Application Report

Draft Ver 1.5– Sept 2008

Using the TCI648x Antenna Interface (AIF) for inter-DSP communication

Jelena Nikolic-Popovic, Albert Bae — Communication Infrastructure

ABSTRACT

The TMS320TCI6487x Antenna Interface (AIF) is a CPRI and OBSAI-compliant peripheral whose primary purpose is to transfer baseband antenna samples, via a high-speed SerDes interface, between a radio sub-system and a baseband sub-system in cellular infrastructure applications.

In this document, we describe how the AIF peripheral can be used for generic inter-DSP communication. We start by introducing relevant concepts of the OBSAI protocol. Next, we define a software protocol, built on top of OBSAI, for transmission of variable size packets, at a variable (bursty) transfer rate, between two DSP devices connected via the AIF.

This software protocol has been implemented and is available as IACIib (Inter-DSP AIF Communication Library), which can be considered a low-level driver for inter-DSP AIF communication. The driver, as well as a sample application demonstrating the use of the driver is explained in this document, and the corresponding source code is made available as a starting point for further development and customization.

Contents

Figures

1 Introduction

Consider the standard AIF operation. After all required components (AIF, FSYNC, EDMA and interrupts) have been configured at initialization time, the interface is continuously transmitting and receiving data and the timing is strictly controlled by hardware.

The nature of inter-DSP data traffic, however, is often asynchronous and bursty, rather than continuous. This type of traffic is best suited for a packet interface like gigabit Ethernet or sRIO, but in some applications, these interfaces are already used for other purposes.

Given the continuous nature of the typical AIF traffic, the easiest approach to implement bursty traffic would be to have the AIF transmitter continuously send dummy data, and insert useful data when there is actually something to send. The receiver would have to periodically check if there is any data available. Given the data rates involved, the polling at the receiver could represent a very high load on the DSP (both the CPU and the internal buses).

In this document, we introduce a protocol designed to remove the above mentioned polling overhead at the receiver. Rather than having to poll periodically looking for useful data, the receiver is notified by the transmitter, via a special message (called a Start Packet), that a data burst is about to be sent, giving it information about its location (within the OBSAI frame) and size. The receiver then performs necessary setup to be able to extract the data burst from the incoming bit stream.

2 Some elements of the OBSAI protocol

The Antenna Interface (AIF) peripheral provides an interface between a DSP and another device via multiple high-speed SerDes links running at up to 3.072Gbaud/sec. The peripheral was designed for communication between radio equipment and the DSP (which is found on the baseband module). There are two standardized communication protocols that can be used in this application – OBSAI and CPRI.

In this section, we will focus on explaining the relevant aspects of OBSAI which are needed to understand the non-standard DSP-to-DSP protocol which will be described in subsequent sections. Detailed explanation of the operation of the AIF in the OBSAI mode is available in [1].

2.1 Overview

Once the antenna interface is up and running, data samples are continuously transmitted and received. The AIF peripheral mainly provides buffering capabilities so that no data would be lost. These buffers need to be filled and emptied at a certain rate in order to prevent overflow (losing data) and underflow (reading stale data).

The data (typically antenna samples) can consist of multiple streams (or channels, or antenna containers) which are combined on a single physical link in a TDM (time-division multiplexing) manner. In addition to antenna samples, control channels are also supported in the OBSAI interface and are time-division multiplexed with antenna streams in specific reserved locations.

The OBSAI protocol defines important timing information which guarantees that the transmitter and the receiver are synchronized. Various timing events which are needed for the AIF are generated by the FSYNC module (see [2]) based on an external clock.

2.2 Frame structure

OBSAI frame structure is described in detail in Section 3.1 in [1]. Our intent here is to highlight some of the relevant concepts.

The frame structure in OBSAI is based on a 10 msec master frame which consists of nx768000 bytes of payload, where n is the link rate with n=1,2 or 4. The master frame is split into message groups (where the number of message groups per master frame also depends on the physical link rate). Each message group consists of 21 messages, denoted M1, M2, …, M21. Each message consists of 16 bytes of payload and 3 bytes of header. The message is constructed as an RP3 packet and its format is shown in Figure 1.

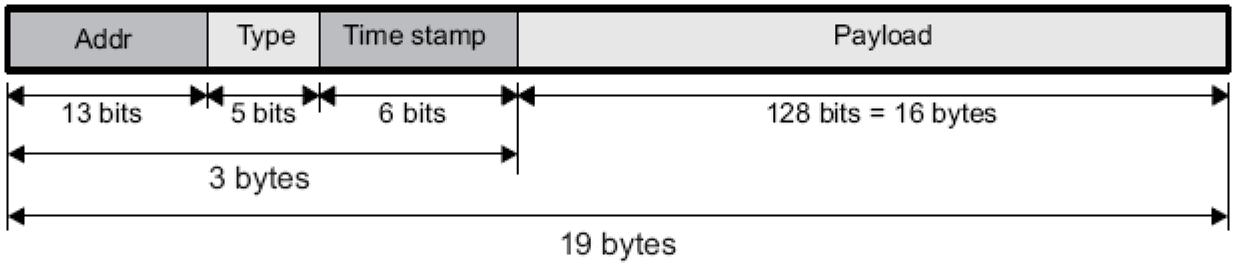


Figure 1 OBSAI RP3 Protocol Packet Structure

2.2.1 Time Stamp

The time stamp is embedded into each OBSAI message, as shown in Figure 1, and it is used to make sure that the transmitter and the receiver are synchronized. This is important for antenna data because each antenna sample needs to be traced back to a particular location in the UMTS frame hierarchy. We will take advantage of this feature in our application in an entirely different way, as will be described in Section 3.

Time stamp is set to 0 at each frame boundary. In UMTS systems, the frame duration is 10msec, and the smallest unit of time is 1 chip period, which is 1/3.84Mhz = 260.4nsec. The time stamp increments once every 4 chip periods, which translates to ~1.04usec. The incrementing is done based on counting external clock pulses.

The time stamp is generated and inserted into the message header by AIF transmit hardware. It is also accessible to software via a read-only register.

The time stamp is verified by AIF receive hardware. If the time stamp value contained in the message received from the SerDes link differs from the local time stamp (which is based on the local FSYNC counter), the message is discarded. If the received and the local time stamp are equal, the AIF hardware places the message in the location in the AIF receive buffer (AIF RX RAM) which corresponds to the time stamp. The AIF buffer organization will be discussed in Section 2.3.2.

2.3 Packet Switched vs Circuit Switched Messages

The AIF can transfer packet-switched (PS) messages typically used for control, or circuit switched (CS) messages typically used for antenna samples.

The OBSAI messages are organized into message groups of 21 times slots. 20 of those time slots are used for data messages, and one is used for a control message.

Packet-switched messages can be sent both through data slots and control slots. This is configured by software via a look-up table. The two types of messages are therefore time-division multiplexed on the SerDes bus, and the above mentioned look up table is used by hardware to decide at which point in time it needs to insert a packet switched message from the packet buffer, vs. a circuit-switched message which comes from the data buffer (AIF TX RAM).

2.3.1 PS FIFOs

There are multiple FIFOs which are used for buffering of packet-switched messages: 3 inbound FIFOs for incoming messages, and 30 outbound FIFOs for outgoing messages. The FIFOs can be programmed to generate an event (CPU or EDMA sync) when a certain number of packets has left/entered the FIFO.

On transmit, the EDMA or CPU places the control message in the transmit FIFO. It is sent out on the next opportunity (i.e. the next slot configured for a PS message).

On receive, message (payload and header) are placed into RX FIFO and after a (programmable) number of messages have been received, an event is generated which can interrupt the CPU or trigger an EDMA transfer.

2.3.2 CS RAM

AIF TX and RX RAM are used for circuit switched messages. The size of the RAM is 2Kbytes per link. Typically, these memories are accessed via EDMA. On transmit, the AIF takes data from the TX RAM, inserts header (including the time stamp) and transmits it over the SerDes. On the receive, the AIF checks the received time stamp and then places the message at the appropriate place in the AIF RX RAM. Therefore, the location of data in the AIF RAMs is closely related to the timestamp at which the data is to be sent or has been received.

From the point of view of AIF and the SerDes link, the AIF RAMs operate like circular buffers. On the receive side, the data is continuously being written into the RX RAM based on the time stamp, and on the transmit side, it is continuously being pulled out of TX RAM based on the time stamp. This is shown in Figure 2. Therefore, the DSP side needs to be able to stay synchronized with the SerDes operation, i.e. the data needs to be written to the TX RAM at the same rate at which it is pulled out for SerDes transmission, and it needs to be pulled out of the RX RAM at the same rate at which it is being written to by SerDes. This is accomplished via synchronized EDMA transfers. For each synchronization event, 1 time stamp worth of data (16 AxC containers, 16 bytes each) is transferred. Therefore, a synchronization event is used which is generated once per TS increment, or once every 4 chips. The role of the CPU is merely to setup the FSYNC and EDMA prior to activating the AIF link, and (optionally) to respond to EDMA transfer completion interrupts.

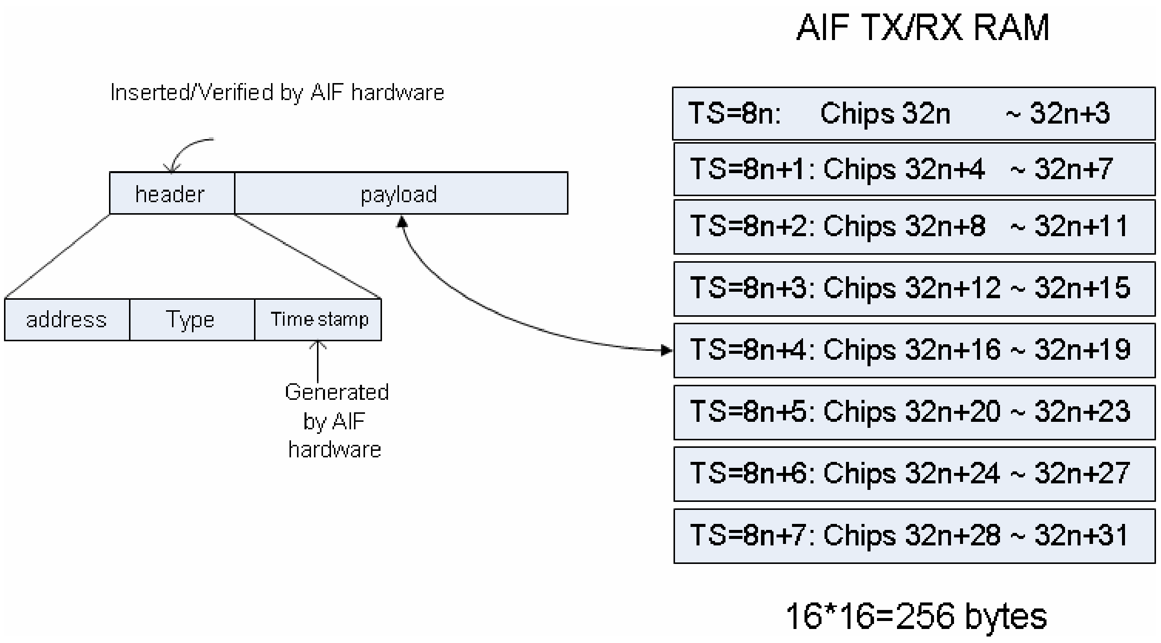

AIF TX/RX RAM
Inserted/Verified by AIF hardware
header
payload
address
Type
Time stamp
Generated by AIF hardware
TS=8n: Chips 32n ~ 32n+3
TS=8n+1: Chips 32n+4 ~ 32n+7
TS=8n+2: Chips 32n+8 ~ 32n+11
TS=8n+3: Chips 32n+12 ~ 32n+15
TS=8n+4: Chips 32n+16 ~ 32n+19
TS=8n+5: Chips 32n+20 ~ 32n+23
TS=8n+6: Chips 32n+24 ~ 32n+27
TS=8n+7: Chips 32n+28 ~ 32n+31
16*16=256 bytes

Figure 2 AIF TX/RX RAM and CS message format

2.4 Synchronization (FSYNC)

The synchronization events used by EDMA to synchronize accesses to the AIF RAMs between the DSP/EDMA side and AIF/SerDes side are generated by the FSYNC module (see [2]).

In simplified terms, the FSYNC generates its events based on counting the FSYNC clock input pulses. The smallest transmission interval in the UMTS systems is one chip period, or 1/3.84MHz. To allow for timing alignment and offset compensation between the transmitter and the receiver, the FSYNC module actually counts sub-chips (1/8th of chip duration).

Due to the relationship between the time stamps and AIF RAM storage described in previous sections, the FSYNC event which is of interest to our application is the 4-chip event (generated once every time stamp increment, or approximately once every 1usec).

3 Inter-DSP AIF Communication (IAC) Protocol

3.1 Concept : Hybrid PS/CS mode

As seen in the previous section, OBSAI PS mode would be the natural candidate for inter-DSP communication, due to the availability of FIFO receive interrupts, but the bandwidth limitation to about 1-2Gbps means that the AIF can only be utilized at a fraction of its capabilities. On the other hand, OBSAI CS mode is exactly the opposite of “bursty” and “asynchronous” communication that we are looking for: it is running continuously, and inserting bursty data into a continuously running stream of dummy data can potentially present high overhead: (1) The EDMA needs to be emptying receive buffers (AIF RX RAM) continuously, and (2) the CPU needs to be checking the contents of received buffers continuously, and potentially discarding them most of the time.

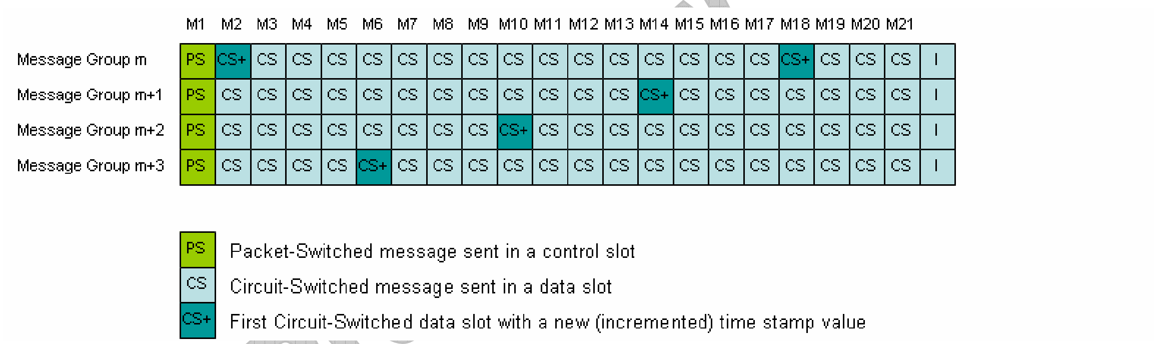
The solution which we describe in this document is to use a “hybrid PS/CS” mode, in which the transmitter first sends a special message which we call the Start Packet (transmitted as a PS message) with the information about the location of the upcoming data burst, followed by the actual data burst (transmitted as a contiguous block of CS messages).

The advantage of sending the Start Packet as a PS message in this context is that, on the receiving side, an event can be generated as soon as the packet arrives in the receive FIFO, which can interrupt the CPU. This is one of the few asynchronous mechanisms provided by the AIF.

3.2 Message Slot Assignments

The location of PS messages within a message group for our application is shown in Figure 3. It is the first message slot in a message group. This slot is configured as a control slot.

Each of these control slots represents an opportunity for sending the Start Packet, and each of the CS slots at which the time stamp increments, denoted “CS+ in Figure 3, represents an opportunity for starting the transmission of the data burst.

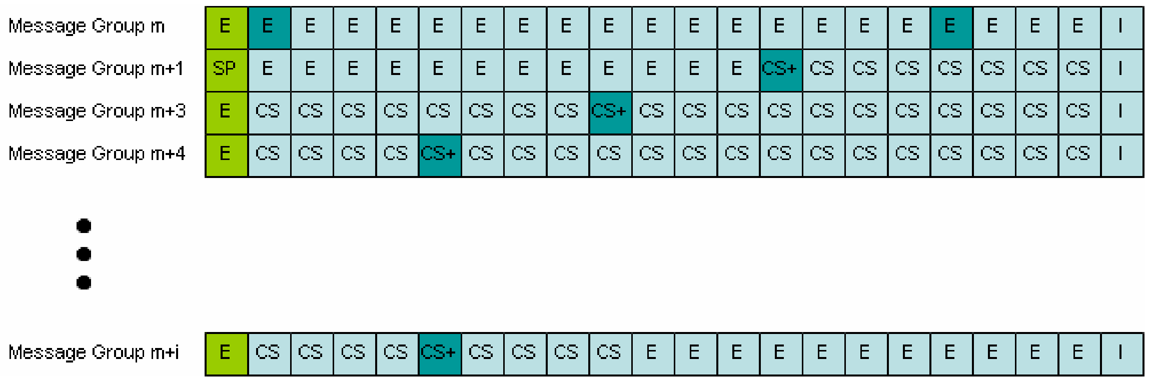


	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	
Message Group m	PS	CS+	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS+	CS	CS	CS	I
Message Group m+1	PS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS+	CS	CS	CS	CS	CS	CS	CS	I
Message Group m+2	PS	CS	CS	CS	CS	CS	CS	CS	CS	CS+	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	I
Message Group m+3	PS	CS	CS	CS	CS	CS+	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	I

PS	Packet-Switched message sent in a control slot
CS	Circuit-Switched message sent in a data slot
CS+	First Circuit-Switched data slot with a new (incremented) time stamp value

Figure 3 Message Slot assignments (4x link example)

While Figure 3 graphically represents the configuration of the AIF lookup tables and the packet types assigned to each slot, Figure 4 shows an example of a data burst transmission. The Start Packet is sent in the first slot of Message Group (m+1), and the data burst transmission starts in slot 14 of Message Group (m+1), which is when the time stamp increments. The data transmission finishes in message group (m+i), where i is a function of the burst length and can be variable.



Message Group m	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	I
Message Group m+1	SP	E	E	E	E	E	E	E	E	E	E	E	E	CS+	CS	CS	CS	CS	CS	CS	CS	I
Message Group m+3	E	CS	CS	CS	CS	CS	CS	CS	CS	CS+	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	I
Message Group m+4	E	CS	CS	CS	CS	CS+	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	I
●																						
●																						
●																						
Message Group m+i	E	CS	CS	CS	CS	CS+	CS	CS	CS	CS	E	E	E	E	E	E	E	E	E	E	E	I

Figure 4 Burst Transmission

In practice, it is also possible that the transmission of the data burst starts before the Start Packet is transmitted, depending on when the next configured control slot falls relative to the time stamp increment slot. This will not prevent the data to be received correctly because the AIF RX RAM stores 8 time stamps worth of information and the EDMA should have enough time to pull the data out of the AIF RX RAM prior to it being overwritten by new data, 8 time stamps later.

3.3 Start Packet Message Format

The Start Packet contains, at a minimum, information about where in the continuous stream the data burst transmission starts, as well as the length of the data burst. This information is used at the receiver to program EDMA transfer parameters to transfer the data burst from AIF RX RAM to DSP memory (internal or external).

Since the Start Packet is simply a message sent in a control slot, its size is 16 bytes. Therefore, additional information can also be sent, such as control data and CRC. A possible Start Packet format is shown in Figure 5. See Section 5 for details of the current implementation and customization suggestions.

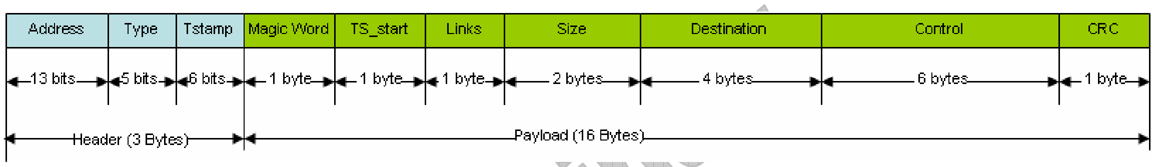


Figure 5 A proposed Start Packet format

The fields in the Start Packet are:

- **Magic Word** (1 byte): Denotes the start of a valid start packet (value: 0x5A)
- **TS_start** (1 byte) and **Links** (1 byte): Specifies the location of the start of the data burst in AIF RAM (see Figure 2).
- **Size** (2 bytes): number of 16-byte blocks. If the size is 0, then this could be just a control message (no data).
- **Destination** (4 bytes): This value can be either a DSP memory address (similar to directIO mode in sRIO), or a mailbox ID (similar to message passing in sRIO).
- **Control**: This field can be used by the application to send ACK/NACK or other control messages with or without sending the data.
- **CRC** (1 byte) – A CRC can be implemented for protection against SerDes transmission errors. A higher-level ACK/NACK protocol can be initiated in case of a failed CRC check.

3.4 Transmitter operation

Once the application makes a request for sending a block of data, the following sequence of events takes place at the transmitter:

- The CPU determines when data transmission needs to start, based on the current value of the time stamp (contained in a counter which increments once every 4 chips). Assume the current time stamp is TS_current. The actual burst transmission is configured to start at the next time stamp, i.e. TS_start = TS_current + 1, to allow time for the CPU to configure the EDMA. The TS_start value is transmitted to the receiver via the Start Packet.

- The CPU constructs the Stack Packet and sets up a non-synchronized, one-shot EDMA transfer of the Start Packet to the TX FIFO. The Start Packet is transmitted at the next opportunity (see Figure 3).

- The transmitter configures the EDMA to copy the data to be transmitted from DSP memory to AIF TX RAM, starting at the location corresponding to timestamp TS_start. The EDMA channel is synchronized to the 4-chip FSYNC event, transferring one time stamp worth of data (256bytes) for each sync event.. The 4-chip event is enabled as soon as the transfer is configured. Because the AIF is taking data from the TX RAM location corresponding to the current time stamp to place it on the SerDes bus, and because 8 time stamps worth of data are stored in the AIF RAM it is important that the EDMA transfer into location corresponding to TS_start completes before the local time stamp reaches TS_start.

3.5 Receiver operation

A priori, the receiver has set up the AIF link to be up and running, and dummy data is continuously being received from the physical link but not consuming any resources outside of AIF. When the start packet arrives in one of the AIF RX FIFOs, the following sequence of events takes place:

- An event is generated by hardware which is used to trigger an EDMA transfer from the RX FIFO to DSP memory. The completion of this EDMA transfer in turns interrupts the CPU.

- The CPU parses the Start Packet to determine the location of the data burst in the AIF RX RAM (based on TS_start and Links), size and the desired destination address. The CPU reads the local time stamp to make sure that it does not exceed TS_start+7, i.e. that it is not too late to receive the data. The CPU then configures the EDMA.

- The CPU waits for the appropriate time to enable the 4-chip EDMA synchronization event. Because the AIF RX RAM is being written to as the data is received over the SerDes bus, it is important that the EDMA does not start reading too soon from the AIF RX RAM location corresponding to TS_start, otherwise old data will be read, and it should not start reading too late, otherwise data will be lost. The EDMA should start no sooner than when local time stamp reaches TS_start+1, and no later than the time when the local time stamp reaches TS_start+7.

3.6 Topologies

The AIF interface consists of 6 independent links. For the purpose of the application described in this document, each of these links is to be treated as a different point-to-point connection and to be initialized and used in its own context.

4 IAC Library APIs

The driver consists of five basic APIs: _Open, _Close, _Read, _Write and _Control. The APIs are explained in this section and the source code is provided in the associated code download.

IACLIB_Handle IACLIB_Open(GLOBAL_StateData *stateDataPIn,Uint32 *InboundPSbufferP,Uint32 *OutboundPSbufferP);
➔ IACLIB module initialization function. This function must be called before any other function in the library. It performs initialization of AIF, FSYNC and EDMA modules.

AIF Link speed is fixed to 4x (3 Gbps data rate). Links can be enabled or disabled individually, as explained in Section 5.1.

GLOBAL_StateData structure must be declared before it is used by IACLIB_Open. It is initialized by IACLIB_Open().InboundPSbuffer and OutboundPS buffer should be allocated by application software prior to calling IACLIB_Open() since its addresses will be used for EDMA setup.

IACLIB_Status IACLIB_Close(IACLIB_Handle hAif);
➔ IACLIB module close function. This function should be called when stopping or restarting the AIF interface. It performs de-initialization of each module.

IACLIB_Status IACLIB_Read(IACLIB_Handle hAif, Uint32 *size, Uint32 *InboundCSbufferP, Uint32 *InboundPSbufferP,Uint32 *linkNum);
➔ IACLIB burst data read function. This function should be called after the Start Packet has been received. Its job is to configure the EDMA for receiving the data burst. After the function returns, the application should wait for the EDMA completion interrupt before it accesses the data burst.

IACLIB_Status IACLIB_Write(IACLIB_Handle hAif, Uint32 size, Uint32 *OutboundCSbufferP, Uint32 *OutboundPSbufferP,Uint32 linkNum);
➔IACLIB burst data write function. This function configures the EDMA for transmitting the Start Packet from DSP memory to TX FIFO, as well as the data burst from DSP memory to AIF TX buffer.

IACLIB_Status IACLIB_Control(IACLIB_Handle hAif, Uint32 cmd);
➔IACLIB hardware control function. It only supports command 0 for now to disable Outbound PS EDMA channel. It could be used for debug.

5 Customizing IACLIB

This section describes the default configuration of AIF, EDMA and FSYNC provided with the default IACLIB in the associated code download, and explains how to customize some of the configuration parameters to suit the specific application needs.

5.1 AIF links

The default IACLIB supports 4 links (Link0 ~ Link3).

Links are chosen via the GLOBAL_StateData.LinkStatus[4] array:

```
GLOBAL_StateData GDataStruct; //declare structure

GDataStruct.LinkStatus[0] = 0; //Disable link0
GDataStruct.LinkStatus[1] = 1; // Enable link1
GDataStruct.LinkStatus[2] = 1; // Enable link2
GDataStruct.LinkStatus[3] = 0; //Disable link3
```

. In this code, Link1 and Link2 are enabled while Link0 and Link3 are disabled.
It is possible to extend the driver to support more than 4 links, with careful considerations of the frequency of AIF-related interrupts and associated CPU loading.

5.2 EDMA channels and PS FIFOs

The default IACLIB uses 11 EDMA channels:

- EDMA channels 16 to 19 (synchronized to FSevent4 ~FSevent7) are used to transfer the data burst from AIF RX RAM to DSP memory (L2/DDR), with one channel per link. This is called Inbound CS EDMA transfer.
- EDMA channels 20 to 23 (synchronized to FSevent8 ~ FSevent11) are used to transfer the data burst from DSP memory (L2/DDR) to AIF TX RAM. This is called Outbound CS EDMA transfer.
- EDMA channels 40 and 42 (synchronized to PS events 1 and 3, respectively) are used for receiving the Start Packet from FIFO0 and FIFO1, respectively. This is called Inbound PS EDMA transfer.
- EDMA channel 12 is used for sending the Start Packet. This is called Outbound PS transfer.

The packet-switched FIFO usage is as follows:
For receiving the Start Packet message

- Link0, Link1 use Inbound FIFO0
- Link2, Link3 use Inbound FIFO1

For transmitting the Start Packet message, each link uses its own Outbound FIFO:

- Link0 uses FIFO0
- Link1 uses FIFO5
- Link2 uses FIFO10
- Link3 uses FIFO15

The user can change the assigned EDMA channels for CS Inbound or Outbound EDMA. If the user wants to change the channel number or assigned FS event, LOCAL_edmaConfig in IAClib_config.c and definitions in IAClib_config.h should be modified.

5.3 FS events

This library uses a total of 16 events to trigger CS EDMA data transfers.

- FS events 4 ~ 7 are mask based trigger events used for CS inbound EDMA transfer.

- FS events 8 ~ 11 are used to trigger EDMA for CS outbound data transfer. FS events 8 and 9 are mask based trigger events and FS events 10 and 11 are counter based trigger events because FS event 10 ~ 17 must use counter based trigger event but trigger duration is same (4chip time).
- FS event 18~21 are used for 4chip done strobe for PE
- FS events 24~27 are used for PE preparation triggers.

Before changing the default FS Event configuration, the user needs to consult the "FSYNC Event connection" table in the TCI6487/8 Datasheet (see [3]) to check if a particular event can be used to trigger EDMA (i.e. if there is a connection from the particular event to the TPCC or CIC3).

Note that FSEVT18 ~ 29 are reserved for 4chip done strobe for PE and PE preparation trigger, and FSEVT0, 1 are reserved for Frame sync trigger. Other FS events can be used for other purposes.

5.4 FSYNC module inputs

The default IACLIB uses RP3 timer instead of System timer. If the user wants to use System timer, every `configMasktrigger` or `configCountertrigger` should be changed from:

```
ConfigMaskTrigger[0].timerUsed = CSL_FSYNC_RP3_TIMER;
```

to:

```
ConfigMaskTrigger[0].timerUsed = CSL_FSYNC_SYSTEM_TIMER;
```

By default, system timer uses TRT and TRT_CLK and RP3 timer uses UMTS_SYNC and Frame_Sync_CLK. This can be modified to use a different clock and sync source as shown in the code snippet below:

```
 myFsyncCfg.syncRP3Timer = CSL_FSYNC_UMTS_SYNC;//CSL_FSYNC_FRAME_BURST;
 myFsyncCfg.syncSystemTimer = CSL_FSYNC_TRT;
 myFsyncCfg.clkRP3Timer = CSL_FSYNC_FRAME_SYNC_CLK;//CSL_FSYNC_UMTS_CLK;
 myFsyncCfg.clkSystemTimer = CSL_FSYNC_TRT_CLK;//CSL_FSYNC_UMTS_CLK;
```

Please refer to table "Frame Sync Module Input Options" in [1] for options for clock/sync inputs.

5.5 Loopback mode vs. non-loopback mode

The driver can be compiled for loopback or for non-loopback mode. The loopback mode is used for testing using a single DSP device. For non-loopback mode, two DSP devices connected via AIF are needed (available, for example, on the TCI6487 EVM).

To choose the desired configuration, select IAClib_aif.pjt as the active project in the CCS project window, select the desired configuration ("Debug" or "Debug_loopback") and rebuild. For each configuration, the resulting library file will be stored in a different location (as specified under Project -> Build Options -> Archiver).

5.6 Start Packet CRC

The Start Packet format shown in Figure 5 features a CRC field as the last byte. The default IACLIB does not use this field but it could be used for verifying the validity of the Start Packet contents.

On the transmitter, the CRC is calculated over 15 bytes and inserted into the CRC field. The receiver recomputes the CRC and compares it with the received CRC. Below is an example CRC routine for 8bit CRC. This code can be added to IACLIB_Read() and IACLIB_Write() APIs.

```
Int8 generate_8bit_crc(char* data, int16 length, int8 pattern)
{
   int8  *current_data;
   int8   crc_byte;
   int16 byte_counter;
   int8   bit_counter;

   current_data = data;
   crc_byte = *current_data++;

   for(byte_counter=0; byte_counter < (length-1); byte_counter++)
   {
      for(bit_counter=0; bit_counter < 8; bit_counter++)
      {
         if(!bit_test(crc_byte,7))
         {
            crc_byte <<= 1;
            bit_test(*current_data, 7 - bit_counter) ?
               bit_set(crc_byte,0) : bit_clear(crc_byte,0);
            continue;
         }
         crc_byte <<= 1;
         bit_test(*current_data, 7 - bit_counter) ?
            bit_set(crc_byte,0) : bit_clear(crc_byte,0);
         crc_byte ^= pattern;
      }
      current_data++;
   }
   for(bit_counter=0; bit_counter < 8; bit_counter++)
   {
      if(!bit_test(crc_byte,7))
      {
         crc_byte <<= 1;
         continue;
      }
      crc_byte <<= 1;
      crc_byte ^= pattern;
   }
   return crc_byte;
}
```

6 Application Example

A demo application is provided along with the IAC library to demonstrate its usage. The application is developed for and tested on the TCI6487/8 EVM. The associated CCS project consists of two configurations

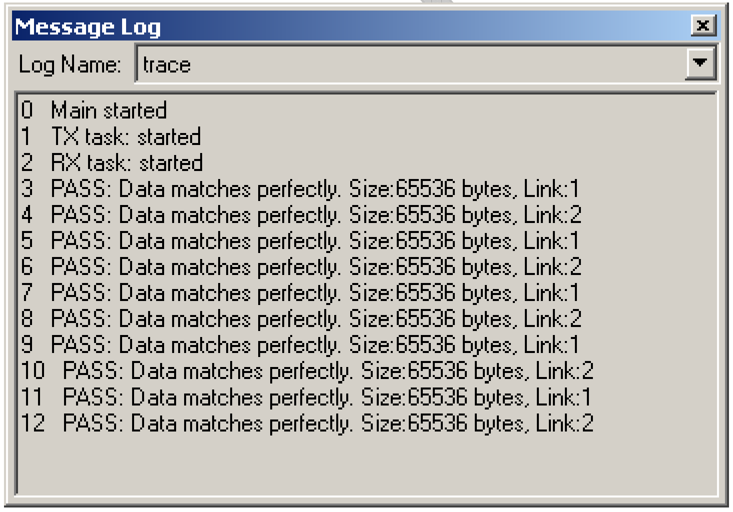
1. **Loopback** – Includes IAClib_aif_loopback.lib. Runs on core 0 of a single Faraday device (Faraday 1). The receiver and the transmitter reside on the same device and the loopback is performed at the SerDes level.

2. **Non-Loopback** – Includes IAClib.lib. Runs on both Faraday devices. Faraday 2 needs to run first. Faraday 1 runs second and provides the initial pulse which starts the FSYNC_CLK. This initiates full-duplex communication between the two devices.

6.1 How to run the example on Faraday EVM

The procedure for running the application is as follows:

1. Configure CCS Setup for the Faraday EVM (via USB), save and run CCS
2. **Loopback**: Open C64PLUS_F1A. **Non-Loopback**: also open C64PLUS_F2A.
3. Connect all open cores. Open project interDSPAIF.pjt and choose the desired configuration (Loopback or Non_Loopback).
4. Build and Load Program on all open cores.
5. Open DSP/BIOS -> Message Log.
6. Debug-> Go Main on all cores (this step seems necessary for the Message Log display to work)
7. **Non-Loopback** only: Run C64PLUS_F2A
8. Run C64PLUS_F1A.
9. The message log display should look as follows (after about a minute):



```
0  Main started
1  TX task: started
2  RX task: started
3  PASS: Data matches perfectly. Size:65536 bytes, Link:1
4  PASS: Data matches perfectly. Size:65536 bytes, Link:2
5  PASS: Data matches perfectly. Size:65536 bytes, Link:1
6  PASS: Data matches perfectly. Size:65536 bytes, Link:2
7  PASS: Data matches perfectly. Size:65536 bytes, Link:1
8  PASS: Data matches perfectly. Size:65536 bytes, Link:2
9  PASS: Data matches perfectly. Size:65536 bytes, Link:1
10  PASS: Data matches perfectly. Size:65536 bytes, Link:2
11  PASS: Data matches perfectly. Size:65536 bytes, Link:1
12  PASS: Data matches perfectly. Size:65536 bytes, Link:2
```

Note: When restarting the program, it is necessary to Disconnect, hit POR button (SW2), reconnect and reload the program.

6.2 Example Overview

The application consists of two statically created tasks, TSK_RX and TSK_TX with associated functions _taskRX and _taskTX, respectively.

TSK_TX is the transmit task. It pends on SEM_TX. When the semaphore is posted, the task initializes the data packet to be transferred and calls IACLIB_Write().

TSK_RX is the receive task. Initially, it pends on SEM_RX_PS. When the semaphore is posted (meaning that the Start Packet has been received), it calls IACLIB_Read(), which configures the AIF to receive the data burst, and pends on SEM_RX_CS. Once the semaphore is posted (meaning the EDMA has completed transferring the data from AIF RX RAM to DSP memory), it verifies its contents and reports the result via a LOG_printf().

The following interrupts are used:

1. EDMA ISR (HWI_4): This ISR runs upon completion of one of the following EDMA transfers:
    a. A PS message transfer from the AIF FIFO to DSP memory. In this case, the SEM_RX_PS semaphore is posted.
    b. Data packet transfer from AIF CS RAM to DSP memory. In this case, SEM_RX_CS is posted.
2. FS_EVT1 ISR (HWI_INT5): This ISR runs once every 10msec. In our application, it is used to initiate the transfer of a data packet, i.e. it posts SEM_TX.

The default size of the data packet is 64Kbytes, defined via TOTAL_DATA_SIZE.

7 References

1. *TMS320TCI6487/8 Antenna Interface User's Guide* (SPRUEF4)
2. *TMS320TCI6487/8 Frame Synchronization User's Guide* (SPRUEF5)
3. *TMS320TCI6487/8 datasheet (SPRS358)*