# ADI_AD1871
# DEVICE DRIVER

**DATE: FEBRUARY 13, 2006**

# Table of Contents

**Document Revision History**

| Date | Description of Changes |
|---|---|
| 2006/02/13 | Initial release |

# 1. Overview

The driver configures SPORT 0 of a Blackfin device to receive the digital audio data from AD1871. The client can set the SPORT device to I2S mode by issuing a command specific to the driver.

# 2. Files

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

## 2.1. Include Files

adi_ad1871.h

## 2.2. Source Files

adi_ad1871.c

# 3. Lower Level Drivers

AD1871 driver is layered on top of SPORT driver.

## 3.1. SPORT Device Driver

The driver is set to use Serial Port (SPORT) device 0 of the Blackfin device to receive the digital audio data from AD1871.

# 4. Resources Required

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

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 access and control any required hardware. This section describes the resources used by this driver and memory requirements of the System Services to support to the driver.

The AD1871 device driver is build upon DMA operated SPORT driver

## 4.1. Interrupts

No specific interrupts or interrupt handlers are used by this driver.

## 4.2. DMA

The driver doesn't support DMA directly, but uses a DMA driven SPORT to receive digital audio data from AD1871. The driver supports only inbound dataflow and enough memory should be allocated for the DMA channels.

## 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

No Programmable Flags are used by this driver

## 4.6. Pins

Pins specific to SPORT 0 of the Blackfin device should be wired up to the digital audio output pins of AD1871. Please check corresponding device reference manual for more information.

# 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. |
| | |
| | |

## 5.2. Dataflow Methods

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

| ADI_DEV_MODE | Description |
|---|---|
| ADI_DEV_MODE_CIRCULAR | Supports the circular buffer method |
| ADI_DEV_MODE_CHAINED | Supports the chained buffer method |
| ADI_DEV_MODE_CHAINED_LOOPBACK | Supports the chained buffer with loopback method |
| | |
| | |

## 5.3. Buffer Types

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

- ADI_DEV_CIRCULAR_BUFFER
  - Circular buffer
  - pAdditionalInfo – ignored
- ADI_DEV_1D_BUFFER
  - Linear one-dimensional buffer
  - pAdditionalInfo – ignored
- ADI_DEV_2D_BUFFER
  - Two-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)

### 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. The driver also supports commands specific to SPORT driver. Please refer to SPORT driver documentation for further information.

- ADI_AD1871_CMD_SET_I2S_MODE
  - Sets SPORT device 0 in I2S Mode
  - Value – NULL

## 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 callback function of the client should be prepared to process each of the events in these 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 chained or sequential I/O buffer has been processed by the device driver. This event is also used to notify that an entire circular buffer has been processed if the driver was directed to generate a callback upon completion of an entire circular buffer.
  - Value – For chained or sequential I/O dataflow methods, this value is the CallbackParameter value that was supplied in the buffer that was passed to the adi_dev_Read(), adi_dev_Write() or adi_dev_SequentialIO() function. For the circular dataflow method, this value is the address of the buffer provided in the adi_dev_Read() or adi_dev_Write() function.
- ADI_DEV_EVENT_SUB_BUFFER_PROCESSED
  - Notifies callback function that a sub-buffer within a circular buffer has been processed by the device driver.
  - Value – The address of the buffer provided in 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.

This driver doesn't have any unique events.

## 5.6. Return Codes

This section enumerates the return codes that the device driver API returns. All API functions of the device driver return status indicating either successful completion of the function or an indication that an error has occurred. 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. Wherever functions in the device driver API are called, the application should be prepared to process any of these return codes.

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_TWI_LOCKED
  - Indicates the present TWI device is locked in other operation
- ADI_DEV_RESULT_REQUIRES_TWI_CONFIG_TABLE
  - Client need to supply a configuration table for the TWI driver
- 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.

This driver doesn’t have any specific return codes.

# 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. 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 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

## 6.2. 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 |
|---|---|---|
| SPORT 0 mode | I2S | ADI_AD1871_CMD_SET_I2S_MODE |
| | | |
| | | |

# 7. Hardware Considerations

Pin 2 of AD1871 is used to select the clock rate of the device. This pin is used to select between an MCLK of 256 x $f_S$ (pin low) or 512 x $f_S$ (pin high).