TEXAS INSTRUMENTS
20450 Century Boulevard
Germantown, MD 20874
Fax: (301) 515-7954

Fapi Tracing Library

Software Design Specification (SDS)

Revision C

QRSA000XXXX

September 24, 2015

Revision Record	
Document Title: **Software Design Specification**	
Revision	Description of Change
A	1. Initial Release
B	2. Feature update to support FAPI trace capture
C	3. FAPI 2.0 support for used with Syslib 4.0
D	4. FAPI MemoryLogging support

Note: Be sure the Revision of this document matches the QRSA 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 FAPI TRACING LIBRARY 1

4.1 INTRODUCTION 1
4.2 FAPI MESSAGE TRACING 2
4.2.1 *FAPI tracing Instance creation 2*
4.2.2 *FAPI tracing initialization 2*
4.2.3 *Enable FAPI tracing 4*
4.2.4 *FAPI Message Tracing 4*
4.2.4.1 Sequence for sending FAPI Message from Stack to PHY 4
4.2.4.2 Sequence for receiving FAPI Message from PHY 5
4.2.5 *Data Tracing 7*
4.2.6 *FAPI tracing Library version 7*
4.2.7 *FAPI Tracing Memory Logging 7*
4.3 FAPI MESSAGE COMPRESSION 7

1 Scope

This document describes the functionality of FAPI Tracing Library which allows applications to send TI FAPI messages exchanged between LTE stack and LTE PHY through Ethernet interface for offline analysis. It also provides APIs to compress FAPI messages constructed on LTE stack side.

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	Syslib User Guide		SYSLIB User Guide
2	FAPI Tracing Documentation		FAPI Tracing Doxygen API documentation
3	LTE eNB L1 API Definition V2.1		L1 API definition from Small Cell Forum
4	Ti_fapi.h		FAPI interface header file between Stack and PHY

Table 2.1 Referenced materials

3 Definitions

Acronym	Description
API	Application Programming Interface
DSP	Digital Signal Processor
NetFP	Network Fast Path
FAPI	Femto Forum API
PHY	LTE Physical module running on DSP cores

Table 3.1 Definitions

4 FAPI Tracing Library

4.1 Introduction

The FAPI tracing library provides two functions:

1. Functionality to trace FAPI messages exchanged between Stack and LTE PHY.
2. Functionality to compress FAPI messages sent from LTE stack to LTE PHY.

FAPI tracing packets are sent through Ethernet interface to a PC running Wireshark FAPI dissector for offline analysis. FAPI Tracing library can ship out both FAPI messages and Uplink/Downlink payload data inside TX.Request and RX_ULSCH.indication.

NOTE: The DDR CAPTURE mode is not supported by the current build of the FAPI Tracing library.

4.2 FAPI message Tracing

4.2.1 FAPI tracing Instance creation

The `FapiTracing_createInstance()` API should be called to create an instance of the library before any further FAPI Tracing API can be called. The API takes the parameters defined in FapiTracing_InstCfg. The following table lists the parameters. Refer to Doxygen documentation for the latest parameter list for the API. The API returns the FAPI Tracing instance handle used by most FAPI tracing APIs.

Name	Description	Notes
malloc	Function pointer for malloc OSAL function	Pointer to function used to provide memory malloc capability to the FAPI Tracing instance. Refer to section **Error! Reference source not found.** or further details.
`free`	Function pointer for free OSAL function	Pointer to function used to provide memory free capability to the FAPI Tracing instance. Refer to section **Error! Reference source not found.** or further details.

4.2.2 FAPI tracing initialization

`FapiTracing_init()` API should be called to initialize the FAPI tracing framework before any FAPI tracing can be done. This API takes parameters defined in Fapi_TracingCfg. The following table lists main parameters, always refer to the Doxygen for the latest parameter list of the API.

Name	Description	Notes
`isEnabled`	Flag to enable FAPI tracing	It can be changed through `FapiTracing_configure()`
`isDataTracingEnabled`	Flag to enable FAPI data tracing	It can be changed through `FapiTracing_configure()`
`netfpClientHandle`	NetFP Client Handle	Application needs to create a NetFP client for use by the FAPI tracing library.

ingressFastPathHandle	NetFP Ingress Fast Path handle	Application needs to create ingress fast path and passes the handle to FAPI tracing library.
egressFastPathHandle	NetFP Egress Fast Path handle	Application needs to create egress fast path and passes the handle to FAPI tracing library.
pktlibInstHandle	PKTLIB Instance Handle	Application needs to create a PKTLIB Instance for use by the FAPI tracing library. This instance should be associated with the following heapHandle.
heapHandle	Heap handle used for sending out packets	Fapi tracing library requires zero buffer Heap for shipping tracing packets. This heap should be created by application (stack). If FAPI trace capture is enabled, packets with data buffers (128 bytes) are also needed. The data buffers are used for time stamp .
destUDPPort	Destination UDP port number	This port number will be used to create socket for sending out tracing packets. Application needs to provide a proper non-zero port number.
packetDSCP	Packet priority	
transportType	Enum value to determine how FAPI packets are sent out	Use TRACING_TRANSPORT_NETFP to send through NETCP; Use TRACING_TRANSPORT_CAPTURE to save in DDR memory

NOTE: transportType = TRACING_TRANSPORT_CAPTURE is not supported by the current build of the FAPI Tracing library.

4.2.3 Enable FAPI tracing

FAPI tracing and FAPI data tracing can be enable/disable through **`FapiTracing_configure().`**

4.2.4 FAPI Message Tracing

FAPI message tracing is fulfilled by calling **`FapiTracing_trace()`** API. FAPI message compression is not done inside this function. The trace length is the FAPI message packet length set from LTE Stack or PHY. The preferred sequence for FAPI message tracing are described in the following two sections.

4.2.4.1 Sequence for sending FAPI Message from Stack to PHY

Steps for FAPI tracing message from stack to PHY:

1. Stack prepares FAPI message

```
/* allocate memory from the FAPI Heap. */
 ptrFAPIPkt = Pktlib_allocPacket(pktlibInstHandle, gTiFapiMsgHeapDdr,sizeof(TiFapi_nonTlvMsg_s));

/* Fill in UL_CONFIG message */
 Application_prepareULConfigMsg(ptrFAPIPkt);
```

2. Stack calls FAPI compression function to get compressed message body length
3. Stack updates FAPI header "msgLen" with the compressed length

```
/* Get the FAPI message */
Pktlib_getDataBuffer(ptrFAPIPkt, (uint8_t**)&ptrFAPIMessage, &messageLength);

/* Compress the packet and corrected the msglen in FAPI header */
 ptrFAPIMessage->msgLen  = FapiTracing_compressUlConfig((void *)ptrFAPIMessage);
```

4. Stack writes back FAPI message data buffer and packet with (compressed length + FAPI header length)

```
/* Includes FAPI message header for the total message length. */
messageLength = sizeof(TiFapi_nonTlvMsgHeader_s) + ptrFAPIMessage->msgLen;
Pktlib_setPacketLen(ptrFAPIPkt, messageLength);

/* Writeback the data buffer */
appWritebackBuffer(ptrFAPIMessage, messageLength);
```

5. Stack calls **FapiTracing_trace()** to ship out tracing message

```
   /* Pass the message to the FAPI Tracing Module, tracing packet will be shipped out
    * via NetFP fast path.*/
   if(FapiTracing_trace(fapiTracingInstHandle, ptrFAPIPkt, &errCode) < 0)
      System_printf("Error: FapiTracing for packet(msgid=%x) failed, errCode=%d\n",
msgType, errCode);
```

6. Stack calls msgcom_putMessage() to send FAPI message to PHY

```
    /* Send the message out to PHY. */
    Msgcom_putMessage(fapiChannel, (MsgCom_Buffer*)ptrFAPIPkt);
```

4.2.4.2 Sequence for receiving FAPI Message from PHY

There are two ways to do FAPI tracing for received FAPI messages from PHY: send FAPI tracing packet first and then process the FAPI message in stack; Or Process FAPI message first and then do FAPI tracing.

Method 1: FAPI Tracing first

1. Stack receives FAPI message from PHY by calling msgcom_getMessage()

```
    /* Receive FAPI message from msgcom channel. */
    Msgcom_getMessage (fapiChannelHandle, (MsgCom_Buffer**)&ptrFAPIPkt);
```

2. Stack invalidates FAPI message and data buffer if it exists

```
    /* Get the actual FAPI Message. */
    Pktlib_getDataBuffer(ptrFAPIPkt, (uint8_t**)&ptrFAPIMessage, &fapiMessageLen);

    /* The actual length of the FAPI message is stored in the packet length */
    fapiMessageLen = Pktlib_getPacketLen(ptrFAPIPkt);

    /* Invalidate the FAPI Message */
    appInvalidateBuffer(ptrFAPIMessage, fapiMessageLen);
```

3. Stack calls **FapiTracing_trace()** to ship out tracing message

```
    /* Send the FAPI Packet to the Tracer. */
    if(FapiTracing_trace(fapiTracingInstHandle, ptrFAPIPkt, &errCode) < 0)
        System_printf("Error: FapiTracing for packet(msgid=%x) failed,
errCode=%d\n", ptrFAPIMessage->msgId, errCode);
```

4. Stack processes FAPI message

```
    /* Process FAPI message */
    Application_ProcessFAPIMsg(ptrFAPIPkt);
```

5. Stack frees FAPI message and buffer(s)
   Note: It is up to Stack's implementation to decide when to free messages and data buffers. It can be freed in MAC or RLC layer also.

Method 2: process FAPI message first

1. Stack receives FAPI message from PHY by calling msgcom_getMessage()

```
    /* Receive FAPI message from msgcom channel. */
    Msgcom_getMessage (fapiChannelHandle, (MsgCom_Buffer**)&ptrFAPIPkt);
```

2. Stack invalidates FAPI message

```
    /* Get the actual FAPI Message. */
    Pktlib_getDataBuffer(ptrFAPIPkt, (uint8_t**)&ptrFAPIMessage, &fapiMessageLen);

    /* The actual length of the FAPI message is stored in the packet length */
    fapiMessageLen = Pktlib_getPacketLen(ptrFAPIPkt);

    /* Invalidate the FAPI Message */
    appInvalidateBuffer(ptrFAPIMessage, fapiMessageLen);
```

3. Stack processes FAPI message

```
    /* Process FAPI message */
    Application_ProcessFAPIMsg(ptrFAPIPkt);
```

Please note that, since FAPI tracing has not finished yet, FAPI message and data buffer cannot be free at this point.

4. Stack calls **FapiTracing_trace()** to ship out tracing message

```
    /* Send the FAPI Packet to the Tracer. */
    if(FapiTracing_trace(fapiTracingInstHandle, ptrFAPIPkt, &errCode) < 0)
        System_printf("Error: FapiTracing for packet(msgid=%x) failed,
errCode=%d\n", ptrFAPIMessage->msgId, errCode);
```

5. Stack frees FAPI message and buffer(s) – based on Stack's implementation, FAPI message and buffer can be freed in MAC or RLC layer

4.2.5 Data Tracing

FAPI data tracing is enabled through **`FapiTracing_init()`** during initialization, or through **`FapiTracing_configure()`** at run time. When data tracing is enabled, data buffers included in TX.Request or RX_ULSCH.indication message will also be shipped out through NetFP fast path. Wireshark FAPI dissector can further dissect these buffers, such as MAC PDUs. In order to show data buffer properly, the following rules need to be followed by application:

1. For TX.Request message, data buffer should be written back before calling **`FapiTracing_trace().`**
2. For Ulsch indication message, data buffer should be invalidated before calling **`FapiTracing_trace().`**

4.2.6 FAPI tracing Library version

In order for FAPI dissector to work properly, it is required to match FAPI tracing library and Wireshark FAPI dissector version.
`FapiTracing_getVersion()` can be used to get FAPI tracing library version.

4.2.7 FAPI Tracing Memory Logging

FAPI logs can be saved in Memory through a MEMLOG channel and socket post routing hook function. Example of FAPI Tracing in memory is shown in SYSLIB demo apps.
Here are the steps for FAPI Tracing Memory Logging:

1. Create a MEMLOG channel with required resources.
2. Create a timestamp packet heap.
3. Create a FAPI capture Hook function for socket post routing that does the following:
    a. Allocate timestamp packet from a timestamp packet heap.
    b. Get the current timestamp and save in timestamp packet.
    c. Merge the timestamp packet with the FAPI tracing packet.
    d. Call MEMLOG API to save the packet in memory.
    e. Return Netfp_HookReturn_STOLEN.
4. Pass the FAPI capture Hook function to Fapi Tracing lib during FapiTracing_init()

At runtime, NetFP socket interface attaches the ETH/IP header to the tracing packet. Before the packet is shipped out to NETCP, it will call FAPI capture Hook function to save the packet in Memory.

Post Processing script is used to convert the memory dump to Wireshark file format and re-order the packets according to the timestamp.

4.3 FAPI message compression

The goal of compression is to remove unused data fields from FAPI message. With compression, the packet length is reduced; hence it is more efficient for cache operations and it also saves Ethernet bandwidth for FAPI message tracing.

The compression APIs are provided to the messages generated from Stack side, since messages from PHY are already compressed inside LTE PHY. The following is the list of FAPI messages that compression is supported:

1. DL_CONFIG.request message
2. UL_CONFIG.request message
3. HI_DCI0.request message
4. TX.request message
5. RX_ULSCH.indication message

FAPI message compression function takes TiFapi_nonTlvMsg_s pointer as input; it returns the message body length which does not include FAPI header length.