ATS-GPU-NUFFT

Version 24.1.0
January 16, 2024

CONTENTS

Note: This is the documentation for AlazarTech's ATS-GPU version 24.1.0. Please visit our documentation homepage to find documentation for other versions or products.

CHAPTER

ONE

LICENSE AGREEMENT

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.

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.

CHAPTER

TWO

INTRODUCTION

ATS-GPU-NUFFT provides a framework to allow real-time, non-uniform Fourier transform processing from AlazarTech PCIe digitizers on a CUDA-compatible GPU.

ATS-GPU-NUFFT internally calls ATS-CUDA-NUFFT, which is a low-level library that performs all the necessary operations to perform the non-uniform Fourier Transform. ATS-CUDA-NUFFT is described later in this guide in the section *ATS-CUDA-NUFFT*.

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.

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 also requires a working installation of the same version of ATS-GPU-BASE and ATS-GPU-OCT. It was tested with a GeForce RTX 2080 Ti and a 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
: 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 releated to ATS-GPU-NUFFT are installed by default in %WINDIR%System32. For applications to link approripately to them, %WINDIR%System32 must be added to the Windows PATH Environment Variable.

CHAPTER

FOUR

ATS-GPU-NUFFT

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

4.1 Usage

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

The central function of the ATS-GPU-NUFFT interface is `ATS_GPU_NUFFT_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 should be a power of two for efficiency, and it also must be equal to or larger than the record length.
- `NUFFTFlags`: Used to define the source of the linearization function. Linearization can either be user defined by selecting `ATS_GPU_NUFFT_PRESET_LINEARIZATION` or be determined from a K-clock signal when `ATS_GPU_NUFFT_KCLOCK_LINEARIZATION` is used.

```
rc = ATS_GPU_NUFFT_Setup(
         boardHandle, channelMask, -(int) preTriggerSamples,
         samplesPerRecordPerChannel, recordsPerBuffer,
         buffersPerAcquisition * recordsPerBuffer, autoDMAFlags,
         OCTOptions, FFTLength, NUFFTFlags, NULL, &fftBytesPerBuffer);
// Error handling
```

If `NUFFTFlags` is setup with `ATS_GPU_NUFFT_KCLOCK_LINEARIZATION`, the channelMask must contain `CHANNEL_A` and at least one other active channel. With this flag, ATS-GPU-NUFFT will determine the linearization function for each record from the k-clock signal acquired on `CHANNEL_A` and use it to perform non-uniform FFT on every other active channel.

If NUFFTFlags was setup with ATS_GPU_NUFFT_PRESET_LINEARIZATION, the user is required to specify a precalibrated linearization function. This linearization function will be used to perform non-uniform FFT on every record of every active channel.

```
int precalibratedFunctionLength = 1000;
std::vector<float> precalibratedFunction(precalibratedFunctionLength);
for (int i = 0; i < precalibratedFunction.size(); i++) {
    precalibratedFunction[i] = i;
}
rc = ATS_GPU_NUFFT_SetLinearizationFunction(
            boardhandle, precalibratedFunction,
            &precalibratedFunction[0]);
// Error handling
```

Here, we generated a linear linearization function. Setting a linear precalibrated function represents a signal that is sampled linearly in k-space, thus equivalent to applying regular FFT.

The precalibrated linearization function can have any length as ATS-GPU-NUFFT will internally take care of re-sampling the function to a length equal to samplesPerRecordPerChannel. ATS-GPU-NUFFT will also normalize the function. The precalibrated linearization can therefore have any start and end values. The values of the precalibrated linearization function must always be increasing such as x[i]<x[i+1].

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_NUFFT_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_NUFFT_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_NUFFT_AllocBuffer(). This function allocates buffers on the GPU, and sets up all the intermediary states necessary for ATS-GPU-NUFFT to successfully transfer data. It also allocates data on the CPU to receive the processed data.

```
for (int i = 0; i < numberOfBuffers; i++)
{
    buffers[i] = (float*) ATS_GPU_NUFFT_AllocBuffer(
```

(continues on next page)

(continued from previous page)

```
        boardHandle, bytesPerResultBuffer, NULL);

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

We can then start the acquisition with `ATS_GPU_NUFFT_StartCapture()`. Once the acquisition is started, `ATS_GPU_NUFFT_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_NUFFT_GetBuffer(
        boardHandle, buffers[bufferIndex],timeout_ms);
    // Error handling

    // TODO: Process sample data in this buffer.

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

When acquisition is complete, `ATS_GPU_NUFFT_AbortCapture()` must be called. Buffers allocated with `ATS_GPU_NUFFT_AllocBuffer()` should then be freed with `ATS_GPU_NUFFT_FreeBuffer()`.

```
ATS_GPU_NUFFT_AbortCapture(boardHandle);

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

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

4.1.1 LabVIEW Programming

LabVIEW applications must use the managed interface which allows the API to allocate and manage a list of buffers available to be filled by the board. These applications should call `ATS_GPU_NUFFT_Setup()` with the AMDA_ALLOC_BUFFERS option selected in the "autoDMAFlags" parameter. This option will cause the API to allocate and manage a list of buffers available to be filled by the board. It is therefore not necessary for the application to call `ATS_GPU_NUFFT_AllocBuffer()` or `ATS_GPU_NUFFT_FreeBuffer()`. The application must call `ATS_GPU_NUFFT_ManageGetBuffer()` to wait for a buffer to be filled. When the board receives sufficient trigger events to fill a buffer, the API will copy the data from the internal buffer to the user-supplied buffer. `ATS_GPU_NUFFT_ManageGetBuffer()` internally calls `ATS_GPU_NUFFT_GetBuffer()` and `ATS_GPU_NUFFT_PostBuffer()` so applcation should not use these API calls when using the managed interface.

LabVIEW users might find it convenient to edit the VI search paths to locate the appropriate subVIs for the different ATS-GPU packages and ATS-SDK. The VI Search Path can be set in the "Tools" menu under "Options", in the "Path" category. Then select the "VI Search Path" from the drop down list. By unselecting "Use default" custom VI search paths can be added.

4.2 API Reference

Note: Errors from ATS-GPU-NUFFT 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_GPU_NUFFT_OPTIONS**

Linearization source specifier. If ATS_GPU_NUFFT_KLCOCK_LINEARIZATION is used, k-clock signal must be connected to CHANNEL_A. If ATS_GPU_NUFFT_PRESET_LINEARIZATION is used, a linearization calibration function must be set using ATS_GPU_NUFFT_SetLinearizationWindowFunction(). This is used in *ATS_GPU_NUFFT_Setup()*.

Values:

enumerator **ATS_GPU_NUFFT_PRESET_LINEARIZATION**

enumerator **ATS_GPU_NUFFT_KCLOCK_LINEARIZATION**

RETURN_CODE **ATS_GPU_NUFFT_AbortCapture**(HANDLE boardHandle)

Stops the acquisition.

Aborts an acquisition, stops data processing, and releases allocated resources.

Parameters

boardHandle – Handle to the board

Returns

ApiSuccess

void ***ATS_GPU_NUFFT_AllocBuffer**(HANDLE boardHandle, U32 bytesPerBuffer, void *reserved)

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

This function must be called after *ATS_GPU_NUFFT_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_NUFFT_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_NUFFT_FreeBuffer**(HANDLE boardHandle, void *buffer)

Free buffers allocated with *ATS_GPU_NUFFT_AllocBuffer()*;.

Parameters

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

RETURN_CODE **ATS_GPU_NUFFT_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.

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.

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

Returns
`ApiNotInitialized` if *ATS_GPU_NUFFT_StartCapture()* was not called before calling this function, or it was called and failed.

Returns
`ApiInvalidHandle` if 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 or internal error occured.

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

Get ATS-GPU-NUFFT version number.

Parameters

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

RETURN_CODE **ATS_GPU_NUFFT_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-NUFFT. Buffers posted must have previously been allocated with *ATS_GPU_NUFFT_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.

RETURN_CODE **ATS_GPU_NUFFT_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_NUFFT_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_NUFFT_SetLinearizationFunction**(HANDLE boardHandle, U32 precalibratedFunctionLength, float *precalibratedFunction)

Set linearization function used in NUFFT calculation. This call should be made if *ATS_GPU_NUFFT_Setup* was called using ATS_GPU_NUFFT_PRESET_LINEARIZATION as a parameter for NUFFTFlags.

Parameters

- **boardHandle** – Handle to the board
- **precalibratedFunctionLength** – Length of the linearization function, can be different from samplesPerRecordPerChannel.
- **precalibratedFunction** – Pointer to array of size precalibratedFunction-Length that contains the linearization function. Passing `null` is equivalent to passing a linearly spaced linearization grid.

RETURN_CODE **ATS_GPU_NUFFT_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_NUFFT_Setup**(HANDLE boardHandle, U32 channelSelect, long transferOffset, U32 samplesPerFFT, U32 FFTsPerBuffer, U32 FFTsPerAcquisition, U32 autoDMAFlags, U32 OCTFlags, U32 FFTLength, U32 NUFFTFlags, 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 NUFFT 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_NUFFT_AbortCapture()*
- **autoDMAFlags** – ATSApi flags for `AlazarBeforeAsyncRead`
- **OCTFlags** – Defines the types of data outputs to be obtained from the NUFFT acquisition. This parameter can receive one or more elements of ATS_GPU_OCT_OPTIONS and ATS_GPU_PSOCT_OPTIONS, or'ed with the binary OR operator.
- **FFTLength** – Length of FFT, should be a power of 2.
- **NUFFTFlags** – Determines source of linearization. This parameter can receive one element of *ATS_GPU_NUFFT_OPTIONS*.
- **reserved** – Pass NULL
- **bytesPerResultBuffer** – Returns the size of a result buffer

RETURN_CODE **ATS_GPU_NUFFT_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_NUFFT_GetBuffer()* to prevent data overflows

Parameters

boardHandle – Handle to the board

CHAPTER

FIVE

ATS-CUDA-NUFFT

ATS-CUDA-NUFFT provides a framework to allow non-uniform data processing on a CUDA-enabled GPU. ATS-CUDA-NUFFT internally calls ATS-CUDA and ATS-CUDA-OCT and should be used with ATS-CUDA for buffer and stream manipulation. ATS-CUDA-NUFFT requires an AlazarTech board on the system in order to be used.

5.1 API Reference

Note: Errors from ATS-CUDA-NUFFT 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/.

atsNuFFTPlan ***ATS_CUDA_NUFFT_CreateNuFFTPlan**(U32 FFTLength, U32 samplesPerRecordPerChannel, U32 FFTsPerBuffer, cudaStream_t stream)

Creates a non-uniform FFT plan and associates it with a CUDA stream. A non-uniform FFT plan contains all the data and GPU resources necessary to perform a non-uniform fast Fourier transform.

This function is used to allocate resources on a GPU and configure a GPU kernel to perform non-uniform FFT processing. It also associates the newly created non-uniform FFT plan with a CUDA stream. All kernels executed with this plan are to be run on this stream.

Parameters

- **FFTLength** – Length of FFT, should be a power of 2 for performance.
- **samplesPerRecordPerChannel** – Number of samples in a record.
- **FFTsPerBuffer** – Number of FFTs to perform per buffer.
- **stream** – The CUDA stream to run the FFT plan with.

Returns

This function returns a pointer to the created non-uniform FFT plan.

RETURN_CODE **ATS_CUDA_NUFFT_NuFFT**(atsNuFFTPlan *NuFFTPlan, void *GPUBaseBuffer, void *GPUNuFFTOut, void *GPULinearizationBuffer, ATS_CUDA_Input_DataType inputDataType, void *GPUWindow)

Launches a kernel on the GPU to perform the non-uniform Fast Fourier Transform.

Parameters

- **NuFFTPlan** – Pointer to a non-uniform FFT plan created with ATS_CUDA_NUFFT_CreateNuFFTPlan().
- **GPUBaseBuffer** – Pointer to a GPU buffer on which to apply NuFFT kernel. This buffer should have 8 bits, 16 bits or float32 data packing and have de-interleaved channels.
- **GPUNuFFTOut** – Pointer to a GPU NuFFT result buffer. Output buffer has complex float32 precision.
- **GPULinearizationBuffer** – Pointer to a GPU linearization buffer with float32 precision. Must have the same samplesPerRecordPerChannel and same recordsPerBuffer as GPUBaseBuffer. Can be generated from ATS_CUDA_NUFFT_GetLinearizationFromPrecalibratedFunction() or ATS_CUDA_NUFFT_GetLinearizationFromKclock()
- **inputDataType.** – Data type of GPUBaseBuffer. This parameter must receive one element of ATS_CUDA_Input_DataType.
- **GPUWindow** – Pointer to a GPU window buffer allocated with ATS_CUDA_OCT_GenerateGPUWindowFunction()

RETURN_CODE **ATS_CUDA_NUFFT_DestroyNuFFTPlan**(atsNuFFTPlan *NuFFTPlan)

Destroy a non-uniform plan.

Frees all GPU resources associated with a non-uniform FFT plan.

Parameters

NuFFTPlan – Pointer to the non-uniform FFT plan to be destroyed.

atsLinearizationPlan ***ATS_CUDA_NUFFT_CreateLinearizationPlan**(U32 samplesPerRecordPerChannel, U32 recordsPerBuffer, ATS_CUDA_Input_DataType inputDataType, cudaStream_t stream)

Creates a linearization plan and associates it with a CUDA stream. A linearization plan contains all the data and GPU resources necessary to compute the linearization function from a k-clock signal.

This function is used to allocate the required resources and configure a GPU kernel to perform the necessary processing to obtain a linearization buffer from a k-clock signal. It also associates the newly created linearization plan with a CUDA stream. All kernels executed with this plan are to be run on this stream.

Parameters

- **samplesPerRecordPerChannel** – Number of samples in each k-clock record.
- **recordsPerBuffer** – Number of records in the k-clock signal.
- **inputDataType.** – Data type of the k-clock signal. This parameter must receive one element of ATS_CUDA_Input_DataType.
- **stream** – The CUDA stream to run the linearization plan with.

Returns

This function returns a pointer to the created linearization plan.

RETURN_CODE **ATS_CUDA_NUFFT_GetLinearizationFromKclock**(atsLinearizationPlan *linPlan, void *pKclock, void *GPULinearizationBuffer)

Launches a kernel on the GPU to get the linearization function from a k-clock signal.

Parameters

- **linPlan** – Pointer to a linearization plan created with ATS_CUDA_NUFFT_CreateLinearizationPlan().
- **pKclock** – Pointer to a GPU buffer containing k-clock data. K-clock buffer data type must be as specified in ATS_CUDA_NUFFT_CreateLinearizationPlan().
- **GPULinearizationBuffer** – Pointer to a GPU buffer containing the linearization buffer that can be passed to ATS_CUDA_NUFFT_NuFFT(). Linearization buffer has float32 precision and has same size as pkclock buffer.

RETURN_CODE **ATS_CUDA_NUFFT_DestroyLinearizationPlan**(atsLinearizationPlan *linPlan)

Destroy a linearization plan.

Frees all GPU resources associated with a linearization plan.

Parameters

linPlan – Pointer to the linearization plan to be destroyed.

RETURN_CODE **ATS_CUDA_NUFFT_GetLinearizationFromPrecalibratedFunction**(void *GPUPrecalibratedFunction, void *GPULinearizationBuffer, U32 samplesPerRecordIn, U32 samplesPerRecordOut, U32 recordsPerBuffer, cudaStream_t stream)

Generates a linearization buffer from a precalibrated function.

This function prepares a GPU linearization buffer that can be passed to ATS_CUDA_NUFFT_NuFFT().

Parameters

- **GPUPrecalibratedFunction** – Pointer to a GPU buffer of size samplesPerRecordIn of type float32 that contains the linearization function.
- **GPULinearizationBuffer** – Pointer to a GPU buffer of size samplesPerRecordOut * recordsPerBuffer and float32 precision where the linearization buffer is to be written.
- **samplesPerRecordIn** – Length of the GPU PrecalibratedFunction.
- **samplesPerRecordOut** – Length of each record of the GPU LinearizationBuffer.
- **recordsPerBuffer** – Number of times the GPUPrecalibratedFunction is repeated in GPULinearizationBuffer.
- **stream** – Stream identifier on which processing is to take place.

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

Get ATS-CUDA-NUFFT version number.

Parameters

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

INDEX

A