



165 Gibraltar Ct,
Sunnyvale, CA 94089

DFE Low Level Driver

Software Design Specification (SDS)

Revision A

<9/18/2013>

NOTICE OF CONFIDENTIAL AND PROPRIETARY INFORMATION

Information contained herein is subject to the terms of the Non-Disclosure Agreement between Texas Instruments Incorporated and your company, and is of a highly sensitive nature. It is confidential and proprietary to Texas Instruments Incorporated. It shall not be distributed, reproduced, or disclosed orally or in written form, in whole or in part, to any party other than the direct recipients without the express written consent of Texas Instruments Incorporated.

Revision Record	
Document Title: DFE LLD Software Design Specification	
Revision	Description of Change
A	Initial Release

Note: Be sure the Revision of this document matches the Approval record Revision letter. The revision letter increments only upon approval via the Quality Record System.

TABLE OF CONTENTS

1	SCOPE	1
2	REFERENCES.....	1
3	DEFINITIONS	1
4	OVERVIEW	2
5	DESIGN	3
5.1	OBJECTS AND LIMITATIONS	3
5.2	SOFTWARE RESOURCE REQUIREMENTS	4
5.3	INTERFACE DATA STRUCTURES	4
5.4	DFE PERIPHERAL CONFIGURATION	7
6	DFE LLD APIS	7
6.1	OPEN	7
6.2	CLOSE	8
6.3	LOAD TARGET CONFIGURATION	9
6.4	SOFT RESET	9
6.5	INITIALIZATION SEQUENCE FOR TRANSMIT PATH	10
6.6	INITIALIZATION SEQUENCE FOR RECEIVE PATH	11
6.7	GET DEVICE INFORMATION.....	12
6.8	ISSUE SYNC	13
6.9	GET SYNC STATUS	14
6.10	PROGRAM SYNC COUNTER.....	15
6.11	ISSUE SYNC START SYNC COUNTER.....	16
6.12	PROGRAM BBTX GAIN	16
6.13	ISSUE SYNC UPDATE BBTX GAIN.....	17
6.14	GET BBTX GAIN UPDATE COMPLETE.....	18
6.15	PROGRAM BBTX POWER METER.....	19
6.16	ISSUE SYNC UPDATE BBTX POWER METER	20
6.17	CLEAR BBTX POWER METER DONE STATUS.....	21
6.18	GET BBTX POWER METER DONE STATUS.....	21
6.19	READ BBTX POWER METER.....	22
6.20	OPEN BBTX POWER METER DMA	23
6.21	CLOSE BBTX POWER METER DMA	24
6.22	ENABLE BBTX POWER METER DMA.....	25
6.23	DISABLE BBTX POWER METER DMA	25
6.24	PROGRAM BBRX GAIN.....	26
6.25	ISSUE SYNC UPDATE BBRX GAIN	27
6.26	GET BBRX GAIN UPDATE COMPLETE	28
6.27	PROGRAM BBRX POWER METER.....	28
6.28	ISSUE SYNC UPDATE BBRX POWER METER	30
6.29	CLEAR BBRX POWER METER DONE STATUS	30
6.30	GET BBRX POWER METER DONE STATUS.....	31
6.31	READ BBRX POWER METER.....	32
6.32	OPEN BBRX POWER METER DMA	32
6.33	CLOSE BBRX POWER METER DMA	34
6.34	ENABLE BBRX POWER METER DMA.....	34
6.35	DISABLE BBRX POWER METER DMA.....	35
6.36	ENABLE DISABLE BB AID LOOPBACK	36
6.37	PROGRAM BB BUF LOOPBACK.....	37
6.38	SET BB AID UL STROBE DELAY	38

6.39	PROGRAM BB SIGGEN RAMP	38
6.40	ISSUE SYNC UPDATE BB SIGGEN	39
6.41	PROGRAM BB TESTBUS	40
6.42	PROGRAM DDUC MIXER NCO FREQUENCY	41
6.43	ISSUE SYNC UPDATE DDUC MIXER NCO FREQUENCY	41
6.44	PROGRAM DDUC MIXER PHASE	42
6.45	ISSUE SYNC UPDATE DDUC MIXER PHASE	43
6.46	PROGRAM DDUC FARROW PHASE	44
6.47	ISSUE SYNC UPDATE DDUC FARROW PHASE	44
6.48	PROGRAM DISTRIBUTOR MAP	45
6.49	ISSUE SYNC UPDATE DISTRIBUTOR MAP	46
6.50	PROGRAM SUMMER SHIFT	47
6.51	PROGRAM SUMMER MAP	47
6.52	ISSUE SYNC UPDATE SUMMER MAP	48
6.53	PROGRAM CFR COEFFICIENTS	49
6.54	ISSUE SYNC UPDATE CFR COEFFICIENTS	50
6.55	PROGRAM CFR PREGAIN	50
6.56	ISSUE SYNC UPDATE CFR PREGAIN	51
6.57	PROGRAM CFR POSTGAIN	52
6.58	ISSUE SYNC UPDATE CFR POSTGAIN	53
6.59	PROGRAM CFR PROTECTION GAIN	53
6.60	PROGRAM TX MIXER	54
6.61	ISSUE SYNC UPDATE TX MIXER	55
6.62	PROGRAM TX PA PROTECTION	56
6.63	GET TX PA PROTECTION INTERRUPT STATUS	57
6.64	CLEAR TX PA PROTECTION INTERRUPT STATUS	58
6.65	READ TX PA PROTECTION POWER STATUS	59
6.66	PROGRAM JESD TX TO LANE MAP	60
6.67	PROGRAM JESD Tx SigGEN RAMP	61
6.68	ISSUE SYNC UPDATE JESD TX SIGGEN	62
6.69	PROGRAM JESD TX TESTBUS	63
6.70	GET JESD TX LINK STATUS	63
6.71	GET JESD TX LANE STATUS	64
6.72	CLEAR JESD TX LINK ERRORS	65
6.73	CLEAR JESD TX LANE ERRORS	66
6.74	PROGRAM JESD LANE TO RX MAP	67
6.75	PROGRAM JESD LOOPBACK	68
6.76	PROGRAM JESD RX TESTBUS	69
6.77	GET JESD RX LINK STATUS	69
6.78	GET JESD RX LANE STATUS	70
6.79	CLEAR JESD RX LINK ERROR	72
6.80	CLEAR JESD RX LANE ERROR	72
6.81	PROGRAM RX IBPM GLOBAL	73
6.82	PROGRAM RX IBPM	74
6.83	ISSUE SYNC UPDATE RX IBPM	75
6.84	ISSUE RX IBPM READ REQUEST	76
6.85	GET RX IBPM READ ACK	77
6.86	READ RX IBPM RESULT	78
6.87	PROGRAM RX EQUALIZER	78
6.88	ISSUE SYNC UPDATE RX EQUALIZER	80
6.89	PROGRAM RX MIXER NCO	80
6.90	ISSUE SYNC UPDATE RX MIXER NCO FREQUENCY	81
6.91	PROGRAM RX TESTBUS	82
6.92	PROGRAM FB EQUALIZER	82
6.93	ISSUE SYNC UPDATE FB EQUALIZER	83

6.94	PROGRAM FB MIXER NCO	84
6.95	ISSUE SYNC UPDATE FB MIXER NCO.....	85
6.96	PROGRAM FB IO MUX	86
6.97	PROGRAM FB PRE-CB GAIN.....	86
6.98	ISSUE SYNC UPDATE FB PRE-CB GAIN.....	87
6.99	PROGRAM CB NODE CONFIG	88
6.100	PROGRAM CB BUF CONFIG	89
6.101	ARM CB AND ISSUE SYNC.....	90
6.102	GET CB DONE STATUS.....	90
6.103	READ CB BUF	91
6.104	OPEN CB BUF DMA	92
6.105	CLOSE CB BUF DMA.....	93
6.106	ENABLE CB BUF DMA	94
6.107	DISABLE CB BUF DMA	94
6.108	DISABLE ALL TESTBUS.....	95
6.109	PROGRAM DFE GPIO PINMUX.....	95
6.110	SET DFE GPIO SYNC OUT SOURCE.....	97
6.111	SET DFE GPIO BANK OUTPUT	97
6.112	GET DFE GPIO BANK INPUT	98
6.113	OPEN GENERIC IO DMA	99
6.114	CLOSE GENERIC IO DMA	100
6.115	PREPARE GENERIC DMA EMBEDDED HEADER	100
6.116	ENABLE LUT TOGGLE	102
6.117	SETSYNCSEL FOR LUT	102
6.118	ISSUE SYNC UPDATE LUT.....	103
6.119	GET CURRENT LUT MEMORY INDEX.....	104
6.120	PROGRAM LUT TABLE	104
6.121	GET DPD CONFIGURATION.....	105
6.122	LOAD DPDA IMAGE.....	106
6.123	READ DPDA IG REGISTER	107
6.124	READ DPDA PARAMETERS	108
6.125	READ DPDA SCALAR REGISTER.....	108
6.126	START DPDA.....	109
6.127	WRITE DPDA IG REGISTER.....	110
6.128	WRITE DPDA SAMPLES	110
6.129	WRITE DPDA SCALAR REGISTER	111
7	INTEGRATION.....	112
7.1	OSAL	112
7.1.1	Logging API.....	112
7.2	INTEGRATION ON ARM LINUX.....	112
7.3	INTEGRATION ON DSP SYMBIOS.....	113
8	FUTURE EXTENSIONS.....	113

1 Scope

This document describes the functionality, architecture, and operation of the Digital Radio Front-End (DFE) Low Level Driver (LLD).

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	IQN2 User Guide	Version x.x.x	IQN2 User Guide
2	CPPI User Guide	Version x.x.x	CPPI User Guide
3	DEF CSL API Document	Version x.x.x	DOXYGEN generated API documentation located in the package under the “docs” directory in CHM format.

Table 1. Referenced Materials

3 Definitions

Acronym	Description
DFE	Digital radio Front-End
API	Application Programming Interface
IQN2	IQNet2
CPPI	Communication Port Programming Interface
LLD	Low Level Driver
CTL	IQN2 ConTroL channel, for shipping non-iq data packets
CPP/DMA	CTL Packet Process, a DMA engine in DFE
CTL	Control data, non i/q stream
CB	Capture Buffer in DFE
BB	Baseband
DDUC	Digital Down/Up Convertor
SUM	Summer, map DDUC Tx Channels to a Tx stream
CFR	Crest Factor Reduction
DPD	Digital Pre-Distortion
JESD	

Acronym	Description
FB	Feedback path in DFE
DIST	Distributor, map DDUC Rx channels to a Rx stream
TX	Transmit, an IP block between DPD and JESD TX
RX	Receive, an IP block between DDUC and JESDRX
NCO	Numeric Controlled Oscillator

Table 2. Definitions

4 Overview

DFE is a high performance wideband digital IF transmit and receive signal processing peripheral for small cell base station applications. It implements advanced algorithms for RF power amplifier linearization including crest factor reduction (CFR) and digital pre-distortion (DPD), and for correcting other receiver RF impairments like IQ imbalance, DC offset and distortion.

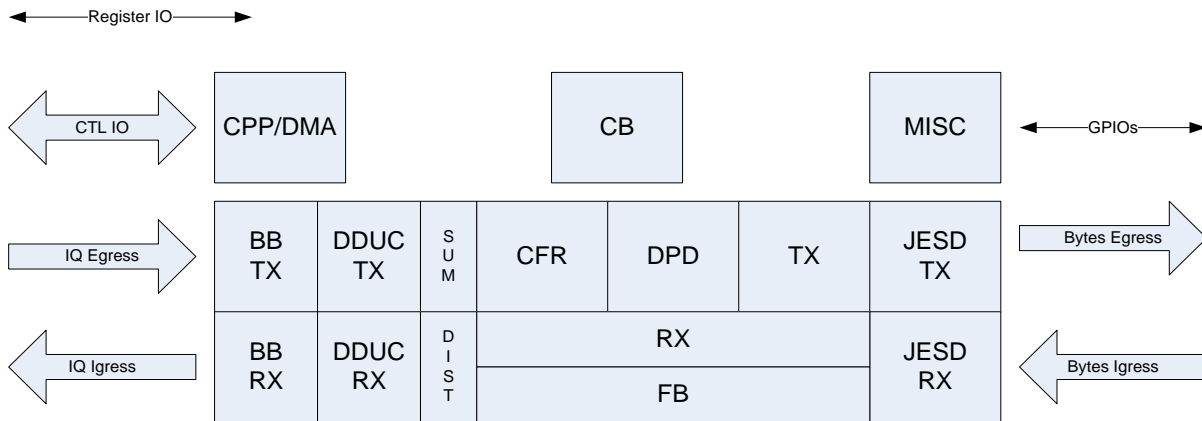


Figure 1, DFE Diagram

In signal processing Tx path,

- BB TX has gain adjustment and power meter per carrier;
- DDUC TX setups carrier mixer frequency and phase, PFIR coefficients, Farrow channel phase;
- SUM maps DDUC Tx channel to Tx stream and then adjusts stream gain by shift control;
- CFR has preCFR gain, postCFR gain adjustments, CFR filter coefficients update;
- DPD LUTs are updated after software complete iteration;
- TX has a PA protection block which signals CFR to reduce gain when abnormal peak and/or power detected;
- JESD TX reports link/lane status for transmit connection.

In signal processing Rx path,

- JESD TX reports link/lane status for receiving connection, and controls JESD lane loopback;
- RX has programmable equalizer, NCO mixer;
- FB has programmable equalizer, NCO mixer, IO Mux, and gain adjustment before CB;
- DIST maps RX stream to DDUC Rx channel;
- BB RX has gain adjustment and power meter per carrier;

SoC reads/writes DFE via memory mapped registers. DFE has a built-in DMA engine, CPP/DMA, which allows big chunks of data move in/out very efficiently via PktDMA. Software should use CPP/DMA as much as possible.

CB can capture up to 32K complex samples at hardware nodes and test-bus positions in signal path. DPD adaptation uses CB to get reference and feedback buffers. CB is also a great tool to peek what is going on in signal path.

MISC block controls feedback switch GPIOs and top level interrupts aggregation.

5 Design

5.1 Objects and Limitations

DFE LLD just supports open single device instance for each DFE peripheral. The application software should take care of necessary serialization or arbitration in multi-cores multi-threads operation system. In Linux, the LLD is running in user mode.

DFE LLD just configures, controls, writes, reads DFE block. Any other block accessing is out of scope. IQN2 is the bridge between DFE and other parts of SoC; PktDMA is the preferred way to move data bulks into/out of DFE.

- For using IQN2, please refer to IQN2 User Guide
- For using PktDMA, please refer to CPPI User Guide

DFE LLD API should run to completion without any waiting loop longer than 1ms, especially no polling internally for any event, such as a sync event, a done interrupt event. One solution is to break the operation to several APIs, and let the calling thread do the waiting part. The example is capture buffer operation, whose pseudo calling sequence in high layer is like,

```
GetCB()
{
    /* setup CB Nodes and Buffers */
    Setup();
    /* arm CB block with selected sync */
    ArmCb();
    /* wait for CB done */
    While(get_done_status() == FALSE) { /* spin wait */ }
    /* read data back */
    Read_data();
}
```


Another solution is using callback function when there need a waiting function. The example is DFE initialization sequence, which has many steps need wait for sync counter sync events.

To be flexible, DFE LLD doesn't maintain antenna mapping, which should be taken care by high level software. On the other hand, it supports conversion between friendly arguments and hardware recognized values. For example, signal power reading API returns peak and rms power in floating number dB, rather than raw register values.

DFE LLD doesn't have built-in interrupt service routines. To check if an interrupt event has come, high level software has to use polling method.

5.2 Software Resource Requirements

In each direction, IQN2 has total 48 channels connect to DFE. Any channel can map to either BB(IQ) or CPP/DMA(CTL). In practice, first 32 ones are recommended mapping to BB, next 16 ones mapping to CPP/DMA. Enable all 16 CTL channels are highly recommended when configure IQN2.

Lamarr IQN2 Egress Queue	Recommend Map
832 ~ 863	Transferring IQ packets to DFE BBTX
864 ~ 879	Transferring CTL packets to DFE CPP/DMA

DFE LLD does not maintain any global object and context.

5.3 Interface Data Structures

All LLD APIs are running within the device instance context DFE_Obj, which is user allocated but initialized in DFE_open().

```
typedef struct _DFE_Obj
{
    // DFE CSL object
    DfeFl_Obj      objDfe;
    // DFE_BB CSL object
    DfeFl_BbObj    objDfeBb[DFE_FL_BB_PER_CNT];
    // DFE_DUC CSL object
    DfeFl_DducObj  objDfeDduc[DFE_FL_DUC_PER_CNT];
    // DFE_SUMMER CSL object
    DfeFl_SummerObj objDfeSummer[DFE_FL_SUMMER_PER_CNT];
    // DFE_AUTOCP CSL object
    DfeFl_AutocpObj objDfeAutocp[DFE_FL_AUTOCP_PER_CNT];
    // DFE_CFR CSL object
    DfeFl_CfrObj    objDfeCfr[DFE_FL_CFR_PER_CNT];
    // DFE_CDFR CSL object
    DfeFl_CdfrObj   objDfeCdfr[DFE_FL_CDFR_PER_CNT];
    // DFE_DPD CSL object
    DfeFl_DpdObj    objDfeDpd[DFE_FL_DPD_PER_CNT];
    // DFE_DPDA CSL object
    DfeFl_DpdaObj   objDfeDpda[DFE_FL_DPDA_PER_CNT];
    // DFE_TX CSL object
    DfeFl_TxObj     objDfeTx[DFE_FL_TX_PER_CNT];
    // DFE_RX CSL object
```

```

DfeFl_RxObj objDfeRx[DFE_FL_RX_PER_CNT];
// DFE_CB CSL object
DfeFl_CbObj objDfeCb[DFE_FL_CB_PER_CNT];
// DFE_JESD CSL object
DfeFl_JesdObj objDfeJesd[DFE_FL_JESD_PER_CNT];
// DFE_FB CSL object
DfeFl_FbObj objDfeFb[DFE_FL_FB_PER_CNT];
// DFE_MISC CSL object
DfeFl_MiscObj objDfeMisc[DFE_FL_MISC_PER_CNT];

// DFE CSL context
DfeFl_Context dfeCtx;
// DFE CSL param
DfeFl_Param dfeParam;

// DFE CSL Handle
DfeFl_Handle hDfe;
// DFE_BB CSL Handle
DfeFl_BbHandle hDfeBb[DFE_FL_BB_PER_CNT];
// DFE_DDUC CSL Handle
DfeFl_DducHandle hDfeDduc[DFE_FL_DDUC_PER_CNT];
// DFE_SUMMER CSL Handle
DfeFl_SummerHandle hDfeSummer[DFE_FL_SUMMER_PER_CNT];
// DFE_AUTOCP CSL Handle
DfeFl_AutocpHandle hDfeAutocp[DFE_FL_AUTOCP_PER_CNT];
// DFE_CFR CSL Handle
DfeFl_CfrHandle hDfeCfr[DFE_FL_CFR_PER_CNT];
// DFE_CDFR CSL Handle
DfeFl_CdfrHandle hDfeCdfr[DFE_FL_CDFR_PER_CNT];
// DFE_DPD CSL Handle
DfeFl_DpdHandle hDfeDpd[DFE_FL_DPD_PER_CNT];
// DFE_DPDA CSL Handle
DfeFl_DpdaHandle hDfeDpda[DFE_FL_DPDA_PER_CNT];
// DFE_TX CSL Handle
DfeFl_TxHandle hDfeTx[DFE_FL_TX_PER_CNT];
// DFE_RX CSL Handle
DfeFl_RxHandle hDfeRx[DFE_FL_RX_PER_CNT];
// DFE_CB CSL Handle
DfeFl_CbHandle hDfeCb[DFE_FL_CB_PER_CNT];
// DFE_JESD CSL Handle
DfeFl_JesdHandle hDfeJesd[DFE_FL_JESD_PER_CNT];
// DFE_FB CSL Handle
DfeFl_FbHandle hDfeFb[DFE_FL_FB_PER_CNT];
// DFE_MISC CSL Handle
DfeFl_MiscHandle hDfeMisc[DFE_FL_MISC_PER_CNT];

// DFE_CPP CSL resource manager
DfeFl_CppResMgr cppResMgr;

// BBTX power meter for CPP/DMA
uint32_t bbtXPowmtr;
// dma handle for BBTX power meter
DfeFl_CppDmaHandle hDmaBbtXPowmtr;
// descriptor handle for BBTX power meter
DfeFl_CppDescriptorHandle hDescripBbtXPowmtr;
// IQN2 CTL Ingress Channel for BBTX power meter

```

```
uint32_t  bbtXPowmtrIqnChnl;

// BBRX power meter for CPP/DMA
uint32_t  bbrXPowmtr;
// dma handle for BBRX power meter
DfeFl_CppDmaHandle hDmaBbrXPowmtr;
// descriptor handle for BBRX power meter
DfeFl_CppDescriptorHandle hDescripBbrXPowmtr;
// descriptor handle for BBTX power meter
uint32_t  bbrXPowmtrIqnChnl;

// flag to read all 18 bit Cb
uint32_t  flag_18bit;
// dma handle for cb
DfeFl_CppDmaHandle hDmaCb;
// descriptor handle for cb
DfeFl_CppDescriptorHandle hDescripCb[8];
// IQN2 CTL Channel
uint32_t  cbIqnChnl;

// sync counter ssel
DfeFl_MiscSyncGenSig sync_cnter_ssel;
// bbtX siggen ssel
uint32_t  bbtX_siggen_ssel;
// bbrX checksum ssel
uint32_t  bbrX_chksum_ssel;
// ulStrobe strobe
DfeFl_MiscSyncGenSig ulStrobe_Sync;

// RX IBPM Unity Magnitude Sqaure value (=I^2 + Q^2)
uint64_t  rxIbpmUnityMagsq;

// generic DMA hndle
DfeFl_CppDmaHandle hDmaGeneric;
// IQN2 CTL Egress channel for generic writing DMA
uint32_t  genericDmaIqnChnlDl;
// IQN2 CTL Ingress channel for generic reading DMA
uint32_t  genericDmaIqnChnlUl;
} DFE_Obj;
typedef DFE_Obj * DFE_Handle;
```

User also defines CPP/DMA reserved channels and reserved descriptors in the resource management table, which will be saved to the device context in Dfe_open(). Reserved channel/descriptor can only be opened using explicit Id of the resource; non-reserved resource can be opened using DFE_FL_CPP_OPEN_ANY.

```
/** CPP resource manager */
typedef struct _DFE_CppResTbl
{
    // reserved dma bitmask
    Uint32 dmaRsvd;

    // discrete trigger out
    Uint32 discreteTrig[DFE_FL_CPP_NUM_DISCRETE_TRIGGERS];

    // four 32-bits words, each bit corresponding to one descriptor
```

```
// reserved descriptor bitmask
Uint32 descripRsvd[4];

} DFE_CppResTbl;
```

5.4 DFE Peripheral Configuration

To properly configure DFE, the following sequence must be followed,

Step	Description	DFE LLD API
1	Program DFE PLL to correct operation rate	
2	Power up IQN2 and DFE power domains	
3	Program SERDES to corresponding rate	
4	Program analogue front-end which connects with DFE	
5	Open DFE LLD	Dfe_open()
5	Load DFE target configuration	Dfe_loadTgtCfg()
	Soft Reset DFE peripheral	Dfe_softReset()
	Check and wait SERDES PLL_OK	
6	Do DFE initialization sequence for transmit path	Dfe_initTgtTx()
	Check and wait SERDES OK and !LOSS	
	Do DFE initialization sequence for receive path	Dfe_initTgtRx()
7	QMSS, CPPI configuration	
8	IQN2 configuration and enable	

6 DFE LLD APIs

6.1 Open

Open DFE LLD device.

Prototype

```
DFE_Handle Dfe_open
(
    int dfeInst,
    DFE_Obj *dfeObj,
    DFE_CppResTbl *dfeResTbl,
    DFE_Err *err
);
```

Description

Open DFE low level driver device instance. Before call the API, the caller should allocate the memory buffer for DFE_Obj, which will be the context for the life of DFE LLD. The buffer shall not be allocated from stack.

Upon Dfe_open() succeeds, the error code is set to DFE_ERR_NONE.

- DFE LLD object dfeObj has been initialized.
- User defined resource management table has been saved to instance

<p>context.</p> <ul style="list-style-type: none"> • A valid DFE_Handle value is returned. <p>When the error code is DFE_ERR_INVALID_DEVICE, the LLD doesn't find the specified dfeInst device. The API should only be called once.</p>
--

Arguments

dfeInst	[in] DFE device instance number
dfeObj	[out] DFE LLD device context
dfeResTbl	[in] DFE resource management table
err	[out] exit error code

Return Value

A valid DFE_Handle value is returned when the call succeeds; otherwise a NULL is returned.
--

Constrains

<ol style="list-style-type: none"> 1) dfeObj buffer shall not be allocated in stack. 2) Dfe open() should be called only once.
--

6.2 Close

Close DFE LLD device opened by Dfe_open().

Prototype

<pre>DFE_Err Dfe_close (DFE_Handle hDfe);</pre>

Description

Close DFE low level driver device instance, which was opened by Dfe_open().

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

<p>DFE_ERR_NONE, if close properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL</p>
--

Constrains

<ol style="list-style-type: none"> 1) hDfe should be a valid handle opened by Dfe_open().
--

6.3 Load Target Configuration

Load DFE target configuration, i.e. registers contents.

Prototype

```
DFE_Err Dfe_loadTgtCfg
(
    DFE_Handle hDfe,
    DFE_RegPair tgtCfgPairs[]
);
```

Description

Write target configuration i.e. registers contents, to DFE hardware. The target config format is defined as,

```
// (addr, data) register pair
typedef struct
{
    // offset address from DFE base address
    Uint32 addr;
    // data to/from addr
    Uint32 data;
} DFE_RegPair;
```

The last entity of tgtCfgPairs must be (0xfffffffffu, 0xfffffffffu), which is the end marker.

Arguments

hDfe	[in] DFE device handle
tgtCfgPairs	[in] pointer to (addr, adta) pairs buffer

Return Value

DFE_ERR_NONE, if write to DFE properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) The last entity of tgtCfgPairs must be (0xfffffffffu, 0xfffffffffu)
- 3) DFE PLL and PSCs shall be already up running.

6.4 Soft Reset

Do soft reset for DFE peripheral.

Prototype

```
DFE_Err Dfe_softReset
(
    DFE_Handle hDfe
);
```

Description

The API does a software reset for DFE peripheral. Like power up reset, soft reset does assert `init_clk_gate`, `init_state`, and `clear_data` to all sub-blocks in DFE. On the other hand unlike power up reset, soft reset keeps all registers values unchanged, except for `init` register in each block.

Soft reset should be done after `Dfe_loadTgtCfg()` but before `Dfe_initTgtTx()`, and `Dfe_initTgtRx()`.

Arguments

<code>hDfe</code>	[in] DFE device handle
-------------------	------------------------

Return Value

`DFE_ERR_NONE`, if soft reset properly
`DFE_ERR_INVALID_HANDLE`, if `hDfe` is `NULL`
`DFE_ERR_HW_CTRL`, if `CSL HwControl()` failed

Constraints

- 1) `hDfe` should be a valid handle opened by `Dfe_open()`.
- 2) DFE PLL and PSCs shall be already up running.
- 3) `Dfe_loadTgtCfg()` already called.

6.5 Initialization Sequence for Transmit Path

DFE initialization sequence for transmit path.

Prototype

```
DFE_Err Dfe_initTgtTx
(
    DFE_Handle hDfe,
    DFE_CallbackContext waitSyncCtx,
    DFE_CallbackWaitSync waitSyncFxn
);
```

Description

The API runs initialization sequence for transmit path.

When JESD Tx lanes configured connecting to SERDES, this initialization shouldn't be made until SERDES Tx lock (`pll_ok`).

In the sequence, there're many steps need wait a sync signal. This is done by callback `waitSyncFxn()`.

```
// callback function for waiting sync signal
// return DFE_ERR_OK if sync signal come properly
// return DFE_ERR_TIMEDOUT if sync signal not come before timed out
typedef DFE_Err (*DFE_CallbackWaitSync)
(
    // callback context
    DFE_CallbackContext cbkCtx,
```

```
// DFE device handle
DFE_Handle hDfe,
// sync signal to be waiting
DfeFl_MiscSyncGenSig syncSig;
);
```

When complete with DFE_ERR_NONE, DFE transmit path is ready working, otherwise DFE peripheral is in unknown condition.

Arguments

hDfe	[in] DFE device handle
waitSyncCtx	[in] callback context pass to waitSyncFxn
waitSyncFxn	[in] callback function for waiting a sync signal

Return Value

DFE_ERR_NONE, if initialization properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_SYNC_NOT_COME, if sync signal not come
 DFE_ERR_CALLBACKFXN_IS_NULL, if callback function required but given a NULL pointer
 DFE_ERR_HW_CTRL, if CSL HwControl() failed
 DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) Dfe_softReset() already called.

6.6 Initialization Sequence for Receive Path

DFE initialization sequence for receive path.

Prototype

```
DFE_Err Dfe_initTgtRx
(
    DFE_Handle hDfe,
    DFE_CallbackContext waitSyncCtx,
    DFE_CallbackWaitSync waitSyncFxn
);
```

Description

The API runs initialization sequence for receive path.

When JESD Rx lanes configured connecting to SERDES, this initialization shouldn't be made until SERDES Rx OK bit set and LOSS bit cleared.

In the sequence, there're many steps need wait a sync signal. This is done by callback waitSyncFxn().

```
// callback function for waiting sync signal
// return DFE_ERR_OK if sync signal come properly
```



```
// return DFE_ERR_TIMEDOUT if sync signal not come before timed out
typedef DFE_Err (*DFE_CallbackWaitSync)
(
    // callback context
    DFE_CallbackContext cbkCtx,
    // DFE device handle
    DFE_Handle hDfe,
    // sync signal to be waiting
    DfeFl_MiscSyncGenSig syncSig;
);
```

When complete with DFE_ERR_NONE, DFE receive path is ready working, otherwise DFE peripheral is in unknown condition.

Arguments

hDfe	[in] DFE device handle
waitSyncCtx	[in] callback context pass to waitSyncFxn
waitSyncFxn	[in] callback function for waiting a sync signal

Return Value

DFE_ERR_NONE, if initialization properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_SYNC_NOT_COME, if sync signal not come
 DFE_ERR_CALLBACKFXN_IS_NULL, if callback function required but given a NULL pointer
 DFE_ERR_HW_CTRL, if CSL HwControl() failed
 DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) Dfe_initTgtTx() already called.

6.7 Get Device Information

Get back DFE device information, such as PID, base address etc.

Prototype

```
DFE_Err Dfe_getDevInfo
(
    DFE_Handle hDfe,
    DFE_DevInfo *devInfo
);
```

Description

Get back DFE device information, such as PID, base address etc.

Write target configuration i.e. registers contents, to DFE hardware.
 The target config format is defined as,
 // DFE Device information

```
typedef struct
{
    // DFE peripheral ID
    Uint32 pid;
    // DFE peripheral base address
    void *baseAddr;
    // DFE LLD version
    Uint32 version;
} DFE_DevInfo;
```

Arguments

hDfe	[in] DFE device handle
devInfo	[out] pointer to devInfo buffer

Return Value

```
DFE_ERR_NONE, if get info properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if devInfo is NULL
```

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.

6.8 Issue Sync

Issue a sync signal.

Prototype

```
DFE_Err Dfe_issueSync
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig syncSig,
    Uint32 waitCnt
);
```

Description

Issue a sync signal.

- If waitCnt is DFE_FL_MISC_SYNC_NOWAIT, the function just does issue the sync and returns DFE_ERR_NONE immediately. Dfe_getSyncStatus() can be called later to check if the sync has come.
- If waitCnt is DFE_FL_MISC_SYNC_WAITFOREVER, the function waits until the signal has really come. And then returns DFE_ERR_NONE.
- If waitCnt is DFE_FL_MISC_SYNC_WAIT(n), 0 < n < 0xffffffffu, the function waits until
 - 1) either the signal has come before n loops complete, returns DFE_ERR_NONE;
 - 2) or n loops complete but the signal not come, returns DFE_ERR_SYNC_NOT_COME

For performance and flexible considerations, using DFE_FL_MISC_SYNC_NOWAIT for waitCnt is recommended.

Arguments

hDfe	[in] DFE device handle
syncSig	[in] sync signal to be issued
waitCnt	[in] wait loop count

Return Value

DFE_ERR_NONE, if issue sync done properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_SYNC_NOT_COME, if sync signal not coming within waitCnt loops.
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.9 Get Sync Status

Get a sync signal status.

Prototype

```
DFE_Err Dfe_getSyncStatus
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig syncSig,
    Uint32 *signaled
);
```

Description

Get status of a sync signal. When return, if *signaled is 1, the sync has come; if *signaled is 0, the sync hasn't come.

Arguments

hDfe	[in] DFE device handle
syncSig	[in] sync signal to be issued
signalled	[out] pointer to signal status buffer

Return Value

DFE_ERR_NONE, if issue sync done properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed

Constrains

- 4) hDfe should be a valid handle opened by Dfe_open().
- 5) DFE PLL and PSCs shall be already up running.
- 6) DFE has loaded target config and completed initialize sequence.

6.10 Program Sync Counter

Program a sync counter.

Prototype

```
DFE_Err Dfe_progSyncCounter
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenCntr cntr,
    Uint32 delay,
    Uint32 period,
    Uint32 pulseWidth,
    Uint32 repeat,
    Uint32 invert
);
```

Description

The API first resets the counter and then reprogram to specified parameters.
NOTE, Dfe_issueSyncStartSyncCounter() should be then called to start the counter.

Arguments

hDfe	[in] DFE device handle
cntr	[in] sync counter number
delay	[in] number of clocks to wait after sync select source before sending initial sync. If set to 0, sync counter output will be high if sync select source is high
period	[in] number of clocks to wait between syncs when repeat is 1. Does nothing when repeat is 0.
pulseWidth	[in] set to X for pulse width of X clocks; 0 means it will never go high.
repeat	[in] If 0, counter counts down delay clocks once, sends a sync, and stops. If 1, it counts down delay clocks sends a sync, then continuously sends more syncs every period clocks.
invert	[in] set to 1 to invert entire bus

Return Value

DFE_ERR_NONE, if sync counter programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncStartSyncCounter() should be called later to issue sync to start the counter.

6.11 Issue Sync Start Sync Counter

Issue sync to start the sync counter, and return without waiting.

Prototype

```
DFE_Err Dfe_issueSyncStartSyncCounter
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenCntr cntr,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to start the sync counter that has been already programmed by Dfe_progSyncCounter(). It programs the counter's starting sync select with ssel (using ALWAYS sync signal) and returns immediately after issue the sync. So Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
cntr	[in] sync counter number
ssel	[in] sync select to re-start sync counter

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.12 Program BBTX Gain

Program BBTX AxCs with new gains.

Prototype

```
DFE_Err Dfe_progBbtGain
```

```
(
    DFE_Handle hDfe,
    Uint32 numAxCs,
    Uint32 axc[],
    float gain[]
);
```

Description

Write new BBTX AxCs' gains to shadow memory. The range for a gain is -84dB ~ +6dB. The API changes gain's real part only.

NOTE, Dfe_issueSyncUpdateBbtXGain should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
numAxCs	[in] number of AxCs whose gains are to be changed
axc	[in] array of AxCs whose gains are to be changed
gain	[in] array of new real gains, in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateBbtXGain () should be called later to copy gains to working memory.

6.13 Issue Sync Update BBTX Gain

Issue sync update BBTX gain to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateBbtXGain
(
    DFE_Handle hDfe,
    Uint32 ct,
    DfeFl_MiscSyncGenSig ssl
);
```

Description

Issue sync to copy BBTX gains from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

This API also clears update complete interrupt status bit of the

corresponding carrier type.

To check if update complete, call `Dfe_getBbtxGainUpdateComplete()`.

NOTE, Both `axc_valid` bit and `gain_en` bit (in register `dfe.bb.bbtixif_axc_config0`) have to be set '1', in order to see effectiveness of the gains.

Arguments

<code>hDfe</code>	[in] DFE device handle
<code>ct</code>	[in] carrier type
<code>ssel</code>	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if `hDfe` is NULL
DFE_ERR_HW_CTRL, if CSL `HwControl()` failed

Constraints

- 1) `hDfe` should be a valid handle opened by `Dfe_open()`.
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.14 Get BBTX Gain Update Complete

Get BBTX gain update complete status.

Prototype

```
DFE_Err Dfe_getBbtxGainUpdateComplete
(
    DFE_Handle hDfe,
    Uint32 ct,
    Uint32 *complete
);
```

Description

Get BBTX gain update complete status. The API first reads `txgain_update_status`, if corresponding `ct` bit is clear, then the update is still in progress. Otherwise, it further reads back and returns the update complete interrupt status.

Arguments

<code>hDfe</code>	[in] DFE device handle
<code>ct</code>	[in] carrier type
<code>complete</code>	[out] buffer of update complete status 0 = still in progress 1 = update complete

Return Value

```
DFE_ERR_NONE, if API complete properly  
DFE_ERR_INVALID_HANDLE, if hDfe is NULL  
DFE_ERR_HW_QUERY, if CSL HwGetStatus() failed
```

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.15 Program BBTX Power Meter

Program BBTX power meter with new configuration.

Prototype

```
DFE_Err Dfe_progBbtXPowmtr  
(  
    DFE_Handle hDfe,  
    Uint32 pmId,  
    DFE_BbtXPowmtrConfig *mtrCfg  
);
```

Description

Write new BBTX power meter configuration.

```
// BBTX power meter config  
typedef struct  
{  
    // enable power meter function  
    DfeFl_BbPowMtrEnable enable;  
    // carrier type  
    Uint32 countSource;  
    // power meter input source  
    DfeFl_BbPowMtrInSource inSource;  
    // tdd mode  
    DfeFl_BbPowMtrTddMode tddMode;  
    // delay from sync  
    Uint32 syncDly;  
    // meter interval  
    Uint32 interval;  
    // integration period  
    Uint32 intgPd;  
    // count of measurements, i.e. count of intervals  
    Uint32 powUptIntvl;  
} DFE_BbtXPowmtrConfig;
```

NOTE, Dfe_issueSyncUpdateBbtXPowmtr should be called later to let hardware take the configuration.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBTX power meter Id, 0 ~ 15
mtrCfg	[in] new meter configuration

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateBbtPwmtr() should be called later to let hardware take new configuration.

6.16 Issue Sync Update BBTX Power Meter

Issue sync update BBTX power meter to new configuration.

Prototype

```
DFE_Err Dfe_issueSyncUpdateBbtPwmtr
(
    DFE_Handle hDfe,
    Uint32 pmId,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to let BBTX power meter run with new configuration. Dfe_getSyncStatus() may be called later to check if the sync has come.

NOTE, Both axc_valid bit and pm_en bit (in register dfe.bb.bbtXif_axc_config0) have to be set '1', in order to make power meter run measurement.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBTX power meter Id, 0 ~ 15
ssel	[in] sync select to update power meter

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.17 Clear BBTX Power Meter Done Status

Clear BBTX power meter complete interrupt status.

Prototype

```
DFE_Err Dfe_clearBbtXPowmtrDoneIntrStatus
(
    DFE_Handle hDfe,
    Uint32 pmId
);
```

Description

Clear complete interrupt status of a BBTX power meter.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBTX power meter Id, 0 ~ 15

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.18 Get BBTX Power Meter Done Status

Get BBTX power meter complete interrupt status.

Prototype

```
DFE_Err Dfe_getBbtXPowmtrDoneIntrStatus
(
    DFE_Handle hDfe,
    Uint32 pmId,
    Uint32 *complete
);
```

Description

Get complete interrupt status of a BBTX power meter.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBTX power meter Id, 0 ~ 15
complete	[out] BBTX power meter complete status, 0 = measurement not complete yet 1 = measurement complete

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed

Constrains

- 4) hDfe should be a valid handle opened by Dfe_open().
- 5) DFE PLL and PSCs shall be already up running.
- 6) DFE has loaded target config and completed initialize sequence.

6.19 Read BBTX Power Meter

Read BBTX power meter result via CPU.

Prototype

```
DFE_Err Dfe_readBbtXPowmtr
(
    DFE_Handle hDfe,
    Uint32 pmId,
    float *peak,
    float *rms
);
```

Description

Read peak and RMS power results of a BBTX power meter via CPU. It reads DFE registers directly and convert them to friendly values in dB.

The API doesn't wait for a new measurement; it read back whatever current result.

NOTE, Both axc_valid bit and pm_en bit (in register dfe.bb.bbtXif_axc_config0) have to be set '1', in order to make power meter run measurement.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBTX power meter Id, 0 ~ 15
peak	[out] peak power of AxC
rms	[out] RMS power of AxC

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.20 Open BBTX Power Meter DMA

Open CPP/DMA for reading BBTX power meters.

Prototype

```
DFE_Err Dfe_openBbtXPowmtrDma
(
    DFE_Handle hDfe,
    Uint32 cppDmaId,
    Uint32 cppDescripId,
    Uint32 iqnChnl
);
```

Description

Allocate CPP/DMA channel and descriptor for BBTX power meter results uploading.
Every power meter results 4 words, first twos for peak power, last twos for RMS power. Both peak power and RMS power are formatted in floating point (26 bit mantissa + 6 bit exponent). The mantissa is a <10.16> format.

Word#	Bit[31..16]	Bit[15..0]
0	Not Used	Peak power Bit[15..6], 10-bits integer portion of mantissa. Bit[5..0], 6-bits exponent (of 2-based)
1	Not Used	Peak power 16-bits fraction portion of mantissa.
2	Not Used	RMS power Bit[15..6], 10-bits integer portion of mantissa. Bit[5..0], 6-bits exponent (of 2-based)
3	Not Used	RMS power 16-bits fraction portion of mantissa.

There're total 16 power meters for BBTX, result total 64 words. A DMA transact uploads all 64 words in one shot.

When BBTX power meter DMA already opened, call this API again will get error DFE_ERR_ALREADY_OPENED.

Arguments

hDfe	[in] DFE device handle
cppDmaId	[in] CPP/DMA channel Id 0 ~ 31, open with specified channel DFE_FL_CPP_OPEN_ANY, open with any available channel
cppDescripId	[in] CPP/DMA descriptor Id 0 ~ 127, open with specified descriptor DFE_FL_CPP_OPEN_ANY, open with any available descriptor
iqnChnl	[in] IQN2 CTL Ingress channel number, 0 ~ 15

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_AVAILABLE, if CPP/DMA channel not available
 DFE_ERR_CPP_DESCRIP_NOT_AVAILABLE, if CPP/Descriptor not available
 DFE_ERR_ALREADY_OPENED, if BBTX power meter DMA already opened
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.21 Close BBTX Power Meter DMA

Close BBTX Power Meter DMA and free resources.

Prototype

```
DFE_Err Dfe_closeBbtXPowmtr
(
    DFE_Handle hDfe
);
```

Description

Close BBTX power meter DMA and free resources allocated by Dfe_openBbtXPowmtrDma().

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbtXPowmtrDma() has called OK.

6.22 Enable BBTX Power Meter DMA

Enable BBTX Power Meter DMA.

Prototype

```
DFE_Err Dfe_enableBbtXPowmtr
(
    DFE_Handle hDfe,
    Uint32 pmId
);
```

Description

Enable CPP/DMA for BBTX Power Meter DMA by doing following

- 1) Set dma_ssel to sense ALT_BBTX_PWRMTR
- 2) Set txpm_auxint_mask to (1u << pmId)

When power meter of pmId completes a measurement, ALT_BBTX_PWRMTR interrupt will trigger CPP/DMA to start transferring.

Arguments

hDfe	[in] DFE device handle
pmId	[in] Id of BBTX power meter that triggers CPP/DMA

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbtXPowmtrDma() has called OK.

6.23 Disable BBTX Power Meter DMA

Disable BBTX Power Meter DMA.

Prototype

```
DFE_Err Dfe_disableBbtXPowmtr
(
    DFE_Handle hDfe
);
```

Description

Disable BBTX power meter DMA by doing following steps,
1) Clear txpm_auxint_mask, cut off BBTX power meter complete signal to CPP/DMA
2) Change dma_ssel not to sense any signal

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbtXPowmtrDma() has called OK.

6.24 Program BBRX Gain

Program BBRX AxCs with new gains.

Prototype

```
DFE_Err Dfe_progBbrxGain
(
    DFE_Handle hDfe,
    Uint32 numAxCs,
    Uint32 axc[],
    float gain[]
);
```

Description

Write new BBRX AxCs' gains to shadow memory. The range of gain is -48.16dB ~ +96.3dB. When gain less than -48.2dB, samples are clamped to 0.

NOTE, Dfe_issueSyncUpdateBbrxGain should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
numAxCs	[in] number of AxCs whose gains are to be changed

axc	[in] array of AxCs whose gains are to be changed
gain	[in] array of new gains, in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateBbtxGain () should be called later to copy gains to working memory.

6.25 Issue Sync Update BBRX Gain

Issue sync update BBRX gain to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateBbrxGain
(
    DFE_Handle hDfe,
    Uint32 ct,
    DfeFl_MiscSyncGenSig ssl
);
```

Description

Issue sync to copy BBRX gains from shadow to working memory.
Dfe_getSyncStatus() should be called later to check if the sync has come.

NOTE, in order to make gains effectively,
dfe.bb.bbrxif_axc_config0.axc_valid bit have be '1', and
dfe.bb.bbrxif_axc_config0.beagc mode value must be in range 0 ~ 3.

Arguments

hDfe	[in] DFE device handle
ct	[in] carrier type
ssl	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.26 Get BBRX Gain Update Complete

Get BBRX gain update complete status.

Prototype

```
DFE_Err Dfe_getBbrxGainUpdateComplete
(
    DFE_Handle hDfe,
    Uint32 ct,
    Uint32 *complete
);
```

Description

Get BBRX gain update complete status. The API first reads rxgain_update_status, if corresponding ct bit is clear, then the update is still in progress. Otherwise, it further reads back and returns the update complete interrupt status.

Arguments

hDfe	[in] DFE device handle
ct	[in] carrier type
complete	[out] buffer of update complete status 0 = still in progress 1 = update complete

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL HwGetStatus() failed

Constrains

- 4) hDfe should be a valid handle opened by Dfe_open().
- 5) DFE PLL and PSCs shall be already up running.
- 6) DFE has loaded target config and completed initialize sequence.

6.27 Program BBRX Power Meter

Program BBRX power meter with new configuration.

Prototype

```
DFE_Err Dfe_progBbrxPowmtr
(
    DFE_Handle hDfe,
```

```
    Uint32 pmId,  
    DFE_BbrxPowmtrConfig *mtrCfg  
);
```

Description

Write new BBRX power meter configuration.

```
// BBTX power meter config  
typedef struct  
{  
    // enable power meter function  
    DfeFl_BbPowMtrEnable enable;  
    // carrier type  
    Uint32 countSource;  
    // power meter input source  
    DfeFl_BbPowMtrInSource inSource;  
    // tdd mode  
    DfeFl_BbPowMtrTddMode tddMode;  
    // delay from sync  
    Uint32 syncDly;  
    // meter interval  
    Uint32 interval;  
    // integration period  
    Uint32 intgPd;  
    // count of measurements, i.e. count of intervals  
    Uint32 powUpIntvl;  
    // RX Maximum full scale power in dB for the programmed power  
    interval time.  
    // Used by feed forward power update.  
    // Value is in units of 0.05dB resolution  
    Uint32 maxdB  
} DFE_BbrxPowmtrConfig;
```

NOTE, Dfe_issueSyncUpdateBbrxPowmtr should be called later to let hardware take the configuration.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBRX power meter Id, 0 ~ 15
mtrCfg	[in] new meter configuration

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateBbrxPowmtr() should be called later to let

hardware take new configuration.

6.28 Issue Sync Update BBRX Power Meter

Issue sync update BBRX power meter to new configuration.

Prototype

```
DFE_Err Dfe_issueSyncUpdateBbrxPowmtr
(
    DFE_Handle hDfe,
    Uint32 pmId,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to let BBRX power meter run with new configuration. Dfe_getSyncStatus() may be called later to check if the sync has come.

NOTE, Both dfe.bb.bbrxif_axc_config0.axc_valid bit and dfe.bb.bbrxif_axc_config1.pm_en bit have to be set '1', in order to make power meter run measurement.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBRX power meter Id, 0 ~ 15
ssel	[in] sync select to update power meter

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.29 Clear BBRX Power Meter Done Status

Clear BBRX power meter complete interrupt status.

Prototype

```
DFE_Err Dfe_clearBbrxPowmtrDoneIntrStatus
(
    DFE_Handle hDfe,
    Uint32 pmId
);
```

```
);
```

Description

Clear complete interrupt status of a BBRX power meter.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBRX power meter Id, 0 ~ 15

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.30 Get BBRX Power Meter Done Status

Get BBRX power meter complete interrupt status.

Prototype

```
DFE_Err Dfe_getBbrxPowmtrDoneIntrStatus
(
    DFE_Handle hDfe,
    Uint32 pmId,
    Uint32 *complete
);
```

Description

Get complete interrupt status of a BBRX power meter.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBRX power meter Id, 0 ~ 15
complete	[out] BBRX power meter complete status, 0 = measurement not complete yet 1 = measurement complete

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().

- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.31 Read BBRX Power Meter

Read BBRX power meter result via CPU.

Prototype

```
DFE_Err Dfe_readBbrxPowmtr
(
    DFE_Handle hDfe,
    Uint32 pmId,
    float *peak,
    float *rms
);
```

Description

Read peak and RMS power results of a BBRX power meter via CPU. It reads DFE registers directly and convert them to friendly values in dB.

The API doesn't wait for a new measurement; it read back whatever current result.

NOTE, Both dfe.bb.bbrxif_axc_config0.axc_valid bit and dfe.bb.bbrxif_axc_config1.pm_en bit have to be set '1', in order to make power meter run measurement.

Arguments

hDfe	[in] DFE device handle
pmId	[in] BBRX power meter Id, 0 ~ 15
peak	[out] peak power of AxC
rms	[out] RMS power of AxC

Return Value

DFE_ERR_NONE, if API complete programmed properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
 DFE_ERR_INVALID_PARAMS, if invalid parameters

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.32 Open BBRX Power Meter DMA

Open CPP/DMA for reading BBTX power meters.

Prototype

```
DFE_Err Dfe_openBbrxPowmtrDma
(
    DFE_Handle hDfe,
    Uint32 cppDmaId,
    Uint32 cppDescripId,
    Uint32 ignChnl
);
```

Description

Allocate CPP/DMA channel and descriptor for BBRX power meter results uploading.
Every power meter results 4 words, first twos for peak power, last twos for RMS power. Both peak power and RMS power are formatted in floating point (26 bit mantissa + 6 bit exponent). The mantissa is a <10.16> format.

Word#	Bit[31..16]	Bit[15..0]
0	Not Used	Peak power Bit[15..6], 10-bits integer portion of mantissa. Bit[5..0], 6-bits exponent (of 2-based)
1	Not Used	Peak power 16-bits fraction portion of mantissa.
2	Not Used	RMS power Bit[15..6], 10-bits integer portion of mantissa. Bit[5..0], 6-bits exponent (of 2-based)
3	Not Used	RMS power 16-bits fraction portion of mantissa.

There're total 16 power meters for BBRX, result total 64 words. A DMA transact uploads all 64 words in one shot.

When BBRX power meter DMA already opened, call this API again will get error DFE_ERR_ALREADY_OPENED.

Arguments

hDfe	[in] DFE device handle
cppDmaId	[in] CPP/DMA channel Id 0 ~ 31, open with specified channel DFE_FL_CPP_OPEN_ANY, open with any available channel
cppDescripId	[in] CPP/DMA descriptor Id 0 ~ 127, open with specified descriptor DFE_FL_CPP_OPEN_ANY, open with any available descriptor
ignChnl	[in] IQN2 CTL Ingress channel number, 0 ~ 15

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_AVAILABLE, if CPP/DMA channel not available
DFE_ERR_CPP_DESCRIP_NOT_AVAILABLE, if CPP/Descriptor not available
DFE_ERR_ALREADY_OPENED, if BBRX power meter DMA already opened

DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.33 Close BBRX Power Meter DMA

Close BBRX Power Meter DMA and free recourses.

Prototype

```
DFE_Err Dfe_closeBbrxPowmtr
(
    DFE_Handle hDfe
);
```

Description

Close BBRX power meter DMA and free resources allocated by Dfe_openBbrxPowmtrDma().

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbrxPowmtrDma() has called OK.

6.34 Enable BBRX Power Meter DMA

Enable BBRX Power Meter DMA.

Prototype

```
DFE_Err Dfe_enableBbrxPowmtr
(
    DFE_Handle hDfe,
    Uint32 pmId
);
```

Description

Enable CPP/DMA for BBRX Power Meter DMA by doing following

- 1) Set dma_ssel to sense ALT_BBRX_PWRMTR
- 2) Set rxpm_auxint_mask to (1u << pmId)

When power meter of pmId completes a measurement, ALT_BBRX_PWRMTR interrupt will trigger CPP/DMA to start transferring.

Arguments

hDfe	[in] DFE device handle
pmId	[in] Id of BBRX power meter that triggers CPP/DMA

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbrxPowmtrDma() has called OK.

6.35 Disable BBRX Power Meter DMA

Disable BBRX Power Meter DMA.

Prototype

```
DFE_Err Dfe_disableBbrxPowmtr
(
    DFE_Handle hDfe
);
```

Description

Disable BBRX power meter DMA by doing following steps,

- Clear rxpm_auxint_mask, cut off BBRX power meter complete signal to CPP/DMA
- Change dma_ssel not to sense any signal

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openBbrxPowmtrDma() has called OK.

6.36 Enable Disable BB AID Loopback

Enable/Disable BB AID loopback.

Prototype

```
DFE_Err Dfe_enableDisableBbaidLoopback
(
    DFE_Handle hDfe,
    Uint32 enable
);
```

Description

BB AID loopback is very useful to test IQ data flow before DFE.

For LTE the loop is like,

BB DL buffer → Queue → IQN2 (PktDMA, AID2) → DFE (BB_AID) → IQN2 (AID2, PktDMA) → BB UL Buffer;

For WCDMA the loop is like,

BB DL buffer → IQN2 (DIO2, AID2) → DFE (BB_AID) → IQN2 (AID2, DIO2) → BB UL buffer

Set enable to '1' to enable BB AID loopback; clear it to '0' to disable BB AID loopback.

NOTE, BB AID loopback isn't working well when AxCs having different rates. In this case, BB Buf loopback can be used, see Dfe_progBbbufLoopback().

Arguments

hDfe	[in] DFE device handle
Enable	[in] 1 to enable BB AID loopback, 0 to disable

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.37 Program BB BUF Loopback

Program BB Buf loopback.

Prototype

```
DFE_Err Dfe_progBbbufLoopback
(
    DFE_Handle hDfe,
    DfeFl_BbLoopbackConfig *bufLoopback
);
```

Description

BB Buf loopback loops back IQ data flow before DFE DDUC.

For LTE,

BB DL buffer → Queue → IQN2 (PktDMA, AID2) → DFE (BBAID, BBBUF) → DFE (BB_BUF, BB_AID) → IQN2 (AID2, PktDMA) → BB UL Buffer;

For WCDMA,

BB DL buffer → IQN2 (DIO2, AID2) → DFE (BB_AID, BB_BUF) → DFE (BB_BUF, BB_AID) → IQN2 (AID2, DIO2) → BB UL buffer

Typical setup for argument bufLoopback,

Use Case	bufLoopback member setup	Description
No loopback	All = '0'	Normal operation mode
Two dduc loopback	duc0ToDdc1 = '1'; other = '0'	dduc0 → dduc1
Four dduc loopback	duc1ToDdc2 = '1'; duc0ToDdc3 = '1'; other = '0'	dduc0 → dduc3 dduc1 → dduc2
Eight dduc loopback	duc3ToDdc4 = '1'; duc2ToDdc5 = '1'; duc1ToDdc6 = '1'; duc0ToDdc7 = '1'; other = '0'	dduc0 → dduc7 dduc1 → dduc6 dduc2 → dduc5 dduc3 → dduc4

Arguments

hDfe	[in] DFE device handle
bufLoopback	[in] Buf loopback configuration

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.38 Set BB AID UL Strobe Delay

Set BB AID UpLink strobe delay for specified carrier type.

Prototype

```
DFE_Err Dfe_setBbaidUlstrokeDelay
(
    DFE_Handle hDfe,
    Uint32 ct,
    Uint32 dly
);
```

Description

Set BB AID UL Strobe delay to adjust the time from the UL_STROBE to when FRAME_START is generated. This API is very useful to align uplink FRAME_START to the first sample in radio frame out of BB AID.

Arguments

hDfe	[in] DFE device handle
ct	[in] carrier type of the UL_STROBE
dly	[in] Delay in carrier type samples

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.39 Program BB SigGen Ramp

Program BB Signal Generator to produce a ramp.

Prototype

```
DFE_Err Dfe_progBbsigGenRamp
(
    DFE_Handle hDfe,
    DfeFl_BbTestGenDev sigGenDev,
    Uint32 enable,
    Int16 startIq[2],
    Int16 stopIq[2],
    Int16 slopeIq[2]
);
```

Description

Program a BB signal geneator to produce a ramp. The startIq[0], stopIq[0], and slopeIq[0] are used to control I bus; startIq[1],

stopIq[1], and slopeIq[1] are used to control Q bus. The rules should be followed,

- 1) startIq <= stopIq
- 2) (stopIq - startIq) % slopeIq = 0

When startIq equal to stopIq and slopeIq is 0, a constant is produced.

NOTE: BB AID signal generator produces same ramp to all BBTX channels.

Arguments

hDfe	[in] DFE device handle
sigGenDev	[in] BB SigGen device
Enable	[in] 1 to enable; 0 to disable
startIq	[in] ramp start values for I bus and Q bus
stopIq	[in] ramp stop values for I bus and Q bus
slopeIq	[in] ramp step values for I bus and Q bus

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.40 Issue Sync Update BB SigGen

Issue sync update BB Signal Generator.

Prototype

```
DFE_Err Dfe_issueSyncUpdateBbsigGen
(
    DFE_Handle hDfe,
    DfeFl_BbTestGenDev sigGenDev,
    DfeFl_MiscSyncGenSig ssl
);
```

Description

Issue sync update BB SigGen. Dfe_getSyncStatus() can be called later to check if the sync has come.

When sync ssl comes, the ramp restarts from start value and step up the slope value per clock. When accumulated value equal to stop value, the ramp restarts again.

Arguments

hDfe	[in] DFE device handle
sigGenDev	[in] BB SigGen device

ssel	[in] sync select to drive BB SigGen
------	-------------------------------------

Return Value

DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- | |
|--|
| 1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running.
3) DFE has loaded target config and completed initialize sequence. |
|--|

6.41 Program BB TestBus

Program BB Test Bus.

Prototype

<pre> DFE_Err Dfe_progBbtestbus (DFE_Handle hDfe, DfeFl_BbTestCbCtrl testCbCtrl, Uint32 testCbAxc); </pre>
--

Description

DFE has many test probe points scattered over all sub-modules. CB can be used to capture a train of IQ bus signals at the probe.
--

The API enables the specified BB probe to CB interface.

NOTE , all test probes "AND together" shares single CB interface. So software should enable no more than one probe at any time. LLD internally disables all probes first before arm a new one.

Arguments

hDfe	[in] DFE device handle
testCbCtrl	[in] probe position
testCbAxc	[in] probe axc/buf number

Return Value

DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- | |
|--|
| 1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running. |
|--|

3) DFE has loaded target config and completed initialize sequence.

6.42 Program DDUC Mixer NCO Frequency

Program DDUC Mixer NCO frequency.

Prototype

```
DFE_Err Dfe_progDducMixerNCO
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    float refClock,
    float freq[12]
);
```

Description

Write new DDUC Mixer NCO to shadow memory, this is only in the static frequency mode. The precision of the frequency is $\text{refClock}/2^{48}$.

NOTE, Dfe_issueSyncUpdateDducMixerNCO () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
refClock	[in] reference sample rate
freq	[in] array of frequency value

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateDducMixerNCO () should be called later to copy gains to working memory.

6.43 Issue Sync Update DDUC Mixer NCO Frequency

Issue sync update DDUC Mixer NCO frequency to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateDducMixerNCO
(
```

```

    DFE_Handle hDfe,
    Uint32 dducDev,
    DfeFl_MiscSyncGenSig ssel
);

```

Description

Issue sync to copy DDUC Mixer NCO frequency from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.44 Program DDUC Mixer Phase

Program DDUC Mixer phase with new values.

Prototype

```

DFE_Err Dfe_progDducMixerPhase
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    float phase[12]
);

```

Description

Write new DDUC Mixer phase to shadow memory. The precision for the phase is 360/65536 degree.

NOTE, Dfe_issueSyncUpdateDducMixerPhase () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
------	------------------------

dducDev	[in] Dduc Id, 0 ~ 3
phase	[in] array of new phase, in degree

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateDducMixerPhase () should be called later to copy gains to working memory.

6.45 Issue Sync Update DDUC Mixer Phase

Issue sync update DDUC Mixer phase to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateDducMixerPhase
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy DDUC Mixer phase from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.46 Program DDUC Farrow Phase

Program DDUC Farrow Phase with new values.

Prototype

```
DFE_Err Dfe_progDducFarrowPhase
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    Uint32 fifo[12],
    float phase[12]
);
```

Description

Write new DDUC Farrow phase to shadow memory. The fifo is from 0 ~ 63. The phase range is -0.5 ~ 0.5.

NOTE, Dfe_issueSyncUpdateDducFarrowPhase () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
fifo	[in] array of fifo
phase	[in] array of new phase

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateDducFarrowPhase () should be called later to copy gains to working memory.

6.47 Issue Sync Update DDUC Farrow Phase

Issue sync update DDUC Farrow phase to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateDducFarrowPhase
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    DfeFl MiscSyncGenSig ssel
);
```

```
);
```

Description

Issue sync to copy DDUC farrow phase from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.48 Program Distributor Map

Program DDUC distributor map.

Prototype

```
DFE_Err Dfe_progDducDistMap
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    Uint32 rxSel[12],
    Uint32 chanSel[12]
);
```

Description

Write new DDUC distributor map to shadow memory. Each rxSel and chanSel value is associated with one DDUC channel. The first 4 channels are for mixer0, the second 4 channels are for mixer1 and the last 4 channels are for mixer2.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
rxSel	[in] array of rx selection 0: from Rx 1: from feedback

chanSel	[in] array of chan selection, which channel from selected rx.
---------	---

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateDducDistMap () should be called later to copy gains to working memory.

6.49 Issue Sync Update Distributor Map

Issue sync update DDUC distributor map to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateDducDistMap
(
    DFE_Handle hDfe,
    Uint32 dducDev,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy DDUC distributor map from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
dducDev	[in] Dduc Id, 0 ~ 3
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.50 Program Summer Shift

Program Summer shift.

Prototype

```
DFE_Err Dfe_progSummerShift
(
    DFE_Handle hDfe,
    Uint32 cfrId,
    Uint32 strId,
    int gain
);
```

Description

Write new Summer shift. The range of gain is -36dB ~ 6dB, with step 6dB.

Arguments

hDfe	[in] DFE device handle
cfrId	[in] cfr Id, 0 ~ 1
StrId	[in] stream Id, 0 ~ 1
gain	[in] new gain, in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.51 Program Summer Map

Program Summer map.

Prototype

```
DFE_Err Dfe_progSummerMap
(
    DFE_Handle hDfe,
    Uint32 cfrId,
    Uint32 strId,
    Uint32 sumMap[4]
);
```

Description

Write new Summer map configuration.

NOTE, Dfe_issueSyncUpdateSummerMap () should be called later to let hardware take the configuration.

Arguments

hDfe	[in] DFE device handle
cfrId	[in] Cfr Id, 0 ~ 1
strId	[in] stream Id, 0 ~ 1
sumMap	[in] summer map for 4 DDUC. Each word uses the 12 LSB bits to map 12 carriers for each DDUC.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateSummerMap() should be called later to let hardware take new configuration.

6.52 Issue Sync Update Summer Map

Issue sync update Summer map to new configuration.

Prototype

```
DFE_Err Dfe_issueSyncUpdateSummerMap
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to let Summer run with new configuration.
Dfe_getSyncStatus() may be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to update power meter

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.53 Program CFR Coefficients

Program CFR coefficients.

Prototype

```
DFE_Err Dfe_progCfrCoeff
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    Uint32 numCoeffs,
    Uint32int *cfrCoeff_i,
    Uint32int *cfrCoeff_q
);
```

Description

Write new Cfr coefficients to the shadow memory. The range of each coefficient is 0 ~ 4095. The maximum number of coefficients is 256 for each Cfr instance. It is shared by all antennas in each Cfr instance.

NOTE, Dfe_issueSyncUpdateCfrCoeff () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
numCoeffs	[in] number of coefficients
cfrCoeff_i	[in] pointer to the real part coefficient
cfrCoeff_q	[in] pointer to the image part coefficient

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateCfrCoeff () should be called later to copy

gains to working memory.

6.54 Issue Sync Update CFR Coefficients

Issue sync update Cfr Coefficients.

Prototype

```
DFE_Err Dfe_issueSyncUpdateCfrCoeff
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    Uint32 coeffType,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync update Cfr coefficients. Dfe_getSyncStatus() can be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
coeffType	[in] Cfr coefficient type, 0 means the base coefficient 1 means the half delay coefficient
ssel	[in] sync select to drive BB SigGen

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.55 Program CFR preGain

Program CFR preGain.

Prototype

```
DFE_Err Dfe_progCfrPreGain
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    DfeFl_CfrPath cfrPath,
    float gainGain
);
```

```
);
```

Description

Write new Cfr preGain to shadow memory. The range is -Inf ~ 6dB.

NOTE, Dfe_issueSyncUpdateCfrPreGain () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
cfrPath	[in] Cfr path Id, 0 - 1
gainGain	[in] Cfr pre gain value in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateCfrPreGain() should be called later to let hardware take new configuration.

6.56 Issue Sync Update CFR preGain

Issue sync update CFR preGain.

Prototype

```
DFE_Err Dfe_issueSyncUpdatCfrPreGain
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    DfeFl_CfrPath cfrPath,
    DfeFl_MiscSyncGenSig ssl
);
```

Description

Issue sync to copy CFR pre gain from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1

cfrPath	[in] Cfr path Id, 0 - 1
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running.
3) DFE has loaded target config and completed initialize sequence.

6.57 Program CFR postGain

Program CFR postGain.

Prototype

<pre> DFE_Err Dfe_progCfrPostGain (DFE_Handle hDfe, Uint32 cfrDev, DfeFl_CfrPath cfrPath, float gainGain); </pre>

Description

<p>Write new Cfr postGain to shadow memory. The range is -Inf ~ 6dB.</p> <p>NOTE, Dfe_issueSyncUpdateCfrPostGain () should be called later to let hardware copy gains from shadow to working memory.</p>

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
cfrPath	[in] Cfr path Id, 0 - 1
gainGain	[in] Cfr post gain value in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateCfrPostGain() should be called later to let hardware take new configuration.

6.58 Issue Sync Update CFR postGain

Issue sync update CFR postGain.

Prototype

```
DFE_Err Dfe_issueSyncUpdatCfrPostGain
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    DfeFl_CfrPath cfrPath,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy CFR pre gain from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
cfrPath	[in] Cfr path Id, 0 - 1
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.59 Program CFR Protection Gain

Program CFR protection Gain.

Prototype

```
DFE_Err Dfe_progCfrProtGain
(
    DFE_Handle hDfe,
    Uint32 cfrDev,
    DfeFl_CfrPath cfrPath,
    float gainGain
);
```

Description

Write new Cfr protection gain. It is a backoff gain when PA protection is in alarm mode. The range is -Inf ~ 6dB.

Arguments

hDfe	[in] DFE device handle
cfrDev	[in] Cfr device Id, 0 - 1
cfrPath	[in] Cfr path Id, 0 - 1
gainGain	[in] Cfr protection gain value in dB

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.60 Program TX Mixer

Program TX mixer.

Prototype

```
DFE_Err Dfe_progTxMixer
(
    DFE_Handle hDfe,
    DfeFl_TxPath txPath,
    float refClock,
    float freq[2]
);
```

Description

Write new Tx Mixer NCO frequency to shadow memory. The precision is refClock/2^48.

NOTE, Dfe_issueSyncUpdateTxMixer () should be called later to let

hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
txPath	[in] Tx path Id, 0 - 1
refClock	[in] reference clock
freq	[in] frequency value

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateTxMixer() should be called later to let hardware take new configuration.

6.61 Issue Sync Update TX Mixer

Issue sync update TX Mixer.

Prototype

```
DFE_Err Dfe_issueSyncUpdateTxMixer
(
    DFE_Handle hDfe,
    DfeFl_TxPath txPath,
    DfeFl_MiscSyncGenSig ssl
);
```

Description

Issue sync to let TX Mixer run with new frequency. Dfe_getSyncStatus() may be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
txPath	[in] Tx path Id, 0 ~ 1
ssl	[in] sync select to update power meter

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.62 Program TX PA Protection

Program Tx PA protection.

Prototype

```
DFE_Err Dfe_progTxPaProtection
(
    DFE_Handle hDfe,
    DfeFl_TxDev txDev,
    DFE_TxPAprotPeak txPAprotPeak,
    DFE_TxPAprotRms txPAprotRms,
    Uint32 mask
);
```

Description

Write new Tx PA protection configuration.

```
typedef struct
{
    // square clipper threshold
    Uint32 threshold;
    // clipper counter threshold (C1)
    Uint32 cc_thr;
    // peak threshold (TH0)
    Uint32 TH0;
    // peak counter threshold (C0)
    Uint32 peak_thr;
    // peakgain counter threshold (C2)
    Uint32 peakgain_thr;
} DFE_TxPAprotPeak;

typedef struct
{
    // mu_p for IIR
    Uint32 mu0;
    // mu_q for IIR
    Uint32 mu1;
    // RMS threshold to reduce CFR gain (TH1)
    Uint32 TH1;
    // RMS threshold to shut down (TH2)
    Uint32 TH2;
    // RMS threshold to peak approaching saturation (TH4)
    Uint32 TH4;
    // threshold selection for a1
    Uint32 th1Sel;
    // threshold selection for a2
    Uint32 th2Sel;
```

```
// threshold selection for a6
Uint32 th6Sel;
} DFE_TxPAprotRms;

a1 = 1, RMS power is greater than TH1, the CFR gain will be reduced;
a2 = 1, RMS power is greater than TH2, the TX output will be shut down.
a3 = 1, RMS power gain is less than 1, the power is saturating.
a4 = 1, peak gain counter is greater than peakgain_thr, peak is
clipping.
a5 = 1, circular clipper counter is greater than cc_thr, peak is
clipping.
a6 = 1, RMS power is greater than TH4, power is approaching saturation.
a7 = 1, peak counter is greater than peak_thr, peak is approaching
saturation.
```

Arguments

hDfe	[in] DFE device handle
txDev	[in] Tx Dev Id, 0 ~ 3
txPAprotPeak	[in] Tx PA protection for peak adjustment
txPAprotRms	[in] Tx PA protection for rms adjustment
mask	[in] Tx PA protection mask configuration for stopDPD interrupt, the 5 LSB bits correspond to a5, a4, a3, a2 and a1

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.63 Get TX PA Protection Interrupt Status

Get Tx PA protection interrupt status.

Prototype

```
DFE_Err Dfe_getTxPAprotIntrStatus
(
    DFE_Handle hDfe,
    DfeFl_TxPaIntrpt *txPAprotIntr
);
```

Description

Get Tx PA protection interrupt status of one Tx path.

Interrupt 1: a1 = 1;
 Interrupt 2: a2 = 1;

Interrupt 3:	a3 = 1;
Interrupt 4:	a4 = 1 or a5 = 1;
Interrupt 5:	a6 = 1;
Interrupt 6:	a7 = 1;
Shutdown:	a2 = 1;
lowCFRgain:	a1 = 1;
stopDPD:	programmable with a1, a2, a3, a4, a5 (programmable mask, then OR all unmasked bits)

Arguments

hDfe	[in] DFE device handle
txPAProtIntr	[out] Tx PA protection interrupt status

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constrains

- | |
|--|
| <ol style="list-style-type: none"> 1) hDfe should be a valid handle opened by Dfe_open(). 2) DFE PLL and PSCs shall be already up running. 3) DFE has loaded target config and completed initialize sequence. |
|--|

6.64 Clear TX PA Protection Interrupt Status

Clear TX PA protection interrupt status.

Prototype

<pre>DFE_Err Dfe_clearTxPAProtIntrStatus (DFE_Handle hDfe, DfeFl_TxDev txDev);</pre>
--

Description

Clear complete interrupt status of a TX PA protection.
--

Arguments

hDfe	[in] DFE device handle
txDev	[in] Tx device Id, 0 ~ 3

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- | |
|--|
| <ol style="list-style-type: none"> 1) hDfe should be a valid handle opened by Dfe_open(). 2) DFE PLL and PSCs shall be already up running. |
|--|

3) DFE has loaded target config and completed initialize sequence.

6.65 Read TX PA Protection Power Status

Read Tx PA protection internal power status.

Prototype

```
DFE_Err Dfe_getTxPAprotPwrStatus
(
    DFE_Handle hDfe,
    DfeFl_TxDev txDev,
    Uint32 clrRead,
    DFE_TxPAprotPwrStatus *txPAprotPwrStatus
);
```

Description

Read Tx PA protection internal status of one Tx path.

```
typedef struct
{
    // maximum magnitude of D3
    Uint32 mag;
    // IIR output at D50
    Uint32 d50;
    // IIR output at D51
    Uint32 d51;
} DFE_TxPAprotPwrStatus;
```

Arguments

hDfe	[in] DFE device handle
txDev	[in] Tx device Id, 0 ~ 1
clrRead	[in] flag to set clear after reading for magnitude 0 means no clear after reading 1 means clear after reading
txPAprotPwrStatus	[out] Tx PA protection internal status

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.66 Program JESD Tx to Lane Map

Program JESD Tx bus to lane map.

Prototype

```
DFE_Err Dfe_mapJesdTx2Lane
(
    Uint32 lane,
    DFE_JesdTxLaneMapPos laneMap[4]
);
```

Description

Program how to fill time slots of a lane with Tx bus time slots.

A Tx bus consists of four time slots. A Tx bus time slot, an element of laneMap[], is selected by the bus# and slot# of the bus.

```
// Jesd Tx bus to lane map
typedef struct
{
    // bus#, one of DfeFl_JesdTxTxBus (I0, Q0, I1, Q1)
    Uint32 bus;
    // slot#, 0 ~ 3
    Uint32 busPos;
} DFE_JesdTxLaneMapPos;
```

A lane also consists of four time slots.

- laneMap[0] is mapping to lane time slot 0
- laneMap[1] is mapping to lane time slot 1
- laneMap[2] is mapping to lane time slot 2
- laneMap[3] is mapping to lane time slot 3

For example, TX0 has two interleaved antenna streams, A0 and A1, which are mapping to two lanes, lane0 and lane1, A0 => lane0, A1 => lane1.

TX0 bus format,

Bus#	Slot0	Slot1	Slot2	Slot3
0 (TX0 I)	A0 i	A1 i	Don't care	Don't care
1 (TX0 Q)	A0 q	A1 q	Don't care	Don't care

Lane bus format

Lane#	Slot0	Slot1	Slot2	Slot3
0 (lane0)	A0 i	A0 q	Don't care	Don't care
1 (lane1)	A1 i	A1 q	Don't care	Don't care

Then laneMap for lane0 should be,

	laneMap[0]	laneMap[1]	laneMap[2]	laneMap[3]
Bus#	0	1	Don't care	Don't care
Slot#	0	0	Don't care	Don't care

Then laneMap for lane1 should be,

	laneMap[0]	laneMap[1]	laneMap[2]	laneMap[3]
Bus#	0	1	Don't care	Don't care

Slot#	1	1	Don't care	Don't care
-------	---	---	------------	------------

Arguments

hDfe	[in] DFE device handle
lane	[in] Tx lane#
laneMap	[in] Tx bus slot map

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running.
3) DFE has loaded target config and completed initialize sequence.

6.67 Program JESD Tx SigGen Ramp

Program JESD TX Signal Generator to produce a ramp.

Prototype

<pre> DFE_Err Dfe_progJesdTxSigGenRamp (DFE_Handle hDfe, DfeFl_JesdTxSignalGen sigGenDev, Uint32 enable, Int32 start, Int32 stop, Int32 slope); </pre>
--

Description

<p>Program a JESD TX signal generator to produce a ramp. The rules should be followed,</p> <ol style="list-style-type: none"> 1) start <= stop 2) (stop - start) % slope = 0 3) Start/stop/slope range, -131072 ~ 131071 <p>When start equal to stop and slope is 0, a constant is produced.</p>
--

Arguments

hDfe	[in] DFE device handle
sigGenDev	[in] JESD TX SigGen device
Enable	[in] 1 to enable; 0 to disable
Start	[in] ramp start values for the bus
Stop	[in] ramp stop values for the bus

slope	[in] ramp step values for the bus
-------	-----------------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.68 Issue Sync Update JESD TX SigGen

Issue sync update JESD TX Signal Generator.

Prototype

```
DFE_Err Dfe_issueSyncUpdateJesdTxSigGen
(
    DFE_Handle hDfe,
    CSL_DfeJesdTxSignalGen sigGenDev,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync update JESD TX SigGen. Dfe_getSyncStatus() can be called later to check if the sync has come.

When sync ssel comes, the ramp restarts from start value and step up the slope value per clock. When accumulated value equal to stop value, the ramp restarts again.

Arguments

hDfe	[in] DFE device handle
sigGenDev	[in] BB SigGen device
ssel	[in] sync select to drive BB SigGen

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.69 Program JESD Tx Testbus

Program JESD TX Test Bus.

Prototype

```
DFE_Err Dfe_progJesdTxTestbus
(
    DFE_Handle hDfe,
    DfeFl_JesdTxTestBusSel tp
);
```

Description

DFE has many test probe points scattered over all sub-modules. CB can be used to capture a train of IQ bus signals at the probe.

The API enables the specified JESD TX test probe to CB interface.

NOTE, all test probes "AND together" shares single CB interface. So software should enable no more than one probe at any time. LLD internally disables all probes first before arm a new one.

Arguments

hDfe	[in] DFE device handle
tp	[in] probe position

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.70 Get JESD Tx Link Status

Get JESD TX link status.

Prototype

```
DFE_Err Dfe_getJesdTxLinkStatus
(
    DFE_Handle hDfe,
    DFE_JesdTxLinkStatus *linkStatus
);
```

Description

The API get back following Tx link status,

```
// Jesd Tx link status
```

```
typedef struct
{
    // first sync request received for the link
    // 0 - not seen first sync request
    // 1 - seen first sync request
    Uint32 firstSyncRequest[DFE_FL_JESD_NUM_LINK];
    // error count as reported over SYNC~ interface.
    Uint32 syncErrCount[DFE_FL_JESD_NUM_LINK];
    // SYSREF alignment counter bits
    Uint32 sysrefAlignCount;
    // captured interrupt bit for sysref_request_assert
    // 0 - sysref request not asserted
    // 1 - sysref request asserted
    Uint32 sysrefReqAssert;
    // captured interrupt bit for sysref_request_deassert
    // 0 - sysref request not de-asserted
    // 1 - sysref request de-asserted
    Uint32 sysrefReqDeassert;
    // captured interrupt bit for sysref_err on the link
    // 0 - no sysref error
    // 1 - sysref error
    Uint32 sysrefErr[DFE_FL_JESD_NUM_LINK];
} DFE_JesdTxDLinkStatus;
```

Arguments

hDfe	[in] DFE device handle
linkStatus	[out] pointer to link status buffer

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.71 Get JESD Tx Lane Status

Get JESD TX lanes status.

Prototype

```
DFE_Err Dfe_getJesdTxDLaneStatus
(
    DFE_Handle hDfe,
    DFE_JesdTxDLaneStatus *laneStatus
);
```

Description

```
The API get back following Tx lane status,

// Jesd Tx lane status
typedef struct
{
    // synchronization state machine status for lane
    Uint32 syncState[DFE_FL_JESD_NUM_LANE];
    // FIFO status
    // 0 - fifo not empty; 1 - fifo has been empty
    Uint32 fifoEmpty[DFE_FL_JESD_NUM_LANE];
    // 0 - no read error; 1 - fifo read error
    Uint32 fifoReadErr[DFE_FL_JESD_NUM_LANE];
    // 0 - fifo not full; 1 - fifo has been full
    Uint32 fifoFull[DFE_FL_JESD_NUM_LANE];
    // 0 - no write error; 1 - fifo write error
    Uint32 fifoWriteErr[DFE_FL_JESD_NUM_LANE];
} DFE_JesdTxLaneStatus;
```

Arguments

hDfe	[in] DFE device handle
laneStatus	[out] pointer to lane status buffer

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.72 Clear JESD Tx Link Errors

Clear JESD TX link errors

Prototype

```
DFE_Err Dfe_clearJesdTxLinkErrors
(
    DFE_Handle hDfe
);
```

Description

```
The API clears following Tx link status,
• sysrefReqAssert
• sysrefReqDeassert
• sysrefErr[DFE_FL_JESD_NUM_LINK] for all links
• syncErrCount[DFE_FL_JESD_NUM_LINK] for all links
```

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

<p>DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed</p>
--

Constrains

- | |
|--|
| <ol style="list-style-type: none"> 1) hDfe should be a valid handle opened by Dfe_open(). 2) DFE PLL and PSCs shall be already up running. 3) DFE has loaded target config and completed initialize sequence. |
|--|

6.73 Clear JESD Tx Lane Errors

Clear JESD TX lane errors

Prototype

<pre>DFE_Err Dfe_clearJesdTxLaneErrors (DFE_Handle hDfe);</pre>

Description

<p>The API clears following Tx lane status,</p> <ul style="list-style-type: none"> • fifoEmpty[DFE_FL_JESD_NUM_LANE] for all lanes • fifoReadErr[DFE_FL_JESD_NUM_LANE] for all lanes • fifoFull[DFE_FL_JESD_NUM_LANE] for all anes • fifoWriteErr[DFE_FL_JESD_NUM_LANE] for all lanes

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

<p>DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed</p>
--

Constrains

- | |
|--|
| <ol style="list-style-type: none"> 4) hDfe should be a valid handle opened by Dfe_open(). 5) DFE PLL and PSCs shall be already up running. 6) DFE has loaded target config and completed initialize sequence. |
|--|

6.74 Program JESD Lane to Rx Map

Program JESD lane to Rx bus map.

Prototype

```
DFE_Err Dfe_mapJesdLane2Rx
(
    DFE_Handle hDfe,
    Uint32 rxBus,
    AVV_JesdRxBusMapPos busMap[4]
);
```

Description

Program how to fill time slots of a RX bus with lane time slots.

A Rx lane consists of four time slots. A Rx bus time slot, an element of busMap[], is selected by the lane# and slot# of the lane.

```
// Jesd Rx lane to bus map
typedef struct
{
    // Rx lane#, 0 ~ 3
    Uint32 lane;
    // lane time slot
    Uint32 lanePos;
    // if zero data
    Uint32 zeroBits;
} DFE_JesdRxBusMapPos;
```

A Rx bus also consists of four time slots.

- busMap[0] is mapping to bus time slot 0
- busMap[1] is mapping to bus time slot 1
- busMap[2] is mapping to bus time slot 2
- busMap[3] is mapping to bus time slot 3

For example, two Rx lanes, lane0 and lane1, are mapping to RX0, which will carry two interleaved antenna streams A0 and A1, lane0 => A0, lane1 => A1.

RX0 bus format,

Bus#	Slot0	Slot1	Slot2	Slot3
0 (RX0_I)	A0_i	A1_i	Don't care	Don't care
1 (RX0_Q)	A0_q	A1_q	Don't care	Don't care

Lane bus format

Lane#	Slot0	Slot1	Slot2	Slot3
0 (lane0)	A0_i	A0_q	Don't care	Don't care
1 (lane1)	A1_i	A1_q	Don't care	Don't care

Then busMap for RX0_I should be,

	busMap[0]	busMap[1]	busMap[2]	busMap[3]
lane#	0	1	Don't care	Don't care
Slot#	0	0	Don't care	Don't care

Then busMap for RX0 Q should be,

	busMap[0]	busMap[1]	busMap[2]	busMap[3]
lane#	0	1	Don't care	Don't care
Slot#	1	1	Don't care	Don't care

Arguments

hDfe	[in] DFE device handle
rxBus	[in] Rx bus#
busMap	[in] Rx lane slot map

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.75 Program JESD Loopback

Program JESD loopbacks for sync_n, lanes or links.

Prototype

```
DFE_Err Dfe_progJesdLoopback
(
    DFE_Handle hDfe,
    Uint32 lpbkSync[DFE_FL_JESD_NUM_LINK];
    DfeFl_JesdRxLoopbackConfig lpbkLaneLink;
);
```

Description

Enable/disable sync_n, lanes/links loopback between JESD TX and JESD RX.

Arguments

hDfe	[in] DFE device handle
lpbkSync	[in] 1: enable rx sync out loopback to tx sync n
lpbkLaneLink	[in] lane/link loopback config

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.76 Program JESD Rx Testbus

Program JESDRX Test Bus.

Prototype

```
DFE_Err Dfe_progJesdRxTestbus
(
    DFE_Handle hDfe,
    DfeFl_JesdRxTestBusSel tp
);
```

Description

DFE has many test probe points scattered over all sub-modules. CB can be used to capture a train of IQ bus signals at the probe.

The API enables the specified JESDRX test probe to CB interface.

NOTE, all test probes "AND together" shares single CB interface. So software should enable no more than one probe at any time. LLD internally disables all probes first before arm a new one.

Arguments

hDfe	[in] DFE device handle
testCbCtrl	[in] probe position
testCbAxc	[in] probe axc/buf number

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.77 Get JESD Rx Link Status

Get JESDRX link status.

Prototype

```
DFE_Err Dfe_getJesdRxLinkStatus
(
    DFE_Handle hDfe,
    DFE_JesdRxLinkStatus *linkStatus
);
```

Description

The API get back following Rx link status,

```
// Jesd Rx link status
typedef struct
{
    // SYSREF alignment counter bits
    Uint32 sysrefAlignCount;
    // captured interrupt bit for sysref_request_assert
    // 0 - sysref request not asserted
    // 1 - sysref request asserted
    Uint32 sysrefReqAssert;
    // captured interrupt bit for sysref_request_deassert
    // 0 - sysref request not de-asserted
    // 1 - sysref request de-asserted
    Uint32 sysrefReqDeassert;
    // captured interrupt bit for sysref_err on the link
    // 0 - no sysref error
    // 1 - sysref error
    Uint32 sysrefErr[DFE_FL_JESD_NUM_LINK];
} DFE_JesdRxLinkStatus;
```

Arguments

hDfe	[in] DFE device handle
linkStatus	[out] pointer to link status buffer

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.78 Get JESD Rx Lane Status

Get JESDRX lanes status.

Prototype

```
DFE_Err Dfe_getJesdRxLaneStatus
```

```
(
    DFE_Handle hDfe,
    DFE_JesdRxLaneStatus *laneStatus
);
```

Description

The API get back following Tx lane status,

```
// Jesd Rx lane status
typedef struct
{
    // code group synchronization state machine status for lane
    Uint32 syncStatecodeState[DFE_FL_JESD_NUM_LANE];
    // frame synchronization state machine status for lane
    Uint32 frameState[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - 8B/10B disparity error
    Uint32 decDispErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - 8B/10B not-in-table code error
    Uint32 decCodeErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - code group sync error
    Uint32 codeSyncErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - elastic buffer match error
    // (first non-/K/ doesn't match match_ctrl and match_data)
    Uint32 bufMatchErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - elastic buffer overflow error (bad RBD value)
    Uint32 bufOverflowErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - link configuration error
    Uint32 linkConfigErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - frame alignment error
    Uint32 frameAlignErr[DFE_FL_JESD_NUM_LANE];
    // 0 - no error; 1 - multiframe alignment error
    Uint32 multiframeAlignErr[DFE_FL_JESD_NUM_LANE];
    // FIFO status
    // 0 - normal; 1 - fifo empty
    Uint32 fifoEmpty[DFE_FL_JESD_NUM_LANE];
    // 0 - normal; 1 - fifo read error
    Uint32 fifoReadErr[DFE_FL_JESD_NUM_LANE];
    // 0 - normal; 1 - fifo full
    Uint32 fifoFull[DFE_FL_JESD_NUM_LANE];
    // 0 - normal; 1 - fifo write error
    Uint32 fifoWriteErr[DFE_FL_JESD_NUM_LANE];
    // 0 - normal; 1 - test sequence verification failed
    Uint32 testSeqErr[DFE_FL_JESD_NUM_LANE];
} DFE_JesdRxLaneStatus;
```

Arguments

hDfe	[in] DFE device handle
laneStatus	[out] pointer to lane status buffer

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.79 Clear JESD Rx Link Error

Clear JESDRX link errors

Prototype

```
DFE_Err Dfe_clearJesdRxLinkErrors  
(  
    DFE_Handle hDfe  
);
```

Description

The API clears following Tx link status,

- sysrefReqAssert
- sysrefReqDeassert
- sysrefErr[DFE_FL_JESD_NUM_LINK] for all links

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.80 Clear JESD Rx Lane Error

Clear JESDRX lane errors

Prototype

```
DFE_Err Dfe_clearJesdRxLaneErrors  
(  
    DFE_Handle hDfe  
);
```

Description

The API clears following lane status for all Tx lanes,

- decDispErr[DFE_FL_JESD_NUM_LANE];
- decCodeErr[DFE_FL_JESD_NUM_LANE];
- codeSyncErr[DFE_FL_JESD_NUM_LANE];
- bufMatchErr[DFE_FL_JESD_NUM_LANE];
- bufOverflowErr[DFE_FL_JESD_NUM_LANE];
- linkConfigErr[DFE_FL_JESD_NUM_LANE];
- frameAlignErr[DFE_FL_JESD_NUM_LANE];
- multiframeAlignErr[DFE_FL_JESD_NUM_LANE];
- fifoEmpty[DFE_FL_JESD_NUM_LANE];
- fifoReadErr[DFE_FL_JESD_NUM_LANE];
- fifoFull[DFE_FL_JESD_NUM_LANE];
- fifoWriteErr[DFE_FL_JESD_NUM_LANE];

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- | |
|--|
| <ol style="list-style-type: none"> 1) hDfe should be a valid handle opened by Dfe_open(). 2) DFE PLL and PSCs shall be already up running. 3) DFE has loaded target config and completed initialize sequence. |
|--|

6.81 Program RX IBPM Global

Program global settings of Rx IBPM.

Prototype

<pre> DFE_Err Dfe_progRxIbpmGlobal (DFE_Handle hDfe, DfeFl_RxPowmtrReadMode readMode, float histThresh1, float histThresh2, CSL_Uint64 unityMagsq); </pre>
--

Description

<p>Program global settings of RX IBPM. The power meters measure RMS and peak powers at Rx block input. There're total 4 IBPMs, which can be individually configured by Dfe_progRxIbpm() to meter one antenna.</p>

<p>IBPM has two reading modes, hardware interrupt mode and software</p>

handshake mode.

In the hardware interrupt mode, the power meters run per the programmed configuration and set interrupts on a per antenna basis when a new set of results have been computed and captured in the read registers. It is the responsibility of the user to read the results before the next set of results are computed and captured in the read registers.

In the software handshake mode, the power meters run per the programmed configuration. The user makes a request to read a current set of results and waits until receiving an acknowledge signal from DFE hardware before reading. Upon receiving the request DFE completes the current power meter cycle, stores the results for reading, sets the acknowledge and halts any further updates until the user resets the request.

Each IBPM has two histogram counters to count number of samples whose magnitude is above the specified histThresh.

To convert between raw register values to friendly floatings in dB, LLD needs knowing the raw value of unity magnitude square (unityMagSq), i.e. $I^2 + Q^2$.

Arguments

hDfe	[in] DFE device handle
readMode	[in] meter reading mode
histThread1	[in] threshold in dB for histogram counter1
histThread2	[in] threshold in dB for histogram counter2
unityMagsq	[in] raw value of unity magnitude square

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.82 Program RX IBPM

Program settings of an Rx IBPM.

Prototype

```
DFE_Err Dfe_progRxIbpm
(
    DFE_Handle hDfe,
    Uint32 pmId,
    Uint32 oneShot,
    Uint32 meterMode,
    Uint32 syncDelay,
    Uint32 nSamples,
```

```
    Uint32 interval
);
```

Description

Program settings of an individual RX IBPM, which meters RMS and peak power at Rx block input for one antenna.

When a sync event comes, IBPM starts a new meter cycle after syncDelay samples. The cycle completes at integration time of nSamples. When oneShot is enabled(1), IBPM does no more measurement until next sync event; when oneShot is disabled(0), IBPM starts a new measure after interval samples elapsed, and this repeats every interval. The interval shall be no less than integration time + syncDelay, otherwise IBPM never completes.

When meterMode is 0, IBPM is disabled; when meterMode is 1, IBPM is running according to configuration of sync delay, integration period and interval; when meter mode is 2, IBPM is running over a gated stream.

Arguments

hDfe	[in] DFE device handle
pmId	[in] power meter device ID
oneShot	[in] one shot mode
meterMode	[in] meter operational mode <ul style="list-style-type: none"> • 0 = off • 1 = normal mode • 2 = gated mode
syncDelay	[in] delay from sync event, in samples
nSamples	[in] integration period, in samples
interval	[in] interval period, in samples

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.83 Issue Sync Update RX IBPM

Issue sync updates an Rx IBPM.

Prototype

```
DFE_Err Dfe_issueSyncUpdateRxIbpm
(
```



```

    DFE_Handle hDfe,
    Uint32 pmId,
    DfeFl_MiscSyncGenSig ssel
);

```

Description

Issue sync update a RX IBPM. Dfe_getSyncStatus() can be called later to check if the sync has come.

When sync ssel comes, the power meter starts a new measurement cycle.

Arguments

hDfe	[in] DFE device handle
pmId	[in] Rx IBPM device Id
ssel	[in] sync select

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.84 Issue RX IBPM Read Request

Issue read request to an Rx IBPM for software handshake mode.

Prototype

```

DFE_Err Dfe_issueRxIbpmReadRequest
(
    DFE_Handle hDfe,
    Uint32 pmId
);

```

Description

The API is applied for the software handshake mode only. In this mode, the user makes a request (via this API) to read a current set of results and waits until receiving an acknowledge signal (via polling Dfe_getRxIbpmReadAck()) from DFE hardware before reading. Upon receiving the request DFE completes the current power meter cycle, stores the results for reading, sets the acknowledge and halts any further updates until the user resets the request (via Dfe_readRxIbpmResult()).

Arguments

hDfe	[in] DFE device handle
pmId	[in] Rx IBPM device Id

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.85 Get RX IBPM Read Ack

Get read acknowledge status of an Rx IBPM for software handshake mode.

Prototype

```
DFE_Err Dfe_getRxIbpmReadAck
(
    DFE_Handle hDfe,
    Uint32 pmId,
    Uint32 *ackRead
);
```

Description

The API is applied for the software handshake mode only. In this mode, the user makes a request (via Dfe_issueRxIbpmReadRequest()) to read a current set of results and waits until receiving an acknowledge signal (via polling this API) from DFE hardware before reading. Upon receiving the request DFE completes the current power meter cycle, stores the results for reading, sets the acknowledge and halts any further updates until the user resets the request (via Dfe_readRxIbpmResult()).

Arguments

hDfe	[in] DFE device handle
pmId	[in] Rx IBPM device Id
ackRead	[in] acknowledge status <ul style="list-style-type: none"> • 0, still updating • 1, read acknowledged

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.86 Read RX IBPM Result

Read back results of an Rx IBPM.

Prototype

```
DFE_Err Dfe_readRxIbpmResult
(
    DFE_Handle hDfe,
    Uint32 pmId,
    float *power,
    float *peak,
    Uint32 *histCount1,
    Uint32 *histCount2
);
```

Description

Read back results of a running RX IBPM. After retrieves all results, for hardware interrupt mode, the API writes done reading flag to DFE; for software handshake mode, the API clears read request, which was set by Dfe_IssueRxIbpmReadRequest().

Arguments

hDfe	[in] DFE device handle
pmId	[in] Rx IBPM device Id
power	[out] Power value
peak	[out] peak value
histCount1	[out] the number of samples which power is greater than the histogram one threshold
histCount2	[out] the number of samples which power is greater than the histogram two threshold

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
 DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.87 Program RX Equalizer

Program RX Equalizer.

Prototype

```
DFE_Err Dfe_progRxEqR
(
    DFE_Handle hDfe,
    Uint32 rxDev,
    Uint32 shift,
    Uint32 numCoeff,
    DFE_RxEqrTaps *RxEqRTaps
);
```

Description

Write new RX Equalizer to shadow memory. The range of each tap is -1 ~ 0.9998. Eqr bypass needs to be disabled.

NOTE, Dfe_issueSyncUpdateRxEqR () should be called later to let hardware copy gains from shadow to working memory.

```
typedef struct
{
    // ii taps
    float taps_ii[DFE_FL_RX_EQR_LEN];
    // iq taps
    float taps_iq[DFE_FL_RX_EQR_LEN];
    // qi taps
    float taps_qi[DFE_FL_RX_EQR_LEN];
    // qq taps
    float taps_qq[DFE_FL_RX_EQR_LEN];
} DFE_RxEqrTaps;
```

Arguments

hDfe	[in] DFE device handle
rxDev	[in] Rx Id, 0 ~ 3
shift	[in] shift value, 0 ~ 3
numCoeff	[in] number of coefficients
RxEqrTaps	[in] pointer to the Rx Eqr taps

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateRxEqR () should be called later to copy EQ taps to working memory.

6.88 Issue Sync Update RX Equalizer

Issue sync update RX equalizer to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateRxEqR  
(  
    DFE_Handle hDfe,  
    Uint32 rxDev,  
    DfeFl_MiscSyncGenSig ssel  
);
```

Description

Issue sync to copy Rx equalizer from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
rxDev	[in] Rx Id, 0 ~ 3
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Only the FB block which matches with IO mux config will be updated.

6.89 Program RX Mixer NCO

Program RX Mixer NCO frequency.

Prototype

```
DFE_Err Dfe_progDducMixerNCO  
(  
    DFE_Handle hDfe,  
    Uint32 rxDev,  
    float refClock,  
    float freq  
);
```

Description

Write new RX Mixer NCO to shadow memory. The precision of the frequency is $\text{refClock}/2^{48}$. NCO bypass needs to be disabled.

NOTE, Dfe_issueSyncUpdateRxMixerNCO () should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
rxDev	[in] Rx Id, 0 ~ 3
refClock	[in] reference sample rate
freq	[in] frequency value

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateRxMixerNCO () should be called later to copy gains to working memory.

6.90 Issue Sync Update RX Mixer NCO Frequency

Issue sync update RX Mixer NCO frequency to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateRxMixerNCO
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy RX Mixer NCO frequency from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly

DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.91 Program Rx Testbus

Program RX Testbus.

Prototype

```
DFE_Err Dfe_issueSyncUpdateRxTestbus
(
    DFE_Handle hDfe,
    Uint32 top_ctrl,
    Uint32 imb_ctrl,
    Uint32 feagc_dc_ctrl
);
```

Description

Program RX Test bus.

Arguments

hDfe	[in] DFE device handle
top_ctrl	[in] top testbus control.
imb_ctrl	[in] imb testbus control.
feagc_dc_ctrl	[in] feagc_dc testbus control.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.92 Program FB Equalizer

Program FB Equalizer.

Prototype

```
DFE_Err Dfe_progFbEqr
```

```
(
    DFE_Handle hDfe,
    DfeFl_FbBlk FbBlkId,
    Uint32 numCoeff,
    DFE_FbEqrTaps *FbEqrTaps
);
```

Description

Write new FB Equalizer to shadow memory. The range of each tap is -1 ~ 0.9998.

NOTE, Dfe_issueSyncUpdateFbEqr should be called later to let hardware copy gains from shadow to working memory.

```
typedef struct
{
    // ii taps
    float taps_ii[DFE_FL_FB_EQR_LEN];
    // iq taps
    float taps_iq[DFE_FL_FB_EQR_LEN];
    // qi taps
    float taps_qi[DFE_FL_FB_EQR_LEN];
    // qq taps
    float taps_qq[DFE_FL_FB_EQR_LEN];
} DFE_FbEqrTaps;
```

Arguments

hDfe	[in] DFE device handle
FbBlkId	[in] Fb block Id, 0 ~ 4
numCoeff	[in] number of coefficients
FbEqrTaps	[in] pointer to the Fb Eqr taps.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateFbEqr () should be called later to copy EQ taps to working memory.

6.93 Issue Sync Update FB Equalizer

Issue sync update FB equalizer to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateFbEqr
```



```
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy Fb equalizer from shadow to working memory. Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Only the FB block which matches with IO mux config will be updated.

6.94 Program FB Mixer NCO

Program FB Mixer NCO.

Prototype

```
DFE_Err Dfe_progFbMixerNCO
(
    DFE_Handle hDfe,
    DfeFl_FbBlk FbBlkId,
    float refClock,
    float freq
);
```

Description

Write new FB Mixer NCO frequency to shadow memory. The precision is refClock/2^48.

NOTE, Dfe_issueSyncUpdateFbMixerNCO should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
FbBlkId	[in] Fb block Id, 0~4
refClock	[in] Fb reference clock
freq	[in] frequency value in degree

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateFbMixerNCO () should be called later to copy NCO to working memory.

6.95 Issue Sync Update FB Mixer NCO

Issue sync update FB equalizer to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateFbMixerNCO
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy Fb mixer NCO from shadow to working memory.
Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.

- 3) DFE has loaded target config and completed initialize sequence.
- 4) Only the FB block which matches with IO mux config will be updated.

6.96 Program FB IO Mux

Program FB IO Mux.

Prototype

```
DFE_Err Dfe_progFbIOMux
(
    DFE_Handle hDfe,
    DfeFl_FbIoMux ioMux
);
```

Description

Write new FB IO mux.

NOTE, Additional sync needs to be issued to set new values for EQ, gain or NCO in the new Fb channel.

Arguments

hDfe	[in] DFE device handle
ioMux	[in] io mux mapping.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.97 Program FB pre-CB Gain

Program FB pre-CB gain.

Prototype

```
DFE_Err Dfe_progFbGain
(
    DFE_Handle hDfe,
    DfeFl_FbBlk FbBlkId,
    float FbGain[2]
);
```

Description

Write new FB pre-CB gain to shadow memory. The range of gain is -Inf ~ 6.02dB. To check the gain changing through capture buffer, the cb_output_select need to be enabled.

NOTE, Dfe_issueSyncUpdateFbGain should be called later to let hardware copy gains from shadow to working memory.

Arguments

hDfe	[in] DFE device handle
FbBlkId	[in] Fb block Id, 0 ~ 4
FbGain	[in] Array of Fb gain, [0] is real part and [1] is image part.

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_issueSyncUpdateFbGain () should be called later to copy gains to working memory.

6.98 Issue Sync Update Fb pre-CB Gain

Issue sync update FB gain to new values.

Prototype

```
DFE_Err Dfe_issueSyncUpdateFbGain
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Issue sync to copy Fb gains from shadow to working memory.
Dfe_getSyncStatus() should be called later to check if the sync has come.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly

DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Only the FB block which matches with IO mux config will be updated.

6.99 Program CB Node Config

Program CB node configuration.

Prototype

```
DFE_Err Dfe_progCbNodecfg
(
    DFE_Handle hDfe,
    DfeFl_CbNodeCfg *nodeCfg
);
```

Description

Write new CB node configuration for one CB node.

NOTE, CB node ID defined as nodeCfg->cbNode is 0 ~ 8.

Node 0 = DPD input
Node 1 = DPD output
Node 2 = DDUC input (FB output, IQ interleaved)
Node 3 = FB, resampler input or output (FB input, IQ parallel)
Node 4 = CFR block0 input
Node 5 = CFR block1 input
Node 6 = CFR block0 output
Node 7 = CFR block1 output
Node 8 = testbus

Arguments

hDfe	[in] DFE device handle
nodeCfg	[in] CB node configuration

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.100 Program CB Buf Config

Program CB buf configuration.

Prototype

```
DFE_Err Dfe_progCbBufcfg
(
    DFE_Handle hDfe,
    DFE_CbBufCfg *bufCfg
);
```

Description

Write new CB buf configuration for one CB buffer.

NOTE, CB buf ID defined as bufCfg->cbSet.cbBuf is 0 ~ 3.

```
// CB buf configuration
typedef struct
{
    // cb buf mode set
    DfeFl_CbModeSet cbSet;
    // cb buf delay from sync
    Uint32 dly;
    // 0 = 1s/1c mode; 1 = 2s/1c mode
    Uint32 rate_mode;
    // capture buffer A fractional counter length minus 1; range 0-
    // 15; value depends on the relative sampling rates for different buffers
    Uint32 frac_cnt;
    // fractional counter sync select
    Uint32 frac_cnt_ssel;
    // length counter sync select
    Uint32 len_cnt_ssel;
    // cb buf length, upto 8192 complex data
    Uint32 length;
} DFE_CbBufCfg;
```

Arguments

hDfe	[in] DFE device handle
bufCfg	[in] CB buffer configuration

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.101 Arm CB and Issue Sync

Arm CB and issue sync

Prototype

```
DFE_Err Dfe_armCbIssueSync
(
    DFE_Handle hDfe,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Arm CB and issue sync to prepare for capture. The cb buf mode is also changed to capture mode.

Arguments

hDfe	[in] DFE device handle
ssel	[in] sync select to capture

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.102 Get CB Done Status

Get the capture buffer done status

Prototype

```
DFE_Err Dfe_getCbDoneStatus
(
    DFE_Handle hDfe,
    DfeFl_CbArm *cbDoneStatus
);
```

Description

The API gets back the cbDone Status.

Arguments

hDfe	[in] DFE device handle
cbDoneStatus	[out] pointer to cb done status

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constrains

1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running.
3) DFE has loaded target config and completed initialize sequence.

6.103 Read CB Buf

Read CB buf data via CPU.

Prototype

<pre> DFE_Err Dfe_readCbBuf (DFE_Handle hDfe, DfeFl_CbBuf cbBufId, Uint32 cbLength, Uint32 flag_18bit, DfeFl_CbComplexInt *cbTemp, DfeFl_CbStatus *cbStatus, DFE_CbData *cbData); </pre>
--

Description

<p>Read CB buf data and CB status via CPU. It reads DFE registers directly.</p> <p>The API changes to MPU mode and capture the 16MSB or total 18bit buffer based on requirement.</p> <pre> // CB data typedef struct { // I data Uint32 *Idata; // Q data Uint32 *Qdata; } DFE_CbData; </pre>

Arguments

hDfe	[in] DFE device handle
cBufId	[in] Cb Buffer Id, 0 ~ 3
cbLength	[in] Cb length, 0 ~ 8192

flag_18bit	[in] flag to read 18bit buffer 0 = 16bit MSB 1 = 18bit
cbTemp	[in] pointer to the temporary buffer, size 8192*4 bytes
cbStatus	[out] pointer to the cb status
cbData	[out] pointer to the cb data

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

1) hDfe should be a valid handle opened by Dfe_open().
2) DFE PLL and PSCs shall be already up running.
3) DFE has loaded target config and completed initialize sequence.

6.104 Open CB Buf DMA

Open CPP/DMA for reading CB buffer.

Prototype

<pre> DFE_Err Dfe_openCbBufDma (DFE_Handle hDfe, Uint32 flag_18bit, Uint32 cppDmaId, Uint32 cppDescripId[8], Uint32 iqnChnl); </pre>
--

Description

<p>Allocate CPP/DMA channel and descriptor for CB buf reading.</p> <p>Every CB buf has 8192 words for 16bit MSB and 8192 words for 2bit LSB. Each word has I data in bit [31:16] and Q data in bit [15:0].</p> <p>To transfer all 4 CB buffers, there are total 8 descriptors if all 18bit data are needed.</p>

Arguments

hDfe	[in] DFE device handle
flag_18bit	[in] flag to read 18bit buffer 0 = 16bit MSB 1 = 18bit
cppDmaId	[in] CPP/DMA channel Id 0 ~ 31, open with specified channel DFE_FL_CPP_OPEN_ANY, open with any available

	channel
cppDescripId	[in] pointer to CPP/DMA descriptor Id 0 ~ 127, open with specified descriptor DFE_FL_CPP_OPEN_ANY, open with any available descriptor
iqnChnl	[in] IQN2 CTL Ingress channel number, 0 ~ 15

Return Value

DFE_ERR_NONE, if API complete programmed properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_AVAILABLE, if CPP/DMA channel not available
 DFE_ERR_CPP_DESCRIP_NOT_AVAILABLE, if CPP/Descriptor not available
 DFE_ERR_ALREADY_OPENED, if BBTX power meter DMA already opened

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.105 Close CB Buf DMA

Close CB buf DMA and free recourses.

Prototype

```
DFE_Err Dfe_closeCbBufDma
(
    DFE_Handle hDfe
);
```

Description

Close Cb Buf DMA and free resources allocated by Dfe_openCbBufDma().

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openCbBufDma() has called OK.

6.106 Enable CB Buf DMA

Enable CB Buf DMA.

Prototype

```
DFE_Err Dfe_enableCbBufDma
(
    DFE_Handle hDfe
);
```

Description

Enable CPP/DMA for CB Buf by setting dma_ssel to sense MPU sync. MPU sync will trigger CPP/DMA to start transferring.

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openCbBufDma() has called OK.

6.107 Disable CB Buf DMA

Disable CB Buf DMA.

Prototype

```
DFE_Err Dfe_disableCbBufDma
(
    DFE_Handle hDfe
);
```

Description

Disable CB buf DMA by changing dma_ssel not to sense any signal

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openCbBufDma() has called OK.

6.108 Disable All TestBus

Disable all test bus probes.

Prototype

```
DFE_Err Dfe_disableAllTestbus
(
    DFE_Handle hDfe
);
```

Description

DFE has many test probe points scattered over all sub-modules. CB can be used to capture a train of IQ bus signals at the probe.

All test probes "AND together" shares single CB interface. So software should enable no more than one probe at any time.

The API clears all test probes which was programmed before.

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.109 Program DFE GPIO PinMux

Select pin function by programming DFE GPIO pinmux.

Prototype

```
DFE_Err Dfe_progGpioPinMux
(
    DFE_Handle hDfe,
    DfeFl_MiscGpioPin pinId,
    DfeFl_MiscGpioMux muxSel
);
```

Description

Select pin function by programming DFE GPIO pinmux. There're total 18 GPIOs, every pin can be configured to be different function.

Pin Function	Value	Input/Output	Description
NOTHING	0	Input	Input only, not configured to a function
GPIO_SYNC_IN0	1	Input	General sync input 0
GPIO_SYNC_IN1	2	Input	General sync input 1
JESD_SYNC_IN0	3	Input	JESD sync inputs for interfacing to data converters which do not support LVDS syncs
JESD_SYNC_IN1	4	Input	
GPIO_SYNC_OUT0	8	Output	General sync output 0
GPIO_SYNC_OUT1	9	Output	General sync output 0
JESD_SYNC_OUT0	10	Output	JESD sync outputs for interfacing to data converters which do not support LVDS syncs
JESD_SYNC_OUT1	11	Output	
FB_MUX_CNTL0	12	Output	feedback mux control for Marconi 0
FB_MUX_CNTL1	13	Output	
FB_MUX_CNTL2	14	Output	
FB_MUX_CNTL3	15	Output	feedback mux control for Marconi 1
FB_MUX_CNTL4	16	Output	
FB_MUX_CNTL5	17	Output	
DVGA_CNTL0	18	Output	8 bits DVGA controls
DVGA_CNTL1	19	Output	
DVGA_CNTL2	20	Output	
DVGA_CNTL3	21	Output	
DVGA_CNTL4	22	Output	
DVGA_CNTL5	23	Output	
DVGA_CNTL6	24	Output	
DVGA_CNTL7	25	Output	
SYSREF_REQUEST	26	Output	JESD SYSREF request output
MPU_DRIVE	27	Output	Driven by MPU register write

Note: ALL pins (including those set to "nothing") also function as mpu gpio read. If the pin is an output pin, the mpu gpio read will return whatever is being output.

Arguments

hDfe	[in] DFE device handle
pinId	[in] GPIO pin ID
muxSel	[in] select function of the pin

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.110 Set DFE GPIO Sync Out Source

Select sync source for GPIO Sync out pin.

Prototype

```
DFE_Err Dfe_setGpioSyncOutSource
(
    DFE_Handle hDfe,
    Uint32 syncoutId,
    DfeFl_MiscSyncGenSig ssel
);
```

Description

Select sync source for GPIO_SYNC_OUT pin.

Arguments

hDfe	[in] DFE device handle
syncoutId	[in] sync out pin, 0/1
ssel	[in] sync select

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.111 Set DFE GPIO Bank Output

Write GPIO pins in bank mode.

Prototype

```
DFE_Err Dfe_setGpioBankOutput
(
    DFE_Handle hDfe,
```

```
    Uint32 bankOutput
);
```

Description

Write MPU GPIO Drive register to drive pin output. For pins configured as MPU_DRIVE, DFE drives those pins per bankOutput value. For other pins, it does no effective.

There're total 18 GPIOs, one-to-one mapped to 18 LSBs of bankOutput.

Arguments

hDfe	[in] DFE device handle
bankOutput	[in] pin output bitmap value, bit0 driving pin0, bit1 driving pin1, ..., bit17 driving pin17. <ul style="list-style-type: none"> 0 = drive low 1 = drive high

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.112 Get DFE GPIO Bank Input

Read GPIO pins in bank mode.

Prototype

```
DFE_Err Dfe_getGpioBankInput
(
    DFE_Handle hDfe,
    Uint32 *bankInput
);
```

Description

Read current input status of GPIO pins.

There're total 18 GPIOs, one-to-one mapped to 18 LSBs of *bankInput.

Arguments

hDfe	[in] DFE device handle
bankInput	[out] pin input bitmap value, bit0 is for pin0, bit1 is for pin1, ..., bit17 is for pin17. <ul style="list-style-type: none"> 0 = input low

	<ul style="list-style-type: none"> 1 = input high
--	--

Return Value

DFE_ERR_NONE, if API complete properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_HW_QUERY, if CSL GetHwStatus() failed
 DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.113 Open Generic IO DMA

Open CPP/DMA for Generic IO.

Prototype

```
DFE_Err Dfe_openGenericDma
(
    DFE_Handle hDfe,
    Uint32 cppDmaId,
    Uint32 iqnChnlDl,
    Uint32 iqnChnlUl
);
```

Description

Open and allocate resources for generic IO DMA, which can be used to read/write DFE registers and memories.

Generic IO DMA is using CPP/DMA embedded address mode. The DMA descriptor is embedded at the beginning of the packet.

Generic IO DMA doesn't support linked transfers. Each DMA packet can transfer data size from one 32-bits word to 16K 32-bits words.

Upon the API returns DFE_ERR_NONE, generic IO DMA has already been enabled and ready for use. Data available on iqnChnlDl starts CPP/DMA.

When generic IO DMA already opened, call this API again will get error DFE_ERR_ALREADY_OPENED.

Arguments

hDfe	[in] DFE device handle
cppDmaId	[in] CPP/DMA channel Id 0 ~ 31, open with specified channel DFE_FL_CPP_OPEN_ANY, open with any available channel
iqnChnlDl	[in] IQN2 CTL Egress channel number, 0 ~ 15
iqnChnlUl	[in] IQN2 CTL Ingress channel number, 0 ~ 15

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_AVAILABLE, if CPP/DMA channel not available
DFE_ERR_ALREADY_OPENED, if generic IO DMA already opened
DFE_ERR_HW_CTRL, if CSL HwControl() failed
```

Constrains

- 4) hDfe should be a valid handle opened by Dfe_open().
- 1) DFE PLL and PSCs shall be already up running.
- 2) DFE has loaded target config and completed initialize sequence.

6.114 Close Generic IO DMA

Close Cpp/DMA for Generic IO

Prototype

```
DFE_Err Dfe_closeGenericDma
(
    DFE_Handle hDfe
);
```

Description

Close generic IO DMA and free resources allocated by Dfe_openGenericDma().

Arguments

hDfe	[in] DFE device handle
------	------------------------

Return Value

```
DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid
```

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.
- 4) Dfe_openGenericDma() has called OK.

6.115 Prepare Generic DMA Embedded Header

Prepare CPP/DMA embedded header for generic IO request.

Prototype

```
DFE_Err Dfe_prepareGenericDmaHeader
(
    DFE_Handle hDfe,
```

```

    Uint32 *header
    DFE_GenericDmaReadWriteMode rwMode,
    Uint32 offsetAddrInDref,
    Uint32 sizeOrData
);

```

Description

The API helps building CPP/DMA embedded header that will be sent together with data payload block. The size of the header is fixed 16 bytes, then followed data payload block.

The API supports three read/write modes,

- `DFE_GENERIC_DMA_RW_MODE_WRITE_SINGLE_WORD`, write single 32-bits word to DFE. The `sizeOrData` has the sending value that will be built into the embedded header. There's no other payload to be sent.
- `DFE_GENERIC_DMA_RW_MODE_WRITE_MULTI_WORDS`, write multiple 32-bits words to DFE. The `sizeOrData` has the number of words of data payload, which is immediately following the header. If number of the payload words not multiples of 4, zero shall be padded to fill the remaining words in the final line.
- `DFE_GENERIC_DMA_RW_MODE_READ`, read from DFE. The `sizeOrData` has the number of words of read data, which will be received from IQN2 CTL ingress channel, `iqnChnlUl`, specified when `Dfe_openGenericDma()`.

For writing, `offsetAddrInDref` is the destination 26-bit address within DFE scope; for reading, it is the source 26-bit address within DFE scope.

Arguments

<code>hDfe</code>	[in] DFE device handle
<code>header</code>	[out] pointer to packet header
<code>rwMode</code>	[in] generic IO read/write mode
<code>offsetAddrInDref</code>	[in] 26-bit address within DFE scope.
<code>sizeOrData</code>	[in] size or data. For single word writing, it is the data value; for other modes, it is number of payload words in 32-bits.

Return Value

```

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_CPP_DMA_NOT_VALID, if CPP/DMA handle not valid

```

Constrains

- 1) `hDfe` should be a valid handle opened by `Dfe_open()`.
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.116 Enable Lut toggle

Enable Lut toggle for one dpd block.

Prototype

```
DFE_Err Dfe_enableToggle  
(  
    DFE_Handle hDfe,  
    Uint32 blkId  
);
```

Description

Enable the Lut toggle for one dpd block

Arguments

hDfe	[in] DFE device handle
blkId	[in] block Id [0:3]

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.117 SetSyncSel for Lut

Set sync selection for one dpd block.

Prototype

```
DFE_Err Dfe_setSyncSel  
(  
    DFE_Handle hDfe,  
    Uint32 blkId,  
    Uint32 synch  
);
```

Description

Set synch selection for one dpd block

Arguments

hDfe	[in] DFE device handle
blkId	[in] block Id [0:3]
Sync	[in] Sync: 0: f_sync 1: c_sync 2: f_sync c_sync 3: the 'combined sync' generated internally based on 'sync_b' and sync from 'poly2LUT'

Return Value

DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constrains

1) hDfe should be a valid handle opened by Dfe_open(). 2) DFE PLL and PSCs shall be already up running. 3) DFE has loaded target config and completed initialize sequence.
--

6.118 Issue Sync Update Lut

Issue sync update Lut to new values.

Prototype

<pre> DFE_Err Dfe_issueSyncUpdateLut (DFE_Handle hDfe, Uint32 numBlks, Uint32 blkId[], DfeFl_MiscSyncGenSig ssel); </pre>

Description

Issue sync to switch Lut table for one block.

Arguments

hDfe	[in] DFE device handle
numBlks	[in] number of blocks which will be updated, [1:4]
blkId[]	[in] array of block Ids which will be updated, [0:3]
ssel	[in] sync select to copy gains from shadow to working memory.

Return Value

DFE_ERR_NONE, if API complete properly DFE_ERR_INVALID_HANDLE, if hDfe is NULL DFE_ERR_HW_CTRL, if CSL HwControl() failed

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.119 Get Current Lut memory index

Get current Lut memory index for one dpd block.

Prototype

```
DFE_Err Dfe_setSyncSel
(
    DFE_Handle hDfe,
    Uint32 blkId,
    Uint32 *LutIdx
);
```

Description

Get the current Lut memory index for one dpd block. There are total 6 rows in one dpd block. There are total 3 cells in one row. Each cell will return the Lut memory index. 0 means the datapath is using LUT0 (bottom half of memory), 1 means the datapath is using LUT1 (top half of memory).

The LutIdx will store the results in bit masking format.

MSB		B23	B22	B21	B20	...	B3	B2	B1	B0
		Not used	Row5, cell12	Row5, cell11	Row5, cell10	...	Not used	Row0, cell12	Row0, cell11	Row0, cell10

Arguments

hDfe	[in] DFE device handle
blkId	[in] block Id [0:3]
LutIdx	[out] current Lut memory index

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.120 Program Lut table

Program Lut table for one dpd cell.

Prototype

```
DFE_Err Dfe_setSyncSel
(
    DFE_Handle hDfe,
    Uint32 blkId,
    Uint32 rowId,
    Uint32 cellId,
    DFE_DpdData *DpdData
);
```

Description

Program Lut table for one cell. The dpd lut will be used in dpd datapath after issue sync to update lut.

```
typedef struct _DFE_DpdData
{
    // lutGain
    DfeFl_DpdComplexInt lutGain;
    // lutSlope
    DfeFl_DpdComplexInt lutSlope;
} DFE_DpdData;
```

Arguments

hDfe	[in] DFE device handle
blkId	[in] block Id [0:3]
rowId	[in] row Id [0:5]
cellId	[in] cell Id [0:2]
DpdData	[in] pointer to the dpd data

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constraints

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.121 Get Dpd configuration

Read back dpd configuration.

Prototype

```
DFE_Err Dfe_getDpdCfg
(
    DFE_Handle hDfe,
    DFE_DpdCfg *dpdCfg
);
```

Description

Read back the dpd configuration

```
typedef struct _DFE_DpdCfg
{
    // subchip mode
    uint32_t subchip_mode;
    // subsample
    uint32_t subsample;
    // dpd input scale
    uint32_t dpdInputScale;
    // x2 sqrt
    uint32_t x2_sqrt;
} DFE_DpdCfg;
```

Arguments

hDfe	[in] DFE device handle
dpdCfg	[out] pointer to the dpd configure

Return Value

DFE_ERR_NONE, if API complete properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL

Constrains

- 1) hDfe should be a valid handle opened by Dfe_open().
- 2) DFE PLL and PSCs shall be already up running.
- 3) DFE has loaded target config and completed initialize sequence.

6.122 Load DPDA image

Load image to DPDA instruction RAM and lookup tables. Reset and initialize DPDA.

Prototype

```
DFE_Err Dfe_loadDpda
(
    DFE_Handle hDfe,
    Uint32 imageSize,
    DFE_RegPair *imagePtr
)
```

Description

The API

- disables the arbiter
- resets DPDA
- forces DPDA into the IDLE state
- clears interrupt mask and status registers
- clears command, test and debug registers
- clears the scalar and IG register files
- clears the stack
- loads the given image
- releases DPDA from reset

```
// (addr, data) register pair
typedef struct
{
    // offset address from DFE base address
    Uint32 addr;
    // data to/from addr
    Uint32 data;
} DFE_RegPair;
```

Arguments

hDfe	[in] DFE device handle
igId	[in] Identifies IG register, 0 ~ 63
igPtr	[out] Points to value

Return Value

DFE_ERR_NONE, if API complete programmed properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

1) hDfe should be a valid handle opened by Dfe_open().

6.123 Read DPDA IG register

Read value from DPDA IG register file.

Prototype

```
DFE_Err Dfe_wroteDpdaIg
(
    DFE_Handle hDfe,
    Uint8 igId,
    Uint32 *igPtr
)
```

Description

The API reads a value from the given IG register.

Arguments

hDfe	[in] DFE device handle
igId	[in] Identifies IG register, 0 ~ 63
igPtr	[out] Points to value

Return Value

DFE_ERR_NONE, if API complete programmed properly
 DFE_ERR_INVALID_HANDLE, if hDfe is NULL
 DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

2) hDfe should be a valid handle opened by Dfe_open().

6.124 Read DPDA parameters

Read complex parameters from DPDA solution RAM.

Prototype

```
DFE_Err Dfe_readDpdaParams
(
    DFE_Handle hDfe,
    Uint16 lineId,
    Uint16 lineNum,
    float *paramTbl
)
```

Description

The API reads all parameters stored in the given lines from the solution RAM. Note that each line stores 24 complex parameters. It also converts the I and Q parts of each parameter from the custom floating point format used by the DPDA and re-orders the parameters as follows:

```
for (i = 0; i < lineNum; i++) {
    for (j = 0; j < 8; j++) {
        for (k = 0; k < 3; k++) {
            paramTbl[24*2*i+8*2*k+2*j] = tmpTbl[24*2*i+3*2*j+k*2];
            paramTbl[24*2*i+8*2*k+2*j+1] = tmpTbl[24*2*i+3*2*j+k*2+1];
        }
    }
}
```

Arguments

hDfe	[in] DFE device handle
lineId	[in] Identifies first line, 0 ~ 767
lineNum	[in] Specifies number of lines, 0 ~ 767
paramTbl	[out] Points to table of parameters. Even locations store the I parts of the corresponding parameters and odd locations the Q parts.

Return Value

```
DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters
```

Constraints

3) hDfe should be a valid handle opened by Dfe_open().

6.125 Read DPDA scalar register

Read complex value from DPDA scalar register file.

Prototype

```
DFE_Err Dfe_writeDpdaScalar
(
    DFE_Handle hDfe,
    Uint8 scalarId,
```

```
float *iScalarPtr,
float *qScalarPtr
)
```

Description

The API reads a complex value from the given scalar register and it converts the I and Q parts of the complex value from the custom floating point format used by DPDA.

Arguments

hDfe	[in] DFE device handle
scalarId	[in] Identifies scalar register, 0 ~ 63
iScalarPtr	[in] Points to I part of scalar value
qScalarPtr	[in] Points to Q part of scalar value

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

1) hDfe should be a valid handle opened by Dfe_open().

6.126 Start DPDA

Start DPDA.

Prototype

```
DFE_Err Dfe_startDpda
(
    DFE_Handle hDfe,
    Uint16 startAddress
)
```

Description

The API

- clears interrupt flag in the command register
- clears idle status, read status and processed status
- writes the given start address to the command register
- sets the interrupt flag in the command register
- polls the idle status until the DPDA returns to the idle state

Arguments

hDfe	[in] DFE device handle
Uint16	[in] Specifies start address in the instruction RAM, 0 ~ 4095

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

1) hDfe should be a valid handle opened by Dfe_open().

6.127 Write DPDA IG register

Write value to DPDA IG register file.

Prototype

```
DFE_Err Dfe_writeDpdaIg
(
    DFE_Handle hDfe,
    Uint8 igId,
    Uint32 ig
)
```

Description

The API writes the given value to the given IG register.

Arguments

hDfe	[in] DFE device handle
igId	[in] Identifies IG register, 0 ~ 63
ig	[in] Value

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

2) hDfe should be a valid handle opened by Dfe_open().

6.128 Write DPDA samples

Write samples to capture buffer. DPDA will read them from capture buffer.

Prototype

```
DFE_Err Dfe_writeDpdaSamples
(
    DFE_Handle hDfe,
    DfeFl_CbBuf cbBufId,
    Uint8 fbFlag,
    Uint16 cbLength,
    DfeFl_CbComplexInt *cbTemp,
    DFE_CbData *cbData
)
```

Description

For the given CB buffer, the API writes each given sample to the 16MSB of the corresponding 18-bit CB sample and clears the 2 LSB. It sets the buffer in fine mode and tags it as storing reference or feedback samples. It also makes sure that the buffer won't skip chunks.

```
// CB complex sample
typedef struct
{
    // real value
    Uint16 real;
    // image value
    Uint16 imag;
} DfeFl_CbComplexInt;

// CB data
typedef struct
{
    // I data
    Uint32 *Idata;
    // Q data
    Uint32 *Qdata;
} DFE_CbData;
```

Arguments

hDfe	[in] DFE device handle
cBufId	[in] Identifies CB buffer, 0 ~ 3
fbFlag	[in] Indicates whether buffer is used for capturing feedback signal 0 = no feedback signal 1 = feedback signal
cbLength	[in] Specifies number of samples, 0 ~ 8192
cbTemp	[in] Points to the temporary buffer, size 8192*4 bytes
cbData	[in] Points to the cb data

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

1) hDfe should be a valid handle opened by Dfe_open().

6.129 Write DPDA scalar register

Write complex value to DPDA scalar register file.

Prototype

```
DFE_Err Dfe_writeDpdaScalar
(
    DFE_Handle hDfe,
    Uint8 scalarId,
    float iScalar,
    float qScalar
)
```

Description

The API converts the I and Q parts of the given complex value to the

custom floating point format used by DPDA and writes them to the given scalar register.

Arguments

hDfe	[in] DFE device handle
scalarId	[in] Identifies scalar register, 0 ~ 63
iScalar	[in] I part of scalar value
qScalar	[in] Q part of scalar value

Return Value

DFE_ERR_NONE, if API complete programmed properly
DFE_ERR_INVALID_HANDLE, if hDfe is NULL
DFE_ERR_INVALID_PARAMS, if invalid parameters

Constraints

2) hDfe should be a valid handle opened by Dfe_open().

7 Integration

7.1 OSAL

The OSAL is the operating system abstraction layer which is used to port DFE LLD to a specific OS. The OSAL callouts are implemented in the “dfe_osal.h” header file and need to be ported by the application developers to their specific operating system.

7.1.1 Logging API

Internally DFE LLD uses the Dfe_osalLog macro to perform all logging operations. The OSAL adaptation layer ports this macro to the following API prototype:-

```
void Osal_dfeLog( char* fmt, ... )
```

The parameter ‘fmt’ is a printf style formatted string. This should only be defined and used for debugging purposes.

7.2 Integration on ARM Linux

DFE LLD works on the top of DFE_FL Function Level (FL) library. Both libraries are running from Linux user space. DFE_FL FL API dfeFl_GetBaseAddress() should be customized to use mmap() in un-cached mode to open the whole 48M-bytes window of DFE.

To move big chunks of data into/out of DFE memories, such as capture buffers and BBTX/RX power meter results, the application should use DFE LLD together with UDMA/IQN2 user mode driver.

7.3 Integration on DSP SysBios

Both DFE LLD and CSL library are packaged in PDK. There's no customization needed for CSL.

To move big chunks of data into/out of DFE memories, such as capture buffers and BBTX/RX power meter results, the application should use DFE LLD together with QMSS/CPPI/IQN2 LLDs.

8 Future Extensions

- Support of programming DPD LUTs
- Support of programming Poly2Lut
- Support of programming TX filter coefficients
- Support of programming BeAGC
- Support of programming FeAGC