ANALOG DEVICES

ADI_UART_DMA DEVICE DRIVER

DATE: MARCH 23, 2007

Table of Contents

Table of Figures

Document Revision History

Date	Description of Changes
2007-03-23	Initial version
2007-04-20	Corrected setting of even/odd parity
2008-12-22	Added command to set transfer mode

Table 1 - Revision History

1. Overview

This document describes the usage of UART device driver. As written, the driver has no hardware dependencies and has been tested on the ADSP-BF533, ADSP-BF537 and ADSP-BF561 EZ-Kit Lite development boards.

The UART device driver supports both DMA driven and interrupts driven versions.

Two source files adi_uart_dma.c and adi_uart_int.c are provided to select DMA driven or interrupt driven version of the UART device driver. Using adi_uart.c file by itself in a project will build interrupt driven version of UART driver by default.

This document only describes the DMA driven UART driver.
To select DMA-driven version of the UART device driver, the application code shall

- Include adi_uart_dma.c in the project files.
- Use DMA entry point “ADIUARTDmaEntryPoint” to open the SPI device driver.

The DMA driven UART device driver is common for all legacy-type UART device 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/uart/adi_uart.h>
  - This file contains all commands, event and return codes specific to the UART device driver.

2.2. Source Files

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

- <Blackfin/lib/src/drivers/uart/adi_uart_dma.c>
  - This file defines a macro (ADI_UART_DMA) and includes all source code from the mother file (adi_uart.c). The macro directs the compiler to build a DMA driven version of UART driver. All source code is written in ‘C’. There are no assembly level functions in the driver.

3. Lower Level Drivers

The UART device driver does not use any lower level device drivers.

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()).

4.1. Interrupts

Unless overridden with the appropriate commands, the UART device driver uses the default, power-up, interrupt settings, Interrupt Vector Group (IVG) mappings etc. . Each UART device that is opened requires 3 interrupt resources if opened in unidirectional mode and 5 interrupt resources if opened in bidirectional mode. The UART device driver hooks interrupt handlers as follows:

- Transmit (Tx) Interrupt – This interrupt is hooked when the UART is opened for outbound or bidirectional data flow. The interrupt is hooked when the UART is opened (adi_dev_Open()). Any changes to the IVG mapping for the UART transmit interrupt should be done before the UART is opened. The interrupt is unhooked when the device is closed (adi_dev_close()).
- Receive (Rx) Interrupt – This interrupt is hooked when the UART is opened for inbound or bidirectional data flow. The interrupt is hooked when the UART is opened (adi_dev_Open()). Any changes to the IVG mapping for the UART receive interrupt should be done before the UART is opened. The interrupt is unhooked when the device is closed (adi_dev_Close()).
- Error/Line Status Interrupt – This interrupt is hooked/unhooked when the client passes the ADI_UART_CMD_SET_LINE_STATUS_EVENTS command to the driver. By default, line status interrupts are not generated so if line status interrupts are desired they must be explicitly turned on with the command. When the adi_dev_Control() function receives the command with an accompanying argument of TRUE, the line status interrupts are enabled and the interrupt handler is hooked into the IVG chain. When the adi_dev_Control() function receives the command with an accompanying argument of FALSE, the line status interrupts are disabled and the interrupt handler is unhooked from the IVG chain. If enabled and hooked, line status interrupts are disabled and the interrupt handler is automatically unhooked when the driver is closed (adi_dev_Close()).
- Timer Interrupt – Used only during Autobaud detection and only via the Timer Control service. See Timer section below for additional information.

4.2. DMA

The DMA-driven UART device driver uses two DMA channels, one to transmit and one to receive.

4.3. Timers

The UART device driver uses timer resources only during autobaud detection. If autobaud capability is not used then no timer resources are used. If autobaud capability is used, the appropriate timer resource is used as described below.

When passed the ADI_DEV_CMD_AUTOBAUD command, the UART device driver uses the Timer Control service to access timer resources for the timer to which the specific UART device is associated. For example on the ADSP-

BF533 processor, Timer 1 is used by the UART for autobaud detection. The selection of which timer is used for autobaud detection is fixed by the processor and cannot be changed. In response to the command, the driver resets the timer, installs a callback on the timer and configures the timer as appropriate for autobaud detection. The timer is used until the autobaud detection process completes, at which time the timer is disabled, the callback is removed and the timer can be re-used by the client or any other component.

4.4. Real-Time Clock

The UART device driver does not use any real-time clock services.

4.5. Programmable Flags

The UART device driver does not explicitly use any programmable flag pins. However, on most processors the UART input and output pins are muxed with programmable flag pins. The user should use caution to insure that pins used by the UART do not interfere with any general purpose I/O use and vice-versa.

4.6. Pins

The UART makes use of two pins, receive and transmit, plus ground. On processors where pin muxing is used, the UART device driver uses the Port Control service to automatically configure pins for use by the UART as described below:

- Bidirectional data flow – Both receive and transmit pins are configured for UART use.
- Inbound only data flow – Only the receive pin is configured for UART use.
- Outbound only data flow – Only the transmit pin is configured for UART use.

The pins are configured when the UART is opened (adi_dev_Open()).

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 an 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_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
- ADI_DEV_CMD_GET_2D_SUPPORT
  - Determines if the driver can support 2D buffers. The UART driver always returns FALSE to this command.
  - Value – u32 * (location where the response (TRUE/FALSE) is stored)
- ADI_DEV_CMD_SET_ERROR_REPORTING
  - Enables/Disables error reporting from the device driver. This command is used to enable or disable the generation of the line status events; overrun, parity and framing errors and the break interrupt.
  - Value – TRUE/FALSE
- ADI_DEV_CMD_GET_PERIPHERAL_DMA_SUPPORT
  - Determines if the device driver is supported by peripheral DMA. The DMA-driven UART driver always returns TRUE to this command.
  - Value – u32 * (location where the response (TRUE/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.

- ADI_UART_CMD_SET_DATA_BITS
  - Sets the number of data bits in the word. The value passed is the number of bits. Acceptable values are 5, 6, 7 or 8.
  - Value – u16 (number of bits).
- ADI_UART_CMD_SET_STOP_BITS
  - Sets the number of stop bits. This command sets the number of stop bits. A value of 1 sets one stop bit, a value of 2 sets two stop bits for non-5 bit word lengths and 1 ½ stop bits for a 5 bit word length.
  - Value – u16 (number of stop bits)

- ADI_UART_CMD_SET_PARITY
  - Sets even or odd parity. Note that this command simply sets the parity mode, it does not enable parity checking. See ADI_UART_CMD_ENABLE_PARITY.
  - Value – even number for even parity, odd number for odd parity
- ADI_UART_CMD_ENABLE_PARITY
  - Enables parity checking. This command turns on/off parity transmission and checking. Line status events must also be turned on for reporting of parity events.
  - Value – TRUE turns parity checking on, FALSE turns parity checking off
- ADI_UART_CMD_SET_LINE_STATUS_EVENTS
  - Enables/disables the generation of line status events to the client. When line status event reporting is turned on, the error interrupt for the UART is hooked into IVG chain. When a line status event is detected, the client is notified via the callback function whenever an event occurs. When line status event reporting is turned off, the error interrupt for the UART is unhooked from the IVG chain and line status event checking and reporting is disabled.
  - Value – TRUE turns on line status event reporting, FALSE turns off line status event reporting
- ADI_UART_CMD_SET_BAUD_RATE
  - Sets the baud rate of the UART. This command causes the driver to update the UART's frequency control registers, based on the current SCLK value to obtain the specified baud rate.
  - Value – u32 (baud rate in hertz)
- ADI_UART_CMD_GET_TEMT
  - Senses the value of the TEMT bit. This command is used to sense when the TSR and THR registers are both empty. It is primarily used to determine when all pending transmit characters have been shifted out so that the driver can be closed.
  - Value – u32 * (location where the value of the TEMT bit is stored)
- ADI_UART_CMD_AUTOBAUD
  - Automatically sense the baud rate of the serial line. This command causes the UART driver to automatically sense the baud rate of the serial line and configure the UART accordingly. The UART driver scans the serial line for the autobaud character, using that character to determine the line speed.
  - Value – NULL
- ADI_UART_CMD_SET_AUTOBAUD_CHAR
  - Sets the character for autobaud detection. This command specifies the character the driver should expect for autobaud detection.
  - Value – u8 (ASCII character)
- ADI_UART_CMD_SET_DIVISOR_BITS
  - Alternative method to specifying the autobaud character. This command specifies the number divisor bits in the autobaud calculation. This command can be used in place of the ADI_UART_CMD_SET_AUTOBAUD_CHAR in instances where specifying a specific character for autobaud detection may not be appropriate.
  - Value – u16 (number of bits)
- ADI_UART_CMD_ENABLE_CTS_RTS
  - Enables or disables the CTS/RTS hardware flow control logic for the UART. (NOTE: only UART devices that provide CTS/RTS hardware flow control support this command.)
  - Value – TRUE enables CTS/RTS flow control, FALSE disables CTS/RTS flow control
- ADI_UART_CMD_SET_CTS_RTS_POLARITY
  - Sets active low or active high polarity for the CTS/RTS signals. By default these signals are active low. (NOTE: only UART devices that provide CTS/RTS hardware flow control support this command.)
  - Value – TRUE sets active high polarity, FALSE sets active low polarity
- ADI_UART_CMD_SET_CTS_RTS_THRESHOLD
  - Sets the threshold for CTS/RTS assertion either low (FIFOs half full) or high (FIFOs completely full). By default the threshold is set to low. (NOTE: only UART devices that provide CTS/RTS hardware flow control support this command.)
  - Value – TRUE sets the high threshold, FALSE sets the low threshold

- ADI_UART_CMD_SET_TRANSFER_MODE
    - Allows half duplex operation. One direction may be disabled to provide better control of UART data transfer. (NOTE: This command is valid only in bi-directional dataflow mode)
    - Value – 0=Full Duplex (Tx and Rx); 1=Half Duplex (Tx Only); 2=Half Duplex (Rx Only); 3=Disable both Tx and Rx

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 buffer has been processed by the device driver.
    - Value – 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.

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_UART_EVENT_BREAK_INTERRUPT
    - The driver detected the break interrupt. Line status event reporting must be turned on for this event to be detected.
    - Value – NULL
- ADI_UART_EVENT_FRAMING_ERROR
    - The driver detected a framing error. Line status event reporting must be turned on for this event to be detected.
    - Value – NULL
- ADI_UART_EVENT_PARITY_ERROR
    - The driver detected a parity error. Line status event reporting must be turned on for this event to be detected.
    - Value – NULL
- ADI_UART_EVENT_OVERRUN_ERROR
    - The driver detected an overrun error. Line status event reporting must be turned on for this event to be detected.
    - Value – NULL
- ADI_UART_EVENT_AUTOBAUD_COMPLETE

- The autobaud detection function has completed.
- Value – u32. The value parameter contains the baud rate that was detected.

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. Wherever functions in the device driver API are called, the client should be prepared to process any of these return codes.

Typically the client 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.
- ADI_DEV_RESULT_DATAFLOW_NOT_ENABLED
  - In sync mode, buffers provided before dataflow enabled.
- ADI_DEV_RESULT_BAD_DIRECTION_FIELD
  - In sequential I/O mode, buffers provided with an invalid direction value.
- ADI_DEV_RESULT_BAD_IVG
  - Bad IVG number detected.
- ADI_DEV_RESULT_ATTEMPTED_BUFFER_TABLE_NESTING
  - Nesting of Buffer table is not allowed.
- ADI_DEV_RESULT_DMA_CHANNEL_UNAVAILABLE
  - Failed to submit buffers as this device has no open DMA channel.
- ADI_DEV_RESULT_SWITCH_BUFFER_PAIR_INVALID
  - Invalid buffer pair provided with Switch/Update switch buffer type.

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_UART_RESULT_TIMER_ERROR
  - An error was detected when attempting to control and configure the timer for autobaud detection.
- ADI_UART_RESULT_BAD_BAUD_RATE
  - The baud rate of the UART is invalid. This error is usually a result of the client attempting to enable dataflow before the baud rate has been set or detected by the autobaud feature.
- ADI_UART_RESULT_NO_BUFFER
  - The driver has no buffer to process or from which to sense the processed element count.
- ADI_UART_RESULT_NOT_MAPPED_TO_DMA_CHANNEL
  - The driver has not mapped to a DMA channel interrupt.
- ADI_UART_RESULT_DMA_CHANNEL_NOT_MAPPED_TO_INTERRUPT
  - The driver DMA channel is not mapped to a peripheral interrupt.

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.

- ADIUARTDmaEntryPoint

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 client 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
Line status reporting	FALSE	TRUE/FALSE	ADI_UART_CMD_SET_LINE_STATUS_EVENTS
Number of data bits	5	5, 6, 7, 8	ADI_UART_CMD_SET_DATA_BITS
Number of stop bits	1	1, 1 ½, 2	ADI_UART_CMD_SET_STOP_BITS
Parity select	Odd	FALSE - odd, TRUE- even	ADI_UART_CMD_SET_PARITY
Parity checking	Disabled	FALSE – disabled, TRUE- enabled	ADI_UART_CMD_ENABLE_PARITY
Line status reporting	Disabled	FALSE – disabled, TRUE - enabled	ADI_UART_CMD_SET_LINE_STATUS_EVENTS

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 client to specify the additional configuration information listed in the table below.

Item	Possible Values	Command ID
Dataflow method	See section 5.2	ADI_DEV_CMD_SET_DATAFLOW_METHOD
Specify baud rate or enable autobaud detection		ADI_UART_CMD_SET_BAUD_RATE or ADI_UART_CMD_AUTOBAUD

Table 5 – Additional Required Settings

7. Hardware Considerations

There are no special hardware considerations for the UART device driver.