ATS-SDK USER GUIDE
ANNEX 1

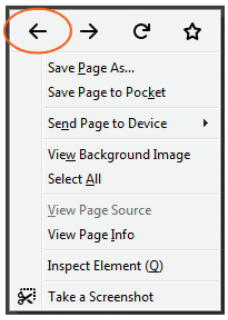
July 2018

AlazarTech

DOCUMENT NAVIGATION

This document contains intra-document links. You will need a PDF viewer with "Previous View" functionality to navigate through the manual with ease.

If you are opening this PDF with the built-in Mozilla® Firefox® PDF viewer, you can right-click anywhere in the PDF window to access page navigation:

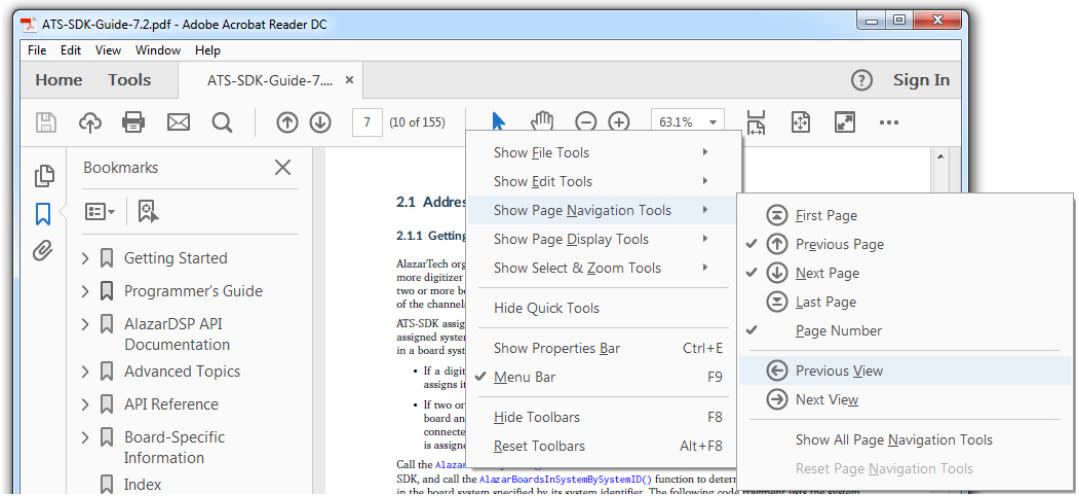


Otherwise, this PDF is best viewed using a PDF viewer with "Previous View" functionality. If your preferred PDF viewer does not include this functionality, you may wish to use one of the following[†] options:

- Foxit® Reader: https://www.foxitsoftware.com/pdf-reader/ (available for Linux® & Windows®)
- PDF Studio 2018: https://www.qoppa.com/pdfstudioviewer/download/ (available for Linux & Windows)
- Adobe® Acrobat® Reader DC: https://get.adobe.com/reader/ (available for Windows)

If you are using Adobe Acrobat Reader, you will need to enable the *Previous View* and *Next View* Page Navigation tools:

Right-click on the top toolbar and go to **Show Page Navigation Tools**, then select **Previous View**. Repeat the process for **Next View**.



[†]This document includes links to information created and maintained by other private and/or public organizations. Alazar Technologies Inc. (AlazarTech) provides these links solely for our users' information and convenience. AlazarTech does not control or guarantee the accuracy, relevance, or completeness of information contained on a linked website. Furthermore, AlazarTech does not endorse these organizations or the views they express or the products/services they offer. AlazarTech is not responsible for transmissions users receive from linked websites, nor is it responsible for or liable in any way for commercial transactions which users transact with linked websites.

ANNEX

1

DEPRECATED APIs & FUNCTIONS

A-1. Deprecated APIs

The following API is deprecated. Do not use in new designs.

A-1.1. Synchronous AutoDMA API

Synchronous AutoDMA API allows a board to transfer a segment of an AutoDMA acquisition into one buffer while – at the same time – the application processes a previous segment of the acquisition in another buffer.

Synchronous AutoDMA is deprecated; it is no longer maintained for compatibility with existing applications. The last ATSApi compatible with **Synchronous AutoDMA** is version 5.10.25. There are no bug fixes or updates for **Synchronous AutoDMA** after ATSApi version 5.10.25.

The **Asynchronous AutoDMA** API is recommended for all new applications.

The following table compares the asynchronous and synchronous AutoDMA APIs.

Attribute	Asynchronous AutoDMA	Synchronous AutoDMA
DMA buffer count	Application defined.	Two API allocated buffers.
CPU usage	Interrupt driven, so very low. More CPU cycles are available to application threads.	Polling loop, so very high. Less CPU cycles are available to application threads.
Data transfer	DMA directly into user-supplied buffer. No CPU cycles are used to copy data.	DMA into API allocated buffer, then copy to user-supplied buffer. CPU cycles used to copy data are not available to application threads.
DMA re- arm time	Next DMA started by hardware interrupt. Latency is lowest and data throughput is highest.	Next DMA started in polling loop. Latency is higher and data throughput is lower.
Master/slave systems	Fully supported.	Not recommended.

A-1.1.1. Using synchronous AutoDMA

Synchronous DMA API assumes that the PCI digitizer being controlled has dual-port acquisition memory.

As shown below, the user program consumes data synchronously with the acquisition loop. Hence the name Synchronous DMA.

A typical sequence of API calls for Synchronous DMA API is shown below. For readability purposes, the following is pseudo-code. Please refer to the sample programs provided for exact syntax and details of what the various parameters passed to these routines mean:

```
// Set up two AutoDMA buffers and start the DMA engine
// Data will be captured in the two buffers in a pin-pong
// mode.  You will be able to process the first buffer while
// data is being captured into the second buffer and
// vice-versa

AlazarStartAutoDMA(h,
                   UserData[0],
                   UseHeader,
                   mode,
                   -(long)bd.PreDepth,
                   transferLength,
                   RecsPerBuffer,
                   bd.RecordCount,
                   &error,
                   CFlags,
                   in1,
                   &r3,
                   &r4);

// Issue Start Capture Command. No data transfer happens before this

AlazarStartCapture( h );


// Wait until all required records have been captured

while (looping == 1)
{
   // Check if one of the AutoDMA buffers has been
   // fully populated or not

   AlazarGetNextAutoDMABuffer(h,
                              UserData[0],
                              UserData[1],
                              &WhichOne,
                              &RecsTransferred,
                              &error,
                              in1,
                              in1,
                              &TriggersOccurred,
                              &r4);

   // If WhichOne is equal to 0 or 1, that particular buffer
   // has been populated and hardware is DMAing
```

```
    // into the other buffer

    if ((WhichOne == 0)||(WhichOne == 1))
    {
       // Process Your Data here
       // Note that while you process data,
       // new data is still being captured into
       // on-board dual port memory and transferred into
       // the other AutoDMA buffer

       SaveToChannelFiles(UserData[WhichOne]);
    }


    // Check if all records have been captured
    if (RecsTransferred == (long)RecordCount)
    {
       // If all records have been captured, stop the while loop
       looping = 0;
    }

 }
```

Note:

- The synchronous AutoDMA API gives poor performance with master-slave systems, and is not recommended for use with such systems.
- Use the CFlags parameter in the call to AlazarStartAutoDMA to select the AutoDMA mode.
- Record headers are only available in Traditional AutoDMA mode. To enable record headers, call AlazarStartAutoDMA with the UseHeader parameter set to 1, and with the mode in the CFlags parameter set to ADMA_TRADITIONAL_MODE.
- AlazarGetNextAutoDMABuffer copies sample data from internally allocated AutoDMA buffers to an application buffer. An application may call this function with a pointer to a single application allocated buffer, rather than two application allocated buffers (Buffer[0] and Buffer[1] above) without affecting AutoDMA operation.
- Calling AlazarWaitNextAsyncBufferComplete in a polling loop is equivalent to calling AlazarEvents, AlazarWaitForBufferReady, and AlazarGetNextAutoDMABuffer, but provides more internally allocated buffers, better throughput, and a simpler programming interface.

A-2. Deprecated Functions

These functions are deprecated. Do not use in new designs.

A-2.1. AlazarAbortAutoDma

This routine is used to terminate the AutoDMA capture in cases where the trigger system stopped generating triggers before the buffer was filled by the AutoDMA engine. The routine will populate the buffer with the appropriate number of records that have been successfully captured.

Syntax

```
C/C++
RETURN_CODE
AlazarAbortAutoDMA(
      HANDLE h,
      void* Buffer,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      U32 *r3,
      U32 *r4
);
```

Parameters

h

[in] Board identification handle.

Buffer

[out] This Buffer is used to transfer a set of Records from the Device back to the user application.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set

ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

r1

[in] RESERVED.

r2

[in] RESERVED.

r3

[out] RESERVED.

r4

[out] RESERVED.

Return values

See **enum RETURN_CODE** in the ATS-SDK User Guide for a list of error codes.

See Also

AlazarStartAutoDMA
AlazarCloseAUTODma
Using synchronous AutoDMA

A-2.2. AlazarClose

Close a board handle.

Syntax

```
C/C++
void AlazarClose(
      HANDLE BoardHandle
);
```

Parameters

BoardHandle
[in] Handle to board.

Return value

If the board is acquiring to on-board memory, this function returns 1. Otherwise, this function returns 0.

Remarks

The API manages board handles internally. This function should only be used in applications that are written for single board digitizer systems.

See Also

AlazarOpen

A-2.3. AlazarCloseAUTODma

This routine will close the AUTODMA capabilities of the device. Only call this upon exit or error.

Syntax

C/C++

```
RETURN_CODE
AlazarCloseAUTODma(
      HANDLE h,
);
```

Parameters

h

[in] Board identification handle.

Return values

See **enum RETURN_CODE** in the ATS-SDK User Guide for a list of error codes.

See Also

AlazarAbortAutoDma
Using synchronous AutoDMA

A-2.4. AlazarEvents

This function allows a user to enable or disable usage of software events in AutoDMA mode. The driver manages the event processing and a user can only use an event in conjunction with the API AlazarWaitForBufferReady (...). When the events are enabled AlazarWaitForBufferReady(...) will wait until an AutoDMA buffer is available to the users application. For a complete understanding of the Usage of the API **AlazarEvents** (...) refer to the pseudo-code example provided in the API AlazarWaitForBufferReady (...).

Syntax

```
C/C++
RETURN_CODE
AlazarEvents(
      HANDLE h,
      U32 enable
);
```

Parameters

h

[in] Handle to the device.

enable

[in] This parameter may have one of the following values.

Identifier	Value	Description
SW_EVENTS_OFF	0	Disable events usage
SW_EVENTS_ON	1	Enable event usage

Return value

ApiSuccess (512) signifies that the API was able to enable the events

ApiFailed (513) signifies that the current driver does not support this feature.

Remarks

This functionality is only present on the ATS460, ATS660 and ATSS860 devices. It must be called before calling AlazarStartAutoDMA().

If AlazarEvents(h,1) was not used, calling AlazarWaitForBufferReady(…) will return 672 and will not disrupt any ongoing signal captures.

See Also

AlazarWaitForBufferReady
Using synchronous AutoDMA

A-2.5. AlazarFlushAutoDMA

The primary use of the API is to stop a Synchronous NPT acquisition. Scanning type applications are usually configured such that the data capture is ongoing and stopping is done by an external event. In this case trigger events have stopped and this API permits the last buffer to be returned to the application.

Syntax

```
C/C++
long AlazarFlushAutoDMA (HANDLE h);
```

Parameters

h

[in] Handle to the device.

Return value

The number of valid triggers in the last buffer.

Remarks

Suppose an acquisition is running and all of the sudden, triggers stop coming in. Once the software has determined that the acquisition is to be aborted, **AlazarFlushAutoDMA** should be called. The routine will automatically generate the missing triggers in order to complete the last buffer.

A last call to AlazarGetNextAutoDMABuffer is needed to read the LAST buffer. You will get ApiFailed as a return value from AlazarGetNextAutoDMABuffer indicating a successful last buffer. At this point, depending on your design, you may terminate the program or start a new acquisition.

NOTE:
Internally, this routine calls AlazarStopAutoDMA so as not to allow the software to re arm any new DMA requests. Only a call to AlazarStartAutoDMA will reset this action.

See Also

AlazarGetNextAutoDMABuffer
AlazarStartAutoDMA
Using synchronous AutoDMA

A-2.6. AlazarGetAutoDMAHeaderTimeStamp

This routine is a helper function, which can be used to retrieve the 40-bit TimeStamp from the header of a particular record. The resulting number is composed of both the TimeStampHighPart and TimeStampLowPart thus alleviating the user from calculating the time stamp using the header values.

Syntax

```
C/C++
float
AlazarGetAutoDMAHeaderTimeStamp(
      HANDLE h,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the given Data Buffer.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress

ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

Upon success, i.e. error==ADMA_Success, the TimeStamp will be returned in a floating-point format.

See Also

AlazarGetAutoDMAHeaderValue
AlazarGetAutoDMAPtr

A-2.7. AlazarGetAutoDMAHeaderValue

This routine is a helper function that can be used to retrieve all the various elements available in the header of an AutoDMA record. It will only operate on records that were captured when the Use Header variable in AlazarStartAutoDMA was set to a 1.

Syntax

```
C/C++
U32
AlazarGetAutoDMAHeaderValue(
      HANDLE h,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      U32 Parameter,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the provided Data Buffer.

Parameter

[in] Signifies which element the routine should extract from the record's header.

This parameter may be one of the following identifiers or values.

Identifier	Value
ADMA_CLOCKSOURCE	1
ADMA_CLOCKEDGE	2
ADMA_SAMPLERATE	3
ADMA_INPUTRANGE	4
ADMA_INPUTCOUPLING	5
ADMA_IMPUTIMPEDENCE	6
ADMA_EXTTRIGGERED	7
ADMA_CHA_TRIGGERED	8
ADMA_CHB_TRIGGERED	9
ADMA_TIMEOUT	10
ADMA_THISCHANTRIGGERED	11
ADMA_SERIALNUMBER	12

ADMA_SYSTEMNUMBER	13
ADMA_BOARDNUMBER	14
ADMA_WHICHCHANNEL	15
ADMA_SAMPLERESOLUTION	16
ADMA_DATAFORMAT	17

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

IF error==ADMA_Success, then the value of the asked Parameter is returned.

See Also

AlazarGetAutoDMAPtr
AlazarGetAutoDMAHeaderTimeStamp

A-2.8. AlazarGetAutoDMAPtr

This routine is a helper function used to retrieve a pointer to the first data element or first header element of a particular record. If DataOrHeader is set to 1, then the resulting pointer must be cast to PALAZAR_HEADER type. The user can then use the pointer to access any of the header variables.

Ex. PALAZAR_HEADER p = (PALAZAR_HEADER) AlazarGetAutoDMAPtr (…);

Syntax

```
C/C++
void *
AlazarGetAutoDMAPtr(
      HANDLE h,
      U32 DataOrHeader,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

DataOrHeader

[in] Instruct the routine to return a pointer for the data or header portion. This parameter may be one of the following values.

Value	Meaning
0	Return the pointer for the data portion.
1	Return the pointer for the header portion.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the given Data Buffer.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer

ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

See **enum RETURN_CODE** in the ATS-SDK User Guide for a list of error codes.

See Also

AlazarGetAutoDMAHeaderTimeStamp
AlazarGetAutoDMAHeaderValue

A-2.9. AlazarGetNextAutoDMABuffer

After an application has called AlazarStartAutoDMA the application must call **AlazarGetNextAutoDMABuffer** to retrieve the data buffers. Because of the nature of Auto DMA, two buffers are required. The device driver dll will arbitrate to which buffer the data will be returned. After a buffer has been filled, variable WhichOne equals the buffer id, thus if the id is 0 then Buffer1 was used and likewise if the id is 1 then Buffer2 was used. In the case where data is not available WhichOne will equal -1. This routine will always return ApiSuccess (512) when either data has been transferred or when WhichOne = -1. A return value of ApiFailed (513) indicates that all the Records Per Buffer has been transferred

Syntax

C/C++

```
RETURN_CODE
AlazarGetNextAutoDMABuffer(
      HANDLE h,
      void* Buffer1,
      void* Buffer2,
      long* WhichOne,
      long* RecordsTransferred,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      long *TriggersOccurred,
      U32 * r4
);
```

Parameters

h

[in] Handle to the device.

Buffer1

[out] This Buffer is used to transfer a complete set of Records from the Device back to the user application. It is one of two buffers that are alternated between. The second buffer is Buffer2.

Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

Buffer2

[out] This Buffer is used to transfer a complete set of Records from the Device back to the user. It is one of two buffers that are alternated between. The other buffer is Buffer1.

Buffer2 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer2 should be large enough to hold.

WhichOne

[out] This is a return value that indicates to the user which of the two Buffers (Buffer1 or Buffer2) the data was transferred into.

RecordsTransferred

[in | out] Indicates how many records have been transferred. This value will always be a multiple of RecordsPerBuffer. It is the application's responsibility to initialize the variable to 0 prior to the first call

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

r1

[in] RESERVED.

r2

[in] RESERVED.

TriggersOccurred

[out] This is the total number of triggers that have been captured since the last start capture.

r4

[out] RESERVED.

Return value

See **enum RETURN_CODE** in the ATS-SDK User Guide for a list of error codes.

Remarks

Both Buffer1 and Buffer2 will be used in transferring the data from the device back to the user application. However, if the RecordsPerBuffer is set in conjunction with TransferLength such that all the data will fit in only one Buffer, then Only Buffer1 will be used and the WhichOne variable will equal 0. Only one transaction will take place.

RecordsTransferred will be modified by the routine and is used to accumulate the number of record that has been transferred. Always set the variable to 0 before calling this routine and never modify its contents between repeating calls.

The user must ensure that Buffer1 and Buffer2 are valid buffers.

Buffer1 and Buffer2 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 and Buffer2 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values (VB-Integer, C&C++-short).

AlazarGetNextBuffer and **AlazarGetNextAutoDMABuffer** are identical.

See Also

AlazarStartAutoDMA
AlazarAbortAutoDma
AlazarGetNextBuffer
Using synchronous AutoDMA

A-2.10. AlazarGetNextBuffer

AlazarGetNextBuffer and AlazarGetNextAutoDMABuffer are identical. Please refer to AlazarGetNextAutoDMABuffer.

Syntax

```
C/C++
RETURN_CODE
AlazarGetNextBuffer(
      HANDLE h,
      void* Buffer1,
      void* Buffer2,
      long* WhichOne,
      long* RecordsTransferred,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      long *TriggersOccurred,
      U32 * r4
);
```

Remarks

AlazarGetNextBuffer and AlazarGetNextAutoDMABuffer are identical.

See Also

AlazarGetNextAutoDMABuffer

A-2.11. AlazarOpen

Open and initialize a board.

Syntax

```
C/C++
HANDLE
AlazarOpen(
      char *BoardName
);
```

Parameters

BoardName

[in] Name of board created by driver. For example “ATS850-0”.

Return value

A handle to the board.

Remarks

The ATS library manages board handles internally. This function should only be used in applications that are written for single board digitizer systems.

See Also

AlazarClose

A-2.12. AlazarStartAutoDMA

This routine is used to enable the AUTODMA functionalities of the device. It must be called prior to calling AlazarGetNextBuffer(...).

Syntax

```
C/C++
RETURN_CODE
AlazarStartAutoDMA(
      HANDLE h,
      void*
      Buffer1, U32
      UseHeader,
      U32 ChannelSelect,
      long TransferOffset,
      U32 TransferLength,
      U32 RecordsPerBuffer,
      U32 RecordCount,
      AUTODMA_STATUS*
      error, U32 cFlags,
      U32 r2,
      U32
      *r3,
      U32 *r4
);
```

Parameters

h

[in] Handle to the device.

Buffer1

[out] Data buffer for the first set of transferred records. Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

UseHeader

[in] If equal to 1 then the AUTODMA record header will precede each record in the Buffer

ChannelSelect

[in] This parameter may be one of the following identifiers or values.

Identifier	Value	Meaning
CHANNEL_A	1	Single channel mode
CHANNEL_B	2	Single channel mode
CHANNEL_A | CHANNEL_B	3	Dual channel mode

TransferOffset

[in] Transfer offset relative to the Trigger point for each record.

TransferLength

[in] The amount to transfer for each record.

RecordsPerBuffer

[in] The number of records that will be transferred into Buffer1. (Please note the size information in Buffer1 description).

RecordCount

[in] The number of records to be captured during this acquisition. Infinite Record Count can be used to create an endless capture for any AutoDMA mode. To use Inifinite records, set the RecordCount parameter of AlazarStartAutoDMA(…) to 0x7FFFFFFF. It is the user's responsibility to set the criteria for stopping an acquisition.

Note that AlazarStartAutoDMA routine will overwrite any previous settings for this parameter with the value passed in the RecordCount parameter (Please note the size information in Buffer1 description).

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

cFlags

[in] Control Flags,{0 = The routine will automatically start the acquisition, 1 = The user application must call AlazarStartCapture to start the acquisition}. The constants available are as follows:

Identifier	Meaning
ADMA_EXTERNAL_STARTCAPTURE 0x00000001	The User must call AlazarStartCapture to start the acquisition
ADMA_TRADITIONAL_MODE 0x00000000	Traditional Auto Dma mode captures
ADMA_CONTINUOUS_MODE 0x00000100	Continuous Streaming mode without trigger
ADMA_NPT 0x00000200	No-Pre-Trigger Auto DMA mode

r2

[in] RESERVED.

r3

[out] RESERVED.

r4

[out] RESERVED.

Return value

See **enum RETURN_CODE** in the ATS-SDK User Guide for a list of error codes.

Remarks

The user must ensure that Buffer1 is a valid buffer of the appropriate size.

Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16- bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

See Also

AlazarAbortAutoDma
AlazarGetNextAutoDMABuffer
Using synchronous AutoDMA

A-2.13. AlazarStopAutoDMA

This API is used to inhibit the software from issuing any new DMA request to the device. It is meant as a helper function for the AlazarFlushAutoDMA API function.

Syntax

```
C/C++
Void AlazarStopAutoDMA(HANDLE h);
```

Parameters

h

[in] Handle to board.

Return value

None

Remarks

This function is useful in situations where the application software has multiple threads. The software can call this routine to stop the device from issuing DMA requests in preparation for calling API AlazarFlushAutoDMA.

See Also

AlazarFlushAutoDMA
Using synchronous AutoDMA

A-2.14. AlazarWaitForBufferReady

This function will stall the current thread of execution for tms number milliseconds or until a buffer has been successfully transferred to a user space AutoDMA buffer. The function must be called after API AlazarEvents(h,1) and before API AlazarGetNextAutoDMABuffer(…). It will wait on the driver to signal the Driver's Internal registered event for up to tms number of milliseconds. When the DMA completes, the signaling event will wake up the Api.

Syntax

```
C/C++
RETURN_CODE
AlazarWaitForBufferReady(
      HANDLE h,
      U32 tms
);
```

Parameters

h

[in] Handle to the device.

tms

[in] time in milliseconds.

Return values

670 - signifies that a NULL was used for the handle

671 - signifies that the current device driver does not support events. 672 – Events were not activated using API AlazarEvents.

ApiSuccessFul or 512 signifies that the internal wait event was successfully registered and signaled by the ISR.

ApiFailed or 513 signifies that the internal wait event did not register.

ApiWaitTimeOut or 579 signifies that the internal wait event was not signaled by the ISR.

Remarks

This functionality is only present on the ATS460, ATS660 and ATSS860 devices.

If AlazarEvents(h,1) was not used, calling **AlazarWaitForBufferReady**(...) will return ApiFailed and will not disrupt any ongoing signal captures.

Below is a pseudo-code fragment that shows the operations of API AlazarEvents(…) and API **AlazarWaitForBufferReady**(…).

Pseudo-code:

```
AlazarSetRecordSize(...);
AlazarSetCaptureClock(...);
AlazarInputControl(...);
AlazarInputControl(...);
AlazarSetTriggerOperation(...)
//
AlazarEvents(h,1);
//
AlazarStartAutoDMA(...);
while (looping == 1)
{
      AlazarWaitForBufferReady(h, 10);
      status = AlazarGetNextAutoDMABuffer();
      if (status == 513)
      {
             looping = 0;
       }
       //Valid data exists in either UserData[0] or UserData[1]
       if ((WhichOne == 0)||(WhichOne == 1))
       {
             //Process Your Data here
             ...
       }
       if (error == ADMA_OverFlow)
       {
             looping = 0;
             returnValue = -4;
       }

}
AlazarCloseAUTODma(...);
//
AlazarEvents(h,0);
//
```

See Also

AlazarEvents
Using synchronous AutoDMA

INDEX