ATS-GMA-BASE Programmer's Guide

Release 4.0.0

AlazarTech

Apr 16, 2018

Contents

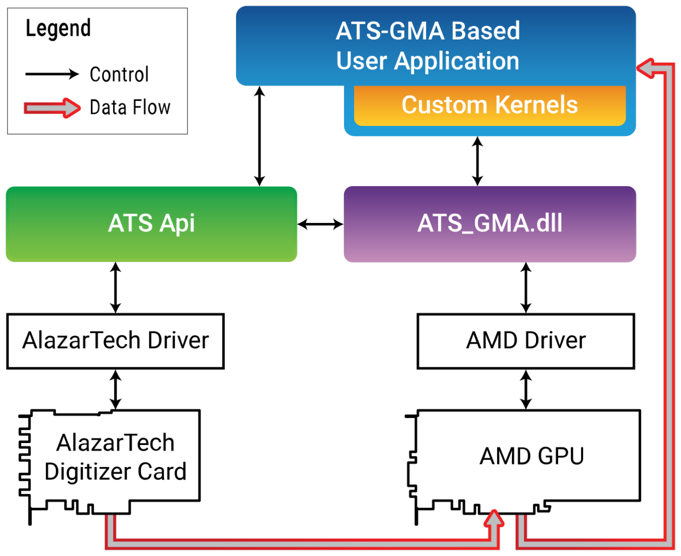
1 Introduction

The ATS-GMA SDK provides a framework to allow direct memory accessing of data from AlazarTech PCIe digitizers on a AMD Radeon Pro GPU. This programmer's guide covers the use of ATS-GMA-BASE.

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 OpenCL programming is assumed, because using ATS-GMA-BASE involves OpenCL programming.

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



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-BASE 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.

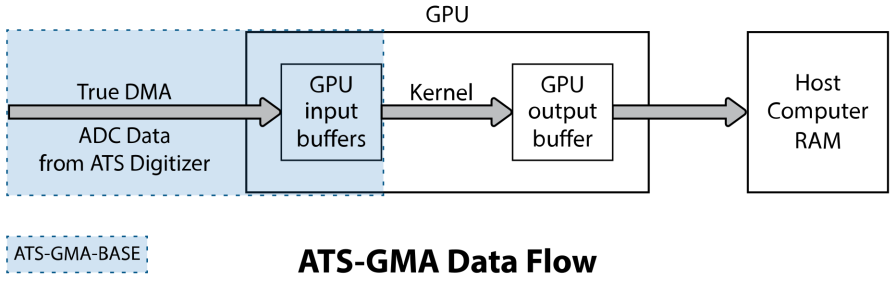
Compatible GPUs ATS-GMA is designed to be compatible with AMD Radeon Pro GPUs using AMD APP SDK version 2.9 and higher. It should be noted that the current version of ATS-GMA supports only one GPU at a time. If multiple GPUs are installed in the computer, ATS-GMA will let you select one of them.

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-BASE

ATS-GMA-BASE is designed to use True DMA technology to transfer data directly from an ATS PCI express digitizer to a GPU for processing, without using any temporary storage in host memory. The user application, which includes custom OpenCL kernels, can then access data buffers on the GPU. The user is then responsible to perform data processing and copy data back to the host memory (RAM) if required. A code example is provided as an example of a user application that performs very simple signal processing (data inversion).

3.1 Usage



ATS-GMA Data Flow

ATS-GMA-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 sample defines 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 OpenCL-enabled GPU to use for the data transfer using `ATS_GMA_GetComputeDevice()`. If more than one OpenCL device is available, you must select a specific one using *deviceIndex*. Output parameter is a pointer to a specific device ID. This OpenCL device pointer will be required to build programs. Note that if DirectGMA is not enabled for the chosen GPU, this function will return a *VerifyExtension()* error.

```
rc = ATS_GMA_GetComputeDevice(boardHandle, deviceIndex, &clDevice);
// Error handling
```

We must then setup parameters of the acquisition to GPU with `ATS_GMA_Setup()`. 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-GMA acquisitions.

```
rc = ATS_GMA_Setup(boardHandle, channelSelect, -preTriggerSamples,
                   samplesPerRecordPerChannel, recordsPerBuffer,
                   recordsPerAcquisition, autoDMAFlags, ATSGMAFlags,
                   &clContext, NULL);
// Error handling
```

`ATSGMAFlags` is used to specify if data unpacking and/or deinterleaving is to be performed on the GPU. It can also be used to specify if a user defined context is used. If no user defined context flags is specified, the function returns a pointer to an OpenCL context. This parameter will allow the user to prepare custom OpenCL programs and kernels as well as create OpenCL memory buffers.

Following this setup, the user is responsible for creating and building specific programs as well as creating kernels on the OpenCL specific context returned by `ATS_GMA_Setup()`. The user is also responsible for creating OpenCL buffers for kernel inputs and outputs. These steps are done using the OpenCL APIs. The provided sample shows the user how to build such programs and kernels.

We then allocate memory on the GPU for data to be transferred to. For this purpose, we use `ATS_GMA_AllocBuffer()`. This function allocates a buffer on the GPU, and sets up all the intermediary states necessary for ATS-GMA to successfully transfer data. We then post those buffers to the board using `ATS_GMA_PostBuffer()`. Please note that if you'd 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 (int i = 0; i < numberOfGMABuffers; i++)
{
      BufferArray[i] = ATS_GMA_AllocBuffer(boardHandle,
                                           bytesPerBuffer);
      // Error handling
}
for (int i = 0; i < numberOfGMABuffers; i++)
{
      rc = ATS_GMA_PostBuffer(boardHandle,
                              BufferArray[i]);
      // 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_GMA_StartCapture(HANDLE boardHandle);
// Error handling
```

Once the acquisition is started, `ATS_GMA_GetBuffer()` must be called as often as possible to retrieve a buffer containing data already copied on the GPU. `userBuffer` contains the GPU data after unpacking and deinterleaving. This buffer can then be processed by your custom kernel on the GPU. clQueue points to a queue created by the library for unpacking and deinterleaving for the specific buffer. This queue can be used to perform custom processing. 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 (int i = 0 ; i < buffersPerAcquisition; i++)
{
    rc =  ATS_GMA_GetBuffer(boardHandle,
                            BufferArray[i],
                            &userBuffer,
                            &clQueue,
                            timeout_ms,
                            &endProcessingEvent);
    // Error handling

    // TODO: Process buffer. This is where you can call your own processing
    //       function that launches the GPU kernels, such as with the OpenCL function
    //       clEnqueueNDRangeKernel()

    clWaitForEvents(1, &endProcessingEvent);

    rc = ATS_GMA_PostBuffer(boardHandle, buffer);
    // Error handling
}
```

User is responsible for synchronizing data acquisition and processing, if required. If data processing requires more time than it takes to acquire it, synchronization will be required in order to process the corresponding incoming buffer. When acquisition is complete, buffers allocated with `ATS_GMA_AllocBuffer()` should be freed with `ATS_GMA_FreeBuffer()`. `ATS_GMA_AbortCapture()` must then be called.

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

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

Here is an example of what the data inversion kernel on the GPU can look like:

```
__kernel void Process(__global const ushort *bufferIn,
                  __global ushort *bufferOut) {
const int sampleInBuffer = get_global_id(0);
bufferOut[sampleInBuffer] = 65535 - bufferIn[sampleInBuffer];
}
```

3.2 Performance guidelines

To maximize transfer rates from the ATS digitizer to the AMD-compatible GPU, a modern chipset with PCIe Gen 3 connectors is highly recommended (X99, X299, etc.).

Also, optimization of OpenCL kernels will affect the overall performance. For more demanding computation processes on the GPU, we recommend using higher end GPUs.

3.3 Benchmarks

Performance benchmarks using ATS-GMA-BASE on a Asus X99 Deluxe motherboard:

PCIe Link Speed	GPU	Transfer Rate
Gen 3: ATS9373	Radeon Pro WX9100	Up to 6.9 GB/s
Gen 3: ATS9373	Radeon Pro WX7100	Up to 6.9 GB/s

3.4 ATS_GMA.log file

ATS-GMA logs relevant information about specific ATS-GMA calls in ATS_GMA.log. Most importantly, ATS_GMA.log stores information concerning errors occuring with functions from ATS-GMA.

3.5 API Reference

`enum ATS_GMA_SETUP_FLAG`

Flags to be passed to *ATS_GMA_Setup()*.

Note When using 12 bits packing, deinterleaving multiple channels without unpacking data to 16 bits is not possible.

Values:

`ATS_GMA_SETUP_FLAG_DEINTERLEAVE` = 0x1

De-interleave the data in the returned GMA buffer.

`ATS_GMA_SETUP_FLAG_UNPACK` = 0x2

Unpack the data in the returned GMA buffer.

`ATS_GMA_SETUP_FLAG_USER_DEFINED_CONTEXT` = 0x4

Use this flag to provide a custom OpenCL context instead of letting the library create one for you.

RETURN_CODE **`ATS_GMA_QueryDeviceCount`**(HANDLE *boardHandle*, U32 **deviceCount*)

Query the number of available OpenCL devices.

Return `ApiSuccess` if it succeeded

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

Parameters

- `boardHandle`: Handle to an AlazarTech board.
- `deviceCount`: Outputs the number of devices detected on the system.

RETURN_CODE **`ATS_GMA_GetComputeDevice`**(HANDLE *boardHandle*, U32 *deviceIndex*, cl_device_id **clDevice*)

OpenCL-specific function used to associate a OpenCL-enabled GPU device with a digitizer board. `ATS_GMA.log` will show information on the chosen OpenCL device name.

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

Return `ApiSuccess` if it succeeded

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

Parameters

- `boardHandle`: Handle to the ATS board.
- `deviceIndex`: 0-based index to the OpenCL device.
- `clDevice`: Output the address of a specific OpenCL device ID.

RETURN_CODE **`ATS_GMA_QueryDeviceName`**(HANDLE *boardHandle*, U32 *deviceIndex*, char **deviceName*)

OpenCL-specific function to get the name of a specific OpenCL-enabled device.

Return `ApiSuccess` if it succeeded

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

Parameters

- `boardHandle`: Handle to the ATS board.
- `deviceIndex`: 0-based index to the OpenCL device.
- `deviceName`: Output the address of a specific OpenCL device name.

RETURN_CODE **ATS_GMA_Setup**(HANDLE *boardHandle*, U32 *channelSelect*, long *transferOffset*, U32 *transferLength*, U32 *recordsPerBuffer*, U32 *recordsPerAcqusition*, U32 *autoDMAFlags*, U32 *ATSGMAFlags*, cl_context **clContext*, cl_command_queue **clQueue*)

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 and ATS_GMA.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.
- `autoDMAFlags`: ATSApi flags for AlazarBeforeAsyncRead.
- `ATSGMAFlags`: ATS-GMA specific flags. See `ATS_GMA_SETUP_FLAG`.
- `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.
- `clQueue`: Return a pointer to an array of four (4) OpenCL command queues.

cl_mem **`ATS_GMA_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_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 buffer 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_PostBuffer`**(HANDLE *boardHandle*, cl_mem *GpuBuffer*)

Informs the library that a particular buffer is ready to be used for data transfer.

This function is the equivalent of `AlazarPostAsyncBuffer()` for ATS_GMA. Buffers posted must have previously been allocated with *ATS_GMA_AllocBuffer()*.

Return `ApiSuccess` if it succeeded

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

Parameters

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

RETURN_CODE **`ATS_GMA_StartCapture`**(HANDLE *boardHandle*)

Starts the acquisition.

This function calls `AlazarStartCapture()` internally.

Parameters

- `boardHandle`: Hande to the board.

RETURN_CODE **`ATS_GMA_GetBuffer`**(HANDLE *boardHandle*, cl_mem *GpuBuffer*, cl_mem **GpuResultBuffer*, 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 retreived using *ATS_GMA_GetBuffer()*, it must be posted back to the board using *ATS_GMA_PostBuffer()*.

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

Return `ApiNotInitialized` if `AlazarStartCapture()` 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 GMA 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.
- `GpuBuffer`: GPU buffer allocated by `ATS_GMA_AllocBuffer()`.
- `GpuResultBuffer`: Pointer to the unpacked and deinterleaved GPU buffer.
- `clQueue`: Pointer to a OpenCL command queue created by the library on which unpacking and deinterleaving 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_FreeBuffer`**(HANDLE *boardHandle*, cl_mem *GpuBuffer*)

Frees a buffer allocated with *ATS_GMA_AllocBuffer()*

Return `ApiSuccess` if it succeeded

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

Parameters

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

RETURN_CODE **`ATS_GMA_AbortCapture`**(HANDLE *boardHandle*)

Stops the acquisition.

Aborts an acquisition, stops data processing, and releases resources allocated by *ATS_GMA_Setup()*. This function *must* be called at the end of every ATS-GMA acquisition.

Return `ApiSuccess` if it succeeded

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

Parameters

- `boardHandle`: Handle to the board.

Index