ADP5520
(KEYPAD CONTROLLER)
DEVICE DRIVER

DATE: 2009-11-24

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
2009-11-24	Initial release.

Table 1 – Revision History

1. Overview

This driver gives an application access to the keypad features of the *ADP5520 Backlight Driver with I/O Expander* multi-function controller. The ADP5520 is used on the ADSP-BF527 EZ-KIT Lite Revision 2.1 and later to give access to the 16-key keypad mounted on the board.

The driver gives access to all the ADP5520's configuration and status registers and also provides a single command that sets up the keypad controller function in a predetermined configuration suitable for its use on the EZ-KIT.

The ADP5520 driver uses the System Services Library's Device Access service to perform the low-level accesses to the actual device via TWI.

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/keypad/adi_adp5520.h>
  This file includes many useful definitions for the ADP5520's internal registers, definitions of the device-specific driver commands, events and result codes, and a declaration of the driver's entry-point.

2.2. Source Files

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

- <Blackfin/lib/src/drivers/keypad/adi_adp5520.c>
  This is the source file that implements the ADP5520 device driver.

3. Lower Level Drivers

3.1. Device Access Service

The Device Access Service is part of the System Services Library and provides a standardised way for device drivers to access the control and status registers of devices connected to the processor via SPI or TWI. Use of the Device Access Service is largely hidden from clients of the ADP5520 driver, except for the need to provide TWI addressing information if the driver-supplied defaults are inappropriate for the target platform.

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.

4.1. Interrupts

The driver makes no direct use of interrupts. The ADP5520 has one interrupt line (nINT) which the driver expects to be connected to one of the processor's programmable flags. The driver makes use of the Programmable Flag Service in order to be informed of ADP5520-generated interrupts (see 4.5).

4.2. DMA

The driver makes no use of DMA.

4.3. Timers

The driver uses no timers.

4.4. Real-Time Clock

The driver does not use the Real-Time Clock.

4.5. Programmable Flags

The ADP5520 driver requires to be configured with the identity of the programmable flag to which the ADP5520's interrupt line (nINT) is connected. The driver uses the Programmable Flag service to configure the flag and installs one call back for it.

4.6. Pins

The driver makes no direct use of any processor pins other than the one programmable flag already mentioned. The Device Access service that the ADP5520 driver uses to communicate with the ADP5520 device uses the processor's TWI controller and the external pins that it requires. See the Device Access documentation for further details.

5. Supported Features of the Device Driver

This section describes what features are supported by the device driver. Since the driver is totally event driven it does not support any of the dataflow concepts of directionality, dataflow methods or buffer types.

5.1. Directionality

There is no dataflow between keypad hardware and the processor. The driver does not support dataflow directionality.

5.2. Dataflow Methods

This driver does not support any dataflow methods.

5.3. Buffer Types

This driver does not support any buffer types.

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_CMD_GET_PERIPHERAL_DMA_SUPPORT
  - Determines if the device driver is supported by peripheral DMA
  - Value – u32 * (location where 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_BLOCK_READ
  - Reads a block of consecutive device registers
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (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_REGISTER_BLOCK_WRITE
  - Writes to a block of consecutive device registers
  - Value – ADI_DEV_ACCESS_REGISTER_BLOCK * (register specifics)

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_ADP5520_CMD_SET_TWI_DEVICE_NUMBER
  - Sets TWI Device Number to use to access ADP5520 registers. Type is u32 and default value is 0.
  - Value – TWI Device number to use.

- ADI_ADP5520_CMD_SET_TWI_CONFIG_TABLE
  - Supplies TWI Configuration table specific to the application. Default is a minimal set of commands that sets the dataflow method to ADI_DEV_MODE_SEQ_CHAINED and then enables dataflow.
  - Value – pointer to ADI_DEV_CMD_VALUE_PAIR.

- ADI_ADP5520_CMD_SET_TWI_DEVICE_ADDRESS
    - Supplies TWI Device Address specific to the HW platform. Type is u32 and default value is ADP5520_TWI_DEVICE_ADDRESS (defined in the header file).
    - Value – TWI Device address to use.

- ADI_ADP5520_CMD_ENABLE_INTERRUPTS
    - Specifies which ADP5520 interrupts are to be enabled and which processor flag receives the ADP5520's nINT signals. The ADP5520 interrupt enable bits are set first and then the processor flag is opened and configured and then its processor interrupt is enabled.
    - Value – Pointer to an ADI_ADP5520_INTERRUPT_INFO structure containing the interrupt information.

- ADI_ADP5520_CMD_DISABLE_INTERRUPTS
    - Specifies which, if any, ADP5520 interrupts are to be disabled and whether the processor flag interrupt is to be disabled and the flag closed.
    - Value – Pointer to an ADI_ADP5520_INTERRUPT_INFO structure containing the interrupt information. '1' bits in the 'interruptMask' field represent ADP5520 interrupts that are to be disabled. A value of ADI_FLAG_UNDEFINED in the 'flagId' field means that processor interrupts from the ADP5520 are to be disabled and the corresponding processor flag closed. Other values are ignored and processor interrupts remain enabled.

- ADI_ADP5520_CMD_BLOCK_INTERRUPT
    - Blocks the processor interrupt associated with nINT by clearing the appropriate system interrupt controller enable bit.
    - Value – none.

- ADI_ADP5520_CMD_UNBLOCK_INTERRUPT
    - Unblocks the processor interrupt associated with nINT.
    - Value – none.

- ADI_ADP5520_CMD_ENABLE_KEYPAD
    - Initialises the ADP5520’s keypad scanner in a standard configuration (suitable for the ADSP-BF527 EZ-KIT Lite), enables the scanner, and starts delivering key press and/or release events to the client’s call-back function.
    - Value – ADI_ADP5520_WANT_KEY_PRESSES, ADI_ADP5520_WANT_KEY_RELEASES or these two values 'ORed' together.

- ADI_ADP5520_CMD_DISABLE_KEYPAD
    - Disables the ADP5520’s keypad scanner and stops delivering ket press and/or release events to the client.
    - Value – none.

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

This driver does not emit any of the common data transfer events.

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_ADP5520_EVENT_KEYPAD_PRESS
  - One or more of the keypad keys has raised a 'pressed' interrupt.
  - Value – The event argument is a u32 value holding a 16-bit mask in bits 15:0. Each 1-bit represents a key that has raised a 'pressed' interrupt:

Bit	Key	Row	Col
15	'A'	R0	C0
14	'B'	R0	C1
13	'C'	R0	C2
...			
1	'O'	R3	C2
0	'P'	R3	C3

- ADI_ADP5520_EVENT_KEYPAD_RELEASE
  - One or more of the keypad keys has raised a 'released' interrupt.
  - Value – As for ADI_ADP5520_EVENT_KEYPAD_PRESS except that each 1-bit represents a key that has raised a ‘released’ interrupt.

- ADI_ADP5520_EVENT_LIGHT_SENSOR_TRIGGER
  - The ambient light sensor comparators have signalled a change.
  - Value – The event argument is a u32 value holding the current values of the L3_OUT and L2_OUT status bits from the ADP5520's ALS_CMPR_CFG register. The status bit values occupy positions ADP5520_POS_L3_OUT and ADP5520_POS_L2_OUT respectively in the event argument.

- ADI_ADP5520_EVENT_OVERVOLTAGE_TRIGGER
  - The overvoltage protection circuit has raised an interrupt.
  - Value – There is no associated event argument.

- ADI_ADP5520_EVENT_GPIO_INPUT
  - One or more GPIO input pins has raised an interrupt.
  - Value – The event argument is a u32 item holding the current value of the ADP5520's GPIO_INT_STAT register in bits 7-0.

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_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.

- ADI_ADP5520_RESULT_CMD_NOT_SUPPORTED
  - Returned when client issues a command that is not supported by this driver.

- ADI_ADP5520_RESULT_FEATURE_CONFLICT
  - The requested feature conflicts with an already-enabled feature. Due to multiplexing within the ADP5520, not all features can be active simultaneously.

- ADI_ADP5520_RESULT_INVALID_KEY_REQUIREMENT
  - Notification of neither keypad presses nor releases was requested when enabling the keypad.

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.

- ADIADP5520EntryPoint

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
TWI device number	0	Identifier of TWI controller to which ADP5520 is connected.	ADI_ADP5520_CMD_SET_TWI_DEVICE_NUMBER
TWI device address	0x32u	TWI address to which ADP5520 responds.	ADI_ADP5520_CMD_SET_TWI_DEVICE_ADDRESS
Rows x Columns	4 x 4	See ADP5520 data-sheet.	ADI_DEV_CMD_REGISTER_*_WRITE to configure keypad scannner manually

Table 2 – 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
nINT -> FLAG connectivity	ADI_ADP5520_INTERRUPT_INFO struct containing FLAG information	ADI_ADP5520_CMD_ENABLE_INTERRUPTS

Table 3 – Additional Required Settings

7. Hardware Considerations

There are no special hardware considerations relating to this driver.