeId,
                                 uint32_t *  miscStepAttkWinLen
                               )
```

◆ get_ncoFreqMode()

```
uint8_t get_ncoFreqMode ( uint8_t    afeId,
                          uint8_t *  ncoFreqMode
                        )
```

◆ get_numBandsRx()

```
uint8_t get_numBandsRx ( uint8_t    afeId,
                         uint8_t *  numBandsRx
                       )
```

◆ get_numBandsTx()

```
uint8_t get_numBandsTx ( uint8_t    afeId,
                         uint8_t *  numBandsTx
                       )
```

◆ get_numFbNCO()

```
uint8_t get_numFbNCO ( uint8_t    afeId,
                       uint8_t *  numFbNCO
                     )
```

◆ get_numRxNCO()

```
uint8_t get_numRxNCO ( uint8_t    afeId,
                       uint8_t *  numRxNCO
                     )
```

◆ get_numTxNCO()

```
uint8_t get_numTxNCO ( uint8_t    afeId,
                       uint8_t *  numTxNCO
                     )
```

◆ get_powerAttkEn()

```
uint8_t get_powerAttkEn ( uint8_t    afeId,
                          uint8_t *  powerAttkEn
                        )
```

◆ get_powerAttkThresh()

```
uint8_t get_powerAttkThresh ( uint8_t    afeId,
                              uint8_t *  powerAttkThresh
                            )
```

◆ get_powerDecEn()

```
uint8_t get_powerDecEn ( uint8_t    afeId,
                         uint8_t *  powerDecEn
                       )
```

◆ get_powerDecThresh()

```
uint8_t get_powerDecThresh ( uint8_t    afeId,
                             uint8_t *  powerDecThresh
                           )
```

◆ get_smallStepAttkEn()

```
uint8_t get_smallStepAttkEn ( uint8_t    afeId,
                              uint8_t *  smallStepAttkEn
                            )
```

◆ get_smallStepAttkThresh()

```
uint8_t get_smallStepAttkThresh ( uint8_t    afeId,
                                  uint8_t *  smallStepAttkThresh
                                )
```

◆ get_smallStepDecEn()

```
uint8_t get_smallStepDecEn ( uint8_t    afeId,
                             uint8_t *  smallStepDecEn
                           )
```

◆ get_smallStepDecThresh()

```
uint8_t get_smallStepDecThresh ( uint8_t    afeId,
                                 uint8_t *  smallStepDecThresh
                               )
```

◆ get_spiInUseForPllAccess()

```
uint8_t get_spiInUseForPllAccess ( uint8_t    afeId,
                                   uint8_t *  spiInUseForPllAccess
                                 )
```

◆ get_syncLoopBack()

```
uint8_t get_syncLoopBack ( uint8_t    afeId,
                           uint8_t *  syncLoopBack
                         )
```

◆ get_txToFbMode()

```
uint8_t get_txToFbMode ( uint8_t    afeId,
                         uint8_t *  txToFbMode
                       )
```

◆ get_useSpiSysref()

```
uint8_t get_useSpiSysref ( uint8_t    afeId,
                           uint8_t *  useSpiSysref
                         )
```

◆ get_X()

```
uint8_t get_X ( uint8_t     afeId,
                uint32_t *  X
              )
```

◆ set_agcMode()

```
uint8_t set_agcMode ( uint8_t  afeId,
                      uint8_t  agcMode
                    )
```

◆ set_bigStepAttkEn()

```
uint8_t set_bigStepAttkEn ( uint8_t    afeId,
                            uint8_t *  bigStepAttkEn
                          )
```

◆ set_bigStepAttkThresh()

```
uint8_t set_bigStepAttkThresh ( uint8_t    afeId,
                                uint8_t *  bigStepAttkThresh
                              )
```

◆ set_bigStepAttkWinLen()

```
uint8_t set_bigStepAttkWinLen ( uint8_t     afeId,
                                uint32_t *  bigStepAttkWinLen
                              )
```

◆ set_bigStepDecEn()

```
uint8_t set_bigStepDecEn ( uint8_t    afeId,
                           uint8_t *  bigStepDecEn
                         )
```

◆ set_bigStepDecThresh()

```
uint8_t set_bigStepDecThresh ( uint8_t    afeId,
                               uint8_t *  bigStepDecThresh
                             )
```

◆ set_chipId()

```
uint8_t set_chipId ( uint8_t   afeId,
                     uint32_t  chipId
                   )
```

◆ set_chipVersion()

```
uint8_t set_chipVersion ( uint8_t  afeId,
                          uint8_t  chipVersion
                        )
```

◆ set_ddcFactorFb()

```
uint8_t set_ddcFactorFb ( uint8_t    afeId,
                          uint8_t *  ddcFactorFb
                        )
```

◆ set_ddcFactorRx()

```
uint8_t set_ddcFactorRx ( uint8_t    afeId,
                          uint8_t *  ddcFactorRx
                        )
```

◆ set_decayWinLen()

```
uint8_t set_decayWinLen ( uint8_t     afeId,
                          uint32_t *  decayWinLen
                        )
```

◆ set_ducFactorTx()

```
uint8_t set_ducFactorTx ( uint8_t    afeId,
                          uint8_t *  ducFactorTx
                        )
```

◆ set_enableDacInterleavedMode()

```
uint8_t set_enableDacInterleavedMode ( uint8_t  afeId,
                                       uint8_t  enableDacInterleavedMode
                                     )
```

◆ set_FadcFb()

```
uint8_t set_FadcFb ( uint8_t  afeId,
                     float    FadcFb
                   )
```

◆ set_FadcRx()

```
uint8_t set_FadcRx ( uint8_t  afeId,
                     float    FadcRx
                   )
```

◆ set_Fdac()

```
uint8_t set_Fdac ( uint8_t  afeId,
                   float    Fdac
                 )
```

◆ set_FRef()

```
uint8_t set_FRef ( uint8_t  afeId,
                   float    FRef
                 )
```

◆ set_halfRateModeFb()

```
uint8_t set_halfRateModeFb ( uint8_t    afeId,
                             uint8_t *  halfRateModeFb
                           )
```

◆ set_halfRateModeRx()

```
uint8_t set_halfRateModeRx ( uint8_t    afeId,
                             uint8_t *  halfRateModeRx
                           )
```

◆ set_halfRateModeTx()

```
uint8_t set_halfRateModeTx ( uint8_t    afeId,
                             uint8_t *  halfRateModeTx
                           )
```

◆ set_jesdProtocol()

```
uint8_t set_jesdProtocol ( uint8_t  afeId,
                           uint8_t  jesdProtocol
                         )
```

◆ set_miscStepAttkWinLen()

```
uint8_t set_miscStepAttkWinLen ( uint8_t     afeId,
                                 uint32_t *  miscStepAttkWinLen
                               )
```

◆ set_ncoFreqMode()

```
uint8_t set_ncoFreqMode ( uint8_t  afeId,
                          uint8_t  ncoFreqMode
                        )
```

◆ set_numBandsRx()

```
uint8_t set_numBandsRx ( uint8_t   afeId,
                         uint8_t * numBandsRx
                       )
```

◆ set_numBandsTx()

```
uint8_t set_numBandsTx ( uint8_t   afeId,
                         uint8_t * numBandsTx
                       )
```

◆ set_numFbNCO()

```
uint8_t set_numFbNCO ( uint8_t afeId,
                       uint8_t numFbNCO
                     )
```

◆ set_numRxNCO()

```
uint8_t set_numRxNCO ( uint8_t afeId,
                       uint8_t numRxNCO
                     )
```

◆ set_numTxNCO()

```
uint8_t set_numTxNCO ( uint8_t afeId,
                       uint8_t numTxNCO
                     )
```

◆ set_powerAttkEn()

```
uint8_t set_powerAttkEn ( uint8_t   afeId,
                          uint8_t * powerAttkEn
                        )
```

◆ set_powerAttkThresh()

```
uint8_t set_powerAttkThresh ( uint8_t   afeId,
                              uint8_t * powerAttkThresh
                            )
```

◆ set_powerDecEn()

```
uint8_t set_powerDecEn ( uint8_t   afeId,
                         uint8_t * powerDecEn
                       )
```

◆ set_powerDecThresh()

```
uint8_t set_powerDecThresh ( uint8_t   afeId,
                             uint8_t * powerDecThresh
                           )
```

◆ set_smallStepAttkEn()

```
uint8_t set_smallStepAttkEn ( uint8_t   afeId,
                              uint8_t * smallStepAttkEn
                            )
```

◆ set_smallStepAttkThresh()

```
uint8_t set_smallStepAttkThresh ( uint8_t   afeId,
                                  uint8_t * smallStepAttkThresh
                                )
```

◆ set_smallStepDecEn()

```
uint8_t set_smallStepDecEn ( uint8_t   afeId,
                             uint8_t * smallStepDecEn
                           )
```

◆ set_smallStepDecThresh()

```
uint8_t set_smallStepDecThresh ( uint8_t   afeId,
                                 uint8_t * smallStepDecThresh
                               )
```

◆ set_spiInUseForPllAccess()

```
uint8_t set_spiInUseForPllAccess ( uint8_t afeId,
                                   uint8_t spiInUseForPllAccess
                                 )
```

◆ set_syncLoopBack()

```
uint8_t set_syncLoopBack ( uint8_t  afeId,
                           uint8_t  syncLoopBack
                         )
```

◆ set_txToFbMode()

```
uint8_t set_txToFbMode ( uint8_t  afeId,
                         uint8_t  txToFbMode
                       )
```

◆ set_useSpiSysref()

```
uint8_t set_useSpiSysref ( uint8_t  afeId,
                           uint8_t  useSpiSysref
                         )
```

◆ set_X()

```
uint8_t set_X ( uint8_t   afeId,
                uint32_t  X
              )
```

Generated by doxygen 1.8.17

serDes.h File Reference

Go to the source code of this file.

Functions

uint8_t	**serdesTx1010Pattern** (uint8_t afeId, uint8_t laneNo) Send 1010 toggling pattern on AFE SerDes TX. More...
uint8_t	**serdesTxSendData** (uint8_t afeId, uint8_t laneNo) Send JESD data on AFE SerDes TX. More...
uint8_t	**SetSerdesTxCursor** (uint8_t afeId, uint8_t laneNo, uint8_t mainCursorSetting, uint8_t preCursorSetting, uint8_t postCursorSetting) Set SerDes TX Cursor. More...
uint8_t	**getSerdesRxPrbsError** (uint8_t afeId, uint8_t laneNo, uint32_t *errorRegValue) Read the AFE SerDes RX PRBS error. More...
uint8_t	**clearSerdesRxPrbsErrorCounter** (uint8_t afeId, uint8_t laneNo) Clear the AFE SerDes RX PRBS error counter. More...
uint8_t	**enableSerdesRxPrbsCheck** (uint8_t afeId, uint8_t laneNo, uint8_t prbsMode, uint8_t enable) Enables the AFE SerDes RX PRBS check. More...
uint8_t	**sendSerdesTxPrbs** (uint8_t afeId, uint8_t laneNo, uint8_t prbsMode, uint8_t enable) Sends the AFE SerDes TX PRBS pattern. More...
uint8_t	**getSerdesRxLaneEyeMarginValue** (uint8_t afeId, uint8_t laneNo, uint16_t *regValue)

	Reads the AFE SerDes RX Eye margin value. More...
uint8_t	**resetSerDesDfeLane** (uint8_t afeId, uint8_t laneNo) Resets the AFE SerDes RX DFE lane. More...
uint8_t	**reAdaptSerDesLane** (uint8_t afeId, uint8_t laneNo) Readapts the AFE SerDes RX lane. More...
uint8_t	**resetSerDesDfeAllLanes** (uint8_t afeId) Resets DFE of all the AFE SerDes RX lanes. More...
uint8_t	**reAdaptSerDesAllLanes** (uint8_t afeId) Readapts all the AFE SerDes RX lanes. More...
uint8_t	**getSerdesEye** (uint8_t afeId, uint8_t laneNo, uint16_t *ber, uint16_t *extent) Reads the SerDes Eye for a given lane. More...

Function Documentation

◆ clearSerdesRxPrbsErrorCounter()

```
uint8_t clearSerdesRxPrbsErrorCounter ( uint8_t  afeId,
                                        uint8_t  laneNo
                                      )
```

Clear the AFE SerDes RX PRBS error counter.

Clearss the AFE SerDes RX PRBS error counter.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ enableSerdesRxPrbsCheck()

```
uint8_t enableSerdesRxPrbsCheck ( uint8_t  afeId,
                                  uint8_t  laneNo,
                                  uint8_t  prbsMode,
                                  uint8_t  enable
                                )
```

Enables the AFE SerDes RX PRBS check.

Enables the AFE SerDes RX PRBS check.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
prbsMode	PRBS Mode Selection. 0 for PRBS9, 1 for PRBS15, 2 for PRBS23 and 3 for PRBS31.
enable	1 will enable the PRBS check, 0 will disable the PRBS check.

Returns

Returns if the function execution passed or failed.

◆ getSerdesEye()

```
uint8_t getSerdesEye ( uint8_t     afeId,
                       uint8_t     laneNo,
                       uint16_t *  ber,
                       uint16_t *  extent
                     )
```

Reads the SerDes Eye for a given lane.

Reads the SerDes Eye for a given lane. The ber array and the extent returned by this function should be fed to the python script to plot the eye diagram.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
ber	Pointer of array with 3135 elements.
extent	scaling factor of the ber needed by the function.

Returns

Returns if the function execution passed or failed.

◆ getSerdesRxLaneEyeMarginValue()

```
uint8_t getSerdesRxLaneEyeMarginValue ( uint8_t    afeId,
                                        uint8_t    laneNo,
                                        uint16_t * regValue
                                      )
```

Reads the AFE SerDes RX Eye margin value.

Reads the AFE SerDes RX Eye margin value and returns the value as a pointer. This value*0.5 is the eye margin in mV post equalization.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
regValue	Eye Margin value status.

Returns

Returns if the function execution passed or failed.

◆ getSerdesRxPrbsError()

```
uint8_t getSerdesRxPrbsError ( uint8_t    afeId,
                               uint8_t    laneNo,
                               uint32_t * errorRegValue
                             )
```

Read the AFE SerDes RX PRBS error.

Reads the AFE SerDes RX PRBS error and returns the error value as pointer.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
errorRegValue	PRBS error register value. This value increments by 3 for each PRBS error.

Returns

Returns if the function execution passed or failed.

◆ reAdaptSerDesAllLanes()

```
uint8_t reAdaptSerDesAllLanes ( uint8_t  afeId )
```

Readapts all the AFE SerDes RX lanes.

Readapts all the AFE SerDes RX lanes. This calls reAdaptSerDesLane within this function for all the lanes.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ reAdaptSerDesLane()

```
uint8_t reAdaptSerDesLane ( uint8_t  afeId,
                            uint8_t  laneNo
                          )
```

Readapts the AFE SerDes RX lane.

Readapts the AFE SerDes RX lane. This calls resetSerDesDfeLane within this function for the specific lane.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ resetSerDesDfeAllLanes()

```
uint8_t resetSerDesDfeAllLanes ( uint8_t  afeId )
```

Resets DFE of all the AFE SerDes RX lanes.

Resets DFE of all the AFE SerDes RX lanes. This calls resetSerDesDfeLane within this function for all the lanes.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ resetSerDesDfeLane()

```
uint8_t resetSerDesDfeLane ( uint8_t  afeId,
                             uint8_t  laneNo
                           )
```

Resets the AFE SerDes RX DFE lane.

Resets the AFE SerDes RX DFE lane.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ sendSerdesTxPrbs()

```
uint8_t sendSerdesTxPrbs ( uint8_t  afeId,
                           uint8_t  laneNo,
                           uint8_t  prbsMode,
                           uint8_t  enable
                         )
```

Sends the AFE SerDes TX PRBS pattern.

Sends the AFE SerDes TX PRBS pattern.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.
prbsMode	PRBS Mode Selection. 0 for PRBS9, 1 for PRBS15, 2 for PRBS23 and 3 for PRBS31.
enable	1 will enable the PRBS transmission, 0 will disable the PRBS pattern transmission.

Returns

Returns if the function execution passed or failed.

◆ serdesTx1010Pattern()

```
uint8_t serdesTx1010Pattern ( uint8_t  afeId,
                              uint8_t  laneNo
                            )
```

Send 1010 toggling pattern on AFE SerDes TX.

Send 1010 toggling pattern on AFE SerDes TX.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ serdesTxSendData()

```
uint8_t serdesTxSendData ( uint8_t  afeId,
                           uint8_t  laneNo
                         )
```

Send JESD data on AFE SerDes TX.

Send JESD data pattern on AFE SerDes TX.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ SetSerdesTxCursor()

```
uint8_t SetSerdesTxCursor ( uint8_t  afeId,
                            uint8_t  laneNo,
                            uint8_t  mainCursorSetting,
                            uint8_t  preCursorSetting,
                            uint8_t  postCursorSetting
                          )
```

Set SerDes TX Cursor.

Set SerDes TX Cursor. Below table shows the mapping between different settings and the equalization it provides.

Column (1):Pre-Cursor equalization acheived. (dB in relative to post cursor)

Column (2):Main Cursor equalization acheived.(dB in relative to pre cursor)

Column (3):Post-Cursor equalization acheived.(dB in relative to pre cursor)

Column (4):Pre-Cursor Setting to be programmed.

Column (5):Main Setting to be programmed.

Column (6):Post-Cursor setting to be programmed.

(1)	(2)	(3)	(4)	(5)	(6)
0	25	0	0	0	0
0	23	0	0	1	0
0	21	0	0	2	0
0	19	0	0	3	0
0	17	0	0	4	0
0	15	0	0	5	0
0	13	0	0	6	0
0	11	0	0	7	0
0	24	0.72	0	0	1
0	20	0.87	0	2	1
0	16	1.09	0	4	1
0	12	1.45	0	6	1
0	23	1.51	0	0	2
0	21	1.66	0	1	2
0	15	2.33	0	4	2
0	22	2.38	0	0	3
0	13	2.69	0	5	2
0	21	3.35	0	0	4
0	19	3.71	0	1	4
0	14	3.78	0	4	3
0	17	4.17	0	2	2
0	20	4.44	0	0	5
0	15	4.75	0	3	2
0	16	5.62	0	2	5
0	19	5.68	0	0	6
0	17	6.41	0	1	6
0	18	7.13	0	0	7
0.72	24	0	1	0	0
0.87	20	0	1	2	0
0.87	22	1.66	1	0	2
1.09	16	0	1	4	0
1.09	20	3.71	1	0	4
1.45	12	0	1	2	0
1.45	14	2.69	1	4	2

1.45	16	4.75	1	2	4
1.45	18	6.41	1	0	6
1.51	23	0	2	0	0
1.66	21	0	2	1	0
1.66	22	0.87	2	0	1
2.33	15	0	2	4	0
2.33	19	4.17	2	0	4
2.38	22	0	3	0	0
2.69	18	5.62	2	0	5
2.69	13	0	2	5	0
2.69	14	1.45	2	4	1
2.69	17	4.75	2	1	4
3.35	21	0	4	0	0
3.71	19	0	4	1	0
3.71	20	1.09	4	0	1
3.78	18	4.75	3	0	4
3.78	14	0	3	4	0
4.17	17	0	4	2	0
4.17	19	2.33	4	0	2
4.44	20	0	5	0	0
4.75	15	0	4	3	0
4.75	16	1.45	4	2	1
4.75	18	3.78	4	0	3
4.75	17	2.69	4	1	2
5.62	16	0	5	2	0
5.62	18	2.69	5	0	2
5.68	19	0	6	0	0
6.41	18	1.45	6	0	1
6.41	17	0	6	1	0
7.13	18	0	7	0	0

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.
mainCursorSetting	Main Cursor Setting.
preCursorSetting	Pre Cursor Setting.
postCursorSetting	Post Cursor Setting.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

Src Directory Reference

Files

file **agc.c**

This file has AGC related functions.

Version 2.2:

file **basicFunctions.c**

This file has Basic SPI functions.

Version 2.2:

file **calibrations.c**

This file has Factory calibration related functions.

Version 2.1:

file **controls.c**

This file has generic control related functions.

Version 2.2:

file **dsaAndNco.c**

This file has DSA and NCO related functions.

Version 2.2:

file **hMacro.c**

This file has Macros related functions.

Version 2.1:

file **init.c**

file **jesd.c**

This file has JESD related functions.

Version 2.2:

file **pap.c**

This file has PAP related functions.

Version 2.1:

file **serDes.c**

This file has SerDes related functions.

Version 2.2:

Generated by doxygen 1.8.17

agc.c File Reference

This file has AGC related functions.

Version 2.2:

More...

```
#include <stdint.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "afeParameters.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "agc.h"
#include "hMacro.h"
```

Functions

uint8_t **agcStateControlConfig** (uint8_t afeId, uint8_t chNo, uint16_t agcstate)

AGC State Control Macro. More...

uint8_t **agcDigDetConfig** (uint8_t afeId, uint8_t chNo, uint8_t bigStepAttkEn, uint8_t smallStepAttkEn, uint8_t bigStepDecEn, uint8_t smallStepDecEn, uint8_t powerAttkEn, uint8_t powerDecEn, uint8_t bigStepAttkThresh, uint8_t smallStepAttkThresh, uint8_t bigStepDecThresh, uint8_t smallStepDecThresh, uint8_t powerAttkThresh, uint8_t powerDecThresh)

ADC Digital Detector Threshold configuration. More...

uint8_t **agcDigDetTimeConstantConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkWinLen, uint32_t miscStepAttkWinLen, uint32_t decayWinLen)

ADC Digital Detector Window Length configuration. More...

uint8_t **agcDigDetAbsoluteNumCrossingConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkNumHits, uint32_t smallStepAttkNumHits, uint32_t bigStepDecNumHits, uint32_t smallStepDecNumHits)

ADC Digital Detector Absolute NumHits configuration. More...

uint8_t **agcDigDetRelativeNumCrossingConfig** (uint8_t afeId, uint8_t chNo, uint32_t bigStepAttkNumHits, uint32_t smallStepAttkNumHits, uint32_t bigStepDecNumHits, uint32_t smallStepDecNumHits)

ADC Digital Detector Relative NumHits configuration. More...

uint8_t **externalAgcConfig** (uint8_t afeId, uint8_t chNo, uint16_t pin0sel, uint16_t pin1sel, uint16_t pin2sel, uint16_t pin3sel, uint8_t pkDetPinLsbSel, uint8_t pulseExpansionCount, uint8_t noLsbsToSend)

External AGC Configuration. More...

uint8_t **minMaxDsaAttnConfig** (uint8_t afeId, uint8_t chNo, uint8_t minDsaAttn, uint8_t maxDsaAttn)

Internal AGC Min-Max Attenuation Configuration. More...

uint8_t **agcGainStepSizeConfig** (uint8_t afeId, uint8_t chNo, uint8_t bigStepAttkStepSize, uint8_t smallStepAttkStepSize, uint8_t bigStepDecayStepSize, uint8_t smallStepDecayStepSize)

Internal AGC Gain-Step Configuration. More...

uint8_t **internalAgcConfig** (uint8_t afeId, uint8_t chNo, uint8_t tdd_freeze_agc, uint16_t blank_time_extcomp, uint8_t en_agcfreeze_pin, uint8_t extCompControlEn)

Internal AGC Configuration. More...

uint8_t **rfAnalogDetConfig** (uint8_t afeId, uint8_t chNo, uint8_t rfdeten, uint8_t rfDetMode, uint8_t rfDetNumHitsMode, uint32_t rfdetnumhits, uint8_t rfdetThreshold, uint8_t rfdetstepsize)

Analog RF Detector Configuration. More...

uint8_t **extLnaConfig** (uint8_t afeId, uint8_t chNo, uint8_t singleDualBandMode, uint8_t lnaGainMargin, uint8_t enBandDet, uint8_t tapOffPoint)

External LNA Configuration. More...

uint8_t **extLnaGainConfig** (uint8_t afeId, uint8_t chNo, uint16_t lnaGainB0, uint16_t lnaPhaseB0, uint16_t lnaGainB1, uint16_t lnaPhaseB1)

External LNA Fixed Gain Configuration. More...

uint8_t **alcConfig** (uint8_t afeId, uint8_t chNo, uint8_t alcMode, uint8_t totalGainRange, uint8_t minAttnAlc, uint8_t useMinAttnAgc)

ALC Configuration. More...

uint8_t **fltPtConfig** (uint8_t afeId, uint8_t chNo, uint8_t fltPtMode, uint8_t fltPtFmt)

Floating Point Configuration. More...

uint8_t **coarseFineConfig** (uint8_t afeId, uint8_t chNo, uint8_t stepSize, uint8_t nBitIndex, uint8_t indexInvert, uint8_t indexSwapIQ, uint8_t sigBackOff, uint8_t gainChangeIndEn)

Coarse-Fine Mode Configuration. More...

Detailed Description

This file has AGC related functions.

Version 2.2:

1. Updated the agcStateControlMacro description
2. Fixed macro opcode bug in agcDigDetRelativeNumCrossingConfig.

   Version 2.1.1:

1. Added more functions. agcDigDetRelativeNumCrossingConfig, externalAgcConfig, internalAgcConfig, rfAnalogDetConfig, extLnaConfig, extLnaGainConfig
2. Removed Automatically calling State Control Macro to giver user better flexibility.

Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Changed the C macro for executing the executeMacro function to AFE_FUNC_EXEC.
3. Changed hard coded OPCODES to #defines in **afe79xxTypes.h**.
4. Changed the C macros for all the spi wrapper function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.

Function Documentation

◆ agcDigDetAbsoluteNumCrossingConfig()

```
uint8_t agcDigDetAbsoluteNumCrossingConfig ( uint8_t   afeId,
                                             uint8_t   chNo,
                                             uint32_t  bigStepAttkNumHits,
                                             uint32_t  smallStepAttkNumHits,
                                             uint32_t  bigStepDecNumHits,
                                             uint32_t  smallStepDecNumHits
                                           )
```

ADC Digital Detector Absolute NumHits configuration.

ADC Digital Detector Absolute NumHits configuration. This represents the exact number of crossings threshold and this may need to be adjusted whenever window length is reconfigured to ensure the NumHits threshold will be lower than the Window Length configuration.
Note that only this function or agcDigDetAbsoluteNumCrossingConfig should be used. Both shouldn't be called.
agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkNumHits	Absolute Number of Threshold crossing hits threshold of big step attack detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
smallStepAttkNumHits	Absolute Number of Threshold crossing hits threshold of Small step attack detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
bigStepDecNumHits	Absolute Number of Threshold crossing hits threshold of big step decay detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.
smallStepDecNumHits	Absolute Number of Threshold crossing hits threshold of small step decay detectors. Range is 0-AFE_AGC_MAX_ABS_NUM_HITS.

Returns

Returns if the function execution passed or failed.

◆ agcDigDetConfig()

```
uint8_t agcDigDetConfig ( uint8_t  afeId,
                          uint8_t  chNo,
                          uint8_t  bigStepAttkEn,
                          uint8_t  smallStepAttkEn,
                          uint8_t  bigStepDecEn,
                          uint8_t  smallStepDecEn,
                          uint8_t  powerAttkEn,
                          uint8_t  powerDecEn,
                          uint8_t  bigStepAttkThresh,
                          uint8_t  smallStepAttkThresh,
                          uint8_t  bigStepDecThresh,
                          uint8_t  smallStepDecThresh,
                          uint8_t  powerAttkThresh,
                          uint8_t  powerDecThresh
                        )
```

ADC Digital Detector Threshold configuration.

Enables or disables the detectors. agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkEn	0 disables and 1 enables the corresponding detector.
smallStepAttkEn	0 disables and 1 enables the corresponding detector.
bigStepDecEn	0 disables and 1 enables the corresponding detector.
smallStepDecEn	0 disables and 1 enables the corresponding detector.
powerAttkEn	0 disables and 1 enables the corresponding detector.
powerDecEn	0 disables and 1 enables the corresponding detector.
bigStepAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
smallStepAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
bigStepDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
smallStepDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
powerAttkThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).
powerDecThresh	This -threshValue/4 is the threshold set. That is, to set the threshold of -2.25dbfs, this value should be 2.25*4=9. Supported range is 0-(AFE_RX_DSA_MAX_ANA_DSA_DB*4).

Returns

Returns if the function execution passed or failed.

◆ agcDigDetRelativeNumCrossingConfig()

```
uint8_t agcDigDetRelativeNumCrossingConfig ( uint8_t   afeId,
                                             uint8_t   chNo,
                                             uint32_t  bigStepAttkNumHits,
                                             uint32_t  smallStepAttkNumHits,
                                             uint32_t  bigStepDecNumHits,
                                             uint32_t  smallStepDecNumHits
                                           )
```

ADC Digital Detector Relative NumHits configuration.

ADC Digital Detector Relative NumHits configuration. This specifies the threshold relative to the window length of the corresponding detector. The advantage of this approach is, this will scale automatically when the window length is changed.
Note that only this function or agcDigDetAbsoluteNumCrossingConfig should be used. Both shouldn't be called.
agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkNumHits	Relative Number of Threshold crossing hits threshold of big step attack detectors. The window counter threhold is (value*bigStepAttkWinLen/2^16). Range is 0-0xffff.
smallStepAttkNumHits	Relative Number of Threshold crossing hits threshold of Small step attack detectors. The window counter threhold is (value*smallStepAttkNumHits/2^16).Range is 0-0xffff.
bigStepDecNumHits	Relative Number of Threshold crossing hits threshold of big step decay detectors. The window counter threhold is (value*bigStepDecNumHits/2^16). Range is 0-0xffff.
smallStepDecNumHits	Relative Number of Threshold crossing hits threshold of small step decay detectors. The window counter threhold is (value*smallStepDecNumHits/2^16). Range is 0-0xffff.

Returns

Returns if the function execution passed or failed.

◆ agcDigDetTimeConstantConfig()

```
uint8_t agcDigDetTimeConstantConfig ( uint8_t   afeId,
                                      uint8_t   chNo,
                                      uint32_t  bigStepAttkWinLen,
                                      uint32_t  miscStepAttkWinLen,
                                      uint32_t  decayWinLen
                                    )
```

ADC Digital Detector Window Length configuration.

Configures the Window Length (or time constant) of the detectors. agcStateControlConfig function should be called with internal or external AGC enable appropriately after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkWinLen	Window length of big step attack and RF Analog (also called Customer RF) detectors. Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.
miscStepAttkWinLen	Window length of all other attack detectors. Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.
decayWinLen	Window Length of all the decay detectors.Window length is this value *10ns. Range is 0-AFE_AGC_MAX_WIN_LEN.

Returns

Returns if the function execution passed or failed.

◆ agcGainStepSizeConfig()

```
uint8_t agcGainStepSizeConfig ( uint8_t  afeId,
                                uint8_t  chNo,
                                uint8_t  bigStepAttkStepSize,
                                uint8_t  smallStepAttkStepSize,
                                uint8_t  bigStepDecayStepSize,
                                uint8_t  smallStepDecayStepSize
                              )
```

Internal AGC Gain-Step Configuration.

Configures the Step size of the AGC (DSA index by which to change on detector triggering).
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
bigStepAttkStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
smallStepAttkStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
bigStepDecayStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)
smallStepDecayStepSize	Step Size of big step attack detector. (1LSB = 0.5dB)

Returns

Returns if the function execution passed or failed.

◆ agcStateControlConfig()

```
uint8_t agcStateControlConfig ( uint8_t   afeId,
                                uint8_t   chNo,
                                uint16_t  agcstate
                              )
```

AGC State Control Macro.

Controls the state of the AGC

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
agcstate	Bit wise parameter controlling the state of the AGC. Making a bit 1 does the corresponding operation. Bit 0: Start Internal AGC with entire configuration redone Bit 1: Freeze the Internal AGC loop Bit 2: Unfreeze the Internal AGC loop (takes effect only if the loop is already in freeze) Bit 3: Disable Internal AGC loop Bit 4: ALC Block enable Bit 5: ALC Block disable Bit 6: External AGC enable Bit 7: External AGC disable Bit 8: Restart the Internal AGC. (Step1: Disable Internal AGC, Step2:Enable Internal AGC) Bit 9: Restart ALC(Step1: Disable ALC, Step2:Enable ALC) Bit 10: Restart external AGC(Step1: Disable external AGC, Step2:Enable external AGC) All the bits should not be set together.For example, the enables and disables should not be set together. Invalid combinations include: 1. No other AGC related bit should be enabled when AGC enable is 1. 2. Enable and disable of the ALC should not be set at the same time. 3. Enable and disable of the AGC should not be set at the same time.

Returns

Returns if the function execution passed or failed.

◆ alcConfig()

```
uint8_t alcConfig ( uint8_t  afeId,
                    uint8_t  chNo,
                    uint8_t  alcMode,
                    uint8_t  totalGainRange,
                    uint8_t  minAttnAlc,
                    uint8_t  useMinAttnAgc
                  )
```

ALC Configuration.

Configures ALC. Note that this only informs the MCU of the mode. agcStateControlConfig function should be called with appropriate parameter after this to enable or disable it. agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
alcMode	ALC Mode #0: Floatingpoint #2: coarsefineI #3: coarsefineIQ #4: coarsefineALCpin #5: inputALC
totalGainRange	Total gain range used by ALC for gain compensation. should be <AFE_RX_DSA_MAX_ANA_DSA_DB
minAttnAlc	Minimum Attenuation used by ALC for compensation when useMinAttnAgc = 0. should be <32. Value doesn;t matter when useMinAttnAgc=1
useMinAttnAgc	Configure the Min Attenuation Mode. 0: Use minAttnAlc for minimum attenuation for which compensation is required. 1: Enable ALC to use minimum attenuation from AGC for which compensation is required.

Returns

Returns if the function execution passed or failed.

◆ coarseFineConfig()

```
uint8_t coarseFineConfig ( uint8_t  afeId,
                           uint8_t  chNo,
                           uint8_t  stepSize,
                           uint8_t  nBitIndex,
                           uint8_t  indexInvert,
                           uint8_t  indexSwapIQ,
                           uint8_t  sigBackOff,
                           uint8_t  gainChangeIndEn
                         )
```

Coarse-Fine Mode Configuration.

Configures Coarse-Fine Mode related parameters. This needs to be called only when alcMode in alcConfig is set to coarse-fine mode. Note that this only informs the MCU of the mode.
agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
stepSize	Choose the coarse step size. Appropriate value has to be chosen which can represent the complete attenuation range of operation. 0x00 → 0 dB 0x01 → 1 dB 0x02 → 2 dB 0x03 → 3 dB 0x04 → 4 dB 0x05 → 5 dB 0x06 → 6 dB 0x08 → 8 dB
nBitIndex	Choose the number of bits of coarse index. Supported Values are 0,2,3,4.
indexInvert	Coarse Index Invert. If this value is 0: coarse index is transmitted as is. 1: (15-coarse index) is transmitted
indexSwapIQ	Coarse Index Swap. If to swap coarse index on I and Q. 0: LSB on I, MSB on Q 1: MSB on I, LSB on Q
sigBackOff	This is the signal back-off, the offset attenuation applied. (in dB) This should be less than totalGainRange.
gainChangeIndEn	Applicable only when nBitIndex is 3. If this is set, in the bit-4 indicates if the DSA changed. Otherwise, 0 will be sent.

Returns

Returns if the function execution passed or failed.

◆ externalAgcConfig()

```
uint8_t externalAgcConfig ( uint8_t  afeId,
                            uint8_t  chNo,
                            uint16_t pin0sel,
                            uint16_t pin1sel,
                            uint16_t pin2sel,
                            uint16_t pin3sel,
                            uint8_t  pkDetPinLsbSel,
                            uint8_t  pulseExpansionCount,
                            uint8_t  noLsbsToSend
                          )
```

External AGC Configuration.

Configures the External AGC.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
pin0sel	Pin0/I_BIT0 configuration. Determines what detectors come out. It can be configured to carry ORed combination of selected bits. Setting a particular bit gets the detector on to the corresponding pin/LSB. Bit 14: OVR Bit Bit 13: Band 0 power detector Bit 12: Band 0 peak detector Bit 11: RF detector Bit 10: Band 1 power detector Bit 9: Band 1 peak detector Bit 8: Reserved Bit 7: Digital big step attack Bit 6: Digital small step attack Bit 5: Digital big step decay Bit 4: Digital small step decay Bit 3: Dig power attack Bit 2: Dig power decay Bit 1: Reserved (0) Bit 0: Reserved (0)
pin1sel	Pin1/I_BIT1 configuration. Determines what detectors come out. Description same as pin1Sel.
pin2sel	Pin2/Q_BIT0 configuration. Determines what detectors come out. Description same as pin2Sel.
pin3sel	Pin3/Q_BIT1 configuration. Determines what detectors come out. Description same as pin30Sel.
pulseExpansionCount	Pulse Expansion Count. This value here is in steps of 10 ns. This pulseExpansionCount*10ns is the pulse width.Supported Range: 0-0xff
pkDetPinLsbSel	Determines whether to send detector data on LSB in External AGC mode. 0: send on Pin 1: send on Pin and LSB
noLsbsToSend	0-Send only on Bits 0 of I and Q. 1- Send on both Bits 0 and 1.

Returns

Returns if the function execution passed or failed.

◆ extLnaConfig()

```
uint8_t extLnaConfig ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  singleDualBandMode,
                       uint8_t  lnaGainMargin,
                       uint8_t  enBandDet,
                       uint8_t  tapOffPoint
                     )
```

External LNA Configuration.

Configures External LNA Configuration.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
singleDualBandMode	Whether to use Single LNA control or dual LNA control in dual-band configuration. 0: Single LNA control, 1: Dual LNA control
lnaGainMargin	LNA gain margin (this value is in dB scale where 1 LSB= 0.5 dB) LNA reenable will happen when Current DSA Attenuation ≤ Maximum DSA Attenuation - LNA Gain - LNA Gain Margin in Single LNA Control Mode. Not Applicable in Dual LNA Control
enBandDet	0: Disable Band Detectors 1: Enable band detectors Applicable only when Dual LNA control is enabled.
tapOffPoint	Band Detector Bandwidth Selection (Applicable only when dual LNA control and band detectors are enabled) 0: Higher bandwidth 1: Output bandwidth

Returns

Returns if the function execution passed or failed.

◆ extLnaGainConfig()

```
uint8_t extLnaGainConfig ( uint8_t   afeId,
                           uint8_t   chNo,
                           uint16_t  lnaGainB0,
                           uint16_t  lnaPhaseB0,
                           uint16_t  lnaGainB1,
                           uint16_t  lnaPhaseB1
                         )
```

External LNA Fixed Gain Configuration.

External LNA Fixed Gain Configuration.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
lnaGainB0	LNA Gain for Band 0 in dB. (1LSB=1/32dB). Supported Range 0-0x7ff.
lnaPhaseB0	LNA Phase for Band 0 in degrees. (1LSB=360/1024 degrees). Supported Range 0-0x3ff.
lnaGainB1	LNA Gain for Band 1 in dB. Valid only in dual band operation with dual LNA control enabled. (1LSB=1/32dB). Supported Range 0-0x7ff.
lnaPhaseB1	LNA Phase for Band 1 in degrees. Valid only in dual band operation with dual LNA control enabled. (1LSB=360/1024 degrees). Supported Range 0-0x3ff.

Returns

Returns if the function execution passed or failed.

◆ fltPtConfig()

```
uint8_t fltPtConfig ( uint8_t  afeId,
                      uint8_t  chNo,
                      uint8_t  fltPtMode,
                      uint8_t  fltPtFmt
                    )
```

Floating Point Configuration.

Configures Floating Point Mode related parameters. This needs to be called only when alcMode in alcConfig is set to floating point mode. Note that this only informs the MCU of the mode.
agcStateControlConfig function should be called with ALC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fltPtMode	ALC Floating Point Mode. Sets whether to send MSB of mantissa always in Floating Point mode of ALC. 0: If exponent > 0, do not send MSB 1: Send MSB always
fltPtFmt	Floating Point Format. Number of Mantissa and Exponent bits to be used in floating point mode of ALC 0: 2 bit exponent , 13 bit mantissa and 1 bit sign 1: 3 bit exponent, 12 bit mantissa and 1 bit sign 2: 4 bit exponent, 11 bit mantissa and 1 bit sign

Returns

Returns if the function execution passed or failed.

◆ internalAgcConfig()

```
uint8_t internalAgcConfig ( uint8_t   afeId,
                            uint8_t   chNo,
                            uint8_t   tdd_freeze_agc,
                            uint16_t  blank_time_extcomp,
                            uint8_t   en_agcfreeze_pin,
                            uint8_t   extCompControlEn
                          )
```

Internal AGC Configuration.

Configures the internal AGC related settings.
agcStateControlConfig function should be called with internal AGC enable after this function call to update the configuration.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
tdd_freeze_agc	Whether to reset or freeze the attack detectors during the OFF period of TDD. 0: Reset 1: Freeze
blank_time_extcomp	Blanking Time for all Detectors when external component (LNA or DVGA) gain change. This is interpreted as number of clocks of FadcRx/8. Supported range: 0-0xffff
en_agcfreeze_pin	Enable or Disable pin based AGC freeze. 0: Disable 1: Enable
extCompControlEn	External Component control to enable. 0x00: Neither of the controls are active 0x01: External LNA control is active 0x02: External DVGA control is active Others: invalid

Returns

Returns if the function execution passed or failed.

◆ minMaxDsaAttnConfig()

```
uint8_t minMaxDsaAttnConfig ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  minDsaAttn,
                              uint8_t  maxDsaAttn
                            )
```

Internal AGC Min-Max Attenuation Configuration.

Configures the Minimum and Maximum DSA index between which the internal AGC operates. This is a dynamic Macro and AGC state macro needn't be called after this.

Parameters

afeId	AFE ID
chNo	Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
minDsaAttn	Minimum DSA index. (1LSB = 0.5dB)
maxDsaAttn	Maximum DSA index. (1LSB = 0.5dB)

Returns

Returns if the function execution passed or failed.

◆ rfAnalogDetConfig()

```
uint8_t rfAnalogDetConfig ( uint8_t   afeId,
                            uint8_t   chNo,
                            uint8_t   rfdeten,
                            uint8_t   rfDetMode,
                            uint8_t   rfDetNumHitsMode,
                            uint32_t  rfdetnumhits,
                            uint8_t   rfdetThreshold,
                            uint8_t   rfdetstepsize
                          )
```

Analog RF Detector Configuration.

Analog RF Detector Configuration. agcStateControlConfig function should be called with internal or external AGC enable after this function call to update the configuration.

Parameters

afeId AFE ID

chNo Bit wise channel select
Bit0 for RXA
Bit1 for RXB
Bit2 for RXC
Bit3 for RXD

rfdeten Use RF Analog detector for internal AGC. 0-Disable. 1-Enable

rfDetMode Mode to use the RF Analog Detector Mode in AGC.
0: extAgc: Use RF Analog detector in External AGC.
1: bigStepAtk : Use RF Analog detector as very big step attack in internal AGC.
2: lnaBypass : Use RF Analog detector for external LNA bypass in internal AGC.

rfDetNumHitsMode Mode of input of the rfDetNumHitsMode. 0- Absolute. 1- Relative

rfdetnumhits When rfDetNumHitsMode=0, this is the absolute Number of times signal crosses threshold above which attack is declared. This detector operates at FadcRx rate. Supported Range: <2^32.
When rfDetNumHitsMode=1, this is the relative Number of times signal crosses threshold above which attack is declared. The actual threshold is floor(rfdetnumhits*bigStepAttkWinLen/2^32). Supported Range: <2^32.

rfdetThreshold RF detect Threshold in dBm (for rfDetMode= 0 or 2) and in dbfs (for rfDetMode =1)

rfdetstepsize Step Size of big step attack in dB. Valid only when rfDetMode=1

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

basicFunctions.c File Reference

This file has Basic SPI functions.

Version 2.2:

More...

```
#include <stdint.h>
#include <stdio.h>
#include "afe79xxTypes.h"
#include "afe79xxLog.h"
#include "afeCommonMacros.h"
```

```
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
```

Macros

#define	**MASK_BYTE**(lsb, msb) (uint8_t)(((1 << ((msb) - (lsb) + 1)) - 1) << lsb)
#define	**MASK_SHORT**(lsb, msb) (uint16_t)(((1 << ((msb) - (lsb) + 1)) - 1) << lsb)
#define	**CFG_SPI_READ_POLL_MAX_COUNT** 500
#define	**AFE_REQ_SPI_ACCESS_MAX_COUNT** 100

Typedefs

typedef enum **PLL_SPI_REG_TYPE**   **PllSpiRegType_e**

Enumerations

enum   **PLL_SPI_REG_TYPE** { **PLL_SPI_REG_OFF** = 0, **PLL_SPI_REG_A**, **PLL_SPI_REG_B**, **PLL_SPI_REG_TYPE_SIZE** }

Functions

uint8_t	**serdesRawRead** (uint8_t afeId, uint16_t addr, uint16_t *readVal) SerDes Read. More...
uint8_t	**serdesRawWrite** (uint8_t afeId, uint16_t addr, uint16_t data) SerDes Write. More...
uint8_t	**afeSpiWriteWrapper** (uint8_t afeId, uint16_t addr, uint8_t data, uint8_t lsb, uint8_t msb) SPI Write Wrapper. More...
uint8_t	**afeSpiReadWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t *readVal) SPI Read Wrapper. More...
uint8_t	**serdesWriteWrapper** (uint8_t afeId, uint16_t addr, uint16_t data, uint8_t lsb, uint8_t msb) SerDes Write Wrapper. More...
uint8_t	**serdesReadWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint16_t *readVal) SerDes Read Wrapper. More...
uint8_t	**serdesLaneWriteWrapper** (uint8_t afeId, uint16_t addr, uint8_t laneNo, uint16_t data, uint8_t lsb, uint8_t msb) SerDes Lane Write Wrapper. More...
uint8_t	**serdesLaneReadWrapper** (uint8_t afeId, uint16_t addr, uint32_t laneNo, uint8_t lsb, uint8_t msb, uint16_t *readVal) SerDes Lane Read Wrapper. More...
uint8_t	**afeSpiCheckWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t data, uint8_t *pbSame) AFE SPI Check Wrapper. More...
uint8_t	**afeSpiPollWrapper** (uint8_t afeId, uint16_t addr, uint8_t expectedData, uint8_t lsb, uint8_t msb) AFE SPI Poll Wrapper. More...
uint8_t	**afeSpiPollLogWrapper** (uint8_t afeId, uint16_t addr, uint8_t lsb, uint8_t msb, uint8_t expectedData) AFE SPI Poll Wrapper. More...
uint8_t	**requestPllSpiAccess** (uint8_t afeId, uint32_t regType) Requesting PLL Spi Access. More...
uint8_t	**readTopMem** (uint8_t afeId, uint32_t addr, uint64_t *readVal, uint32_t noBytes) Reads the MCU Memory. More...
uint8_t	**closeAllPages** (uint8_t afeId) Close All Pages. More...

Detailed Description

This file has Basic SPI functions.

Version 2.2:

1. Updated the log comment in serdesRawWrite function.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Changed the C macros for all the spi wrapper function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.
3. Added closeAllPages function.

Macro Definition Documentation

◆ AFE_REQ_SPI_ACCESS_MAX_COUNT

```
#define AFE_REQ_SPI_ACCESS_MAX_COUNT   100
```

◆ CFG_SPI_READ_POLL_MAX_COUNT

```
#define CFG_SPI_READ_POLL_MAX_COUNT   500
```

◆ MASK_BYTE

```
#define MASK_BYTE (   lsb,
                      msb
                   )      (uint8_t)(((1 << ((msb) - (lsb) + 1)) - 1) << lsb)
```

◆ MASK_SHORT

```
#define MASK_SHORT (   lsb,
                       msb
                    )      (uint16_t)(((1 << ((msb) - (lsb) + 1)) - 1) << lsb)
```

Typedef Documentation

◆ PllSpiRegType_e

```
typedef enum PLL_SPI_REG_TYPE PllSpiRegType_e
```

Enumeration Type Documentation

◆ PLL_SPI_REG_TYPE

enum **PLL_SPI_REG_TYPE**

Enumerator	
PLL_SPI_REG_OFF	
PLL_SPI_REG_A	
PLL_SPI_REG_B	
PLL_SPI_REG_TYPE_SIZE	

Function Documentation

◆ afeSpiCheckWrapper()

```
uint8_t afeSpiCheckWrapper ( uint8_t   afeId,
                             uint16_t  addr,
                             uint8_t   lsb,
                             uint8_t   msb,
                             uint8_t   data,
                             uint8_t * pbSame
                           )
```

AFE SPI Check Wrapper.

Reads and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb);

Parameters

afeId	AFE ID
addr	SPI address
data	Expected Value.
lsb	lsb of the field.
msb	msb of the field.
pbSame	Pointer return. Returns 0 if the check passes and if check fails.

Returns

Returns if the function execution passed or failed.

◆ afeSpiPollLogWrapper()

```
uint8_t afeSpiPollLogWrapper ( uint8_t  afeId,
                               uint16_t addr,
                               uint8_t  lsb,
                               uint8_t  msb,
                               uint8_t  expectedData
                             )
```

AFE SPI Poll Wrapper.

Polls and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb); Function definition reordered from afeSpiPollWrapper to suit the log format.

Parameters

afeId	AFE ID
addr	SPI address
expectedData	Expected Value.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed. It returns fail even when the read data didn't match the expected value.

◆ afeSpiPollWrapper()

```
uint8_t afeSpiPollWrapper ( uint8_t  afeId,
                            uint16_t addr,
                            uint8_t  expectedData,
                            uint8_t  lsb,
                            uint8_t  msb
                          )
```

AFE SPI Poll Wrapper.

Polls and checks if the value of the field is as expected. Check Pass condition is (readValue&mask)==(data&mask) where mask = (((1 << ((msb) - (lsb) + 1)) - 1) << lsb);

Parameters

afeId	AFE ID
addr	SPI address
expectedData	Expected Value.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed. It returns fail even when the read data didn't match the expected value.

◆ afeSpiReadWrapper()

```
uint8_t afeSpiReadWrapper ( uint8_t   afeId,
                            uint16_t  addr,
                            uint8_t   lsb,
                            uint8_t   msb,
                            uint8_t * readVal
                          )
```

SPI Read Wrapper.

Reads the value to the specified bits of the register and returns as a pointer.

Parameters

afeId	AFE ID
addr	SPI address
lsb	lsb of the field.
msb	msb of the field.
readVal	pointer of the read value.

Returns

Returns if the function execution passed or failed.

◆ afeSpiWriteWrapper()

```
uint8_t afeSpiWriteWrapper ( uint8_t   afeId,
                             uint16_t  addr,
                             uint8_t   data,
                             uint8_t   lsb,
                             uint8_t   msb
                           )
```

SPI Write Wrapper.

Writes the value to the specified bits of the register.

Parameters

afeId	AFE ID
addr	SPI address
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

◆ closeAllPages()

```
uint8_t closeAllPages ( uint8_t  afeId )
```

Close All Pages.

This function closes all the pages. Need to be called in case of a SPI/function to ensure no open page is present.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ readTopMem()

```
uint8_t readTopMem ( uint8_t     afeId,
                     uint32_t    addr,
                     uint64_t *  readVal,
                     uint32_t    noBytes
                   )
```

Reads the MCU Memory.

This reads the MCU memory and returns the value as a pointer.

Parameters

afeId	AFE ID
addr	Memory Address.
readVal	Value read returned as a pointer.
noBytes	Number of bytes to be read. Supported values: 0<noBytes<=8

Returns

Returns if the function execution passed or failed.

◆ requestPllSpiAccess()

```
uint8_t requestPllSpiAccess ( uint8_t   afeId,
                              uint32_t  regType
                            )
```

Requesting PLL Spi Access.

For access PLL registers, the access to the PLL page should be requested and we should proceed only after it is granted. After the access is complete, the SPI access should be. This function does these operations. This access is independent for SPIA and SPIB.

Parameters

afeId	AFE ID
regType	0-Relinquish SPI access 1- Request Access for SPIA 2- Request Access for SPIB

Returns

Returns if the function execution passed or failed. It returns fail even when the request has not been granted.

◆ serdesLaneReadWrapper()

```
uint8_t serdesLaneReadWrapper ( uint8_t    afeId,
                                uint16_t   addr,
                                uint32_t   laneNo,
                                uint8_t    lsb,
                                uint8_t    msb,
                                uint16_t * readVal
                              )
```

SerDes Lane Read Wrapper.

Reads the value to the specified bits of the SerDes lane register of the corresponding lane by adding the appropriate offset. Returns the value as pointer.

Parameters

afeId	AFE ID
addr	SerDes lane base address
laneNo	SerDes lane number. 0-7 is the supported range.
lsb	lsb of the field.
msb	msb of the field.
readVal	Pointer of the value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesLaneWriteWrapper()

```
uint8_t serdesLaneWriteWrapper ( uint8_t  afeId,
                                 uint16_t addr,
                                 uint8_t  laneNo,
                                 uint16_t data,
                                 uint8_t  lsb,
                                 uint8_t  msb
                               )
```

SerDes Lane Write Wrapper.

Writes the value to the specified bits of the SerDes lane register of the corresponding lane by adding the appropriate offset.

Parameters

afeId	AFE ID
addr	SerDes lane base address
laneNo	SerDes lane Number. Values supported are: 0-7.
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

◆ serdesRawRead()

```
uint8_t serdesRawRead ( uint8_t    afeId,
                        uint16_t   addr,
                        uint16_t * readVal
                      )
```

SerDes Read.

SerDes registers are 16-bit wide while SPI is 8-bit. This necessitates a translation between SPI and SerDes. This function reads SerDes registers and returns the read value as a pointer.

Parameters

afeId	AFE ID
addr	SerDes address
readVal	Pointer returning the read value

Returns

Returns if the function execution passed or failed.

◆ serdesRawWrite()

```
uint8_t serdesRawWrite ( uint8_t  afeId,
                         uint16_t addr,
                         uint16_t data
                       )
```

SerDes Write.

SerDes registers are 16-bit wide while SPI is 8-bit. This necessitates a translation between SPI and SerDes. This function writes SerDes registers.

Parameters

afeId	AFE ID
addr	SerDes address
data	Value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesReadWrapper()

```
uint8_t serdesReadWrapper ( uint8_t    afeId,
                            uint16_t   addr,
                            uint8_t    lsb,
                            uint8_t    msb,
                            uint16_t * readVal
                          )
```

SerDes Read Wrapper.

Reads the value to the specified bits of the SerDes register and returns as a pointer.

Parameters

afeId	AFE ID
addr	SerDes address
lsb	lsb of the field.
msb	msb of the field.
readVal	Pointer of the value to be written.

Returns

Returns if the function execution passed or failed.

◆ serdesWriteWrapper()

```
uint8_t serdesWriteWrapper ( uint8_t  afeId,
                             uint16_t addr,
                             uint16_t data,
                             uint8_t  lsb,
                             uint8_t  msb
                           )
```

SerDes Write Wrapper.

Writes the value to the specified bits of the SerDes register.

Parameters

afeId	AFE ID
addr	SerDes address
data	Value to be written.
lsb	lsb of the field.
msb	msb of the field.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

calibrations.c File Reference

This file has Factory calibration related functions.

Version 2.1:

More...

```
#include <stdint.h>
#include <math.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "afeParameters.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "controls.h"
#include "calibrations.h"
#include "hMacro.h"
```

Functions

uint8_t	**doRxDsaCalib** (uint8_t afeId, uint8_t rxChainForCalib, uint8_t fbChainForCalib, uint8_t useTxForCalib, uint8_t rxDsaBandCalibMode, uint8_t *readPacket, uint16_t *readPacketSize) Perform ADC DSA Calibration. More...
uint8_t	**doTxDsaCalib** (uint8_t afeId, uint8_t txChainForCalib, uint8_t txDsaCalibMode, uint8_t txDsaBandCalibMode, uint8_t *readPacket, uint16_t *readPacketSize) Perform DAC DSA Calibration. More...
uint8_t	**loadTxDsaPacket** (uint8_t afeId, uint8_t *array, uint8_t arraySize) Load the TX DSA Calibration Packet. More...
uint8_t	**loadRxDsaPacket** (uint8_t afeId, uint8_t *array, uint8_t arraySize) Load the ADC DSA Calibration Packet. More...

Detailed Description

This file has Factory calibration related functions.

Version 2.1:

1. Added this file only in version 2.1.
2. Added documentation and improved the parameter validity checks.
3. Modified the RX DSA calibration function to add placeholder function for channel inputs.
4. Added TX DSA calibration function.

Function Documentation

◆ doRxDsaCalib()

```
uint8_t doRxDsaCalib ( uint8_t    afeId,
                       uint8_t    rxChainForCalib,
                       uint8_t    fbChainForCalib,
                       uint8_t    useTxForCalib,
                       uint8_t    rxDsaBandCalibMode,
                       uint8_t *  readPacket,
                       uint16_t * readPacketSize
                     )
```

Perform ADC DSA Calibration.

This function Performs the RX DSA calibration. giveAfeAdcInput function in **baseFunc.c** file contents should be coded by the user as needed. However, in a single band case, if all the channels can be given input at the same time, this function needn't do any operation and all the channels should be given input before calling this function.

Parameters

afeId	AFE ID
rxChainForCalib	Bit Wise RX Channel Select. Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fbChainForCalib	Bit Wise FB Channel Select. Bit0 for FBAB Bit1 for FBCD
useTxForCalib	When Set to 1, TX TDD will be kept on so that TX can be used for the calibration. The data should be still sent from the ASIC/FPGA through JESD.
rxDsaBandCalibMode	Sets the RX DSA Band Calibration Mode. 0 -One Band at a time 1 - both bands together
readPacket	Pointer returns Array of the Read packet. This should be stored in the host memory and be loaded post initialization in normal mode of operation.
readPacketSize	Pointer returns the size of the array.

Returns

Returns if the function execution passed or failed.

◆ doTxDsaCalib()

```
uint8_t doTxDsaCalib ( uint8_t    afeId,
                       uint8_t    txChainForCalib,
                       uint8_t    txDsaCalibMode,
                       uint8_t    txDsaBandCalibMode,
                       uint8_t *  readPacket,
                       uint16_t * readPacketSize
                     )
```

Perform DAC DSA Calibration.

This function Performs the TX DSA calibration. connectAfeTxToFb function in **baseFunc.c** file contents should be coded by the user as needed. However, in a single band case, if all the channels can be given input at the same time, this function needn't do any operation and all the channels should be given input before calling this function.

Parameters

afeId	AFE ID
txChainForCalib	Bit Wise TX Channel Select. Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD
txDsaCalibMode	DSA Calibration Mode. 0 -Single Fb Mode FB AB ; 1 -Single Fb Mode FB CD ; 2- Dual Fb_Mode
txDsaBandCalibMode	Sets the TX DSA Band Calibration Mode. 0 -One Band at a time 1 - both bands together
readPacket	Pointer returns Array of the Read packet. This should be stored in the host memory and be loaded post initialization in normal mode of operation.
readPacketSize	Pointer returns the size of the array.

Returns

Returns if the function execution passed or failed.

◆ loadRxDsaPacket()

```
uint8_t loadRxDsaPacket ( uint8_t    afeId,
                          uint8_t *  array,
                          uint8_t    arraySize
                        )
```

Load the ADC DSA Calibration Packet.

This function loads the ADC DSA Calibration Packet

Parameters

afeId	AFE ID
array	Pointer of array of the packet which was stored in host after calibration.
arraySize	Value of the size of the array.

Returns

Returns if the function execution passed or failed.

◆ loadTxDsaPacket()

```
uint8_t loadTxDsaPacket ( uint8_t    afeId,
                          uint8_t *  array,
                          uint8_t    arraySize
                        )
```

Load the TX DSA Calibration Packet.

This function loads the TX DSA Calibration Packet

Parameters

afeId	AFE ID
array	Pointer of array of the packet which was stored in host after calibration.
arraySize	Value of the size of the array.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

controls.c File Reference

This file has generic control related functions.

Version 2.2:

More...

```
#include <stdint.h>
#include <math.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
```

```
#include "controls.h"
#include "hMacro.h"
#include "agc.h"
#include "jesd.h"
#include "pap.h"
```

Functions

uint8_t	**getChipVersion** (uint8_t afeId) Reads the Chip version of the AFE. More...
uint8_t	**checkSysref** (uint8_t afeId, uint8_t clearSysrefFlag, uint8_t *sysrefReceived) Check if the Sysref Reached. More...
uint8_t	**sendSysref** (uint8_t afeId, uint8_t spiSysref, uint8_t getSpiAccess) Give a new Sysref to the AFE. More...
uint8_t	**overrideTdd** (uint8_t afeId, uint8_t rx, uint8_t fb, uint8_t tx, uint8_t enableOverride) Override TDD Control Signals and set the SPI override value. More...
uint8_t	**overrideTddPins** (uint8_t afeId, uint8_t rx, uint8_t fb, uint8_t tx) Override TDD Control Signals. More...
uint8_t	**checkPllLockStatus** (uint8_t afeId, uint8_t *pllLockStatus) Checks the PLL Lock Status. More...
uint8_t	**clearPllStickyLockStatus** (uint8_t afeId) Clears the PLL Lock Sticky Status. More...
uint8_t	**readAlarmPinStatus** (uint8_t afeId, uint8_t alarmNo, uint8_t *status) Checks the Alarm Pin Status. More...
uint8_t	**clearSpiAlarms** (uint8_t afeId) Clears the SPI Alarm Status. More...
uint8_t	**readSpiAlarms** (uint8_t afeId, uint8_t *alarmStatus) Checks the SPI Alarm Status. More...
uint8_t	**readTxPower** (uint8_t afeId, uint8_t chNo, uint16_t windowLen, double *powerReadB0, double *powerReadB1, double *combinedRead) Read the TX power. More...
uint8_t	**getRxRmsPower** (uint8_t afeId, uint8_t chNo, double *avg_pwrdb) Read the RX power. More...
uint8_t	**clearAllAlarms** (uint8_t afeId) Clear all the alarms. More...
uint8_t	**overrideAlarmPin** (uint8_t afeId, uint8_t alarmNo, uint8_t overrideSel, uint8_t overrideVal) Override Alarm Pin output and set the SPI override value. More...
uint8_t	**overrideRelDetPin** (uint8_t afeId, uint8_t chNo, uint8_t overrideSel, uint8_t overrideVal) Override RX Reliability Pin output and set the SPI override value. More...
uint8_t	**overrideDigPkDetPin** (uint8_t afeId, uint8_t chNo, uint8_t pinNo, uint8_t overrideSel, uint8_t overrideVal) Override RX Peak Detector Pin output and set the SPI override value. More...
uint8_t	**checkDeviceHealth** (uint8_t afeId, uint16_t *allOk) Checks the Device Health. More...

Detailed Description

This file has generic control related functions.

Version 2.2:

1. Fixed the bug in function getChipVersion.
2. Updated description of checkPllLockStatus.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Removed redundant writes in functions.
3. Changed the C macros for all the spi wrapper function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.
4. Checking if the sysref reached added to the sendSysref function. The function returns fail if the sysref fails to return.
5. For all PLL Access, the selection between the SPIA and SPIB is changed to a system parameter to cut down the redundant need to pass it in all the functions, since in a typical use case, only one SPI (SPIA) is used for it.
6. Added checkDeviceHealth function to return a overall status of the device.

Function Documentation

◆ checkDeviceHealth()

```
uint8_t checkDeviceHealth ( uint8_t     afeId,
                            uint16_t *  allOk
                          )
```

Checks the Device Health.

This function Reads the complete device health and returns as a pointer.

Parameters

afeId AFE ID

allOk Pointer return of the device health status.

If there is no error, allOk will be 0.

If it is non-zero, below is the interpretation

Bit 0: PLL Not Okay

Bit 1: DAC JESD Not Okay

Bit 2: ADC JESD Not Okay

Bit 3: SPI Not Okay

Bit 4: MCU Not Okay

Bit 5: PAP Triggered

Returns

Returns if the function execution passed or failed.

◆ checkPllLockStatus()

```
uint8_t checkPllLockStatus ( uint8_t    afeId,
                             uint8_t *  pllLockStatus
                           )
```

Checks the PLL Lock Status.

This function checks the PLL Lock Status and returns it as a pointer.

Parameters

afeId	AFE ID
pllLockStatus	Pointer Return of the lock statud of the PLL. 3 is ideal good state. 0: LOCK is low and LOCK_LOST is high. PLL is currently not locked but locked some time in the past since the status clear bit was last toggled. 1: LOCK is high and LOCK_LOST is high. PLL is currently locked but lost lock since the status clear bit was last toggled. (since clearPllStickyLockStatus was called) 2: LOCK is low and LOCK_LOST is low. PLL is currently not locked and never locked since the status clear bit was last toggled. 3: LOCK is high and LOCK_LOST is low. PLL is currently locked and didn't lose lock since the status clear bit was last toggled. (since clearPllStickyLockStatus was called).

Returns

Returns if the function execution passed or failed.

◆ checkSysref()

```
uint8_t checkSysref ( uint8_t    afeId,
                      uint8_t    clearSysrefFlag,
                      uint8_t *  sysrefReceived
                    )
```

Check if the Sysref Reached.

This function Checks if the Sysref is detected by the AFE.

Parameters

afeId	AFE ID
clearSysrefFlag	Setting this to 1 clear the Sysref flag before reading.
sysrefReceived	Pointer return. Value of 1 means Sysref reached. 0 means Sysref reached. 0 means it didn't reach.

Returns

Returns if the function execution passed or failed.

◆ clearAllAlarms()

uint8_t clearAllAlarms (uint8_t afeId)

Clear all the alarms.

Clears all the AFE alarms

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearPllStickyLockStatus()

uint8_t clearPllStickyLockStatus (uint8_t afeId)

Clears the PLL Lock Sticky Status.

This function clears the PLL Lock sticky Status.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearSpiAlarms()

uint8_t clearSpiAlarms (uint8_t afeId)

Clears the SPI Alarm Status.

This function clears the SPI Alarm Sticky Status. This is important when multiple SPIs are used and not critical when single SPI is being used.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ getChipVersion()

uint8_t getChipVersion (uint8_t afeId)

Reads the Chip version of the AFE.

This function Reads the Chip version, logs it and also updates the same in the System Params (systemParams[afeId].chipVersion).

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ getRxRmsPower()

```
uint8_t getRxRmsPower ( uint8_t   afeId,
                        uint8_t   chNo,
                        double *  avg_pwrdb
                      )
```

Read the RX power.

This function reads the RX Power.

Note that this detector is near the ADC-DDC interface and needs the RX TDD to be ON.

For reading FB power needed in ADC shared case, it should be operated in RX Mode and correponding RX channel should be read.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
avg_pwrdb	Pointer Return of RX Power Read

Returns

Returns if the function execution passed or failed.

◆ overrideAlarmPin()

```
uint8_t overrideAlarmPin ( uint8_t  afeId,
                           uint8_t  alarmNo,
                           uint8_t  overrideSel,
                           uint8_t  overrideVal
                         )
```

Override Alarm Pin output and set the SPI override value.

This function overrides Alarm Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
alarmNo	Select the Alarm number, Alarm Pin Number 0/1.
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideDigPkDetPin()

```
uint8_t overrideDigPkDetPin ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  pinNo,
                              uint8_t  overrideSel,
                              uint8_t  overrideVal
                            )
```

Override RX Peak Detector Pin output and set the SPI override value.

This function RX Peak Detector Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
chNo	Select the RX channel number. 0 for RXA 1 for RXB 2 for RXC 3 for RXD
pinNo	Pin Number to be overridden. Supported values are 0-3.
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideRelDetPin()

```
uint8_t overrideRelDetPin ( uint8_t  afeId,
                            uint8_t  chNo,
                            uint8_t  overrideSel,
                            uint8_t  overrideVal
                          )
```

Override RX Reliability Pin output and set the SPI override value.

This function overrides the RX Reliability Pin output and sets it to SPI override value

Parameters

afeId	AFE ID
chNo	Select the RX channel number. 0 for RXA 1 for RXB 2 for RXC 3 for RXD
overrideSel	0-Don't override. 1-Override the pin output.
overrideVal	When overrideSel is 1, this is the value sent out onto pin (0/1).

Returns

Returns if the function execution passed or failed.

◆ overrideTdd()

```
uint8_t overrideTdd ( uint8_t  afeId,
                      uint8_t  rx,
                      uint8_t  fb,
                      uint8_t  tx,
                      uint8_t  enableOverride
                    )
```

Override TDD Control Signals and set the SPI override value.

This function overrides SPI TDD Control Signals and set the SPI override value

Parameters

afeId	AFE ID
rx	Override Value of the RX chain. This is Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD
fb	Override Value of the FB chain. This is Bit wise channel select Bit0 for FBAB Bit1 for FBCD
tx	Override Value of the TX chain. This is Bit wise channel select Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD

enableOverride	Enables the Override.
	if enableOverride=0, it disables the TDD override
	if enableOverride=1, it enables the TDD override && also sets the TDD values
	if enableOverride=2, it only sets the TDD values

Returns

Returns if the function execution passed or failed.

◆ overrideTddPins()

```
uint8_t overrideTddPins ( uint8_t  afeId,
                          uint8_t  rx,
                          uint8_t  fb,
                          uint8_t  tx
                        )
```

Override TDD Control Signals.

This function Set the override values for each of RX,FB,TX TDD pins.

Parameters

afeId	AFE ID
rx	Override enable Value of the RX chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.
fb	Override enable Value of the FB chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.
tx	Override enable Value of the TX chain. 1 sets the pin value in override state. 0 removes the override and gives control to pins.

Returns

Returns if the function execution passed or failed.

◆ readAlarmPinStatus()

```
uint8_t readAlarmPinStatus ( uint8_t    afeId,
                             uint8_t    alarmNo,
                             uint8_t *  status
                           )
```

Checks the Alarm Pin Status.

This function reads the Alarm Pin Status and returns it as a pointer.

Parameters

afeId	AFE ID
alarmNo	Choose the Alarm Pin Number (0/1)
status	Pointer return Status of the alarm pin. 0 means there is no alarm and 1 means there is alarm.

Returns

Returns if the function execution passed or failed.

◆ readSpiAlarms()

```
uint8_t readSpiAlarms ( uint8_t   afeId,
                        uint8_t * alarmStatus
                      )
```

Checks the SPI Alarm Status.

This function reads the Alarm Status and returns it as a pointer. It also prints the error description.

Parameters

afeId	AFE ID
alarmStatus	Pointer return status of the SPI alarm. 0 means there are no alarms and 1 means there is a alarm.

Returns

Returns if the function execution passed or failed.

◆ readTxPower()

```
uint8_t readTxPower ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint16_t  windowLen,
                      double *  powerReadB0,
                      double *  powerReadB1,
                      double *  combinedRead
                    )
```

Read the TX power.

This function reads the TX Power.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
windowLen	Determines the window length for number of samples. 2^(windowLen+5) samples at the interface rate will be used for power measurement. Range of this is 0-0xfff
powerReadB0	Pointer Return of Band 0 Power Read
powerReadB1	Pointer Return of Band 1 Power Read
combinedRead	Pointer Return of Power Read after the combiner

Returns

Returns if the function execution passed or failed.

◆ sendSysref()

```
uint8_t sendSysref ( uint8_t  afeId,
                     uint8_t  spiSysref,
                     uint8_t  getSpiAccess
                   )
```

Give a new Sysref to the AFE.

This function is used to send a new sysref to the AFE. This enables the latch and performs the required operations for AFE to accept a new Sysref.
Note the following:

1. Contents of the giveSingleSysrefPulse function should be replaced by host function to give Pin Sysref to AFE. This is used only in case of a single shot sysref.
2. For Continuous Syref mode, external Pin Sysref should be enabled before this function is called. Note that even in this case, only one pulse edge will be captured by the AFE. In this mode, giveSingleSysrefPulse needn't do any operation.
3. systemParams[afeId].spiInUseForPllAccess should be set before the function call to the appropriate value for selecting SPIA/SPIB. In Normal use-case SPIA is used and hence can be left at the default.
4. The selection between the single shot and continuous sysref mode should be done in Latte during generation of the configuration log.

Parameters

afeId	AFE ID
spiSysref	If this is set to 0, external pin based Sysref is used. If this is set to 1, then the internal override of the Sysref pin will be used. Note that in this case, deterministic latency will not satisfied.
getSpiAccess	Setting this to 1 will take PLL SPI access. This should always be set 1.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

dsaAndNco.c File Reference

This file has DSA and NCO related functions.

Version 2.2:

More...

```
#include <stdint.h>
#include <math.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
#include "hMacro.h"
#include "dsaAndNco.h"
```

Functions

uint8_t	**setTxDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the TX Analog DSA. More...
uint8_t	**setFbDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the FB Analog DSA. More...
uint8_t	**setRxDsa** (uint8_t afeId, uint8_t chNo, uint8_t dsaSetting) Set the RX Analog DSA. More...

uint8_t	**setRxDigGain** (uint8_t afeId, uint8_t chNo, uint8_t bandNo, uint8_t dsaSetting) Set the RX Digital DSA. More...
uint8_t	**setRxDsaMode** (uint8_t afeId, uint8_t topNo, uint8_t mode) Set the RX DSA Mode. More...
uint8_t	**setPinRxDsaSettings** (uint8_t afeId, uint8_t chNo, uint8_t dsaInit, uint8_t dsaStep, uint8_t maxDelay) Configure Settings related to the 4-pin based DSA control mode. More...
uint8_t	**setTxDigGain** (uint8_t afeId, uint8_t chNo, uint8_t bandNo, int16_t dig_gain) Set the TX Digital DSA. More...
uint8_t	**txDsaIdxGainSwap** (uint8_t afeId, uint8_t chNo, uint8_t anaAttn0, uint8_t anaAttn1, int8_t digB0Gain0, int8_t digB0Gain1, int8_t digB1Gain0, int8_t digB1Gain1) Set the TX DSA Gain Swap Attenuation. More...
uint8_t	**updateTxGainParam** (uint8_t afeId, uint8_t mode, uint8_t transitTime, uint8_t maxAnaDsa) Set the TX DSA Update Mode. More...
uint8_t	**updateTxGain** (uint8_t afeId, uint8_t txChainSel, uint8_t gainValidity, uint16_t tx0B0Dsa, uint16_t tx0B1Dsa, uint16_t tx1B0Dsa, uint16_t tx1B1Dsa) Set the TX DSA. More...
uint8_t	**updateTxNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t nco) Set the TX NCO for single band. More...
uint8_t	**updateTxNcoDb** (uint8_t afeId, uint8_t chNo, uint8_t nco, uint32_t band0Nco0, uint32_t band1Nco0, uint32_t band0Nco1, uint32_t band1Nco1) Set the TX NCO for Dual band. More...
uint8_t	**rxNCOSel** (uint8_t afeId, uint8_t chNo, uint8_t BandId, uint8_t ovr, uint8_t NCOId) Set the RX NCO Select. More...
uint8_t	**fbNCOSel** (uint8_t afeId, uint8_t topno, uint8_t ovr, uint8_t NCOId) Set the FB NCO Select. More...
uint8_t	**updateRxNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t band, uint8_t nco) Set the RX NCO. More...
uint8_t	**updateFbNco** (uint8_t afeId, uint8_t chNo, uint32_t mixer, uint8_t nco) Set the FB NCO. More...
uint8_t	**readRxNco** (uint8_t afeId, uint8_t chNo, uint8_t band, uint8_t nco, double *ncoFreq) Read the RX NCO. More...
uint8_t	**readFbNco** (uint8_t afeId, uint8_t chNo, uint8_t nco, double *ncoFreq) Read the FB NCO. More...
uint8_t	**readTxNco** (uint8_t afeId, uint8_t chNo, uint8_t band, uint8_t nco, int64_t *val) Read the TX NCO. More...
uint8_t	**setFbDsaPerTx** (uint8_t afeId, uint8_t pinNo, uint8_t dsaSetting) Set the FB Analog DSA for pin select mode. More...
uint8_t	**fbDsaPerTxEn** (uint8_t afeId, uint8_t en) Enable the pin select based Mode for FB DSA. More...

Detailed Description

This file has DSA and NCO related functions.

Version 2.2:

This file has DSA and NCO related functions.

Version 2.3:

Moved these functions from **baseFunc.c**.

1. Updated setTxDigGain and txDsaIdxGainSwap along with the description and parameter validity.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Removed redundant functions related to older device version.
3. Fixed data types of parameters if function: updateTxGain
4. Removed redundant writes in functions.
5. Changed the function input definition of updateTxNco, updateRxNco and updateFbNco in FCW mode from KHz to FCW word. This is done to give finer control of frequency preventing rounding errors which is expected in FCW mode.
6. Changed the C macros for all the spi wrapper and executeMacro function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.
7. Fixed bugs in readTxNco.

Function Documentation

◆ fbDsaPerTxEn()

```
uint8_t fbDsaPerTxEn ( uint8_t  afeId,
                       uint8_t  en
                     )
```

Enable the pin select based Mode for FB DSA.

AFE has a feature to select the FB DSA value from a set of pre-programmed values using pins. This function sets the FB Analog DSA index.

Parameters

afeId	AFE ID
en	en as 1 will enable the feature to set FB DSA per TX based on the GPIO.

Returns

Returns if the function execution passed or failed.

◆ fbNCOSel()

```
uint8_t fbNCOSel ( uint8_t  afeId,
                   uint8_t  topno,
                   uint8_t  ovr,
                   uint8_t  NCOId
                 )
```

Set the FB NCO Select.

This function sets the override to the FB NCO select. This is useful only when more than 1 NCO is used.

Parameters

afeId	AFE ID
topno	Select the FB Channel 0 for FBAB 1 for FBCD
ovr	1 will override the pin. 0 will give control to the pin.
NCOId	NCO number which is to be selected. Supported range is 0 to numFbNco set in the initial configuration.

Returns

Returns if the function execution passed or failed.

◆ readFbNco()

```
uint8_t readFbNco ( uint8_t   afeId,
                    uint8_t   chNo,
                    uint8_t   nco,
                    double *  ncoFreq
                  )
```

Read the FB NCO.

This function reads the FB NCO.

Parameters

afeId	AFE ID
chNo	Select the FB Channel 0 for FBAB 1 for FBCD
nco	NCO number. 0-NCO0, 1-NCO1.
ncoFreq	Pointer Return. Returns the value of the NCO frequency read in MHz.

Returns

Returns if the function execution passed or failed.

◆ readRxNco()

```
uint8_t readRxNco ( uint8_t   afeId,
                    uint8_t   chNo,
                    uint8_t   band,
                    uint8_t   nco,
                    double *  ncoFreq
                  )
```

Read the RX NCO.

This function reads the RX NCO and returns it as a pointer. systemParams[afeId].ncoFreqMode should be matched with the value set in the initial configuration.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.
ncoFreq	Pointer Return. Returns the value of the NCO frequency read in MHz.

Returns

Returns if the function execution passed or failed.

◆ readTxNco()

```
uint8_t readTxNco ( uint8_t   afeId,
                    uint8_t   chNo,
                    uint8_t   band,
                    uint8_t   nco,
                    int64_t * val
                  )
```

Read the TX NCO.

This function reads the RX NCO and returns it as a pointer. systemParams[afeId].ncoFreqMode should be matched with the value set in the initial configuration.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.
val	Pointer Return. Returns the value of the NCO frequency read in MHz.

Returns

Returns if the function execution passed or failed.

◆ rxNCOSel()

```
uint8_t rxNCOSel ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  BandId,
                   uint8_t  ovr,
                   uint8_t  NCOId
                 )
```

Set the RX NCO Select.

This function sets the override to the RX NCO select. This is useful only when more than 1 NCO is used.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
BandId	NCO number. 0-NCO0, 1-NCO1.
ovr	1 will override the pin. 0 will give control to the pin.
NCOId	NCO number which is to be selected. Supported range is 0 to numRxNco set in the initial configuration.

Returns

Returns if the function execution passed or failed.

◆ setFbDsa()

```
uint8_t setFbDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the FB Analog DSA.

Sets the FB Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the FB Channel 0 for FBAB 1 for FBCD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*0.5dB

Returns

Returns if the function execution passed or failed.

◆ setFbDsaPerTx()

```
uint8_t setFbDsaPerTx ( uint8_t  afeId,
                        uint8_t  pinNo,
                        uint8_t  dsaSetting
                      )
```

Set the FB Analog DSA for pin select mode.

AFE has a feature to select the FB DSA value from a set of pre-programmed values using pins. This function sets the FB Analog DSA index. systemParams [afeId].txToFbMode should be set as needed in the initialization.

Parameters

afeId AFE ID

pinNo Select the pin value for which to program the DSA. The range of this is 0-3.

dsaSetting Analog dsaSetting is FB DSA for the corresponding pin value. dsaSetting*0.5 is the attenuation in dB applied when the pin value is pinNo.

Returns

Returns if the function execution passed or failed.

◆ setPinRxDsaSettings()

```
uint8_t setPinRxDsaSettings ( uint8_t  afeId,
                              uint8_t  chNo,
                              uint8_t  dsaInit,
                              uint8_t  dsaStep,
                              uint8_t  maxDelay
                            )
```

Configure Settings related to the 4-pin based DSA control mode.

Configure Settings related to the 4-pin based DSA control mode. Effective DSA attenuation is ((pin_value * dsaStep) +dsaInit)*0.5dB.

Parameters

afeId AFE ID

chNo Select the RX Channel
0 for RXA
1 for RXB
2 for RXC
3 for RXD

dsaInit Offset of the DSA.

dsaStep DSA Step Value.

maxDelay This is the delay after the change of pin change to latch the values. This is to account for the latency variation between pins.
This should be the maximum latency variation between the earliest pin and the last pin.
This is the common control for 2RX. The unit is in cycles of FadcRx/8 clock. Supported values: 0<=maxDelay<=255.

Returns

Returns if the function execution passed or failed.

◆ setRxDigGain()

```
uint8_t setRxDigGain ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  bandNo,
                       uint8_t  dsaSetting
                     )
```

Set the RX Digital DSA.

Sets the RX Digital DSA.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
bandNo	Select the RX Band. 0-Band0, 1-Band1
dsaSetting	(dsaSetting*0.5-3)dB is the applied DSA gain (if positive) and attenuation (if negative). Range for dsaSetting is 0 to 47.

Returns

Returns if the function execution passed or failed.

◆ setRxDsa()

```
uint8_t setRxDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the RX Analog DSA.

Sets the RX Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*0.5dB

Returns

Returns if the function execution passed or failed.

◆ setRxDsaMode()

```
uint8_t setRxDsaMode ( uint8_t  afeId,
                       uint8_t  topNo,
                       uint8_t  mode
                     )
```

Set the RX DSA Mode.

Sets the RX DSA Control Mode.

Parameters

afeId	AFE ID
topNo	Select the RX Channel 0 for RXAB 1 for RXCD
mode	DSA Control Mode Setting. 1-8-Pin Based DSA Control 2-Internal AGC 3-SPI AGC 4-4-Pin Based DSA Control

Returns

Returns if the function execution passed or failed.

◆ setTxDigGain()

```
uint8_t setTxDigGain ( uint8_t  afeId,
                       uint8_t  chNo,
                       uint8_t  bandNo,
                       int16_t  dig_gain
                     )
```

Set the TX Digital DSA.

Sets the TX Digital DSA.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
bandNo	Select the TX Band. 0-Band0, 1-Band1
dig_gain	dig_gain is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain. (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed.

Returns

Returns if the function execution passed or failed.

◆ setTxDsa()

```
uint8_t setTxDsa ( uint8_t  afeId,
                   uint8_t  chNo,
                   uint8_t  dsaSetting
                 )
```

Set the TX Analog DSA.

Sets the TX Analog DSA.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
dsaSetting	Analog DSA Index. Attenuation applied is dsaSetting*1dB

Returns

Returns if the function execution passed or failed.

◆ txDsaIdxGainSwap()

```
uint8_t txDsaIdxGainSwap ( uint8_t  afeId,
                           uint8_t  chNo,
                           uint8_t  anaAttn0,
                           uint8_t  anaAttn1,
                           int8_t   digB0Gain0,
                           int8_t   digB0Gain1,
                           int8_t   digB1Gain0,
                           int8_t   digB1Gain1
                         )
```

Set the TX DSA Gain Swap Attenuation.

Set the TX DSA Gain Swap Attenuation. There are 2 Gain Swap settings possible which can be chosen using the pin.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
anaAttn0	Analog Attenuation for Swap Attenuation 0, from 0 to 29. (1dB steps)
anaAttn1	Analog Attenuation for Swap Attenuation 1, from 0 to 29. (1dB steps)
digB0Gain0	Digital Attenuation*8 for Swap Attenuation 0 for band 0 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB0Gain1	Digital Attenuation*8 for Swap Attenuation 1 for band 0 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB1Gain0	Digital Attenuation*8 for Swap Attenuation 0 for band 1 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed
digB1Gain1	Digital Attenuation*8 for Swap Attenuation 1 for band 1 Is integer value ranging from +24 to -167, that maps to +3dBfs to -20.875dBfs gain (negative values refers to attenuation) The needed attenuation*8 is the dig_gain value to be passed

Returns

Returns if the function execution passed or failed.

◆ updateFbNco()

```
uint8_t updateFbNco ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint32_t  mixer,
                      uint8_t   nco
                    )
```

Set the FB NCO.

This function updates the FB NCO.

Parameters

afeId AFE ID

chNo Select the FB Channel

0 for FBAB

1 for FBCD

mixer Mixer frequency.

Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script.

In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/FadcRx).

nco NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateRxNco()

```
uint8_t updateRxNco ( uint8_t  afeId,
                      uint8_t  chNo,
                      uint32_t mixer,
                      uint8_t  band,
                      uint8_t  nco
                    )
```

Set the RX NCO.

This function updates the RX NCO.

Parameters

afeId	AFE ID
chNo	Select the RX Channel 0 for RXA 1 for RXB 2 for RXC 3 for RXD
mixer	Mixer frequency. Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/FadcRx).
band	Band number. 0-band0, 1-band1.
nco	NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateTxGain()

```
uint8_t updateTxGain ( uint8_t   afeId,
                       uint8_t   txChainSel,
                       uint8_t   gainValidity,
                       uint16_t  tx0B0Dsa,
                       uint16_t  tx0B1Dsa,
                       uint16_t  tx1B0Dsa,
                       uint16_t  tx1B1Dsa
                     )
```

Set the TX DSA.

This function sets the TX DSA (analog+digital) through Macro.
When the value is less or equal to than the maxAnaDsa setting in updateTxGainParam function, the integer part of the value will be applied to analog and fractional part will be applied to digital.
When the value is more than the maxAnaDsa setting in updateTxGainParam function, maxAnaDsa will be applied to the analog and rest will be applied in digital.
For single band case, set same value as band0 to band1 and apply gain validity accordingly.

Parameters

afeId AFE ID

txChainSel Selects if the DSA attenuation needs to be applied to AB or CD channels.
0-AB
1-CD

gainValidity Selects where all to set the DSA. This is a bit wise field.
bit 0- TXA/C Band0
bit 1- TXA/C Band1
bit 2- TXB/D Band0
bit 3- TXB/D Band1

tx0B0Dsa TXA/C Band 0 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.

tx0B1Dsa TXA/C Band 1 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.

tx1B0Dsa TXB/D Band 0 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.

tx1B1Dsa TXB/D Band 1 DSA setting *8. For getting attenuation of 2.25dB, this value should be 18. Supported Range is 0-320.

Returns

Returns if the function execution passed or failed.

◆ updateTxGainParam()

```
uint8_t updateTxGainParam ( uint8_t  afeId,
                            uint8_t  mode,
                            uint8_t  transitTime,
                            uint8_t  maxAnaDsa
                          )
```

Set the TX DSA Update Mode.

This function sets the Params of applying TX DSA through Macro.

Parameters

afeId	AFE ID
mode	Mode of TX DSA Update 0-oneshot (Immediately update) 1-smoothening (Enable smooth transition of DSA) 2-TDD mode (Set DSA on TX TDD off state)
transitTime	This value/8 us is the time taken for each step in smoothening mode.
maxAnaDsa	This is the maximum analog DSA (in dB) beyond which the digital gain/attenuation will be applied. Maximum value of this is 29

Returns

Returns if the function execution passed or failed.

◆ updateTxNco()

```
uint8_t updateTxNco ( uint8_t   afeId,
                      uint8_t   chNo,
                      uint32_t  mixer,
                      uint8_t   nco
                    )
```

Set the TX NCO for single band.

This function updates the TX NCO and should be used only single band of operation.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
mixer	Mixer frequency. Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
nco	NCO number. 0-NCO0, 1-NCO1.

Returns

Returns if the function execution passed or failed.

◆ updateTxNcoDb()

```
uint8_t updateTxNcoDb ( uint8_t   afeId,
                        uint8_t   chNo,
                        uint8_t   nco,
                        uint32_t  band0Nco0,
                        uint32_t  band1Nco0,
                        uint32_t  band0Nco1,
                        uint32_t  band1Nco1
                      )
```

Set the TX NCO for Dual band.

This function updates the TX NCO and should be used only single band of operation.
For all the mixer frequency values, should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script.
In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
In case second NCO is not used, set the band1 parameters to same value as band0.

Parameters

afeId	AFE ID
chNo	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
nco	NCO number. 0-NCO0, 1-NCO1.
band0Nco0	Band0, NCO0 Mixer frequency.
band1Nco0	Band1, NCO0 Mixer frequency.
band0Nco1	Band0, NCO1 Mixer frequency.
band1Nco1	Band1, NCO1 Mixer frequency.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

hMacro.c File Reference

This file has Macros related functions.

Version 2.1:

More...

```
#include <stdint.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "hMacro.h"
```

Functions

uint8_t	**writeOperandList** (uint8_t afeId, uint8_t *operandList, uint8_t numOfOperands) Write the Macro Operands. More...
uint8_t	**splitToByte** (uint64_t val, uint8_t numBytes, uint8_t *splitByteList) Converts a value into byte wise array. More...
uint8_t	**readResultRegSpi** (uint8_t afeId, uint8_t regNum, uint32_t *result) Read Macro Result Register. More...
uint8_t	**waitForMacroReady** (uint8_t afeId) Poll for Macro Ready. More...
uint8_t	**waitForMacroDone** (uint8_t afeId) Poll for Macro Done. More...
uint8_t	**waitForMacroAck** (uint8_t afeId) Poll for Macro Acknowledgement. More...
uint8_t	**checkForMacroError** (uint8_t afeId, uint8_t *macroErrorStatus) Checks if there is a Macro Error. More...
uint8_t	**triggerMacro** (uint8_t afeId, uint8_t opcode) Writes Opcode and triggers the Macro. More...
uint8_t	**executeMacro** (uint8_t afeId, uint8_t *byteList, uint8_t numOfOperands, uint8_t opcode) Execute a Macro. More...
uint8_t	**enableMemAccess** (uint8_t afeId, uint8_t en) Enables MCU Memory Access for SPI. More...
uint8_t	**doSystemTuneSelective** (uint8_t afeId, uint8_t rxChList, uint8_t fbChList, uint8_t txChList, uint8_t sectionEnable) Reconfigures the selected Chains. More...
uint8_t	**updateSystemTxChannelFreqConfig** (uint8_t afeId, uint8_t txChList, uint8_t listNCO, uint32_t txNCO, uint8_t immUpdt, uint8_t reload) Reconfigures the TX NCO info to the MCU. More...
uint8_t	**checkMcuHealth** (uint8_t afeId, uint8_t *healthOk) Checks for MCU Health. More...
uint8_t	**txCalibSiggen** (uint8_t afeId, uint8_t chNo, uint8_t configOption, uint32_t freq0, uint8_t freq0Amp) TX Tone Generator. More...

Detailed Description

This file has Macros related functions.

Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Deleted redundant function: doPrepareTune
3. Added TX Tone Gen function
4. Changed the C macros for all the spi wrapper and executeMacro function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.

Function Documentation

◆ checkForMacroError()

```
uint8_t checkForMacroError ( uint8_t    afeId,
                             uint8_t *  macroErrorStatus
                           )
```

Checks if there is a Macro Error.

Checks if there is a Macro Error and returns the error status as pointer.

Parameters

afeId	AFE ID
macroErrorStatus	Macro Error Status return as pointer.

Returns

Returns if the function execution passed or failed.

◆ checkMcuHealth()

```
uint8_t checkMcuHealth ( uint8_t    afeId,
                         uint8_t *  healthOk
                       )
```

Checks for MCU Health.

Checks for MCU Health and returns the status as a pointer.

Parameters

afeId	AFE ID
healthOk	Return Pointer of the status of the MCU. This value is 1 if the MCU is working properly and 0 if MCU is stuck.

Returns

Returns if the function execution passed or failed.

◆ doSystemTuneSelective()

```
uint8_t doSystemTuneSelective ( uint8_t  afeId,
                                uint8_t  rxChList,
                                uint8_t  fbChList,
                                uint8_t  txChList,
                                uint8_t  sectionEnable
                              )
```

Reconfigures the selected Chains.

Reconfigures the selected Chains. This function is called in updateTxNco function and is not recommended to be called independently.

Returns

Returns if the function execution passed or failed.

◆ enableMemAccess()

```
uint8_t enableMemAccess ( uint8_t  afeId,
                          uint8_t  en
                        )
```

Enables MCU Memory Access for SPI.

Enables MCU Memory Access for SPI. Note that this should be relinquished after the access is complete.

Parameters

afeId	AFE ID
en	1 enable MCU memory access for SPI. 0 disable MCU memory access for SPI

Returns

Returns if the function execution passed or failed.

◆ executeMacro()

```
uint8_t executeMacro ( uint8_t    afeId,
                       uint8_t *  byteList,
                       uint8_t    numOfOperands,
                       uint8_t    opcode
                     )
```

Execute a Macro.

Executes the Macro by calling other sub functions.

Parameters

afeId	AFE ID
byteList	Byte-wise array of operands to be written.
numOfOperands	Size of operandList.
opcode	Opcode of the Macro.

Returns

Returns if the function execution passed or failed.

◆ readResultRegSpi()

```
uint8_t readResultRegSpi ( uint8_t    afeId,
                           uint8_t    regNum,
                           uint32_t * result
                         )
```

Read Macro Result Register.

Read Macro Result Register

Parameters

afeId	AFE ID
regNum	Result register number.
result	Returns result register as a pointer.

Returns

Returns if the function execution passed or failed.

◆ splitToByte()

```
uint8_t splitToByte ( uint64_t  val,
                      uint8_t   numBytes,
                      uint8_t * splitByteList
                    )
```

Converts a value into byte wise array.

Converts a value into byte wise array.

Parameters

val	Value to be converted
numBytes	Number of Bytes to convert it to.
splitByteList	Pointer return of the resultant array.

Returns

Returns if the function execution passed or failed.

◆ triggerMacro()

```
uint8_t triggerMacro ( uint8_t  afeId,
                       uint8_t  opcode
                     )
```

Writes Opcode and triggers the Macro.

Writes Opcode and triggers the Macro.

Parameters

afeId	AFE ID
opcode	Opcode of the Macro.

Returns

Returns if the function execution passed or failed.

◆ txCalibSiggen()

```
uint8_t txCalibSiggen ( uint8_t   afeId,
                        uint8_t   chNo,
                        uint8_t   configOption,
                        uint32_t  freq0,
                        uint8_t   freq0Amp
                      )
```

TX Tone Generator.

This function sends a single tone to the TX.

Parameters

afeId	AFE ID
chNo	TX Channel Select. 0 for TXA 1 for TXB 2 for TXC 3 for TXD
configOption	Tone Generation Command 0 → RESERVED 1 → The current mixer configuration will be saved and the mixers will be configured to give the new tone frequency. 2 → The mixers will be configured to give the new tone but the saved configuration will not be modified. 3 → RESERVED 4 → Restore saved configuration. This can be called to restore the last saved mixer configuration.
freq0	RF tone Frequency Should pass value in KHz in 1KHz ncoFreqMode and the frequency word value in FCW mode. The Mode is determined by the ncoFreqMode set in Latte while generating the bringup script. In FCW mode, the value can be calculate using the equation: mixer = (uint32_t) (2^32*mixerFrequency/Fdac).
freq0Amp	Tone Backoff in dB

Returns

Returns if the function execution passed or failed.

◆ updateSystemTxChannelFreqConfig()

```
uint8_t updateSystemTxChannelFreqConfig ( uint8_t   afeId,
                                          uint8_t   txChList,
                                          uint8_t   listNCO,
                                          uint32_t  txNCO,
                                          uint8_t   immUpdt,
                                          uint8_t   reload
                                        )
```

Reconfigures the TX NCO info to the MCU.

Reconfigures the TX NCO info to the MCU. This function is called in updateTxNco function and is not recommended to be called independently.

Returns

Returns if the function execution passed or failed.

◆ waitForMacroAck()

```
uint8_t waitForMacroAck ( uint8_t  afeId )
```

Poll for Macro Acknowledgement.

Polls for Macro Acknowledgement

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_ACK doesn't become 1.

◆ waitForMacroDone()

```
uint8_t waitForMacroDone ( uint8_t  afeId )
```

Poll for Macro Done.

Polls for Macro Done

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_Done doesn't become 1.

◆ waitForMacroReady()

```
uint8_t waitForMacroReady ( uint8_t  afeId )
```

Poll for Macro Ready.

Polls for Macro Ready

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed. It returns as failed even if the Macro_Ready doesn't become 1.

◆ writeOperandList()

```
uint8_t writeOperandList ( uint8_t    afeId,
                           uint8_t *  operandList,
                           uint8_t    numOfOperands
                         )
```

Write the Macro Operands.

Write the Macro Operands.

Parameters

afeId	AFE ID
operandList	Byte-wise array of operands to be written.
numOfOperands	Size of operandList.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

init.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
#include "hMacro.h"
```

Functions

int8_t	**configAfeFromFileFormat0** (uint8_t afeId, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail) Bringup function configuration function from log file for format 0 of Latte log. More...
int8_t	**configAfeFromFileFormat5** (uint8_t afeId, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail) Bringup function configuration function from log file for format 5 of Latte log. More...

int8_t **configAfeFromFile** (uint8_t afeId, uint8_t logFormat, char *file, uint8_t breakAtPollFail, uint8_t breakAtReadCheckFail)

Common Bringup function configuration function. More...

Function Documentation

◆ configAfeFromFile()

```
int8_t configAfeFromFile ( uint8_t  afeId,
                           uint8_t  logFormat,
                           char *   file,
                           uint8_t  breakAtPollFail,
                           uint8_t  breakAtReadCheckFail
                         )
```

Common Bringup function configuration function.

Common Bringup function configuration function from log file. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
logFormat	Choose the format between 0 and 5.
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

◆ configAfeFromFileFormat0()

```
int8_t configAfeFromFileFormat0 ( uint8_t  afeId,
                                  char *   file,
                                  uint8_t  breakAtPollFail,
                                  uint8_t  breakAtReadCheckFail
                                )
```

Bringup function configuration function from log file for format 0 of Latte log.

Bringup function configuration function from log file for format 0 of Latte log. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

◆ configAfeFromFileFormat5()

```
int8_t configAfeFromFileFormat5 ( uint8_t  afeId,
                                  char *   file,
                                  uint8_t  breakAtPollFail,
                                  uint8_t  breakAtReadCheckFail
                                )
```

Bringup function configuration function from log file for format 5 of Latte log.

Bringup function configuration function from log file for format 5 of Latte log. This function needs to be changed if the input to the function is not as file path.

Parameters

afeId	AFE ID
file	Log File Path as generated by Latte.
breakAtPollFail	If this is 0, then configuration will continue when some poll fails. If it is 1, configuration will stop when some poll fails.
breakAtReadCheckFail	If this is 0, then configuration will continue when some SPI Read Check fails. If it is 1, configuration will stop when some SPI Read Checks fails.

Returns

Returns if AFE initialization passed or failed.

Generated by doxygen 1.8.17

jesd.c File Reference

This file has JESD related functions.

Version 2.2:

More...

```
#include <stdint.h>
#include <afe79xxLog.h>
#include <afe79xxTypes.h>
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
#include "controls.h"
#include "jesd.h"
```

Functions

uint8_t	**dacJesdSendData** (uint8_t afeId, uint8_t topno) Send JESD Data from SerDes to DAC. More...
uint8_t	**dacJesdConstantTestPatternValue** (uint8_t afeId, uint8_t topno, uint8_t enable, uint8_t chNo, uint8_t bandNo, uint16_t valueI, uint16_t valueQ) Send Constant Test Pattern to DAC. More...
uint8_t	**dacJesdSendRampTestPattern** (uint8_t afeId, uint8_t topno, uint8_t increment) Send Ramp Test pattern to DAC. More...
uint8_t	**getJesdRxLaneErrors** (uint8_t afeId, uint8_t laneNo, uint8_t *error) Read the DAC JESD Lane Errors. More...
uint8_t	**getJesdRxLaneFifoErrors** (uint8_t afeId, uint8_t laneNo, uint8_t *error) Read the DAC JESD Lane FIFO Errors. More...
uint8_t	**getJesdRxMiscSerdesErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t *errorValue) Read the DAC JESD Miscellaneous Errors. More...
uint8_t	**getJesdRxAlarms** (uint8_t afeId, uint8_t *error) Read all the DAC JESD Errors. More...
uint8_t	**getJesdRxLinkStatus** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for DAC JESD. More...
uint8_t	**getJesdRxLinkStatus204B** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for for DAC JESD204B. More...
uint8_t	**getJesdRxLinkStatus204C** (uint8_t afeId, uint16_t *linkStatus) Read Link Status for for DAC JESD204B. More...
uint8_t	**clearJesdRxAlarms** (uint8_t afeId) Clears DAC JESD JESD204 alarms going to the pin. More...
uint8_t	**clearJesdRxAlarmsForPap** (uint8_t afeId) Clears DAC JESD JESD204 alarms going to the PAP block. More...
uint8_t	**jesdRxClearSyncErrorCnt** (uint8_t afeId, uint8_t jesdNo) Clears the Sync Error counter for DAC JESD. More...
uint8_t	**jesdRxGetSyncErrorCnt** (uint8_t afeId, uint8_t jesdNo, uint8_t *linkErrorCount) Reads the Sync Error counter for DAC JESD. More...
uint8_t	**clearJesdTxAlarms** (uint8_t afeId) Clears ADC JESD JESD204 alarms. More...
uint8_t	**jesdTxGetSyncErrorCnt** (uint8_t afeId, uint8_t jesdLaneNo, uint8_t *linkErrorCount) Reads the Sync Error counter for ADC JESD. More...
uint8_t	**adcRampTestPattern** (uint8_t afeId, uint8_t topno, uint8_t chNo, uint8_t enable, uint8_t rampIncr) Send Ramp Test Pattern from ADC JESD. More...
uint8_t	**toggleSync** (uint8_t afeId, uint8_t overrideValue) Toggles the ADC JESD204B Sync Override. More...

uint8_t	**setJesdTxSyncOverride** (uint8_t afeId, uint8_t syncNo, uint8_t overrideValue, uint8_t syncValue) Overrides the SyncIn of the ADC JESD204B. More...
uint8_t	**setJesdRxSyncOverride** (uint8_t afeId, uint8_t syncNo, uint8_t overrideValue, uint8_t syncValue) Overrides the SyncOut of the DAC JESD204B. More...
uint8_t	**getJesdTxFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t *errors) Reads the ADC JESD Lane FIFO Errors. More...
uint8_t	**jesdRxFullResetToggle** (uint8_t afeId, uint8_t jesdNo) Resets the DAC JESD Block. More...
uint8_t	**jesdTxFullResetToggle** (uint8_t afeId, uint8_t jesdNo) Resets the ADC JESD Block. More...
uint8_t	**jesdRxClearDataPath** (uint8_t afeId, uint8_t jesdNo) Clears the DAC JESD Data path. More...
uint8_t	**jesdTxClearDataPath** (uint8_t afeId, uint8_t jesdNo) Clears the ADC JESD Data path. More...
uint8_t	**adcDacSync** (uint8_t afeId, uint8_t pinSysref) Resets and relinks the AFE JESD. More...
uint8_t	**jesdRxResetStateMachine** (uint8_t afeId, uint8_t linkNo) Resets the DAC JESD State Machine. More...
uint8_t	**checkIfRbdIsGood** (uint8_t afeId, uint8_t jesdNo, uint8_t *rbdStatus) Checks if the set RBD value is okay or not. More...
uint8_t	**getAllLaneReady** (uint8_t afeId, uint8_t jesdNo, uint8_t *rbdOffset) Returns the all lane ready counter. More...
uint8_t	**setGoodRbd** (uint8_t afeId, uint8_t jesdNo) Set Good RBD of DAC JESD. More...
uint8_t	**maskJesdRxLaneErrors** (uint8_t afeId, uint8_t laneNo, uint8_t maskValue) Mask DAC JESD Lane Errors to Pin. More...
uint8_t	**maskJesdRxLaneFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t losMaskValue, uint8_t fifoMaskValue) Mask DAC JESD FIFO Errors to Pin. More...
uint8_t	**maskJesdRxMiscSerdesErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t maskSerdesPllLock) Mask DAC JESD Miscellaneous Errors to Pin. More...
uint8_t	**maskJesdTxFifoErrors** (uint8_t afeId, uint8_t jesdNo, uint8_t maskValue) Mask ADC JESD FIFO Errors to Pin. More...
uint8_t	**maskJesdRxLaneErrorsToPap** (uint8_t afeId, uint8_t laneNo, uint8_t maskValue) Mask DAC JESD Lane Errors to PAP. More...
uint8_t	**maskJesdRxLaneFifoErrorsToPap** (uint8_t afeId, uint8_t jesdNo, uint8_t losMaskValue, uint8_t fifoMaskValue) Mask DAC JESD FIFO Errors to PAP. More...
uint8_t	**maskJesdRxMiscSerdesErrorsToPap** (uint8_t afeId, uint8_t jesdNo, uint8_t maskSerdesPllLock) Mask DAC JESD Miscellaneous Errors to PAP. More...
uint8_t	**setManualRbd** (uint8_t afeId, uint8_t jesdNo, uint8_t value) Sets the RBS valuw. More...

Detailed Description

This file has JESD related functions.

Version 2.2:

1. Fixed bug in adcDacSync.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Added function getAllLaneReady which is useful for setting the RBD.
3. Fixed dacJesdConstantTestPatternValue function for higher DAC interface rates.
4. Removed redundant writes in functions.
5. Changed the C macros for all the spi wrapper and executeMacro function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.
6. Added functions jesdRxFullResetToggle, jesdTxFullResetToggle, jesdRxClearDataPath and jesdTxClearDataPath. Cleaned up adcDacSync by calling these sub functions.

Function Documentation

◆ adcDacSync()

```
uint8_t adcDacSync ( uint8_t  afeId,
                     uint8_t  pinSysref
                   )
```

Resets and relinks the AFE JESD.

This resets all the JESD blocks, gives sysref to the AFE and checks for the DAC link status.

Note the following:

1. Contents of the giveSingleSysrefPulse function should be replaced by host function to give Pin Sysref to AFE. This is used only in case of a single shot sysref.
2. For Continuous Syref mode, external Pin Sysref should be enabled before this function is called. Note that even in this case, only one pulse edge will be captured by the AFE. In this mode, giveSingleSysrefPulse needn't do any operation.
3. systemParams[afeId].spiInUseForPllAccess should be set before the function call to the appropriate value for selecting SPIA/SPIB. In Normal use-case SPIA is used and hence can be left at the default.
4. The selection between the single shot and continuous sysref mode should be done in Latte during generation of the configuration log.
5. systemParams[afeId].syncLoopBack and systemParams[afeId].jesdProtocol should be appropriately set according to what is there in the initialization.
6. This doesn't perform any SerDes operations
7. Any ASIC operations needed for the relink should be done as needed.

Parameters

afeId	AFE ID
pinSysref	Chooses between pinSysref and internal copy of Sysref 0-Uses the internal copy of the Sysref to relink the JESD. 1-Uses Pin sysref for relink. Note that when the pin sysref is made 0, the internal copy of the Sysref is used which is not same as the SPI override of the pin sysref. The internal copy of the sysref will be synchronous to the previous copy of the sysref the AFE received. And since the Sysref frequency is calculated accounts for it, deterministic latency will be achieved even in this mode, assuming the phase of the external sysref is not disturbed.

Returns

Returns if the function execution passed or failed or if the relink is not successful.

◆ adcRampTestPattern()

```
uint8_t adcRampTestPattern ( uint8_t  afeId,
                             uint8_t  topno,
                             uint8_t  chNo,
                             uint8_t  enable,
                             uint8_t  rampIncr
                           )
```

Send Ramp Test Pattern from ADC JESD.

Send Ramp Test Pattern from ADC JESD. This test pattern is near the ADC-JESD interface.

Parameters

afeId	AFE ID
topno	Select the JESD instance. 0-AB. 1-CD.
chNo	0 for RXA/C; 1 for RX B/D; 2 for FB AB/CD
enable	1 to enable the Ramp pattern. 0 to disable.
rampIncr	rampIncr+1 is the increment of the steps.

Returns

Returns if the function execution passed or failed.

◆ checkIfRbdIsGood()

```
uint8_t checkIfRbdIsGood ( uint8_t    afeId,
                           uint8_t    jesdNo,
                           uint8_t *  rbdStatus
                         )
```

Checks if the set RBD value is okay or not.

Checks if the set RBD value is okay or not.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.
rbdStatus	Pointer return. Value will be 1 if the RBD set is good, else returns 0.

Returns

Returns if the function execution passed or failed.

◆ clearJesdRxAlarms()

uint8_t clearJesdRxAlarms (uint8_t afeId)

Clears DAC JESD JESD204 alarms going to the pin.

Clears DAC JESD JESD204 alarms going to the pin of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearJesdRxAlarmsForPap()

uint8_t clearJesdRxAlarmsForPap (uint8_t afeId)

Clears DAC JESD JESD204 alarms going to the PAP block.

Clears DAC JESD JESD204 alarms going to the PAP block of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ clearJesdTxAlarms()

uint8_t clearJesdTxAlarms (uint8_t afeId)

Clears ADC JESD JESD204 alarms.

Clears ADC JESD JESD204 alarms of all lanes.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ dacJesdConstantTestPatternValue()

```
uint8_t dacJesdConstantTestPatternValue ( uint8_t   afeId,
                                          uint8_t   topno,
                                          uint8_t   enable,
                                          uint8_t   chNo,
                                          uint8_t   bandNo,
                                          uint16_t  valueI,
                                          uint16_t  valueQ
                                        )
```

Send Constant Test Pattern to DAC.

Send Constant Test Pattern to DAC. This test pattern is near the JESD-DUC interface. The output of the DAC will be a single tone for each band at the mixer frequency.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.
enable	0-Send Data From SERDES. 1- Send Constant Test Pattern. This is common for AB/CD.
chNo	0-A/C, 1-B/D
bandNo	0-Band 0, 1- Band1. In single band case, this should always be 0.
valueI	Value to be sent on I
valueQ	Value to be sent on Q

Returns

Returns if the function execution passed or failed.

◆ dacJesdSendData()

```
uint8_t dacJesdSendData ( uint8_t  afeId,
                          uint8_t  topno
                        )
```

Send JESD Data from SerDes to DAC.

Send JESD Data from SerDes. This should be called to change from test pattern mode to normal data mode.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.

Returns

Returns if the function execution passed or failed.

◆ dacJesdSendRampTestPattern()

```
uint8_t dacJesdSendRampTestPattern ( uint8_t  afeId,
                                     uint8_t  topno,
                                     uint8_t  increment
                                   )
```

Send Ramp Test pattern to DAC.

Send Ramp Test pattern to DAC. This test pattern is near the JESD-DUC interface.

Parameters

afeId	AFE ID
topno	0-AB and 1-CD.
increment	increment+1 is the step value of the ramp.

Returns

Returns if the function execution passed or failed.

◆ getAllLaneReady()

```
uint8_t getAllLaneReady ( uint8_t    afeId,
                          uint8_t    jesdNo,
                          uint8_t *  rbdOffset
                        )
```

Returns the all lane ready counter.

This function reads the all lane ready counter which is the offset between the internal LMFC boundary and the multiframe boundary (in JESD204B) or extended multi block boundary (in JESD204C) of the last lane of arrival. This value after an offset (of say, 2) with modulus of 64 should be writen to the RBD register.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.
rbdOffset	Pointer return. Value of the last all lane ready counter.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxAlarms()

```
uint8_t getJesdRxAlarms ( uint8_t   afeId,
                          uint8_t * error
                        )
```

Read all the DAC JESD Errors.

Reads all the DAC JESD Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId AFE ID

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLaneErrors()

```
uint8_t getJesdRxLaneErrors ( uint8_t   afeId,
                              uint8_t   laneNo,
                              uint8_t * error
                            )
```

Read the DAC JESD Lane Errors.

Reads the DAC JESD Lane Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId AFE ID

laneNo JESD Lane Number. Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLaneFifoErrors()

```
uint8_t getJesdRxLaneFifoErrors ( uint8_t   afeId,
                                  uint8_t   laneNo,
                                  uint8_t * error
                                )
```

Read the DAC JESD Lane FIFO Errors.

Reads the DAC JESD Lane FIFO Errors, logs their meaning and returns the alarm status as pointer. These are SerDes FIFO errors. If this error is present, either the SerDes is likely seeing some eye based issues .

Parameters

afeId AFE ID

laneNo JESD Lane Number. Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).

error Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus()

```
uint8_t getJesdRxLinkStatus ( uint8_t    afeId,
                              uint16_t * linkStatus
                            )
```

Read Link Status for DAC JESD.

Reads link status for DAC JESD for all the enabled lanes and returns it. This calls functions getJesdRxLinkStatus204B/getJesdRxLinkStatus204C based on set systemParams[afeId].jesdProtocol.

Parameters

afeId AFE ID

linkStatus Pointer return of the status.

Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes.

=0 Idle state. No change in state.

=1 CGS Passed. Still in K characters mode.

=2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus204B()

```
uint8_t getJesdRxLinkStatus204B ( uint8_t     afeId,
                                  uint16_t *  linkStatus
                                )
```

Read Link Status for for DAC JESD204B.

Reads link status for all the enabled lanes and returns it.

Parameters

afeId	AFE ID
linkStatus	Pointer return of the status. Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes. =0 Idle state. No change in state. =1 In JESD204B: CGS Passed. Still in K characters mode. In JESD204C:Header Aligned but EoEMB lock yet to happen. =2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxLinkStatus204C()

```
uint8_t getJesdRxLinkStatus204C ( uint8_t     afeId,
                                  uint16_t *  linkStatus
                                )
```

Read Link Status for for DAC JESD204B.

Reads link status for all the enabled lanes and returns it.

Parameters

afeId	AFE ID
linkStatus	Pointer return of the status. Return Value is 4 bits. 2 bits for top 4 lanes and 2 bits for bottom 4 lanes. =0 Idle state. No change in state. =1 Header Aligned but EoEMB lock yet to happen. =2 Link is up.

Returns

Returns if the function execution passed or failed.

◆ getJesdRxMiscSerdesErrors()

```
uint8_t getJesdRxMiscSerdesErrors ( uint8_t   afeId,
                                    uint8_t   jesdNo,
                                    uint8_t * errorValue
                                  )
```

Read the DAC JESD Miscellaneous Errors.

Reads the DAC JESD Miscellaneous Errors, logs their meaning and returns the alarm status as pointer.

Parameters

afeId	AFE ID
jesdNo	JESD Instance Number (0/1). Note that this is the JESD Lane Number which is post JESD-SerDes Mux(towards the AFE side).
errorValue	Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ getJesdTxFifoErrors()

```
uint8_t getJesdTxFifoErrors ( uint8_t   afeId,
                              uint8_t   jesdNo,
                              uint8_t * errors
                            )
```

Reads the ADC JESD Lane FIFO Errors.

Reads the ADC JESD Lane FIFO Errors, logs their meaning and returns the alarm status as pointer

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.
errors	Pointer return of the status. Value of zero means there is no error and any non-zero value refers to an error.

Returns

Returns if the function execution passed or failed.

◆ jesdRxClearDataPath()

```
uint8_t jesdRxClearDataPath ( uint8_t  afeId,
                              uint8_t  jesdNo
                            )
```

Clears the DAC JESD Data path.

Clears the DAC JESD Data Path by sending 0s to the AFE DAC JESD IP Layer and again sends data for the SerDes. Note that relink is needed after this.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdRxClearSyncErrorCnt()

```
uint8_t jesdRxClearSyncErrorCnt ( uint8_t  afeId,
                                  uint8_t  jesdNo
                                )
```

Clears the Sync Error counter for DAC JESD.

Clears the Sync Error counter for DAC JESD.

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.

Returns

Returns if the function execution passed or failed.

◆ jesdRxFullResetToggle()

```
uint8_t jesdRxFullResetToggle ( uint8_t  afeId,
                                uint8_t  jesdNo
                              )
```

Resets the DAC JESD Block.

Resets the DAC JESD Block. Sysref should be given to the complete AFE after doing this. It is recommended to always keep jesdNo as 3.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdRxGetSyncErrorCnt()

```
uint8_t jesdRxGetSyncErrorCnt ( uint8_t   afeId,
                                uint8_t   jesdNo,
                                uint8_t * linkErrorCount
                              )
```

Reads the Sync Error counter for DAC JESD.

Reads the Sync Error counter for DAC JESD and returns it as a pointer.

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
linkErrorCount	Pointer returns the counter value of the sync error. This denotes the number of times the resync request was given since the last time it is cleared.

Returns

Returns if the function execution passed or failed.

◆ jesdRxResetStateMachine()

```
uint8_t jesdRxResetStateMachine ( uint8_t afeId,
                                  uint8_t linkNo
                                )
```

Resets the DAC JESD State Machine.

Resets the DAC JESD State Machine. In this case no Sysref is needed to be given to the AFE. The LMFC boundary will remain aligned to the previous sysref AFE received.

Parameters

afeId	AFE ID
linkNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdTxClearDataPath()

```
uint8_t jesdTxClearDataPath ( uint8_t  afeId,
                              uint8_t  jesdNo
                            )
```

Clears the ADC JESD Data path.

Clears the ADC JESD Data Path by sending 0s on the SerDes and again sends data. Note that for some time (approximately time of one SPI write + time between consecutive SPI writes), the AFE SerDes TX transmits 0s and it may be needed for ASIC/FPGA SerDes RX to freeze during this time.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdTxFullResetToggle()

```
uint8_t jesdTxFullResetToggle ( uint8_t  afeId,
                                uint8_t  jesdNo
                              )
```

Resets the ADC JESD Block.

Resets the ADC JESD Block. Sysref should be given to the complete AFE after doing this. It is recommended to always keep jesdNo as 3.

Parameters

afeId	AFE ID
jesdNo	1 for JESD AB Instance. 2 for JESD CD Instance. 3 for JESD AB & CD Instance.

Returns

Returns if the function execution passed or failed.

◆ jesdTxGetSyncErrorCnt()

```
uint8_t jesdTxGetSyncErrorCnt ( uint8_t   afeId,
                                uint8_t   jesdLaneNo,
                                uint8_t * linkErrorCount
                              )
```

Reads the Sync Error counter for ADC JESD.

Reads the Sync Error counter for ADC JESD and returns it as a pointer.

Parameters

afeId	AFE ID
jesdLaneNo	0-7 is the lane number pre-lane mux (towards the AFE).
linkErrorCount	Pointer returns the counter value of the sync error. This denotes the number of times the resync request was given since the last time the JESD was reset. There is no clear counter for this.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneErrors()

```
uint8_t maskJesdRxLaneErrors ( uint8_t afeId,
                               uint8_t laneNo,
                               uint8_t maskValue
                             )
```

Mask DAC JESD Lane Errors to Pin.

Mask DAC JESD Lane Errors to Pin.

Parameters

afeId	AFE ID
laneNo	The laneNo is the post-laneMux lane number 0-7 (towards the AFE).

maskValue	The bits made 1 will be masked and not reflect on pin.
	Bit No 7 = "multiframe alignment error";
	Bit No 6 = "frame alignment error";
	Bit No 5 = "link configuration error";
	Bit No 4 = "elastic buffer overflow (bad RBD value)";
	Bit No 3 = "elastic buffer match error. The first no-/K/ does not match 'match_ctrl' and 'match_data' programmed values";
	Bit No 2 = "code synchronization error";
	Bit No 1 = "JESD 204B: 8b/10b not-in-table code error. JESD 204C: sync_header_invalid_err";
	Bit No 0 = "JESD 204B: 8b/10b disparty error. JESD 204C: sync_header_parity_err";

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneErrorsToPap()

```
uint8_t maskJesdRxLaneErrorsToPap ( uint8_t  afeId,
                                    uint8_t  laneNo,
                                    uint8_t  maskValue
                                  )
```

Mask DAC JESD Lane Errors to PAP.

Mask DAC JESD Lane Errors to PAP.

Parameters

afeId	AFE ID
laneNo	The laneNo is the post-laneMux lane number 0-7 (towards the AFE).

maskValue	The bits made 1 will be masked and not reflect on PAP.
	Bit No 7 = "multiframe alignment error";
	Bit No 6 = "frame alignment error";
	Bit No 5 = "link configuration error";
	Bit No 4 = "elastic buffer overflow (bad RBD value)";
	Bit No 3 = "elastic buffer match error. The first no-/K/ does not match 'match_ctrl' and 'match_data' programmed values";
	Bit No 2 = "code synchronization error";
	Bit No 1 = "JESD 204B: 8b/10b not-in-table code error. JESD 204C: sync_header_invalid_err";
	Bit No 0 = "JESD 204B: 8b/10b disparty error. JESD 204C: sync_header_parity_err";

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneFifoErrors()

```
uint8_t maskJesdRxLaneFifoErrors ( uint8_t  afeId,
                                   uint8_t  jesdNo,
                                   uint8_t  losMaskValue,
                                   uint8_t  fifoMaskValue
                                 )
```

Mask DAC JESD FIFO Errors to Pin.

Mask DAC JESD FIFO Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
losMaskValue	The bits made 1 will be masked and not reflect on pin. Bits0-3 for SerDes Rx Lane Loss of lock error for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.
fifoMaskValue	The bits made 1 will be masked and not reflect on pin. Bits0-3 for SerDes Rx Lane FIFO Error for for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxLaneFifoErrorsToPap()

```
uint8_t maskJesdRxLaneFifoErrorsToPap ( uint8_t  afeId,
                                        uint8_t  jesdNo,
                                        uint8_t  losMaskValue,
                                        uint8_t  fifoMaskValue
                                      )
```

Mask DAC JESD FIFO Errors to PAP.

Mask DAC JESD FIFO Errors to PAP

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
losMaskValue	The bits made 1 will be masked and not reflect on PAP. Bits0-3 for SerDes Rx Lane Loss of lock error for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.
fifoMaskValue	The bits made 1 will be masked and not reflect on PAP. Bits0-3 for SerDes Rx Lane FIFO Error for for lanes 0-3 when jesdNo=0 and lanes 4-7 when jesdNo=1. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxMiscSerdesErrors()

```
uint8_t maskJesdRxMiscSerdesErrors ( uint8_t  afeId,
                                     uint8_t  jesdNo,
                                     uint8_t  maskSerdesPllLock
                                   )
```

Mask DAC JESD Miscellaneous Errors to Pin.

Mask DAC JESD Miscellaneous Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskSerdesPllLock	The bits made 1 will be masked and not reflect on pin. Bit 0 for SRX1-4 and Bit 1 for SRX 5-8. These are Actual SerDes Lane numbers. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdRxMiscSerdesErrorsToPap()

```
uint8_t maskJesdRxMiscSerdesErrorsToPap ( uint8_t  afeId,
                                          uint8_t  jesdNo,
                                          uint8_t  maskSerdesPllLock
                                        )
```

Mask DAC JESD Miscellaneous Errors to PAP.

Mask DAC JESD Miscellaneous Errors to PAP

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskSerdesPllLock	The bits made 1 will be masked and not reflect on PAP. Bit 0 for SRX1-4 and Bit 1 for SRX 5-8. These are Actual SerDes Lane numbers. These lane numbers are post lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ maskJesdTxFifoErrors()

```
uint8_t maskJesdTxFifoErrors ( uint8_t  afeId,
                               uint8_t  jesdNo,
                               uint8_t  maskValue
                             )
```

Mask ADC JESD FIFO Errors to Pin.

Mask ADC JESD FIFO Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
maskValue	The bits made 1 will be masked and not reflect on pin. bit 0 is for lane 0 errors, bit 1 for lane 1 errors bit 2 for lane 2 errors bit 3 for lane 3 errors The lane number is the lane number of this instance pre-lane mux, towards the AFE.

Returns

Returns if the function execution passed or failed.

◆ setGoodRbd()

```
uint8_t setGoodRbd ( uint8_t  afeId,
                     uint8_t  jesdNo
                   )
```

Set Good RBD of DAC JESD.

This function does the following:

1. Reads the internal counter between the internal LMFC counter to the received LMFC boundary (multi frame boundary in 204B and Extended Multi Block Boundary in 204C).
2. Sets the RBD by giving an offset of 4 to the LMFC Counter.
3. Relinks by calling the adcDacSync function with pinSysref=1. So all the related conditions of adcDacSync are to be satisfied here too.
   Note that this sequence may not ensure deterministic latency across bring-ups and devices. For acheiving it the above 3 steps should be executed using the functions getAllLaneReady, setManualRbd, adcDacSync with external pin sysref. The getAllLaneReady should be done only once and the same value should be loaded each time during the intialization. This can be input as a parameter to Latte.

Parameters

afeId	AFE ID
jesdNo	0 for JESD AB Instance. 1 for JESD CD Instance.

Returns

Returns if the function execution passed or failed.

◆ setJesdRxSyncOverride()

```
uint8_t setJesdRxSyncOverride ( uint8_t  afeId,
                                uint8_t  syncNo,
                                uint8_t  overrideValue,
                                uint8_t  syncValue
                              )
```

Overrides the SyncOut of the DAC JESD204B.

Overrides the SyncOut of the DAC JESD204B.

Parameters

afeId	AFE ID
syncNo	syncNo the sync value. 0-3
overrideValue	0- do not override. 1- override the SyncIn pin
syncValue	Pin state

Returns

Returns if the function execution passed or failed.

◆ setJesdTxSyncOverride()

```
uint8_t setJesdTxSyncOverride ( uint8_t  afeId,
                                uint8_t  syncNo,
                                uint8_t  overrideValue,
                                uint8_t  syncValue
                              )
```

Overrides the SyncIn of the ADC JESD204B.

Overrides the SyncIn of the ADC JESD204B.

Parameters

afeId	AFE ID
syncNo	syncNo the sync value. 0-5
overrideValue	0- do not override. 1- override the SyncIn pin
syncValue	0- Send K characters. 1- Send Data

Returns

Returns if the function execution passed or failed.

◆ setManualRbd()

```
uint8_t setManualRbd ( uint8_t  afeId,
                       uint8_t  jesdNo,
                       uint8_t  value
                     )
```

Sets the RBS valuw.

Mask DAC JESD Miscellaneous Errors to Pin

Parameters

afeId	AFE ID
jesdNo	0 for AB and 1 for CD.
value	RBD Value.

Returns

Returns if the function execution passed or failed.

◆ toggleSync()

```
uint8_t toggleSync ( uint8_t  afeId,
                     uint8_t  overrideValue
                   )
```

Toggles the ADC JESD204B Sync Override.

Override the SyncIN override, forces K characters for 100ms and then sends the data. This is to be used only in software sync mode in JESD 204B.

Parameters

afeId	AFE ID
overrideValue	Overrides the Sync Pin to this value during K characters mode. Bits 0-5 refer to syncin numbers 0-5. This can be made 0x3f to send K characters on all the links. To toggle sync of only a particular link, need to set only that bit to 1. For example, to toggle only link 2 using SyncIn2, need to set this value to 0x04.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

pap.c File Reference

This file has PAP related functions.

Version 2.1:

More...

```
#include <stdint.h>
#include <math.h>
#include "afe79xxLog.h"
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
#include "pap.h"
```

Functions

uint8_t	**configurePapMaDet** (uint8_t afeId, uint8_t chno, uint8_t maEnable, uint16_t maNumSample, uint16_t maWindowCntr, uint16_t maWindowCntrTh, uint16_t maThreshB0, uint16_t maThreshB1, uint16_t maThreshComb) Configure the Moving Average PAP Detector. More...
uint8_t	**configurePapHpfDet** (uint8_t afeId, uint8_t chno, uint8_t hpfEnable, uint16_t hpfNumSample, uint16_t hpfWindowCntr, uint16_t hpfWindowCntrTh, uint16_t hpfThreshB0, uint16_t hpfThreshB1, uint16_t hpfThreshComb) Configure the High Pass Filter PAP Detector. More...
uint8_t	**configurePap** (uint8_t afeId, uint8_t chno, uint8_t enable, uint8_t multMode, uint8_t rampDownStartVal, uint8_t attnStepSize, uint8_t gainStepSize, uint8_t detectInWaitState, float triggerToRampDown, float waitCounter, float triggerClearToRampUp, float amplUpdateCycles, float alarmPulseGPIO, uint8_t alarmMask, uint8_t alarmChannelMask, uint8_t alarmPinDynamicMode, uint8_t rampStickyMode) Configure the PAP Block. More...
uint8_t	**rampStickyClear** (uint8_t afeId, uint8_t chno) Clear the ramp sticky state. More...
uint8_t	**papAlarmStatus** (uint8_t afeId, uint8_t chno, uint8_t *alarmTriggered) Reads the PAP alarm Status. More...
uint8_t	**clearPapAlarms** (uint8_t afeId, uint8_t chno)

	Clears the PAP alarm Status. More...
uint8_t	**configLaneErrorsForTxPap** (uint8_t afeId, uint8_t chno, uint8_t laneMask) Map the DAC JESD lane errors to the PAP block. More...

Detailed Description

This file has PAP related functions.

Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Changed the C macros for all the spi wrapper and executeMacro function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.

Function Documentation

◆ clearPapAlarms()

```
uint8_t clearPapAlarms ( uint8_t  afeId,
                         uint8_t  chno
                       )
```

Clears the PAP alarm Status.

Clears the PAP alarm sticky status which also goes to GPIO.

Parameters

afeId AFE ID

chno Value 1 = Clear, Value 0 = No Clear

bit [0] :TxA Alarm

bit [1]: TxB Alarm

bit [2]: TxC Alarm

bit [3]: TxD Alarm

Returns

Returns if the function execution passed or failed.

◆ configLaneErrorsForTxPap()

```
uint8_t configLaneErrorsForTxPap ( uint8_t  afeId,
                                   uint8_t  chno,
                                   uint8_t  laneMask
                                 )
```

Map the DAC JESD lane errors to the PAP block.

This function chooses which lanes' errors should go to a particular TX PAP.

Parameters

afeId AFE ID

chno Select the TX Channel

0 for TXA

1 for TXB

2 for TXC

3 for TXD

laneMask This is 8-bit field with bit wise enable for lane errors.

Bit0 is for lane0, Bit1 for lane1, Bit2 for lane 2 and so on.

The errors from lanes with corresponding bits set to 1 will reach the PAP.

The lane numbers are pre-lane mux (towards the AFE).

For example, for TXA PAP to get errors from lanes 0 and 1, laneMask should be 0b00000011.

Registers written are (tx<a/b/c/d>_lane_alarms_to_pap_en) in DAC JESD.s

Returns

Returns if the function execution passed or failed.

◆ configurePap()

```
uint8_t configurePap ( uint8_t  afeId,
                       uint8_t  chno,
                       uint8_t  enable,
                       uint8_t  multMode,
                       uint8_t  rampDownStartVal,
                       uint8_t  attnStepSize,
                       uint8_t  gainStepSize,
                       uint8_t  detectInWaitState,
                       float    triggerToRampDown,
                       float    waitCounter,
                       float    triggerClearToRampUp,
                       float    amplUpdateCycles,
                       float    alarmPulseGPIO,
                       uint8_t  alarmMask,
                       uint8_t  alarmChannelMask,
                       uint8_t  alarmPinDynamicMode,
                       uint8_t  rampStickyMode
                     )
```

Configure the PAP Block.

Configure the PAP Block. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
enable	1 to enable PAP. 0 To disable PAP
multMode	Mode of Ramp up/down. 0:Cosine 1:Linear
rampDownStartVal	This is the starting value for the ramp down. Supported range: 0 to 127. For cosine mode, the start phase in radians is (128-rampDownStartVal)* pi /128. For linear mode, (rampDownStartVal/128) is the start value.
attnStepSize	This is the ramp step size while ramping down (actualSampleStep=attnStepSize*(last good sample)/1024). Supported Range is 0 to 127.
gainStepSize	This is the ramp step size while gaining up (actualSampleStep=GainStepSize*(last good sample)/1024). Supported Range is 0 to 127.
detectInWaitState	This determines if the PAP trigger should be acknowledged in the wait state. 0:Do not detect in wait state 1:detect in wait state
triggerToRampDown	Time from trigger occurance to ramp down time (ns). Supported Range: 0 to floor(65520000.0/Fdac) where Fdac is in MHz.
waitCounter	Wait time counter (ns). Supported Range: 0 to floor(1048560000.0/Fdac) where Fdac is in MHz.
triggerClearToRampUp	Time from end of wait state to Ramp up (ns). Supported Range: 0 to floor(65520000.0/Fdac) where Fdac is in MHz.
amplUpdateCycles	Time for each step during ramp up or down. (ns). Supported Range: 0 to floor(2032000.0/Fdac) where Fdac is in MHz.
alarmPulseGPIO	Pulse width of PAP alarm going to GPIO (ns). Supported Range: 0 to floor(1048560000.0/Fdac) where Fdac is in MHz.
alarmMask	

	Bit wise alarms. Bit value 0 will make corresponding alarm to trigger PAP state machine. BitNo: Alarm 0 : pll_alarm, 1 : serdes_alarm, 2 : fifo_alarm, 3 : ovr_saturation_alarm, 4 : dual band det alarm, 5 : combined band det alarm, 6 : spi trigger
alarmChannelMask	Mask other channels (bit-wise). For each channel the bit-wise description is different. Ch : BitNo 3-2-1-0 TxA : D-C-B-A, TxB : D-C-A-B, TxC : B-A-D-C, TxD : B-A-C-D
alarmPinDynamicMode	Determines if the PAP Pin is sticky or non-sticky. 1:dynamic, 0:sticky
rampStickyMode	Determines if the Ramp up mode is sticky or non-sticky. 0:Automatically come to ramp up mode after wait state. 1: Wait for pap clear bit to be written

Returns

Returns if the function execution passed or failed.

◆ configurePapHpfDet()

```
uint8_t configurePapHpfDet ( uint8_t   afeId,
                             uint8_t   chno,
                             uint8_t   hpfEnable,
                             uint16_t  hpfNumSample,
                             uint16_t  hpfWindowCntr,
                             uint16_t  hpfWindowCntrTh,
                             uint16_t  hpfThreshB0,
                             uint16_t  hpfThreshB1,
                             uint16_t  hpfThreshComb
                           )
```

Configure the High Pass Filter PAP Detector.

Configure the High Pass Filter PAP Detector. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
hpfEnable	0: Disable High Pass Filter based PAP detector. 1:Enable High Pass Filter based PAP detector.
hpfNumSample	Number of samples in a window. Supported values: 1-32; 2-64; 3-128 Samples
hpfWindowCntr	Number of windows. Supported Range: 0 to 2**12-1
hpfWindowCntrTh	Window Counter Threshold. When the number of windows in a set of maWindowCntr windows have filter trigger. This should be lower than maWindowCntr. Supported Range: 0:2**12-1.
hpfThreshB0	(128*val) is the filter threshold for band 0 detector. Supported Range: 0-511
hpfThreshB1	(128*val) is the filter threshold for band 1 detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.
hpfThreshComb	(128*val) is the filter threshold for combined detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.

Returns

Returns if the function execution passed or failed.

◆ configurePapMaDet()

```
uint8_t configurePapMaDet ( uint8_t   afeId,
                            uint8_t   chno,
                            uint8_t   maEnable,
                            uint16_t  maNumSample,
                            uint16_t  maWindowCntr,
                            uint16_t  maWindowCntrTh,
                            uint16_t  maThreshB0,
                            uint16_t  maThreshB1,
                            uint16_t  maThreshComb
                          )
```

Configure the Moving Average PAP Detector.

Configure the Moving Average PAP Detector. Note that all the System Parameters should be set as per the configuration before calling this.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
maEnable	0: Disable Moving Average based PAP detector. 1:Enable Moving Average based PAP detector.
maNumSample	Number of samples in a window. Supported values: 1-32; 2-64; 3-128 Samples
maWindowCntr	Number of windows. Supported Range: 0 to 2**12-1
maWindowCntrTh	Window Counter Threshold. When the number of windows in a set of maWindowCntr windows have power above the power threshold. This should be lower than or equal to maWindowCntr. Supported Range: 0:2**12-1.
maThreshB0	(128*val) is the power threshold for band 0 detector. Supported Range: 0-511
maThreshB1	(128*val) is the power threshold for band 1 detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.
maThreshComb	(128*val) is the power threshold for combined detector. Supported Range: 0-511. Valid only in dual band use case. In single band usecase, make this equal to maThreshB0.

Returns

Returns if the function execution passed or failed.

◆ papAlarmStatus()

```
uint8_t papAlarmStatus ( uint8_t   afeId,
                         uint8_t   chno,
                         uint8_t * alarmTriggered
                       )
```

Reads the PAP alarm Status.

Reads the PAP alarm Status and returns it as a pointer. This is sticky status and clearPapAlarms needs to be called to clear the status.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD
alarmTriggered	Pointer return of the status. If this value is 1, then there was a PAP trigger.

Returns

Returns if the function execution passed or failed.

◆ rampStickyClear()

```
uint8_t rampStickyClear ( uint8_t afeId,
                          uint8_t chno
                        )
```

Clear the ramp sticky state.

In case where rampStickyMode is set, this function should be called to clear the PAP alarm and move to ramp up state.

Parameters

afeId	AFE ID
chno	Select the TX Channel 0 for TXA 1 for TXB 2 for TXC 3 for TXD

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

serDes.c File Reference

This file has SerDes related functions.

Version 2.2:

More...

```
#include <stdint.h>
#include "afe79xxLog.h"
```

```
#include "afe79xxTypes.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeParameters.h"
#include "serDes.h"
```

Functions

uint8_t	**serdesTx1010Pattern** (uint8_t afeId, uint8_t laneNo) Send 1010 toggling pattern on AFE SerDes TX. More...
uint8_t	**serdesTxSendData** (uint8_t afeId, uint8_t laneNo) Send JESD data on AFE SerDes TX. More...
uint8_t	**SetSerdesTxCursor** (uint8_t afeId, uint8_t laneNo, uint8_t mainCursorSetting, uint8_t preCursorSetting, uint8_t postCursorSetting) Set SerDes TX Cursor. More...
uint8_t	**getSerdesRxPrbsError** (uint8_t afeId, uint8_t laneNo, uint32_t *errorRegValue) Read the AFE SerDes RX PRBS error. More...
uint8_t	**clearSerdesRxPrbsErrorCounter** (uint8_t afeId, uint8_t laneNo) Clear the AFE SerDes RX PRBS error counter. More...
uint8_t	**enableSerdesRxPrbsCheck** (uint8_t afeId, uint8_t laneNo, uint8_t prbsMode, uint8_t enable) Enables the AFE SerDes RX PRBS check. More...
uint8_t	**sendSerdesTxPrbs** (uint8_t afeId, uint8_t laneNo, uint8_t prbsMode, uint8_t enable) Sends the AFE SerDes TX PRBS pattern. More...
uint8_t	**getSerdesRxLaneEyeMarginValue** (uint8_t afeId, uint8_t laneNo, uint16_t *regValue) Reads the AFE SerDes RX Eye margin value. More...
uint8_t	**resetSerDesDfeLane** (uint8_t afeId, uint8_t laneNo) Resets the AFE SerDes RX DFE lane. More...
uint8_t	**reAdaptSerDesLane** (uint8_t afeId, uint8_t laneNo) Readapts the AFE SerDes RX lane. More...
uint8_t	**resetSerDesDfeAllLanes** (uint8_t afeId) Resets DFE of all the AFE SerDes RX lanes. More...
uint8_t	**reAdaptSerDesAllLanes** (uint8_t afeId) Readapts all the AFE SerDes RX lanes. More...
uint8_t	**parse_response** (uint8_t afeId, uint16_t *responseRet) Checks the status of the SerDes Eye Read. More...
uint8_t	**em_start** (uint8_t afeId, uint8_t lane_num, uint8_t ber_exp, uint8_t mode) Initiates the SerDes Eye Read. More...
uint8_t	**em_report_progress** (uint8_t afeId, uint8_t *progress) Checks the Progress of the SerDes Eye Read. More...
uint8_t	**em_read** (uint8_t afeId, uint16_t *ber) Reads the SerDes Eye parameters. More...
uint8_t	**em_cancel** (uint8_t afeId) Stops the the SerDes Eye Read. More...
uint8_t	**getSerdesEye** (uint8_t afeId, uint8_t laneNo, uint16_t *ber, uint16_t *extent) Reads the SerDes Eye for a given lane. More...

Detailed Description

This file has SerDes related functions.

Version 2.2:

1. Fixed a bug in getSerDesEye function.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Updated function definition of: getSerdesEye.
3. Changed the C macros for all the spi wrapper and executeMacro function calls to AFE_FUNC_EXEC from AFE_SPI_EXEC.
4. Added functions resetSerDesDfeLane, reAdaptSerDesLane, resetSerDesDfeAllLanes and reAdaptSerDesAllLanes.

Function Documentation

◆ clearSerdesRxPrbsErrorCounter()

```
uint8_t clearSerdesRxPrbsErrorCounter ( uint8_t  afeId,
                                        uint8_t  laneNo
                                      )
```

Clear the AFE SerDes RX PRBS error counter.

Clearss the AFE SerDes RX PRBS error counter.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ em_cancel()

```
uint8_t em_cancel ( uint8_t  afeId )
```

Stops the the SerDes Eye Read.

Stops the the SerDes Eye Read. This function is called getSerdesEye and shouldn't be called independently.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ em_read()

```
uint8_t em_read ( uint8_t    afeId,
                  uint16_t * ber
                )
```

Reads the SerDes Eye parameters.

Reads the SerDes Eye parameters. This function is called getSerdesEye and shouldn't be called independently.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ em_report_progress()

```
uint8_t em_report_progress ( uint8_t   afeId,
                             uint8_t * progress
                           )
```

Checks the Progress of the SerDes Eye Read.

Checks the Progress of the SerDes Eye Read. This function is called getSerdesEye and shouldn't be called independently.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ em_start()

```
uint8_t em_start ( uint8_t afeId,
                   uint8_t lane_num,
                   uint8_t ber_exp,
                   uint8_t mode
                 )
```

Initiates the SerDes Eye Read.

Initiates the SerDes Eye Read. This function is called getSerdesEye and shouldn't be called independently.

Parameters

afeId AFE ID

Returns

Returns if the function execution passed or failed.

◆ enableSerdesRxPrbsCheck()

```
uint8_t enableSerdesRxPrbsCheck ( uint8_t  afeId,
                                  uint8_t  laneNo,
                                  uint8_t  prbsMode,
                                  uint8_t  enable
                                )
```

Enables the AFE SerDes RX PRBS check.

Enables the AFE SerDes RX PRBS check.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
prbsMode	PRBS Mode Selection. 0 for PRBS9, 1 for PRBS15, 2 for PRBS23 and 3 for PRBS31.
enable	1 will enable the PRBS check, 0 will disable the PRBS check.

Returns

Returns if the function execution passed or failed.

◆ getSerdesEye()

```
uint8_t getSerdesEye ( uint8_t    afeId,
                       uint8_t    laneNo,
                       uint16_t * ber,
                       uint16_t * extent
                     )
```

Reads the SerDes Eye for a given lane.

Reads the SerDes Eye for a given lane. The ber array and the extent returned by this function should be fed to the python script to plot the eye diagram.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
ber	Pointer of array with 3135 elements.
extent	scaling factor of the ber needed by the function.

Returns

Returns if the function execution passed or failed.

◆ getSerdesRxLaneEyeMarginValue()

```
uint8_t getSerdesRxLaneEyeMarginValue ( uint8_t    afeId,
                                        uint8_t    laneNo,
                                        uint16_t * regValue
                                      )
```

Reads the AFE SerDes RX Eye margin value.

Reads the AFE SerDes RX Eye margin value and returns the value as a pointer. This value*0.5 is the eye margin in mV post equalization.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
regValue	Eye Margin value status.

Returns

Returns if the function execution passed or failed.

◆ getSerdesRxPrbsError()

```
uint8_t getSerdesRxPrbsError ( uint8_t    afeId,
                               uint8_t    laneNo,
                               uint32_t * errorRegValue
                             )
```

Read the AFE SerDes RX PRBS error.

Reads the AFE SerDes RX PRBS error and returns the error value as pointer.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.
errorRegValue	PRBS error register value. This value increments by 3 for each PRBS error.

Returns

Returns if the function execution passed or failed.

◆ parse_response()

```
uint8_t parse_response ( uint8_t    afeId,
                         uint16_t * responseRet
                       )
```

Checks the status of the SerDes Eye Read.

Checks the status of the SerDes Eye Read. This function is called getSerdesEye and shouldn't be called independently.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ reAdaptSerDesAllLanes()

```
uint8_t reAdaptSerDesAllLanes ( uint8_t  afeId )
```

Readapts all the AFE SerDes RX lanes.

Readapts all the AFE SerDes RX lanes. This calls reAdaptSerDesLane within this function for all the lanes.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ reAdaptSerDesLane()

```
uint8_t reAdaptSerDesLane ( uint8_t  afeId,
                            uint8_t  laneNo
                          )
```

Readapts the AFE SerDes RX lane.

Readapts the AFE SerDes RX lane. This calls resetSerDesDfeLane within this function for the specific lane.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ resetSerDesDfeAllLanes()

```
uint8_t resetSerDesDfeAllLanes ( uint8_t  afeId )
```

Resets DFE of all the AFE SerDes RX lanes.

Resets DFE of all the AFE SerDes RX lanes. This calls resetSerDesDfeLane within this function for all the lanes.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ resetSerDesDfeLane()

```
uint8_t resetSerDesDfeLane ( uint8_t  afeId,
                             uint8_t  laneNo
                           )
```

Resets the AFE SerDes RX DFE lane.

Resets the AFE SerDes RX DFE lane.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to SRX1-SRX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ sendSerdesTxPrbs()

```
uint8_t sendSerdesTxPrbs ( uint8_t  afeId,
                           uint8_t  laneNo,
                           uint8_t  prbsMode,
                           uint8_t  enable
                         )
```

Sends the AFE SerDes TX PRBS pattern.

Sends the AFE SerDes TX PRBS pattern.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.
prbsMode	PRBS Mode Selection. 0 for PRBS9, 1 for PRBS15, 2 for PRBS23 and 3 for PRBS31.
enable	1 will enable the PRBS transmission, 0 will disable the PRBS pattern transmission.

Returns

Returns if the function execution passed or failed.

◆ serdesTx1010Pattern()

```
uint8_t serdesTx1010Pattern ( uint8_t  afeId,
                              uint8_t  laneNo
                            )
```

Send 1010 toggling pattern on AFE SerDes TX.

Send 1010 toggling pattern on AFE SerDes TX.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ serdesTxSendData()

```
uint8_t serdesTxSendData ( uint8_t  afeId,
                           uint8_t  laneNo
                         )
```

Send JESD data on AFE SerDes TX.

Send JESD data pattern on AFE SerDes TX.

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.

Returns

Returns if the function execution passed or failed.

◆ SetSerdesTxCursor()

```
uint8_t SetSerdesTxCursor ( uint8_t  afeId,
                            uint8_t  laneNo,
                            uint8_t  mainCursorSetting,
                            uint8_t  preCursorSetting,
                            uint8_t  postCursorSetting
                          )
```

Set SerDes TX Cursor.

Set SerDes TX Cursor. Below table shows the mapping between different settings and the equalization it provides.

Column (1):Pre-Cursor equalization acheived. (dB in relative to post cursor)

Column (2):Main Cursor equalization acheived.(dB in relative to pre cursor)

Column (3):Post-Cursor equalization acheived.(dB in relative to pre cursor)

Column (4):Pre-Cursor Setting to be programmed.

Column (5):Main Setting to be programmed.

Column (6):Post-Cursor setting to be programmed.

(1)	(2)	(3)	(4)	(5)	(6)
0	25	0	0	0	0
0	23	0	0	1	0
0	21	0	0	2	0
0	19	0	0	3	0
0	17	0	0	4	0
0	15	0	0	5	0
0	13	0	0	6	0
0	11	0	0	7	0
0	24	0.72	0	0	1
0	20	0.87	0	2	1
0	16	1.09	0	4	1
0	12	1.45	0	6	1
0	23	1.51	0	0	2
0	21	1.66	0	1	2
0	15	2.33	0	4	2
0	22	2.38	0	0	3
0	13	2.69	0	5	2
0	21	3.35	0	0	4
0	19	3.71	0	1	4
0	14	3.78	0	4	3
0	17	4.17	0	2	2
0	20	4.44	0	0	5
0	15	4.75	0	3	2
0	16	5.62	0	2	5
0	19	5.68	0	0	6
0	17	6.41	0	1	6
0	18	7.13	0	0	7
0.72	24	0	1	0	0
0.87	20	0	1	2	0
0.87	22	1.66	1	0	2
1.09	16	0	1	4	0
1.09	20	3.71	1	0	4
1.45	12	0	1	2	0
1.45	14	2.69	1	4	2

1.45	16	4.75	1	2	4
1.45	18	6.41	1	0	6
1.51	23	0	2	0	0
1.66	21	0	2	1	0
1.66	22	0.87	2	0	1
2.33	15	0	2	4	0
2.33	19	4.17	2	0	4
2.38	22	0	3	0	0
2.69	18	5.62	2	0	5
2.69	13	0	2	5	0
2.69	14	1.45	2	4	1
2.69	17	4.75	2	1	4
3.35	21	0	4	0	0
3.71	19	0	4	1	0
3.71	20	1.09	4	0	1
3.78	18	4.75	3	0	4
3.78	14	0	3	4	0
4.17	17	0	4	2	0
4.17	19	2.33	4	0	2
4.44	20	0	5	0	0
4.75	15	0	4	3	0
4.75	16	1.45	4	2	1
4.75	18	3.78	4	0	3
4.75	17	2.69	4	1	2
5.62	16	0	5	2	0
5.62	18	2.69	5	0	2
5.68	19	0	6	0	0
6.41	18	1.45	6	0	1
6.41	17	0	6	1	0
7.13	18	0	7	0	0

Parameters

afeId	AFE ID
laneNo	Values 0-7, refer to STX1-STX8, the physical SerDes lanes.
mainCursorSetting	Main Cursor Setting.
preCursorSetting	Pre Cursor Setting.
postCursorSetting	Post Cursor Setting.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

Afe79xxUser Directory Reference

Directories

directory **Src**

Generated by doxygen 1.8.17

Src Directory Reference

Files

file **afeParameters.c**

This file contains System Parameters used in AFE initialization.

file **baseFunc.c**

This file has functions which can be edited by customers to integrate it into their system.

Version 2.1.1:

afeParameters.c File Reference

This file contains System Parameters used in AFE initialization. More...

```
#include <stdint.h>
#include <string.h>
#include "afe79xxTypes.h"
#include "afe79xxLog.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "afeParameters.h"
```

Variables

struct **afeSystemParamsStruct** **systemParams** [**NUM_OF_AFE**]

Detailed Description

This file contains System Parameters used in AFE initialization.

Variable Documentation

◆ systemParams

struct **afeSystemParamsStruct** systemParams[**NUM_OF_AFE**]

systemParams is the Array of structures contains the System Parameters used in the intialization script for each AFE.

Some of the system parameters, which are static for a use case, like sampling and interface rates, are captured in this structure, systemParams. This is to prevent passing these redundantly for related functions. For some variables this may act as a state variable to capture current state.

This can be generated for each AFE configuration by running AFE.saveCAfeParamsFile() in Latte after generating the initial configuration.

baseFunc.c File Reference

This file has functions which can be edited by customers to integrate it into their system.

Version 2.1.1:

More...

```
#include <stdio.h>
#include <stdint.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>
#include "afe79xxTypes.h"
#include "afe79xxLog.h"
#include "baseFunc.h"
#include "basicFunctions.h"
#include "afeCommonMacros.h"
```

Functions

uint8_t	**dev_spi_write** (uint8_t afeId, uint16_t addr, uint8_t data) AFE SPI Write driver function. More...
uint8_t	**dev_spi_read** (uint8_t afeId, uint16_t addr, uint8_t *readVal) AFE SPI read driver function. More...
uint8_t	**giveSingleSysrefPulse** (uint8_t afeId) AFE single shot Pin Sysref. More...
uint8_t	**giveAfeAdcInput** (uint8_t afeId, uint8_t chNo, uint8_t bandNo) Give ADC input for factory Calibration. More...
uint8_t	**connectAfeTxToFb** (uint8_t afeId, uint8_t txChNo, uint8_t fbChNo, uint8_t bandNo) Connect TX Output to FB for factory Calibration. More...
uint8_t	**wait** (uint32_t wait_s) Wait in Seconds. More...
uint8_t	**waitMs** (uint32_t wait_ms) Wait in milli Seconds. More...
void	**setAfeLogLvl** (uint32_t level) Set the AFE Log Level. More...
uint32_t	**getAfeLogLvl** () Get the AFE Log Level. More...
void	**afeLogmsg** (uint32_t level, const char *pcLogFmt,...) Logging function. More...

Detailed Description

This file has functions which can be edited by customers to integrate it into their system.

Version 2.1.1:

1. Fixed warnings in the bringup functions.

   Version 2.1:

1. Added documentation and improved the parameter validity checks.
2. Added functions to bringup from file.
3. Added functions for calibration.

Function Documentation

◆ afeLogmsg()

```
void afeLogmsg ( uint32_t     level,
                 const char * pcLogFmt,
                              ...
               )
```

Logging function.

The contents of this function should be replaced by host driver function.
Can handle different log levels differently.

Parameters

level	This logger level of the caller function.

Returns

Returns the AFE Log Level.

◆ connectAfeTxToFb()

```
uint8_t connectAfeTxToFb ( uint8_t afeId,
                           uint8_t txChNo,
                           uint8_t fbChNo,
                           uint8_t bandNo
                         )
```

Connect TX Output to FB for factory Calibration.

Connect TX Output to FB for factory Calibration.
The contents of this function contents should be replaced by host driver function.

Parameters

afeId	AFE ID
txChNo	Bit Wise TX Channel Select. Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD
fbChNo	Bit Wise FB Channel Select. Bit0 for FBAB Bit1 for FBCD When this is 1 or 2, connect the TX represented by txChNo to FBAB or FBCD respectively. When this is 3, connect the TX represented by txChNo[1:0] to FBAB and TX represented by txChNo[3:2] to FBCD
bandNo	Bit Wise Band Select. Bit0 for Band 0 Bit1 for Band 1

Returns

Returns if the function execution passed or failed.

◆ dev_spi_read()

```
uint8_t dev_spi_read ( uint8_t   afeId,
                       uint16_t  addr,
                       uint8_t * readVal
                     )
```

AFE SPI read driver function.

AFE SPI read driver function and returns the read value as pointer. The contents of this function should be replaced by host SPI driver function.

Parameters

afeId	AFE ID
addr	Address to be read from.
readVal	Pointer return of the value read.

Returns

Returns if the function execution passed or failed.

◆ dev_spi_write()

```
uint8_t dev_spi_write ( uint8_t  afeId,
                        uint16_t addr,
                        uint8_t  data
                      )
```

AFE SPI Write driver function.

AFE SPI Write driver function. The contents of this function should be replaced by host SPI driver function.

Parameters

afeId	AFE ID
addr	Address to be written to.
data	value to be written.

Returns

Returns if the function execution passed or failed.

◆ getAfeLogLvl()

```
uint32_t getAfeLogLvl (  )
```

Get the AFE Log Level.

Returns the AFE Log Level. There are multiple levels of logging as below.

AFE_LOG_LEVEL_ERROR 0 : Error conditions

AFE_LOG_LEVEL_WARNING 1 : warning conditions

AFE_LOG_LEVEL_INFO 2 : informational

AFE_LOG_LEVEL_SPILOG 3 : SPI-level messages

AFE_LOG_LEVEL_DEBUG 4 : debug-level messages

Returns

Returns the AFE Log Level.

◆ giveAfeAdcInput()

```
uint8_t giveAfeAdcInput ( uint8_t  afeId,
                          uint8_t  chNo,
                          uint8_t  bandNo
                        )
```

Give ADC input for factory Calibration.

Give ADC input for factory Calibration. The contents of this function contents should be replaced by host driver function.

Parameters

afeId	AFE ID
chNo	Channel Number. 0-RXA 1-RXB 2-RXC 3-RXD 4-FBAB 5-FBCD
bandNo	Band Number 0/1.

Returns

Returns if the function execution passed or failed.

◆ giveSingleSysrefPulse()

```
uint8_t giveSingleSysrefPulse ( uint8_t  afeId )
```

AFE single shot Pin Sysref.

AFE single shot pin sysref driver function. The contents of this function should be replaced by host driver function.

Parameters

afeId	AFE ID

Returns

Returns if the function execution passed or failed.

◆ setAfeLogLvl()

```
void setAfeLogLvl ( uint32_t  level )
```

Set the AFE Log Level.

Sets the AFE Log Level. There are multiple levels of logging as below.

AFE_LOG_LEVEL_ERROR 0 : Error conditions

AFE_LOG_LEVEL_WARNING 1 : warning conditions

AFE_LOG_LEVEL_INFO 2 : informational

AFE_LOG_LEVEL_SPILOG 3 : SPI-level messages

AFE_LOG_LEVEL_DEBUG 4 : debug-level messages

Parameters

level	Log level.

◆ wait()

```
uint8_t wait ( uint32_t  wait_s )
```

Wait in Seconds.

Wait in Seconds. The contents of this function should be replaced by host driver function.

Parameters

wait_s Wait time in seconds.

Returns

Returns if the function execution passed or failed.

◆ waitMs()

```
uint8_t waitMs ( uint32_t  wait_ms )
```

Wait in milli Seconds.

Wait in milli Seconds. The contents of this function should be replaced by host driver function.

Parameters

wait_ms Wait time in seconds.

Returns

Returns if the function execution passed or failed.

Generated by doxygen 1.8.17

example Directory Reference

Files

file	**main.c**

Generated by doxygen 1.8.17

main.c File Reference

```
#include <stdint.h>
#include <stdio.h>
#include "afe79xxTypes.h"
#include "afe79xxLog.h"
#include "afeCommonMacros.h"
#include "baseFunc.h"
#include "afeParameters.h"
#include "basicFunctions.h"
#include "hMacro.h"
#include "controls.h"
#include "calibrations.h"
#include "dsaAndNco.h"
#include "jesd.h"
#include "serDes.h"
```

```
#include "pap.h"
#include "agc.h"
```

Functions

int **main** (void)

Function Documentation

◆ main()

int main (void)

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- a -

- adcDacSync() : **jesd.h** , **jesd.c**
- adcRampTestPattern() : **jesd.c** , **jesd.h**
- AFE_AGC_MAX_ABS_NUM_HITS : **afe79xxTypes.h**
- AFE_AGC_MAX_WIN_LEN : **afe79xxTypes.h**
- AFE_FB_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_FB_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_FUNC_EXEC : **afeCommonMacros.h**
- AFE_ID_VALIDITY : **afeCommonMacros.h**
- AFE_LOG_LEVEL_DEBUG : **afe79xxLog.h**
- AFE_LOG_LEVEL_ERROR : **afe79xxLog.h**
- AFE_LOG_LEVEL_INFO : **afe79xxLog.h**
- AFE_LOG_LEVEL_SPILOG : **afe79xxLog.h**
- AFE_LOG_LEVEL_WARNING : **afe79xxLog.h**
- AFE_MACRO_DONE_POLL_FAIL : **afeCommonMacros.h**
- AFE_MACRO_ERROR_IN_EXECUTION : **afe79xxTypes.h**
- AFE_MACRO_ERROR_IN_OPCODE : **afe79xxTypes.h**
- AFE_MACRO_ERROR_IN_OPERAND : **afe79xxTypes.h**
- AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED : **afe79xxTypes.h**
- AFE_MACRO_EXEC_ERROR : **afeCommonMacros.h**
- AFE_MACRO_EXTENDED_ERROR_CODE_REG_ADDR : **afe79xxTypes.h**
- AFE_MACRO_NO_ERROR : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_DET_TIME_CONST_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_DIG_DET_ABSOLUTE_NUM_CROSSINGS_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_DIG_DET_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_DIG_DET_RELATIVE_NUM_CROSSINGS_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_EXT_LNA_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_EXT_LNA_GAIN_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_GAIN_STEP_SIZE_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_RF_ANALOG_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_AGC_STATE_CONTROL : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_ALC_CONFIGURATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_APPLY_DSA_GAIN_PHASE_COMPENSATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_COARSE_FINE_MODE_ALC : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_CONFIG_SIGGEN_FOR_CAL : **afe79xxTypes.h**

- AFE_MACRO_OPCODE_EXT_AGC_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_FACTORY_RX_DSA_GAIN_PHASE_CALIBRATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_FACTORY_TX_DSA_GAIN_PHASE_CALIBRATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_FLOATING_POINT_CONFIG_ALC : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_INT_AGC_CONTROLLER_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_MIN_MAX_DSA_ATTN_CONFIG : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_PREPARE_FOR_TUNE : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_REG_ADDR : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_SYSTEM_TUNE : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_SYSTEM_TUNE_SELECTIVE : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_FB_CHANNEL_FREQUENCY_CONFIGURATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_RX_CHANNEL_FREQUENCY_CONFIGURATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION_ALL_BANDS : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_TX_DIG_PARAM : **afe79xxTypes.h**
- AFE_MACRO_OPCODE_UPDATE_TX_GAIN : **afe79xxTypes.h**
- AFE_MACRO_OPERAND_START_REG_ADDR : **afe79xxTypes.h**
- AFE_MACRO_PAGE_REG_ADDR : **afe79xxTypes.h**
- AFE_MACRO_PAGE_SEL_VAL : **afe79xxTypes.h**
- AFE_MACRO_READY_POLL_FAIL : **afeCommonMacros.h**
- AFE_MACRO_RESULT_START_REG_ADDR : **afe79xxTypes.h**
- AFE_MACRO_STATUS_REG_ADDR : **afe79xxTypes.h**
- AFE_NUM_BANDS_PER_FB : **afe79xxTypes.h**
- AFE_NUM_BANDS_PER_RX : **afe79xxTypes.h**
- AFE_NUM_BANDS_PER_TX : **afe79xxTypes.h**
- AFE_NUM_CH_PER_JESD_INSTANCE : **afe79xxTypes.h**
- AFE_NUM_FB_CHANNELS : **afe79xxTypes.h**
- AFE_NUM_FB_CHANNELS_BITWISE : **afe79xxTypes.h**
- AFE_NUM_JESD_INSTANCES : **afe79xxTypes.h**
- AFE_NUM_RX_CHANNELS : **afe79xxTypes.h**
- AFE_NUM_RX_CHANNELS_BITWISE : **afe79xxTypes.h**
- AFE_NUM_SERDES_LANES : **afe79xxTypes.h**
- AFE_NUM_TX_CHANNELS : **afe79xxTypes.h**
- AFE_NUM_TX_CHANNELS_BITWISE : **afe79xxTypes.h**
- AFE_PAGE_END_ADDR : **afe79xxTypes.h**
- AFE_PAGE_START_ADDR : **afe79xxTypes.h**
- AFE_PARAMS_VALID : **afeCommonMacros.h**
- AFE_REQ_SPI_ACCESS_MAX_COUNT : **basicFunctions.c**
- AFE_RX_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_RX_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_RX_DSA_MAX_DIG_DSA_INDEX : **afe79xxTypes.h**
- AFE_SPI_EXEC : **afeCommonMacros.h**
- AFE_TX_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_TX_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_TX_DSA_MAX_ANA_PLUS_DIG_DSA_DB : **afe79xxTypes.h**
- afeLogDbg : **afe79xxLog.h**
- afeLogErr : **afe79xxLog.h**
- afeLogInfo : **afe79xxLog.h**
- afeLogmsg() : **baseFunc.h** , **baseFunc.c**
- afeLogSpiLog : **afe79xxLog.h**
- afeSpiCheckWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiPollLogWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiPollWrapper() : **basicFunctions.h** , **basicFunctions.c**

- afeSpiReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiWriteWrapper() : **basicFunctions.c** , **basicFunctions.h**
- agcDigDetAbsoluteNumCrossingConfig() : **agc.h** , **agc.c**
- agcDigDetConfig() : **agc.c** , **agc.h**
- agcDigDetRelativeNumCrossingConfig() : **agc.c** , **agc.h**
- agcDigDetTimeConstantConfig() : **agc.h** , **agc.c**
- agcGainStepSizeConfig() : **agc.h** , **agc.c**
- agcStateControlConfig() : **agc.c** , **agc.h**
- alcConfig() : **agc.c** , **agc.h**
- ARRAY_SIZE : **afeCommonMacros.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- c -

- CFG_SPI_READ_POLL_MAX_COUNT : **basicFunctions.c**
- checkDeviceHealth() : **controls.h** , **controls.c**
- checkForMacroError() : **hMacro.h** , **hMacro.c**
- checkIfRbdIsGood() : **jesd.h** , **jesd.c**
- checkMcuHealth() : **hMacro.h** , **hMacro.c**
- checkPllLockStatus() : **controls.h** , **controls.c**
- checkSysref() : **controls.h** , **controls.c**
- clearAllAlarms() : **controls.h** , **controls.c**
- clearJesdRxAlarms() : **jesd.h** , **jesd.c**
- clearJesdRxAlarmsForPap() : **jesd.h** , **jesd.c**
- clearJesdTxAlarms() : **jesd.h** , **jesd.c**
- clearPapAlarms() : **pap.h** , **pap.c**
- clearPllStickyLockStatus() : **controls.h** , **controls.c**
- clearSerdesRxPrbsErrorCounter() : **serDes.c** , **serDes.h**
- clearSpiAlarms() : **controls.h** , **controls.c**
- closeAllPages() : **basicFunctions.c** , **basicFunctions.h**
- coarseFineConfig() : **agc.h** , **agc.c**
- configAfeFromFile() : **init.h** , **init.c**
- configAfeFromFileFormat0() : **init.h** , **init.c**
- configAfeFromFileFormat5() : **init.h** , **init.c**
- configLaneErrorsForTxPap() : **pap.c** , **pap.h**
- configurePap() : **pap.c** , **pap.h**
- configurePapHpfDet() : **pap.c** , **pap.h**
- configurePapMaDet() : **pap.h** , **pap.c**
- connectAfeTxToFb() : **baseFunc.h** , **baseFunc.c**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- d -

- dacJesdConstantTestPatternValue() : **jesd.h** , **jesd.c**
- dacJesdSendData() : **jesd.c** , **jesd.h**
- dacJesdSendRampTestPattern() : **jesd.h** , **jesd.c**
- dev_spi_read() : **baseFunc.c** , **baseFunc.h**
- dev_spi_write() : **baseFunc.h** , **baseFunc.c**
- doRxDsaCalib() : **calibrations.h** , **calibrations.c**
- doSystemTuneSelective() : **hMacro.h** , **hMacro.c**

- doTxDsaCalib() : **calibrations.h** , **calibrations.c**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- e -

- em_cancel() : **serDes.c**
- em_read() : **serDes.c**
- em_report_progress() : **serDes.c**
- em_start() : **serDes.c**
- enableMemAccess() : **hMacro.h** , **hMacro.c**
- enableSerdesRxPrbsCheck() : **serDes.c** , **serDes.h**
- executeMacro() : **hMacro.h** , **hMacro.c**
- externalAgcConfig() : **agc.h** , **agc.c**
- extLnaConfig() : **agc.h** , **agc.c**
- extLnaGainConfig() : **agc.c** , **agc.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- f -

- fbDsaPerTxEn() : **dsaAndNco.h** , **dsaAndNco.c**
- fbNCOSel() : **dsaAndNco.c** , **dsaAndNco.h**
- fltPtConfig() : **agc.c** , **agc.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- g -

- get_agcMode() : **paramsSetterGetter.h**
- get_bigStepAttkEn() : **paramsSetterGetter.h**
- get_bigStepAttkThresh() : **paramsSetterGetter.h**
- get_bigStepAttkWinLen() : **paramsSetterGetter.h**
- get_bigStepDecEn() : **paramsSetterGetter.h**
- get_bigStepDecThresh() : **paramsSetterGetter.h**
- get_chipId() : **paramsSetterGetter.h**
- get_chipVersion() : **paramsSetterGetter.h**
- get_ddcFactorFb() : **paramsSetterGetter.h**
- get_ddcFactorRx() : **paramsSetterGetter.h**
- get_decayWinLen() : **paramsSetterGetter.h**
- get_ducFactorTx() : **paramsSetterGetter.h**
- get_enableDacInterleavedMode() : **paramsSetterGetter.h**
- get_FadcFb() : **paramsSetterGetter.h**
- get_FadcRx() : **paramsSetterGetter.h**
- get_Fdac() : **paramsSetterGetter.h**
- get_FRef() : **paramsSetterGetter.h**
- get_halfRateModeFb() : **paramsSetterGetter.h**
- get_halfRateModeRx() : **paramsSetterGetter.h**
- get_halfRateModeTx() : **paramsSetterGetter.h**
- get_jesdProtocol() : **paramsSetterGetter.h**
- get_miscStepAttkWinLen() : **paramsSetterGetter.h**

- get_ncoFreqMode() : **paramsSetterGetter.h**
- get_numBandsRx() : **paramsSetterGetter.h**
- get_numBandsTx() : **paramsSetterGetter.h**
- get_numFbNCO() : **paramsSetterGetter.h**
- get_numRxNCO() : **paramsSetterGetter.h**
- get_numTxNCO() : **paramsSetterGetter.h**
- get_powerAttkEn() : **paramsSetterGetter.h**
- get_powerAttkThresh() : **paramsSetterGetter.h**
- get_powerDecEn() : **paramsSetterGetter.h**
- get_powerDecThresh() : **paramsSetterGetter.h**
- get_smallStepAttkEn() : **paramsSetterGetter.h**
- get_smallStepAttkThresh() : **paramsSetterGetter.h**
- get_smallStepDecEn() : **paramsSetterGetter.h**
- get_smallStepDecThresh() : **paramsSetterGetter.h**
- get_spiInUseForPllAccess() : **paramsSetterGetter.h**
- get_syncLoopBack() : **paramsSetterGetter.h**
- get_txToFbMode() : **paramsSetterGetter.h**
- get_useSpiSysref() : **paramsSetterGetter.h**
- get_X() : **paramsSetterGetter.h**
- getAfeLogLvl() : **baseFunc.c** , **baseFunc.h**
- getAllLaneReady() : **jesd.h** , **jesd.c**
- getChipVersion() : **controls.h** , **controls.c**
- getJesdRxAlarms() : **jesd.h** , **jesd.c**
- getJesdRxLaneErrors() : **jesd.h** , **jesd.c**
- getJesdRxLaneFifoErrors() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus204B() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus204C() : **jesd.c** , **jesd.h**
- getJesdRxMiscSerdesErrors() : **jesd.c** , **jesd.h**
- getJesdTxFifoErrors() : **jesd.h** , **jesd.c**
- getRxRmsPower() : **controls.h** , **controls.c**
- getSerdesEye() : **serDes.c** , **serDes.h**
- getSerdesRxLaneEyeMarginValue() : **serDes.c** , **serDes.h**
- getSerdesRxPrbsError() : **serDes.h** , **serDes.c**
- getSystemParam() : **afeParameters.h**
- giveAfeAdcInput() : **baseFunc.h** , **baseFunc.c**
- giveSingleSysrefPulse() : **baseFunc.h** , **baseFunc.c**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- i -

- internalAgcConfig() : **agc.h** , **agc.c**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- j -

- jesdRxClearDataPath() : **jesd.c**
- jesdRxClearSyncErrorCnt() : **jesd.h** , **jesd.c**
- jesdRxFullResetToggle() : **jesd.h** , **jesd.c**
- jesdRxGetSyncErrorCnt() : **jesd.h** , **jesd.c**

- jesdRxResetStateMachine() : **jesd.h** , **jesd.c**
- jesdToSerdesLaneMapping : **afe79xxTypes.h**
- jesdTxClearDataPath() : **jesd.c**
- jesdTxFullResetToggle() : **jesd.c** , **jesd.h**
- jesdTxGetSyncErrorCnt() : **jesd.h** , **jesd.c**

Here is a list of all file members with links to the files they belong to:

- l -

- loadRxDsaPacket() : **calibrations.h** , **calibrations.c**
- loadTxDsaPacket() : **calibrations.h** , **calibrations.c**

Here is a list of all file members with links to the files they belong to:

- m -

- main() : **main.c**
- MASK_BYTE : **basicFunctions.c**
- MASK_SHORT : **basicFunctions.c**
- maskJesdRxLaneErrors() : **jesd.h** , **jesd.c**
- maskJesdRxLaneErrorsToPap() : **jesd.h** , **jesd.c**
- maskJesdRxLaneFifoErrors() : **jesd.h** , **jesd.c**
- maskJesdRxLaneFifoErrorsToPap() : **jesd.h** , **jesd.c**
- maskJesdRxMiscSerdesErrors() : **jesd.h** , **jesd.c**
- maskJesdRxMiscSerdesErrorsToPap() : **jesd.h** , **jesd.c**
- maskJesdTxFifoErrors() : **jesd.c** , **jesd.h**
- minMaxDsaAttnConfig() : **agc.c** , **agc.h**

Here is a list of all file members with links to the files they belong to:

- n -

- NULL : **afe79xxTypes.h**
- NUM_OF_AFE : **afeCommonMacros.h**

Here is a list of all file members with links to the files they belong to:

- o -

- overrideAlarmPin() : **controls.h** , **controls.c**
- overrideDigPkDetPin() : **controls.c** , **controls.h**
- overrideRelDetPin() : **controls.h** , **controls.c**
- overrideTdd() : **controls.h** , **controls.c**
- overrideTddPins() : **controls.c** , **controls.h**

Here is a list of all file members with links to the files they belong to:

- p -

- papAlarmStatus() : **pap.h** , **pap.c**
- parse_response() : **serDes.c**
- PLL_SPI_REG_A : **basicFunctions.c**
- PLL_SPI_REG_B : **basicFunctions.c**
- PLL_SPI_REG_OFF : **basicFunctions.c**
- PLL_SPI_REG_TYPE : **basicFunctions.c**
- PLL_SPI_REG_TYPE_SIZE : **basicFunctions.c**
- PllSpiRegType_e : **basicFunctions.c**

Here is a list of all file members with links to the files they belong to:

- r -

- rampStickyClear() : **pap.h** , **pap.c**
- readAlarmPinStatus() : **controls.c** , **controls.h**
- reAdaptSerDesAllLanes() : **serDes.h** , **serDes.c**
- reAdaptSerDesLane() : **serDes.c** , **serDes.h**
- readFbNco() : **dsaAndNco.c**
- readResultRegSpi() : **hMacro.h** , **hMacro.c**
- readRxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- readSpiAlarms() : **controls.h** , **controls.c**
- readTopMem() : **basicFunctions.h** , **basicFunctions.c**
- readTxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- readTxPower() : **controls.h** , **controls.c**
- requestPllSpiAccess() : **basicFunctions.h** , **basicFunctions.c**
- resetSerDesDfeAllLanes() : **serDes.c** , **serDes.h**
- resetSerDesDfeLane() : **serDes.c** , **serDes.h**
- RET_EXEC_FAIL : **afe79xxTypes.h**
- RET_OK : **afe79xxTypes.h**
- RET_TYPE : **afe79xxTypes.h**
- RetType_e : **afe79xxTypes.h**
- rfAnalogDetConfig() : **agc.c** , **agc.h**
- rxNCOSel() : **dsaAndNco.h** , **dsaAndNco.c**

Here is a list of all file members with links to the files they belong to:

- s -

- sendSerdesTxPrbs() : **serDes.h** , **serDes.c**
- sendSysref() : **controls.c** , **controls.h**
- serdesLaneReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- serdesLaneWriteWrapper() : **basicFunctions.c** , **basicFunctions.h**
- serdesRawRead() : **basicFunctions.h** , **basicFunctions.c**
- serdesRawWrite() : **basicFunctions.h** , **basicFunctions.c**
- serdesReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- serdesTx1010Pattern() : **serDes.c** , **serDes.h**
- serdesTxSendData() : **serDes.h** , **serDes.c**
- serdesWriteWrapper() : **basicFunctions.h** , **basicFunctions.c**
- set_agcMode() : **paramsSetterGetter.h**
- set_bigStepAttkEn() : **paramsSetterGetter.h**

- set_bigStepAttkThresh() : **paramsSetterGetter.h**
- set_bigStepAttkWinLen() : **paramsSetterGetter.h**
- set_bigStepDecEn() : **paramsSetterGetter.h**
- set_bigStepDecThresh() : **paramsSetterGetter.h**
- set_chipId() : **paramsSetterGetter.h**
- set_chipVersion() : **paramsSetterGetter.h**
- set_ddcFactorFb() : **paramsSetterGetter.h**
- set_ddcFactorRx() : **paramsSetterGetter.h**
- set_decayWinLen() : **paramsSetterGetter.h**
- set_ducFactorTx() : **paramsSetterGetter.h**
- set_enableDacInterleavedMode() : **paramsSetterGetter.h**
- set_FadcFb() : **paramsSetterGetter.h**
- set_FadcRx() : **paramsSetterGetter.h**
- set_Fdac() : **paramsSetterGetter.h**
- set_FRef() : **paramsSetterGetter.h**
- set_halfRateModeFb() : **paramsSetterGetter.h**
- set_halfRateModeRx() : **paramsSetterGetter.h**
- set_halfRateModeTx() : **paramsSetterGetter.h**
- set_jesdProtocol() : **paramsSetterGetter.h**
- set_miscStepAttkWinLen() : **paramsSetterGetter.h**
- set_ncoFreqMode() : **paramsSetterGetter.h**
- set_numBandsRx() : **paramsSetterGetter.h**
- set_numBandsTx() : **paramsSetterGetter.h**
- set_numFbNCO() : **paramsSetterGetter.h**
- set_numRxNCO() : **paramsSetterGetter.h**
- set_numTxNCO() : **paramsSetterGetter.h**
- set_powerAttkEn() : **paramsSetterGetter.h**
- set_powerAttkThresh() : **paramsSetterGetter.h**
- set_powerDecEn() : **paramsSetterGetter.h**
- set_powerDecThresh() : **paramsSetterGetter.h**
- set_smallStepAttkEn() : **paramsSetterGetter.h**
- set_smallStepAttkThresh() : **paramsSetterGetter.h**
- set_smallStepDecEn() : **paramsSetterGetter.h**
- set_smallStepDecThresh() : **paramsSetterGetter.h**
- set_spiInUseForPllAccess() : **paramsSetterGetter.h**
- set_syncLoopBack() : **paramsSetterGetter.h**
- set_txToFbMode() : **paramsSetterGetter.h**
- set_useSpiSysref() : **paramsSetterGetter.h**
- set_X() : **paramsSetterGetter.h**
- setAfeLogLvl() : **baseFunc.c** , **baseFunc.h**
- setFbDsa() : **dsaAndNco.h** , **dsaAndNco.c**
- setFbDsaPerTx() : **dsaAndNco.h** , **dsaAndNco.c**
- setGoodRbd() : **jesd.c** , **jesd.h**
- setJesdRxSyncOverride() : **jesd.c** , **jesd.h**
- setJesdTxSyncOverride() : **jesd.h** , **jesd.c**
- setManualRbd() : **jesd.h** , **jesd.c**
- setPinRxDsaSettings() : **dsaAndNco.h** , **dsaAndNco.c**
- setRxDigGain() : **dsaAndNco.c** , **dsaAndNco.h**
- setRxDsa() : **dsaAndNco.c** , **dsaAndNco.h**
- setRxDsaMode() : **dsaAndNco.h** , **dsaAndNco.c**
- SetSerdesTxCursor() : **serDes.h** , **serDes.c**
- setTxDigGain() : **dsaAndNco.h** , **dsaAndNco.c**
- setTxDsa() : **dsaAndNco.h** , **dsaAndNco.c**

- splitToByte() : **hMacro.h** , **hMacro.c**
- systemParams : **afeParameters.c** , **afeParameters.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- t -

- toggleSync() : **jesd.h** , **jesd.c**
- triggerMacro() : **hMacro.c** , **hMacro.h**
- txCalibSiggen() : **hMacro.h** , **hMacro.c**
- txDsaIdxGainSwap() : **dsaAndNco.c** , **dsaAndNco.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- u -

- updateFbNco() : **dsaAndNco.h** , **dsaAndNco.c**
- updateRxNco() : **dsaAndNco.c** , **dsaAndNco.h**
- updateSystemTxChannelFreqConfig() : **hMacro.h** , **hMacro.c**
- updateTxGain() : **dsaAndNco.c** , **dsaAndNco.h**
- updateTxGainParam() : **dsaAndNco.h** , **dsaAndNco.c**
- updateTxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- updateTxNcoDb() : **dsaAndNco.c** , **dsaAndNco.h**

Generated by doxygen 1.8.17

Here is a list of all file members with links to the files they belong to:

- w -

- wait() : **baseFunc.h** , **baseFunc.c**
- waitForMacroAck() : **hMacro.c** , **hMacro.h**
- waitForMacroDone() : **hMacro.h** , **hMacro.c**
- waitForMacroReady() : **hMacro.c** , **hMacro.h**
- waitMs() : **baseFunc.c** , **baseFunc.h**
- writeOperandList() : **hMacro.h** , **hMacro.c**

Generated by doxygen 1.8.17

- a -

- adcDacSync() : **jesd.h** , **jesd.c**
- adcRampTestPattern() : **jesd.c** , **jesd.h**
- afeLogmsg() : **baseFunc.h** , **baseFunc.c**
- afeSpiCheckWrapper() : **basicFunctions.c** , **basicFunctions.h**
- afeSpiPollLogWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiPollWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- afeSpiWriteWrapper() : **basicFunctions.c** , **basicFunctions.h**
- agcDigDetAbsoluteNumCrossingConfig() : **agc.h** , **agc.c**
- agcDigDetConfig() : **agc.c** , **agc.h**
- agcDigDetRelativeNumCrossingConfig() : **agc.h** , **agc.c**

- agcDigDetTimeConstantConfig() : **agc.h** , **agc.c**
- agcGainStepSizeConfig() : **agc.h** , **agc.c**
- agcStateControlConfig() : **agc.c** , **agc.h**
- alcConfig() : **agc.h** , **agc.c**

Generated by doxygen 1.8.17

- c -

- checkDeviceHealth() : **controls.h** , **controls.c**
- checkForMacroError() : **hMacro.c** , **hMacro.h**
- checkIfRbdIsGood() : **jesd.h** , **jesd.c**
- checkMcuHealth() : **hMacro.c** , **hMacro.h**
- checkPllLockStatus() : **controls.h** , **controls.c**
- checkSysref() : **controls.h** , **controls.c**
- clearAllAlarms() : **controls.h** , **controls.c**
- clearJesdRxAlarms() : **jesd.c** , **jesd.h**
- clearJesdRxAlarmsForPap() : **jesd.h** , **jesd.c**
- clearJesdTxAlarms() : **jesd.h** , **jesd.c**
- clearPapAlarms() : **pap.h** , **pap.c**
- clearPllStickyLockStatus() : **controls.h** , **controls.c**
- clearSerdesRxPrbsErrorCounter() : **serDes.h** , **serDes.c**
- clearSpiAlarms() : **controls.h** , **controls.c**
- closeAllPages() : **basicFunctions.h** , **basicFunctions.c**
- coarseFineConfig() : **agc.c** , **agc.h**
- configAfeFromFile() : **init.h** , **init.c**
- configAfeFromFileFormat0() : **init.c** , **init.h**
- configAfeFromFileFormat5() : **init.h** , **init.c**
- configLaneErrorsForTxPap() : **pap.h** , **pap.c**
- configurePap() : **pap.c** , **pap.h**
- configurePapHpfDet() : **pap.c** , **pap.h**
- configurePapMaDet() : **pap.c** , **pap.h**
- connectAfeTxToFb() : **baseFunc.c** , **baseFunc.h**

Generated by doxygen 1.8.17

- d -

- dacJesdConstantTestPatternValue() : **jesd.h** , **jesd.c**
- dacJesdSendData() : **jesd.c** , **jesd.h**
- dacJesdSendRampTestPattern() : **jesd.h** , **jesd.c**
- dev_spi_read() : **baseFunc.c** , **baseFunc.h**
- dev_spi_write() : **baseFunc.h** , **baseFunc.c**
- doRxDsaCalib() : **calibrations.h** , **calibrations.c**
- doSystemTuneSelective() : **hMacro.h** , **hMacro.c**
- doTxDsaCalib() : **calibrations.h** , **calibrations.c**

Generated by doxygen 1.8.17

- e -

- em_cancel() : **serDes.c**
- em_read() : **serDes.c**
- em_report_progress() : **serDes.c**
- em_start() : **serDes.c**
- enableMemAccess() : **hMacro.h** , **hMacro.c**
- enableSerdesRxPrbsCheck() : **serDes.c** , **serDes.h**
- executeMacro() : **hMacro.h** , **hMacro.c**
- externalAgcConfig() : **agc.h** , **agc.c**
- extLnaConfig() : **agc.h** , **agc.c**
- extLnaGainConfig() : **agc.c** , **agc.h**

- f -

- fbDsaPerTxEn() : **dsaAndNco.h** , **dsaAndNco.c**
- fbNCOSel() : **dsaAndNco.c** , **dsaAndNco.h**
- fltPtConfig() : **agc.c** , **agc.h**

- g -

- get_agcMode() : **paramsSetterGetter.h**
- get_bigStepAttkEn() : **paramsSetterGetter.h**
- get_bigStepAttkThresh() : **paramsSetterGetter.h**
- get_bigStepAttkWinLen() : **paramsSetterGetter.h**
- get_bigStepDecEn() : **paramsSetterGetter.h**
- get_bigStepDecThresh() : **paramsSetterGetter.h**
- get_chipId() : **paramsSetterGetter.h**
- get_chipVersion() : **paramsSetterGetter.h**
- get_ddcFactorFb() : **paramsSetterGetter.h**
- get_ddcFactorRx() : **paramsSetterGetter.h**
- get_decayWinLen() : **paramsSetterGetter.h**
- get_ducFactorTx() : **paramsSetterGetter.h**
- get_enableDacInterleavedMode() : **paramsSetterGetter.h**
- get_FadcFb() : **paramsSetterGetter.h**
- get_FadcRx() : **paramsSetterGetter.h**
- get_Fdac() : **paramsSetterGetter.h**
- get_FRef() : **paramsSetterGetter.h**
- get_halfRateModeFb() : **paramsSetterGetter.h**
- get_halfRateModeRx() : **paramsSetterGetter.h**
- get_halfRateModeTx() : **paramsSetterGetter.h**
- get_jesdProtocol() : **paramsSetterGetter.h**
- get_miscStepAttkWinLen() : **paramsSetterGetter.h**
- get_ncoFreqMode() : **paramsSetterGetter.h**
- get_numBandsRx() : **paramsSetterGetter.h**
- get_numBandsTx() : **paramsSetterGetter.h**
- get_numFbNCO() : **paramsSetterGetter.h**
- get_numRxNCO() : **paramsSetterGetter.h**
- get_numTxNCO() : **paramsSetterGetter.h**
- get_powerAttkEn() : **paramsSetterGetter.h**

- get_powerAttkThresh() : **paramsSetterGetter.h**
- get_powerDecEn() : **paramsSetterGetter.h**
- get_powerDecThresh() : **paramsSetterGetter.h**
- get_smallStepAttkEn() : **paramsSetterGetter.h**
- get_smallStepAttkThresh() : **paramsSetterGetter.h**
- get_smallStepDecEn() : **paramsSetterGetter.h**
- get_smallStepDecThresh() : **paramsSetterGetter.h**
- get_spiInUseForPllAccess() : **paramsSetterGetter.h**
- get_syncLoopBack() : **paramsSetterGetter.h**
- get_txToFbMode() : **paramsSetterGetter.h**
- get_useSpiSysref() : **paramsSetterGetter.h**
- get_X() : **paramsSetterGetter.h**
- getAfeLogLvl() : **baseFunc.c** , **baseFunc.h**
- getAllLaneReady() : **jesd.h** , **jesd.c**
- getChipVersion() : **controls.h** , **controls.c**
- getJesdRxAlarms() : **jesd.h** , **jesd.c**
- getJesdRxLaneErrors() : **jesd.h** , **jesd.c**
- getJesdRxLaneFifoErrors() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus204B() : **jesd.c** , **jesd.h**
- getJesdRxLinkStatus204C() : **jesd.c** , **jesd.h**
- getJesdRxMiscSerdesErrors() : **jesd.c** , **jesd.h**
- getJesdTxFifoErrors() : **jesd.h** , **jesd.c**
- getRxRmsPower() : **controls.h** , **controls.c**
- getSerdesEye() : **serDes.c** , **serDes.h**
- getSerdesRxLaneEyeMarginValue() : **serDes.c** , **serDes.h**
- getSerdesRxPrbsError() : **serDes.h** , **serDes.c**
- getSystemParam() : **afeParameters.h**
- giveAfeAdcInput() : **baseFunc.h** , **baseFunc.c**
- giveSingleSysrefPulse() : **baseFunc.h** , **baseFunc.c**

Generated by doxygen 1.8.17

- i -

- internalAgcConfig() : **agc.h** , **agc.c**

Generated by doxygen 1.8.17

- j -

- jesdRxClearDataPath() : **jesd.c**
- jesdRxClearSyncErrorCnt() : **jesd.h** , **jesd.c**
- jesdRxFullResetToggle() : **jesd.h** , **jesd.c**
- jesdRxGetSyncErrorCnt() : **jesd.h** , **jesd.c**
- jesdRxResetStateMachine() : **jesd.h** , **jesd.c**
- jesdTxClearDataPath() : **jesd.c**
- jesdTxFullResetToggle() : **jesd.c** , **jesd.h**
- jesdTxGetSyncErrorCnt() : **jesd.c** , **jesd.h**

Generated by doxygen 1.8.17

- l -

- loadRxDsaPacket() : **calibrations.h** , **calibrations.c**
- loadTxDsaPacket() : **calibrations.h** , **calibrations.c**

Generated by doxygen 1.8.17

- m -

- main() : **main.c**
- maskJesdRxLaneErrors() : **jesd.h** , **jesd.c**
- maskJesdRxLaneErrorsToPap() : **jesd.h** , **jesd.c**
- maskJesdRxLaneFifoErrors() : **jesd.h** , **jesd.c**
- maskJesdRxLaneFifoErrorsToPap() : **jesd.h** , **jesd.c**
- maskJesdRxMiscSerdesErrors() : **jesd.c** , **jesd.h**
- maskJesdRxMiscSerdesErrorsToPap() : **jesd.c** , **jesd.h**
- maskJesdTxFifoErrors() : **jesd.h** , **jesd.c**
- minMaxDsaAttnConfig() : **agc.h** , **agc.c**

Generated by doxygen 1.8.17

- o -

- overrideAlarmPin() : **controls.h** , **controls.c**
- overrideDigPkDetPin() : **controls.c** , **controls.h**
- overrideRelDetPin() : **controls.h** , **controls.c**
- overrideTdd() : **controls.h** , **controls.c**
- overrideTddPins() : **controls.c** , **controls.h**

Generated by doxygen 1.8.17

- p -

- papAlarmStatus() : **pap.h** , **pap.c**
- parse_response() : **serDes.c**

Generated by doxygen 1.8.17

- r -

- rampStickyClear() : **pap.h** , **pap.c**
- readAlarmPinStatus() : **controls.c** , **controls.h**
- reAdaptSerDesAllLanes() : **serDes.h** , **serDes.c**
- reAdaptSerDesLane() : **serDes.c** , **serDes.h**
- readFbNco() : **dsaAndNco.c**
- readResultRegSpi() : **hMacro.h** , **hMacro.c**
- readRxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- readSpiAlarms() : **controls.h** , **controls.c**
- readTopMem() : **basicFunctions.h** , **basicFunctions.c**

- readTxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- readTxPower() : **controls.h** , **controls.c**
- requestPllSpiAccess() : **basicFunctions.h** , **basicFunctions.c**
- resetSerDesDfeAllLanes() : **serDes.c** , **serDes.h**
- resetSerDesDfeLane() : **serDes.h** , **serDes.c**
- rfAnalogDetConfig() : **agc.h** , **agc.c**
- rxNCOSel() : **dsaAndNco.h** , **dsaAndNco.c**

Generated by doxygen 1.8.17

- s -

- sendSerdesTxPrbs() : **serDes.h** , **serDes.c**
- sendSysref() : **controls.c** , **controls.h**
- serdesLaneReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- serdesLaneWriteWrapper() : **basicFunctions.c** , **basicFunctions.h**
- serdesRawRead() : **basicFunctions.h** , **basicFunctions.c**
- serdesRawWrite() : **basicFunctions.h** , **basicFunctions.c**
- serdesReadWrapper() : **basicFunctions.h** , **basicFunctions.c**
- serdesTx1010Pattern() : **serDes.c** , **serDes.h**
- serdesTxSendData() : **serDes.h** , **serDes.c**
- serdesWriteWrapper() : **basicFunctions.h** , **basicFunctions.c**
- set_agcMode() : **paramsSetterGetter.h**
- set_bigStepAttkEn() : **paramsSetterGetter.h**
- set_bigStepAttkThresh() : **paramsSetterGetter.h**
- set_bigStepAttkWinLen() : **paramsSetterGetter.h**
- set_bigStepDecEn() : **paramsSetterGetter.h**
- set_bigStepDecThresh() : **paramsSetterGetter.h**
- set_chipId() : **paramsSetterGetter.h**
- set_chipVersion() : **paramsSetterGetter.h**
- set_ddcFactorFb() : **paramsSetterGetter.h**
- set_ddcFactorRx() : **paramsSetterGetter.h**
- set_decayWinLen() : **paramsSetterGetter.h**
- set_ducFactorTx() : **paramsSetterGetter.h**
- set_enableDacInterleavedMode() : **paramsSetterGetter.h**
- set_FadcFb() : **paramsSetterGetter.h**
- set_FadcRx() : **paramsSetterGetter.h**
- set_Fdac() : **paramsSetterGetter.h**
- set_FRef() : **paramsSetterGetter.h**
- set_halfRateModeFb() : **paramsSetterGetter.h**
- set_halfRateModeRx() : **paramsSetterGetter.h**
- set_halfRateModeTx() : **paramsSetterGetter.h**
- set_jesdProtocol() : **paramsSetterGetter.h**
- set_miscStepAttkWinLen() : **paramsSetterGetter.h**
- set_ncoFreqMode() : **paramsSetterGetter.h**
- set_numBandsRx() : **paramsSetterGetter.h**
- set_numBandsTx() : **paramsSetterGetter.h**
- set_numFbNCO() : **paramsSetterGetter.h**
- set_numRxNCO() : **paramsSetterGetter.h**
- set_numTxNCO() : **paramsSetterGetter.h**
- set_powerAttkEn() : **paramsSetterGetter.h**
- set_powerAttkThresh() : **paramsSetterGetter.h**

- set_powerDecEn() : **paramsSetterGetter.h**
- set_powerDecThresh() : **paramsSetterGetter.h**
- set_smallStepAttkEn() : **paramsSetterGetter.h**
- set_smallStepAttkThresh() : **paramsSetterGetter.h**
- set_smallStepDecEn() : **paramsSetterGetter.h**
- set_smallStepDecThresh() : **paramsSetterGetter.h**
- set_spiInUseForPllAccess() : **paramsSetterGetter.h**
- set_syncLoopBack() : **paramsSetterGetter.h**
- set_txToFbMode() : **paramsSetterGetter.h**
- set_useSpiSysref() : **paramsSetterGetter.h**
- set_X() : **paramsSetterGetter.h**
- setAfeLogLvl() : **baseFunc.c** , **baseFunc.h**
- setFbDsa() : **dsaAndNco.h** , **dsaAndNco.c**
- setFbDsaPerTx() : **dsaAndNco.h** , **dsaAndNco.c**
- setGoodRbd() : **jesd.h** , **jesd.c**
- setJesdRxSyncOverride() : **jesd.c** , **jesd.h**
- setJesdTxSyncOverride() : **jesd.h** , **jesd.c**
- setManualRbd() : **jesd.h** , **jesd.c**
- setPinRxDsaSettings() : **dsaAndNco.c** , **dsaAndNco.h**
- setRxDigGain() : **dsaAndNco.h** , **dsaAndNco.c**
- setRxDsa() : **dsaAndNco.h** , **dsaAndNco.c**
- setRxDsaMode() : **dsaAndNco.h** , **dsaAndNco.c**
- SetSerdesTxCursor() : **serDes.c** , **serDes.h**
- setTxDigGain() : **dsaAndNco.c** , **dsaAndNco.h**
- setTxDsa() : **dsaAndNco.h** , **dsaAndNco.c**
- splitToByte() : **hMacro.h** , **hMacro.c**

Generated by doxygen 1.8.17

- t -

- toggleSync() : **jesd.h** , **jesd.c**
- triggerMacro() : **hMacro.c** , **hMacro.h**
- txCalibSiggen() : **hMacro.h** , **hMacro.c**
- txDsaIdxGainSwap() : **dsaAndNco.c** , **dsaAndNco.h**

Generated by doxygen 1.8.17

- u -

- updateFbNco() : **dsaAndNco.h** , **dsaAndNco.c**
- updateRxNco() : **dsaAndNco.c** , **dsaAndNco.h**
- updateSystemTxChannelFreqConfig() : **hMacro.h** , **hMacro.c**
- updateTxGain() : **dsaAndNco.c** , **dsaAndNco.h**
- updateTxGainParam() : **dsaAndNco.h** , **dsaAndNco.c**
- updateTxNco() : **dsaAndNco.h** , **dsaAndNco.c**
- updateTxNcoDb() : **dsaAndNco.c** , **dsaAndNco.h**

Generated by doxygen 1.8.17

- w -

- wait() : **baseFunc.h** , **baseFunc.c**
- waitForMacroAck() : **hMacro.c** , **hMacro.h**
- waitForMacroDone() : **hMacro.h** , **hMacro.c**
- waitForMacroReady() : **hMacro.c** , **hMacro.h**
- waitMs() : **baseFunc.c** , **baseFunc.h**
- writeOperandList() : **hMacro.h** , **hMacro.c**

- systemParams : **afeParameters.h** , **afeParameters.c**

- PllSpiRegType_e : **basicFunctions.c**
- RetType_e : **afe79xxTypes.h**

- PLL_SPI_REG_TYPE : **basicFunctions.c**
- RET_TYPE : **afe79xxTypes.h**

- PLL_SPI_REG_A : **basicFunctions.c**
- PLL_SPI_REG_B : **basicFunctions.c**
- PLL_SPI_REG_OFF : **basicFunctions.c**
- PLL_SPI_REG_TYPE_SIZE : **basicFunctions.c**
- RET_EXEC_FAIL : **afe79xxTypes.h**
- RET_OK : **afe79xxTypes.h**

- a -

- AFE_AGC_MAX_ABS_NUM_HITS : **afe79xxTypes.h**
- AFE_AGC_MAX_WIN_LEN : **afe79xxTypes.h**
- AFE_FB_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_FB_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_FUNC_EXEC : **afeCommonMacros.h**
- AFE_ID_VALIDITY : **afeCommonMacros.h**
- AFE_LOG_LEVEL_DEBUG : **afe79xxLog.h**
- AFE_LOG_LEVEL_ERROR : **afe79xxLog.h**
- AFE_LOG_LEVEL_INFO : **afe79xxLog.h**
- AFE_LOG_LEVEL_SPILOG : **afe79xxLog.h**
- AFE_LOG_LEVEL_WARNING : **afe79xxLog.h**
- AFE_MACRO_DONE_POLL_FAIL : **afeCommonMacros.h**

- AFE_MACRO_ERROR_IN_EXECUTION : afe79xxTypes.h
- AFE_MACRO_ERROR_IN_OPCODE : afe79xxTypes.h
- AFE_MACRO_ERROR_IN_OPERAND : afe79xxTypes.h
- AFE_MACRO_ERROR_OPCODE_NOT_ALLOWED : afe79xxTypes.h
- AFE_MACRO_EXEC_ERROR : afeCommonMacros.h
- AFE_MACRO_EXTENDED_ERROR_CODE_REG_ADDR : afe79xxTypes.h
- AFE_MACRO_NO_ERROR : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_DET_TIME_CONST_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_DIG_DET_ABSOLUTE_NUM_CROSSINGS_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_DIG_DET_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_DIG_DET_RELATIVE_NUM_CROSSINGS_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_EXT_LNA_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_EXT_LNA_GAIN_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_GAIN_STEP_SIZE_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_RF_ANALOG_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_AGC_STATE_CONTROL : afe79xxTypes.h
- AFE_MACRO_OPCODE_ALC_CONFIGURATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_APPLY_DSA_GAIN_PHASE_COMPENSATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_COARSE_FINE_MODE_ALC : afe79xxTypes.h
- AFE_MACRO_OPCODE_CONFIG_SIGGEN_FOR_CAL : afe79xxTypes.h
- AFE_MACRO_OPCODE_EXT_AGC_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_FACTORY_RX_DSA_GAIN_PHASE_CALIBRATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_FACTORY_TX_DSA_GAIN_PHASE_CALIBRATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_FLOATING_POINT_CONFIG_ALC : afe79xxTypes.h
- AFE_MACRO_OPCODE_INT_AGC_CONTROLLER_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_MIN_MAX_DSA_ATTN_CONFIG : afe79xxTypes.h
- AFE_MACRO_OPCODE_PREPARE_FOR_TUNE : afe79xxTypes.h
- AFE_MACRO_OPCODE_REG_ADDR : afe79xxTypes.h
- AFE_MACRO_OPCODE_SYSTEM_TUNE : afe79xxTypes.h
- AFE_MACRO_OPCODE_SYSTEM_TUNE_SELECTIVE : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_FB_CHANNEL_FREQUENCY_CONFIGURATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_RX_CHANNEL_FREQUENCY_CONFIGURATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_SYSTEM_TX_CHANNEL_FREQUENCY_CONFIGURATION_ALL_BANDS : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_TX_DIG_PARAM : afe79xxTypes.h
- AFE_MACRO_OPCODE_UPDATE_TX_GAIN : afe79xxTypes.h
- AFE_MACRO_OPERAND_START_REG_ADDR : afe79xxTypes.h
- AFE_MACRO_PAGE_REG_ADDR : afe79xxTypes.h
- AFE_MACRO_PAGE_SEL_VAL : afe79xxTypes.h
- AFE_MACRO_READY_POLL_FAIL : afeCommonMacros.h
- AFE_MACRO_RESULT_START_REG_ADDR : afe79xxTypes.h
- AFE_MACRO_STATUS_REG_ADDR : afe79xxTypes.h
- AFE_NUM_BANDS_PER_FB : afe79xxTypes.h
- AFE_NUM_BANDS_PER_RX : afe79xxTypes.h
- AFE_NUM_BANDS_PER_TX : afe79xxTypes.h
- AFE_NUM_CH_PER_JESD_INSTANCE : afe79xxTypes.h
- AFE_NUM_FB_CHANNELS : afe79xxTypes.h
- AFE_NUM_FB_CHANNELS_BITWISE : afe79xxTypes.h
- AFE_NUM_JESD_INSTANCES : afe79xxTypes.h
- AFE_NUM_RX_CHANNELS : afe79xxTypes.h
- AFE_NUM_RX_CHANNELS_BITWISE : afe79xxTypes.h
- AFE_NUM_SERDES_LANES : afe79xxTypes.h
- AFE_NUM_TX_CHANNELS : afe79xxTypes.h

- AFE_NUM_TX_CHANNELS_BITWISE : **afe79xxTypes.h**
- AFE_PAGE_END_ADDR : **afe79xxTypes.h**
- AFE_PAGE_START_ADDR : **afe79xxTypes.h**
- AFE_PARAMS_VALID : **afeCommonMacros.h**
- AFE_REQ_SPI_ACCESS_MAX_COUNT : **basicFunctions.c**
- AFE_RX_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_RX_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_RX_DSA_MAX_DIG_DSA_INDEX : **afe79xxTypes.h**
- AFE_SPI_EXEC : **afeCommonMacros.h**
- AFE_TX_DSA_MAX_ANA_DSA_DB : **afe79xxTypes.h**
- AFE_TX_DSA_MAX_ANA_DSA_INDEX : **afe79xxTypes.h**
- AFE_TX_DSA_MAX_ANA_PLUS_DIG_DSA_DB : **afe79xxTypes.h**
- afeLogDbg : **afe79xxLog.h**
- afeLogErr : **afe79xxLog.h**
- afeLogInfo : **afe79xxLog.h**
- afeLogSpiLog : **afe79xxLog.h**
- ARRAY_SIZE : **afeCommonMacros.h**

- c -

- CFG_SPI_READ_POLL_MAX_COUNT : **basicFunctions.c**

- j -

- jesdToSerdesLaneMapping : **afe79xxTypes.h**

- m -

- MASK_BYTE : **basicFunctions.c**
- MASK_SHORT : **basicFunctions.c**

- n -

- NULL : **afe79xxTypes.h**
- NUM_OF_AFE : **afeCommonMacros.h**

Generated by doxygen 1.8.17