TEXAS INSTRUMENTS

Application Report

SPRABH8–December 2014

Migrating From AIF1 to AIF2 for KeyStone Devices

High-Performance Multicore Processors

ABSTRACT

This application report describes the main differences between AIF1 and AIF2 and how to migrate the AIF configuration to work seamlessly in various usage scenarios.

Contents

List of Figures

List of Tables

1 Introduction[1]

There are many differences between AIF1 and AIF2. AIF2 was designed to support multiple radio standards like WCDMA, LTE, WiMAX, TD-SCDMA, and GSM/EDGE. The ultimate limitation of AIF1 was that the basic design concept of AIF1 was only for WCDMA; however, the 4G market is growing along with the requirements for higher speed and increased bandwidth for next-generation interfaces. AIF2 will be the best solution for these requirements. This document describes the main differences between AIF1 and AIF2 and the AIF2 features that have evolved from the previous version.

1.1 Acronyms

Table 1. Acronyms Used in This Document

Term	Definition
AD	AIF2 DMA (submodule)
AIF	Antenna Interface
AT	AIF2 Timers (submodule) *previously called Frame Sync module*
AxC	Antenna Carrier (stream)
Basic Frame	A CPRI basic frame consists of 16 words
CorePac	A specific DSP core
CPPI	Common Port Programming Interface (*now called Multicore Navigator*)
CPRI	Common Public Radio Interface
CRC	Cyclic Redundancy Check
CSL	Code Support Library
CW	Control Word (CPRI)
DB	AIF2 Data Buffer (submodule)
DL	Downlink
DMA	Direct Memory Access
DSP	Digital Signal Processor
EE	AIF2 Error Event handler (submodule)
FDD	Frequency Division Duplexing
FIFO	First In First Out queue memory structure
HW	Hardware
Hyperframe	1 CPRI Hyperframe = 256 CPRI basic frames
K Character	7-bit line codes representing 8b10b control characters
L2	CorePac DSP Level 2 SRAM
LTE	Long Term Evolution
LUT	Look Up Table
MAC	Media Access Control
MMR	Memory Mapped Register
MOD	Modulo
Multicore Navigator	*previously called CPPI*
OBSAI	Open Base Station Architecture Initiative
OFDM	Orthogonal Frequency Division Multiplexing
Packet DMA	*previously called CPPI DMA*
PD	AIF2 Protocol Decoder (submodule)
PE	AIF2 Protocol Encoder (submodule)
RAC	Receive Accelerator Co-processor
RAM	Random Access Memory
RM	AIF2 Receive MAC (submodule)
RP1	Reference Point 1 (OBSAI)

[1] All trademarks are the property of their respective owners.

Table 1. Acronyms Used in This Document (continued)

Term	Definition
RP3	Reference Point 3 (OBSAI)
RT	AIF2 Re-Transmitter (AIF2 submodule)
SerDes	SERializer / DESerializer
TDD	Time Division Duplexing
TD-SCDMA	Time Division-Synchronous Code Division Multiple Access
TI	Texas Instruments
TM	AIF2 Transmit MAC (submodule)
UL	Uplink
UMTS	Universal Mobile Telecommunication System
VBUSM	Virtual Bus Multi-issue
VBUSP	Virtual Bus Pipeline
WCDMA	Wideband Code Division Multiple Access
WiMax	Worldwide Interoperability for Microwave Access

2 Differences in Physical Level

2.1 Clock Strategy

AIF1 has two clock inputs. One is AI_REF_CLK (the source clock for input to SerDes PLLs with variable rate, SYSCLK clock) and the other one is VBUS_CLK (the main processing clock of the AIF1 modules and it is core clock divided by 3) and the maximum WCDMA link rate was 4x, which uses 307.2 MHz as a Tx-byte clock rate (245.76 MHz for CPRI) .

Clocking of the AIF2 processing occurs in the Tx **dual**-byte clock domain, which will typically be 307.2 MHz (OBSAI) or 245.76 MHz (CPRI). While the one-byte clock processes only 8 bits per clock, the dual-byte clock handles two-byte data at one-clock time. This allows AIF2 to achieve a maximum link rate of 8x and process maximum 32 WCDMA AxC data per link with the same reference clock.

The VBUS clock of the system is in the CPU clock/3 clock domain and is the same rate as AIF1.

In AIF1, TM modules use theTx byte clock and the RM module uses the VBUS clock for Pi calculation, so it calculates Pi and Delta with different clock rates. This can cause confusion when calculating Pi and Delta. AIF2 uses the same clock domain (dual-byte clock) for the TM and RM modules, so Pi and Delta can be set with the same domain clock value.

Table 2 shows the OBSAI RP3 SerDes rate and message grouping. The 1x link rate was supported by AIF1 but is not supported in AIF2. Instead, the 8x link rate is supported in AIF2. To support the 8x link speed, SerDes does special data scrambling to avoid crosstalk. Section 2.2 shows how this is implemented in AIF2.

Table 2. OBSAI RP3 SerDes rates

Link Rate	Line Rate (Gbps)	Data Msg Payload Rate (Gbps)	Control Msg Payload Rate (Gbps)
1x (AIF1 only)	0.768	0.49152	0.024576
2x	1.536	0.98304	0.049152
4x	3.072	1.96608	0.098304
8x (AIF2 only)	6.144	3.93216	0.196608

Data messages are grouped with control messages. First there are *i*20 Data messages, then *i* Control Messages (where *i* depends on link rate *i*={1x, 2x, 4x, 8x}).

Table 3. OBSAI Data/Control Message Grouping

Link Rate	Message Groups	Data Messages	Control Messages
1x	1	20	1
2x	2	40	2
4x	4	80	4
8x	8	160	8

The position of the control slot can easily be changed in AIF1, which makes the transmission rule and policy very flexible; however, this is not supported in AIF2 and a fixed control-slot position must be used because the dual bit-map rule of theOFDM radio standard does not allow a flexible control-slot position. DBMR supports a very flexible and strong transmission rule for both OBSAI and CPRI. For more information, see Section 4.

2.2 6 GHz SerDes

AIF2 supports 6 GHz SerDes to get larger data bandwidth; however, controlling crosstalk is the key issue for safe data transmission.

The main concern is crosstalk between transmitters through the local SerDes power supply. With all transmitters having different scrambling offsets, randomness between transmitting lanes is achieved. The assignment of unique scrambler offsets for receivers is optional as crosstalk between receivers and transmitters is non-critical.

The RP3 transmitter is configured by higher layers with a starting value of the seven-degree-polynomial scrambling code generator. Higher layers should configure unique seed values for adjacent RP3 Tx links. The RP3 receiver is a slave to the transmitter; it receives the seed value during a training sequence.

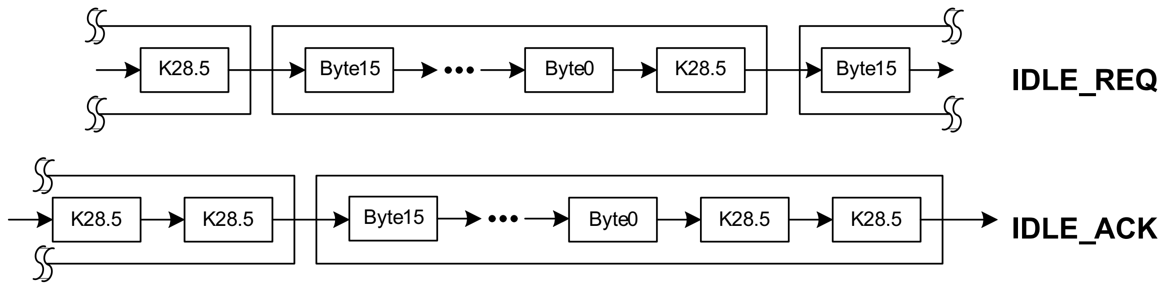

Figure 1. Scrambling Training Patterns

The scrambling code generator increments by one bit position for each bit of every byte. In each bit position of the scrambling code generator, one scrambling bit is created that is XOR with each single bit of a data byte. The bits of a byte are processed in order from MSB to LSB according to the order the scrambling bit sequence is generated. On every K28.5 or K28.7 character, the scrambling code generator is reset to the starting seed value.

The seed value and checking sequence is transmitted as training patterns from the RP3 transmitter to the receiver during the IDLE period of the transmit state machine. Only 8x-rate links use these special patterns during the IDLE period. There are two substates in the IDLE state: IDLE_REQ and IDLE_ACK; two different training patterns are transmitted in the two substates:

- IDLE_REQ: K28.5, byte0, …, byte15… repeat
- IDLE_ACK: K28.5, K28.5, byte0, …, byte15… repeat

Bit-level scrambling is performed on 8x-rate links to reduce crosstalk between links and reduce intersymbol interference (ISI). The RP3 transmitter applies a seven-degree polynomial to data bytes and the inverse operation is performed by the RP3 receiver. Scrambling applies only to 6-GHz operation (8x link rate). Link rates {1x, 2x, 4x} are backward compatible with no scrambling applied.

The scrambler is a seven-degree polynomial, linear-feedback shift register (LFSR). The polynomial is X^7+X^6+1. K28.5 or K28.7 characters reset the LFSR to the seed value. The bit pattern repeats every 127 bits.

The RP3 physical layer provides coding and serialization of the transmission path. An LFSR scrambling algorithm, applied to 8x links only, smooths the data stream before 8b10b encoding. The data link layer provides a method for creating messages of the bit stream. The transport layer uses the address field messages to control message routing for processing the application layer. The application layer terminates the payload of messages into packets.

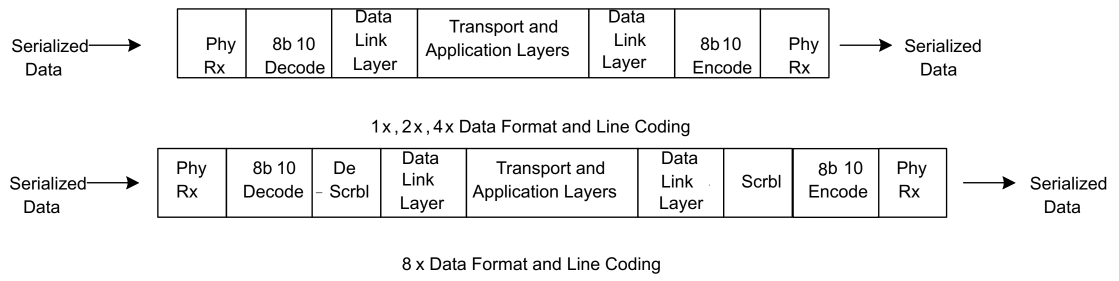

Figure 2. Data Format and Line Coding

2.3 AIF2 Reset

AIF works in continuous mode and this means that you cannot easily sense when it is turned off or reset while running; this creates some problematic situations when you attempt to debug or test the AIF application or try to reconfigure one or all links during runtime. That is why AIF2 supports dynamic configuration and an easier Reset function.

AIF2 supports three levels of reset methodology. Figure 3 shows the reset methodologies and the brief concept of software reset and reset Isolation.

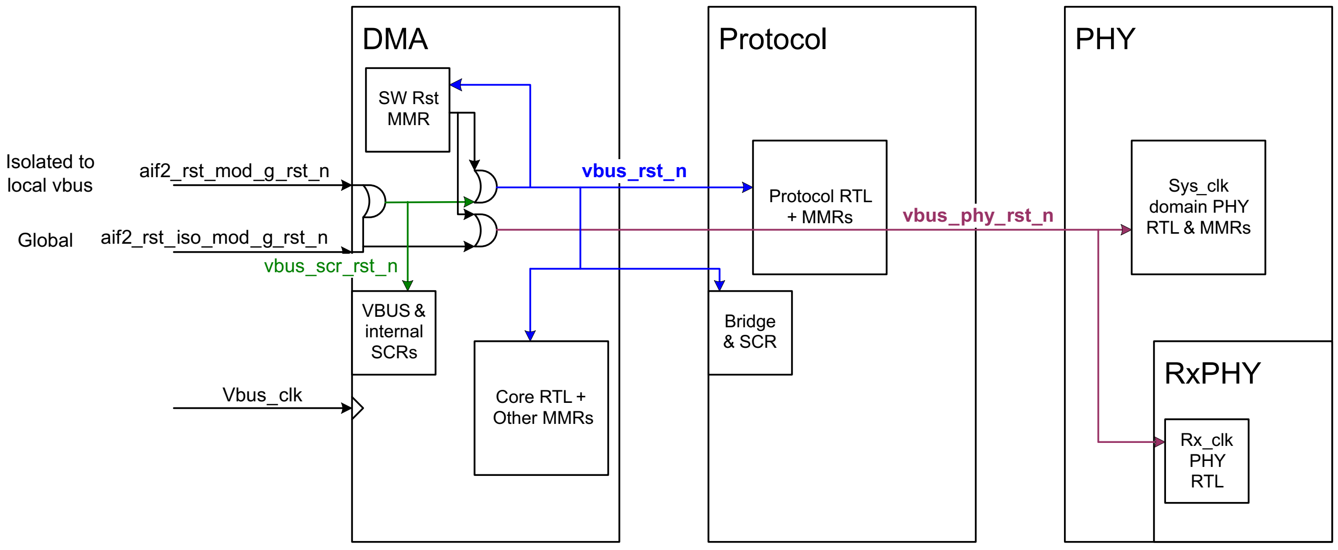

Figure 3. AIF2 Reset Strategy (Software Reset and Reset Isolation)

The SerDes reset is driven by vbus_phy_rst_n, which means that it will be driven by the “or” of the aif2_rst_iso_mod_g_rst_n (global hard reset) or the MMR driven software reset.

Circuitry that is not reset for Reset Isolation includes:

- PHY submodules: RM, CI, RT, CO, TM
- PHY SerDes: SD
- All MMRs in the PHY
- EE : Masks and configurations of EE

The PE and PD submodule interfaces are in the sys_clk domain and interface to the PHY modules. PD has PHY inputs only, and has no sensitivity to reset isolation. Once reset, PD will disregard any inputs so PHY operation will have no ill effect. The PE outputs signals that drive the PHY. After the PE is reset, the links are disabled. PE ceases all data output. The fact that the PE is disabled is also communicated to the RT. RT automatically switches into a mode that no longer requires PE inputs.

AIF2 has a special single MMR that contains a bit that is used to reset the hardware by software control. This bit is OR'ed with the hardware reset pins coming from the core. With the exception of the VBUS infrastructure, all AIF2 hardware is reset when the software reset pin is activated

The VBUSP infrastructure is exempt from software reset, as resetting this logic could result in crashing the VBUS.

Circuitry that is not reset during software reset includes:

- Config VBUS
- VBUSP Interface
- internal SCR circuits.
- vbus_clk to sys_clk re-timing bridge

Software reset is pulsed a minimum of 16 clock cycles. VBUSP write ready is held off (wait-stated) for these 16 cycles and is then released. This prevents MMR access during software reset.

2.4 Internal Memory for DMA

AIF1 has 32 WCDMA chip size (2 Kbytes) CSRAM per link and it is a circular buffer, so the data could be overwritten after transferring 32 chips per AxC. This RAM size was adequate for WCDMA DMA (for both DL, UL) and AIF1 also has four FIFOs to cover packet traffic. But there are some problems and limitations from that kind of CS memory RAM structure, as listed:

- CSRAM structure makes EDMA scheme too complex
- This is good for continuous data stream but not good for packet type data
- AIF1 has only 4 FIFOs for packet traffic but it is not enough to cover User's requirement.

To resolve these kinds of problems, AIF2 chose a full FIFO structure instead of CS RAM. The AIF2 data buffer consists of two independent subsystems: the Ingress DB (IDB) and the Egress DB (EDB). Data in the IDB is transferred to other DSP resources via the VBUSM interface. Data from other DSP resources is transferred into the EDB via the VBUSM interface. Each DB supports a 128 FIFO-type buffer for each 128 AxC or packet channels. The basic data transaction size is 16 bytes or "Quad Word" (QW) and is fully flexible for both AxC data and pure packet-mode data.

To support WCDMA more efficiently, AIF2 also uses a special simple Direct DMA mode, which is called "DirectIO" and uses FIFOs like circular buffers as in AIF1. This special mode supports only two buffer sizes for WCDMA: (128 byte) and LTE (256 byte). The 128-byte circular buffer size is large enough to cover any WCDMA data rate because each buffer for each AxC channel is separated. For example, to transfer 16 AxC data, 16 128-byte buffers can be assigned for each channel and each buffer can hold a maximum of 32 chips of WCDMA data inside like AIF1.

The size of each Ingress and Egress FIFO is programmable but only a limited number of sizes will be supported. The minimum size is eight quad words or 128 bytes. All of the other sizes are power-of-two multiples of eight quad words up to a maximum of 256 quad words or 4K bytes. Table 4 shows the AIF2 supported FIFO buffer sizes.

Table 4. Table 1 – AIF2 Supported FIFO Buffer Sizes

Supported FIFO Buffer Sizes (Quad-Words)	Max Number of Buffer Channels Supported	BUF_DEPTH[2:0]
8	128	0
16	64	1
32	32	2
64	16	3
128	8	4

Table 4. Table 1 – AIF2 Supported FIFO Buffer Sizes (continued)

Supported FIFO Buffer Sizes (Quad-Words)	Max Number of Buffer Channels Supported	BUF_DEPTH[2:0]
256	4	5,6,7

The size and starting location of each FIFO and the starting location of each circular buffer are programmable by DB MMR. This variable buffer size is applied only in non- DIO cases. DIO only supports 128 bytes or 256 bytes as described above.

3 Frame Sync Module vs. AIF2 Timer

The AIF2 Timer (AT) does the same job as the Frame sync module in TCI6488. It generates the heartbeat for several IPs. The AT generates a frame-boundary or symbol-boundary signal so that external events or internal events can be synchronized. There are two basic timer types used as a reference for PHY and radio timers (PHYT and RADT).

The PHY timer (PHYT) is used as a reference for link-based event generation and it is the same concept as the RP3 timer or system timer in the Frame sync module. This timer is closely associated with timing of received and transmitted link-data traffic. It is used to direct link traffic and is used as a reference to set transmit Delta time and to check receive Pi time.

The radio timer (RADT) is used as a reference for radio-standard event generation. This timer is synchronized to a selected standard and works identically to the PHY timer. It provides great flexibility to support multiple radio standards and radio frame sizes.

The RADT timer has two offset versions: ULRADT and DLRADT; These are used to mark uplink and downlink radio standard time. These times are offset from the referenced RADT.

3.1 Interface and Architecture Changes

3.1.1 Interface

The Frame synchronization module for AIF1 is intended to generate controllable timed events. The system clock control module sends synchronization and clock inputs to the FS. Figure 4 shows the functional block diagram.

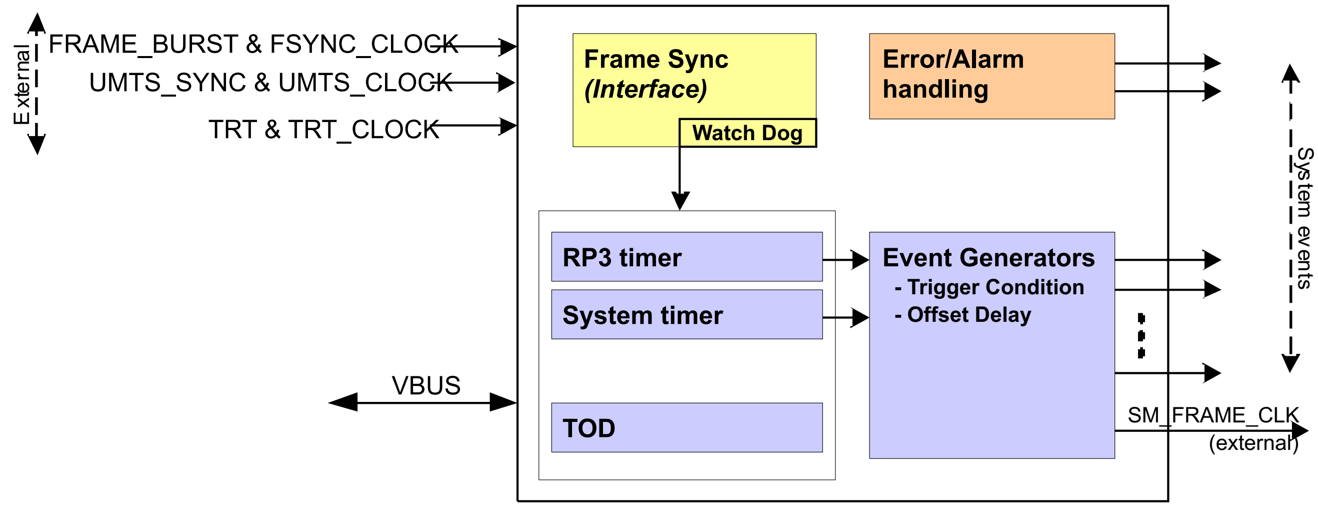

Figure 4. Top-Level Frame Synchronization Block Diagram

The Frame synchronization module has UMTS sync and clock and also has TRT sync and clock external pins. Internally, it has an RP3 timer and System timer, either of which can be selected as an output event generator.

There are many changes in the AIF2 AT. The AT has a PHY timer and RAD timer instead of RP3 and System timers; the two timers work simultaneously but operate independently. There are PHY and RAD sync input pins for each timer instead of UMTS sync, but the clock input pin was removed. The AIF2 dual-byte clock is used to run the timers.

TRT sync input and clock were also removed but RP1 frame burst and clock are the same as in the AIF1. Figure 5 shows the AIF2 Timer chip IO.

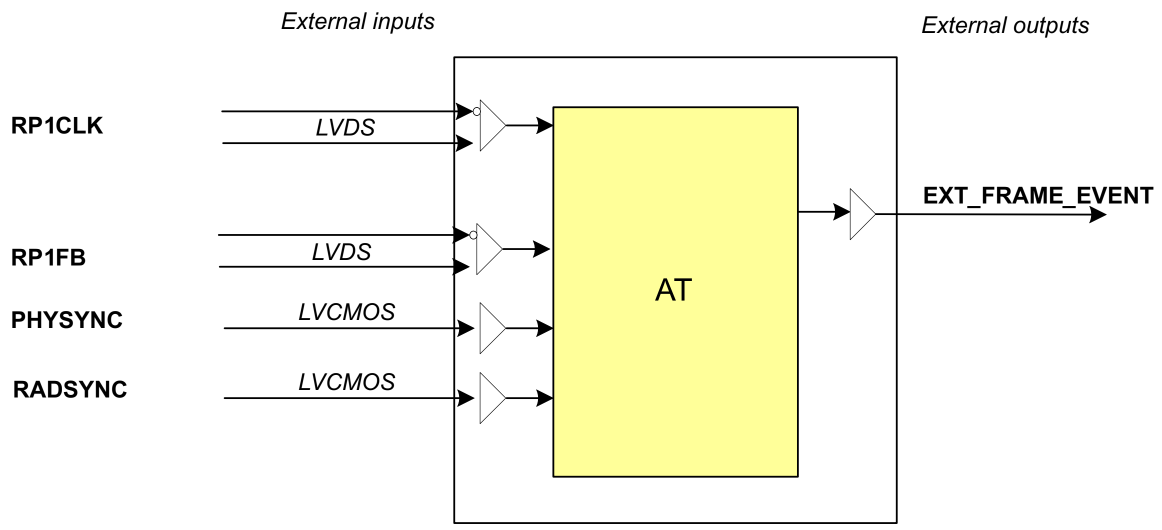

Figure 5. AIF2 Timer Chip IO

Physync and Radsync could be triggered at the same time or different times. AIF2 allows several sync-trigger-select options for both timers.

3.1.2 General Architecture

The AIF1 and Frame sync module are required to synchronize to the external UMTS frame or standard-specific alignment. Optionally, differential signals {FSYNC_BURST, FSYNC_CLOCK} or single-ended input signals {UMTS_SYNC, UMTS_CLOCK} for the RP3 UMTS timer and the system timer selects between {UMTS_SYNC, UMTS_CLOCK} and {TRT, TRT_CLOCK} for the non-OBSAI supported standard can be chosen.

AIF2 AT removed all non-RP1 clock and sync from the frame-sync module. Instead, it uses the PHY timer and RAD timer for RP3 timing and other general purpose timing. PHY or RAD sync can activate one or both timers.

Below is the frame-sync option field in the AT control 1 register:

phy_syncsel	READ_WRITE		PHY sync selection
		0	Use RP1 interface for synchronizing the PHYT frame boundary
		1	Use PHYTSYNC chip input for synchronizing the PHYT frame boundary
		2	Use software MMR at_sw_sync for synchronizing the PHYT frame boundary
		3	Use Received frame boundary for synchronizing the PHYT frame boundary
rad_syncsel	READ_WRITE		RAD sync selection
		0	Use RP1 interface for synchronizing the RADT frame boundary
		1	Use RADTSYNC chip input for synchronizing the RADT frame boundary
		2	Use software MMR at_sw_sync for synchronizing the RADT frame boundary
		3	Use Received frame boundary for synchronizing the RADT frame boundary
		4	Use compared PHYT value for synchronizing the RADT frame boundary

Rad sync has one more selections. It is “Use compared PHYT value for synchronizing the RADT frame boundary” option. The PHY and RAD timers can be activated together using only the PHYSYNC input and the RAD timer will be synchronized after the value of time clock, which is selected on the **at_phyt_cmp_radsync** register.

The major event generation mechanisms of the Frame sync module were mask-based event generator, counter-based event generator, and trigger offset. The AIF2 AT has a different mechanism to generate events. The AT does not use mask-based event generation and all external and internal events use counter-based event generation, however, it has the same trigger offset mechanism as the Frame sync module. AT has various event input strobes like RADT_SB, RADT_FB, ULRADT_SB, ULRADT_FB, DLRADT_SB, DLRADT_FB. The frame boundary strobe concept is the same as what the frame sync module was doing but the symbol boundary strobe concept gives more flexibility when generating symbol boundary events for multiple sizes of symbols.

The AT event generator adjusted trigger offset delay before making real output events. The event modulo counter can also create additional periodic events between symbol boundary input or frame boundary input. This technique enables the generation of a short period of events like a four-chip DMA event for WCDMA. Figure 6 shows the concept of an AT event generator.

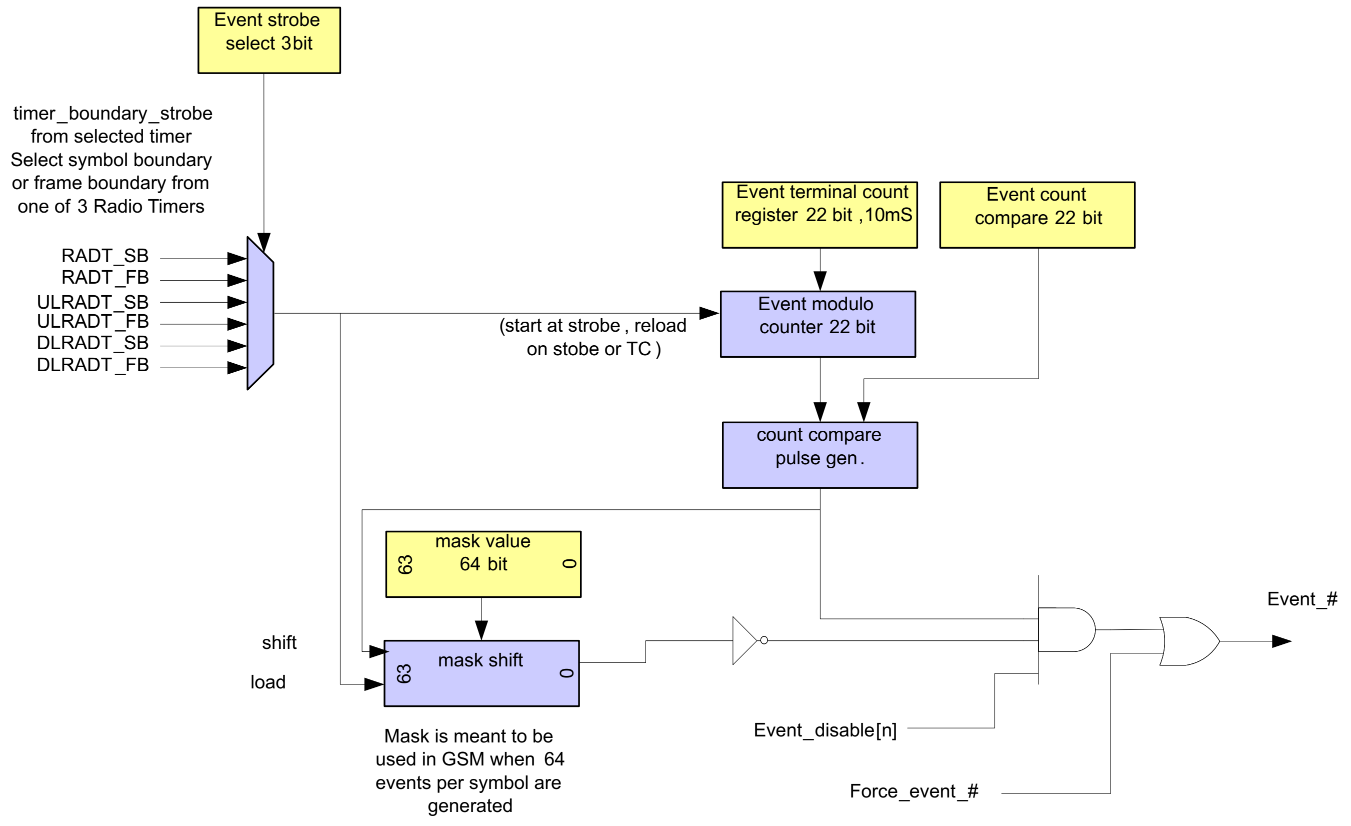
Event strobe select 3bit
timer_boundary_strobe from selected timer Select symbol boundary or frame boundary from one of 3 Radio Timers
RADT_SB
RADT_FB
ULRADT_SB
ULRADT_FB
DLRADT_SB
DLRADT_FB
Event terminal count register 22 bit ,10mS
Event count compare 22 bit
(start at strobe , reload on stobe or TC)
Event modulo counter 22 bit
count compare pulse gen.
mask value 64 bit
63
0
shift
load
mask shift
Mask is meant to be used in GSM when 64 events per symbol are generated
Event_disable[n]
Force_event_#
Event_#

Figure 6. AT Event Generator

3.2 AIF2 11 External Event

AIF2 AT has a total of 11 external events for user application programming. Event 0 ~ 7 is used for general purpose applications like EDMA trigger, WCDMA timestamp, DMA trigger, or general packet transfer trigger. The AIF1 and Frame sync module used its event 0 ~ 17 for these purposes but many of them should be used to trigger four-chip or eight-chip EDMA triggers for WCDMA. AT has a special internal Direct IO event for WCDMA, so these eight external events could be used only by a pure application layer program.

Events 8, 9, and 10 are very special. They are used to generate the heartbeat for TAC and RAC (RACA, RACB).

Event 8 is directly connected to the TAC module and TAC will receive a four-chip event from the AT to get the correct operation timing for transmission. Events 9 and 10 are used for RAC by generating a 32-chip operation event for the RAC front-end interface.

Events can be disabled or enabled on the fly while events are being generated. Disabling an event will remove the event at the next available selected timer frame boundary (frame or symbol). An event that is enabled during event generation will start generation upon its programmed offset after the next detected boundary.

The following code snippet shows how to program an external event using AIF2 CSL:

AIF2 AT external event setup

```
//AT Event setup (Event 7)
AtEventSetup.AtRadEvent[7].EventSelect = CSL_AIF2_EVENT_7;//Select Event 7
AtEventSetup.AtRadEvent[7].EventOffset = 0; //no offset
AtEventSetup.AtRadEvent[7].EvtStrobeSel = CSL_AIF2_RADT_SYMBOL;
AtEventSetup.AtRadEvent[7].EventModulo = 307199; //LTE 14 symbol time (1ms)
AtEventSetup.AtRadEvent[7].EventMaskLsb = 0xFFFFFFFF; //used for GSM
AtEventSetup.AtRadEvent[7].EventMaskMsb = 0xFFFFFFFF;
AtEventSetup.bEnableRadEvent[7] = TRUE;//Enable Event 7
```

AT external events could be used for several radio standards and that means it might support different frame sizes, symbol counts, and clock counts. Table 5 shows to set timer terminal count fields for different radio standards.

Table 5. Use of Timer Fields for Different Radio Standards

Radio Standard	Frame Count	Symbol Count	Clock Count
WCDMA	10ms Frames	Time Slots	Clocks per Slot
LTE	10ms Frames	1ms Sub-Frames	Clocks per Sub-Frame
WiMax (TDD/FDD)	Frames	Symbol Count	Clocks per Symbol
TD-SCDMA (TDD)	Frames	Symbol Count	Clocks per Symbol
GSM	60ms	Time Slots per 60ms	Clocks per Time Slot

3.3 *AIF2 Internal Event*

The AIF2 AT has three kinds of Phy timing internal events per link: PE1, PE2, and Delta event.

These three events work only with Phy timing and independent with Radio timing. PE1 and PE2 events are very similar in concept to events 18 ~ 23 and 24 ~ 29 in the Frame sync module. The PE1 event lets the Protocol Encoder know the RT preparation timing for redirection or aggregation. The PE 2 event gives channel enable timing info to PE and is helpful in calculating egress AxC offset.

The event offset value needs to be set by using the AT link configuration below and the modulo is automatically set by AIF2 CSL. Strobe selection is fixed to the frame boundary. The following code snippet shows the AIF2 AT internal physical event setup:

```
//AT link setup
AtLinkSetup.PE1Offset = 600;
AtLinkSetup.PE2Offset = 610;
AtLinkSetup.DeltaOffset = 670;
AtLinkSetup.PiMin = 670;
AtLinkSetup.PiMax = 690;
AtLinkSetup.IsNegativeDelta = FALSE;//positive delta
```

The AIF2 AT also has a special internal DMA event for Direct IO engine in the AD module. The AD is the AIF2 DMA module that communicates with the AIF2 PKTDMA module. The AD has three DIO engines for both Ingress and Egress and each engine require its own DMA event to trigger four-chip or eight-chip data chunks to transfer between PKTDMA and memory in the DSP.

Each engine has a DIO DMA event (four-chip or eight-chip) and a DIO Frame event. Those two events work together to get the optimal amount of event offset and modulo. The DIO Frame event is used to generate a large event offset because the maximum offset size cannot be larger than the modulo size (if modulo is four-chip, the max offset value should be smaller than four-chip). If the DIO DMA event offset value could be smaller than its modulo, the DIO frame event offset (set zero) does not need to be used, but for an uplink event configuration, it is required to have a very large DMA event offset.

Figure 7 shows the mechanism for those two events working together.

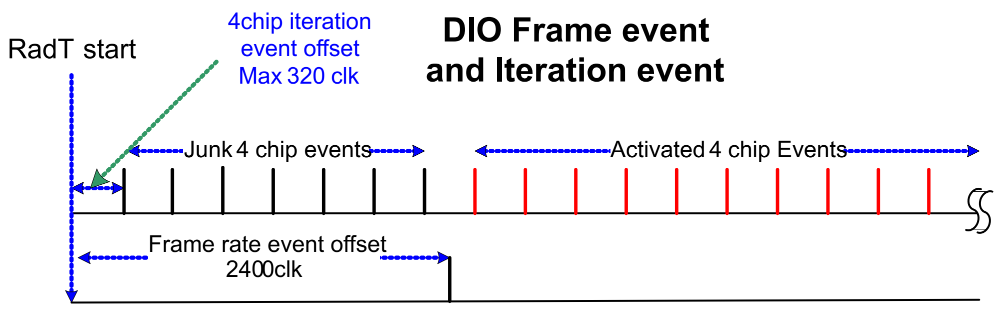
RadT start
4chip iteration event offset Max 320 clk
DIO Frame event and Iteration event
Junk 4 chip events
Activated 4 chip Events
Frame rate event offset 2400clk

Figure 7. Internal Events Pair Working Mechanism for Direct IO

The following code snippet shows how to configure a DIO event. The CSL merges the DIO event pair configuration into one to make it look simpler. The Frame event modulo is set to normal WCDMA and LTE Radio frame size (10 ms) by CSL.

```
//AT Event setup (In DIO 4chip Event and frame event)
AtEventSetup.AtIngrDioEvent[0].EventSelect = CSL_AIF2_IN_DIO_EVENT_0;
AtEventSetup.AtIngrDioEvent[0].EventOffset = 0;
AtEventSetup.AtIngrDioEvent[0].EvtStrobeSel = CSL_AIF2_RADT_FRAME;
AtEventSetup.AtIngrDioEvent[0].EventModulo = 319; //WCDMA 4 chip time
AtEventSetup.AtIngrDioEvent[0].DioFrameEventOffset = 1190;//for UL DMA timing
AtEventSetup.AtIngrDioEvent[0].DioFrameStrobeSel = CSL_AIF2_RADT_FRAME;
AtEventSetup.bEnableIngrDioEvent[0] = TRUE;//Enable In DIO Event for engine 0
```

4 Transmission Rule (Modulo, DBMR, Channel LUT)

The AIF1 was designed primarily to support WCDMA and the frame size is exactly 10 ms to transfer 38400 chip data per AxC, so 84 and 21 Look up table was adequate for OBSAI and the data packing system per link for CPRI was suitable as a transmission method. But it is hard to support several radio standards that have different sample rates from WCDMA. The AIF2 protocol encoder and decoder had to be modified for the transmission mechanism to transfer AxC data with some bubble data to enable data rate adjustment for other radio standards.

This section describes the difference between transmission rules for AIF1 and AIF2 and how to setup PD and PE registers to include AxC offset.

4.1 AIF1 Transmission Rule

This section describes how the AIF1 transmission rule works and its limitations. The number of bytes in an OBSAI RP3 message group for 1x, 2x, and 4x links is 400. This consists of 20 data slots (each has 19 bytes), 1 control slot (19 bytes), and 1 IDLE byte. Every (10 ms/1,920) the bus manager provides detailed rules for message transmission for each slot. Rules for data and control messages are provided separately. The physical layer of the bus provides counter values for the data and control message slots. Transmission of messages is done with respect to these counters.

Circuit-switched slots normally support modulos of 4, 8, or 16 for 1x, 2x, or 4x link. Packet-ssswitched message in data slots can have modulos of 1, 2, 4, 5, 8, 10, 16, 20, 40, 80 and any 2N*80 multiple. Packet-switched slots support a once-per-frame modulo: 1920 for 1x, 3840 for 2x, and 7680 for 4x link rate. The control slot can occur at any place in a message group.

To support this rule, AIF1 supports two basic look-up tables. It is 84 CNT LUT and 21 ID LUT. 84 count LUT decides the characteristic of each data slot within the max 84 number of the slot map. The 21 ID LUT decides several OBSAI header info matching and data type for each basic 20 data slots and 1 control slot. For CPRI, any combination of data type and packing are already defined by hardware and there is no flexibility to change it into a different style. Even though this scheme fully supports any kind of UMTS requirement, it is not adequate to cover non-UMTS radio standards (LTE, WiMAX, TD-SCDMA, GSM) and generic packet-mode data transmission.

To support those kinds of radio standards, the following restrictions had to be changed:

- AIF1 is basically designed for UMTS and can not fully match with other radio standard data rate.
- The current 84 LUT and 21 ID LUT cannot handle various data rate and bubble data slot
- The CPRI packing mechanism for AIF1 has fixed position and no flexibility to cover special usage for other radio data rate and pure packet data mode.
- UMTS frame timing can not-support different sizes of radio frames.

4.2 AIF2 Transmission Rule (Modulo, DBMR, Channel LUT)

To overcome the restrictions and limitations of AIF1, AIF2 introduced a new transmission mechanism into the Protocol Encoder. For OBSAI, three kinds of rules have been grouped together in sequential order: the Modulo rule, the Dual-bitmap rule, and channel-lookup table.

For CPRI, the fixed-position-data-packing mechanism is disappeared. The OBSAI standard has specified a very powerful control data or packet data approach. AIF2 fully supports OBSAI requirements and overlays some of the supporting hardware with CPRI usage. In particular, CPRI also allows for very high bandwidth by allowing unused AxC slots to pass control data or generic packet data with a specially modified Dual-bitmap rule and DBMX ID LUT.

4.2.1 OBSAI Transmission Rules

OBSAI supports 64 modulo rules and each modulo rule is connected to each Dual-bitmap rule (DBMR); these 64 modulo rules are shareable among all six links. In AIF1, the AxC number of modulo rules has to be set up to configure each data slot. In AIF2, each link requires only two modulo rules: one for AxC data and the other for control data. It is possible to assign additional modulo rules for special cases, but normally one modulo and one DBMR pair can handle most AxC data or control data transfers for the specific link.

The basic concept of DBMR looks like the following:

- AIF2 extends some of the Dual-Bitmap FSM fields to extend the capabilities of this feature
- AIF2 allows the “Bubble” output of the Dual-Bitmap FSM to be mapped to packet-switched traffic
- Dual Bitmap Rules can map to either CirSw or PktSw streams and can even support a mixture
- Max X (AxC number)
  - Normal use, may not exceed 64. (May not exceed 63 if bubble bandwidth is being used)
  - Paired, two bitmaps can be concatenated to form a 128 LUT, consumes two adjacent rules (starting at an even indexed rule)

Normally, X means the number of AxC or number of packets supported by the link. The key concept of DBMR is usinga “Bubble” OBSAI slot to adjust the data rates of several radio standards. The bitmap detail is described in OBSAI spec 4.0, especially about LTE and WiMAX.

The output index of DBMR goes into the Channel LUT to assign specific DB channel to each dedicated OBSAI data slot. These three parts are the key of OBSAI transmission rule implementation; Figure 8 shows the how those three rules are connected.

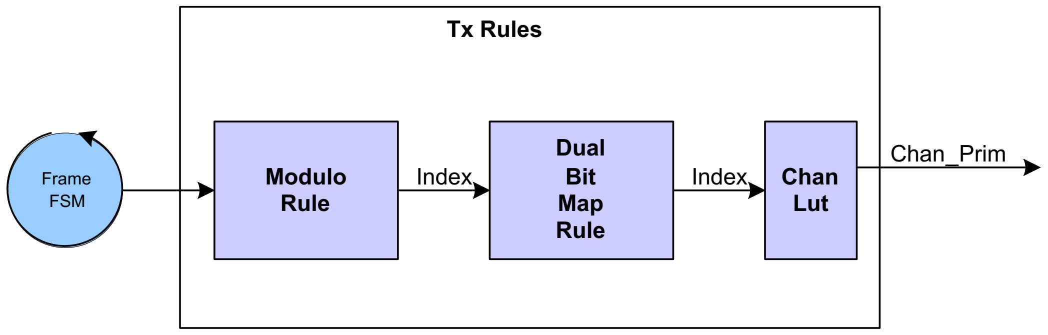

Figure 8. PE: OBSAI Transmission Rules

The Dual-Bitmap Rules (DBMR) circuit is basically a counter that circularly TDMs X channels. Every time the Modulo Rule fires, the next channel is serviced. At the end of round-robin TDM servicing of X channels, a gap of one message is added by the programmed MMR. The gap is controlled by the “Bitmaps”.

The Bubble FSM consists of a state machine following the Dual Bitmap algorithm. In every state, one bit of either of the two “Bitmaps” indicates:

- 1’b1: after the “X” count, one additional count prior to rewinding X count
- 1’b0: after the “X” count, simple start over again

The extra “count” is referred to here as a “Bubble”. During this phase of the count, the output reflects this “Bubble” condition and will be padded with zero or other type of data if you wants to use the bandwidth for other packet data transfer.

The channel LUT uses mapping transmission rule indexes from the 64 rules into DB channels, is split across eight different RAMs. The partition of this function facilitates the reuse of the LUT for CPRI mode where a LUT is dedicated per link. Figure 9 shows how eight pieces separated DBMR is matched with the Channel LUT.

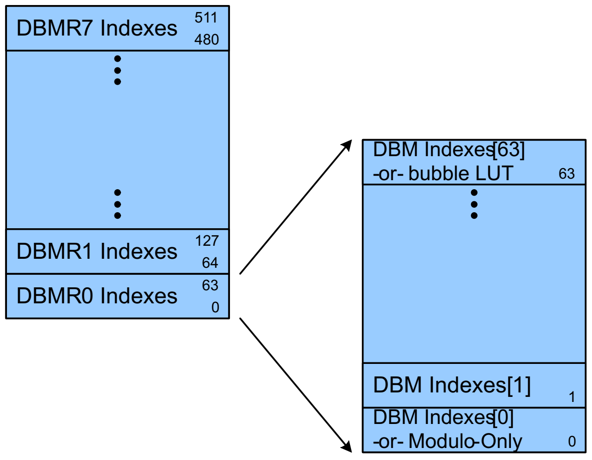

Figure 9. OBSAI Channel Lookup Table

4.2.2 CPRI Transmission Rules

AIF2 uses the OBSAI Dual Bitmap FSM (DBMF) concept for configuring the use of CPRI BW between AxC. This is a very special use case that is not specified in CPRI protocol specification. The DBMF is essentially a simple round robin TDM of AxC with the addition of a programmable bubble insertion at the end of each cycle of round robin. CPRI DBM has one clear difference to the OBSAI DBM. CPRI DBM uses a 32-bit or 30-bit sample (16-bit or 15-bit I, Q) as X value instead of a 16-byte OBSAI data slot and this is also applied to bubble data size.

The DBMF algorithm calculates values through the bitmaps—one bit per burst of AxC samples. If the bit is 0x1, a burst of bubbles (zeros) is inserted before the next burst of AxC samples. Bitmap1 will repeat several programmable times, followed by one sequence of Bitmap 2. The used length of map1 and map2 is programmable. If map2 is programmed to a length of 0, map2 goes unused.

The CPRI transmission rule does not use the Modulo rule, which is used for OBSAI as a base rule of DBMR; instead, CPRI uses a packet data pattern and each link has its own DBM rule, so it does not need to use an index or link number setup like the OBSAI Modulo rule. CPRI also uses channel LUT 0 ~ 5 for each link and uses only 128 rules from each LUT instead of using 512 rules for OBSAI.

The channel LUT used mapping transmission rule indexes from the 64 rules into DB channels is split across eight different RAMs. The partition of this function facilitates the reuse of the LUT for CPRI mode where a LUT is dedicated per link.

CPRI RAM usage:

- Link0: Ram0, 0-127 (address 128-511 unused)
- Link1: Ram1, 0-127
- …
- Link5 Ram5, 0-127
- Ram 6 and 7 are unused in CPRI

4.2.3 Transmission Rule Setup Example

This section describes how to configure PE, PD to transfer different data combinations of the DMA Channel, DBMR, and Channel LUT setup for both OBSAI and CPRI.

4.2.3.1 OBSAI

Example 1

4x link rate, WCDMA 16 AxC channel, X = 16 (X means DBM X value)

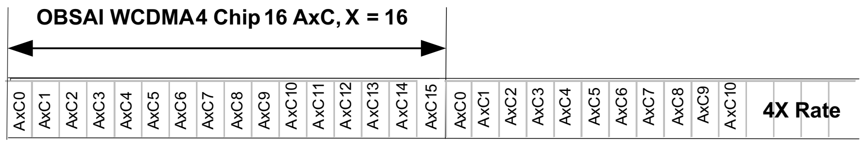

Figure 10. OBSAI WCDMA Example

Use only one modulo rule for all 16 AxC channels and 16 for X value, so each AxC channel four-chip data is matched with the X number. For OBSAI, the X number means the number of OBSAI message slots for the connected modulo rule. AIF2 Channel LUT has a total of 4096 tables. This example consumes 16 of those tables and does not show control word slot usage.

Example 2

4x link rate, 20 MHz LTE Option 1 for 2 AxC channel, X = 2

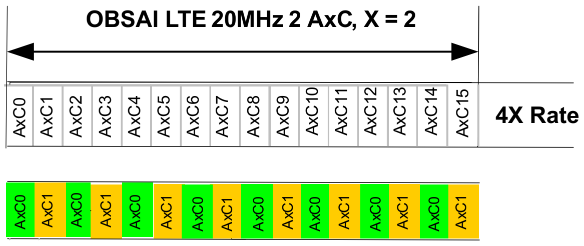

Figure 11. OBSAI LTE Option 1 Example

This example shows 20 MHz LTE usage with two AxCs and X = 2 condition. Only two channel LUT tables could be set for AxC channel 0 and 1. In this case, four samples of each AxC data is transferred alternately and that allows smooth data transmission without showing much time gap between channels.

Example 3

4x link rate, 20 MHz LTE Option 2 for 2 AxC channel, X = 16

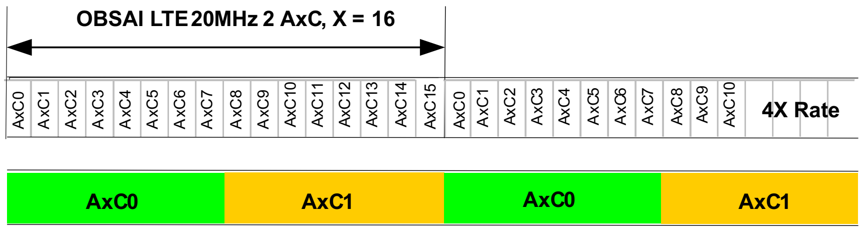

Figure 12. OBSAI LTE Option 2 Example

This example shows 20 MHz LTE usage with two AxCs and X = 16 condition. In this case, the first seven OBSAI slots transfer AxC0 data and the other seven OBSAI slots transfer AxC1 data. The first seven Channel LUT tables set the channel to zero and the next seven channel LUT tables set the channel to one.

This option demonstrates the flexibility of the AIF2 transmission methodology.

Example 4

4x link rate Generic Packet data transmission for one packet channel

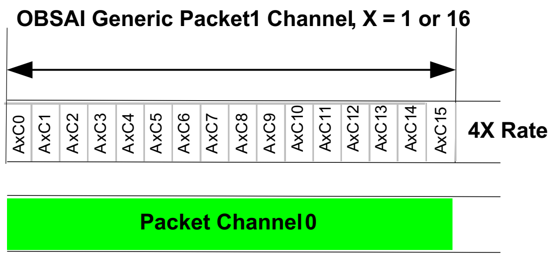

Figure 13. OBSAI Generic Packet Mode Example

This example shows the OBSAI generic-packet mode, which uses full bandwidth for only one channel. To split the bandwidth into multiple channels, channel LUT can be configured to use multiple DB channels. For Packet mode, full data bandwidth is not required. The AIF2 PE does not care about AxC offset and radio timing when the channel mode is set to packet mode. Only valid packet data marked by SOP and EOP signal will be transferred.

4.2.3.2 CPRI

Example 1

4x link rate WCDMA 16 bit IQ, 15 AxC channel, X = 15

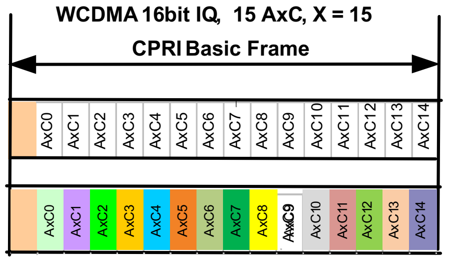
WCDMA 16bit IQ, 15 AxC, X = 15
CPRI Basic Frame
AxC0 AxC1 AxC2 AxC3 AxC4 AxC5 AxC6 AxC7 AxC8 AxC9 AxC10 AxC11 AxC12 AxC13 AxC14
AxC0 AxC1 AxC2 AxC3 AxC4 AxC5 AxC6 AxC7 AxC8 AxC9 AxC10 AxC11 AxC12 AxC13 AxC14

Figure 14. CPRI WCDMA Example

X means the number of message slots in OBSAI, but CPRI uses the X value as sample iteration count. There are four kinds of sample sizes in CPRI (7-bit, 8-bit, 15-bit, 16-bit IQ sample). This example shows 16-bit IQ sample case, so the total 15 AxC channels sample data (one chip) could be packed within one CPRI basic frame in case of 4x link speed. CPRI does not use the modulo rule but DBMR is supported per link and CPRI ID LUT is applied to each link to configure each X sample slot. Channel LUT usage is the same as OBSAI.

Example 2

4x link rate LTE 20 MHz 16 bit IQ, 2 AxC channel, X = 2

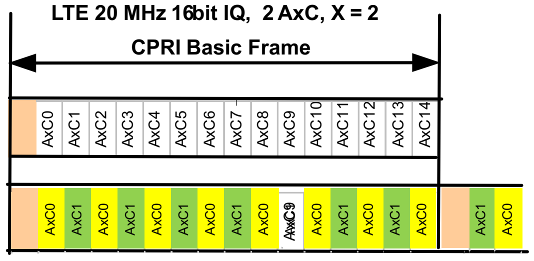
LTE 20 MHz 16bit IQ, 2 AxC, X = 2
CPRI Basic Frame
AxC0 AxC1 AxC2 AxC3 AxC4 AxC5 AxC6 AxC7 AxC8 AxC9 AxC10 AxC11 AxC12 AxC13 AxC14
AxC0 AxC1 AxC0 AxC1 AxC0 AxC1 AxC0 AxC1 AxC0 AxC9 AxC0 AxC1 AxC0 AxC1 AxC0
AxC1 AxC0

Figure 15. CPRI LTE 20 MHz Example

This example uses X = 2 (like OBSAI example Option 1), so the CPRI ID LUT [0 ~ 1] register should be set for each AxC0 slot and AxC1 slot.

Example 3

4x link rate TD-SCDMA 16 bit IQ, 5 AxC channel, X = 40 Option 1

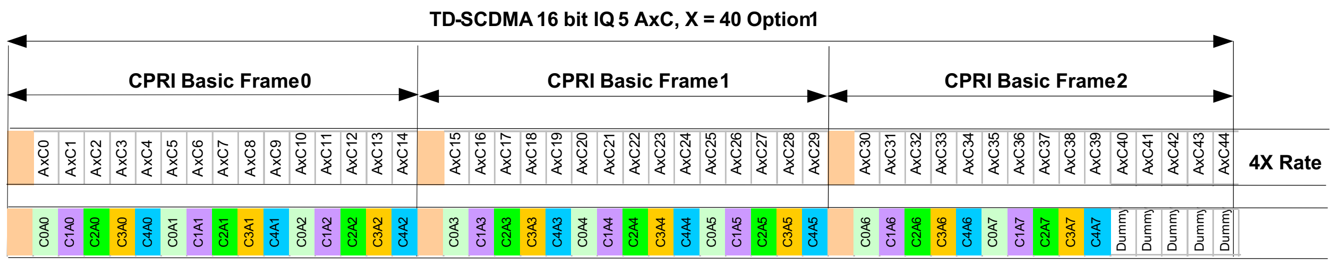
TD-SCDMA 16 bit IQ 5 AxC, X = 40 Option1
CPRI Basic Frame0
CPRI Basic Frame1
CPRI Basic Frame2
AxC0 AxC1 AxC2 AxC3 AxC4 AxC5 AxC6 AxC7 AxC8 AxC9 AxC10 AxC11 AxC12 AxC13 AxC14
AxC15 AxC16 AxC17 AxC18 AxC19 AxC20 AxC21 AxC22 AxC23 AxC24 AxC25 AxC26 AxC27 AxC28 AxC29
AxC30 AxC31 AxC32 AxC33 AxC34 AxC35 AxC36 AxC37 AxC38 AxC39 AxC40 AxC41 AxC42 AxC43 AxC44
4X Rate
C0A0 C1A0 C2A0 C3A0 C4A0 C0A1 C1A1 C2A1 C3A1 C4A1 C0A2 C1A2 C2A2 C3A2 C4A2
C0A3 C1A3 C2A3 C3A3 C4A3 C0A4 C1A4 C2A4 C3A4 C4A4 C0A5 C1A5 C2A5 C3A5 C4A5
C0A6 C1A6 C2A6 C3A6 C4A6 C0A7 C1A7 C2A7 C3A7 C4A7 Dummy Dummy Dummy Dummy Dummy

Figure 16. CPRI TD-SCDMA Option 1 Example

This example shows a special configuration for the TD-SCDMA five-AxC channel case. TD-SCDMA consumes three CPRI basic frames to transfer eight samples per AxC and it has five dummy samples at the end. The dummy samples are considered bubble data slot and configured by DBMR registers. Option 1 is sample level interleaved model and that can be set by using the PE channel LUT and CPRI ID LUT for the link.

Example 4

4x link rate TD-SCDMA 16 bit IQ, 5 AxC channel, X = 40 Option 2

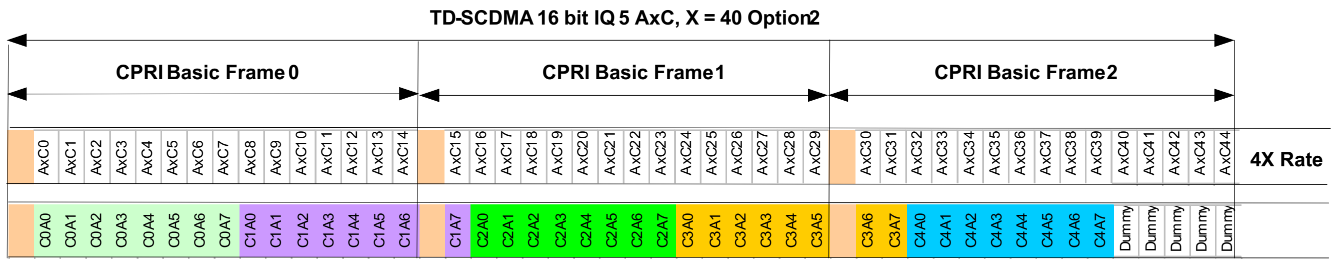
TD-SCDMA 16 bit IQ 5 AxC, X = 40 Option2
CPRI Basic Frame 0
CPRI Basic Frame 1
CPRI Basic Frame 2
AxC0 AxC1 AxC2 AxC3 AxC4 AxC5 AxC6 AxC7 AxC8 AxC9 AxC10 AxC11 AxC12 AxC13 AxC14
AxC15 AxC16 AxC17 AxC18 AxC19 AxC20 AxC21 AxC22 AxC23 AxC24 AxC25 AxC26 AxC27 AxC28 AxC29
AxC30 AxC31 AxC32 AxC33 AxC34 AxC35 AxC36 AxC37 AxC38 AxC39 AxC40 AxC41 AxC42 AxC43 AxC44
4X Rate
C0A0 C0A1 C0A2 C0A3 C0A4 C0A5 C0A6 C0A7 C1A0 C1A1 C1A2 C1A3 C1A4 C1A5 C1A6
C1A7 C2A0 C2A1 C2A2 C2A3 C2A4 C2A5 C2A6 C2A7 C3A0 C3A1 C3A2 C3A3 C3A4 C3A5
C3A6 C3A7 C4A0 C4A1 C4A2 C4A3 C4A4 C4A5 C4A6 C4A7 Dummy Dummy Dummy Dummy Dummy

Figure 17. CPRI TD-SCDMA Option 2 Example

Option 2 is a non-sample level interleaving model. It transfers all eight samples for AxC0 first, then eight samples for AxC1, and so on. PE channel LUT and CPRI ID LUT configuration is different from Option 1.

4.3 *PE, PD setup (OBSAI)*

The AIF2 PD, PE configuration has three major parts: link setup, global setup, and channel setup. Each link might have a different setup like PD OBSAI type LUT and PE_DB delay. For CPRI, DBMR and most transmission-rule setup is done in the link setup, but OBSAI does this in a common setup (global setup and channel setup). The following code snippet shows how to setup each part for OBSAI.

4.3.1 PD Link Setup

```
//PD link setup
PdLinkSetup.Crc8Poly = CRC8_POLY;
PdLinkSetup.Crc8Seed = CRC8_SEED;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].ObsaiTsFormat = CSL_AIF2_TSTAMP_FORMAT_NORM_TS;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].PdCrcType = CSL_AIF2_CRC_16BIT;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].bEnableCrc = FALSE;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].PdObsaiMode = CSL_AIF2_PD_DATA_AXC;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].bEnableEnetStrip = FALSE;
PdLinkSetup.PdTypeLut[OBSAI_TYPE_WCDMA_FDD].bEnableCrcHeader = FALSE;
```

The CRC 8 poly and seed is only used when the CRC 8 mode is enabled. The OBSAI type LUT is important for PD to differentiate radio standard.

4.3.2 PE Link Setup

```
//PE link setup
PeLinkSetup.PeCppiDioSel = CSL_AIF2_DIO;
PeLinkSetup.Crc8Poly = CRC8_POLY;
PeLinkSetup.Crc8Seed = CRC8_SEED;
PeLinkSetup.bEnObsaiBubbleBW = FALSE;
PeLinkSetup.PeDelay = DB_PE_DELAY_OBSAI;//28 sys_clks delay between DB and PE
```

4.3.3 PD Global Setup

```
//PD global setup
PdCommonSetup.PdCppiDioSel = CSL_AIF2_DIO;//DIO
PdCommonSetup.AxCOffsetWin = AXC_OFFSET_WIN;//AxC offset window
PdCommonSetup.PdRadtTC = 3071999;// Radio frame size for OBSAI
PdCommonSetup.PdFrameTC[0].FrameIndexSc = 0;//start index
PdCommonSetup.PdFrameTC[0].FrameIndexTc = 0;//termical index
PdCommonSetup.PdFrameTC[0].FrameSymbolTc = 14;//15 slots for WCDMA
PdCommonSetup.PdFrameMsgTc[0] = 639;
```

PD and PE has its own framing timer inside. It is mainly used to check frame and symbol timing for several radio standards. PdRadtTC means the total radio frame size and FrameMsgTc is used to count OBSAI message number within one slot or symbol. This counter works independently to AT Radio timing.

4.3.4 PE Global Setup

```
//PE global setup
PeCommonSetup.PeTokenPhase = 0;
PeCommonSetup.EnetHeaderSelect = 1;//bit order for Ethernet preamble and SOF
PeCommonSetup.GlobalDioLen = CSL_AIF2_DB_DIO_LEN_128;
PeCommonSetup.PeFrameTC[0].FrameIndexSc = 0;//start index
PeCommonSetup.PeFrameTC[0].FrameIndexTc = 0;//termical index
PeCommonSetup.PeFrameTC[0].FrameSymbolTc = 14;//Set 14 for PeCommonSetup.PeFrameMsgTc[0] = 639;

//modulo rule 0 setup
PeCommonSetup.PeModuloTc[0].bEnableRule = TRUE;
PeCommonSetup.PeModuloTc[0].RuleModulo = 0;//modulo termical count (Modulo -1)
PeCommonSetup.PeModuloTc[0].bRuleObsaiCtlMsg = FALSE;
PeCommonSetup.PeModuloTc[0].RuleIndex = 0;
PeCommonSetup.PeModuloTc[0].RuleLink = CSL_AIF2_LINK_0;

//DBM rule 0 setup
PeCommonSetup.PeObsaiDualBitMap[0].DbmX = 31;//DbmX number. set X-1 value
PeCommonSetup.PeObsaiDualBitMap[0].DbmXBubble = 0;
PeCommonSetup.PeObsaiDualBitMap[0].Dbm1Mult = 0;//Dbm1 repetition number
PeCommonSetup.PeObsaiDualBitMap[0].Dbm1Size = 0;//Dbm1 size (1 ~ 100)
PeCommonSetup.PeObsaiDualBitMap[0].Dbm1Map[0] = 0x0;// no bubble
PeCommonSetup.PeObsaiDualBitMap[0].Dbm2Size = 0; //Dbm2 size (0 ~ 70)
PeCommonSetup.PeObsaiDualBitMap[0].Dbm2Map[0] = 0x0;
```

PE global setup includes PE frame timer setup and modulo, DBMR setup.

4.3.5 PD Channel Setup

```
//PD channel setup
PdCommonSetup.PdRoute[i].RouteTs = 0;//Route OBSAI time stamp
PdCommonSetup.PdRoute[i].RouteType = OBSAI_TYPE_WCDMA_FDD;//Route OBSAI type
PdCommonSetup.PdRoute[i].RouteAddr = i;//Route OBSAI address
PdCommonSetup.PdRoute[i].RouteLink = CSL_AIF2_LINK_0;//Route link
PdCommonSetup.PdRoute[i].RouteMask = CSL_AIF2_ROUTE_MASK_NONE;//Route TS mask
PdCommonSetup.PdChConfig[i].bChannelEn = TRUE;//Channel enable
PdCommonSetup.PdChConfig[i].DataFormat = CSL_AIF2_LINK_DATA_TYPE_NORMAL;
PdCommonSetup.AxCOffset[i] = 610; //same offset like Egress
PdCommonSetup.PdChConfig1[i].bTsWatchDogEn = FALSE;//disable watchdog
PdCommonSetup.PdChConfig1[i].DataFormat = CSL_AIF2_GSM_DATA_OTHER;
PdCommonSetup.PdChConfig1[i].FrameCounter = 0;//framing counter group number
PdCommonSetup.PdChConfig1[i].DioOffset = 0;//Use zero offset for simple test
PdCommonSetup.PdChConfig1[i].TddEnable = 0xFFFF;//enables all symbols (FDD)
PdCommonSetup.TddEnable1[i] = 0xFFFFFFFF;//enables all symbols (FDD)
PdCommonSetup.TddEnable2[i] = 0xFFFFFFFF;//enables all symbols (FDD)
PdCommonSetup.TddEnable3[i] = 0xFFFFFFFF;//enables all symbols (FDD)
PdCommonSetup.TddEnable4[i] = 0xFFFFFFFF;//enables all symbols (FDD)
```

PD channel setup has special characteristics of each PD channel include OBSAI routing data like timestamp, address, type, and link number. PD compares all value and routes the data payload to the dedicated DB channel.

4.3.6 PE Channel Setup

```
//PE channel setup
PeCommonSetup.bEnableCh[i] = TRUE;//Enable PE channel
PeCommonSetup.PeDmaCh0[i].bCrcEn = FALSE;//disable CRC
PeCommonSetup.PeDmaCh0[i].FrameTC = 0;//use framing terminal count 0
PeCommonSetup.PeDmaCh0[i].RtControl = CSL_AIF2_PE_RT_INSERT;
PeCommonSetup.PeDmaCh0[i].CrcType = CSL_AIF2_CRC_8BIT;//CRC type
PeCommonSetup.PeDmaCh0[i].isEthernet = FALSE;//AxC data
PeCommonSetup.PeDmaCh0[i].CrcObsaiHeader = FALSE;
PeCommonSetup.PeInFifo[i].SyncSymbol = 0;//Sync symbol offset
PeCommonSetup.PeInFifo[i].MFifoWmark = 2;//Message FIFO water mark
PeCommonSetup.PeInFifo[i].MFifoFullLevel = 3;//Message FIFO full level
PeCommonSetup.PeAxcOffset[i] = 611;
PeCommonSetup.PeChObsaiType[i] = OBSAI_TYPE_WCDMA_FDD;//OBSAI header type
PeCommonSetup.PeChObsaiTS[i] = 0;//OBSAI header Time Stamp
PeCommonSetup.PeChObsaiAddr[i] = i;//OBSAI header address
PeCommonSetup.PeChObsaiTsMask[i] = CSL_AIF2_ROUTE_MASK_NONE;
PeCommonSetup.PeChObsaiTsfomat[i] = CSL_AIF2_TSTAMP_FORMAT_NORM_TS;
PeCommonSetup.PeObsaiPkt[i] = FALSE;//Select OBSAI AxC or packet mode
PeCommonSetup.PeBbHop[i] = FALSE;

//channel rule LUT setup
PeCommonSetup.ChIndex0[i] = i; //channel 0 ~ 3
PeCommonSetup.bEnableChIndex0[i] = TRUE;//Route egress channel 0 ~ 3 dbm rule to modulo rule 0
```

PE channel setup works opposite to the way that PD does. It configures the OBSAI channel route data and channel specific data for transmission.

AxC offset is a new concept for AIF2 only. AxC offset means that the coaxial cable time delay as an extension of the air propagation delay where the sampling at the RF card is considered to be time zero. AxC offset also decides the PD,PE channel data on/off timing. PE frame data generation for that channel cannot be done if the PE channel is not enabled and AxC offset is the enable/disable switch for each channel. Even though the external delay is zero, the AxC offset could be bigger than zero because PE, PD requires some delay for DMA and PHY-level operation timing like PE event, Delta, and Pi.

The programming of AxC offset is a fixed value. For each hop of the daisy chain, the AxC offset is programmed as a different value to compensate for the time propagation through each daisy chain node. If Ingress AxC offset is zero, PD will start processing when it detects the first AxC data within the AxC offset window boundary. If PD failed to find the first correct AxC data within the window, the PD link will not work until it meets the next radio frame boundary. On the Egress side, the minimum AxC offset can not normally be zero because the PE channel should be turned on after the channel data transfer is ready; so the min AxC offset will be the PE2 Event offset plus one and additional fiber delay (four-chip, eight-chip, …) would be added based on that value.

4.4 PE, PD setup (CPRI)

CPRI also has three major PD, PE configuration parts but link configuration has DBMR, which is different to OBSAI that has its DBMR in global configuration part. The following code snippet shows how to setup each part for CPRI.

4.4.1 PD Link Setup

```
//PD link setup
PdLinkSetup.CpriEnetStrip = 1;//enable ethernet strip
PdLinkSetup.Crc8Poly = CRC8_POLY;
PdLinkSetup.Crc8Seed = CRC8_SEED;
PdLinkSetup.CpriCwNullDelimitor = 0xFB;//K 27.7 character
PdLinkSetup.CpriCwPktDelimitor[0] = CSL_AIF2_CW_DELIM_NULLDELM;//4
PdLinkSetup.PdCpriCrcType[0] = CSL_AIF2_CRC_16BIT;
PdLinkSetup.bEnableCpriCrc[0] = TRUE;//enable CPRI CRC
PdLinkSetup.PdPackDmaCh[0] = 124;//Set DB channel 124 as a DMA channel PdLinkSetup.bEnablePack[0]
= FALSE;//enable CPRI control channel 0 packing
```

```
//set DBMR for link
PdLinkSetup.PdCpriDualBitMap.DbmX = 15;//set X-1
PdLinkSetup.PdCpriDualBitMap.DbmXBubble = 1;//2 bubbles of 1 AxC sample
PdLinkSetup.PdCpriDualBitMap.Dbm1Mult = 0;//set n-1
PdLinkSetup.PdCpriDualBitMap.Dbm1Size = 0;//set n-1
PdLinkSetup.PdCpriDualBitMap.Dbm1Map[0] = 0x0;
PdLinkSetup.PdCpriDualBitMap.Dbm2Size = 0;
PdLinkSetup.PdCpriDualBitMap.Dbm2Map[0] = 0x0;

//DbmX ID LUT setup
PdLinkSetup.CpriDmaCh[i]= i; //match Dbm X to PD channel num
PdLinkSetup.bEnableCpriX[i]= TRUE; //enable CPRI Dbm X slot
PdLinkSetup.bEnableCpriPkt[i]= FALSE;//use AxC data mode
PdLinkSetup.Cpri8WordOffset[i]= 0;//more detailed CPRI AxC offset
```

For packet-switched data transfer like Ethernet, the Control word and generic packet mode use four DB channels per link. Any four of 128 channels can be assigned, but four is the maximum.

4.4.2 PE Link Setup

```
//PE link setup
PeLinkSetup.PeCppiDioSel = CSL_AIF2_DIO;
PeLinkSetup.Crc8Poly = CRC8_POLY;
PeLinkSetup.Crc8Seed = CRC8_SEED;
PeLinkSetup.PeDelay = DB_PE_DELAY_CPRI;
//set DBMR for link
PeLinkSetup.PeCpriDualBitMap.DbmX = 15;//16 set X-1
PeLinkSetup.PeCpriDualBitMap.DbmXBubble = 1;//2 bubbles of 1 AxC sample
PeLinkSetup.PeCpriDualBitMap.Dbm1Mult = 0;//set n-1
PeLinkSetup.PeCpriDualBitMap.Dbm1Size = 0;//set n-1
PeLinkSetup.PeCpriDualBitMap.Dbm1Map[0] = 0x0;
PeLinkSetup.PeCpriDualBitMap.Dbm2Size = 0;
PeLinkSetup.PeCpriDualBitMap.Dbm2Map[0] = 0x0;
...
PeLinkSetup.CpriAxCPack = CSL_AIF2_CPRI_15BIT_SAMPLE;
PeLinkSetup.CpriCwNullDelimitor = 0xFB;//K 27.7 character
PeLinkSetup.CpriCwPktDelimitor[0] = CSL_AIF2_CW_DELIM_NULLDELM;
PeLinkSetup.PePackDmaCh[0] = 124;
PeLinkSetup.bEnablePack[0] = FALSE;
```

PE link setup looks very similar to PD link setup.

4.4.3 PD Global Setup

```
//PD global setup
PdCommonSetup.PdCppiDioSel = CSL_AIF2_DIO;//DIO
PdCommonSetup.AxCOffsetWin = AXC_OFFSET_WIN;//AxC offset window
PdCommonSetup.PdRadtTC = 2457599;// Radio frame size for CPRI
PdCommonSetup.PdFrameTC[0].FrameIndexSc = 0;//start index
PdCommonSetup.PdFrameTC[0].FrameIndexTc = 0;//teminal index
PdCommonSetup.PdFrameTC[0].FrameSymbolTc = 14;//15 slots for WCDMA
PdCommonSetup.PdFrameMsgTc[0] = 639; // 640 CPRI quad samples (16 byte) are in WCDMA slot time
```

CPRI PD, PE setup also has its own framing timer inside. In OBSAI, FrameMsgTc was the count of OBSAI message slot (16 byte payload) but CPRI frameMsgTc counts the number of samples (30-bit or 32-bit data). The PD count unit is quad samples but the PE count unit is just sample, so frameMsgTc value for PD and PE will be different. (For OBSAI, it uses same unit for both sides.)

4.4.4 PE Global Setup

```
//PE global setup
PeCommonSetup.PeTokenPhase = 0;//Phase alignment for scheduling DMA
PeCommonSetup.EnetHeaderSelect = 1;//bit order for Ethernet preamble and SOF
PeCommonSetup.GlobalDioLen = CSL_AIF2_DB_DIO_LEN_128;
PeCommonSetup.PeFrameTC[0].FrameIndexSc = 0;//start index
PeCommonSetup.PeFrameTC[0].FrameIndexTc = 0;//teminal index
PeCommonSetup.PeFrameTC[0].FrameSymbolTc = 14;//Set 14 for WCDMA
PeCommonSetup.PeFrameMsgTc[0] = 2559;//2560 CPRI samples (4 byte) are in WCDMA slot time
```

4.4.5 PD Channel Setup

```
//PD channel setup
PdCommonSetup.PdChConfig[i].bChannelEn = TRUE;//Channel enable
PdCommonSetup.PdChConfig[i].DataFormat = CSL_AIF2_LINK_DATA_TYPE_NORMAL;
PdCommonSetup.AxCOffset[i] = 0;// same to Egress AxC offset
PdCommonSetup.PdChConfig1[i].bTsWatchDogEn = FALSE;//disable watchdog
PdCommonSetup.PdChConfig1[i].DataFormat = CSL_AIF2_GSM_DATA_OTHER;
PdCommonSetup.PdChConfig1[i].FrameCounter = 0;//framing counter group number
PdCommonSetup.PdChConfig1[i].DioOffset = 0;//Use zero offset for simple test
PdCommonSetup.PdChConfig1[i].TddEnable = 0xFFFF;//enables all symbols(FDD)
PdCommonSetup.TddEnable1[i] = 0xFFFFFFFF;//enables all symbols (FDD
PdCommonSetup.TddEnable2[i] = 0xFFFFFFFF;//enables all symbols(FDD
PdCommonSetup.TddEnable3[i] = 0xFFFFFFFF;//enables all symbols(FDD
PdCommonSetup.TddEnable4[i] = 0xFFFFFFFF;//enables all symbols(FDD
```

Channel setup is much simpler than the OBSAI case because CPRI gets routing info from the link setup.

4.4.6 PE Channel Setup

```
//PE channel setup
PeCommonSetup.bEnableCh[i] = TRUE;//Enable PE channel
PeCommonSetup.PeDmaCh0[i].bCrcEn = FALSE;//disable CRC
PeCommonSetup.PeDmaCh0[i].FrameTC = 0;//use framing terminal count 0
PeCommonSetup.PeDmaCh0[i].RtControl = CSL_AIF2_PE_RT_INSERT;
PeCommonSetup.PeDmaCh0[i].CrcType = CSL_AIF2_CRC_8BIT;//CRC type
PeCommonSetup.PeDmaCh0[i].isEthernet = FALSE;//AxC data
PeCommonSetup.PeInFifo[i].SyncSymbol = 0;//Sync symbol offset
PeCommonSetup.PeInFifo[i].MFifoWmark = 2;//Message FIFO water mark
PeCommonSetup.PeInFifo[i].MFifoFullLevel = 3;//Message FIFO full level
PeCommonSetup.PeAxcOffset[i] = 0;// No external AxC offset
// PE channel LUT setup
PeCommonSetup.ChIndex0[i] = i;
PeCommonSetup.bEnableChIndex0[i] = TRUE;
PeCommonSetup.CpriPktEn0[i] = FALSE;
```

For OBSAI AxC offset, the basic unit was the dual-byte clock and the counter starts from radio frame time, but CPRI AxC offset uses the sample number as the basic unit and the counter starts from PHY frame time. For more information about how to calculate AxC offset, see the *KeyStone I Architecture Antenna Interface 2 (AIF2) User's Guide* (SPRUGV7) and *KeyStone II Architecture Antenna Interface 2 (AIF2) User's Guide* (SPRUHL2).

5 DMA Methodology

There is a basic difference in the requirements and handling for WCDMA data versus all other kinds of data transport. WCDMA uses antenna data in a streaming manner with little demarcation between timeslots of frames, however, OFDM standards do processing based on groups of samples and are therefore handled as packet data. This comprises the main differences between DMA methodologies for AIF1 and AIF2. AIF1 supported circuit-switched data transport based on the WCDMA data rate and timing but AIF2 has extended features to support packet-type data streams with SOP (start of packet) and EOP (end of stream) marks. These requirements allow use of the concept of Multicore Navigator and DirectIO.

Multicore Navigator transfers data based on packet or symbol but AIF2 uses the 64-byte chunk as a basic burst size between the PKTDMA and AD module. The Packet DMA (PKTDMA) scheme is used mainly for the OFDM radio standard like LTE, WiMAX, TD-SCDMA, and GSM/Edge and it is also used for pure packet-mode data like Ethernet, control word, and generic packet transfer on OBSAI or CPRI AxC data slots. WCDMA does not use Multicore Navigator to avoid unnecessary internal processing delay; instead, it uses a DirectIO scheme, which is very close to the legacy EDMA style of AIF1.

The following sections show how AIF2 can handle packet data and circuit data efficiently by using these two new DMA methodologies.

5.1 Multicore Navigator

Multicore Navigator is a methodology and a series of hardware accelerator modules, which allow DSP cores and peripherals to transfer packets effectively. It is a safe and managed way that memory can be used to pass data. The DSP host allocates blocks of memory and configures the Multicore Navigator hardware to utilize the memory region. A key aspect of the Multicore Navigator is that the memory is broken up into small grain buffers that are link-listed together via the Multicore Navigator hardware, creating virtually any size packets.

Multicore Navigator has a large degree of flexibility. Some applications are not tolerant to breaking up packets into multiple buffers. For these applications, Multicore Navigator has an alternate operation where only a single fixed-size buffer is used per packet. Clearly the buffer size needs to be chosen in advance to handle the largest possible packet size. (Each memory region supports only one buffer size.)

5.1.1 Multicore Navigator Descriptors

Multicore Navigator has the concept of descriptors, which are a form of packet header or buffer that contains information specific to Multicore Navigator. Pointers to buffers are contained in descriptors as well as many other useful fields for both software and the PKTDMA engine. Descriptor fields vary according to the Multicore Navigator packet type being supported.

5.1.1.1 Host Mode Descriptor

As well as other information, holds an additional pointer to the “next” Host Mode Descriptor in the packet (forming a linked list). The Descriptor and Buffer are separate entities stored in separate areas of memory.

5.1.1.2 Monolithic and Monolithic Descriptor

Descriptor and Buffer are merged into one contiguous portion of memory (basically, the Buffer contains the Descriptors in the first n words). PKTDMA is aware of the offset between the descriptor and beginning of the payload.

5.1.2 Multicore Navigator Queues

Queues are at the heart of the Multicore Navigator concept. There are several types of Multicore Navigator Queues. Queues can represent either individual Descriptor/Buffer pairs or “Packets”. The Free Queue is a queue of Descriptors/Buffers, while all other queues are queues of packets. The queues of packets are a linked list of SOP Descriptor pointers only. If more than one Descriptor/Buffer Pair is used to store a packet, the linked list within the packet is contained in the Descriptors. In other words (for Host Mode), packet queues can be thought of as a double-linked list where the first order linking is contained in the queue and the second-order linking is contained in the Descriptors.

5.1.3 Multicore Navigator Scheduler

Normally QM and PKTDMA schedule the data transfer for themselves; but for AIF2, scheduling is done by the AIF2 core. As data comes into the AIF2 core, the PD aggregates the data into quad words and writes the quad words into the DB buffers. For Multicore Navigator packet data, the PD is aware of packet boundaries and identifies the end (EOP). The DB Multicore Navigator scheduling circuitry counts the Quad words. For every four Qwords or EOP, a transfer token is scheduled.

AIF2 normally performs 64-byte transfers over the VBUS and therefore accummulates four Qwords before making a transfer. Transfer requests are scheduled “first come- first served” into the token FIFO. Transfer request tokens are passed to the PKTDMA block one at a time. PKTDMA performs the appropriate DMA activity.

For Egress, PE requests Qword-size chunks of data based on the potential for data consumption. PE is constantly counting through the transmission rules. When transmission rules have dictated that PE had the potential to consume two Qwords, PE requests a DMA burst. The DB is qualifying PE DMA requests with the availability of space in the input FIFO. If DB does not have sufficient space for the transfer, the request is simply dropped. The PKTDMA performs DMA transfers only for channels that have data to be transported. PKTDMA gets a dedicated signal from the Multicore Navigator QM when a channel has data to be transported.

5.1.4 Multicore Navigator Interfaces to AIF2

Figure 18 shows the interfaces from the AIF2 core to the PKTDMA and from the PKTDMA to the Queue Manager.

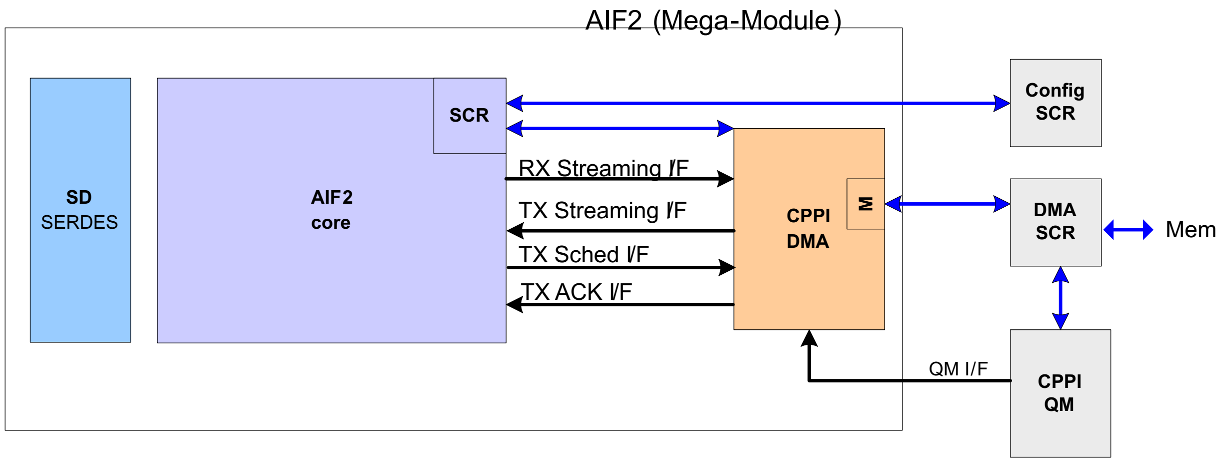

Figure 18. AIF2 Core, PKTDMA, and QM Connectivity

The PKTDMA, QM interface is simply a FIFO not empty flag per transmit DMA channel. The Queue manager supplies a bit per PKTDMA channel queue to indicate whether there is a new packet available to transmit. The transmit portion of the PKTDMA uses the QM queue status when the AIF2 Tx Scheduler issues a read request for a queue.

In the Tx Scheduler Interface, the AIF2 core controls the rate of data flow and the order in which DMA channels are serviced by supplying “requests” to the PKTDMA engine. A “request” tells the PKTDMA to move *x* number of bytes for DMA channel *y*. The PKTDMA engine actually counts through the packet, determining packet boundaries based on the Length field in the descriptor.

The data path connection between the AIF2 core and PKTDMA is the “Streaming Interface”. Ingress and Egress are independent operations with their own dedicated streaming interface. AIF2 uses 128-bit streaming interfaces. The 128-data bus is used to pass the Multicore Navigator packet payload.

5.1.5 Multicore Navigator Packet Types

Multicore Navigator is extremely general and flexible. The intended use of AIF2 is to use Multicore Navigator in a specific and limited way with only two Multicore Navigator packet types:

- Monolithic—Mainly used for {LTE, WiMax, TD-SCDMA, GSM} antenna data
- Host—Mainly used for {control, generic packet, Ethernet} traffic

Host mode has considerable performance overhead with all the descriptor and pointer operations. In Host mode, the first descriptor is known as the packet descriptor and only this descriptor is placed on a transport queue in the QM. It is not recommended to use host mode for CPRI, because CPRI needs to pack the data in a short time and it cannot withstand a long DMA time delay when compared to OBSAI. Therefore, Mono mode is highly recommended, even for Ethernet or generic packet traffic.

Multicore Navigator Monolithic packet types have the descriptor and buffer concatenated in one contiguous memory buffer. Because the descriptor and buffer are in contiguous memory, the PKTDMA operations are simplified when addressing memory. The Monolithic packet descriptor is exactly 16 bytes. Many of the fields are required by Multicore Navigator, but four bytes are allocated for protocol-specific use. The Monolithic packet type uses some of the protocol-specific bits for Radio Standardinformation like:

- Ingress/Egress
- AxC number
- Symbol Number

In the special case of GSM Baseband Hopping, where application software assigns an OBSAI address to Control Packets, the protocol-specific word contains the OBSAI address (13 bits).

If Monolithic is used for non-AxC data, AIF2 fills the Egress protocol-specific fields with zeros and the Ingress protocol-specific fields can be disabled.

5.2 Direct IO

The term DirectIO means that a peripheral has dedicated custom logic, which implements data movement. For AIF2, custom circuitry is built to handle data movement requirements unique to WCDMA. The PKTDMA module has DirectIO support features that allow AIF2 to pass VBUS reads/writes to the VBUS, using the 128-bit VBUSM master port on the PKTDMA. The AIF2 Multicore Navigator Scheduler gives DirectIO accesses higher priority than Multicore Navigator packet transfers.

DirectIO is a state machine that is triggered to transfer data when internal AT system events fire. Example transfer times could be every 4, 8, or 32 chips of time. The time granularity of the data transfer depends on UL/DL and the preferred packing at the destination. For AIF1 EDMA, dedicated AxC number of data was packed together in one of the eight parts of the Circular RAM; but AIF2 has 128 FIFOs for each AxC channel and it is delivered by the DIO engine, which could be set up by AD MMR. (AxC num, block num, burst stride, and block stride instead of A, B, C count of EDMA.)

5.2.1 Direct IO for WCDMA

- Circular Buffers
  - 32 –or- 64 chips (128 or 256 bytes)
- Ingress Destinations
  - RAC—UL data
  - DDR3—UL data for delay
  - L2—UL data for RSA processing
- Egress Source
  - TAC—DL data
  - DDR3—Delayed DL data
  - L2—RSA generated DL data

DirectIO is legacy support for WCDMA traffic. Because of the nature of WCDMA, the DMA does more circular buffering, particularly with the legacy support of the RAC hardware accelerator module that has a predefined interface. The concept of a FIFO for WCDMA data does not work very well. DirectIO uses the DB RAM as a set of circular memory regions, one for each of the required WCDMA AxC. DirectIO only supports two AIF2 DB internal buffer sizes:

- 128 byte: intended for WCDMA use
- 256 byte: intended for higher rate LTE uses

DirectIO is a precisely-timed event relative to Multicore Navigator that is a data-arrival-driven event. The AT creates internal system events that control the timing. Two separate system events control the DirectIO DMA:

- Frame rate strobe
- Iteration strobe
  - Four chips for TAC DL
  - Eight chips and 32 chips for RAC UL
  - Flexible for other burst sizes

5.2.2 DIO engine control MMRs in AD

DirectIO is fully controlled by configuring the AD registers below.

Register	Description
num_qw	Number of Qword per AxC (DL : 1 qw, UL : 2 qw)
num_axc	Number of AxCs for the specific DIO engine.
dma_base_addr	VBUS source or destination base address
dma_brst_ln	Maximum DMA burst length. Normally set to four QW
dma_ch_en	DMA channel enable/disable
rsa_en	Egress DIO data type selection (DL, UL RSA)
dma_num_blks	Number of data blocks to transfer before wrapping back to **dma_base_addr**
dma_brst_addr_stride	DMA burst address stride (in multiples of 0x10 internally) after each DMA burst (64 bytes or less), the DMA address will increment by this amount
dma_blk_addr_stride	DMA block address stride (in multiples of 0x10 internally) after transferring each DMA block (every event time), the DMA address will increment by this amount
dbcn0 ~ 63	Match dbcn order to each DB channel number

The legacy mode EDMA methodology has A, B, and C count and source, destination B index, and C index concepts and these counters and indexes are matched to the AIF2 registers below.

Register	Description
DirectIO ACnt	num_qw
DirectIO BCnt	num_axc
DirectIO CCnt	dma_num_blks
DirectIO SBIDX	dma_brst_addr_stride (Egress), No need for Ingress
DirectIO DBIDX	dma_brst_addr_stride (Ingress), No need for Egress
DirectIO SCIDX	dma_blk_addr_stride (Egress), No need for Ingress
DirectIO DCIDX	dma_blk_addr_stride (Ingress), No need for Egress

5.3 Example Multicore Navigator Usage

This section shows how to configure Ingress and Egress Multicore Navigator configurations for LTE.

5.3.1 Ingress (DMA to FFTC)

When antenna data arrives at the AIF2 Ingress side, the Rx PKTDMA of AIF2 will pop a packet descriptor from the Rx free descriptor queue (FDQ) for the antenna carrier, fill the data in the buffer, and push the packet descriptor in the output queue when data is complete for the packet.

Because FFTC is also a peripheral using Multicore Navigator, it is possible to connect the AIF2 and FFTC directly through queues. In this case, the output queue of AIF2 can be specified as one of the input queues of FFTC. When AIF2 pushes the packet descriptor into the queue, it triggers the QPEND status of the corresponding queue. The Tx PKTDMA of FFTC will pop the descriptor from the input queue and start DMA the data. When the processing is done, the Rx PKTDMA will pop a descriptor from FFTC Rx FDQ, fill the results in the buffer and push the packet descriptor to the FFTC output queue. The output queue of FFTC can be further processed by other Multicore Navigator modules or CorePac.

Figure 19 shows the process. The blue curved arrows indicate the possible routes for recycling. If, after the packet in the input queue of FFTC is retrieved by FFTC Tx PKTDMA, there are no other modules that need to use the same data, the packet can be set to return to the FDQ for AIF2 Rx, which is done automatically by FFTC Tx PKTDMA. Depending on the next stage of processing after the FFTC, either CorePac needs to recycle the FFTC output queue back to the FFTC Rx FDQ or the next Multicore Navigator module can do it automatically.

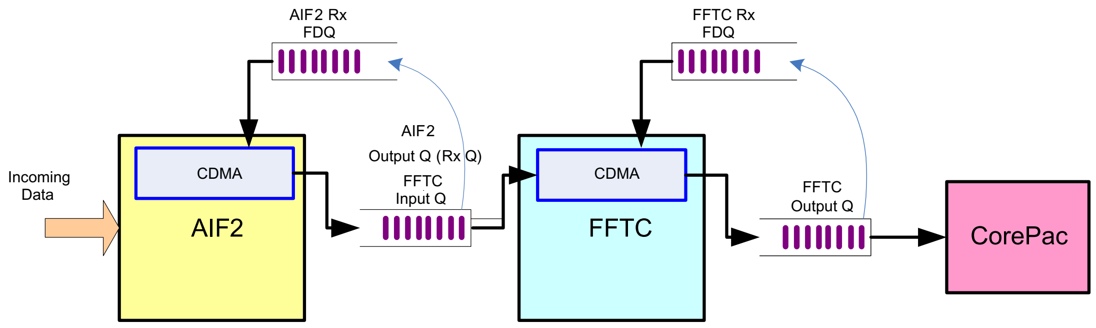

Figure 19. Ingress LTE Data Flow (Direct DMA to FFTC)

Because it is not always desirable to have the output queue of the AIF2 to be the same as the input queue of FFTC, they can be set to use different queues. (Figure 20). In this case, when a packet descriptor is pushed into the AIF2 output queue, the packet must be reconstructed by popping a descriptor from the FFTC Tx FDQ, linking the data buffer to the received antenna data, then the packet can be pushed into an FFTC input queue. This can be done using CorePac or other methods, which will be described later. When it comes to recycling, the AIF2 output queue, FFTC input queue, and FFTC output queue all need to be recycled. In this figure, only the FFTC input queue can be recycled automatically by FFTC Tx PKTDMA. The other two need to be recycled by CorePac or other methods.

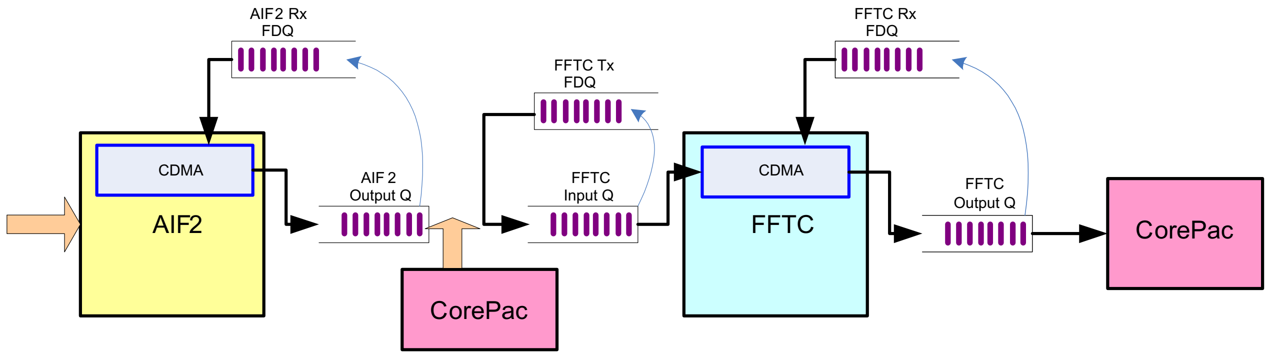

Figure 20. Ingress LTE Data Flow (CorePac Intervention Model)

5.3.2 Egress (DMA to AIF2)

On the Egress side, it is the responsibility of the host to feed the input queue of FFTC. For example, the host needs to pop a packet descriptor from Tx FDQ, configure all the fields of the packet descriptor, prepare the payload data, and push the packet descriptor into one of the input queues of FFTC. The Tx PKTDMA of FFTC will pop the descriptor, stream the data in, and recycle the packet descriptor to the queue specified in the packet descriptor.

When FFTC is done processing, it pops a packet descriptor from Rx FDQ, fills in the data, and deposits the packet descriptor into the output queue specified. If no other processing is needed before the data goes out to antenna interface, the output queue of FFTC can be one of the input queues of AIF2. After the Tx PKTDMA of AIF2 pops the packet from its input queue, it will stream the data out and recycle the packet descriptor to the queue specified in the packet descriptor. Figure 21 shows the data flow.

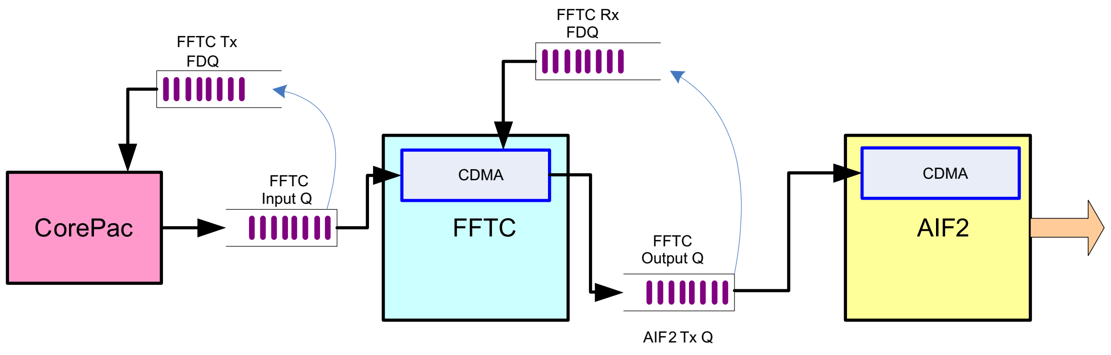

Figure 21. Egress LTE Data Flow

5.4 *Example Direct IO Usage*

This section shows how to configure Ingress and Egress AD DirectIO configurations for WCDMA.

5.4.1 RAC Example

Figure 22 shows an example of how the DIO is programmed for RAC. The DMA Event period is eight chips.

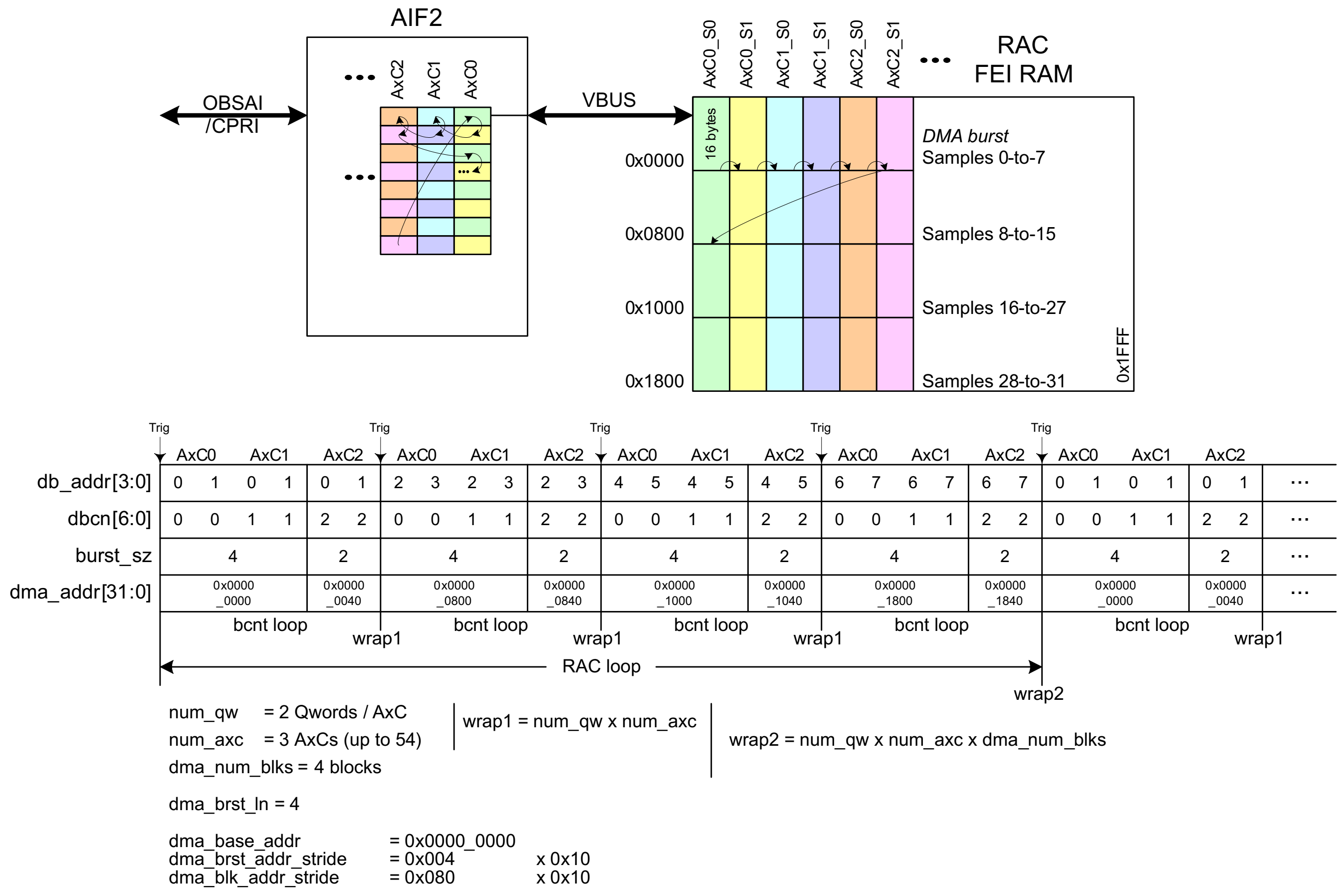

Figure 22. RAC Example

Register	Description
num_qw	Two Qwords for UL
num_axc	Three AxCs
dma_base_addr	VBUS destination base address is RAC front end interface
dma_brst_ln	Four QWords
dma_ch_en	Enable for three channels
rsa_en	Enabled for UL RSA data
dma_num_blks	Four blocks (fixed value for RAC)
dma_brst_addr_stride	Four QWords
dma_blk_addr_stride	0x80 QWords
dbcn0 ~ 63	dbcn0 ~ 2 should be set for three channels

5.4.2 TAC Example:

Figure 23 shows an example of how the DIO is programmed for TAC. The DMA Event period is four chips.

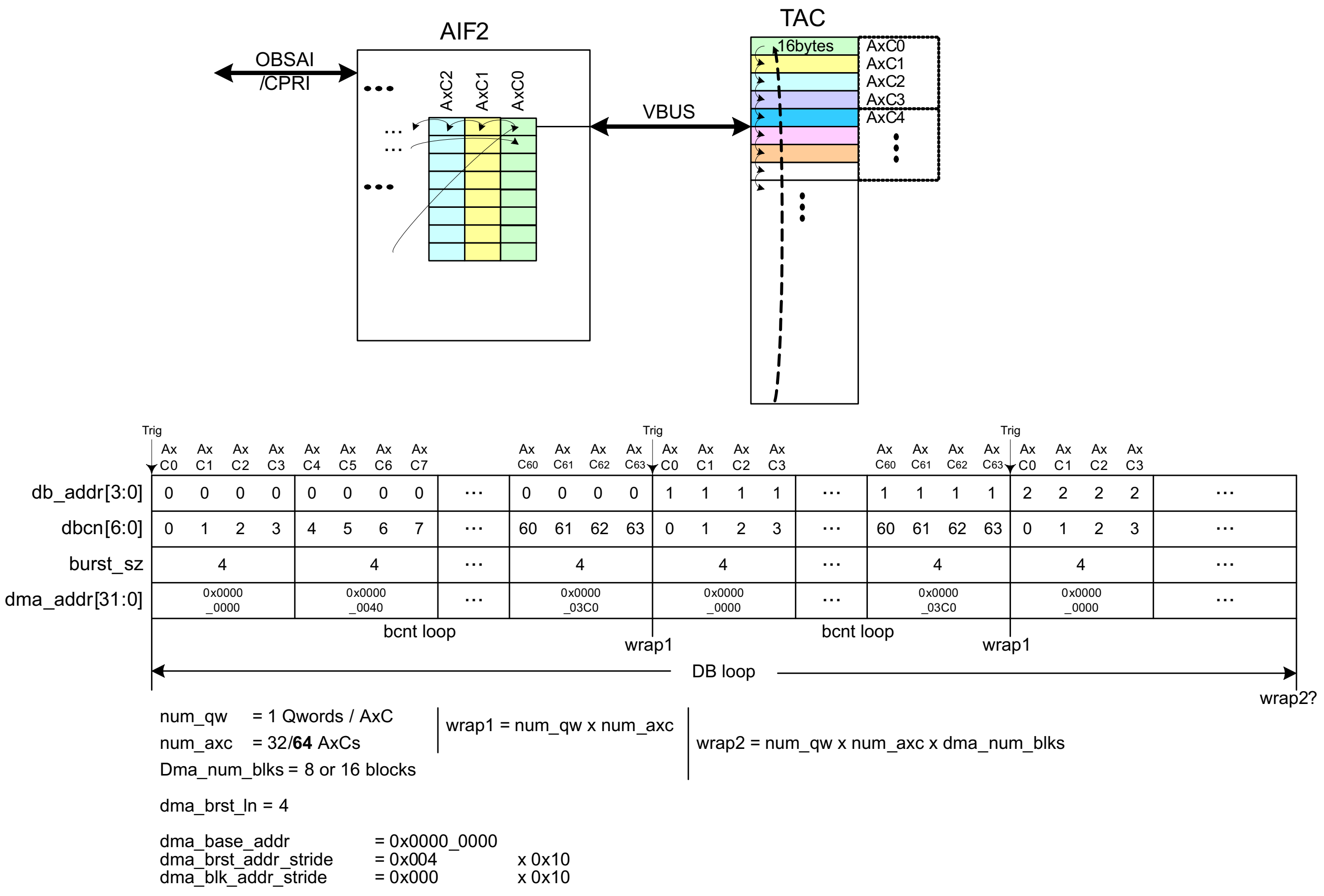

Figure 23. TAC Example

Register	Description
num_qw	One Qwords for DL
num_axc	64 AxCs
dma_base_addr	VBUS source base address is TAC back end interface
dma_brst_ln	Four QWords
dma_ch_en	Enable for 64 channels
rsa_en	Disabled
dma_num_blks	8 blocks or 16 blocks
dma_brst_addr_stride	Four QWords
dma_blk_addr_stride	Zero Qwords (No block stride for TAC)
dbcn0 ~ 63	dbcn0 ~ 63 should be set for each channel

6 Dynamic Configuration

AIF1 could only support static configuration at initialization time, so it was necessary to stop all links and reconfigure all modules again to add or delete an AxC or link. The Frame sync module also had limited functionality about event reconfiguration. AIF2 supports several forms of dynamic changes intended to support changes to the base station radio configuration. These types of changes are split into two basic categories:

- On-the-fly: Change occurs from one frame to the next without any “off” or error period of time. Requires special ping/pong buffering of configuration data where the ping/pong will toggle on the next frame boundary
- Normal Changes: An antenna carrier is torn down, later rebuilt to accomplish a change. There is the expectation that some small amount of time of no transmission will elapse.

The major functionalities of AIF2 dynamic configuration are as follows:

- Link Add/Delete (add link without resetting AIF2 timer)
- AxC Add/Delete (add or delete AxC channel)
- GSM Base Band Hopping
- LTE (TDD) and WiMax (TDD)—change UL/DL ratio (On the fly)
- AT Timing
  - System Event Add/Delete (On the fly)
  - RadT re-synchronization

Basic operation of links and antenna carriers allow Add/Delete support. On-the-fly modification is generally not supported. To perform a modify operation, delete the link/AxC followed by an add with the new desired characteristics.

When adding or deleting a Channel, there are three levels of configuration required:

- PHY level: AIF2 Links need to be enabled and preferably up and running
- Protocol level: OBSAI address, type, transmission rules setup should be properly done
- DMA level: DMA channels should be assigned and turned on (by DB, AD, PKTDMA modules)

For WCMA, AxC channel Add/Delete should incorporate with the Add/Delete of the assigned DirectIO DMA configuration. The DirectIO times the application of configuration data to the RadT frame boundary. With the PKTDMA module, you have a choice to Add/Delete Multicore Navigator Channels when performing Add/Delete of AxC, or keeping the Multicore Navigator channel always active.

The GSM Baseband hopping special requirements are not handled as reconfiguration, but rather by allowing software to direct where data goes. For AxC data, all DMA channels are allocated and software steers data into the appropriate DMA channel FIFO. For control data, software writes the destination OBSAI addressing into the Multicore Navigator packet (in protocol-specific fields).

The standards that support TDD have separate UL and DL regions. On-the-fly update of the DL/UL ration is required. On-the-fly update requires AIF2 to ping-pong buffer control information and transition to new control information at a precise timing (on frame boundary).

Table 6. On-the-fly Update Requirements

	On-the-Fly Update		
Radio Standard	**UL/DL Ratio**	**Symbol Size**	**Sample Freq-or- BW**
LTE FDD	N/A	No	No
LTE TDD	Yes	No	No
WiMax (TDD)	Yes	No	No
TD-SCDMA (TDD)	No	N/A	No

LTE supports extended and normal cyclic prefix length in symbols and multiple bandwidth. LTE bandwidth change (for example from 20 MHz to 10 MHz) or change of cyclic prefix length is a radical reconfiguration of an antenna carrier. For this change, the antenna carrier is deleted and later added with the new parameters. This is not considered on-the-fly as it requires a short period of downtime (10 ms frame time) before it is reactivated. You have a choice of handling Multicore Navigator when supporting multiple LTE bandwidths simultaneously. The easiest solution is to allocate the Multicore Navigator buffer size to accommodate the largest symbol size supported. When an antenna of lesser bandwidth accesses these buffers, only a fraction of the memory is actually used.

Changes in system-event generation are supported by first turning a system event off by AT-event enable/disable registers, reconfiguring the system event, then turning it on. It is envisioned that system events are mainly used by application software and that having an “off” period is required for the software to re-sync to the difference in system-event timing. Each system event is independent of the others and it gives the flexibility to change one without corrupting any others. The AT radio timer is synchronized at startup to an external timing source or internal software timing source. Dynamic synchronization is also supported to re-synchronize the radio timer. In this operation, the radio timer is synchronized to the next frame boundary.

7 References

- *KeyStone I Architecture Antenna Interface 2 (AIF2) User's Guide* (SPRUGV7)
- *KeyStone II Architecture Antenna Interface 2 (AIF2) User's Guide* (SPRUHL2)