ANALOG DEVICES

ADI_AD1980
DEVICE DRIVER

DATE: NOVEMBER 14, 2007.

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
Nov 13 ,2007	Initial release
Nov 14, 2007	Added example code to update AD1980 ADC/DAC sampling rate

Table 1 – Revision History

1. Overview

AD1980 is an AC ’97 version 2.3 Compatible Audio CODEC with an Integrated Stereo Headphone Amplifier. The CODEC supports up to 6 DAC Channels for 5.1 Surround audio, S/PDIF Output, Variable Rate Audio mode (for 7kHz to 48kHz ADC/DAC sampling rate) and Double Rate Audio mode (for 96 kHz and 88.2kHz DAC sampling rate). The CODEC is ideal for embedded low-cost audio applications such as automotive voice and stereo applications (MP3 playback, speech recognition etc.), automotive head-units, portable device interface systems and hands-free car kits.

This document describes the functionality of AD1980 driver that adheres to Analog Devices Device Driver and System Services Model. AD1980 driver is built on top of SPORT and AC’97 Digital Controller drivers.

2. Files

The files listed below comprise the device driver API and source files.

2.1. Include Files

The driver sources include the following include files:

- **<services/services.h>**
  This file contains all definitions, function prototypes etc. for all the System Services.

- **<drivers/adi_dev.h>**
  This file contains all definitions, function prototypes etc. for the Device Manager and general device driver information.

- **<drivers/sport/adi_sport.h>**
  This file contains all definitions, function prototypes etc. specific to SPORT device.

- **<drivers/codec/adi_ac97.h>**
  This file contains all definitions, function prototypes etc. specific to AC ‘97 Digital Controller driver.

- **<drivers/codec/adi_ad1980.h>**
  This file contains all definitions, function prototypes etc. specific to AD1980 device.

2.2. Source Files

The driver sources are contained in the following files, as located in the default installation directory:

- **<Blackfin/lib/src/drivers/codec/adi_ac97.c>**
  This file contains all the source code for AC ‘97 Digital Controller driver. All source code is written in ‘C’. There are no assembly level functions in this driver.

- **<Blackfin/lib/src/drivers/codec/adi_ad1980.c>**
  This file contains all the source code for the AD1980 Device Driver. All source code is written in ‘C’. There are no assembly level functions in this driver.

3. Lower Level Drivers

AD1980 driver is layered on SPORT and AC'97 Digital Controller drivers.

3.1. AC'97 Device Driver

AC '97 Digital Controller driver provides a simple and effective way to interface ADI Blackfin embedded processors with ADI AC'97 compliant Audio CODECS. Please refer to AC'97 Digital Controller driver documentation (adi_ac97.pdf) for more information.

3.2. SPORT Device Driver

Serial Port (SPORT) is used to handle audio dataflow between Blackfin and the codec. Please refer to SPORT driver documentation (adi_sport.pdf) for more information.

4. Resources Required

Device drivers typically consume some amount of system resources. This section describes the resources required by the device driver.

Unless explicitly noted in the sections below, this device driver uses the System Services to access and control any required hardware. The information in this section may be helpful in determining the resources this driver requires, such as the number of interrupt handlers or number of DMA channels etc., from the System Services.

Because dynamic memory allocations are not used in the Device Drivers or System Services, all memory used by the Device Drivers and System Services must be supplied by the application. The Device Drivers and System Services supply macros that can be used by the application to size the amount of base memory and/or the amount of incremental memory required to support the needed functionality. Memory for the Device Manager and System Services is provided in the initialization functions (adi_xxx_Init()).

Wherever possible, this device driver uses the System Services to perform the necessary low-level hardware access and control.

The AD1980 device driver is build upon SPORT and AC'97 Digital Controller drivers.

4.1. Interrupts

AD1980 driver requires **five** secondary interrupt memory of size **`ADI_INT_SECONDARY_MEMORY.`**

- **4** x **`ADI_INT_SECONDARY_MEMORY`** to handle SPORT Transmit and Receive DMA interrupts (two DMA Data interrupt handlers and two DMA error interrupt handlers).
- **1** x **`ADI_INT_SECONDARY_MEMORY`** to handle SPORT peripheral error interrupt

4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven SPORT for its audio dataflow. AD1980 supports inbound, outbound and bi-directional dataflow, but the SPORT device allocated to AD1980 is always configured in bi-directional dataflow to maintain AC'97 link with the CODEC hardware.

AD1980 driver requires memory to handle **two** DMA channels (SPORT Transmit and Receive) of size **`ADI_DMA_CHANNEL_MEMORY`** for each.

4.3. Timers

Timer service is not used by this driver.

4.4. Real-Time Clock

RTC service is not used by this driver

4.5. Programmable Flags

Flag service is used to issue a hardware reset signal over a Blackfin GPIO flag connected to AD1980 reset pin.

4.6. Pins

Blackfin SPORT device port pins connected to AC'97 interface port of AD1980.

5. Supported Features of the Device Driver

This section describes what features are supported by the device driver.

5.1. Directionality

The driver supports the dataflow directions listed in the table below.

ADI_DEV_DIRECTION	Description
ADI_DEV_DIRECTION_INBOUND	Supports the reception of data in through the device.
ADI_DEV_DIRECTION_OUTBOUND	Supports the transmission of data out through the device.
ADI_DEV_DIRECTION_BIDIRECTIONAL	Supports both the reception of data and transmission of data through the device.

Table 2 – Supported Dataflow Directions

5.2. Dataflow Methods

The driver supports the dataflow methods listed in the table below.

ADI_DEV_MODE	Description
ADI_DEV_MODE_CHAINED	Supports the chained buffer method
ADI_DEV_MODE_CHAINED_LOOPBACK	Supports the chained buffer with loopback method

Table 3 – Supported Dataflow Methods

5.3. Buffer Types

The driver supports the buffer types listed in the table below.

- **ADI_DEV_1D_BUFFER**
  - Linear one-dimensional buffer
  - pAdditionalInfo – ignored

5.4. Command IDs

This section enumerates the commands that are supported by the driver. The commands are divided into three sections. The first section describes commands that are supported directly by the Device Manager. The next section describes common commands that the driver supports. The remaining section describes driver specific commands.

Commands are sent to the device driver via the adi_dev_Control() function. The adi_dev_Control() function accepts three arguments:

- DeviceHandle – This parameter is a **ADI_DEV_DEVICE_HANDLE** type that uniquely identifies the device driver. This handle is provided to the client in the adi_dev_Open() function call.
- CommandID – This parameter is a u32 data type that specifies the command ID.
- Value – This parameter is a void * whose value is context sensitive to the specific command ID.

The sections below enumerate the command IDs that are supported by the driver and the meaning of the Value parameter for each command ID.

5.4.1. Device Manager Commands

The commands listed below are supported and processed directly by the Device Manager. As such, all device drivers support these commands.

- **ADI_DEV_CMD_TABLE**
  - Table of command pairs being passed to the driver
  - Value – ADI_DEV_CMD_VALUE_PAIR *
- **ADI_DEV_CMD_END**
  - Signifies the end of a command pair table
  - Value – ignored
- **ADI_DEV_CMD_PAIR**
  - Single command pair being passed
  - Value – ADI_DEV_CMD_PAIR *
- **ADI_DEV_CMD_SET_SYNCHRONOUS**
  - Enables/disables synchronous mode for the driver
  - Value – TRUE/FALSE

5.4.2. Common Commands

The command IDs described in this section are common to many device drivers. The list below enumerates all common command IDs that are supported by this device driver.

- **ADI_DEV_GET_PERIPHERAL_DMA_SUPPORT**
  - Determines if the device driver is supported by peripheral DMA
  - Value – u32 * (location where TRUE or FALSE is stored)
- **ADI_DEV_CMD_REGISTER_READ**
  - Reads a single device register
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- **ADI_DEV_CMD_REGISTER_FIELD_READ**
  - Reads a specific field location in a single device register
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- **ADI_DEV_CMD_REGISTER_TABLE_READ**
  - Reads a table of selective device registers
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- **ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ**
  - Reads a table of selective device register fields
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- **ADI_DEV_CMD_REGISTER_WRITE**
  - Writes to a single device register
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics)
- **ADI_DEV_CMD_REGISTER_FIELD_WRITE**
  - Writes to a specific field location in a single device register
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- **ADI_DEV_CMD_REGISTER_TABLE_WRITE**
  - Writes to a table of selective device registers
  - Value – **ADI_DEV_ACCESS_REGISTER *** (register specifics)
- **ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE**
  - Writes to a table of selective device register fields
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics)
- **ADI_DEV_CMD_SET_DATAFLOW_METHOD**
  - Specifies the dataflow method the device is to use. The list of dataflow types supported by the device driver is specified in section 5.2.
  - Value – ADI_DEV_MODE enumeration

- **ADI_DEV_CMD_SET_DATAFLOW**
  - Enables/disables dataflow through the device
  - Value – TRUE/FALSE

5.4.3. Device Driver Specific Commands

The command IDs listed below are supported and processed by the device driver. These command IDs are unique to this device driver.

AD1980 Driver specific commands

- **ADI_AD1980_CMD_INIT_DRIVER**
  - Initialises AD1980 driver
  - Value = **ADI_AD1980_INIT_DRIVER *** (address of **ADI_AD1980_INIT_DRIVER** type structure)
  - **Note: This Command MUST be issued immediately after opening AD1980 driver.** Please refer to section 6.4 for more information

```
/* Data structure to hold AD1980 driver initialisation information */
typedef struct ADI_AD1980_INIT_DRIVER
{
        /* Pointer to AD1980 AC'97 Data Frame memory allocated by the application   */
        void                                    *pDataFrame;
        /* Size of AC'97 Data Frame memory allocated by the application (in bytes) */
        u32                                     DataFrameSize;
        /* Pointer to AC'97 Driver Instance memory allocated by the application */
        ADI_AC97_DRIVER_INSTANCE        *pAC97;
        /* Flag ID connected to the reset pin of AD1980 Audio Codec */
        ADI_FLAG_ID                             ResetFlagId;
        /* SPORT Device Number allocated for AD1980 Audio Codec */
        u32                                     SportDevNumber;
} ADI_AD1980_INIT_DRIVER;
```

- **ADI_AD1980_CMD_USE_DCB_TO_PROCESS_FRAMES**
  - Sets AD1980 driver to use Deferred Callback instead of live callback to process AC'97 frames.
  - Value = Handle to Deferred Callback Service (ADI_DCB_HANDLE)
  - Default = Live callback, means AC'97 Frames are processed at SPORT Rx DMA IVG level
  - Note: To switch from Deferred to live callback, pass command argument as NULL (Value = NULL)

AC'97 Digital Controller Driver specific commands

- **ADI_AC97_CMD_ENABLE_MULTICHANNEL_AUDIO**
  - Enable/Disable Multi-channel Audio.
  - Value = TRUE/FALSE, TRUE for Multi-channel Audio, FASLE for Stereo
  - Default = FALSE (Stereo mode)
  - Note: All DAC buffers supplied by the application MUST have audio data in interleaved format. Please refer to Table 7 (page 20) for audio buffer data format supported by AC'97 driver.

- **ADI_AC97_CMD_ENABLE_TRUE_MULTICHANNEL_AUDIO**
  - Enable/Disable true Multi-channel audio mode.
  - Value = TRUE/FALSE, TRUE for Enable True Multi-channel audio, FALSE for Pseudo Multi-channel audio
  - Default = FALSE (Pseudo Multi-channel audio)
  - Note: This command enables Multi-channel audio regardless of the command value parameter being TRUE or FALSE.
    - TRUE Multi-channel mode means DAC buffers submitted by the application hold samples for all six DAC channels in interleaved format mentioned in Table 7.
    - Pseudo Multi-channel mode means DAC buffers submitted by the application hold samples only for Line-out DAC channels in interleaved format mentioned in Table 7. AC'97 driver mimics Multi-channel audio by copying Line-out Left channel data to Center & Surround Left channels and Line-out Right channel data to LFE & Surround Right channels.
    - Multi-channel audio mode can be disabled ONLY by issuing **ADI_AC97_CMD_ENABLE_MULTICHANNEL_AUDIO** command with FALSE as command argument (Value = FALSE).

- **ADI_AC97_CMD_CLEAR_DAC_APP_BUFFER_PTRS**
  - AC'97 driver maintains separate queue to manage ADC/DAC buffers supplied by the application. This command can be used to flush the DAC buffer queue maintained by driver
  - Value = NULL

- **ADI_AC97_CMD_CLEAR_ADC_APP_BUFFER_PTRS**
  - AC'97 driver maintains separate queue to manage ADC/DAC buffers supplied by the application. This command can be used to flush the ADC buffer queue maintained by driver
  - Value = NULL

AD1980 register access commands

- Use following Device Access Commands (defined in adi_dev.h) to access AD1980 hardware registers. Refer section 5.4.2 for command arguments and section 8.2 for examples

Command ID	Comments
ADI_DEV_CMD_REGISTER_READ	Reads a single AD1980 register
ADI_DEV_CMD_REGISTER_FIELD_READ	Reads a specific AD1980 register field
ADI_DEV_CMD_REGISTER_TABLE_READ	Reads a table of AD1980 registers
ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ	Reads a table of AD1980 register(s) field(s)
ADI_DEV_CMD_REGISTER_WRITE	Writes to a single AD1980 register
ADI_DEV_CMD_REGISTER_FIELD_WRITE	Writes to a AD1980 register field
ADI_DEV_CMD_REGISTER_TABLE_WRITE	Writes to a table of AD1980 registers
ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE	Writes to a table of AD1980 register(s) field(s)

Table 4 – Device Access Commands supported by AD1980 driver

Note: AC'97 driver can accept or process only one register access table at any given time. The driver posts ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) after processing a register access table. Application MUST wait for this event after issuing a register read access command or before issuing a new register access request.

5.5. Callback Events

This section enumerates the callback events the device driver is capable of generating. The events are divided into two sections. The first section describes events that are common to many device drivers. The next section describes driver specific event IDs. The client should prepare its callback function to process each event described in these two sections.

The callback function is of the type **ADI_DCB_CALLBACK_FN**. The callback function is passed three parameters. These parameters are:

- ClientHandle – This void * parameter is the value that is passed to the device driver as a parameter in the adi_dev_Open() function.
- EventID – This is a u32 data type that specifies the event ID.
- Value – This parameter is a void * whose value is context sensitive to the specific event ID.

The sections below enumerate the event IDs that the device driver can generate and the meaning of the Value parameter for each event ID.

5.5.1. Common Events

The events described in this section are common to many device drivers. The list below enumerates all common event IDs that are supported by this device driver.

- **ADI_DEV_EVENT_BUFFER_PROCESSED**
  - Notifies callback function that a single or chained buffer has been processed by the device driver.
  - Value – For chained dataflow methods, this value is the CallbackParameter value that was supplied in the buffer that was passed to the adi_dev_Read() or adi_dev_Write()function.
- **ADI_DEV_EVENT_DMA_ERROR_INTERRUPT**
  - Notifies the callback function that a DMA error occurred.
  - Value – Null.

5.5.2. Device Driver Specific Events

The events listed below are supported and processed by the device driver. These event IDs are unique to this device driver.

- **ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE** (Hex Value = 0x402C0001)
  - Notifies callback function that AC'97 / AD1980 driver has completed processing a register access table
  - Value – Address of register access table that's been processed.

5.6. Return Codes

All API functions of the device driver return status indicating either successful completion of the function or an indication that an error has occurred. This section enumerates the return codes that the device driver is capable of returning to the client. A return value of **ADI_DEV_RESULT_SUCCESS** indicates success, while any other value indicates an error or some other informative result. The value **ADI_DEV_RESULT_SUCCESS** is always equal to the value zero. All other return codes are a non-zero value.

The return codes are divided into two sections. The first section describes return codes that are common to many device drivers. The next section describes driver specific return codes. The client should prepare to process each of the return codes described in these sections.

Typically, the application should check the return code for **ADI_DEV_RESULT_SUCCESS**, taking appropriate corrective action if **ADI_DEV_RESULT_SUCCESS** is not returned. For example:

```
if (adi_dev_Xxxx(…) == ADI_DEV_RESULT_SUCCESS) {
      // normal processing
} else {
      // error processing
}
```

5.6.1. Common Return Codes

The return codes described in this section are common to many device drivers. The list below enumerates all common return codes that are supported by this device driver.

- **ADI_DEV_RESULT_SUCCESS**
  - The function executed successfully.
- **ADI_DEV_RESULT_NOT_SUPPORTED**
  - The function is not supported by the driver.
- **ADI_DEV_RESULT_DEVICE_IN_USE**
  - The requested device is already in use.
- **ADI_DEV_RESULT_NO_MEMORY**
  - There is insufficient memory available.
- **ADI_DEV_RESULT_BAD_DEVICE_NUMBER**
  - The device number is invalid.
- **ADI_DEV_RESULT_DIRECTION_NOT_SUPPORTED**
  - The device cannot be opened in the direction specified.
- **ADI_DEV_RESULT_BAD_DEVICE_HANDLE**
  - The handle to the device driver is invalid.
- **ADI_DEV_RESULT_BAD_MANAGER_HANDLE**
  - The handle to the Device Manager is invalid.
- **ADI_DEV_RESULT_BAD_PDD_HANDLE**
  - The handle to the physical driver is invalid.
- **ADI_DEV_RESULT_INVALID_SEQUENCE**
  - The action requested is not within a valid sequence.
- **ADI_DEV_RESULT_ATTEMPTED_READ_ON_OUTBOUND_DEVICE**
  - The client attempted to provide an inbound buffer for a device opened for outbound traffic only.
- **ADI_DEV_RESULT_ATTEMPTED_WRITE_ON_INBOUND_DEVICE**
  - The client attempted to provide an outbound buffer for a device opened for inbound traffic only.
- **ADI_DEV_RESULT_DATAFLOW_UNDEFINED**
  - The dataflow method has not yet been declared.
- **ADI_DEV_RESULT_DATAFLOW_INCOMPATIBLE**
  - The dataflow method is incompatible with the action requested.

- **ADI_DEV_RESULT_BUFFER_TYPE_INCOMPATIBLE**
  - The device does not support the buffer type provided.
- **ADI_DEV_RESULT_CANT_HOOK_INTERRUPT**
  - The Interrupt Manager failed to hook an interrupt handler.
- **ADI_DEV_RESULT_CANT_UNHOOK_INTERRUPT**
  - The Interrupt Manager failed to unhook an interrupt handler.
- **ADI_DEV_RESULT_NON_TERMINATED_LIST**
  - The chain of buffers provided is not NULL terminated.
- **ADI_DEV_RESULT_NO_CALLBACK_FUNCTION_SUPPLIED**
  - No callback function was supplied when it was required.
- **ADI_DEV_RESULT_REQUIRES_UNIDIRECTIONAL_DEVICE**
  - Requires the device be opened for either inbound or outbound traffic only.
- **ADI_DEV_RESULT_REQUIRES_BIDIRECTIONAL_DEVICE**
  - Requires the device be opened for bidirectional traffic only.

Return codes specific to TWI/SPI Device access service

- **ADI_DEV_RESULT_CMD_NOT_SUPPORTED**
  - Command not supported by the Device Access Service
- **ADI_DEV_RESULT_INVALID_REG_ADDRESS**
  - The client attempting to access an invalid register address
- **ADI_DEV_RESULT_INVALID_REG_FIELD**
  - The client attempting to access an invalid register field location
- **ADI_DEV_RESULT_INVALID_REG_FIELD_DATA**
  - The client attempting to write an invalid data to selected register field location
- **ADI_DEV_RESULT_ATTEMPT_TO_WRITE_READONLY_REG**
  - The client attempting to write to a read-only location
- **ADI_DEV_RESULT_ATTEMPT_TO_ACCESS_RESERVE_AREA**
  - The client attempting to access a reserved location
- **ADI_DEV_RESULT_ACCESS_TYPE_NOT_SUPPORTED**
  - Device Access Service does not support the access type provided by the driver

5.6.2. Device Driver Specific Return Codes

The return codes listed below are supported and processed by the device driver. These event IDs are unique to this device driver.

AD1980 Driver specific Return codes

- **ADI_AD1980_RESULT_CMD_NOT_SUPPORTED** (Hex Value = 0x402D0001)
  - Command supplied by the client is not supported by AD1980 device driver
- **ADI_AD1980_RESULT_AC97_INSTANCE_INVALID** (Hex Value = 0x402D0002)
  - AC'97 instance allocated by the application to this driver is invalid.
- **ADI_AD1980_RESULT_REG_ACCESS_ALREADY_IN_USE** (Hex Value = 0x402D0003)
  - Results when client issues a register access request when AD1980 - AC'97 driver is in the middle of processing a register access table.

AC'97 Digital Controller Driver specific Return codes

- **ADI_AC97_RESULT_CMD_NOT_SUPPORTED** (Hex Value = 0x402C0001)
  - Command supplied by the client is not supported by AC'97 Digital Controller driver.
- **ADI_AC97_RESULT_INSUFFICIENT_DATA_FRAME_MEMORY** (Hex Value = 0x402C0002)
  - Results when AC'97 data frame memory provided by the client is not sufficient.

- **ADI_AC97_RESULT_DRIVER_INSTANCE_INVALID** (Hex Value = 0x402C0003)
  - Results when AC'97 driver instance provided by the client is invalid.
- **ADI_AC97_RESULT_DRIVER_ONLY_SUPPORT_1D_BUFFERS** (Hex Value = 0x402C0004)
  - AC'97 Digital Controller driver only supports 1D type buffers. The driver returns above error code when client submits buffer(s) other than 1D.
- **ADI_AC97_RESULT_APP_DATA_POINTS_TO_NULL** (Hex Value = 0x402C0005)
  - Results when ADC/DAC buffer (1D) submitted by the client has data field (pointer) as NULL.
- **ADI_AC97_RESULT_CODEC_NOT_READY** (Hex Value = 0x402C0006)
  - Results when the corresponding AC'97 Audio CODEC is not ready for audio streaming.
- **ADI_AC97_RESULT_ADC_SAMPLE_RATE_NOT_SUPPORTED** (Hex Value = 0x402C0007)
  - Results when ADC sampling rate issued by the client is not supported by AC'97 driver.
- **ADI_AC97_RESULT_DAC_SAMPLE_RATE_NOT_SUPPORTED** (Hex Value = 0x402C0008)
  - Results when DAC sampling rate issued by the client is not supported by AC'97 driver.

6. Configuring the Device Driver

This section describes the default configuration settings for the device driver and any additional configuration settings required from the client application.

6.1. Entry Point

When opening the device driver with the adi_dev_Open() function call, the client passes a parameter to the function that identifies the specific device driver that is being opened. This parameter is called the entry point. The entry point for this driver is listed below.

- ADIAD1980EntryPoint

6.2. Default Settings

The table below describes the default configuration settings for the device driver. If the default values are inappropriate for the given system, the application should use the command IDs listed in the table to configure the device driver appropriately.

Item	Default Value	Possible Values	Command ID
Deferred / Live callback to process AC'97 frames	NULL	ADI_DCB_HANDLE/ NULL	ADI_AD1980_CMD_USE_DCB_TO_PROCESS_FRAMES
Mult-Channel Audio mode / Stereo mode	FALSE	TRUE/FALSE	ADI_AC97_CMD_ENABLE_MULTICHANNEL_AUDIO
TRUE Multi-channel / Pseudo Mult-channel	FALSE	TRUE/FALSE	ADI_AC97_CMD_ENABLE_TRUE_MULTICHANNEL_AUDIO

Table 5 – Default Settings

6.3. Additional Required Configuration Settings

In addition to the possible overrides of the default driver settings, the device driver requires the application to specify the additional configuration information listed in the table below.

Item	Possible Values	Command ID
Initialise AD1980 Driver	Depends on AD1980 reset flag ID and SPORT device number	ADI_AD1980_CMD_INIT_DRIVER

Table 6 – Additional Required Settings

6.4. Initializing AD1980 device instance

This section explains how to initialize AD1980 driver using **ADI_AD1980_CMD_INIT_DRIVER** command with pointer to structure of type **ADI_AD1980_INIT_DRIVER**.

- AC'97 Digital Controller driver requires a minimum of **ADI_AC97_DATA_FRAME_BASE_MEMORY** sized buffer (termed as AC'97 Data Frame Buffer) to handle AC'97 packets.

- Data transfer between AC'97 CODEC and Digital Controller, termed as AC-Link, is fixed at 48kHz sample sync which generates a SPORT interrupt approximately every 22.5us. This interrupt/callback overload can be adjusted by providing a bigger AC'97 Data Frame Buffer. For example, providing AC'97 Data Frame Buffer which is 44 times of **ADI_AC97_DATA_FRAME_BASE_MEMORY** generates a SPORT interrupt every 1ms.

- Each AC'97 compatible Audio CODEC device requires memory for one AC'97 Digital Controller Driver instance (of size **ADI_AC97_DRIVER_INSTANCE**) to maintain AC-Link between Audio CODEC and the Digital Controller (Blackfin).

```
/********** Example code to initialize AD1980 driver **********/

/* AC'97 Audio frame size to set SPORT interrupt at 1ms interval */
#define     AC97_AUDIO_FRAME_SIZE       (ADI_AC97_DATA_FRAME_BASE_MEMORY * 44)
/* Flag ID connected to AD1980 reset pin on ADSP-BF548 Ez-Kit Lite rev 1.3 (PB3) */
#define     AD1980_RESET_FLAG           ADI_FLAG_PB3
/* SPORT device connected to AD1980 on ADSP-BF548 Ez-Kit Lite rev 1.3 */
#define     AD1980_SPORT_NUMBER         0

/* Memory for AD1980 AC'97 audio data frame */
u8          AC97_AudioFrame[AC97_AUDIO_FRAME_SIZE];
/* memory for one AC'97 driver instance */
section ("L1_data") ADI_AC97_DRIVER_INSTANCE    AC97_Instance;

/* Structure to init AD1980 driver */
ADI_AD1980_INIT_DRIVER          InitAD1980;

/* Populate AD1980 driver Initialization structure */
InitAD1980.pDataFrame           = (void *) &AC97_AudioFrame[0];
InitAD1980.DataFrameSize        = AC97_AUDIO_FRAME_SIZE;
InitAD1980.pAC97                = &AC97_Instance;
InitAD1980.ResetFlagId          = AD1980_RESET_FLAG;
InitAD1980.SportDevNumber       = AD1980_SPORT_NUMBER;

/*… Initialize system services, hook exception/hardware interrupt, application code, etc …*/

/* Open AD1980 - Device 0*/
Result = adi_dev_Open(  adi_dev_ManagerHandle,          /* Device manager Handle                    */
                        &ADIAD1980EntryPoint,           /* Entry point for AD1980 driver            */
                        0,                              /* AD1980 Device number to use              */
                        NULL,                           /* No client handle                         */
                        &AD1980DriverHandle,            /* Location to store AD1980 driver handle   */
                        ADI_DEV_DIRECTION_BIDIRECTIONAL,/* Data Direction                           */
                        adi_dma_ManagerHandle,          /* Handle to DMA Manager                    */
                        DCBManagerHandle,               /* Handle to callback manager               */
                        AD1980Callback);                /* Callback Function                        */

/* Initialize above AD1980 driver instance */
Result = adi_dev_Control (AD1980DriverHandle, ADI_AD1980_CMD_INIT_DRIVER, (void *)&InitAD1980);

/*… Application code …*/
```

6.5. ADC/DAC Buffer – Audio Data format

AC'97 Digital Controller driver handles AD1980 ADC/DAC buffers and the driver only supports one dimensional type buffers (`ADI_DEV_1D_BUFFER`). AC'97 driver stores received ADC data in interleaved format. Similarly, all DAC buffers supplied by the application MUST have audio data in interleaved format. Following table shows the audio data format supported by AC'97 Digital Controller diver.

Audio Mode	ADC Buffer data format	DAC Buffer data format
Stereo Audio	Line-In Left, Line-In Right, Line-In Left, Line-In Right, ….	Line-Out Left, Line-Out Right, Line-Out Left, Line-Out Right, ….
Pseudo Multi-Channel Audio	Line-In Left, Line-In Right, Line-In Left, Line-In Right, …. (ADC does not support Multi-Channel Audio)	Line-Out Left, Line-Out Right, Line-Out Left, Line-Out Right, …. (AC'97 driver mimics Multi-channel audio by copying Line-out Left channel data to Center & Surround Left channels and Line-out Right channel data to LFE & Surround Right channels)
True Multi-channel Audio	Line-In Left, Line-In Right, Line-In Left, Line-In Right, …. (ADC does not support Multi-Channel Audio)	Line-Out Left, Line-Out Right, Center, Surround Left, Surround Right, LFE, Line-Out Left, Line-Out Right, Center, Surround Left, Surround Right, LFE, ….

Table 7 – Audio buffer data formats supported by AC'97 driver

6.6. ADC/DAC sampling rates supported AC'97 Digital Controller driver.

ADC Sampling rates supported: All sampling rates between 7kHz and 48kHz, at 1Hz increment

Double Rate Audio Mode (DRA)	Variable Rate Audio Mode (VRA)	Supported DAC Sampling rates
Disabled	Disabled	48kHz
Disabled	Enabled	8kHz, 11.025kHz, 12kHz, 16kHz, 22.05kHz, 24kHz, 32kHz, 44.1kHz, 48kHz
Enabled	Disabled	96kHz *
Enabled	Enabled	64kHz, * 88.2kHz, * 96kHz *

Table 8 – DAC sampling rates supported by AC'97 Digital Controller Driver

* To operate in these sampling rates, Audio CODEC DAC sampling rate register must be configured to half of actual sampling rate value, with DRA mode enabled. Multi-channel audio is not supported when DRA is enabled.

Note: Application MUST configure VRA and DRA bits before configuring/updating DAC sampling rate register.

7. Hardware Considerations

AD1980 reset pin must be connected to/controlled by a Blackfin GPIO flag.

7.1. AD1980 registers

The following table is a list of registers that can be accessed on the AD1980. Refer to the AD1980 hardware reference manual for register description and chip functionality.

Register	Address	Default	Description
AD1980_REG_RESET	0x00	0x0090	Reset Register
AD1980_REG_MASTER_VOL_CTRL	0x02	0x8000	Master Volume Control Register
AD1980_REG_HP_VOL_CTRL	0x04	0x8000	Headphones Volume Control Register
AD1980_REG_MONO_VOL_CTRL	0x06	0x8000	Mono Volume Control Register
AD1980_REG_PHONE_VOL_CTRL	0x0C	0x8008	Phone In Volume Control Register
AD1980_REG_MIC_VOL_CTRL	0x0E	0x8008	MIC Volume Control Register
AD1980_REG_LINE_IN_VOL_CTRL	0x10	0x8808	Line-In Volume Control Register
AD1980_REG_CD_VOL_CTRL	0x12	0x8808	CD Volume Control Register
AD1980_REG_AUX_VOL_CTRL	0x16	0x8808	AUX Volume Control Register
AD1980_REG_PCM_OUT_VOL_CTRL	0x18	0x8808	PCM Out Volume Control Register
AD1980_REG_RECORD_SELECT	0x1A	0x0000	Record select Control Register
AD1980_REG_RECORD_GAIN	0x1C	0x8000	Record gain Register
AD1980_REG_GP	0x20	0x0000	General purpose Register
AD1980_REG_AUDIO_INT_PAGE	0x24	0xxxxx	Audio Interrupt an Paging
AD1980_REG_POWER_CTRL_STAT	0x26	n/a	Power-down Control/Status Register
AD1980_REG_EXTD_AUDIO_ID	0x28	0xx37C	Extended Audio ID Register
AD1980_REG_EXTD_AUDIO_CTRL	0x2A	0x0xx0	Extended Audio Status/Control Register
AD1980_REG_FRONT_DAC_RATE	0x2C	0xBB80	PCM Front DAC Sample Rate Register
AD1980_REG_SURR_DAC_RATE	0x2E	0xBB80	PCM Surround DAC Sample Rate Register
AD1980_REG_CTR_LFE_DAC_RATE	0x30	0xBB80	PCM Center/LFE DAC Sample Rate Register
AD1980_REG_ADC_RATE	0x32	0xBB80	PCM ADC sample rate Register
AD1980_REG_CTR_LFE_VOL_CTRL	0x36	0x8080	Center/LFE Volume Control Register
AD1980_REG_SURR_VOL_CTRL	0x38	0x8080	Surround Volume Control Register
AD1980_REG_SPDIF_CTRL	0x3A	0x2000	S/PDIF Control Register
AD1980_REG_EQ_CTRL	0x60	0x8080	Equaliser Control Register
AD1980_REG_EQ_DATA	0x62	0x0000	Equaliser Data Register
AD1980_REG_SERIAL_CONFIG	0x74	0x1001	Serial Configuration Register
AD1980_REG_MISC_CTRL	0x76	0x0000	Miscellaneous Control Register
AD1980_REG_VENDOR_ID_1	0x7C	0x4144	Vendor ID Register 1
AD1980_REG_VENDOR_ID_2	0x7E	0x5370	Vendor ID Register 2

Table 9 – AD1980 Device registers

7.2. AD1980 register fields

The following table contains the list of AD1980 register fields that can accessed via Register Field access commands. 'Field' indicates macro name corresponding to a register field, 'Mask' indicates Hex equivalent to corresponding 'Field' and 'Size' indicates the actual width of the corresponding 'Field'.

Field	Mask	Size	Description
Reset Register(AD1980_REG_RESET)			
AD1980_RFLD_ID_CODEC_SUPPORT	0x03FF	10	Identify Codec Capabilities
AD1980_RFLD_3D_SURR_ENHANCE	0x7C00	5	3D Stereo Enhancement support code
Master Volume Control Register (AD1980_REG_MASTER_VOL_CTRL) **Headphones Volume Control Register** (AD1980_REG_HP_VOL_CTRL)			
AD1980_RFLD_MASTER_HP_R_VOL	0x003F	6	Master/HP Right Channel Volume
AD1980_RFLD_MASTER_HP_R_MUTE	0x0080	1	Master/HP Right Channel Mute
AD1980_RFLD_MASTER_HP_L_VOL	0x3F00	6	Master/HP Left Channel Volume
AD1980_RFLD_MASTER_HP_L_MUTE	0x8000	1	Master/HP Left Channel Mute
AD1980_RFLD_MASTER_HP_MUTE	0x8000	1	Mute Headphone/Master Volume
Mono Volume Control Register (AD1980_REG_MONO_VOL_CTRL)			
AD1980_RFLD_MONO_VOL	0x003F	6	Mono Volume
AD1980_RFLD_MONO_MUTE	0x8000	1	Mute Mono Volume
Phone In Volume Control Register (AD1980_REG_PHONE_VOL_CTRL)			
AD1980_RFLD_PHONE_VOL	0x001F	5	Phone In Volume
AD1980_RFLD_PHONE_MUTE	0x8000	1	Mute Phone In Volume
MIC Volume Control Register (AD1980_REG_MIC_VOL_CTRL)			
AD1980_RFLD_MIC_VOL_GAIN	0x001F	5	MIC Volume Gain
AD1980_RFLD_MIC_BOOST_GAIN	0x0040	2	MIC Volume Gain
AD1980_RFLD_MIC_MUTE	0x8000	1	Mute MIC
Line-In Volume Control Register (AD1980_REG_LINE_IN_VOL_CTRL) **CD Volume Control Register** (AD1980_REG_CD_VOL_CTRL) **AUX Volume Control Register** (AD1980_REG_AUX_VOL_CTRL)			
AD1980_RFLD_AUX_CD_LINE_IN_R_VOL	0x001F	5	AUX/CD/Line In Right Channel Volume
AD1980_RFLD_AUX_CD_LINE_IN_R_MUTE	0x0080	1	AUX/CD/Line In Right Channel Mute
AD1980_RFLD_AUX_CD_LINE_IN_L_VOL	0x1F00	5	AUX/CD/Line In Left Channel Volume
AD1980_RFLD_AUX_CD_LINE_IN_L_MUTE	0x8000	1	AUX/CD/Line In Left Channel Mute
AD1980_RFLD_AUX_CD_LINE_IN_MUTE	0x8000	1	Mute AUX/CD/Line In Volume
PCM Out Volume Control Register (AD1980_REG_PCM_OUT_VOL_CTRL)			
AD1980_RFLD_PCM_OUT_R_VOL	0x001F	5	PCM Out Right Channel Volume
AD1980_RFLD_PCM_OUT_R_MUTE	0x0080	1	PCM Out Right Channel Mute
AD1980_RFLD_PCM_OUT_L_VOL	0x1F00	5	PCM Out Left Channel Volume
AD1980_RFLD_PCM_OUT_L_MUTE	0x8000	1	PCM Out Left Channel Mute
AD1980_RFLD_PCM_OUT_MUTE	0x8000	1	Mute PCM Out Volume

Field	Mask	Size	Description
Record Select Register (AD1980_REG_RECORD_SELECT)			
AD1980_RFLD_RECORD_SELECT_R	0x0007	3	Right Record Select
AD1980_RFLD_RECORD_SELECT_L	0x0700	3	Left Record Select
Record Gain Register (AD1980_REG_RECORD_GAIN)			
AD1980_RFLD_RECORD_R_GAIN	0x000F	4	Right Input Mixer Gain
AD1980_RFLD_RECORD_R_MUTE	0x0080	1	Right Input Channel Mute
AD1980_RFLD_RECORD_L_GAIN	0x0F00	4	Left Input Mixer Gain
AD1980_RFLD_RECORD_L_MUTE	0x8000	1	Left Input Channel Mute
AD1980_RFLD_RECORD_INPUT_MUTE	0x8000	1	Mute Input Channels
General Purpose Register (AD1980_REG_GP)			
AD1980_RFLD_LOOPBACK	0x0080	1	Loopback Control
AD1980_RFLD_MIC_SELECT	0x0100	1	Selects Mono MIC Input
AD1980_RFLD_DOBULE_RATE_SLOTS	0x0C00	2	Double Rate Slot Select
Power Control/Status Register (AD1980_REG_POWER_CTRL)			
AD1980_RFLD_ADC_READY	0x0001	1	ADC Sections Ready to Transmit
AD1980_RFLD_DAC_READY	0x0002	1	DAC Sections Ready to Transmit
AD1980_RFLD_ANL_READY	0x0004	1	Analog Amps, Attenuators, Mixers Ready
AD1980_RFLD_VREF	0x0008	1	Voltage references up to nominal level
AD1980_RFLD_PDN_ADC	0x0100	1	Power Down PCM In ADCs & Input Mux
AD1980_RFLD_PDN_DAC	0x0200	1	Power Down PCM Out DACs
AD1980_RFLD_PDN_MIXER	0x0400	1	PowerDown Analog Mixer(Vref still on)
AD1980_RFLD_PDN_VREF	0x0800	1	Power Down Analog Mixer(Vref off)
AD1980_RFLD_PDN_AC_LINK	0x1000	1	PowerDown AC Link (Digital interface)
AD1980_RFLD_ICLK_DISABLE	0x2000	1	Internal Clock Disable
AD1980_RFLD_PDN_AUX_OUT	0x4000	1	Power Down AUX Out
AD1980_RFLD_PDN_EX_AMP	0x8000	1	Power Down External power amplifier
Extended Audio ID Register (AD1980_REG_EXTD_AUDIO_ID)			
AD1980_RFLD_VRA_SUPPORT	0x0001	1	Variable rate PCM Audio support
AD1980_RFLD_DRA_SUPPORT	0x0002	1	Double rate Audio support
AD1980_RFLD_SPDIF_SUPPORT	0x0004	1	SPDIF support
AD1980_RFLD_DAC_SLOT_ASSIGN	0x0030	2	DAC Slot Assignments
AD1980_RFLD_CTR_DAC_SUPPORT	0x0040	1	PCM Center DAC Support
AD1980_RFLD_SURR_DAC_SUPPORT	0x0080	1	PCM Surround DAC Support
AD1980_RFLD_LFE_DAC_SUPPORT	0x0100	1	PCM LFE DAC Support
AD1980_RFLD_AMAP_SUPPORT	0x0200	1	Support for Slot Mappings based on codec id
AD1980_RFLD_REV_INFO	0x0C00	2	AD1980 revision compliant info
AD1980_RFLD_CODEC_CONFIG_ID	0xC000	2	Codec Configuration ID
Extended Audio Status/Control Register (AD1980_REG_EXTD_AUDIO_CTRL)			
AD1980_RFLD_ENABLE_VRA	0x0001	1	1 = Enable Variable Rate Audio

Field	Mask	Size	Description
AD1980_RFLD_ENABLE_DRA	0x0002	1	1 = Enable Double Rate Audio
AD1980_RFLD_ENABLE_SPDIF	0x0004	1	1 = Enable SPDIF Tx Subsystem
AD1980_RFLD_SPDIF_SLOT_ASSIGN	0x0030	2	SPDIF Slot Assignment Bits
AD1980_RFLD_CTR_DAC_READY	0x0040	1	Center DAC Status (1 = Ready)
AD1980_RFLD_SURR_DAC_READY	0x0080	1	Surround DAC Status (1 = Ready)
AD1980_RFLD_LFE_DAC_READY	0x0100	1	LFE DAC Status (1 = Ready)
AD1980_RFLD_SPDIF_CONFIG_VALID	0x0200	1	1 = SPDIF Configuration Valid
AD1980_RFLD_PDN_CTR_DAC	0x0800	1	Power Down Center DAC
AD1980_RFLD_PDN_SURR_DAC	0x1000	1	Power Down Surround DAC
AD1980_RFLD_PDN_LFE_DAC	0x2000	1	Power Down LFE DAC
AD1980_RFLD_FORCE_SPDIF_VALID	0x8000	1	Enable/Disable SPDIF stream validity flag control
Center/LFE Volume Control Register (AD1980_REG_CTR_LFE_VOL_CTRL)			
AD1980_RFLD_CTR_VOL	0x003F	6	Center Volume Control
AD1980_RFLD_CTR_MUTE	0x0080	1	Center Volume Mute
AD1980_RFLD_LFE_VOL	0x3F00	6	LFE Volume Control
AD1980_RFLD_LFE_MUTE	0x8000	1	LFE Volume Mute
Surround Volume Control Register (AD1980_REG_SURR_VOL_CTRL)			
AD1980_RFLD_SURR_R_VOL	0x001F	5	Surround Right Channel Volume
AD1980_RFLD_SURR_R_MUTE	0x0080	1	Surround Right Channel Mute
AD1980_RFLD_SURR_L_VOL	0x1F00	5	Surround Left Channel Volume
AD1980_RFLD_SURR_L_MUTE	0x8000	1	Surround Left Channel Mute
S/PDIF Control Register (AD1980_REG_SPDIF_CTRL)			
AD1980_RFLD_SPDIF_PRO	0x0001	1	1 = Professional, 0 = Consumer Audio
AD1980_RFLD_SPDIF_NON_AUDIO	0x0002	1	1 = Non-PCM Audio, 0 = PCM Audio
AD1980_RFLD_SPDIF_COPYRIGHT	0x0004	1	1 = Copyright asserted
AD1980_RFLD_SPDIF_PRE_EMPHASIS	0x0008	1	1 = 50us/15us Pre-Emp, 0 = No Pre-Emp
AD1980_RFLD_SPDIF_CATEFORY_CODE	0x07F0	7	Category Code
AD1980_RFLD_SPDIF_GEN_LEVEL	0x0800	1	Generation Level
AD1980_RFLD_SPDIF_SAMPLE_RATE	0x3000	2	SPDIF Transmit Sample Rate
AD1980_RFLD_SPDIF_VALIDITY	0x8000	1	SPDIF validity flag
Equaliser Control Register (AD1980_REG_EQ_CTRL)			
AD1980_RFLD_BIQUAQ_COEFF_ADDR	0x003F	6	Biquad & Coefficient address pointer
AD1980_RFLD_EQ_CHANNEL_SELECT	0x0040	1	0 = Left channel, 1 = Right channel
AD1980_RFLD_EQ_SYMMETRY	0x0080	1	1 = Right & Left channel coefficients are equal
AD1980_RFLD_EQ_MUTE	0x8000	1	1 = Equaliser mute
Serial Configuration Register (AD1980_REG_SERIAL_CONFIG)			
AD1980_RFLD_SPDIF_LINK	0x0001	1	1 = SPDIF and DAC are linked, 0 = Not linked
AD1980_RFLD_SPDIF_DACZ	0x0002	1	when FIFO under-runs, 1 = Forces mid scale sample out, 0 = Repeat last sample out

Field	Mask	Size	Description
AD1980_RFLD_SPDIF_ADC_LOOPAROUND	0x0004	1	1 = Tx connected to AC-Link Stream, 0 = Tx connected to digital ADC stream
AD1980_RFLD_INT_MODE_SELECT	0x0010	1	1 = Bit 0 SLOT 12, 0 = Slot 6 valid bit
AD1980_RFLD_LOOPBACK_SELECT	0x0060	2	Loopback Select
AD1980_RFLD_ENABLE_CHAIN	0x0100	1	1 = Enable Chain
AD1980_RFLD_DAC_REQUEST_FORCE	0x0400	1	Sync DAC requests
AD1980_RFLD_SLAVE3_CODEC_MASK	0x0800	1	Slave 3 Codec Register Mask
AD1980_RFLD_MASTER_CODEC_MASK	0x1000	1	Master Codec Register Mask
AD1980_RFLD_SLAVE1_CODEC_MASK	0x2000	1	Slave 1 Codec Register Mask
AD1980_RFLD_SLAVE2_CODEC_MASK	0x4000	1	Slave 2 Codec Register Mask
AD1980_RFLD_ENABLE_SLOT16	0x8000	1	1 = Enable Slot 16 Mode
Miscellaneous Control Bit Register (AD1980_REG_MISC_CTRL)			
AD1980_RFLD_MIC_BOOST_GAIN	0x0003	2	MIC Boost gain select
AD1980_RFLD_DISABLE_VREF_OUT	0x0004	1	1 = Set VREFOUT in High impedance mode, 0 = VREFOUT driven by internal reference
AD1980_RFLD_VREF_OUT_HIGH	0x0008	1	1 = VREFOUT set to 3.70 V, 0 = VREFOUT set to 2.25 V
AD1980_RFLD_SAMPLE_RATE_UNLOCK	0x0010	1	1 = DAC sample rates can be set independently, 0 = All DAC sample rates locked to front sample rate
AD1980_RFLD_LINE_OUT_AMP_INPUT	0x0020	1	1 = LINE_OUT Amps driven by surround DAC output, 0 = LINE_OUT Amps driven by Mixer output
AD1980_RFLD_2CHANNEL_MIC_SELECT	0x0040	1	1 = Record from MIC1 & MIC2, 0 = determined by AD1980_RFLD_MIC_SELECT field value
AD1980_RFLD_ENABLE_SPREAD	0x0080	1	1 = Enable Spread
AD1980_RFLD_DOWN_MIX_MODE	0x0300	2	Down Mix Mode Select
AD1980_RFLD_HEADPHONE_AMP_SELECT	0x0400	1	Headphone Amplifier Select
AD1980_RFLD_DISABLE_CTR_LFE_PINS	0x0800	1	1 = Disable Center and LFE Out pins
AD1980_RFLD_DISABLE_LINE_OUT_PINS	0x1000	1	1 = Disable Line Out (L & R) pins
AD1980_RFLD_ENABLE_MUTE_SPLIT	0x2000	1	1 = Enable separate Mute Control
AD1980_RFLD_ENABLE_ADI_MODE	0x4000	1	1 = Enable ADI Compatibility mode
AD1980_RFLD_DAC_ZERO_FILL	0x8000	1	when DAC's starved for data, 1 = DAC data is zero-filled , 0 = repeat DAC data

Table 10 – AD1980 Register Fields

8. Appendix

8.1. Using AC'97 / AD1980 Driver in Applications

This section explains how to use AC'97 Digital Controller and AD1980 drivers in an application.

8.1.1. Interrupt Manager Data memory allocation

This section explains Interrupt manager memory allocation requirements for applications using above drivers. AC'97 is a full-duplex protocol and the application must allocate memory for SPORT Transmit and Receive DMA channels regardless of AD1980 dataflow mode, where each DMA channel requires memory for two secondary interrupt handlers of size `ADI_INT_SECONDARY_MEMORY`. Additional memory of size `ADI_INT_SECONDARY_MEMORY` must be provided for SPORT error reporting.

In total, AC'97 and AD1980 driver requires memory for **five** secondary interrupts.

8.1.2. DMA Manager Data memory allocation

This section explains DMA manager memory allocation requirements for applications using AC'97 and AD1980 drivers. AC'97 is a full-duplex protocol and the application must allocate memory for SPORT Transmit and Receive DMA channels regardless of AD1980 dataflow mode. Application must allocate DMA base memory + memory for two SPORT DMA channels + memory for DMA channels used by other devices in the application

8.1.3. Device Manager Data memory allocation

This section explains device manager memory allocation requirements for applications using AC'97 and AD1980 drivers. Application must allocate Device Manager base memory + memory for one SPORT device + memory for one AD1980 device + memory for other devices used by the application.

8.1.4. Using AC'97 / AD1980 drivers

a. AD1980 driver initialization

Step 1: Open AD1980 Device driver with device specific entry point (refer section 6.1 for valid entry point)

Step 2: Initialize AD1980 driver instance (refer to section 6.4 for example code)

Step 3: Set AD1980 driver to use Deferred or Live callback to process AC'97 frames

```
/* Example: Set AD1980 driver to use Deferred Callback to process AC'97 frames */
adi_dev_Control (AD1980DriverHandle, ADI_AD1980_CMD_USE_DCB_TO_PROCESS_FRAMES,
                                                    (void *) DCBManagerHandle);

                                (OR)

/* Example: Set AD1980 driver to process AC'97 frames Live */
adi_dev_Control (AD1980DriverHandle, ADI_AD1980_CMD_USE_DCB_TO_PROCESS_FRAMES, (void *) NULL);
```

b. AD1980 hardware initialization

Step 4: Configure AD1980 device to specific mode using device access commands
(refer section 8.2 and 8.3 for examples)

c. Set AD1980 Audio mode

Step 4: Set AD1980 Audio mode (Stereo / Pseudo Multi-channel / True Multi-channel mode)

```
/* Example: Set AD1980 in stereo mode */
adi_dev_Control(AD1980DriverHandle, ADI_AC97_CMD_ENABLE_MULTICHANNEL_AUDIO, (void *)FALSE));
```

(OR)

```
/* Example: Set AD1980 in Pseudo Multi-channel Audio mode */
adi_dev_Control(AD1980DriverHandle, ADI_AC97_CMD_ENABLE_TRUE_MULTICHANNEL_AUDIO, (void *)FALSE));
```

(OR)

```
/* Example: Set AD1980 in True Multi-channel Audio mode */
adi_dev_Control (AD1980DriverHandle, ADI_AC97_CMD_ENABLE_TRUE_MULTICHANNEL_AUDIO, (void *)TRUE));
```

d. AD1980 Audio dataflow configuration

Step 5: Set AD1980 Audio dataflow method

```
/* Example: Set AD1980 in chained dataflow mode */
adi_dev_Control(AD1980DriverHandle, ADI_DEV_CMD_SET_DATAFLOW_METHOD,
                                                    (void*)ADI_DEV_MODE_CHAINED);
```

Step 6: Flush DAC buffer queue

```
/* Example: Flush DAC application buffer queue before submitting new buffers */
adi_dev_Read (AD1980DriverHandle,  ADI_AC97_CMD_CLEAR_DAC_APP_BUFFER_PTRS, (void *)NULL);
```

Step 7: Flush ADC buffer queue

```
/* Example: Flush ADC application buffer queue before submitting new buffers */
adi_dev_Read (AD1980DriverHandle,  ADI_AC97_CMD_CLEAR_ADC_APP_BUFFER_PTRS, (void *)NULL);
```

Step 8: Submit Inbound/Outbound buffers

```
/* Example:  Submit Inbound Audio Buffer */
adi_dev_Read (AD1980DriverHandle, ADI_DEV_1D, (ADI_DEV_BUFFER *)&InboundBuffer);
/* Example:  Submit Outbound Audio Buffer */
adi_dev_Write (AD1980DriverHandle, ADI_DEV_1D, (ADI_DEV_BUFFER *)&OutboundBuffer);
```

Step 9: Enable AD1980 audio dataflow

```
/* Example:  Enable AD1980 Audio Dataflow */
adi_dev_Control(AD1980DriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE);
```

e. Terminating AD1980 driver

Step14: Disable AD1980 audio dataflow

```
/* Example: Disable AD1980 Audio Dataflow */
adi_dev_Control(AD1980DriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *)FALSE);
```

Step15: Close AD1980 driver

```
/* Example: Close AD1980 driver */
adi_dev_Close(AD1980DriverHandle);
```

Terminate DMA Manager, Deferred Callback etc.., (application dependent).

8.2. Accessing AD1980 registers

This section explains how to access the AD1980 internal registers using device access commands (refer *'adi_deviceaccess'* documentation for more information).

Refer section 7.1 for list of AD1980 device registers and section 7.2 for list of AD1980 device registers fields

8.2.1. Read AD1980 internal registers

1. Read a single register

```
/*define the structure to access a single device register */
ADI_DEV_ACCESS_REGISTER Read_Reg;

/*Load the register address to be read */
Read_Reg.Address = AD1980_REG_PCM_OUT_VOL_CTRL;

/* Clear the Data location */
Read_Reg.Data = 0;
/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Register value will be read back to location - Read_Reg.Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_READ, (void *) &Read_Reg);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) for a valid register data */
```

2. Read a specific register field

```
/* define the structure to access a specific device register field */
ADI_DEV_ACCESS_REGISTER_FIELD Read_Field;

/* Load the device register address to be accessed */
Read_Field.Address = AD1980_REG_PCM_OUT_VOL_CTRL;
/* Load the device register field location to be read */
Read_Field.Address = AC97_RFLD_PCM_OUT_L_MUTE;

/* clear the Read_Field.Data location */
Read_Field.Data = 0;
/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Register field value will be read back to location - Read_Field.Data */
adi_dev_Control (DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_READ, (void *) &Read_Field);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) for a valid register data */
```

3. Read table of registers

```
/* define the structure to access table of device registers */
ADI_DEV_ACCESS_REGISTER Read_Regs [ ] = {
                                { AD1980_REG_MASTER_VOL_CTRL,      0},
                                { AD1980_REG_PCM_OUT_VOL_CTRL,     0},
                                { AD1980_REG_SERIAL_CONFIG,        0},
   /*MUST include delimiter */ { ADI_DEV_REGEND,                   0}      /* Register access delimiter */
                                };

/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Present value of registers listed above will be read to corresponding Data location in Read_Regs array */
/* i.e., value of AD1980_REG_MASTER_VOL_CTRL will be read to Read_Regs[0].Data,
  AD1980_REG_PCM_OUT_VOL_CTRL to Read_Regs[1].Data and
  AD1980_REG_SERIAL_CONFIG to Read_Regs[2].Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_READ, (void *) &Read_Regs[0]);
```

4. Read table of register(s) fields

```
/* define the structure to access table of device register(s) fields */
ADI_DEV_ACCESS_REGISTER_FIELD Read_Fields [ ] = {
                          { AD1980_REG_EXTD_AUDIO_ID,      AD1980_RFLD_VRA_SUPPORT,          0},
                          { AD1980_REG_EXTD_AUDIO_ID,      AD1980_RFLD_DRA_SUPPORT,          0},
                          { AD1980_REG_EXTD_AUDIO_ID,      AD1980_RFLD_SURR_DAC_SUPPORT,     0},
/*MUST include delimiter */ {ADI_DEV_REGEND,               0,          0}    /* Register access delimiter */
                          };

/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Present value of register fields listed above will be read to corresponding Data location in Read_Fields array */
/* i.e., value of AD1980_RFLD_VRA_SUPPORT will be read to Read_Fields[0].Data,
   AD1980_RFLD_DRA_SUPPORT to Read_Fields [1].Data and
   AD1980_RFLD_SURR_DAC_SUPPORT to Read_Fields [2].Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_READ, (void *) &Read_Fields[0]);
```

8.2.2. Configure AD1980 internal registers

1. Configure a single AD1980 register

```
/* define the structure to access a single device register */
ADI_DEV_ACCESS_REGISTER Cfg_Reg;

/* Load the register address to be configured */
Cfg_Reg.Address = AD1980_REG_SERIAL_CONFIG;

/* Load the configuration value to Cfg_Reg.Data location */
Cfg_Reg.Data = 0x9000;
/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* The device register will be configured with the value in Cfg_Reg.Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_WRITE, (void *) &Cfg_Reg);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) before issuing
a new register access request */
```

2. Configure a specific register field

```
/* define the structure to access a specific device register field */
ADI_DEV_ACCESS_REGISTER_FIELD Cfg_Field;

/* Load the device register address to be accessed */
Cfg_Field.Address = AD1980_REG_PCM_OUT_VOL_CTRL;
/* Load the device register field location to be configured */
Cfg_Field.Address = AC97_RFLD_PCM_OUT_R_VOL;

/* load the new field value */
Cfg_Field.Data = 0x07;
/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Register field will be updated to 'Cfg_Field.Data' value */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_WRITE, (void *) &Cfg_Field);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) before issuing
a new register access request */
```

3. Configure table of registers

```
/* define the structure to access table of device registers (register address, register configuration value)
ADI_DEV_ACCESS_REGISTER Cfg_Regs [ ] = {
                        { AD1980_REG_SERIAL_CONFIG,      0x9000 },
                        { AD1980_REG_RECORD_SELECT,      0x0404 },
                        { AD1980_REG_POWER_CTRL_STAT, 0x030C },
                        { AD1980_REG_FRONT_DAC_RATE,  48000  },
                        { AD1980_REG_ADC_RATE,           48000  },
/*MUST include delimiter */ {ADI_DEV_REGEND,             0      } /* Register access delimiter */
                        };

/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Registers listed in the table will be configured with corresponding table Data values */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *) &Cfg_Regs[0]);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) before issuing
a new register access request */
```

4. Configure a table of register(s) fields

```
/* define the structure to access table of device register(s) fields */
/* register address, register field to configure, field configuration value */
ADI_DEV_ACCESS_REGISTER_FIELD Cfg_Fields [ ] = {
    { AD1980_REG_PCM_OUT_VOL_CTRL,      AC97_RFLD_PCM_OUT_MUTE,      0      },
    { AD1980_REG_PCM_OUT_VOL_CTRL,      AC97_RFLD_PCM_OUT_R_MUTE,    0      },
    { AD1980_REG_PCM_OUT_VOL_CTRL,      AC97_RFLD_PCM_OUT_R_VOL,     0x07   },
    { AD1980_REG_PCM_OUT_VOL_CTRL,      AC97_RFLD_PCM_OUT_R_VOL,     0x05   },
    {ADI_DEV_REGEND,                    0, 0}   /* Register access delimiter (MUST include delimiter) */
};

/* Application calls adi_dev_Control( ) function with corresponding command and value */
/* Register fields listed in the above table will be configured with corresponding Data values */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *) &Cfg_Fields[0]);

/* Wait for ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event (Event code: 0x402C0001) before issuing
a new register access request */
```

8.3. AD1980 Register configuration tables

```
/* AD1980 register configuration table */
ADI_DEV_ACCESS_REGISTER ConfigAD1980Reg[] =
{
        { AD1980_REG_SERIAL_CONFIG,         0x9000      },    /* Enable ADI slot-16 mode                */
        { AD1980_REG_RECORD_SELECT,         0x0404      },    /* Select Line In as record source        */
        { AD1980_REG_MASTER_VOL_CTRL,       0x0000      },    /* Unmute Master Volume, 0dB              */
        { AD1980_REG_LINE_IN_VOL_CTRL,      0x0000      },    /* Unmute Line In Volume, 0dB             */
        { AD1980_REG_PCM_OUT_VOL_CTRL,      0x0F0F      },    /* Unmute PCM Out Volume, -22.5dB         */
        { AD1980_REG_RECORD_GAIN,           0x0000      },    /* Unmute Record Gain, 0dB                */
        { AD1980_REG_CTR_LFE_VOL_CTRL,      0x0000      },    /* Unmute Center/LFE Volume, 0dB          */
        { AD1980_REG_SURR_VOL_CTRL,         0x0000      },    /* Unmute Surround out Volume, 0dB        */
        { ADI_DEV_REGEND,                   0           }     /* End of register access                 */
};

/* AD1980 Sample rate configuration table */
ADI_DEV_ACCESS_REGISTER AD1980SampleRate[] =
{
        { AD1980_REG_POWER_CTRL_STAT,       0x030C  },    /* Power down ADC/DAC before changing sampling rate*/
        { AD1980_REG_EXTD_AUDIO_CTRL,       0x0000  },    /* Disable variable audio rate & Double audio rate */
        { AD1980_REG_FRONT_DAC_RATE,        48000   },    /* Set Front out DAC sample rate (48kHz)           */
        { AD1980_REG_SURR_DAC_RATE,         48000   },    /* Set Surround DAC sample rate (48kHz)            */
        { AD1980_REG_CTR_LFE_DAC_RATE,      48000   },    /* Set Center/LFE DAC sample rate  (48kHz)         */
        { AD1980_REG_ADC_RATE,              48000   },    /* Set ADC sample rate (48kHz)                     */
        { AD1980_REG_POWER_CTRL_STAT,       0x000F  },    /* Power-up the DACs & ADC                         */
        { ADI_DEV_REGEND,                   0       }     /* End of register access                          */
};
```

8.4. Example code to update AD1980 ADC/DAC sampling rate

```
/*********************************************************************
Function:       UpdateSamplingRate
Description:    Updates Audio CODEC sampling rate
*********************************************************************/

u32     UpdateSamplingRate (
        u32     SamplingRate    /* New Sampling rate */
) {
        /* Return code */
        u32     Result = ADI_DEV_RESULT_SUCCESS;

        /* IF (Set Sample rate to 48kHz or 96kHz) */
        if ((SamplingRate == 48000) || (SamplingRate == 96000))
        {
                AD1980SampleRate[1].Data &= ~AD1980_RFLD_ENABLE_VRA;        /* disable Variable rate audio */
        }
        else
        {
                AD1980SampleRate[1].Data |= AD1980_RFLD_ENABLE_VRA;         /* enable Variable rate audio */
        }

        /* IF (Sampling rate is greater than 48kHz) */
        if (SamplingRate > 48000)
        {
                AD1980SampleRate[1].Data |= AD1980_RFLD_ENABLE_DRA;         /* enable Double rate audio */
                SamplingRate = SamplingRate/2;                              /* and half the given sampling rate */
        }
        /* ELSE (Sampling rate must be less than or equal to 48kHz) */
        else
        {
                AD1980SampleRate[1].Data &= ~AD1980_RFLD_ENABLE_DRA;        /* disable Double rate audio */
        }

        /* update sampling rate configuration table */
        AD1980SampleRate[2].Data = SamplingRate;        /* update Front out DAC sample rate        */
        AD1980SampleRate[3].Data = SamplingRate;        /* update Surround DAC sample rate         */
        AD1980SampleRate[4].Data = SamplingRate;        /* update Center/LFE DAC sample rate       */
        AD1980SampleRate[5].Data = SamplingRate;        /* update ADC sample rate                  */

        /* 'AC97RegAccessDoneFlag' is a volatile u8 type global variable defined by the application and is set to TRUE in
           AD1980  callback function on occurrence of  ADI_AC97_EVENT_REGISTER_ACCESS_COMPLETE event */
        while(AC97RegAccessDoneFlag == FALSE)
        {
                idle();         /* wait until the previous AC'97 register access request gets serviced */
        }
        /* lock AC'97 register access */
        AC97RegAccessDoneFlag = FALSE;

        /* Update AD1980 registers with new sampling rate */
        Result = adi_dev_Control(AD1980DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_WRITE,
                                                        (void *)AD1980SampleRate);

        while(AC97RegAccessDoneFlag == FALSE)
        {
                idle();         /* wait until the AD1980 registers are updated with new sampling rate */
        }
        /* return */
        return (Result);
}
```

9. References

1. Analog Devices AD1980 AC'97 version 2.3 compatible Audio CODEC DataSheet, Rev 0, 2002
http://www.analog.com/UploadedFiles/Data_Sheets/AD1980.pdf