ATS-GPU-BASE

Version 25.1.0
April 17, 2025

CONTENTS

Note: This is the documentation for AlazarTech's ATS-GPU version 25.1.0. Please visit our documentation homepage to find documentation for other versions or products.

 ©2008-2025 Alazar Technologies Inc.

CHAPTER ONE

LICENSE AGREEMENT

Copyright (c) @COPYRIGHT_DATE@ Alazar Technologies, Inc.

1.1 Important

CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT. BY CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ANY ACCOMPANYING MEDIA) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO ALAZAR TECHNOLOGIES INC. ("ALAZARTECH") WILL BE SUBJECT TO ALAZARTECH'S THEN-CURRENT POLICY. IF YOU ARE ACCEPTING THESE TERMS ON BEHALF OF AN ENTITY, YOU AGREE THAT YOU HAVE AUTHORITY TO BIND THE ENTITY TO THESE TERMS.

1.2 Ownership

AlazarTech retains the ownership of ATS-GPU software ("Software"). It is licensed to you for use under the following conditions:

1.2.1 Grant of License

You may only concurrently use the Software on the computers that have an AlazarTech waveform digitizer card plugged in (for example, if you have purchased one AlazarTech card, you have a license for one concurrent usage). If the number of users of the Software exceeds the number of AlazarTech cards you have purchased, you must have a reasonable process in place to assure that the number of persons concurrently using the Software does not exceed the number of AlazarTech cards purchased.

This license is non-transferable.

1.2.2 Restrictions

You may not copy the documentation or Software except as described in the installation section of the Software manual. You may not distribute, rent, sub-lease or lease the Software or documentation, including translating or decomposing. You may not modify, reverse-engineer, decompile, or disassemble any part of the Software or documentation, or produce any derivative work other than software applications that communicate with AlazarTech hardware using the published Application Programming Interface (API).

You may not remove, block, or modify any titles, logos, trademarks, copyright and/or patent notices, digital watermarks, disclaimers, or other legal notices that are included in the Software.

1.2.3 Termination

This license and your right to use this Software automatically terminates if you fail to comply with any provision of this license agreement.

1.3 Rights

AlazarTech retains all rights not expressly granted. Nothing in this agreement constitutes a waiver of AlazarTech's rights under the Canadian and U.S. copyright laws or any other Federal or State law.

1.4 Limited Warranty

Although AlazarTech has tested the Software and reviewed the documentation, ALAZARTECH MAKES NO WARRANTY OF REPRESENTATION, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE OR DOCUMENTATION, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE AND DOCUMENTATION IS LICENSED "as is" AND YOU, THE LICENSEE, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE. IN NO EVENT WILL ALAZARTECH BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SOFTWARE OR DOCUMENTATION, even if advised of the possibility of such damages. In particular, AlazarTech shall have no liability for any data acquired, stored or processed with this Software, including the costs of recovering such data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED. No AlazarTech dealer, agent or employee is authorized to make any modifications or additions to this warranty.

Information in this document is subject to change without notice and does not represent a commitment on the part of AlazarTech. The Software described in this document is furnished under this license agreement. The Software may be used or copied only in accordance with the terms of the agreement.

©2008-2025 Alazar Technologies Inc.

Some jurisdictions do not allow the exclusion of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

 ©2008-2025 Alazar Technologies Inc.

CHAPTER

TWO

INTRODUCTION

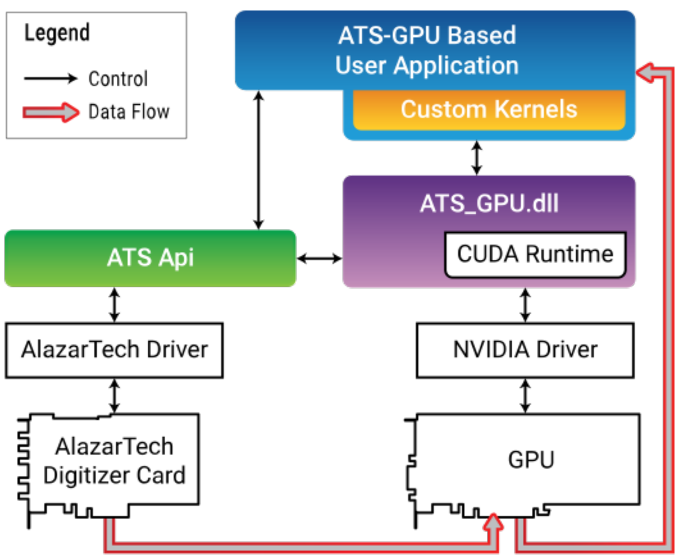
The ATS-GPU SDK provides a framework to allow real-time processing of data from AlazarTech PCIe digitizers on a CUDA-enabled GPU. This programmer's guide covers the use of ATS-GPU-BASE.

ATS-GPU-BASE internally calls ATS-CUDA, which is a wrapper library for simple CUDA calls. ATS-CUDA is described in more detail later in this guide in the section *ATS-CUDA*.

This document assumes that the reader is familiar with ATS-SDK, the standard interface for programming AlazarTech digitizers. Having a copy of the ATS-SDK manual available can be helpful, since many references to ATSApi functions are done here. The latest version of the ATS-SDK manual can be downloaded free of charge from AlazarTech's website.

In addition, expertise in CUDA programming is assumed. This is particularly important for users wishing to use ATS-GPU-BASE, because this task involves CUDA programming.

It is also essential for programmers to have in-depth knowledge of GPU architecture and parallel programming.



 ©2008-2025 Alazar Technologies Inc.

CHAPTER

THREE

PREREQUISITES

3.1 System requirements

This software requires a PC with a CUDA-enabled GPU, and sufficient CPU resources to supply data to the GPU at the desired data acquisition rate. It was tested with GeForce GTX Titan X (Maxwell), GeForce GTX980 and Quadro P5000. DDR4 memory and a modern chipset (X99, X299) will greatly improve transfer speed and overall performance.

Supported operating systems
: 64-bit Windows and 64-bit Linux operating systems are supported. Please verify that your Linux distribution is supported by NVIDIA , which supplies the CUDA toolkit required to use ATS-GPU.

Compiler support
: CMake is required to build C/C++ code. CMake files are provided. On Linux, a C++11 compiler is required to build the library. On older Red Hat distributions, a devtoolset can be obtained to use a more recent version of gcc that supports C++11. NVCC is required to compile the example code, this compiler is included with CUDA toolkit.

CUDA driver requirements
: In order to use ATS-GPU, you must install the appropriate driver for your CUDA-enabled GPU. Drivers can be downloaded at https://www.nvidia.com/Download/index.aspx.

Note: Under Windows operating systems, dynamic link libraries related to ATS-GPU-BASE are installed by default in %WINDIR%System32. For applications to link appropriately to them, %WINDIR%System32 must be added to the Windows PATH Environment Variable.

3.2 Programming experience

Users who wish to use ATS-GPU-BASE to create high-performance custom kernels must have expertise in CUDA programming.

It is also essential for programmers to have in-depth knowledge of GPU architecture and parallel programming.

 ©2008-2025 Alazar Technologies Inc.

CHAPTER FOUR

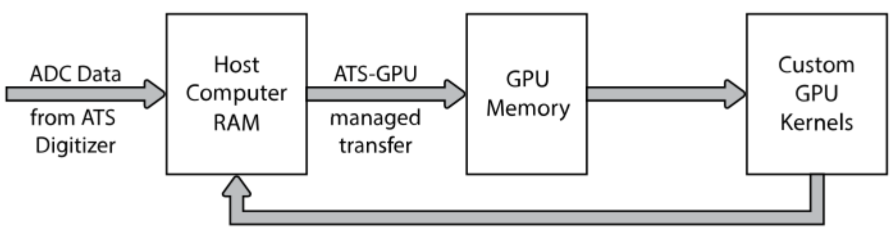
ATS-GPU-BASE

ATS-GPU-BASE is designed to provide highly efficient code to transfer data from an ATS PCIe digitizer to a CUDA-enabled GPU for processing. This transfer is done using multiple DMA transactions. The user application, which includes custom CUDA kernels, can then access data buffers on the GPU. The user is then responsible to perform data processing and copy data back to the CPU if required. A code example is provided as an example of a user application that performs very simple signal processing (data inversion).

Sample programs installation path

- Windows: "C:\AlazarTech\ATS-GPU\25.1.0\base"
- Linux: "/usr/local/AlazarTech/ATS-GPU/25.1.0/base"

4.1 Usage



ATS-GPU Data Flow

ATS-GPU-BASE offers several functions that behave similarly to ATSApi functions. Please refer to the ATS-SDK guide for more details about these APIs. Obtaining a board handle and configuring the board (sampling rate, trigger, input channels, etc.) is performed directly using functions from the ATS-SDK. By convention, the code samples define a `ConfigureBoard()` function that handles all these tasks.

```
if (!ConfigureBoard(boardHandle)) {
    // Error handling
}
```

During the lifetime of an application, multiple acquisitions can take place. If the board configuration parameters do not change, it is not necessary to call `ConfigureBoard()` again.

The next step is to select the CUDA-enabled GPU to use for the data transfer. This call is optional. If you only have one CUDA capable GPU on your computer, you can skip it.

```
rc = ATS_GPU_SetCUDAComputeDevice(boardHandle, deviceIndex);
// Error handling
```

We must then setup parameters of the acquisition to GPU. This function replaces the call to `AlazarBeforeAsyncRead()` in normal programs. Parameters were kept as close as possible to those of `AlazarBeforeAsyncRead()` to ease transition between standard acquisitions and ATS-GPU acquisitions. To maximize performance, sample interleave should be enabled with `ADMA_INTERLEAVE_SAMPLES`.

```
rc = ATS_GPU_Setup(boardHandle, channelSelect, transferOffset,
                   transferLength, recordsPerBuffer, recordsPerAcquisition,
                   autoDMAFlags, ATSGPUFlags);
// Error handling
```

We then allocate memory on the GPU for data to be transferred to, and we post those buffers to the board. For this purpose, we use `ATS_GPU_AllocBuffer()`. This function allocates a buffer on the GPU and sets up all the intermediary state necessary for ATS-GPU to successfully transfer data. Please note that if you would like to send data back from the GPU to your computer's RAM after having processed it, you will need to allocate memory independently of the AlazarTech APIs.

```
for (size_t i = 0; i < buffers_to_allocate; i++)
{
    buffers[i] = ATS_GPU_AllocBuffer(boardHandle, bytesPerBuffer);

    rc =  ATS_GPU_PostBuffer(boardHandle,
                             buffers[i],
                             bytesPerBuffer);
    // Error handling
}
```

We can then start the acquisition. The board will directly start acquiring data, assuming it receives triggers, and data transfer to posted GPU buffers will also start immediately.

```
rc =  ATS_GPU_StartCapture(HANDLE boardHandle);
// Error handling
```

Once acquisition is started, `ATS_GPU_GetBuffer()` must be called as often as possible to retrieve a buffer containing data already copied on the GPU. This buffer can then be processed by your custom kernel on the GPU. When a buffer is done being used (either data has been copied to a different buffer or processing is complete), the buffer needs to be posted back to the board.

```
for (size_t i; i < buffers_per_acquisition; i++)
{
    rc =  ATS_GPU_GetBuffer(boardHandle,
                            buffers[i],
```

(continues on next page)

 ©2008-2025 Alazar Technologies Inc.

(continued from previous page)

```
                            timeout_ms,
                            nullptr);

    // TODO: Error handling

    // TODO: Process buffer. This is where you can call your own processing
    //       function that launches the GPU kernels, such as ProcessBuffer()
    //       in the code samples.
    ProcessBuffer(buffers[i], bytesPerBuffer);

    rc = ATS_GPU_PostBuffer(boardHandle, buffer, bytesPerBuffer);
}
```

When acquisition is complete, `ATS_GPU_AbortCapture()` must be called. Buffers allocated with `ATS_GPU_AllocBuffer()` should then be freed with `ATS_GPU_FreeBuffer()`.

```
RETURN_CODE ATS_GPU_AbortCapture(HANDLE boardHandle);

for (size_t i = 0; i < number_of_buffers; i++)
{
   rc = ATS_GPU_FreeBuffer(boardHandle, buffers[i]);
   // Error handling
}
```

Here is an example of what the function to process data on the GPU can look like. Since this contains code that is executed on the GPU, it needs to be located in a file with a `.cu` extension:

```
extern “C”__global__ void ProcessBuffer(void* buffer, bytesPerBuffer)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;

    // TODO: Do processing here
}

Bool ProcessBuffer(void* buffer, U32 bytesPerBuffer)
{
    // Launch ProcessBuffer CUDA kernel
    ProcessBuffer<<<threadsPerBlock, BlocksPerGrid>>>(buffer, bytesPerBuffer);

    // Copy result from GPU memory to CPU memory
    cudaMemcpy(resultBuffer,buffer,bytesPerBuffer);
}
```

4.2 Performance guidelines

While GPU solutions are highly customizable and can reach very high processing speeds, care must be taken to preserve performance. The provided libraries use streams to maximise concurrency and hide latency associated with data transfers. The processing functions are optimized to provide the best performance and modifying them can result in a loss of performance. Refer to the CUDA best practices guide for more information on how to improve performance.

> **Warning:** When developing CUDA code, it is very important to check memory accesses with a dedicated tool, as bad memory accesses will not necessarily trigger an error but will lead to bad behavior and can cause a crash. The CUDA toolkit provides the necessary memory checking utilities.

Because data is DMA'd from ATS board to host memory then to GPU memory, speed of host computer memory will influence performance. DDR4 memory and a modern chipset (X99, X299, etc.) will greatly improve transfer speed and overall performance.

4.3 Benchmarks

Performance benchmarks using the optional OCT signal processing library and NVIDIA GeForce GTX Titan X (Maxwell) GPU on an ASUS X99 Deluxe motherboard with an Intel i9-7900X 3.3 GHz CPU, and 2133 MHz DDR4 memory (32 GB RAM):

PCIe Link Speed	Transfer Rate
Gen 3x8: ATS9373, ATS9371	Up to 6.9 GB/s
Gen 2x8: ATS9360, ATS9416	Up to 3.5 GB/s
Gen 2x4: ATS9352 Gen 1x8: ATS9870, ATS9350, ATS9351, ATS9625, ATS9626, ATS9440	Up to 1.6 GB/s
Gen 1x4: ATS9462	Up to 720 MB/s
Gen 1x1: ATS9146, ATS9130, ATS9120	Up to 200 MB/s

 ©2008-2025 Alazar Technologies Inc.

4.4 API Reference

Note: Errors from ATS-GPU-BASE will be logged in ATS_GPU.log. Relevant information about the error will be logged here and can be useful for debugging. For Windows users log file is located in %TEMP%. For Linux users log file is located in /tmp/.

RETURN_CODE **`ATS_GPU_AbortCapture`**(HANDLE boardHandle)

Stops the acquisition.

Aborts an acquisition, stops data processing, and releases resources allocated by *ATS_GPU_Setup()*

Parameters

`boardHandle` – Handle to the board

Returns

ApiSuccess

RETURN_CODE **ATS_GPU_AddProcessingKernel**(HANDLE boardHandle, KERNEL *kernel)

Adds a kernel to an internal list of processing kernels that will be used by API function ATS_GPU_SetProcessingPipeline.

This function must be called after *ATS_GPU_Setup* and before *ATS_GPU_SetProcessingPipeline*.

If this function is called at least once, input parameters "kernelList" and "numberOfKernels" of function ATS_GPU_SetProcessingPipeline are ignored.

Parameters

- **boardHandle** – Handle to the board.
- **kernel** – Pointer to a kernel to add to the list of processing kernels.

See *ATS-GPU-HPC* for more information on types of kernels that can be launched on the GPU.

 ©2008-2025 Alazar Technologies Inc.

void ***ATS_GPU_AllocBuffer**(HANDLE boardHandle, U32 bytesPerBuffer, cudaStream_t *stream)

Allocates page-aligned pinned memory for ATS and GPU boards.

This function can be called after ATS_GPU_Setup to perform the necessary memory allocations. This function returns a GPU or CPU buffer pointer depending on *ATS_GPU_SETUP_FLAG* values used in the setup.

Parameters

- **boardHandle** – Handle to the board
- **bytesPerBuffer** – Total number of bytes to allocate per buffer
- **stream** – CUDA stream associated to the allocated buffer.

RETURN_CODE **ATS_GPU_FreeBuffer**(HANDLE boardHandle, void *buffer)

Free buffers allocated with *ATS_GPU_AllocBuffer()*;.

Parameters

- **boardHandle** – Handle to the board
- **buffer** – Buffer pointer allocated by *ATS_GPU_AllocBuffer()*

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_GenerateCPUBoxcarFunction**(float *boxcarFunction, U32 samplesPerRecord, U32 gateDelay, U32 gateWidth)

Generates a boxcar gate on the CPU, of length samplesPerRecord.

Parameters

- **boxcarFunction** – Array to be filled with the boxcar function. It must have a length of samplesPerRecord.
- **samplesPerRecord** – Samples per record.
- **gateDelay** – Delay of the boxcar gate in number of samples.
- **gateWidth** – Width of the boxcar gate in number of samples.

Returns

Pointer to an array of float elements that contains the boxcar window generated on the CPU.

RETURN_CODE **`ATS_GPU_GetBuffer`**(HANDLE boardHandle, void *buffer, U32 timeout_ms, cudaStream_t *stream)

Get processed buffer.

This function must be called at average rate that is equal to or greater than the rate at which DMA buffers complete. This function returns the GPU-processed buffer.

Parameters

- **`boardHandle`** – Handle to the board
- **`buffer`** – Pointer to the buffer
- **`timeout_ms`** – Time the board will wait for a trigger before throwing an error.
- **`stream`** – CUDA stream associated to the processed buffer. Subsequent processing of processed buffer should occurs on this CUDA stream.

Returns

`ApiSuccess` (512) if the board received sufficient triggers to fill a DMA buffer.

Returns

`ApiNotInitialized` if `ATS_StartCapture` was not called before calling this function, or it was called and failed.

Returns

`ApiInvalidHandle` The `boardHandle` parameter is not valid.

Returns

`ApiBufferOverflow` if the board filled all the available DMA buffers and its on-board memory. This may happen if the acquisition rate exceeds the bus bandwidth or the GPU processing bandwidth.

Returns

`ApiWaitTimeout` if the timeout interval expired before the board received a sufficient number of triggers to fill a buffer.

Returns

`ApiFailed` if a system of internal error occurred.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_GetNumberOfPendingFullDmaBuffers**(HANDLE boardHandle, unsigned int *numberOfBuffersFull)

Returns the number of buffers ready to be consumed.

RETURN_CODE **ATS_GPU_GetVersion**(U8 *major, U8 *minor, U8 *revision)

Get ATS-GPU version number.

Parameters

- **major** – ATS-GPU major version number.
- **minor** – ATS-GPU minor version number.
- **revision** – ATS-GPU revision number.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_ManageGetBuffer**(HANDLE boardHandle, void *buffer, U32 bytesToCopy, U32 timeout_ms)

Query a buffer through the managed DMA buffer API. For LabVIEW programmers view LabVIEW Programming section.

Parameters

- **boardHandle** – Handle to the board
- **buffer** – Pointer to a user-allocated buffer to receive data
- **bytesToCopy** – Number of bytes to copy to the user buffer
- **timeout_ms** – Maximum time to wait for data to be ready to be copied to `buffer` before returning `ApiWaitTimeout`.

RETURN_CODE **ATS_GPU_PostBuffer**(HANDLE boardHandle, void *buffer, U32 bytesPerBuffer)

Signal the library a particular buffer can be used for data transfer.

This function is the equivalent of AlazarPostAsyncBuffer for ATS_GPU. Buffers posted must have previously been allocated with ATS_GPU_AllocBuffer.

Parameters

- **boardHandle** – Handle to the board
- **buffer** – Pointer to a previously allocated buffer
- **bytesPerBuffer** – Size in bytes of the buffer, must be the same size as setup for the acquisition.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_QueryCUDADeviceCount**(U32 *pDeviceCount)

Function to get the number of available CUDA devices.

Parameters

pDeviceCount – Outputs the number of devices detected on the system.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if the GPU driver returned an error.

RETURN_CODE **ATS_GPU_QueryCUDADeviceName**(U32 deviceIndex, char *deviceName, int maxChars)

Function to get the name of a specific CUDA device.

Parameters

- **deviceIndex** – 0-based index to the device.
- **deviceName** – Char array to output the name of the device.
- **maxChars** – Size of the char array.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if the GPU driver returned an error.

Returns

`ApiInvalidIndex` if the index provided is greater than the number of platforms or devices available.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_QueryCUDADeviceSerialNumber**(U32 deviceIndex, char *deviceSerialNumber, int maxChars)

Function to get the serial number of a specific CUDA device.

Parameters

- **deviceIndex** – 0-based index to the device.
- **deviceName** – Char array to output the serial number of the device.
- **maxChars** – Size of the char array.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if the GPU driver returned an error.

Returns

`ApiInvalidIndex` if the index provided is greater than the number of platforms or devices available.

RETURN_CODE **`ATS_GPU_SetCUDAComputeDevice`**(HANDLE boardHandle, U32 deviceIndex)

CUDA-specific function used to associate a CUDA-enabled GPU device with a digitizer board.

Allows you to specify which GPU should be used to process sample data from a digitizer, if more than one GPU is available.

Parameters

- **`boardHandle`** – Handle to the ATS board.
- **`deviceIndex`** – 0-based index to the CUDA device.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if it failed. See `%TEMP%/ATS_GPU.log` (`/tmp/ATS_GPU.log` under Linux) for more information.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_SetLogging**(BOOL activateLogging)

Activate/deactivate logging of calls made to ATS-GPU libraries.

Calls are logged in ATS-GPU.log. Independent of logging state, errors in ATS-GPU calls will always be logged.

Parameters

activateLogging – True activates logging. False deactivates logging.

Returns

ApiSuccess if it succeeded.

Returns

ApiFailed if it failed.

RETURN_CODE **ATS_GPU_SetProcessingPipeline**(HANDLE boardHandle, KERNEL *kernelList, U32 numberOfKernels, U32 *bytesPerResultBuffer)

Prepares the processing pipeline to be performed on the GPU.

This function must be called after *ATS_GPU_Setup* and before *ATS_GPU_AllocBuffer*

Parameters

- **boardHandle** – Handle to the board.
- **kernelList** – Pointer to a list of kernels to launch.
- **numberOfKernels** – Number of kernels in kernelList
- **bytesPerResultBuffer** – Returns the size of a result buffer

See *ATS-GPU-HPC* for more information on types of kernels that can be launched on the GPU.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_GPU_Setup**(HANDLE boardHandle, U32 channelSelect, long transferOffset, U32 transferLength, U32 recordsPerBuffer, U32 recordsPerAcquisition, U32 autoDMAFlags, U32 ATSGPUFlags)

Prepares the ATS board and GPU for acquisition.

This function calls `AlazarBeforeAsyncRead()` internally and most parameters are passed directly to it. In addition, it sets up the GPU for DMA transfers

Parameters

- **`boardHandle`** – Handle to the board.
- **`channelSelect`** – Channel mask with each channel identifier OR'd
- **`transferOffset`** – pass a negative integer for pretrigger samples
- **`transferLength`** – Number of samples in a record or transfer
- **`recordsPerBuffer`** – Number of records in a buffer, 1 for triggered streaming and continuous streaming modes.
- **`recordsPerAcquisition`** – Total number of records in the acquisition. Pass `0x7FFFFFFF` for infinite.
- **`autoDMAFlags`** – ATSApi flags for AlazarBeforeAsyncRead
- **`ATSGPUFlags`** – Combination of elements from *ATS_GPU_SETUP_FLAG* OR'd together. Pass 0 for default

RETURN_CODE **`ATS_GPU_StartCapture`**(HANDLE boardHandle)

Start the acquisition and data transfer to GPU if required.

Use this function in replacement of AlazarStartCapture(). The application must be ready to call *ATS_GPU_GetBuffer()* to prevent data overflows

Parameters

`boardHandle` – Handle to the board

 ©2008-2025 Alazar Technologies Inc.

enum **ATS_GPU_SETUP_FLAG**

GPU data transfer configuration options.

Values:

enumerator **ATS_GPU_SETUP_FLAG_CPU_BUFFER**

Makes ATS-GPU deliver CPU buffers instead of GPU ones. Useful for debugging

enumerator **ATS_GPU_SETUP_FLAG_DEINTERLEAVE**

De-interleave the data in the returned GPU buffer. Does not apply in conjunction with `ATS_GPU_SETUP_FLAG_CPU_BUFFER`

enumerator **ATS_GPU_SETUP_FLAG_UNPACK**

Unpack the data in the returned GPU buffer. It is required for the allocated buffers to be large enough to accommodate unpacked data. Does not apply in conjunction with `ATS_GPU_SETUP_FLAG_CPU_BUFFER` struct **_InputRange**

Structure used to convert data to volts.

Public Members

float **maxValue**

Maximum input range value in volts.

float **minValue**

Minimum input range value in volts.

 ©2008-2025 Alazar Technologies Inc.

4.5 ATS-GPU-HPC

ATS-GPU-HPC is designed to allow users to create their custom processing pipeline with simple structure declarations. These structures contain all the information necessary for the library to perform the desired kernel.

ATS-GPU-HPC is designed to be used by passing a list of KERNEL structures to `ATS_GPU_SetProcessingPipeline()` from the ATS-GPU-BASE library. Some processing kernels require the installation of additional libraries in order to run.

Instead of passing a list of KERNEL structures, it is also possible to use `ATS_GPU_AddProcessingKernel()` to add a kernel to an internal list of processing kernels that will be used by ATS_GPU_SetProcessingPipeline. If this function is called at least once, input parameters "kernelList" and "numberOfKernels" of function ATS_GPU_SetProcessingPipeline are ignored.

4.5.1 Kernel description

enum **_KERNEL_TYPE**

Types of processing kernels.

Values:

enumerator **KERNEL_TYPE_PRE_FFT**

enumerator **KERNEL_TYPE_FFT**

enumerator **KERNEL_TYPE_POST_FFT**

enumerator **KERNEL_TYPE_OCT**

enumerator **KERNEL_TYPE_OUTPUT**

enumerator **KERNEL_TYPE_AVERAGE_RECORDS**

enumerator **KERNEL_TYPE_CONVERT_TO_VOLTS**

enumerator **KERNEL_TYPE_MULTIPLY_RECORDS**

enumerator **KERNEL_TYPE_BOXCAR_AVERAGING**

enumerator **KERNEL_TYPE_AVERAGE_BUFFERS** struct **_KERNEL**

Struct to pass to ATS_GPU_SetProcessingPipeline. The first parameter is used to describe the type of processing kernel in the union.

Public Members

KERNEL_TYPE **type**

Can be any value of see *_KERNEL_TYPE*.

PRE_FFT_KERNEL **pre_fft**

pre fft

FFT_KERNEL **fft**

fft

POST_FFT_KERNEL **post_fft**

post fft

OCT_KERNEL **oct**

oct

OUTPUT_KERNEL **output**

output kernel

AVERAGE_RECORDS_KERNEL **average_records**

average records kernel

CONVERT_TO_VOLTS_KERNEL **convert_to_volts**

convert to volts kernel

MULTIPLY_RECORDS_KERNEL **multiply_records**

multiply records kernel

BOXCAR_AVERAGING_KERNEL **boxcar_averaging**

boxcar averaging kernel

AVERAGE_BUFFERS_KERNEL **average_buffers**

buffer averaging kernel

 ©2008-2025 Alazar Technologies Inc.

union *_KERNEL*::[anonymous] **[anonymous]**

union

struct **_KERNEL_DATA**

Structure that describes the source of the kernel's input data. Processing kernels require information as to what data is inputted to the kernel. This data can be the raw outputted data from the board or data outputted from a previously ran kernel.

Public Members

int **kernel_id**

Kernel id. Can be any one the the previous kernels. -1 implies using the data outputted from the previous kernel. 0 implies using the raw output data. 1 implies using the data outputted from the first kernel.

int **output_id**

Output id. Can be any one of the outputs from the selected kernel id. 0 based. This parameter is ignored if using data outputted from raw data.

4.5.2 Processing kernels

struct **_PRE_FFT_KERNEL**

Pre FFT kernel that prepares a data for FFT processing. Having ATS-CUDA-OCT is required to run this kernel.

Output data description:

- Number of outputs: 1
- Record length per channel: FFTLength*2
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **FFTLength**

Length of the FFT to perform. Equal to the sum of samplesPerRecordPerChannel and the length of zero padding.

U32 **recordsPerBuffer**

Records per buffer per channel.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

float ***realWindowArray**

Pointer to array of size samplesPerRecord that contains the real part of the window. Passing `null` is equivalent to passing an array filled with ones.

float ***imagWindowArray**

Pointer to array of size samplesPerRecord that contains the imaginary part of the window.Passing `null` is equivalent to passing an array filled with zeros.

struct **_FFT_KERNEL**

FFT kernel does FFT processing. Having ATS-CUDA-OCT is required to run this kernel.

Output data description:

- Number of outputs: 1
- Record length per channel: FFTLength/2
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

 ©2008-2025 Alazar Technologies Inc.

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **FFTLength**

Length of the FFT to perform. Equal to the sum of samplesPerRecordPerChannel and the length of zero padding.

U32 **recordsPerBuffer**

Records per buffer per channel.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

struct **_POST_FFT_KERNEL**

Post FFT kernel does Post FFT processing. Having ATS-CUDA-OCT is required to run this kernel.

Output data description:

- Number of outputs: if Amplitude flag is combined to Log and/or Sqrt flags, there is only one output generated. Otherwise, each of the other flags (raw data, phase, real, imaginary) generates one output per flag.
- Record length per channel: FFTLength/2 (Except for raw data output: record length = samplesPerRecordPerChannel)
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32 (Except for raw data output, where the data type depends on the ATS digitizer. Output data type for most ATS digitizers is uint16.)
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B. When multiple outputs are selected, they are placed in the buffer in the following order: 1.Raw data 2.Amplitude (combined or not with Log and/or Sqrt) 3.Phase 4.Real 5.Imaginary

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **FFTLength**

Length of the FFT to perform. Equal to the sum of samplesPerRecordPerChannel and the length of zero padding.

U32 **recordsPerBuffer**

Records per buffer per channel.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

U32 **OCTFlags**

Defines the types of data outputs to be obtained. This parameter can receive one or more elements of ATS_GPU_OCT_OPTIONS, OR'd with the binary OR operator.

struct **_OCT_KERNEL**

OCT kernel does PreFFT-FFT-PostFFT processing. Having ATS-CUDA-OCT is required to run this kernel.

Output data description:

- Number of outputs: if Amplitude flag is combined to Log and/or Sqrt flags, there is only one output generated. Otherwise, each of the other flags (raw data, phase, real, imaginary) generates one output per flag.
- Record length per channel: FFTLength/2 (Except for raw data output: record length = samplesPerRecordPerChannel)
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32 (Except for raw data output, where the data type depends on the ATS digitizer. Output data type for most ATS digitizers is uint16.)
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B. When multiple outputs are selected, they are placed in the buffer in the following order: 1.Raw data 2.Amplitude (combined or not with Log and/or Sqrt) 3.Phase 4.Real 5.Imaginary

 ©2008-2025 Alazar Technologies Inc.

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **FFTLength**

Length of the FFT to perform. Equal to the sum of samplesPerRecordPerChannel and the length of zero padding.

U32 **recordsPerBuffer**

Records per buffer per channel.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

U32 **OCTFlags**

Defines the types of data outputs to be obtained. This parameter can receive one or more elements of ATS_GPU_OCT_OPTIONS, OR'd with the binary OR operator.

float ***realWindowArray**

Pointer to array of size samplesPerRecordPerChannel that contains the real part of the window. Passing `null` is equivalent to passing an array filled with ones.

float ***imagWindowArray**

Pointer to array of size samplesPerRecordPerChannel that contains the imaginary part of the window. Passing `null` is equivalent to passing an array filled with zeros.

struct **_OUTPUT_KERNEL**

Output kernel combines a list of outputs into a single output buffer.

Output data description:

- Number of outputs: Determined by parameter number_of_inputs.
- Data organization in each buffer: Outputs are placed in the same order as they are passed in parameter input_data_list.

Public Members

KERNEL_DATA ***input_data_list**

List of data buffers to assemble in the output buffer see *_KERNEL_DATA*.

int **number_of_inputs**

Length of input_data_list.

struct **_AVERAGE_RECORDS_KERNEL**

Average Records kernel averages the records in a buffer.

Records are converted to floating points, then summed (co-added) and divided by the number of averages (ratio between RecordsPerBufferIn and RecordsPerBufferOut). The averaged records are converted back to the same data type as input data.

Output data description:

- Number of outputs: 1
- Record length per channel: samplesPerRecordPerChannel
- Record count per channel: recordsPerBufferOut
- Channels: determined by channels passed to channelSelect
- Data type: same data type as input data
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **recordsPerBufferIn**

Number of records in the input GPU buffer.

U32 **recordsPerBufferOut**

Desired number of records in the averaged GPU buffer.

 ©2008-2025 Alazar Technologies Inc.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

struct **_CONVERT_TO_VOLTS_KERNEL**

Convert To Volts kernel converts raw data to float32, and optionally convert the data to volts.

Output data description:

- Number of outputs: 1
- Record length per channel: samplesPerRecordPerChannel
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **recordsPerBuffer**

Records per buffer.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

InputRange ***Ranges**

Pointer to the structure with maximum and minimum input range values in volts for each input channel. See InputRange. If nullptr is passed, just convert data to float.

struct **_MULTIPLY_RECORDS_KERNEL**

Multiply Records kernel multiplies the records by a reference record. The input data must have data type float32.

Output data description:

- Number of outputs: 1
- Record length per channel: samplesPerRecordPerChannel
- Record count per channel: recordsPerBuffer
- Channels: determined by channels passed to channelSelect
- Data type: float32
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

float ***multiplierRecord**

Pointer to array of size samplesPerRecordPerChannel that contains the reference record multiplier

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **recordsPerBuffer**

Records per buffer.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

struct **_BOXCAR_AVERAGING_KERNEL**

Boxcar Averaging kernel does AverageRecords - ConvertToVolts - MultiplyRecords processing.

Output data description:

- Number of outputs: 1
- Record length per channel: samplesPerRecordPerChannel
- Record count per channel: recordsPerBufferOut
- Channels: determined by channels passed to channelSelect
- Data type: float32
- Data organization in each buffer: If two channels are selected, all records from channel A followed by all records from channel B

 ©2008-2025 Alazar Technologies Inc.

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

float ***multiplierRecord**

Pointer to array of size samplesPerRecordPerChannel that contains the boxcar window.

U32 **samplesPerRecordPerChannel**

Samples per record per channel.

U32 **recordsPerBufferIn**

Number of records in the input GPU buffer.

U32 **recordsPerBufferOut**

Desired number of records in the averaged GPU buffer.

U32 **channelSelect**

Select the channel(s) to control. This can be one or more of the channels of ALAZAR_CHANNELS, assembled with the OR bitwise operator.

InputRange ***Ranges**

Pointer to the structure with maximum and minimum input range values in volts for each input channel. See InputRange. If nullptr is passed, just convert data to float.

struct **_AVERAGE_BUFFERS_KERNEL**

Average Buffers kernel averages multiple buffers.

Output data description:

- Number of outputs: 1
- Samples per output: samplesPerBuffer
- Data type: same data type as input data
- Data organization in each buffer: Same as the input buffers

Public Members

KERNEL_DATA **input_data**

Input Data see *_KERNEL_DATA*.

U32 **samplesPerBuffer**

Number of samples per buffer.

U32 **buffersToAverage**

Number of buffers to average.

 ©2008-2025 Alazar Technologies Inc.

CHAPTER

FIVE

ATS-CUDA

The ATS-CUDA SDK provides a framework to allow users to perform simple manipulations on CUDA-enabled GPUs. ATS-CUDA is designed to be used with ATS-GPU-BASE, but can also be used independently. This section of the programmer's guide covers the use of ATS-CUDA.

As with ATS-GPU-BASE, using ATS-CUDA requires expertise in CUDA programming because this involves writing custom CUDA kernels.

It is also essential for programmers to have in-depth knowledge of GPU architecture and parallel programming.

5.1 API Reference

Note: Errors from ATS-CUDA-BASE will be logged in ATS_GPU.log. Relevant information about the error will be logged here and can be useful for debugging. For Windows users log file is located in %TEMP%. For Linux users log file is located in /tmp/.

enum **ATS_CUDA_Input_DataType**

Input data types that can be provided.

Values:

enumerator **ATS_CUDA_INPUT_FORMAT_U8**

enumerator **ATS_CUDA_INPUT_FORMAT_U16**

enumerator **ATS_CUDA_INPUT_FORMAT_FLOAT**

enumerator **ATS_CUDA_INPUT_FORMAT_COMPLEXFLOAT**

enumerator **ATS_CUDA_INPUT_FORMAT_S8**

enumerator `ATS_CUDA_INPUT_FORMAT_S16`

 ©2008-2025 Alazar Technologies Inc.

enum **ALAZAR_PACKING**

Types of data packing.

Values:

enumerator **PACKING_16_BITS_PER_SAMPLE**

enumerator **PACKING_12_BITS_PER_SAMPLE**

enumerator **PACKING_8_BITS_PER_SAMPLE** struct **UNPACK_DEINTERLEAVE_OPTIONS**

Structure used to set up unpacking and deinterleaving kernel used in ATS_CUDA_BaseProcessBuffer().

Public Members

bool **unpack**

Flag to activate unpacking;.

bool **deinterleave**

Flag to activate deinterleaving.

U32 **transferLength**

Number of samples per record per channel.

U32 **recordsPerBuffer**

Number of records per buffer per channel.

U32 **channelCount**

channelCount Number of active channels

ALAZAR_PACKING **input_pack_mode**

A member of ALAZAR_PACKING indicating the data packing mode of input buffer

ALAZAR_PACKING **output_pack_mode**

A member of ALAZAR_PACKING indicating the desired output data packing. Ignored if unpack is set to 0.

ALAZAR_INTERLEAVING **input_interleave**

A member of ALAZAR_INTERLEAVE indication the data interleaving of the input buffer

 ©2008-2025 Alazar Technologies Inc.

void ***ATS_CUDA_AllocCPUBuffer**(U32 bytesPerBuffer)

Allocates page-locked memory on the host computer.

This function is used to allocate host memory and is accessible to the device. Memory can be accessed directly by the device and can be written or read at high bandwidth.

Parameters

bytesPerBuffer – Total number of bytes to allocate per buffer

Returns

This function returns a CPU buffer pointer.

void ***ATS_CUDA_AllocGPUBuffer**(U32 bytesPerBuffer)

Allocates memory on the device.

This function is used to allocate memory on the device.

Parameters

bytesPerBuffer – Total number of bytes to allocate per buffer

Returns

This function returns a GPU buffer pointer.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_AverageRecords**(void *GPUBufferIn, void *GPUBufferOut, cudaStream_t stream, U32 samplesPerRecordPerChannel, U32 recordsPerBufferIn, U32 recordsPerBufferOut, U32 channelCount, *ATS_CUDA_Input_DataType* inputDataType)

Launches on the GPU a kernel to average records in a buffer.

Parameters

- **GPUBufferIn** – Pointer to the GPU buffer to be averaged.
- **GPUBufferOut** – Pointer to the averaged output GPU buffer.
- **stream** – Stream identifier on which processing is to take place.
- **samplesPerRecordPerChannel** – Samples per record per channel.
- **recordsPerBufferIn** – Number of records in the input GPU buffer
- **recordsPerBufferOut** – Desired number of records in the averaged GPU buffer
- **channelCount** – Number of input channels.
- **inputDataType.** – Data type of the input data. This parameter must receive one element of ATS_CUDA_Input_DataType.

RETURN_CODE **ATS_CUDA_BaseProcessBuffer**(void *GPUBufferIn, void *GPUBufferOut, cudaStream_t stream, *UNPACK_DEINTERLEAVE_OPTIONS* opt)

Launches on the GPU a kernel to unpack and/or deinterleave a buffer acquired with an AlazarTech digitizer.

Parameters

- **GPUBufferIn** – Pointer to a GPU buffer to on which to apply unpacking/deinterleaving.
- **GPUBufferOut** – Pointer to a GPU buffer where data is to be outputted.
- **stream** – Stream identifier on which processing is to take place
- **opt** – Structure that defines how the unpacking and deinterleaving kernel is to be configured. See *UNPACK_DEINTERLEAVE_OPTIONS*.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_ConvertToVolts**(void *GPUBufferIn, void *GPUBufferOut, U32 samplesPerRecord, U32 recordsPerBuffer, U32 channelCount, InputRange *Ranges, *ATS_CUDA_Input_DataType* inputDataType, cudaStream_t stream)

Launches on the GPU a kernel to convert raw data in float32, and optionally convert the data to volts.

Parameters

- **GPUBufferIn** – Pointer to the GPU buffer of records to be converted.
- **GPUBufferOut** – Pointer to the GPU buffer of records in float32.
- **samplesPerRecord** – Samples per record.
- **recordsPerBuffer** – Records per buffer.
- **channelCount** – Number of input channels.
- **Ranges** – Pointer to the structure with maximum and minimum input range values in volts for each input channel. See InputRange. If nullptr is passed, just convert data to float.
- **inputDataType** – Data type of the input data. This parameter must receive one element of ATS_CUDA_Input_DataType.
- **stream** – Stream identifier on which processing is to take place.

RETURN_CODE **ATS_CUDA_Copy**(void *destination, void *source, U32 bytesPerBuffer, cudaMemcpyKind kind, cudaStream_t stream)

Copies data from host/device to device/host.

Parameters

- **destination** – Pointer to the destination memory address
- **source** – Pointer to the source memory address
- **bytesPerBuffer** – Size in bytes of the buffer to copy
- **kind** – Type of transfer
- **stream** – Stream identifier on which the copy takes place

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_CopyDeviceToHost**(void *GPUBuffer, void *CPUBuffer, U32 bytesPerBuffer, cudaStream_t stream)

Copies data between host and device.

Parameters

- **GPUBuffer** – Pointer to the GPU source memory address
- **CPUBuffer** – Pointer to the CPU destination memory address
- **bytesPerBuffer** – Size in bytes of the buffer to copy
- **stream** – Stream identifier on which the copy takes place

RETURN_CODE **ATS_CUDA_CopyHostToDevice**(void *GPUBuffer, void *CPUBuffer, U32 bytesPerBuffer, cudaStream_t stream)

Copies data between host and device.

Parameters

- **GPUBuffer** – Pointer to the GPU destination memory address
- **CPUBuffer** – Pointer to the CPU source memory address
- **bytesPerBuffer** – Size in bytes of the buffer to copy
- **stream** – Stream identifier on which the copy takes place

 ©2008-2025 Alazar Technologies Inc.

cudaStream_t **ATS_CUDA_CreateStream**()

Create a synchronous stream.

This function returns a pointer to the new stream identifier.

RETURN_CODE **ATS_CUDA_DestroyStream**(cudaStream_t stream)

Destroys and cleans up an asynchronous stream.

Parameters

stream – Stream identifier.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_FreeCPUBuffer**(void *CPUBuffer)

Frees page-locked memory.

This function is used to free host memory allocated by ATS_CUDA_AllocCPUBuffer().

Parameters

CPUBuffer – Pointer to the memory to free

RETURN_CODE **`ATS_CUDA_FreeGPUBuffer`**(void *GPUBuffer)

Frees memory on the device.

This function is used to free GPU memory allocated by ATS_CUDA_AllocGPUBuffer().

Parameters

`GPUBuffer` – Pointer to the device memory to free

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_GetVersion**(U8 *major, U8 *minor, U8 *revision)

Get ATS-CUDA version number.

Parameters

- **major** – ATS-CUDA major version number.
- **minor** – ATS-CUDA minor version number.
- **revision** – ATS-CUDA revision number.

RETURN_CODE **ATS_CUDA_GetComputeCapability**(U32 deviceIndex, int *major, int *minor)

Function to get the compute capability of specified GPU.

Parameters

- **deviceIndex** – 0-based index to the device.
- **major** – Major compute capability version number.
- **minor** – Minor compute capability version number.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if it failed. See `%TEMP%/ATS_GPU.log` (`/tmp/ATS_GPU.log` under Linux) for more information.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_MultiplyRecords**(void *GPUBufferIn, void *multiplierRecord, void *GPUBufferOut, U32 samplesPerRecord, U32 recordsPerBuffer, *ATS_CUDA_Input_DataType* inputDataType, cudaStream_t stream)

Launches on the GPU a kernel to multiply the records by a reference record.

Parameters

- **GPUBufferIn** – Pointer to the GPU buffer of records to be multiplied.
- **multiplierRecord** – Pointer to the reference record multiplying the records.
- **GPUBufferOut** – Pointer to the multiplication result GPU buffer.
- **samplesPerRecord** – Samples per record.
- **recordsPerBuffer** – Records per buffer.
- **inputDataType** – Data type of the input data. This parameter must receive one element of ATS_CUDA_Input_DataType.
- **stream** – Stream identifier on which processing is to take place.

RETURN_CODE **ATS_CUDA_QueryDeviceCount**(U32 *pDeviceCount)

Function to get the number of available CUDA devices.

Parameters

pDeviceCount – Outputs the number of devices detected on the system.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if the CUDA driver returned an error.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_QueryDeviceName**(U32 deviceIndex, char *deviceName, int maxChars)

Function to get the name of a specific CUDA device.

Parameters

- **deviceIndex** – 0-based index to the device.
- **deviceName** – Char array to output the name of the device.
- **maxChars** – Size of the char array.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if the CUDA driver returned an error.

Returns

`ApiInvalidIndex` if the index provided is greater than the number of platforms or devices available.

RETURN_CODE **ATS_CUDA_QueryDeviceSerialNumber**(U32 deviceIndex, char *deviceSerialNumber, int maxChars)

Function to get the serial number of a specific CUDA device.

Parameters

- **deviceIndex** – 0-based index to the device.
- **deviceName** – Char array to output the serial number of the device.
- **maxChars** – Size of the char array.

Returns

ApiSuccess if it succeeded.

Returns

ApiFailed if the CUDA driver returned an error.

Returns

ApiInvalidIndex if the index provided is greater than the number of platforms or devices available.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_SetComputeDevice**(U32 deviceIndex)

Allows you to specify which GPU should be used to process sample data from a digitizer, if more than one GPU is available.

Parameters

deviceIndex – 0-based index to the device.

Returns

`ApiSuccess` if it succeeded.

Returns

`ApiFailed` if it failed. See `%TEMP%/ATS_GPU.log` (`/tmp/ATS_GPU.log` under Linux) for more information.

RETURN_CODE **ATS_CUDA_StreamSynchronize**(cudaStream_t stream)

Waits for a stream to complete.

This function blocks the host thread until stream has completed all operations.

Parameters

stream – Stream identifier.

 ©2008-2025 Alazar Technologies Inc.

bool **ATS_CUDA_StreamQuery**(cudaStream_t stream)

Queries a synchronous stream for completion status.

This function blocks the host thread until stream has completed all operations.

Parameters

stream – Stream identifier.

Returns

This function returns 1 if all operations in stream have completed.

Returns

This function returns 0 if not.

RETURN_CODE **`ATS_CUDA_SeparateDataFromNPTFooters`**(void *GPUBufferIn, void *GPUDataOut, U32 numberOfRecords, U32 bytesPerFooterBlock, U32 footerBlockStrideBytes, cudaStream_t stream)

Launches on the GPU a kernel to extract the digitized data from buffers containing NPT footers.

Parameters

- **`GPUBufferIn`** – Pointer to a GPU buffer of raw data acquired with an ATS board containing NPT footers.
- **`GPUDataOut`** – Pointer to a GPU buffer where the raw data is to be extracted.
- **`numberOfRecords`** – Number of records in GPUBufferIn.
- **`bytesPerFooterBlock`** – Number of bytes per NPT footer block.
- **`footerBlockStrideBytes`** – Distance in bytes between two consecutive NPT footer blocks.
- **`stream`** – Stream identifier on which processing is to take place.

 ©2008-2025 Alazar Technologies Inc.

RETURN_CODE **ATS_CUDA_ExtractNPTFooters**(void *GPUBufferIn, void *GPUFooters, U32 numberOfRecords, U32 bytesPerFooterBlock, U32 footerBlockStrideBytes, cudaStream_t stream)

Launches on the GPU a kernel to extract NPT footers from buffers containing NPT footers.

Parameters

- **GPUBufferIn** – Pointer to a GPU buffer of raw data acquired with an ATS board containing NPT footers.
- **GPUFooters** – Pointer to a GPU buffer where the NPT footers are to be extracted.
- **numberOfRecords** – Number of records in GPUBufferIn.
- **bytesPerFooterBlock** – Number of bytes per NPT footer block.
- **footerBlockStrideBytes** – Distance in bytes between two consecutive NPT footer blocks.
- **stream** – Stream identifier on which processing is to take place.

 ©2008-2025 Alazar Technologies Inc.

INDEX

Symbols

A

 ©2008-2025 Alazar Technologies Inc.

U