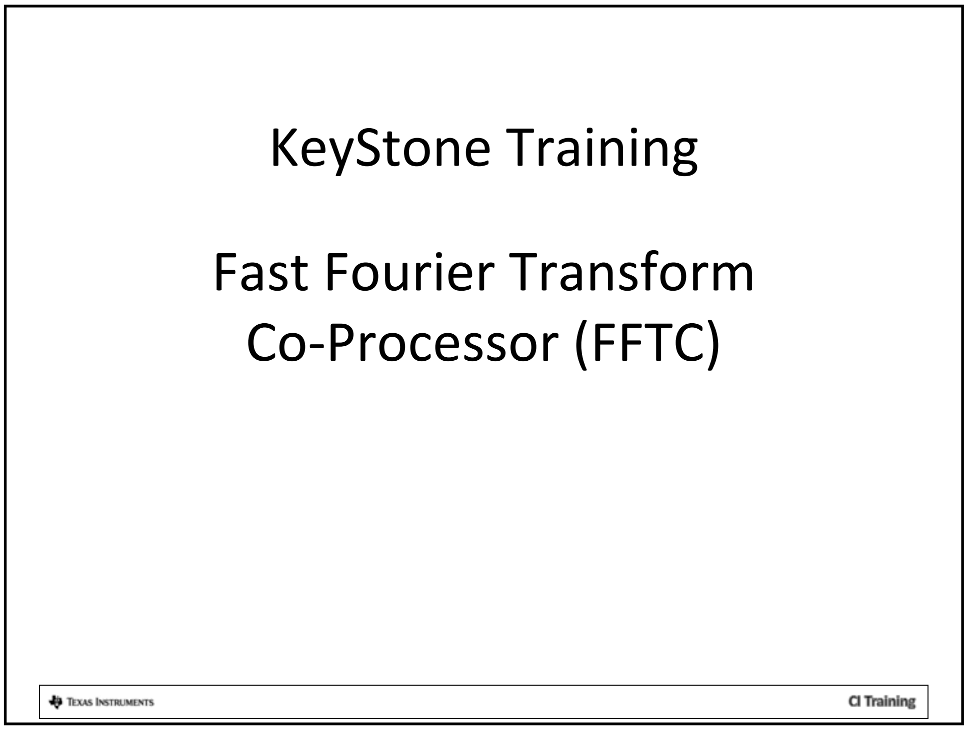
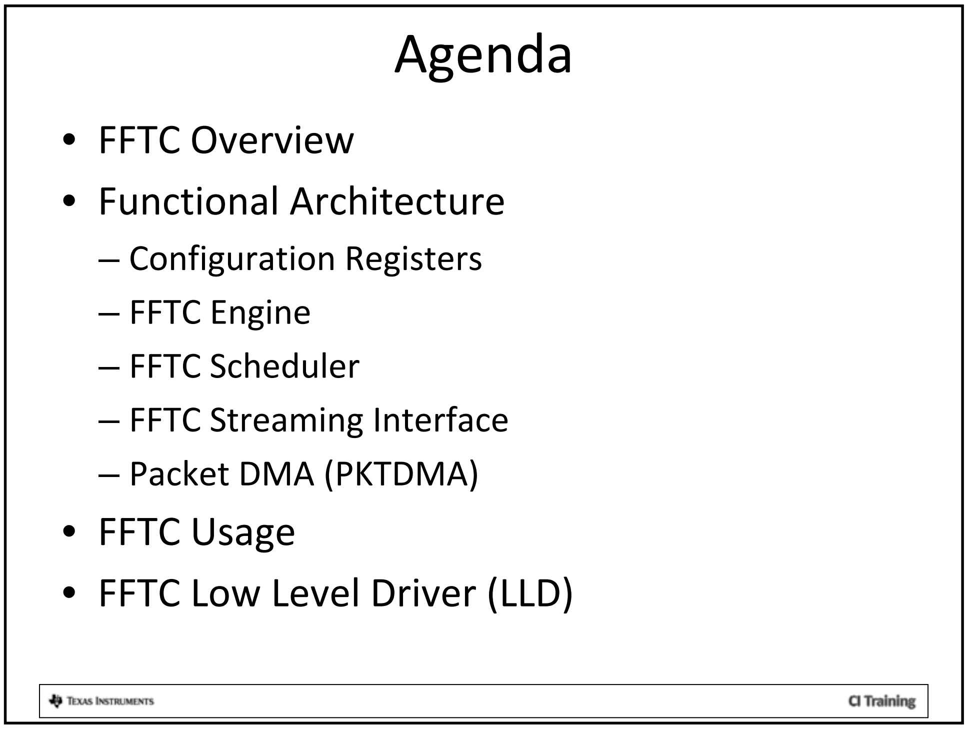


KeyStone Training

Fast Fourier Transform Co-Processor (FFTC)

TEXAS INSTRUMENTS CI Training



Agenda

- FFTC Overview
- Functional Architecture
  - – Configuration Registers
  - – FFTC Engine
  - – FFTC Scheduler
  - – FFTC Streaming Interface
  - – Packet DMA (PKTDMA)
- FFTC Usage
- FFTC Low Level Driver (LLD)

TEXAS INSTRUMENTS CI Training

FFTC Overview

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA)
- FFTC Usage
- FFTC Low-level Driver
- Summary

What is FFTC?

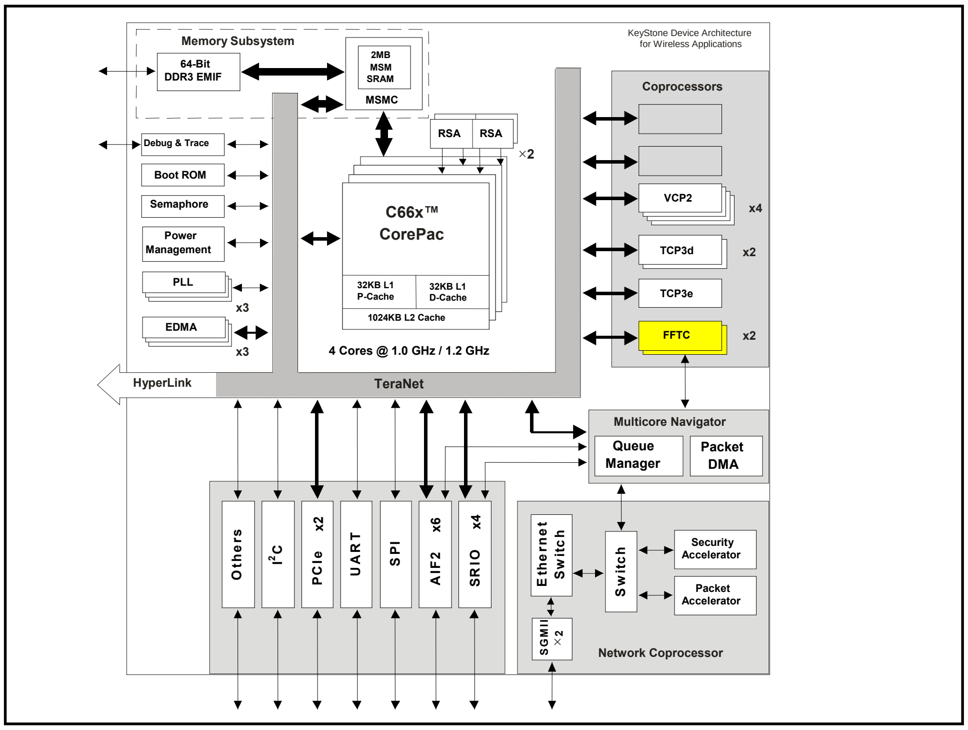
- The FFTC is an accelerator that can be used to perform FFT and Inverse FFT (IFFT) on data.
- The FFTC has been designed to be compatible with various OFDM-based wireless standards like WiMax and LTE.
- The Packet DMA (PKTDMA) is used to move data in and out of the FFTC module.
- The FFTC supports four input (Tx) queues that are serviced in a round-robin fashion.
- Using the FFTC to perform computations that otherwise would have been done in software frees up CPU cycles for other tasks.

FFTC Features

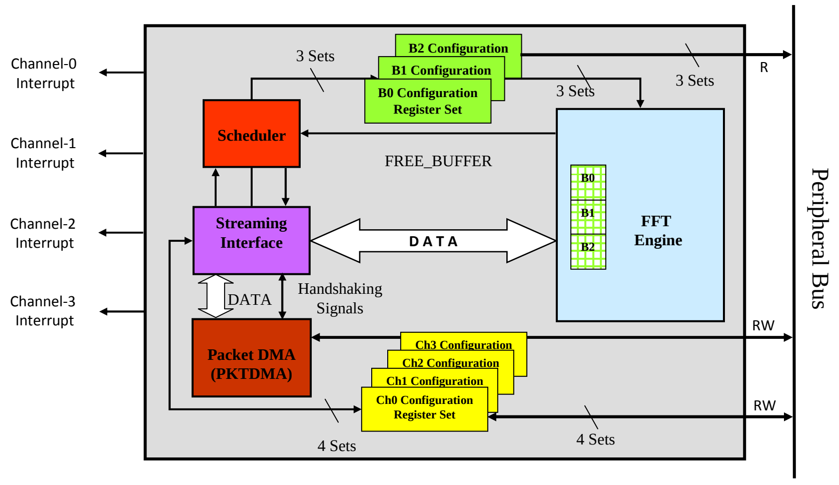
- Provides algorithms for both FFT and IFFT
- Multiple block sizes:
  - Maximum 8192
  - All LTE DFT (Long Term Evolution Discrete Fourier Transform) sizes
- LTE 7.5 kHz frequency shift
- 16 bits I/ 16 bits Q input and output – block floating point output
- Dynamic and programmable scaling modes
  - Dynamic scaling mode returns block exponent
- Support for left-right FFT shift (switch the left/right halves)
- Support for variable FFT shift
  - For OFDM (Orthogonal Frequency Division Multiplexing) downlink, supports data format with DC subcarrier in the middle of the subcarriers
- Support for cyclic prefix
  - Addition and removal
  - Any length supported
- Three-buffer design allows for back-to-back computations
- 128-bit, CPU/3, full-duplex VBUS connection
- Input data scaling with shift eliminates the need for front-end digital AGC (Automatic Gain Control)
- Output data scaling

Functional Architecture

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)



FFTC Functional Block Diagram



Configuration Registers

FFTC Configuration Registers

- The FFTC maintains four sets of configuration registers:
  - These register sets are identical and each set is made up of five registers.
  - Each register set corresponds to one of the four Tx channels delivering the PKTDMA data to the FFTC.
  - Each set specifies the configuration that needs to be applied to the data originating from the respective channels.
- The configuration registers control the following properties:
  - Clipping detection
  - Destination queue for the PKTDMA Rx packet
  - Scaling properties
  - Shifting properties
  - Cyclic prefix parameters
  - LTE frequency-shift parameters

Configuration Methods

- There are two methods for providing FFTC configuration:
  - **Direct Configuration** allows the application to write settings directly to the FFTC configuration registers.
  - **Packet DMA Configuration** allows configuration registers to be set using Protocol-Specific (PS) information included with the PKTDMA packet sent to the FFTC engine.
- Applications can switch to using either of the two methods on a packet-by-packet basis.
- When Packet DMA configuration method is used, the physical register set corresponding to that queue is updated with the values from the packet.
- If Packet DMA configuration is not used for a packet, direct configuration is assumed and the values present in the current register set for that queue are applied to the data.

FFTC Engine

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

What is the FFTC Engine?

- The FFTC engine uses a ‘Decimation-in-Frequency’ method of Fast Fourier Transform (FFT) algorithm to implement the transforms.
- The FFTC engine computes either the Discrete Fourier Transform (DFT) or the Inverse Discrete Fourier Transform (IDFT) of the data samples that are input to the FFTC.
  – The DFT is given by the formula:

$$X_k = \sum_{n=0:(N-1)}(x_n * W_N^{nk}) \qquad k = 0, 1, \ldots, N-1$$

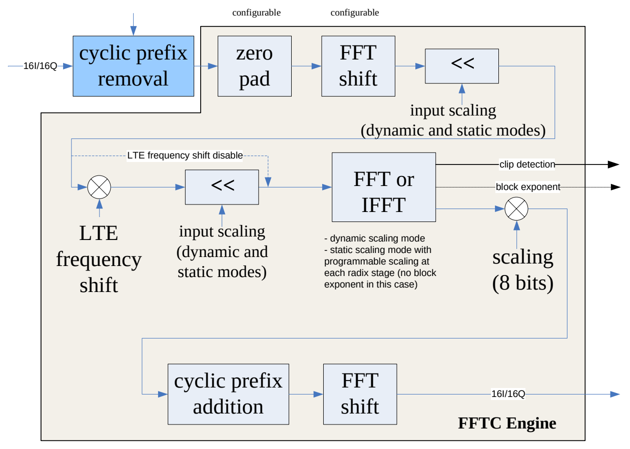
  – The IDFT is given by the formula:

$$x_n = \sum_{k=0:(N-1)}(X_k * W_N^{-nk}) \qquad n = 0, 1, \ldots, N-1$$

Where $W_N^{-nk} = e^{-2\pi nk/N}$

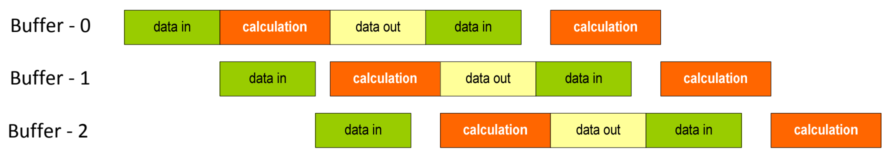
- Each complex FFTC input sample is made up of a 16-bit real and 16-bit imaginary part.
- The engine performs all computations “in place.”

FFTC Engine: Block Diagram



Three-Bank Buffer Design

- The memory buffer of the FFTC engine is divided into three banks.
- Each bank can hold a maximum of 4096 FFT words.



- In cases where the FFT word size does not exceed 4096, the three banks allow the Packet DMA to write to one bank while simultaneously reading from a second bank at the same time that the FFTC engine processes data in the third bank. This buffered write/read/process efficiency allows FFTC calculations to continue without the need to pause for data I/O.
- For FFT word sizes of 6144 and 8192, two of the three banks are required to store each block. As a result, simultaneous reading and writing of data cannot be performed while the FFTC engine is processing data.

FFTC Engine: FFT/IFFT

- The FFTC engine can be configured to compute either the FFT or IFFT.
- The FFTC engine can handle Packet DMA inputs with either uniform or non-uniform FFT/IFFT block sizes.
- In the case of non-uniform block sizes, each packet can have a maximum of 128 blocks.
- At any given time, only one of the four FFTC input queues may contain a non-uniform Packet DMA packet.

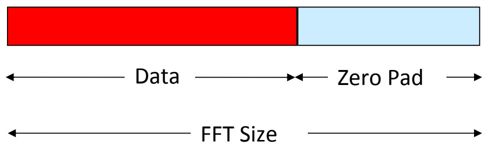
Classification	Supported Sizes
Power of 2	4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192
LTE	12, 24, 36, 48, 60, 72, 96, 108, 120, 144, 180, 192, 216, 240, 288, 300, 324, 360, 384, 432, 480, 540, 576, 600, 648, 720, 768, 864, 900, 960, 972, 1080, 1152, 1200, 1296
Other	1536, 3072, 6144

Zero Padding

- Zero padding is used for upsampling the input data.
- One of two methods can be chosen for interpreting the zero pad value:
  - Zero Pad Add Method

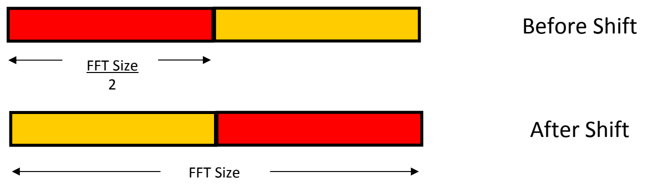
    Data length = FFT size – zero pad value
  - Zero Pad Multiply Method

    Data length = FFT size * 2 $^{-(\text{zero pad value})}$



Left-Right Shift

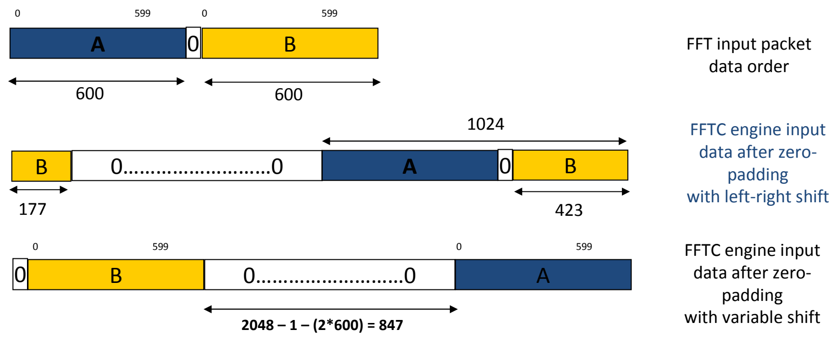
- The left-right shift can be applied to either the input or the output data (but not both).
- If enabled sequence ***{x(0),…, x(N-1)}*** is cyclically shifted to ***{x(N/2)…, x(N-1),0,…, x((N/2-1)}*** where the FFT size is ***N***.
- If zero padding is enabled along with the left-right shift, the swap will not generate the intended frequency shift.



Variable Shift

- Variable shift supports the OFDM transmitter requirements of LTE and WiMAX.
- In both of these radio standards, memory is often organized with subcarriers centered around a DC subcarrier, whereas the IFFT needs the DC subcarrier to be Subcarrier 0.
- The variable shift provides a rotation of the input samples after any zero padding is applied.
- The shift rotates the input samples to the **right** by the amount 2 × *Variable Shift Value.*
- Variable shift is only valid for DFT sizes 128, 256, 512, 1024, 2048, 4096, 8192, 1536, 3072, and 6144.
- The behavior is un-defined for other sizes.

Variable Shift Example


0
599
0
599
A
0
B
600
600
FFT input packet data order
1024
B
0.........................0
A
0
B
177
423
FFTC engine input data after zero-padding with left-right shift
0
599
0
599
0
B
0.........................0
A
2048 – 1 – (2*600) = 847
FFTC engine input data after zero-padding with variable shift

- FFT Size = 2048
- Number of active sub-carriers = 1200
- Variable Shift Value = (2048 – 601)/2 = 724

LTE Frequency Shift

- The LTE frequency shift is a 7.5KHz shift can be applied to the input data according to the LTE requirements.
- The frequency shift is a term-by-term multiplication of the complex input data with the sequence given by

$$H(n) = e^{\pm j2\pi(n+n_o)a/M}$$

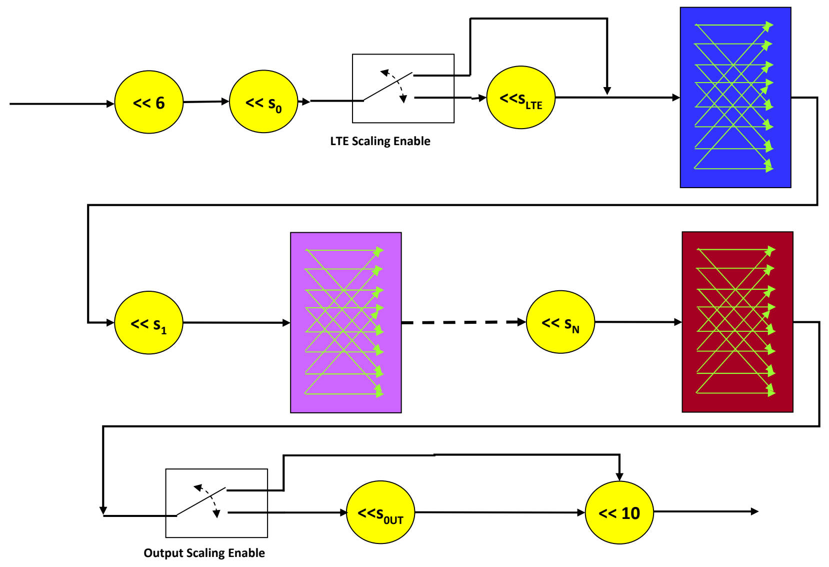
Where

n – Sample index
n_o – Phase offset
a – Multiplication factor.
M – Reference size, either 16384 or 12288

Scaling

- Scaling is performed in the FFTC engine on a per-block basis before each radix stage.
- Additionally, the output of the last stage can also be scaled. The output scaling value is represented as an unsigned Q7 fixed-point value.
- This means that the complex values in the block all have the same scaling applied to them.
- The scaling is a divide operation that is implemented internally as a right shift by 0, 1, 2, or 3 (divide by 1, 2, 4, or 8, respectively) to ensure that saturation does not occur at the output of each radix stage.
- The scaling applied after each stage is tracked and the overall scaling applied is reported back by the engine.

Scaling Flow

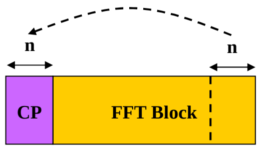


Scaling Modes

- The FFTC engine supports two modes of scaling applied to each stage:
  - Static scaling
    - In this mode the user configures the scaling that is to be applied at each stage.
  - Dynamic scaling
    - In this mode all internal scaling values (except for output scaling) are chosen by the hardware based on the norm of the signal.
    - The LTE shift scaling value is also generated dynamically if enabled.
    - The gains are selected such that the signal is as large as possible after each stage.

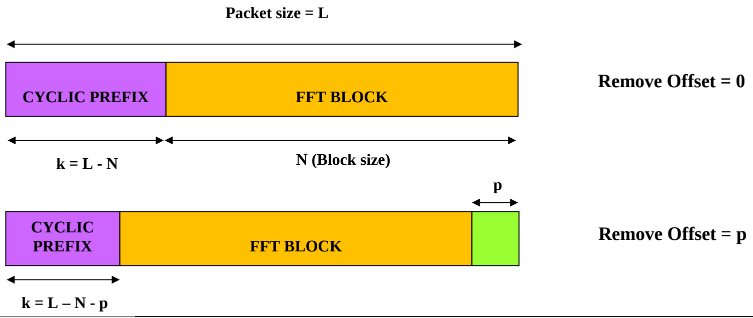
Cyclic Prefix Addition

- The FFTC engine can add the cyclic prefix to the output by transmitting the last ‘n’ samples of the FFT block before transmitting the full block.
- This feature is especially useful when the output is sent directly to the AIF.



Cyclic Prefix Removal

- Cyclic prefix removal is done by writing only the valid samples into the FFTC engine and discarding the rest.
- This feature allows for a seamless interface between the AIF and the FFTC without CorePac intervention.
- Additionally, an offset can also be specified to advance the starting position of the FFT block.
- When cyclic prefix removal is enabled, each data packet can contain only one FFT block.



FFTC Scheduler

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

FFTC Scheduler

- Each FFTC has four dedicated transmit queues:
  - Queues 688 to 691 are attached to FFTC-1
  - Queues 692 to 695 are attached to FFTC-2.
- Placing a PKTDMA data packet on any of the four queues will activate the FFTC.
- When there are packets on more than one queue simultaneously, the FFTC scheduler decides the order based on a starvation prevention, round-robin scheme.
- The FFTC can be configured to have up to four priorities. The FFTC takes the next packet from a queue in prioritized round-robin order.
- The scheduler can interrupt a packet (at block boundaries) if a higher priority packet arrives. The scheduler will not interrupt a packet for another packet at the same or lower priority.

FFTC Streaming Interface

- FFTC Overview
- Functional Architecture
  – Configuration Registers
  – FFTC Engine
  – FFTC Scheduler
  – FFTC Streaming Interface
  – Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

Streaming Interface

- The streaming interface links the Packet DMA to the FFTC engine.
- In addition to providing the necessary handshaking signals to the Packet DMA, the streaming interface keeps track of the status of the FFTC engine and the Packet DMA and manages the flow of data accordingly:
  - Data delivery to the Packet DMA can be delayed if the destination queue is not ready.
  - The Packet DMA can be stalled if the FFTC engine is busy and is unable to accept new data.
- The streaming interface is also responsible for following functions:
  – Decodes the protocol-specific packets from the Packet DMA and updates the FFTC engine configuration registers appropriately.
  – Manages Endianess and delivery of data to the FFTC engine in the correct format
  – Removes cyclic prefix

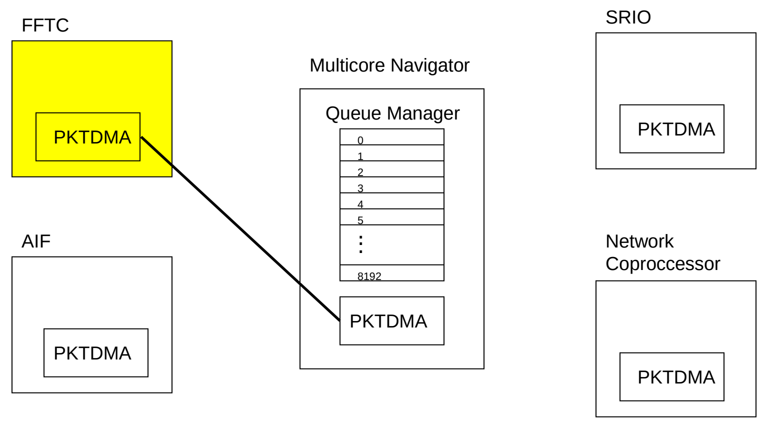
Packet DMA (PKTDMA) Interface

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

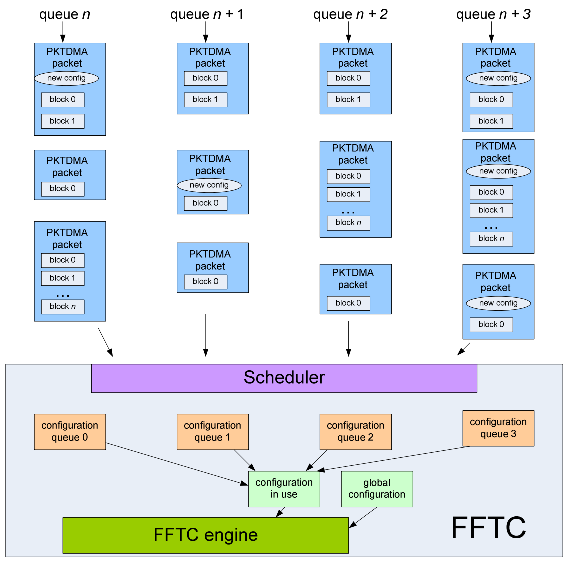
FFTC and PKTDMA

- The FFTC uses the PKTDMA of the Multicore Navigator to receive input data and deliver output results.
- The PKTDMA is the only input and output data interface available on the FFTC.
- The FFTC supports four input hardware queues. One of these four queues must be used in order to deliver the input data and the configuration information (if desired) to the FFTC.
- The FFTC supports both monolithic and host type PKTDMA descriptors.

Packet DMA Topology



PKTDMA Transmit Queues



- FFTC has four dedicated PKTDMA Tx queues.
- Each PKTDMA Tx queue is associated with a unique set of configuration registers in the FFTC.
- Each PKTDMA packet can contain multiple FFT blocks.
- Each queue has its own PKTDMA channel. There is a corresponding PKTDMA Rx channel for every PKTDMA Tx channel.
- Packets with multiple FFT blocks can be interrupted by a higher priority queue at the block boundaries.

FFTC Input: PKTDMA Tx Interface

- Each complex sample of the input data to the FFTC is a 32-bit word (16-bit I & 16-bit Q). The real part (I) always occupies the MSB and the imaginary part (Q) always occupies the LSB of the 32-bit word.
- The PKTDMA interface has some protocol-specific fields allocated in the packet descriptor and Protocol Specific (PS) words that are part of the payload.
- For the FFTC, these PS words are used to send three types of information.
  - Configuration values
  - Pass-through data
  - Debug settings

Protocol-Specific Flag

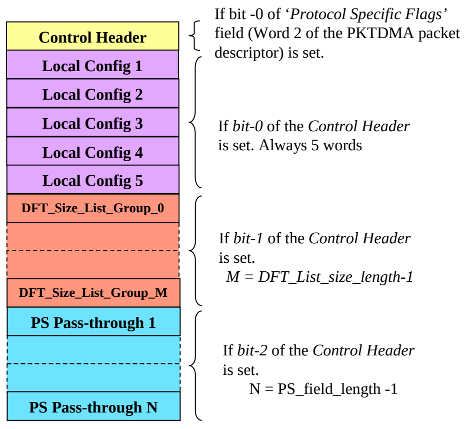
- The four bits of the Protocol-Specific Flags field (present in word 2 of the PKTDMA packet descriptor) are read by one of the four FFTC queues to determine whether the packet includes PS words and/or debug/interrupt settings.
  - Bit 0 denotes if any PS words are present in the packet.
  - Bits 1 and 2 are used for debugging and notification.

Bit	Field	Description
0	Header Present	0: Control header not present in packet. 1: Control header present in packet.
1	Debug Halt	0: Do nothing 1: Issue a halt
2	EOP Interrupt	0: Do nothing 1: FFTC generates the debug interrupt.
3	Reserved	

FFTC Control Header & PS Word Format

Control Header Word Description

Bit	Field	Description
31:29	RESERVED	
28:24	PS_field_length	The length of the pass-through data, in 32-bit words (1 to 4).
23:21	RESERVED	
20:16	DFT_sizes_list_length	Indicates the length of the DFT sizes list in 32-bit words (1 to 26).
15:3	RESERVED	
2	PS pass-through present	Indicates if there is PS pass-through data that should be forwarded on to the receiver. *0 – Not Present* *1 – Present*
1	DFT sizes list present	Indicates that the list of DFT sizes is present. *0 – Not Present* *1 – Present*
0	Local configuration data present	Indicates that the five local control registers are present. If present, configuration data is always five 32-bit words. *0 – Not Present* *1 – Present*


Control Header
Local Config 1
Local Config 2
Local Config 3
Local Config 4
Local Config 5
DFT_Size_List_Group_0
DFT_Size_List_Group_M
PS Pass-through 1
PS Pass-through N
If bit -0 of '*Protocol Specific Flags*' field (Word 2 of the PKTDMA packet descriptor) is set.
If *bit-0* of the *Control Header* is set. Always 5 words
If *bit-1* of the *Control Header* is set. *M = DFT_List_size_length-1*
If *bit-2* of the *Control Header* is set. N = PS_field_length -1

Local Config Words to Register Mapping

Word	Register
1	FFTC_QUEUE_x_DESTINATION_QUEUE_AND_SHIFTING_REGISTER
2	FFTC_QUEUE_x_SCALING_REGISTER
3	FFTC_QUEUE_x_CYCLIC_PREFIX_REGISTER
4	FFTC_QUEUE_x_CONTROL_REGISTER
5	FFTC_QUEUE_x_LTE_FREQUENCY_SHIFT_REGISTER

FFTC Output: PKTDMA Rx Interface

- The FFTC receive packet consists of the FFT or IFFT results, pass-through words, and side information.
- If the Tx packet provided to the FFTC contained multiple blocks, then the transform results of all the blocks that were present in the PKTDMA packet will be combined to form one output PKTDMA packet.
- The pass-through data is returned either as part of the Rx descriptor or at the top of the Rx data packet.
- The FFTC generates the following side information.
  – Block Exponent: Returned in the Rx packet.
  – Clip Detection: Returned in the Rx descriptor (Error Flag Bit-0).
  – Error Indication: Returned in the Rx descriptor (Error Flag Bit-1).
  – Destination Tag: Returned in the Rx packet & descriptor.
  – Source Id: Returned in the Rx packet & descriptor.
  – Flow Id: Returned in the Rx packet & descriptor.

Rx Packet Side Information

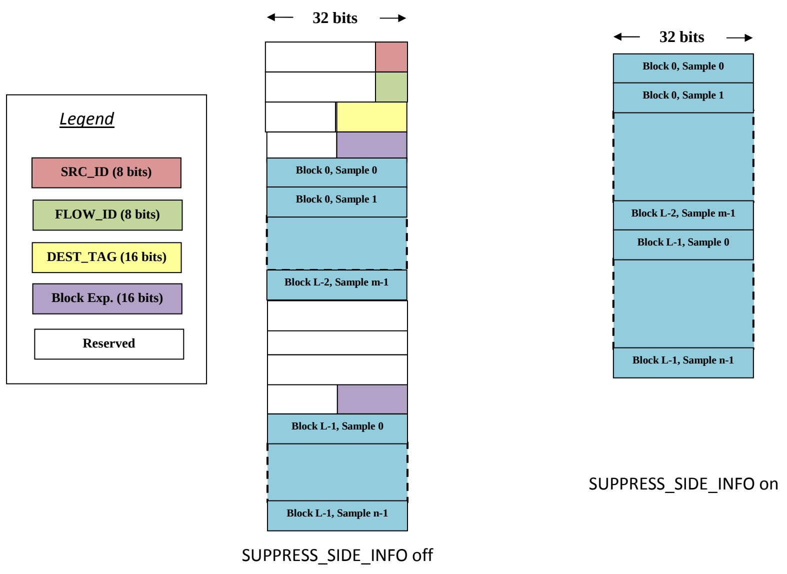
The following side information is included in the PKTDMA Rx data packet:

- Block Exponent is a simple summation of the shift values calculated at each radix stages including the LTE scaling factor

$$\sum s_i + s_{out} - 3 + (LTE\_FREQ\_EN * S_{LTE})$$

- Clip Detection (Error Flag Bit-0) occurs in static scaling mode, where it is possible for clipping to occur during the transform. Clipping occurs when, after any stage, any one of the real or imaginary results exceeds the allocated range of values.
- Error Indication (Error Flag Bit-1) enables the setting of an error bit in the error flag whenever any kind of error has occurred in the FFTC.
- Destination Tag is written in the transmit packet descriptor and is copied and returned in both the receive packet as well as the packet descriptor. The destination tag can be used at the user's discretion to track packets.
- Source ID is written in the transmit packet descriptor and is copied and returned in both the receive packet as well as the packet descriptor. The source ID can be used at the user's discretion to track packets.
- Flow ID refers to the index of the flow table entry to use to configure the receive channel.

Rx Packet Format



FFTC Error and Debug Interrupts

- There are four interrupt lines from the FFTC, one per PKTDMA channel.
- These interrupts are routed to the CP_INTC0 module.
- Interrupts are generated when any of the following conditions are true.
  - FFTC configuration error: *Invalid FFTC configuration values or combinations.*
  - EOP error: *Data length in the Tx packet does not match the configuration.*
  - *Rx buffer starvation error: Rx free queue empty.*
  - Invalid configuration length: *PS data length incorrect.*
  - Debug Halt notification: *Start of a packet by the FFTC whose DEBUG_HALT bit is set in the PS Flag of the descriptor.*
  - EOP notification: *The complete packet is delivered to the PKTDMA and EOP interrupt bit is set in the PS Flag of the packet.*
- Each of these interrupts are maskable on a channel basis and are masked by default.

PKTDMA Accumulator Interrupt

- If one of the 32 high-priority or the 16 low-priority accumulator queues of the Packet DMA are used as the destination queue for any of the Rx channels, then the accumulator interrupt can be used to signal the arrival of new output packets from the FFTC.
- The accumulator interrupt is independent of the debug interrupts and is the preferred method.

QPEND Interrupt Queues

- The Queue Manager Subsystem includes 20 queues that can generate interrupts to the DSPs via the chip interrupt controller (CP_INTC0).
- Using one or more of these queues as RX destination queues is another method for generating interrupts from the FFTC.
- These interrupts are independent of the debug interrupts.

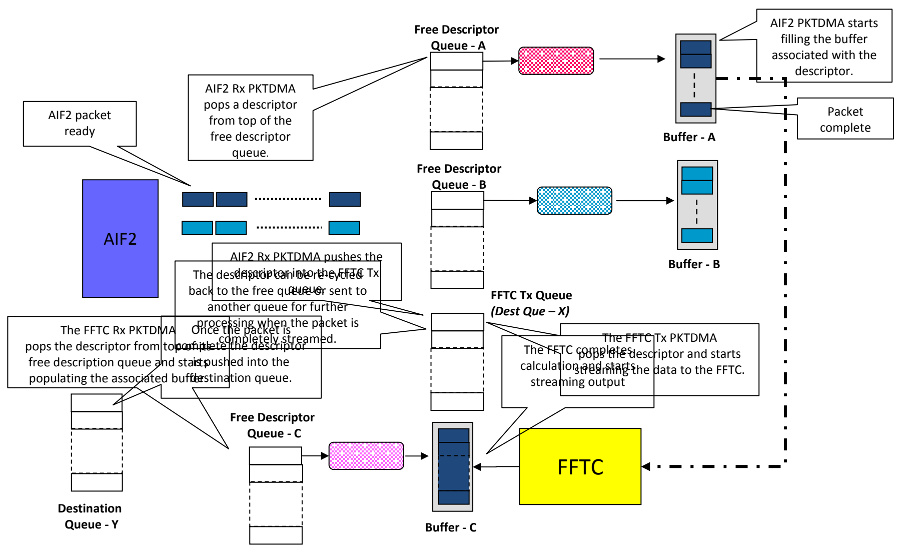
FFTC Usage

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

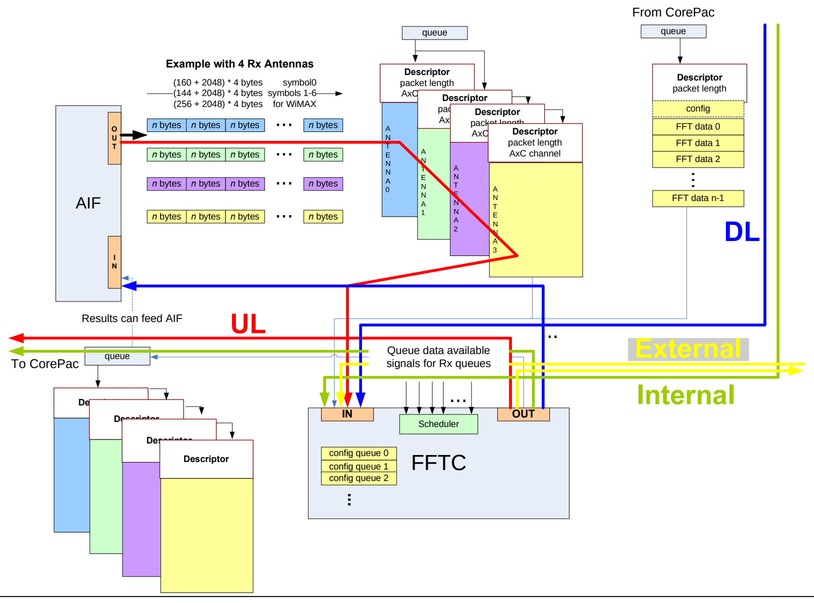
FFTC Sample Use Cases

- AIF2/CorePac to FFTC
- FFTC to CorePac/AIF2
- Uplink Receiver

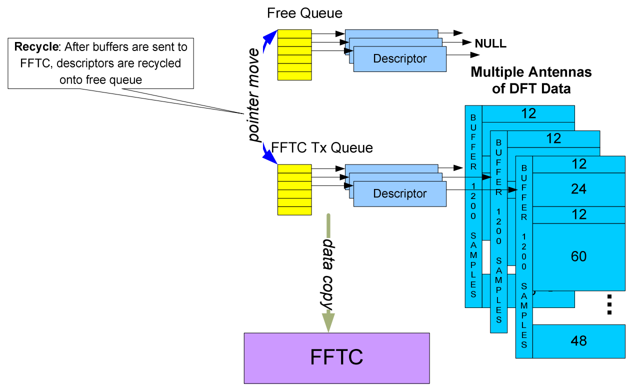
AIF-to-FFTC Data Flow


Free Descriptor Queue - A
AIF2 PKTDMA starts filling the buffer associated with the descriptor.
AIF2 Rx PKTDMA pops a descriptor from top of the free descriptor queue.
AIF2 packet ready
Packet complete
Buffer - A
Free Descriptor Queue - B
AIF2
Buffer - B
AIF2 Rx PKTDMA pushes the descriptor into the FFTC Tx queue
The descriptor can be re-cycled back to the free queue or sent to another queue for further processing when the packet is completely streamed.
FFTC Tx Queue
(Dest Que – X)
The FFTC Rx PKTDMA pops the descriptor from top of the free description queue and starts populating the associated buffer.
Once the packet is complete the descriptor is pushed into the destination queue.
The FFTC Tx PKTDMA pops the descriptor and starts streaming the data to the FFTC.
The FFTC completes calculation and starts streaming output
Free Descriptor Queue - C
FFTC
Destination Queue - Y
Buffer - C

FFTC Data Paths



Sending DFTs to FFTC from CorePac


Free Queue
Recycle: After buffers are sent to FFTC, descriptors are recycled onto free queue
pointer move
Descriptor
NULL
Multiple Antennas of DFT Data
FFTC Tx Queue
Descriptor
data copy
FFTC
BUFFER 1200 SAMPLES
12
12
12
24
12
60
48

- During init: CorePac creates free queue of empty descriptors with "default" FFTC configuration.
- During runtime:
  - CorePac pops descriptor pointer from free queue.
  - Sets pointer to buffer, and optionally changes FFTC configuration information in the descriptor (change only those fields that need changing).
  - CorePac pushes descriptor onto one of the FFTC transmit queues.
- Buffer descriptors are returned to the free queue after transmission to FFTC
  - Return queue is a configuration parameter in each descriptor

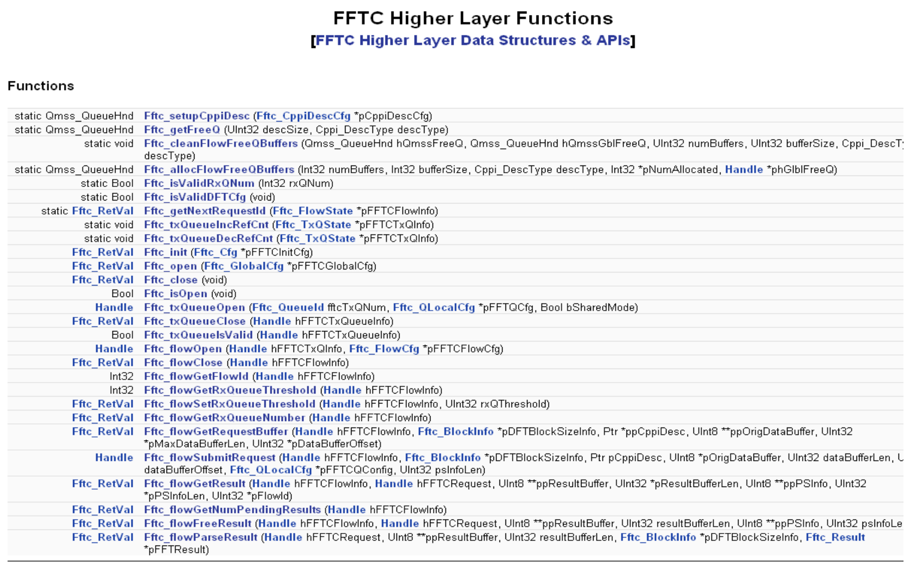
FFTC Low-level Driver

- FFTC Overview
- Functional Architecture
  - Configuration Registers
  - FFTC Engine
  - FFTC Scheduler
  - FFTC Streaming Interface
  - Packet DMA (PKTDMA) Interface
- FFTC Usage
- FFTC Low Level Driver (LLD)

FFTC Software

- Benefits
  - Provides high-speed FFTC register access (R/W) APIs
  - Simplify PKTDMA/QMSS internals required to perform FFT/IFFT
  - OS-independent
  - Multicore-aware
- Provides two levels of APIs:
  - High-level APIs to submit FFT/IFFT requests and retrieve results
    - Validates user inputs
    - Provides call-outs to OS AL for multi-core synchronization
    - Handles all PKTDMA/QMSS details internally
    - Parses FFT result and populates per-block result and error info
  - Low Level APIs to read/write FFTC engine configuration & status
    - Low performance impact APIs
    - No user input validation
    - Utility APIs to build FFTC Control Header, DFT size list etc

FFTC High-Level API



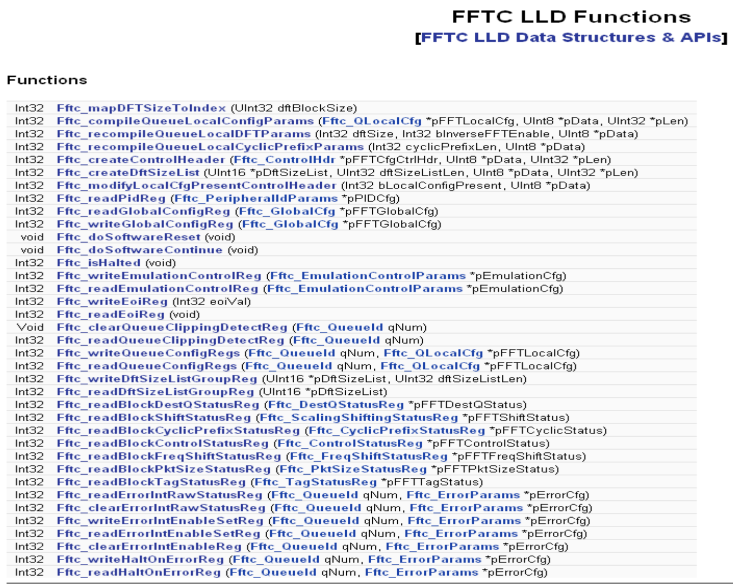
FFTC Higher Layer Functions

[FFTC Higher Layer Data Structures & APIs]

Functions

static Qmss_QueueHnd	**Fftc_setupCppiDesc** (**Fftc_CppiDescCfg** *pCppiDescCfg)
static Qmss_QueueHnd	**Fftc_getFreeQ** (UInt32 descSize, Cppi_DescType descType)
static void	**Fftc_cleanFlowFreeQBuffers** (Qmss_QueueHnd hQmssFreeQ, Qmss_QueueHnd hQmssGblFreeQ, UInt32 numBuffers, UInt32 bufferSize, Cppi_DescT descType)
static Qmss_QueueHnd	**Fftc_allocFlowFreeQBuffers** (Int32 numBuffers, Int32 bufferSize, Cppi_DescType descType, Int32 *pNumAllocated, **Handle** *phGlblFreeQ)
static Bool	**Fftc_isValidRxQNum** (Int32 rxQNum)
static Bool	**Fftc_isValidDFTCfg** (void)
static **Fftc_RetVal**	**Fftc_getNextRequestId** (**Fftc_FlowState** *pFFTCFlowInfo)
static void	**Fftc_txQueueIncRefCnt** (**Fftc_TxQState** *pFFTCTxQInfo)
static void	**Fftc_txQueueDecRefCnt** (**Fftc_TxQState** *pFFTCTxQInfo)
Fftc_RetVal	**Fftc_init** (**Fftc_Cfg** *pFFTCInitCfg)
Fftc_RetVal	**Fftc_open** (**Fftc_GlobalCfg** *pFFTCGlobalCfg)
Fftc_RetVal	**Fftc_close** (void)
Bool	**Fftc_isOpen** (void)
Handle	**Fftc_txQueueOpen** (**Fftc_QueueId** fftcTxQNum, **Fftc_QLocalCfg** *pFFTQCfg, Bool bSharedMode)
Fftc_RetVal	**Fftc_txQueueClose** (**Handle** hFFTCTxQueueInfo)
Bool	**Fftc_txQueueIsValid** (**Handle** hFFTCTxQueueInfo)
Handle	**Fftc_flowOpen** (**Handle** hFFTCTxQInfo, **Fftc_FlowCfg** *pFFTCFlowCfg)
Fftc_RetVal	**Fftc_flowClose** (**Handle** hFFTCFlowInfo)
Int32	**Fftc_flowGetFlowId** (**Handle** hFFTCFlowInfo)
Int32	**Fftc_flowGetRxQueueThreshold** (**Handle** hFFTCFlowInfo)
Fftc_RetVal	**Fftc_flowSetRxQueueThreshold** (**Handle** hFFTCFlowInfo, UInt32 rxQThreshold)
Fftc_RetVal	**Fftc_flowGetRxQueueNumber** (**Handle** hFFTCFlowInfo)
Fftc_RetVal	**Fftc_flowGetRequestBuffer** (**Handle** hFFTCFlowInfo, **Fftc_BlockInfo** *pDFTBlockSizeInfo, Ptr *ppCppiDesc, UInt8 **ppOrigDataBuffer, UInt32 *pMaxDataBufferLen, UInt32 *pDataBufferOffset)
Handle	**Fftc_flowSubmitRequest** (**Handle** hFFTCFlowInfo, **Fftc_BlockInfo** *pDFTBlockSizeInfo, Ptr pCppiDesc, UInt8 *pOrigDataBuffer, UInt32 dataBufferLen, U dataBufferOffset, **Fftc_QLocalCfg** *pFFTCQConfig, UInt32 psInfoLen)
Fftc_RetVal	**Fftc_flowGetResult** (**Handle** hFFTCFlowInfo, **Handle** hFFTCRequest, UInt8 **ppResultBuffer, UInt32 *pResultBufferLen, UInt8 **ppPSInfo, UInt32 *pPSInfoLen, UInt32 *pFlowId)
Fftc_RetVal	**Fftc_flowGetNumPendingResults** (**Handle** hFFTCFlowInfo)
Fftc_RetVal	**Fftc_flowFreeResult** (**Handle** hFFTCFlowInfo, **Handle** hFFTCRequest, UInt8 **ppResultBuffer, UInt32 resultBufferLen, UInt8 **ppPSInfo, UInt32 psInfoLe
Fftc_RetVal	**Fftc_flowParseResult** (**Handle** hFFTCRequest, UInt8 **ppResultBuffer, UInt32 resultBufferLen, **Fftc_BlockInfo** *pDFTBlockSizeInfo, **Fftc_Result** *pFFTResult)

FFTC Low-Level API



FFTC LLD Functions
[FFTC LLD Data Structures & APIs]

Functions

Int32	Fftc_mapDFTSizeToIndex (UInt32 dftBlockSize)
Int32	Fftc_compileQueueLocalConfigParams (Fftc_QLocalCfg *pFFTLocalCfg, UInt8 *pData, UInt32 *pLen)
Int32	Fftc_recompileQueueLocalDFTParams (Int32 dftSize, Int32 bInverseFFTEnable, UInt8 *pData)
Int32	Fftc_recompileQueueLocalCyclicPrefixParams (Int32 cyclicPrefixLen, UInt8 *pData)
Int32	Fftc_createControlHeader (Fftc_ControlHdr *pFFTCfgCtrlHdr, UInt8 *pData, UInt32 *pLen)
Int32	Fftc_createDftSizeList (UInt16 *pDftSizeList, UInt32 dftSizeListLen, UInt8 *pData, UInt32 *pLen)
Int32	Fftc_modifyLocalCfgPresentControlHeader (Int32 bLocalConfigPresent, UInt8 *pData)
Int32	Fftc_readPidReg (Fftc_PeripheralIdParams *pPIDCfg)
Int32	Fftc_readGlobalConfigReg (Fftc_GlobalCfg *pFFTGlobalCfg)
Int32	Fftc_writeGlobalConfigReg (Fftc_GlobalCfg *pFFTGlobalCfg)
void	Fftc_doSoftwareReset (void)
void	Fftc_doSoftwareContinue (void)
Int32	Fftc_isHalted (void)
Int32	Fftc_writeEmulationControlReg (Fftc_EmulationControlParams *pEmulationCfg)
Int32	Fftc_readEmulationControlReg (Fftc_EmulationControlParams *pEmulationCfg)
Int32	Fftc_writeEoiReg (Int32 eoiVal)
Int32	Fftc_readEoiReg (void)
Void	Fftc_clearQueueClippingDetectReg (Fftc_QueueId qNum)
Int32	Fftc_readQueueClippingDetectReg (Fftc_QueueId qNum)
Int32	Fftc_writeQueueConfigRegs (Fftc_QueueId qNum, Fftc_QLocalCfg *pFFTLocalCfg)
Int32	Fftc_readQueueConfigRegs (Fftc_QueueId qNum, Fftc_QLocalCfg *pFFTLocalCfg)
Int32	Fftc_writeDftSizeListGroupReg (UInt16 *pDftSizeList, UInt32 dftSizeListLen)
Int32	Fftc_readDftSizeListGroupReg (UInt16 *pDftSizeList)
Int32	Fftc_readBlockDestQStatusReg (Fftc_DestQStatusReg *pFFTDestQStatus)
Int32	Fftc_readBlockShiftStatusReg (Fftc_ScalingShiftingStatusReg *pFFTShiftStatus)
Int32	Fftc_readBlockCyclicPrefixStatusReg (Fftc_CyclicPrefixStatusReg *pFFTCyclicStatus)
Int32	Fftc_readBlockControlStatusReg (Fftc_ControlStatusReg *pFFTControlStatus)
Int32	Fftc_readBlockFreqShiftStatusReg (Fftc_FreqShiftStatusReg *pFFTFreqShiftStatus)
Int32	Fftc_readBlockPktSizeStatusReg (Fftc_PktSizeStatusReg *pFFTPktSizeStatus)
Int32	Fftc_readBlockTagStatusReg (Fftc_TagStatusReg *pFFTTagStatus)
Int32	Fftc_readErrorIntRawStatusReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_clearErrorIntRawStatusReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_writeErrorIntEnableSetReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_readErrorIntEnableSetReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_clearErrorIntEnableReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_writeHaltOnErrorReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)
Int32	Fftc_readHaltOnErrorReg (Fftc_QueueId qNum, Fftc_ErrorParams *pErrorCfg)

For More Information

- For more information, please refer to the FFT Coprocessor (FFTC) User Guide: http://www.ti.com/lit/SPRUGWS2
- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website.