ATS-GMA-OCT Programmer's Guide

Release 4.0.0

AlazarTech

Apr 16, 2018

Contents

1 Introduction

ATS-GMA-OCT provides a framework to allow real-time OCT data processing from AlazarTech PCIe digitizers on a compatible AMD Radeon Pro 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 an AMD-compatible GPU. It was tested with Radeon Pro WX7100 (Polaris) and Radeon Pro WX9100 (Vega). A modern chipset (X99, X299) will greatly improve transfer speed and overall performance.

Supported operating systems 64 bit Windows 7 and 10 operating system are supported.

Supported AlazarTech drivers ATS-GMA-OCT requires driver version 6.1 and above.

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.

Compatible GPUs ATS-GMA is designed to be compatible with AMD Radeon Pro GPUs using AMD APP SDK version 2.9 and higher. It should be noted that the current version of ATS-GMA supports only one GPU at a time. If multiple GPUs are installed in the computer, ATS-GMA will let you select one of them.

DirectGMA To use ATS-GMA, you must first enable DirectGMA on your GPU. By default, the DirectGMA configuration is disabled on AMD GPUs. To enable DirectGMA on your GPU, you must connect the monitor directly to the specific GPU, without any remote connection. You must then open AMD FirePro Settings and go to Advanced Parameters. This will open the AMD FirePro Control Center. Under the tab SDI/DirectGMA you can enable DirectGMA. Choose the maximum addressable window size. For more information, visit the FirePro DirectGMA website.

3 ATS-GMA-OCT

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

ATS-GMA-OCT acquisitions are very similar to standard ATSApi acquisitions. For brevity, only the differences are listed here.

The central function of the ATS-GMA-OCT interface is `ATS_GMA_OCT_Setup()`. This function calls its ATS-GMA-BASE counterpart `ATS_GMA_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 2, 3, 5, 7, 11, 13 or a combination of those, and it also must be equal to or larger than the record length.
- `clDevice`: A pointer to the openCL device, must also be provided. This pointer should be previously determined with `ATS_GMA_GetComputeDevice()`.

```
rc = ATS_GMA_Setup(boardHandle, channelSelect, -preTriggerSamples,
                    samplesPerRecordPerChannel, recordsPerBuffer,
                    recordsPerAcquisition, FFTLength, autoDMAFlags,
                    ATSGMAFlags, OCTFlags, &clContext, &clDevice);
```

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_GMA_OCT_GenerateWindowFunction()`, then to download it to the board using `ATS_GMA_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_GMA_OCT_GenerateWindowFunction(FFT_WINDOW_HANNING,
                                        &window[0],
                                        samplesPerRecordPerChannel);
    // Error handling

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

We then allocate memory on the GPU for data to be transferred to. For this purpose, we use `ATS_GMA_OCT_AllocBuffer()`. This function allocates a buffer on the GPU, and sets up all the intermediary states necessary for ATS-GMA-OCT to successfully transfer data. We then post those buffers to the board using `ATS_GMA_OCT_PostBuffer()`.

```
for (int i = 0; i < numberOfBuffers; i++)
{
            BufferArray[i] = ATS_GMA_OCT_AllocBuffer(boardHandle,
                                                     bytesPerResultBuffer);
}

for (int i = 0; i < numberOfGMABuffers; i++)
{
            rc = ATS_GMA_OCT_PostBuffer(boardHandle,
```

```
                                        BufferArray[i]);
            // Error handling
}
```

We can then start the acquisition with `ATS_GMA_OCT_StartCapture()`. Once acquisition is started, `ATS_GMA_OCT_GetBuffer()` must be called as often as possible to retrieve a buffer containing data already copied on the GPU. A `userBuffer` contains the GPU data after unpacking and deinterleaving. clQueue points to a queue created by the library on which FFT processing will be done for this specific buffer. Use this queue for custom processing, if required. The data can then be used to do FFT processing. FFT processing is divided into three intermediary steps :

- `ATS_GMA_OCT_PreFFT()` performs windowing and padding of input data in preparation for Fast Fourier Transform.
- `ATS_GMA_OCT_FFT()` performs Fast Fourier Transform using the clFFT library.
- `ATS_GMA_OCT_PostFFT()` takes de FFT and outputs the desired data. The output is set using `ATS_GMA_OCT_OPTIONS`.

When no longer needed, the buffer needs to be posted back. Finally, `ATS_GMA_OCT_ReadOutputBuffer()` is used to send processed buffer back in host memory.

```
for (int i = 0 ; i < buffersPerAcquisition; i++)
{
    rc =  ATS_GMA_OCT_GetBuffer(boardHandle,
                                BufferArray[i],
                                &userBuffer,
                                &clQueue,
                                timeout_ms);
    // Error handling
    rc = ATS_GMA_OCT_PreFFT(boardHandle,
                            &ironOut_output,
                            &sampleToComplex_Output,
                            NULL,
                            NULL);
    // Error handling
            rc = ATS_GMA_OCT_FFT(boardHandle,
                                 &sampleToComplex_Output,
                                 &fft_output,
                                 NULL,
                                 NULL);
    // Error handling
            rc = ATS_GMA_OCT_PostFFT(boardHandle,
                                     &fft_output,
                                     &complexToResult_output,
                                     NULL,
                                     NULL);
    // Error handling
            rc = ATS_GMA_OCT_ReadOutputBuffer(boardHandle,
                                              &complexToResult_output,
                                              &pHostBuffer,
                                              NULL,
                                              NULL)
```

```
    // Error handling
}
```

When acquisition is complete, buffers allocated with `ATS_GMA_AllocBuffer()` should be freed with `ATS_GMA_OCT_FreeBuffer()`. `ATS_GMA_OCT_AbortCapture()` must then be called.

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

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

3.2 Benchmarks

Performance benchmarks using ATS-GMA-OCT on an Asus X99 Deluxe motherboard using an ATS9373 (8 lane PCIe Gen 3) and acquired in NPT mode :

GMA buffer size (MB)	FFT Length	FFTs per second (x1000)	
		Radeon Pro WX7100	Radeon Pro WX9100
1	2048	1700	1700
	4096	850	850
	8192	325	375
4	2048	1900	1900
	4096	950	950
	8192	350	485
16	2048	1900	1900
	4096	950	950
	8192	350	485

3.3 ATS_GMA_OCT.log file

ATS-GMA-OCT logs relevant information about specific ATS-GMA-OCT calls in ATS_GMA_OCT.log. Most importantly ATS_GMA_OCT.log stores information concerning errors occuring with functions from ATS-GMA-OCT. Also refer to ATS_GMA.log for errors related to ATS-GMA.

3.4 API Reference

enum ATS_GMA_OCT_OPTIONS

Types of data output that are generated by the acquisition. This is used in *ATS_GMA_OCT_Setup()*. Several output options can be chosen and output data will be placed in the same output buffer, in sequential order.

Values:

ATS_GMA_OCT_LOG_OUTPUT = 1 << 0

ATS_GMA_OCT_AMPLITUDE_OUTPUT = 1 << 1

ATS_GMA_OCT_PHASE_OUTPUT = 1 << 2

ATS_GMA_OCT_REAL_OUTPUT = 1 << 3

ATS_GMA_OCT_IMAG_OUTPUT = 1 << 4

enum ATS_GMA_OCT_WINDOWS

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

Values:

FFT_WINDOW_NONE

FFT_WINDOW_HANNING

FFT_WINDOW_HAMMING

FFT_WINDOW_BLACKMAN

FFT_WINDOW_BLACKMAN_HARRIS

FFT_WINDOW_BARTLETT

NUM_FFT_WINDOW_ITEMS

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

Generate a window function for FFT.

Parameters

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

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

Set window function on the GPU 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 zeros.
- `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_GMA_OCT_Setup**(HANDLE *boardHandle*, U32 *channelSelect*, long *transferOffset*, U32 *transferLength*, U32 *recordsPerBuffer*, U32 *recordsPerAcqusition*, U32 *FFTLength*, U32 *autoDMAFlags*, U32 *ATSGMAFlags*, U32 *OCTFlags*, cl_context **clContext*, cl_device_id **clDevice*)

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 GMA transfers.

Return `ApiSuccess` if it succeeded

Return An error code if it failed. See error list in the ATS-SDK manual, ATS_GMA.log and ATS_GMA_OCT.log for more information.

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.
- `FFTLength`: Length of the Fourier transform done on the GPU. This value must be a power of 2, 3, 5, 7, 11, 13 or a combination of those. It also must be equal to or larger than the record length.
- `autoDMAFlags`: ATSApi flags for AlazarBeforeAsyncRead.
- `ATSGMAFlags`: ATS-GMA specific flags. See ATS_GMA_SETUP_FLAG in ATS-GMA-BASE documentation. In case of multiple channel acquisitions or data acquired in other than 16-bits packing, ATS_GMA_SETUP_FLAG_DEINTERLEAVE and/or ATS_GMA_SETUP_FLAG_UNPACK must be activated.
- `OCTFlags`: Defines the type of data output to be obtained from the OCT acquisition. Can receive one or several element of *ATS_GMA_OCT_OPTIONS*.
- `clContext`: Pointer to an OpenCL context. Pass the address of an uninitialized OpenCL context. This function will create a context internally and assign it to the variable passed. If ATS_GMA_SETUP_FLAG_USER_DEFINED_CONTEXT is passed in `ATSGMAFlags`, pass the address of the custom context you would like the library to use.
- `clDevice`: Pointer to the OpenCL device used for acquisition. See `ATS_GMA_GetComputeDevice()`.

cl_mem **ATS_GMA_OCT_AllocBuffer**(HANDLE *boardHandle*, **const** U32 *bytesPerBuffer*)

Allocates GPU memory suitable to be used for a True DMA data transfer.

This function must be called after `ATS_GMA_OCT_Setup()` to perform the necessary memory allocations. This function returns a GPU buffer. The number of allocated GMA buffers multiplied by the size of each buffers in bytes must be inferior to 128MB, which is the maximal window size that can be allocated for DirectGMA. This window size was specified while enabling DirectGMA.

Return Returns a `cl_mem` GPU buffer.

Parameters

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

RETURN_CODE **ATS_GMA_OCT_PostBuffer**(HANDLE *boardHandle*, cl_mem *GpuBuffer*)

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

This function calls `AlazarPostAsyncBuffer()` internally. Buffers posted must have previously been allocated with `ATS_GMA_OCT_AllocBuffer()`. It also acts as a synchronization point for buffer acquisition and processing.

Return `ApiSuccess` if it succeeded

Return An error code if it failed. See error list in the ATS-SDK manual, ATS_GMA.log and ATS_GMA_OCT.log for more information.

Parameters

- `boardHandle`: Hande to the board.
- `GpuBuffer`: GPU buffer allocated by `ATS_GMA_OCT_AllocBuffer()`.

RETURN_CODE **ATS_GMA_OCT_GetBuffer**(HANDLE *boardHandle*, cl_mem *GpuInputBuffer*, cl_mem **GpuOutputBuffer*, cl_command_queue **clQueue*, U32 *timeout_ms*, cl_event **endProcessingEvent*)

Get buffers on the GPU.

This function calls `AlazarWaitAsyncBufferComplete()` internally. This function must be called at average rate that is equal to or greater than the rate at which GMA buffers complete. Every time a buffer is retrieved using *ATS_GMA_OCT_GetBuffer()*, it must be posted back to the board using *ATS_GMA_OCT_PostBuffer()*. Processing kernels (PreFFT, FFT, PostFFT, etc.) and read (ReadOutputBuffer) should be called after this function.

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

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

Return `ApiInvalidHandle` if 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.

Parameters

- `boardHandle`: Handle to the board.
- `GpuInputBuffer`: GPU buffer allocated by `ATS_GMA_OCT_AllocBuffer()`.
- `GpuOutputBuffer`: Pointer to the unpacked and deinterleaved `cl_mem` buffer.
- `clQueue`: Pointer to a OpenCL command queue created by the library on which processing occurs for a specific buffer.
- `timeout_ms`: Time the board will wait for a trigger before returning ApiWaitTimeout.
- `endProcessingEvent`: Event indicating the end of unpacking and deinterleaving.

RETURN_CODE **ATS_GMA_OCT_StartCapture**(HANDLE *boardHandle*)

Starts the acquisition.

This function calls `AlazarStartCapture()` internally.

Parameters

- `boardHandle`: Hande to the board.

RETURN_CODE **ATS_GMA_OCT_PreFFT**(HANDLE *boardHandle*, cl_mem **GpuInputBuffer*, cl_mem **GpuOutputBuffer*, cl_event **startProcessingEvent*, cl_event **endProcessingEvent*)

Prepares the buffer for Fast Fourier Transform calculation. All processing and reading calls should be placed after getting a buffer with *ATS_GMA_OCT_GetBuffer()* and before posting it back to the board with *ATS_GMA_OCT_PostBuffer()*.

This function is designed to accept buffers coming from *ATS_GMA_OCT_GetBuffer()*. It is however possible to perform custom processing on this buffer before if needed, and pass the pointer to the newly processed buffer as input. However, the size and the data format (16-bits) must remain the same.

This function launches a kernel on the GPU to prepare data before doing the FFT. Windowing of each record in the buffer is performed depending on the window type used in *ATS_GMA_OCT_SetWindowFunction()*.

If FFT length is greater than the number of samples per records, a padding of the FFT occurs.

ATS_GMA_OCT_FFT() requires complex input buffers. Therefore, the output from *ATS_GMA_OCT_PreFFT()* will have real and complex data interleaved.

Parameters

- `boardHandle`: Hande to the board.
- `GpuInputBuffer`: Pointer to a `cl_mem` buffer outputted from *ATS_GMA_OCT_GetBuffer()*. Buffer must contain de-interleaved channels and 16-bits packing. Refer to ATSGMAFlags to get appropriate buffer format.
- `GpuOutputBuffer`: Pointer to the FFT ready `cl_mem` buffer. The size of the `GpuOutputBuffer` will be the FFTLength x recordsPerBuffer x numberOfChannels x 2, to account for real and complex interleaving. Data type is single precision floating point.
- `startProcessingEvent`: Event on which the `PreFFT` kernel will wait for before launching.
- `endProcessingEvent`: Event indicating the end of `PreFFT` processing.

RETURN_CODE **ATS_GMA_OCT_FFT**(HANDLE *boardHandle*, cl_mem **GpuInputBuffer*, cl_mem **GpuOutputBuffer*, cl_event **startProcessingEvent*, cl_event **endProcessingEvent*)

Perform Fast Fourier Transform. All processing and reading calls should be placed after getting a buffer with *ATS_GMA_OCT_GetBuffer()* and before posting it back to the board with *ATS_GMA_OCT_PostBuffer()*.

This function is designed to accept buffers coming from *ATS_GMA_OCT_PreFFT()*. It is however possible to perform custom processing on this buffer before if needed, and pass the pointer to the newly processed buffer as input. However, the size, FFT length and the data format (float) must remain the same.

This function launches a kernel to perform the FFT on each record within the `GpuInputBuffer` using the clFFT library.

Parameters

- `boardHandle`: Hande to the board.
- `GpuInputBuffer`: Pointer to a `cl_mem` buffer outputted from *ATS_GMA_OCT_PreFFT()*.
- `GpuOutputBuffer`: Pointer to a `cl_mem` buffer after FFT. Has the same format and size as `GpuInputBuffer`.
- `startProcessingEvent`: Event on which the `FFT` kernel will wait for before launching.
- `endProcessingEvent`: Event indicating the end of `FFT` processing.

RETURN_CODE **ATS_GMA_OCT_PostFFT**(HANDLE *boardHandle*, cl_mem **GpuInputBuffer*, cl_mem **GpuOutputBuffer*, cl_event **startProcessingEvent*, cl_event **endProcessingEvent*)

Output the desired output from Fast Fourier Transform. All processing and reading calls should be placed after getting a buffer with *ATS_GMA_OCT_GetBuffer()* and before posting it back to the board with *ATS_GMA_OCT_PostBuffer()*.

This function is designed to accept buffers coming from *ATS_GMA_OCT_FFT()*. It is however possible to perform custom processing on this buffer before if needed, and pass the pointer to the newly processed buffer as input. However, the size, FFT length and the data format (float) must remain the same.

This function launches a kernel to output the data as specified by ATS_GMA_OCT_OPTIONS. If

Parameters

- `boardHandle`: Hande to the board.
- `GpuInputBuffer`: Pointer to a `cl_mem` buffer outputted from *ATS_GMA_OCT_FFT()*.
- `GpuOutputBuffer`: Pointer to a `cl_mem` buffer after PostFFT. The size of the `GpuOutputBuffer` will be the (FFTLength / 2) x recordsPerBuffer x numberOfChannels x numberOfOutputs. Data type is single precision floating point.
- `startProcessingEvent`: Event on which the `PostFFT` kernel will wait for before launching.
- `endProcessingEvent`: Event indicating the end of `PostFFT` processing.

RETURN_CODE **ATS_GMA_OCT_ReadOutputBuffer**(HANDLE *boardHandle*, cl_mem **GpuBuffer*, float ***hostBuffer*, cl_event **startReadEvent*, cl_event **endReadEvent*)

Sends the output data buffer back to host memory. All processing and reading calls should be placed after getting a buffer with *ATS_GMA_OCT_GetBuffer()* and before posting it back to the board with *ATS_GMA_OCT_PostBuffer()*.

This function launches a read on GPU memory in a highly efficient manner. In order to maximize transfer speed between GPU and host memory, the size of the host buffer must be pre-set. Therefore, it has the format and size of `GpuOutputBuffer` of *ATS_GMA_OCT_PostFFT()*.

Parameters

- `boardHandle`: Hande to the board.
- `GpuBuffer`: Pointer to a `cl_mem` buffer outputted from *ATS_GMA_OCT_PostFFT()*
- `hostBuffer.`: Pointer to a `float` host pointer to receive GPU buffer.
- `startProcessingEvent`: Event on which the `ReadOutputBuffer` kernel will wait for before launching.
- `endProcessingEvent`: Event indicating the end of `ReadOutputBuffer` reading.

RETURN_CODE **ATS_GMA_OCT_FreeBuffer**(HANDLE *boardHandle*, cl_mem *GpuBuffer*)
Frees a buffer allocated with `ATS_GMA_OCT_AllocBuffer()`

Return `ApiSuccess` if it succeeded

Return An error code if it failed. See error list in the ATS-SDK manual, ATS_GMA.log and ATS_GMA_OCT.log for more information.

Parameters

- `boardHandle`: Handle to the board.
- `GpuBuffer`: GPU buffer allocated by `ATS_GMA_OCT_AllocBuffer()`.

RETURN_CODE **ATS_GMA_OCT_AbortCapture**(HANDLE *boardHandle*)

Stops the acquisition.

Aborts an acquisition, stops data processing, and releases resources allocated by `ATS_GMA_OCT_Setup()`

Return `ApiSuccess` if it succeeded

Return An error code if it failed. See error list in the ATS-SDK manual and ATS_GMA.log and ATS_GMA_OCT.log for more information.

Parameters

- `boardHandle`: Handle to the board.

Index

A

F

N