ATS-GPU-OCT Programmer's Guide

Release 3.5.0

AlazarTech

Nov 14, 2017

Contents

1 Introduction

ATS-GPU-OCT provides a framework to allow real-time OCT data processing from AlazarTech PCIe digitizers on a CUDA compatible GPU.

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.

2 Prerequisites

2.1 System requirements

This software requires a PC with a CUDA-compatible GPU, and sufficient CPU resources to supply data to the GPU at the desired data acquisition rate. It also requires a working installation of the same version of ATS-GPU-BASE. 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 Windows and 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 The C++ code was written with Microsoft Visual C++ 2015, and requires Microsoft Visual C++ 2015 or later. Please note that a Community Edition of Visual Studio is available for free. It is fully compatible with our code samples. CMake can also be used to build 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.

3 ATS-GPU-OCT

ATS-GPU-OCT leverages ATS-GPU-BASE to transfer data from an ATS digitizer to a GPU in a highly efficient manner. It then takes care of doing OCT processing on the data before sending it back to the host computer's RAM.

3.1 Usage

Note: Installation of ATS-GPU-OCT generates a Dynamic Link Library (.dll) in ../oct/library/${arch_type}. In order to link ATS-GPU-OCT.dll to your application, you must copy it to /Windows/System32.

ATS-GPU-OCT acquisitions are very similar to standard ATSApi acquisitions. Only the differences are listed here for brevity.

The central function of the ATS-GPU-OCT interface is `ATS_GPU_OCT_Setup()`. This function calls its ATS-GPU-BASE counterpart `ATS_GPU_Setup()` internally, which in turns calls `AlazarBeforeAsyncRead()`. It takes a few extra parameters:

- `OCTFlags`: Used to define which data type, such as amplitude and phase, to obtain from the acquisition.
- `FFTLength`: This is used to select the length of the Fourier transform done on the GPU. This value must be a power of two, and it also must be equal to or larger than the record length.

```
rc = ATS_GPU_OCT_Setup(
         boardHandle, channelMask, -(int) preTriggerSamples,
         samplesPerRecordPerChannel, recordsPerBuffer,
         buffersPerAcquisition * recordsPerBuffer, autoDMAFlags,
         OCTOptions, FFTLength, NULL, &fftBytesPerBuffer);
```

We then choose the window function applied to the acquired data before the FFT processing phase. The most common usage pattern is to first generate a window function using `ATS_GPU_OCT_GenerateWindowFunction()`, then to download it to the board using `ATS_GPU_OCT_SetWindowFunction()`. It is possible however to use entirely custom window functions instead of the ones generated by the API. It is also possible to use complex window functions by way of downloading two arrays of points: the first for the real part of the window and the other for the imaginary one.

```
rc = ATS_GPU_OCT_GenerateWindowFunction(
            FFT_WINDOW_HANNING, &window[0],
            samplesPerRecordPerChannel);
// Error handling

rc = ATS_GPU_OCT_SetWindowFunction(
            boardHandle, samplesPerRecordPerChannel,
            &window[0], NULL);
// Error handling
```

We then allocate memory on the GPU and CPU for data to be transferred to, and we post those buffers to the board. For this purpose, we use `ATS_GPU_OCT_AllocBuffer()`. This function allocates buffers on the GPU, and sets up all the intermediary state necessary for ATS-GPU-OCT to successfully transfer data. It also allocates data on the CPU to received the processed OCT data.

```
for (int i = 0; i < numberOfBuffers; i++)
{
    buffers[i] = (float*) ATS_GPU_OCT_AllocBuffer(
        boardHandle, bytesPerResultBuffer, NULL);

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

We can then start the acquisition with `ATS_GP_OCT_StartCapture()`. Once acquisition is started, `ATS_GPU_OCT_GetBuffer()` must be called as often as possible to retrieve a buffer containing processed data on the CPU. The data can then be used by the calling application. When no longer needed, the buffer needs to be posted back.

```
for (size_t i; i < buffers_per_acquisition; i++)
{
    rc = ATS_GPU_OCT_GetBuffer(
        boardHandle, buffers[bufferIndex],timeout_ms);
    // Error handling

    // TODO: Process sample data in this buffer.
```

```
    rc = ATS_GPU_OCT_PostBuffer(
        boardHandle, buffers[bufferIndex], bytesPerResultBuffer);
    // Error handling
}
```

When acquisition is complete, `ATS_GPU_OCT_AbortCapture()` must be called. Buffers allocated with `ATS_GPU_OCT_AllocBuffer()` should then be freed with `ATS_GPU_OCT_FreeBuffer()`.

```
ATS_GPU_OCT_AbortCapture(boardHandle);

if (gpuFile != NULL)
    fclose(gpuFile);

// Free buffers
for (int i = 0; i < numberOfBuffers; i++) {
    ATS_GPU_OCT_FreeBuffer(boardHandle, buffers[i]);
}
```

3.2 API Reference

enum ATS_GPU_OCT_OPTIONS

Types of data output that are generated by the acquisition. This is used in *ATS_GPU_OCT_Setup()*

Values:

ATS_GPU_OCT_LOG_OUTPUT = 1 << 0

ATS_GPU_OCT_AMPLITUDE_OUTPUT = 1 << 1

ATS_GPU_OCT_PHASE_OUTPUT = 1 << 2

ATS_GPU_OCT_REAL_OUTPUT = 1 << 3

ATS_GPU_OCT_IMAG_OUTPUT = 1 << 4

enum ATS_GPU_OCT_WINDOWS

Window functions that can be generated by *ATS_GPU_OCT_GenerateWindowFunction()*

Values:

FFT_WINDOW_NONE = 0

FFT_WINDOW_HANNING

FFT_WINDOW_HAMMING

FFT_WINDOW_BLACKMAN

FFT_WINDOW_BLACKMAN_HARRIS

FFT_WINDOW_BARTLETT

NUM_FFT_WINDOW_ITEMS

RETURN_CODE **ATS_GPU_OCT_AbortCapture**(HANDLE *boardHandle*)

Stops the acquisition.

Aborts an acquisition, stops data processing, and releases resources allocated by ATS_GPU_BeforeCapture().

Return ApiSuccess

Parameters

- `boardHandle`: Handle to the board

void *__ATS_GPU_OCT_AllocBuffer__(HANDLE *boardHandle*, U32 *bytesPerBuffer*, void **reserved*)
Allocates page-aligned pinned memory for ATS and GPU boards.

This function can be called after *ATS_GPU_OCT_Setup()* to perform the necessary memory allocations. This function returns a CPU result buffer pointer.

Parameters

- `boardHandle`: Handle to the board
- `bytesPerBuffer`: Total number of bytes to allocate per buffer
- `reserved`: Pass NULL.

RETURN_CODE **ATS_GPU_OCT_EnableVerificationMode**(BOOL *enable*, U32 *boardType*)

Enable verification mode to supply already acquired data.

Parameters

- `enable`: Pass 1 to enable
- `boardType`: Board identifier used to perform the acquisition.

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

Free buffers allocated with ATS_GPU_OCT_AllocBuffers();.

Parameters

- `boardHandle`: Handle to the board
- `buffer`: Buffer pointer allocated by ATS_GPU_AllocBuffers()

RETURN_CODE **ATS_GPU_OCT_GenerateWindowFunction**(U32 *windowType*, float **window*, U32 *windowLength_samples*)

Generate a window function for FFT.

Parameters

- `windowType`: A member of the *ATS_GPU_OCT_WINDOWS* enum
- `window`: A pointer to a preallocated array where the window will be written.
- `windowLength_samples`: Number of points in the window

RETURN_CODE **ATS_GPU_OCT_GetBuffer**(HANDLE *boardHandle*, void **buffer*, U32 *timeout_ms*)

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.

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

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

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

Return `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.

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

Return `ApiFailed` if a system of internal error occured. Call ATS_GPU_GetLastErrorText() for more information.

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.

RETURN_CODE **ATS_GPU_OCT_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_OCT. Buffers posted must have previously been allocated with ATS_GPU_AllocBuffers.

Parameters

- `boardHandle`: Hande 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.

RETURN_CODE **ATS_GPU_OCT_SetBuffer**(void **dataInputBuffer*, void **CPUResultBuffer*, U32 *samplesPerBuffer*)

Supply a buffer for verification mode.

Parameters

- `dataInputBuffer`: Pointer to data buffer to be processed
- `CPUResultBuffer`: Pointer to data buffer to contain result data
- `samplesPerBuffer`: Size in samples of the buffer

RETURN_CODE **ATS_GPU_OCT_SetWindowFunction**(HANDLE *boardHandle*, U32 *samplesPerRecord*, float **realWindowArray*, float **imagWindowArray*)

Set window function used in FFT calculation.

Parameters

- `boardHandle`: Handle to the board
- `samplesPerRecord`: Length of the window, equal to the number of samples per FFT.
- `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.
- `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.

RETURN_CODE **ATS_GPU_OCT_Setup**(HANDLE *boardHandle*, U32 *channelSelect*, long *transferOffset*, U32 *samplesPerFFT*, U32 *FFTsPerBuffer*, U32 *FFTsPerAcquisition*, U32 *autoDMAFlags*, U32 *OCTFlags*, U32 *FFTLength*, void **reserved*, U32 **bytesPerResultBuffer*)

Prepares the ATS board and GPU for acquisition.

This function calls `ATS_GPU_Setup()` internally and most parameters are passed directly to it. In addition, it sets up the GPU for DMA transfers and receives options specific to OCT processing.

Parameters

- `boardHandle`: Handle to the board. Set to `NULL` for data validation mode.
- `channelSelect`: Channel mask with each channel identifier OR'd
- `transferOffset`: Pass a negative integer for pretrigger samples
- `samplesPerFFT`: Number of samples in a record or transfer
- `FFTsPerBuffer`: Number of records in a buffer, 1 for triggered streaming and continuous streaming modes.
- `FFTsPerAcquisition`: In this version of the library, it is required to pass `0x7FFFFFFF` to this parameter, which stands for an infinite acquisition. It is possible to interrupt the acquisition at any time using *ATS_GPU_OCT_AbortCapture()*
- `autoDMAFlags`: ATSApi flags for `AlazarBeforeAsyncRead`
- `OCTFlags`: Defines the types of data outputs to be obtained from the OCT acquisition. This parameter can receive one or more elements of *ATS_GPU_OCT_OPTIONS*, or'ed with the binary OR operator.
- `FFTLength`: Length of FFT, should be a power of 2.
- `reserved`: Pass NULL
- `bytesPerResultBuffer`: Returns the size of a result buffer

RETURN_CODE **ATS_GPU_OCT_StartCapture**(HANDLE *boardHandle*)

Start the acquisition.

Use this function in replacement of AlazarStartCapture(). It starts the acquisition. The application must be ready to call *ATS_GPU_OCT_GetBuffer()* to prevent data overflows

Parameters

- `boardHandle`: Handle to the board

Index