ANALOG DEVICES

ADI_CNT
DEVICE DRIVER

DATE: MARCH 10, 2007

Table of Contents

List of Tables

Document Revision History

Date	Description of Changes
2007-03-10	Initial version

Table 1 – Revision History

1. Overview

The driver allows the application to control the Rotary Counter device of MOAB Blackfin processor.
The device driver supports the following features of the Counter module:

- Operation Mode including QUAD_ENC,BIN_ENC,UD_CNT,DIR_CNT and DIR_TMR.
- Inputs de-bounce filtering.
- Zero Marker operation including Push-button, ZM-zeroes-Counter, ZM-error and Zero-once mode.
- Boundary Comparison modes including BND_COMP,BND_ZERO,BND_CAPT and BND_AEXT.
- Counter events (interrupts)

As written, the driver does not support ‘Capture Counter Event Timing ‘feature of the Counter device.
The device driver has been tested on the ADSP-BF548 EZ-Kit Lite development boards.

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/counter/adi_cnt.h> This file contains all commands, event and return codes specific to the Counter 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/counter/adi_cnt.c> This file contains all source code for the Counter device driver. All source code is written in ‘C’.

3. Lower Level Drivers

The Counter 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

This driver uses one interrupt, the Counter hardware interrupt.
The Counter device driver uses the default Interrupt Vector Group (IVG) mappings. Changes to the IVG mappings can be made by appropriate calls into the Interrupt Manager service prior to opening the Counter driver.

4.2. DMA

This driver does not use any DMA resources.

4.3. Timers

The counter module has an optional output pin TO (timer output) that can be used by GP Timer module to capture timing information. As written, this driver does NOT support ‘Capture Event Timing’ feature.
This driver does not use any timers.

4.4. Real-Time Clock

This driver does not use the real-time clock.

4.5. Programmable Flags

This driver does not use any programmable flag pins

4.6. Pins

Three input pins (CUD, CDG and CZM) are used by Counter module.
On processors where pins are multiplexed, the Counter device driver uses the Port Control service to automatically configure pins for use by the Counter module.
The pins are configured when the Counter device driver is opened.

5. Supported Features of the Device Driver

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

5.1. Directionality

There is no dataflow between Counter hardware and processor. The driver does not support the dataflow directions.

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 TRUE or FALSE is stored)

5.4.3. Device Driver Specific Commands

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

5.4.3.1. Commands to set device registers

ADI_CNT_CMD_SET_CONFIG_REG
- Sets the CNT_CONFIG register.
- Value – u16 (register value).

ADI_CNT_CMD_SET_IMASK_REG
- Sets the CNT_IMASK register.
- Value – u16 (register value).

ADI_CNT_CMD_SET_COMMAND_REG
- Sets the CNT_COMMAND register.
- Value – u16 (register value).

ADI_CNT_CMD_SET_DEBOUNCE_REG
- Sets the CNT_DEBOUNCE register.
- Value – u16 (register value).

ADI_CNT_CMD_SET_MAX_REG
- Sets the CNT_MAX register.
- Value – u32 (register value).

ADI_CNT_CMD_SET_MIN_REG
- Sets the CNT_MIN register.
- Value – u32 (register value).

5.4.3.2. Commands to read device registers

ADI_CNT_CMD_GET_STATUS_REG
- Senses the CNT_STATUS register
- Value – u16 * (location where contents of CNT_STATUS will be stored)

ADI_CNT_CMD_GET_COUNTER_REG
- Senses the CNT_COUNTER register
- Value – u32 * (location where contents of CNT_COUNTER will be stored)

ADI_CNT_CMD_GET_MAX_REG
- Senses the CNT_MAX register
- Value – u32 * (location where contents of CNT_MAX will be stored)

ADI_CNT_CMD_GET_MIN_REG
- Senses the CNT_MIN register
- Value – u32 * (location where contents of CNT_MIN will be stored)

5.4.3.3. Commands to configure device register fields

5.4.3.3.1. Configuration Register (CNT_CONFIG)

ADI_CNT_CMD_SET_CNT_ENABLE,
- Enables/disables the Counter Module
- Value – (1 – enabled, 0 – disabled)

ADI_CNT_CMD_SET_DEBOUNCE_ENABLE,
- Enables/disables the de-bounce filter(DEBE)
- Value – (1 – enabled, 0 – disabled)

ADI_CNT_CMD_CUD_CDG_DISABLE,
- Enables/disables the CUG and CDG input(INPDIS)
- Value – (1 – disabled, 0 – enabled)

ADI_CNT_CMD_SET_ZMZC_ENABLE,
- Enables/disables CZM Zeroes Counter(ZMZC)
- Value – (1 – enabled, 0 – disabled)

ADI_CNT_CMD_SET_CNTMODE,
- Sets the Counter Operating Mode
- Value – (3 bits)

  QUAD_ENC (0x000) = quadrature encoder mode
  BIN_ENC (0x001) = binary encoder mode
  UD_CNT (0x010) = up/down counter mode
  DIR_CNT (0x100) = direction counter mode
  DIR_TMRCNT (0x101) = direction timer mode

ADI_CNT_CMD_SET_BNDMODE
- Sets boundary comparison modes
- Value – (2 bits)

  BND_COMP (0x00) = boundary comparison mode
  BND_ZERO (0x01) = boundary compare and zero CNT_COUNTER mode
  BND_CAPT (0x10) = boundary capture mode (controlled by CZM)
  BND_AEXT (0x11) = boundary auto-extend mode

ADI_CNT_CMD_SET_CDG_POL_ALOW
- Sets CDG input polarity to active low
- Value – NULL

ADI_CNT_CMD_SET_CUD_POL_ALOW
- Sets CUD input polarity to active low
- Value – NULL

ADI_CNT_CMD_SET_CZM_POL_ALOW
- Sets CZM input polarity to active low

- Value – NULL

ADI_CNT_CMD_SET_CDG_POL_AHIGH
- Sets CDG input polarity to active high
- Value – NULL

ADI_CNT_CMD_SET_CUD_POL_ AHIGH
- Sets CUD input polarity to active high
- Value – NULL

ADI_CNT_CMD_SET_CZM_POL_ AHIGH
- Sets CZM input polarity to active high
- Value – NULL

5.4.3.3.2. Interrupt Mask Register (CNT_IMASK)

ADI_CNT_CMD_ILLEGAL_CODE_INT_EN
- Illegal gray/binary code interrupt enable
- Value – NULL

ADI_CNT_CMD_UPCOUNT_INT_EN
- Up-count interrupt enable.
- Value – NULL

ADI_CNT_CMD_DOWNCOUNT_INT_EN
- Down-count interrupt enable
- Value – NULL

ADI_CNT_CMD_MINCOUNT_INT_EN
- Minimum count (CNT_COUNTER == CNT_MIN) interrupt enable.
- Value – NULL

ADI_CNT_CMD_MAXCOUNT_INT_EN
- Minimum count (CNT_COUNTER == CNT_MIN) interrupt enable.
- Value – NULL

ADI_CNT_CMD_OVERFLOW31_INT_EN
- Bit 31 overflow interrupt enable.
- Value – NULL

ADI_CNT_CMD_OVERFLOW15_INT_EN,
- Bit 15 overflow interrupt enable.
- Value – NULL

ADI_CNT_CMD_COUNTZERO_INT_EN,
- CNT_COUNTER counts to zero interrupt enable.
- Value – NULL

ADI_CNT_CMD_CZMPIN_INT_EN,
- CZM pin (push-button) interrupt enable.
- Value – NULL

ADI_CNT_CMD_CZM_ERROR_INT_EN,
- Zero marker error interrupt enable.
- Value – NULL

ADI_CNT_CMD_CZM_COUNTZERO_INT_EN,

- Counter zeroes by zero marker interrupt enable.
- Value – NULL

ADI_CNT_CMD_ILLEGAL_CODE_INT_DIS
- Disable Illegal gray/binary code interrupt.
- Value – NULL

ADI_CNT_CMD_UPCOUNT_INT_DIS
- Disable Up-count interrupt.
- Value – NULL

ADI_CNT_CMD_DOWNCOUNT_INT_DIS
- Disable Down-count interrupt.
- Value – NULL

ADI_CNT_CMD_MINCOUNT_INT_DIS
- Disable Minimum count (CNT_COUNTER == CNT_MIN) interrupt.
- Value – NULL

ADI_CNT_CMD_MAXCOUNT_INT_DIS
- Disable Minimum count (CNT_COUNTER == CNT_MIN) interrupt.
- Value – NULL

ADI_CNT_CMD_OVERFLOW31_INT_DIS
- Disable Bit 31 overflow interrupt.
- Value – NULL

ADI_CNT_CMD_OVERFLOW15_INT_DIS
- Disable Bit 15 overflow interrupt.
- Value – NULL

ADI_CNT_CMD_COUNTZERO_INT_DIS
- Disable CNT_COUNTER counts to zero interrupt.
- Value – NULL

ADI_CNT_CMD_CZMPIN_INT_DIS,
- Disable CZM pin (push-button) interrupt.
- Value – NULL

ADI_CNT_CMD_CZM_ERROR_INT_DIS
- Disable Zero marker error interrupt.
- Value – NULL

ADI_CNT_CMD_CZM_COUNTZERO_INT_DIS
- Disable Counter zeroes by zero marker interrupt.
- Value – NULL

5.4.3.3.3. Command Register (CNT_COMMAND)

ADI_CNT_CMD_ZERO_CNT_COUNTER
- Command to zero the CNT_COUNTER
- Value – NULL

ADI_CNT_CMD_LOAD_MIN_TO_COUNTER
- Command to load CNT_COUNTER from CNT_MIN
- Value – NULL

ADI_CNT_CMD_LOAD_MAX_TO_COUNTER
- Command to load CNT_COUNTER from CNT_MAX
- Value – NULL

ADI_CNT_CMD_ZERO_CNT_MIN
- Command to zero the CNT_MIN
- Value – NULL

ADI_CNT_CMD_COUNTER_TO_CNT_MIN
- Command to capture CNT_COUNTER to CNT_MIN
- Value – NULL

ADI_CNT_CMD_CNT_MAX_TO_CNT_MIN
- Command to copy old CNT_MAX to new CNT_MIN
- Value – NULL

ADI_CNT_CMD_ZERO_CNT_MAX
- Command to zero the CNT_MAX
- Value – NULL

ADI_CNT_CMD_COUNTER_TO_CNT_MAX
- Command to capture CNT_COUNTER to CNT_MAX
- Value – NULL

ADI_CNT_CMD_CNT_MIN_TO_CNT_MAX
- Command to copy old CNT_MIN to new CNT_MAX
- Value – NULL

ADI_CNT_CMD_CZM_CLEAR_COUNTER_ONCE
- Command to clear CNT_COUNTER on the next active edge on the CZM pin once only.
- Value – NULL

ADI_CNT_CMD_GET_W1ZMONCE_BIT
- Command to read W1ZMONCE bit of the CNT_COMMAND.
- Value – u16 * (location where contents of W1ZMONCE bit will be stored)
  A return value of one indicates that bit has been set command ADI_CNT_CMD_CZM_CLEAR_COUNTER_ONCE before, but no zero marker event has been detected on the CZM pin.
  A return value of zero indicates that CNT_COUNTER has been cleared when zero marker event has been detected on the CZM pin.

5.5. Callback Events

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

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

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

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

5.5.1. Common Events

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

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_CNT_EVENT_ILLEGAL_CODE_INT
- Notifies the callback function that an illegal gray/binary code interrupt has occurred.
- pArg points to NULL.

ADI_CNT_EVENT_UPCOUNT_INT
- Notifies the callback function that up-count interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_DOWNCOUNT_INT
- Notifies the callback function that down-count interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_MINCOUNT_INT
- Notifies the callback function that (CNT_COUNTER == CNT_MIN) interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_MAXCOUNT_INT
- Notifies the callback function that (CNT_COUNTER == CNT_MAX) interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_OVERFLOW31_INT
- Notifies the callback function that Bit 31 overflow interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_OVERFLOW15_INT
- Notifies the callback function that Bit 15 overflow interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_COUNT_TO_ZERO_INT
- Notifies the callback function that (CNT_COUNTER == 0) interrupt has occurred.
- pArg points to ADI_CNT_CBSTATUS.

ADI_CNT_EVENT_CZMPIN_INT
- Notifies the callback function that push-button interrupt has occurred.
- pArg points to NULL.

ADI_CNT_EVENT_CZM_ERROR_INT
- Notifies the callback function that Zero marker error interrupt has occurred.

- pArg points to NULL.

ADI_CNT_EVENT_CZM_COUNTZERO_INT
- Notifies the callback function that CNT_COUNTER zeroes by Zero marker interrupt has occurred.
- pArg points to NULL.

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_CNT_RESULT_CMD_NOT_SUPPORTED
    - This error is a result of the client attempting to issue a command not supported by this driver.

- ADI_CNT_RESULT_CMD_VALUE_INVALID
    - This error is a result of the client attempts to write an invalid data to selected register field location.

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.

- ADCNTEntryPoint

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
De-bounce filter time (DPRESCALE)	4	0 to 7	ADI_CNT_CMD_SET_DEBOUNCE_REG
De-bounce enabled(DEBE)	1	0 or 1	ADI_CNT_CMD_SET_DEBOUNCE_ENABLE (0 = disable, 1 = enable)
CUD & CDG input enabled(INPDIS)	0	0 or 1	ADI_CNT_CMD_CUD_CDG_DISABLE (0 = enable, 1 = disable)
Boundary register mode(BNDMODE)	0	0 to 3	ADI_CNT_CMD_SET_BNDMODE
Counter operating mode (CNTMODE)	0	0 ,1,2,4,5	ADI_CNT_CMD_SET_CNTMODE
Zero-once-mode.(W1ZMONCE)	1	1 or 0	ADI_CNT_CMD_CZM_CLEAR_COUNTER_ONCE
Push-button interrupt enabled(CZMIE)	1	1 or 0	ADI_CNT_CMD_CZMPIN_INT_EN or ADI_CNT_CMD_CZMPIN_INT_DIS
Upcount interrupt enabled(UCII)	1	1 or 0	ADI_CNT_CMD_UPCOUNT_INT_EN or ADI_CNT_CMD_UPCOUNT_INT_DIS
Downcount interrupt enabled(DCII)	1	1 or 0	ADI_CNT_CMD_DOWNCOUNT_INT_EN or ADI_CNT_CMD_DOWNCOUNT_INT_DIS

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
Enables/disables COUNTER	1 or 0	ADI_CNT_CMD_SET_CNT_ENABLE (1 = enable, 0 = disable)

Table 3 – Additional Required Settings

7. Hardware Considerations

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

8. Appendix

8.1. Typical usage of COUNTER Device Driver in Applications

The application should allocate base memory + memory for one Counter device + memory for other devices used by the application.
The application should then make one call to the function adi_ssl_Init() to initialize Hardware (Ez-Kit), Interrupt manager, Deferred Callback Manager, DMA Manager, Device Manager (all application dependent).

Step 1: Open Counter Device driver with device specific entry point (refer section **Error! Reference source not found.** for valid entry points)
Counter driver sets the device in the following configuration:

- Counter operating mode: QUAD_ENC
- Boundary mode: BND_COMP
- CUD and CDG input enabled
- De-bounce enabled
- CDG and CUD pin polarity : active high
- CZM pin polarity : active high
- Upcount interrupt enabled
- Downcount interrupt enabled
- Push-button interrupt enabled
- Zero-once mode enabled.

Optional steps (change Counter settings by passing Counter driver specific commands)

Example:
(Optional) Step : write Maximum and Minimum counter register value.

```
/* sets Maximum counter register value to 65536*/
adi_dev_Control ( DriverHandle, ADI_CNT_CMD_SET_MAX_REG, (void *) 65536);

/* sets Minimum counter register value to - 65535*/
adi_dev_Control ( DriverHandle, ADI_CNT_CMD_SET_MIN_REG, (void *)-65535);
```

(Optional) Step : Enable interrupt for CNT_MAX and CNT_MIN.

```
/* enable interrupt for CNT_COUNTER == CNT_MAX */
adi_dev_Control(DriverHandle, ADI_CNT_CMD_MAXCOUNT_INT_EN, (void *)TRUE);

/* enable interrupt for CNT_COUNTER == CNT_MIN */
adi_dev_Control(DriverHandle, ADI_CNT_CMD_MINCOUNT_INT_EN, (void *)TRUE);
```

Step 2: Enable the Counter device.

```
adi_dev_Control(DriverHandle, ADI_CNT_CMD_SET_CNT_EN, (void*)TRUE);
```

Step 3: The callbacks happen at counter hardware interrupt. The event ID tells us which event has occurred (see 5.5.2 Device Driver Specific Events)
If the callback is generated by the CNT_COUNTER register value, address of structure ADI_CNT_CBSTATUS is passed as pArg.

Example:

```
case ADI_CNT_EVENT_UPCOUNT_INT:/* Up-count interrupt */
ADI_CNT_CBSTATUS *pResult;          /* pointer to the Callback result buffer*/
pResult = (ADI_CNT_CBSTATUS *)pArg;
pResult->CntCounter; /* CNT_COUNTER register value */
```

Step 4: Terminate Counter driver with adi_dev_Terminate( ).