ANALOG
DEVICES

ADI_AD7877
DEVICE DRIVER

DATE: JANUARY 24, 2007

Table of Contents

1. Overview **7**

2. Files **8**

2.1. Include Files 8

2.2. Source Files 8

3. Lower Level Drivers **9**

3.1. SPI Device Driver 9

4. Resources Required **10**

4.1. Interrupts 10

4.2. DMA 10

4.3. Timers 10

4.4. Real-Time Clock 10

4.5. Programmable Flags 10

4.6. Pins 10

5. Supported Features of the Device Driver **11**

5.1. Directionality 11

5.2. Dataflow Methods 11

5.3. Buffer Types 11

5.4. Command IDs 11

5.4.1. Device Manager Commands 11

5.4.2. Common Commands 12

5.4.3. Device Driver Specific Commands 12

5.5. Callback Events 13

5.5.1. Common Events 13

5.5.2. Device Driver Specific Events 13

5.6. Return Codes 14

5.6.1. Common Return Codes 14

5.6.2. Device Driver Specific Return Codes 15

6. Opening and Configuring the Device Driver **16**

6.1. Entry Point 16

6.2. Default Settings 16

6.3. Additional Required Configuration Settings 16

7. Hardware Considerations **17**

7.1. AD7877 registers 17

7.1.1. AD7877 Device read/write type registers 17

7.1.2. AD7877 Device read-only type registers 17

7.1.3. AD7877 Device extended read/write type registers 18

7.2. AD7877 register fields 18

8. Using AD7877 Driver in Applications 21

8.1. Device Manager Data memory allocation 21

8.2. Interrupt Manager Data memory allocation 21

8.3. Flag Manager Data memory allocation 21

8.4. Typical usage of AD7877 driver 21

8.5. Configuring AD7877 on ADSP-BF548 Ez-Kit Lite 22

8.6. Re-use/share the SPI device used by AD7877 22

8.7. Resetting AD7877 device 22

9. Data Structures used by AD7877 driver 23

9.1. ADI_DEV_ACCESS_REGISTER 23

9.2. ADI_DEV_ACCESS_REGISTER_BLOCK 23

9.3. ADI_DEV_ACCESS_REGISTER_FIELD 23

9.4. ADI_AD7877_INTERRUPT_PORT 23

9.5. ADI_AD7877_RESULT_REGS 24

10. Programming Examples 25

10.1. Reading AD7877 device registers 25

10.1.1. Read a single AD7877 device register 25

10.1.2. Read a single AD7877 device register field 25

10.1.3. Read a table of AD7877 device registers 25

10.1.4. Read a table of AD7877 device register(s) Fields 26

10.1.5. Read a block of AD7877 device registers 26

10.2. Configuring AD7877 device registers 27

10.2.1. Configure a single AD7877 register 27

10.2.2. Configure a single AD7877 register field 27

10.2.3. Configure a table of AD7877 registers 27

10.2.4. Configure a table of AD7877 register(s) fields 28

10.2.5. Configure a block of AD7877 registers 28

10.3. Command to set AD7877 SPI Device Number 28

10.4. Command to set AD7877 SPI Chipselect 29

10.5. Commands to enable AD7877 interrupt monitoring 29

10.5.1. Enable PENIRQ Monitoring 29

10.5.2. Enable DAV Monitoring 29

10.5.3. Enable ALERT Monitoring 29
10.6. Commands to Disable AD7877 interrupt monitoring 30
10.6.1. Disable PENIRQ Monitoring 30
10.6.2. Disable DAV Monitoring 30
10.6.3. Disable ALERT Monitoring 30

11. AD7877 Driver Flow 31
11.1. Reading AD7877 registers 31
11.2. Configuring AD7877 registers 32
11.3. Monitoring PENIRQ Interrupt 33
11.4. Monitoring DAV interrupt 34
11.5. Monitoring ALERT interrupt 35

12. References 36

List of Tables

Table 1 – Revision History 6
Table 2 – Supported Dataflow Directions 11
Table 3 – Supported Dataflow Methods 11
Table 4 – Default Settings 16
Table 5 – Additional Required Settings 16
Table 6 – AD7877 device read/write type registers 17
Table 7 – AD7877 device read-only type registers 18
Table 8 – AD7877 device extended read/write type registers 18
Table 9 – AD7877 Register Fields 20

Document Revision History

Date	Description of Changes
Jan 24, 2007	Initial release
May 14, 2007	Added example configuration table for AD7877 on ADSP-BF548 Ez-Kit lite
Aug 21, 2007	Fixed command naming conventions, added two new PENIRQ events.

Table 1 – Revision History

1. Overview

This driver provides an effective and easy way to manage Analog Devices – AD7877 Touch screen controller. The driver can access AD7877 device registers or individual register fields via SPI. The driver is also capable of monitoring AD7877 interrupt signals (PENIRQ, DAV & ALERT) and take appropriate actions, which will highly reduce the application code overhead.

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/spi/adi_spi.h>
  - This file contains all definitions, function prototypes etc. specific to the SPI device
- <drivers/touchscreen/adi_ad7877.h>
  - This file contains all definitions, function prototypes etc. specific to the AD7877

2.2. Source Files

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

- < Blackfin/lib/src/drivers/touchscreen/adi_ad7877.c>
  - This file contains all the source code for the AD7877 device driver. All source code is written in 'C'. There are no assembly level functions in this driver.

3. Lower Level Drivers

AD7877 driver is layered on top of SPI driver

3.1. SPI Device Driver

AD7877 is controlled via SPI. All data transfers to and from the device is done via SPI. The device also has a chip-select pin (CS) to enable or disable the serial interface.

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, AD7877 device driver uses the System Services to perform the necessary low-level hardware access and control. This driver is built on top of Interrupt driven SPI driver. **Each AD7877 device requires two additional (driver) memory of size ADI_DEV_DEVICE_MEMORY, one for AD7877 and one for SPI.**

4.1. Interrupts

AD7877 is capable of generating three interrupt signals: Pen Interrupt (PENIRQ), Data Available (DAV) and Limit Comparison Alert (ALERT). These interrupt signals are usually mapped to Blackfin GPIO ports (flags). It is the applications responsibility to provide additional, secondary, interrupt memory to accommodate these flag interrupts. **This driver requires additional memory for three flag interrupts and one SPI interrupt, where each interrupt requiring memory of size ADI_INT_SECONDARY_MEMORY. One additional memory of ADI_INT_SECONDARY_MEMORY size must be provided when client decides to enable SPI error reporting.**

4.2. DMA

AD7877 driver doesn’t use or support DMA.

4.3. Timers

Timers are not used by this driver.

4.4. Real-Time Clock

Real-time clock is not used by this driver.

4.5. Programmable Flags

AD7877 is capable of generating three interrupt signals: Pen Interrupt (PENIRQ), Data Available (DAV) and Limit Comparison Alert (ALERT) and these interrupt signals are usually mapped to Blackfin GPIO ports (flags). The AD7877 driver can be configured to monitor any or all the above interrupt signals. It is the applications responsibility to initialise the flag service (using adi_flag_Init( )) with enough memory for flag callbacks. **This driver can monitor all three AD7877 interrupt signals and each flag connected to AD7877 interrupt requires memory of size ADI_FLAG_CALLBACK_MEMORY to manage its callback.**

4.6. Pins

Connect Blackfin SPI port pins to AD7877 SPI port.

5. Supported Features of the Device Driver

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

5.1. Directionality

ADI_DEV_DIRECTION	Description
ADI_DEV_ DIRECTION_BIDIRECTIONAL	Supports both the reception of data and transmission of data through the device via SPI. This directionality is fixed and cannot be changed by the application.

Table 2 – Supported Dataflow Directions

5.2. Dataflow Methods

ADI_DEV_MODE	Description
ADI_DEV_MODE_CHAINED	Supports the chained buffer method – used for AD7877 register access via SPI. This dataflow method is also fixed and cannot be changed by the application.

Table 3 – Supported Dataflow Methods

5.3. Buffer Types

AD7877 driver doesn't support buffer submission functions like adi_dev_Read(), adi_dev_Write and adi_dev_SequentialIO(). The application can use adi_dev_Control() function to access AD7877 device registers and register fields. Refer to section 10 for examples.

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.

Device Access Commands to access AD7877 registers

- ADI_DEV_CMD_REGISTER_READ
  - Reads a single device register (Refer to section 10.1.1 for example)
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics) (Refer section 9.1)
- ADI_DEV_CMD_REGISTER_FIELD_READ
  - Reads a specific field location in a single device register (Refer to section 10.1.2 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics) (Refer section 9.3)
- ADI_DEV_CMD_REGISTER_TABLE_READ
  - Reads a table of selective device registers (Refer to section 10.1.3 for example)
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics) (Refer section 9.1)
- ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ
  - Reads a table of selective device register fields (Refer to section 10.1.4 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics) (Refer section 9.3)
- ADI_DEV_CMD_REGISTER_BLOCK_READ
  - Reads a block of consecutive device registers (Refer to section 10.1.5 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics) (Refer section 9.2)
- ADI_DEV_CMD_REGISTER_WRITE
  - Writes to a single device register (Refer to section 10.2.1 for example)
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics) (Refer section 9.1)
- ADI_DEV_CMD_REGISTER_FIELD_WRITE
  - Writes to a specific field location in a single device register (Refer to section 10.2.2 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics) (Refer section 9.3)
- ADI_DEV_CMD_REGISTER_TABLE_WRITE
  - Writes to a table of selective device registers (Refer to section 10.2.3 for example)
  - Value – ADI_DEV_ACCESS_REGISTER * (register specifics) (Refer section 9.1)
- ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE
  - Writes to a table of selective device register fields (Refer to section 10.2.4 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_FIELD * (register specifics) (Refer section 9.3)
- ADI_DEV_CMD_REGISTER_BLOCK_WRITE
  - Writes to a block of consecutive device registers (Refer to section 10.2.5 for example)
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics) (Refer section 9.2)

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.

- ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER
  - Sets Blackfin SPI Device Number to be used to access AD7877 registers (Refer to section 10.3 for example)
  - Value – u8

- ADI_AD7877_CMD_SET_SPI_CS
  - Sets Blackfin SPI Slave select port used to select AD7877 device for SPI access (Refer to section 10.4 for example)
  - Value – u8
- ADI_AD7877_CMD_ENABLE_INTERRUPT_PENIRQ
  - Sets AD7877 driver to monitor PENIRQ interrupt signal (Refer to section 10.5.1 for example)
  - Value – ADI_AD7877_INTERRUPT_PORT* (Refer section 9.4)
- ADI_AD7877_CMD_ENABLE_INTERRUPT_DAV
  - Sets AD7877 driver to monitor DAV interrupt signal (Refer to section 10.5.2 for example)
  - Value – ADI_AD7877_INTERRUPT_PORT* (Refer section 9.4)
- ADI_AD7877_CMD_ENABLE_INTERRUPT_ALERT
  - Sets AD7877 driver to monitor ALERT interrupt signal (Refer to section 10.5.3 for example)
  - Value – ADI_AD7877_INTERRUPT_PORT* (Refer section 9.4)
- ADI_AD7877_CMD_DISABLE_INTERRUPT_PENIRQ
  - Remove PENIRQ interrupt from AD7877 driver interrupt monitoring list (Refer to section 10.6.1 for example)
  - Value – NULL
- ADI_AD7877_CMD_DISABLE_INTERRUPT_DAV
  - Remove DAV interrupt from AD7877 driver interrupt monitoring list (Refer to section 10.6.2 for example)
  - Value – NULL
- ADI_AD7877_CMD_DISABLE_INTERRUPT_ALERT
  - Remove ALERT interrupt from AD7877 driver interrupt monitoring list (Refer to section 10.6.3 for example)
  - 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 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

There are no common event IDs supported by this driver

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_AD7877_EVENT_PENIRQ
  - Indicates that PENIRQ interrupt has detected a screen touch event.
  - Value – NULL

- ADI_AD7877_EVENT_PENIRQ_RELEASE
  - Indicates that PENIRQ interrupt has detected a screen release event.
  - Value – NULL
- ADI_AD7877_EVENT_SINGLE_DAV
  - Occurs when AD7877 is configured in single channel mode and indicates that DAV interrupt has occurred
  - Value – pointer to a u16 location holding the result value of ADC channel selected for conversion
- ADI_AD7877_EVENT_SEQUENCER_DAV
  - Occurs when AD7877 is configured in Sequencer (slave or master) mode and indicates that DAV interrupt has occurred
  - Value – pointer to a structure of type ADI_AD7877_RESULT_REGS (Refer section 9.5) holding result value of selected register sequence
- ADI_AD7877_EVENT_ALERT
  - Indicates that ALERT interrupt has occurred
  - Value – pointer to location holding AD7877 Alert Status/Enable register value

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.

- ADI_AD7877_RESULT_CMD_NOT_SUPPORTED
  - Occurs when client issues a command which is not supported by this driver

6. Opening and 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.

- ADIAD7877EntryPoint

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. Any configuration settings not listed in the table below are undefined.

Item	Default Value	Possible Values	Command ID
AD7877 Control Register 1	0	Configuration dependent (0x000 to 0xFFF)	Use Device Access commands to configure Control Register 1
AD7877 Control Register 2	0	Configuration dependent (0x000 to 0xFFF)	Use Device Access commands to configure Control Register 2
AD7877 Alert Register	0	Configuration dependent (0x000 to 0xFFF)	Use Device Access commands to configure Alert Register
SPI Device Number	0	Processor dependent	ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER
SPI Chipselect	0	1 to 7	ADI_AD7877_CMD_SET_SPI_CS

Table 4 – 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
SPI Device Number	Processor dependent	ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER
SPI Chipselect	1 to 7	ADI_AD7877_CMD_SET_SPI_CS

Table 5 – Additional Required Settings

7. Hardware Considerations

The client should set the SPI device number to be used to for AD7877 before accessing its registers. Command id 'ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER' can be used to set SPI device number for AD7877.

The client should set the SPI chip-select value (configure SPI_FLG) corresponding to AD7877 before accessing the device registers. Command id 'ADI_AD7877_CMD_SET_SPI_CS' can be used to set AD7877's chip-select.

To enable interrupt monitoring, client should connect the corresponding AD7877 to a Blackfin port and configure AD7877 driver to monitor this signal. Commands ADI_AD7877_CMD_ENABLE_INTERRUPT_PENIRQ, ADI_AD7877_CMD_ENABLE_INTERRUPT_DAV and ADI_AD7877_CMD_ENABLE_INTERRUPT_ALERT can be used to enable monitoring PENIRQ, DAV and ALERT interrupts respectively.

7.1. AD7877 registers

The following table is a list of registers that can be accessed on the AD7877. These register and register field macros are defined in AD7877 driver header file (adi_ad7877.h). Application must use the corresponding register / register field macro to access the corresponding AD7877 device register.

Please refer to the AD7877 reference manual for a full description of registers and chip functionality.

7.1.1. AD7877 Device read/write type registers

Register Macro	Address	Default	Description
AD7877_CONTROL_REG1	0x01	-N/A-	Control Register 1
AD7877_CONTROL_REG2	0x02	-N/A-	Control Register 2
AD7877_ALERT_REG	0x03	-N/A-	Alert Status/Enable Register
AD7877_AUX1_HIGH_LIMIT	0x04	-N/A-	High Limit for Auxiliary Input 1
AD7877_AUX1_LOW_LIMIT	0x05	-N/A-	Low Limit for Auxiliary Input 1
AD7877_BAT1_HIGH_LIMIT	0x06	-N/A-	High Limit for Battery Monitoring Input 1
AD7877_BAT1_LOW_LIMIT	0x07	-N/A-	Low Limit for Battery Monitoring Input 1
AD7877_BAT2_HIGH_LIMIT	0x08	-N/A-	High Limit for Battery Monitoring Input 2
AD7877_BAT2_LOW_LIMIT	0x09	-N/A-	Low Limit for Battery Monitoring Input 2
AD7877_TEMP1_LOW_LIMIT	0x0A	-N/A-	Low Limit for Temperature Measurement
AD7877_TEMP1_HIGH_LIMIT	0x0B	-N/A-	High Limit for Temperature Measurement
AD7877_SEQUENCER_REG0	0x0C	-N/A-	Sequencer Register 0
AD7877_SEQUENCER_REG1	0x0D	-N/A-	Sequencer Register 1
AD7877_DAC_REG	0x0E	-N/A-	DAC Register

Table 6 – AD7877 device read/write type registers

7.1.2. AD7877 Device read-only type registers

Register Macro	Address	Default	Description
AD7877_YPOS	0x10	-N/A-	Y position measurement (X+ input)
AD7877_XPOS	0x11	-N/A-	X position measurement (Y+ input)

Register Macro	Address	Default	Description
AD7877_Z2	0x12	-N/A-	Z2 measurement (Y- with Y+ & X- excited)
AD7877_AUX1	0x13	-N/A-	Auxiliary 1 Input value
AD7877_AUX2	0x14	-N/A-	Auxiliary 2 Input value
AD7877_AUX3	0x15	-N/A-	Auxiliary 3 Input value
AD7877_BAT1	0x16	-N/A-	Battery Monitor Input 1 value
AD7877_BAT2	0x17	-N/A-	Battery Monitor Input 2 value
AD7877_TEMP1	0x18	-N/A-	single-ended Temperature measurement
AD7877_TEMP2	0x19	-N/A-	differential Temperature measurement
AD7877_Z1	0x1A	-N/A-	Z1 measurement (X- with Y+ & X- excited)

Table 7 – AD7877 device read-only type registers

7.1.3. AD7877 Device extended read/write type registers

Register Macro	Address	Default	Description
AD7877_CONTROL_REG1	0x01	-N/A-	Control Register 1
AD7877_CONTROL_REG2	0x02	-N/A-	Control Register 2
AD7877_ALERT_REG	0x03	-N/A-	Alert Status/Enable Register
AD7877_AUX1_HIGH_LIMIT	0x04	-N/A-	High Limit for Auxiliary Input 1
AD7877_AUX1_LOW_LIMIT	0x05	-N/A-	Low Limit for Auxiliary Input 1
AD7877_BAT1_HIGH_LIMIT	0x06	-N/A-	High Limit for Battery Monitoring Input 1
AD7877_BAT1_LOW_LIMIT	0x07	-N/A-	Low Limit for Battery Monitoring Input 1
AD7877_BAT2_HIGH_LIMIT	0x08	-N/A-	High Limit for Battery Monitoring Input 2
AD7877_BAT2_LOW_LIMIT	0x09	-N/A-	Low Limit for Battery Monitoring Input 2
AD7877_TEMP1_LOW_LIMIT	0x0A	-N/A-	Low Limit for Temperature Measurement
AD7877_TEMP1_HIGH_LIMIT	0x0B	-N/A-	High Limit for Temperature Measurement
AD7877_SEQUENCER_REG0	0x0C	-N/A-	Sequencer Register 0
AD7877_SEQUENCER_REG1	0x0D	-N/A-	Sequencer Register 1
AD7877_DAC_REG	0x0E	-N/A-	DAC Register

Table 8 – AD7877 device extended read/write type registers

7.2. AD7877 register fields

Register Field Macro	Position	**Length** (Bits)	Description
Control Register 1 (AD7877_CONTROL_REG1)			
AD7877_SER_DFR	11	1	Single-ended or Differential Conversion
AD7877_CHADD	7	4	ADC Channel Address
AD7877_RD	2	5	Register Read Address
AD7877_MODE	0	2	Mode code
Control Register 2 (AD7877_CONTROL_REG2)			

Register Field Macro	**Position**	**Length** (Bits)	**Description**
AD7877_AVG	10	2	ADC Averaging code
AD7877_ACQ	8	2	ADC Acquisition time
AD7877_PM	6	2	ADC Power Management Code
AD7877_FCD	4	2	First Conversion Delay
AD7877_POL	3	1	Polarity of signal on STOPACQ pin
AD7877_REF	2	1	Internal or external reference
AD7877_TMR	0	2	Conversion Interval Timer
Alert Status/Enable Register (AD7877_ALERT_REG)			
AD7877_TEMP1EN	11	1	Enable/Disable TEMP1 as ALERT interrupt source
AD7877_BAT2EN	10	1	Enable/Disable BAT2 as ALERT interrupt source
AD7877_BAT1EN	9	1	Enable/Disable BAT1 as ALERT interrupt source
AD7877_AUX1EN	8	1	Enable/Disable AUX1 as ALERT interrupt source
AD7877_TEMP1HI	7	1	TEMP1 channel below its high limit?
AD7877_BAT2HI	6	1	BAT2 channel above its high limit?
AD7877_BAT1HI	5	1	BAT1 channel above its high limit?
AD7877_AUX1HI	4	1	AUX1 channel above its high limit?
AD7877_TEMP1LO	3	1	TEMP1 channel above its low limit?
AD7877_BAT2LO	2	1	BAT2 channel below its low limit?
AD7877_BAT1LO	1	1	BAT1 channel below its low limit?
AD7877_AUX1LO	0	1	AUX1 channel below its low limit?
Sequencer Register 0 (AD7877_SEQUENCER_REG0) (Slave mode sequence) **Sequencer Register 1** (AD7877_SEQUENCER_REG1) (Master mode sequence)			
AD7877_YPOS_S	11	1	Enable Y Position Measurement
AD7877_XPOS_S	10	1	Enable X Position Measurement
AD7877_Z2_S	9	1	Enable Z2 Touch pressure Measurement
AD7877_AUX1_S	8	1	Enable Auxiliary Input 1 Measurement
AD7877_AUX2_S	7	1	Enable Auxiliary Input 2 Measurement
AD7877_AUX3_S	6	1	Enable Auxiliary Input 3 Measurement
AD7877_BAT1_S	5	1	Enable Battery Monitor Input 1 Measurement
AD7877_BAT2_S	4	1	Enable Battery Monitor Input 2 Measurement
AD7877_TEMP1_S	3	1	Enable Single ended conversion Temperature
AD7877_TEMP2_S	2	1	Enable Differential conversion Temperature
AD7877_Z1_S	1	1	Enable Z1 Touch pressure Measurement(bit1)
DAC Register (AD7877_DAC_REG)			
AD7877_DAC	4	8	DAC Data
AD7877_DAC_PD	3	1	DAC Power Down
AD7877_DAC_V_I	2	1	Voltage Output and Current Output
AD7877_DAC_RANGE	0	1	DAC Output range in Voltage mode
GPIO Control Register 1 (AD7877_GPIO_CONTROL_REG1)			

Register Field Macro	**Position**	**Length** (Bits)	**Description**
AD7877_GPIO2_EN	7	1	Select the function of AUX2 or GPIO2
AD7877_GPIO2_POL	6	1	Determine if GPIO2 is active high or low
AD7877_GPIO2_DIR	5	1	Set GPIO2 Direction
AD7877_GPIO2_ALEN	4	1	Enable/Disable ALERT interrupt on GPIO2
AD7877_GPIO1_EN	3	1	Select the function of AUX1 or GPIO1
AD7877_GPIO1_POL	2	1	Determine if GPIO1 is active high or low
AD7877_GPIO1_DIR	1	1	Set GPIO1 Direction
AD7877_GPIO1_ALEN	0	1	Enable/Disable ALERT interrupt on GPIO1
GPIO Control Register 2 (AD7877_GPIO_CONTROL_REG2)			
AD7877_GPIO4_EN	7	1	Select the function of AUX4 or GPIO4
AD7877_GPIO4_POL	6	1	Determine if GPIO4 is active high or low
AD7877_GPIO4_DIR	5	1	Set GPIO4 Direction
AD7877_GPIO4_ALEN	4	1	Enable/Disable ALERT interrupt on GPIO4
AD7877_GPIO3_EN	3	1	Select the function of AUX3 or GPIO3
AD7877_GPIO3_POL	2	1	Determine if GPIO3 is active high or low
AD7877_GPIO3_DIR	1	1	Set GPIO3 Direction
AD7877_GPIO3_ALEN	0	1	Enable/Disable ALERT interrupt on GPIO3
GPIO Data Register (AD7877_GPIO_DATA_REG)			
AD7877_GPIO1_DAT	7	1	GPIO1 Data bit
AD7877_GPIO2_DAT	6	1	GPIO2 Data bit
AD7877_GPIO3_DAT	5	1	GPIO3 Data bit
AD7877_GPIO4_DAT	4	1	GPIO4 Data bit

Table 9 – AD7877 Register Fields

8. Using AD7877 Driver in Applications

This section explains how to use AD7877 device driver in an application.

8.1. Device Manager Data memory allocation

This section explains device manager memory allocation requirements for applications using this driver. The application should allocate base memory + memory for one SPI device + memory for AD7877 device + memory for other devices used by the application

8.2. Interrupt Manager Data memory allocation

This section explains Interrupt manager memory allocation requirements for applications using this driver. The application should allocate secondary interrupt memory for three Blackfin flags monitoring AD7877 interrupts (PENIRQ, DAV & ALERT)

8.3. Flag Manager Data memory allocation

This section explains Flag manager memory allocation requirements for applications using this driver. The application should allocate memory for three Blackfin flags monitoring AD7877 interrupts (PENIRQ, DAV & ALERT).

Initialize Hardware (Ez-Kit), Interrupt manager, Deferred Callback Manager, Flag Manager, DMA Manager, Device Manager (all application dependent)

8.4. Typical usage of AD7877 driver

a. AD7877 (driver) initialization

Step 1: Open AD7877 Device driver with device specific entry point (refer section 6.1 for valid entry points)

Step 2: Set SPI device number to be used for AD7877 register access (refer section 10.3 for example)

Step 3: Set SPI Slave select number to be used for AD7877 register access (refer section 10.4 for example)

Step 4: Enable AD7877 interrupt monitoring (refer section 10.5 for examples)

b. Initializing and controlling AD7877 (hardware)

Step 5: Configure AD7877 device to specific operating mode using device access service (refer section 10.2 for examples)

Step 6: Respond to AD7877 callbacks

c. Terminating AD7877 driver

Step 7: Disable AD7877 interrupt monitoring (refer section 10.6 for examples)

Step 8: Terminate AD7877 driver with adi_dev_Terminate( )

Terminate DMA Manager, Deferred Callback, Flag Manager, DMA Manager, Device Manager (application dependent)

8.5. Configuring AD7877 on ADSP-BF548 Ez-Kit Lite

```
/* structure to hold Port Information for PENIRQ */
ADI_AD7877_INTERRUPT_PORT   PenIrqPort;
PenIrqPort.FlagId       = ADI_FLAG_PJ12;   /* PENIRQ is connected to BF548 Flag pin PJ12 */
PenIrqPort.FlagIntId    = ADI_INT_PINT2;   /* Interrupt ID forPort J */

/* structure to hold Port Information for DAV interrupt */
ADI_AD7877_INTERRUPT_PORT   DavIrqPort;
DavIrqPort.FlagId       = ADI_FLAG_PJ11;   /* DAV is connected to BF548 Flag pin PJ11 */
DavIrqPort.FlagIntId    = ADI_INT_PINT2;   /* Interrupt ID for Port J */

/* AD7877 Driver Configuration Table for ADSP-BF548 Ez-Kit Lite */
ADI_DEV_CMD_VALUE_PAIR  AD7877_BF548EzKit[ ]=
{
        { ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER,         (void *)0            }, /* SPI device to use */
        { ADI_AD7877_CMD_SET_SPI_CS,                    (void *)2            }, /* SPI CS for AD7877 */
        { ADI_AD7877_CMD_ENABLE_INTERRUPT_PENIRQ,       (void *)&PenIrqPort }, /* Enable PENIRQ monitoring */
        { ADI_AD7877_CMD_ENABLE_INTERRUPT_DAV,          (void *)&DavIrqPort },  /* Enable DAV Monitoring */
        { ADI_DEV_CMD_END,                              NULL                 }  /* Terminate this table */
};

/* Configure AD7877 Driver */
adi_dev_Control(AD7877DriverHandle, ADI_DEV_CMD_TABLE, (void *)AD7877_BF548EzKit);

/* Configuration table for AD7877 device registers */
ADI_DEV_ACCESS_REGISTER_FIELD     AD7877_MasterSequencer[ ] =
{
        { AD7877_CONTROL_REG1,      AD7877_MODE,      3 }, /* ADC in Master sequencer mode */
        { AD7877_CONTROL_REG1,      AD7877_SER_DFR, 0 }, /* Differential conversion */
        { AD7877_CONTROL_REG2,      AD7877_TMR,       3 }, /* convert every 8.19ms */
        { AD7877_CONTROL_REG2,      AD7877_REF,       1 }, /* External reference  */
        { AD7877_CONTROL_REG2,      AD7877_POL,       1 }, /* STOPACQ - Active High */
        { AD7877_CONTROL_REG2,      AD7877_FCD,       2 }, /* First Conversion Delay = 1.024ms */
        { AD7877_CONTROL_REG2,      AD7877_PM,        2 }, /* ADC & reference powered up continuously  */
        { AD7877_CONTROL_REG2,      AD7877_ACQ,       2 }, /* ADC acquisition time = 8us  */
        { AD7877_CONTROL_REG2,      AD7877_AVG,       1 }, /* 4 measurements per channel averaged */
        { AD7877_SEQUENCER_REG1,  AD7877_YPOS_S,    1 }, /* Enable Y position measurement */
        { AD7877_SEQUENCER_REG1,  AD7877_XPOS_S,    1 }, /* Enable X position measurement */
        { AD7877_SEQUENCER_REG1,  AD7877_Z2_S,      1 }, /* Enable Z2 touch pressure measurement */
        { AD7877_SEQUENCER_REG1,  AD7877_Z1_S,      1 }, /* Enable Z1 touch pressure measurement */
        { ADI_DEV_REGEND,           0,                0 }  /* Terminate this configuration table */
};

/* Configure AD7877 Device registers */
adi_dev_Control(AD7877DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE, (void *)AD7877_MasterSequencer);
```

8.6. Re-use/share the SPI device used by AD7877

On receiving register access commands, AD7877 driver opens the allocated SPI device, performs device read/write via SPI and closes the SPI device soon after completing the register access. So the application can reuse/share the same SPI device used by AD7877 device, without closing the AD7877 driver itself.

8.7. Resetting AD7877 device

For ADSP-BF548 Ez-Kit Lite: Use Hardware reset pushbutton.

9. Data Structures used by AD7877 driver

9.1. ADI_DEV_ACCESS_REGISTER

```
/* Data structure used for Single and Selective Register Access */
/* Data structure to access a single register */
/* Array structure to access a table of selective device registers */
typedef struct ADI_DEV_ACCESS_REGISTER
{
        u16     Address;        /* Device register address */
        u16     Data;           /* Data read/written from/to the register */
} ADI_DEV_ACCESS_REGISTER;
```

9.2. ADI_DEV_ACCESS_REGISTER_BLOCK

```
/* Data structure to access a block of consecutive registers */
typedef struct ADI_DEV_ACCESS_REGISTER_BLOCK
{
        u32     Count;          /* number of registers to be accessed */
        u16     Address;        /* starting address of register block */
        u16     *pData;  /* pointer to a 'Count' sized array of register data read/written from/to the device */
} ADI_DEV_ACCESS_REGISTER_BLOCK;
```

9.3. ADI_DEV_ACCESS_REGISTER_FIELD

```
/* Data structure to access individual register fields */
/* Basic element to access single register field */
/* Array structure to access a table of selective device register(s) field(s) */
typedef struct ADI_DEV_ACCESS_REGISTER_FIELD
{
        u16 Address;            /* Register address to access */
        u16 Field;              /* Register field in the above address to be accessed */
        u16 Data;               /* Register field data read/written from/to the device */
} ADI_DEV_ACCESS_REGISTER_FIELD;
```

9.4. ADI_AD7877_INTERRUPT_PORT

```
/* Structure to set AD7877 PENIRQ/DAV/ALERT Interrupt Flag connected to Blackfin */

/*
To enable AD7877 PENIRQ or DAV or ALERT interrupt, client must pass corresponding interrupt enable command with pointer to following structure as value. The 'FlagId' field should hold the Blackfin processor Flag ID connected to the selected Interrupt signal and 'FlagIntId' field should hold the Peripheral Interrupt ID of the corresponding flag
 */

typedef struct
{
        ADI_FLAG_ID                 FlagId;         /* Flag ID connected to AD7877 interrupt signal     */
        ADI_INT_PERIPHERAL_ID       FlagIntId;      /* Peripheral Interrupt ID of the corresponding flag*/
} ADI_AD7877_INTERRUPT_PORT;
```

9.5. ADI_AD7877_RESULT_REGS

```
/* AD7877 Result Register structure */
/* Pointer to this register structure will be passed as callback argument
   for Sequencer Slave and Master Mode read request, provided that the client
   configures the driver to handle the DAV interrupt */
typedef struct
{
        u16    Y;          /* holds Y position measurement (X+ input)  */
        u16    X;          /* holds X position measurement (Y+ input) */
        u16    Z2;         /* holds Z2 measurement (Y- input with Y+ & X- excited) */
        u16    Aux1;       /* holds Auxiliary 1 Input value */
        u16    Aux2;       /* holds Auxiliary 2 Input value */
        u16    Aux3;       /* holds Auxiliary 3 Input value */
        u16    Bat1;       /* holds Battery Monitor Input 1 value */
        u16    Bat2;       /* holds Battery Monitor Input 2 value*/
        u16    Temp1;      /* holds Temperature measurement using single-ended conversion */
        u16    Temp2;      /* holds Temperature measurement using a differential conversion*/
        u16    Z1;         /* holds Z1 measurement (X- input with Y+ & X- excited) */
} ADI_AD7877_RESULT_REGS;
```

10. Programming Examples

10.1. Reading AD7877 device registers

This section explains how to access the AD7877 device registers using device access commands.
Refer to section 7.1 for list of AD7877 device registers and section 7.2 for register fields

10.1.1. Read a single AD7877 device register

Command:	ADI_DEV_CMD_REGISTER_READ
Value:	ADI_DEV_ACCESS_REGISTER* (register specifics)

Example:

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

/* Load the register address to be read */
ReadReg.Address = AD7877_CONTROL_REG1;

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

10.1.2. Read a single AD7877 device register field

Command:	ADI_DEV_CMD_REGISTER_FIELD_READ
Value:	ADI_DEV_ACCESS_REGISTER_FIELD* (register specifics)

Example:

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

/* Load the device register address to be accessed */
ReadField.Address = AD7877_ALERT_REG;
/* Load the device register field location to be read */
ReadField.Address = AD7877_BAT1HI;

/* Application calls adi_dev_Control( ) function with corresponding command and value
   The register field value will be read back to location - ReadField.Data */
adi_dev_Control (DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_READ, (void *) &ReadField);
```

10.1.3. Read a table of AD7877 device registers

Command:	ADI_DEV_CMD_REGISTER_TABLE_READ
Value:	ADI_DEV_ACCESS_REGISTER* (register specifics)

Example:

```
/* define the structure to access table of device registers */
ADI_DEV_ACCESS_REGISTER ReadRegs [ ] =
        {       { AD7877_XPOS,          0 },
                { AD7877_YPOS,          0 },
                {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 ReadRegs array
i.e., value of AD7877_XPOS will be read to ReadRegs[0].Data and AD7877_YPOS to ReadRegs[1].Data
*/
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_TABLE_READ, (void *) &ReadRegs[0]);
```

10.1.4. Read a table of AD7877 device register(s) Fields

Command:	ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ
Value:	ADI_DEV_ACCESS_REGISTER_FIELD* (register specifics)

Example:

```
/* define the structure to access table of device register(s) fields */
ADI_DEV_ACCESS_REGISTER_FIELD ReadFields [ ] =
      {     { AD7877_GPIO_DATA_REG,     AD7877_GPIO1_DAT,     0 },
            { AD7877_ALERT_REG,         AD7877_AUX1HI,        0 },
            { AD7877_ALERT_REG,         AD7877_AUX1LO,        0 },
            { ADI_DEV_REGEND,           0,                    0 }                };

/*
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 ReadFields table
i.e., value of AD7877_GPIO1_DAT will be read to ReadFields[0].Data,  AD7877_AUX1HI to ReadFields [1].Data and
AD7877_AUX1LO to ReadFields [2].Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_TABLE_READ,
                        (void *) &ReadFields [0]);
```

10.1.5. Read a block of AD7877 device registers

Command:	ADI_DEV_CMD_REGISTER_BLOCK_READ
Value:	ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics)

Example:

```
/* define the structure to access a block of registers
ADI_DEV_ACCESS_REGISTER_BLOCK ReadBlock;

/* load the number of registers to be read */
ReadBlock.Count = 11;
/* load the starting address of the register block to be read */
ReadBlock.Address = AD7877_YPOS;
/* define a 'Count' sized array to hold register data read from the device */
u16 DataBlock[ReadBlock.Count] = { 0 };
/* load the start address of the above array to 'ReadBlock' data pointer */
ReadBlock.pData = &DataBlock[0];

/* Application calls adi_dev_Control( ) function with corresponding command and value
   Present value of the registers in the given block will be read to corresponding DataBlock[ ] array
   value of AD7877_YPOS will be read to DataBlock[0] and following 10 registers to remaining
   locations in DataBlock[] array */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_BLOCK_READ, (void *) &ReadBlock);
```

10.2. Configuring AD7877 device registers

10.2.1. Configure a single AD7877 register

Command:	ADI_DEV_CMD_REGISTER_WRITE
Value:	ADI_DEV_ACCESS_REGISTER* (register specifics)

Example:

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

/* Load the register address to be configured */
CfgReg.Address = AD7877_AUX1_HIGH_LIMIT;
/* Load the configuration value to CfgReg.Data location */
CfgReg.Data = 0xEF;

/* Application calls adi_dev_Control( ) function with corresponding command and value. The device register will be configured
with the value in CfgReg.Data */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_WRITE, (void *) &CfgReg);
```

10.2.2. Configure a single AD7877 register field

Command:	ADI_DEV_CMD_REGISTER_FIELD_WRITE
Value:	ADI_DEV_ACCESS_REGISTER_FIELD* (register specifics)

Example:

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

/* Load the device register address to be accessed */
CfgField.Address = AD7877_CONTROL_REG2;
/* Load the device register field location to be configured */
CfgField.Address = AD7877_TMR;
/* load the new field value */
CfgField.Data = 2;

/* Application calls adi_dev_Control( ) function with corresponding command and value
   Selected register field will be configured to the given value */
adi_dev_Control(DriverHandle, ADI_DEV_CMD_REGISTER_FIELD_WRITE, (void *) &CfgField);
```

10.2.3. Configure a table of AD7877 registers

Command:	ADI_DEV_CMD_REGISTER_TABLE_WRITE
Value:	ADI_DEV_ACCESS_REGISTER* (register specifics)

Example:

```
/* define the structure to configure table of device registers (register address, configuration value) */
ADI_DEV_ACCESS_REGISTER CfgRegs [ ] =
        {       { AD7877_CONTROL_REG2,          0x2C },
                { AD7877_AUX1_HIGH_LIMIT,       0xEF },
                {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 *) &CfgRegs [0]);
```

10.2.4. Configure a table of AD7877 register(s) fields

Command:	ADI_DEV_CMD_REGISTER_FIELD_TABLE_WRITE
Value:	ADI_DEV_ACCESS_REGISTER_FIELD* (register specifics)

Example:

```
/* 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 CfgFields [ ] =
        {       { AD7877_ALERT_REG,         AD7877_TEMP1EN,          1 },
                { AD7877_CONTROL_REG2,      AD7877_TMR,              2 },
                { AD7877_CONTROL_REG1,      AD7877_SER_DFR,          1 },
                {ADI_DEV_REGEND,            0,        0}      /* Register access 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_FIELD_TABLE_WRITE,
                        (void *) &CfgFields [0]);
```

10.2.5. Configure a block of AD7877 registers

Command:	ADI_DEV_CMD_REGISTER_BLOCK_WRITE
Value:	ADI_DEV_ACCESS_REGISTER_BLOCK* (register specifics)

Example:

```
/* define the structure to access a block of registers */
ADI_DEV_ACCESS_REGISTER_BLOCK CfgBlock;

/* load the number of registers to be configured */
CfgBlock.Count = 8;
/* load the starting address of the register block to be configured */
CfgBlock.Address = AD7877_AUX1_HIGH_LIMIT;

/* define a 'Count' sized array to hold register data read from the device
  load the array with AD7877 register configuration values */
u16 CfgData [CfgBlock.Count] = {0xEF, 0x0E, 0xF0, 0x20, 0xE9, 0x10, 0xF2, 0x38};

/* load the start address of the above array to CfgData data pointer */
CfgBlock.pData = &CfgData[0];

/* Application calls adi_dev_Control( ) function with corresponding command and value
   Registers in the given block will be configured with corresponding values in CfgData[ ] array */
adi_dev_Control (DriverHandle, ADI_DEV_CMD_REGISTER_BLOCK_WRITE, (void *) &CfgBlock);
```

10.3. Command to set AD7877 SPI Device Number

Command:	ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER
Value:	u8

Example:

```
/* Set AD7877 SPI  Device Number for ADSP-BF548 Ez-Kit Lite (SPI0SEL2) */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_SET_SPI_DEVICE_NUMBER, (void *)0 );
```

10.4. Command to set AD7877 SPI Chipselect

Command:	ADI_AD7877_CMD_SET_SPI_CS
Value:	u8

Example:

```
/* Set AD7877 SPI Chipselect for ADSP-BF548 Ez-Kit Lite (SPI0SEL2) */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_SET_SPI_CS, (void *) 2 );
```

10.5. Commands to enable AD7877 interrupt monitoring

10.5.1. Enable PENIRQ Monitoring

Command:	ADI_AD7877_CMD_ENABLE_INTERRUPT_PENIRQ
Value:	ADI_AD7877_INTERRUPT_PORT*

Example:

```
/* PENIRQ port info for ADSP-BF548 Ez-Kit Lite */
ADI_AD7877_INTERRUPT_PORT          PenIrqPort;
PenIrqPort.FlagId     = ADI_FLAG_PJ12;
PenIrqPort.FlagIntId = ADI_INT_PINT2;

/* Configure AD7877 driver to monitor PENIRQ interrupt */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_ENABLE_INTERRUPT_PENIRQ, (void *) &PenIrqPort );
```

10.5.2. Enable DAV Monitoring

Command:	ADI_AD7877_CMD_ENABLE_INTERRUPT_DAV
Value:	ADI_AD7877_INTERRUPT_PORT*

Example:

```
/* DAV port info for ADSP-BF548 Ez-Kit Lite */
ADI_AD7877_INTERRUPT_PORT     DavPort;
DavPort.FlagId     = ADI_FLAG_PJ11;
DavPort.FlagIntId = ADI_INT_PINT2;

/* Configure AD7877 driver to monitor DAV interrupt */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_ENABLE_INTERRUPT_DAV, (void *) &DavPort );
```

10.5.3. Enable ALERT Monitoring

Command:	ADI_AD7877_CMD_ENABLE_INTERRUPT_ALERT
Value:	ADI_AD7877_INTERRUPT_PORT*

Example:

```
/* ALERT port info (ALERT signal not connected on ADSP-BF548 Ez-Kit Lite) */
ADI_AD7877_INTERRUPT_PORT     AlertPort;
AlertPort.FlagId     = ADI_FLAG_PJ11;
AlertPort.FlagIntId = ADI_INT_PINT2;

/* Configure AD7877 driver to monitor ALERT interrupt */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_ENABLE_INTERRUPT_ALERT, (void *) &AlertPort );
```

10.6. Commands to Disable AD7877 interrupt monitoring

10.6.1. Disable PENIRQ Monitoring

Command:	ADI_AD7877_CMD_DISABLE_INTERRUPT_PENIRQ
Value:	NULL

Example:

```
  /* Remove PENIRQ from monitoring */
  adi_dev_Control (DriverHandle, ADI_AD7877_CMD_DISABLE_INTERRUPT_PENIRQ, (void *) NULL);
```

10.6.2. Disable DAV Monitoring

Command:	ADI_AD7877_CMD_DISABLE_INTERRUPT_DAV
Value:	NULL

Example:

```
  /* Remove DAV from monitoring */
  adi_dev_Control (DriverHandle, ADI_AD7877_CMD_DISABLE_INTERRUPT_DAV, (void *) NULL);
```

10.6.3. Disable ALERT Monitoring

Command:	ADI_AD7877_CMD_DISABLE_INTERRUPT_ALERT
Value:	NULL

Example:

```
  /* Remove ALERT from monitoring */
adi_dev_Control (DriverHandle, ADI_AD7877_CMD_DISABLE_INTERRUPT_ALERT, (void *) NULL);
```

11. AD7877 Driver Flow

This section explains the driver flow for all possible AD7877operating modes

11.1. Reading AD7877 registers

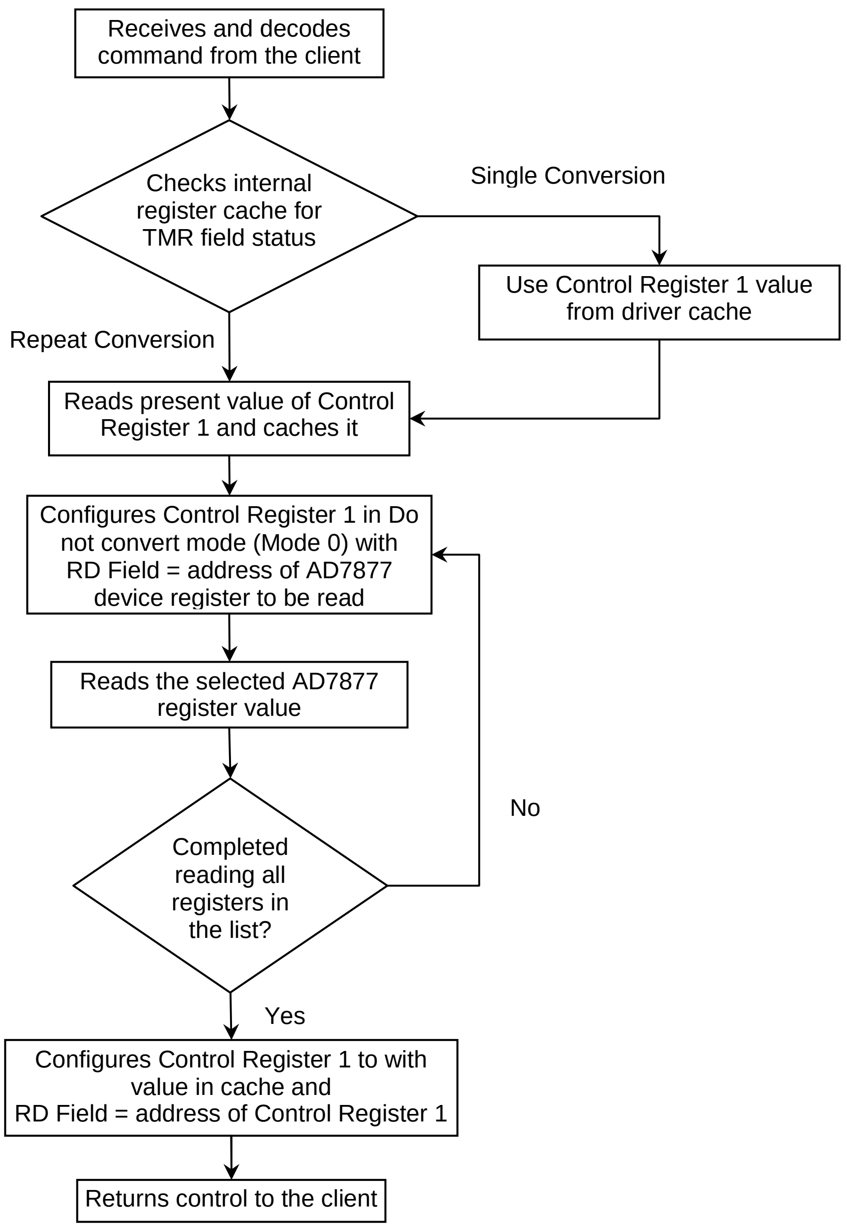
Receives and decodes command from the client
Checks internal register cache for TMR field status
Single Conversion
Use Control Register 1 value from driver cache
Repeat Conversion
Reads present value of Control Register 1 and caches it
Configures Control Register 1 in Do not convert mode (Mode 0) with RD Field = address of AD7877 device register to be read
Reads the selected AD7877 register value
Completed reading all registers in the list?
No
Yes
Configures Control Register 1 to with value in cache and RD Field = address of Control Register 1
Returns control to the client

Figure 1: Reading AD7877 registers

11.2. Configuring AD7877 registers

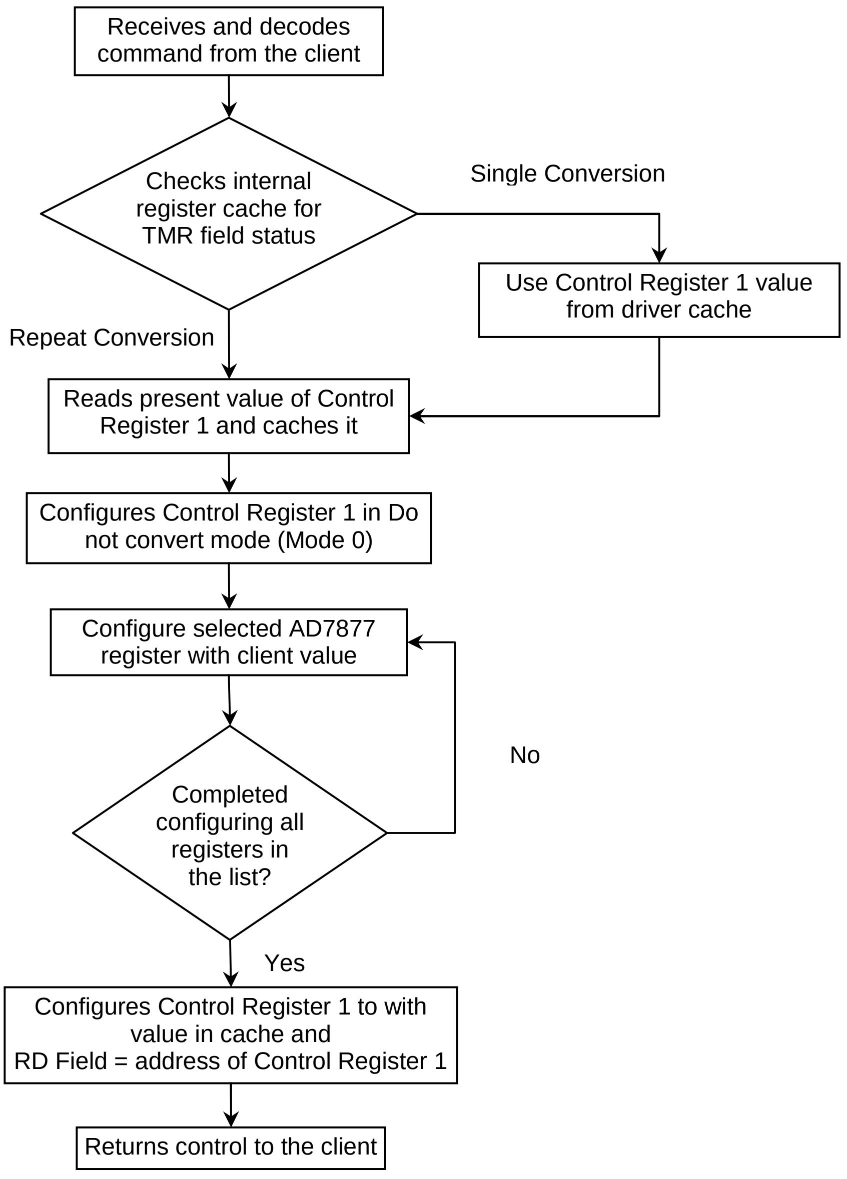
Receives and decodes command from the client
Checks internal register cache for TMR field status
Single Conversion
Use Control Register 1 value from driver cache
Repeat Conversion
Reads present value of Control Register 1 and caches it
Configures Control Register 1 in Do not convert mode (Mode 0)
Configure selected AD7877 register with client value
No
Completed configuring all registers in the list?
Yes
Configures Control Register 1 to with value in cache and RD Field = address of Control Register 1
Returns control to the client

Figure 2: Reading AD7877 registers

11.3. Monitoring PENIRQ Interrupt

Consider that the client has configured AD7877 driver to monitor PENIRQ interrupt.

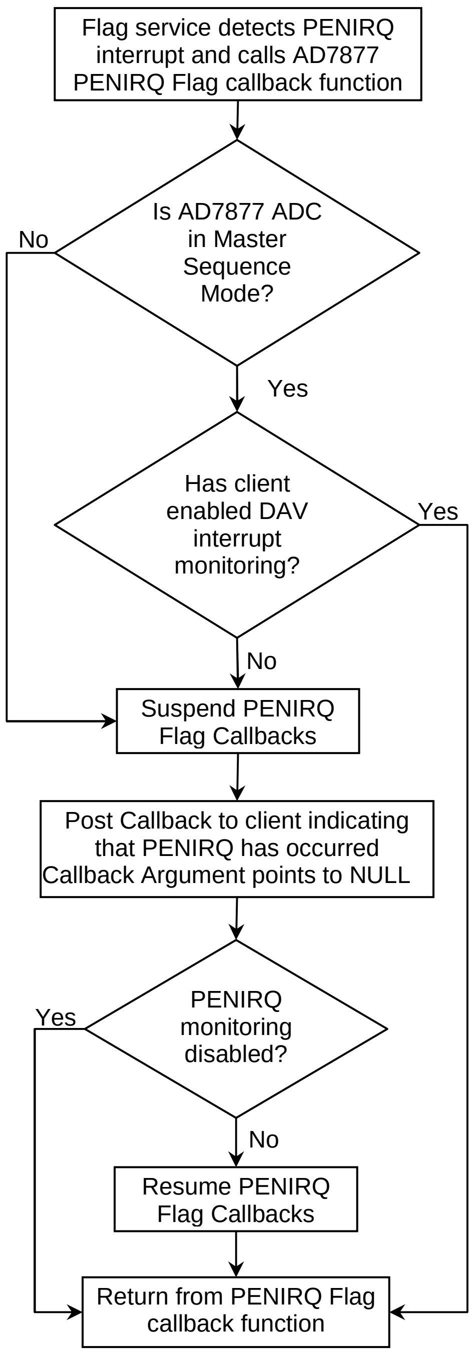
Flag service detects PENIRQ interrupt and calls AD7877 PENIRQ Flag callback function
Is AD7877 ADC in Master Sequence Mode?
No
Yes
Has client enabled DAV interrupt monitoring?
Yes
No
Suspend PENIRQ Flag Callbacks
Post Callback to client indicating that PENIRQ has occurred Callback Argument points to NULL
PENIRQ monitoring disabled?
Yes
No
Resume PENIRQ Flag Callbacks
Return from PENIRQ Flag callback function

Figure 3: Monitoring PENIRQ interrupt

11.4. Monitoring DAV interrupt

Consider that the client has configured AD7877 driver to monitor DAV interrupt.

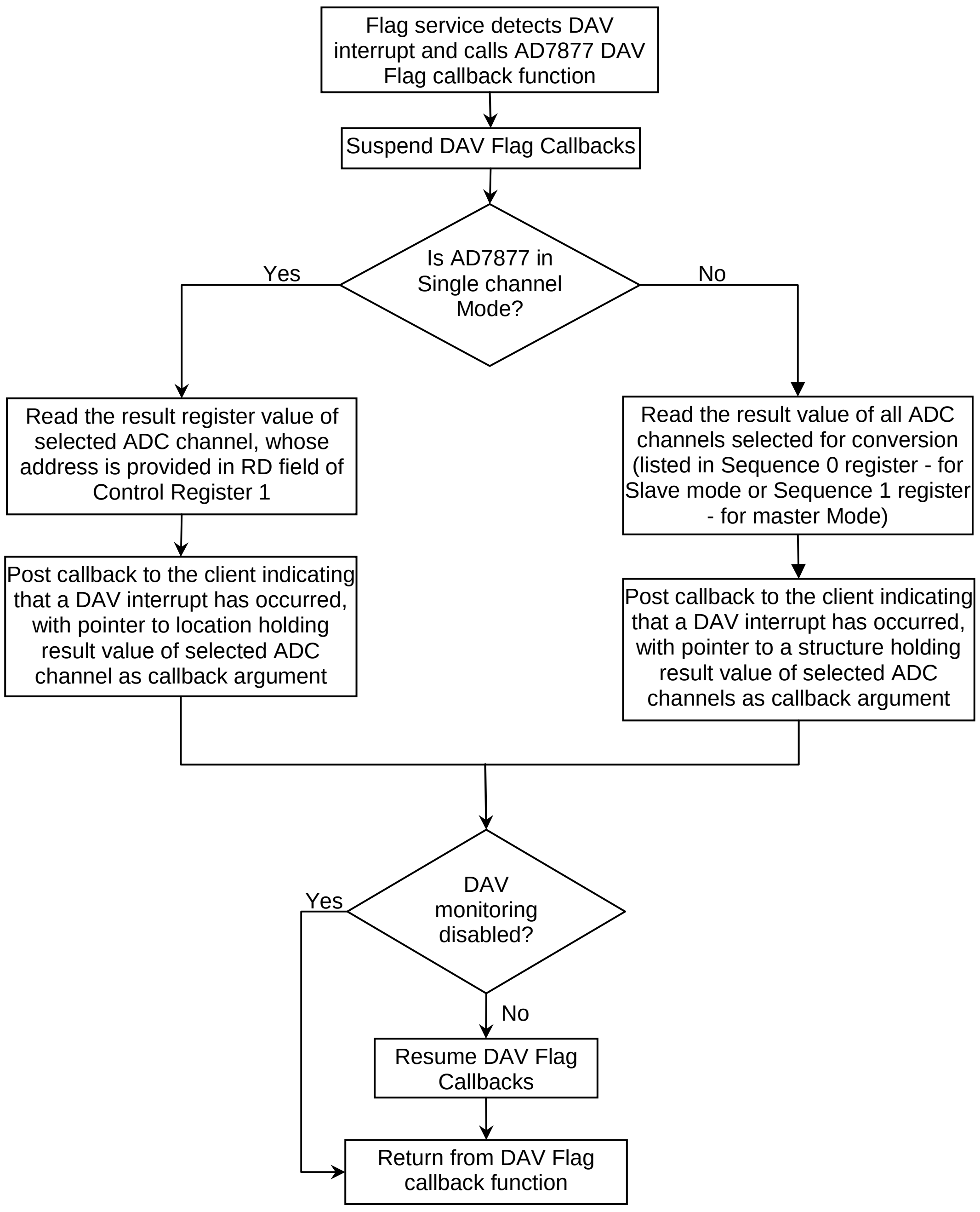
Flag service detects DAV interrupt and calls AD7877 DAV Flag callback function
Suspend DAV Flag Callbacks
Is AD7877 in Single channel Mode?
Yes
No
Read the result register value of selected ADC channel, whose address is provided in RD field of Control Register 1
Read the result value of all ADC channels selected for conversion (listed in Sequence 0 register - for Slave mode or Sequence 1 register - for master Mode)
Post callback to the client indicating that a DAV interrupt has occurred, with pointer to location holding result value of selected ADC channel as callback argument
Post callback to the client indicating that a DAV interrupt has occurred, with pointer to a structure holding result value of selected ADC channels as callback argument
DAV monitoring disabled?
Yes
No
Resume DAV Flag Callbacks
Return from DAV Flag callback function

Figure 4: Monitoring DAV interrupt

11.5. Monitoring ALERT interrupt

Consider that the client has configured AD7877 driver to monitor ALERT interrupt.

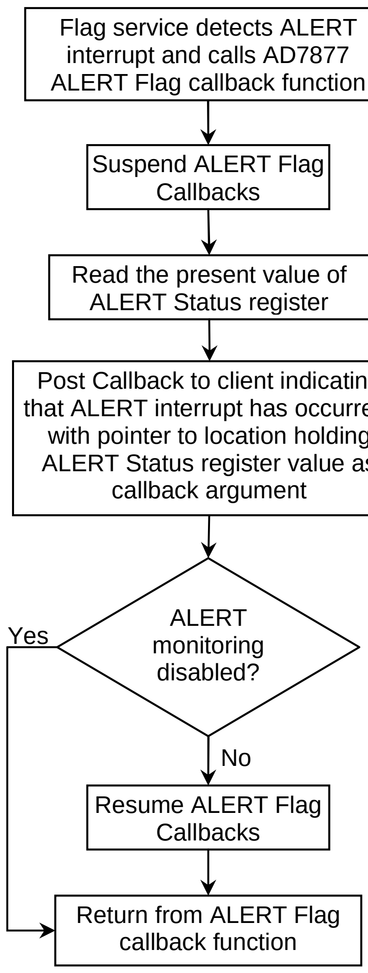
Flag service detects ALERT interrupt and calls AD7877 ALERT Flag callback function
Suspend ALERT Flag Callbacks
Read the present value of ALERT Status register
Post Callback to client indicating that ALERT interrupt has occurred, with pointer to location holding ALERT Status register value as callback argument
ALERT monitoring disabled?
Yes
No
Resume ALERT Flag Callbacks
Return from ALERT Flag callback function

Figure 5: Monitoring ALERT interrupt

12. References

1. Analog Devices AD7877 Touch Screen Controller DataSheet, Rev B, July 2006
   http://www.analog.com/UploadedFiles/Data_Sheets/AD7877.pdf