ACF Reference Guide

UG-10267-04-08

Copyright

Disclaimer

NXP assumes no responsibility for the accuracy or completeness of the information presented which is subject to change without notice. In no event will NXP be liable for any direct, indirect, special, incidental or consequential damages, including lost profits, lost business or lost data, resulting from the use of or reliance upon the information, whether or not NXP has been advised of the possibility of such damages.

Mention of non-NXP products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.

Uncontrolled Copy

The master of this document is stored on NXP's document management system Docushare / Confluence. Viewing of the master electronically ensures access to the current issue. Any hardcopies are considered uncontrolled copies.

Version	Details of Change	Author	Date
01	Initial Revision	C. Moulder	October 27, 2015
02	AcfProfilingInfo related items added, misc. updates	C. Moulder	May 16, 2016
03	Formatting improvements, misc. updates	C. Moulder	Sept 27, 2016
04	Update for UMAT interfaces	C. Moulder	Mar 13, 2017
05	Added interface description for Start w/ callback	C. Moulder	May 3, 2017
06	Misc. edits related to limits	C. Moulder	Feb 16, 2018
07	Edits for Start, Wait, CfgWaitTimeout	C. Moulder	Aug 10, 2018
08	Umat replace by SUmat	K.Pham	Dec 06, 2018

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Module Documentation

4.1 ACF_APU_CFG

Enumerations

4.1.1 Detailed Description

ACF_APU_CFG - the various APU configurations a process may be run on

4.1.2 Enumeration Type Documentation

4.1.2.1 enum _ACF_APU_CFG

Enumerator

ACF_APU_CFG__DEFAULT APU0 with all CUs and all SMEM.

ACF_APU_CFG__APU_0_CU_0_63_SMEM_0_3 APU0 with CUs 0-63 and 128K SMEM.

ACF_APU_CFG__APU_0_CU_0_31_SMEM_0_1 APU0 with CUs 0-31 and 64K SMEM.

ACF_APU_CFG__APU_1_CU_32_63_SMEM_2_3 APU1 with CUs 32-63 and 64K SMEM.

4.2 DATATYPE

Enumerations

4.2.1 Detailed Description

The DATATYPE typedef aliases _DATATYPE, which defines basic 8, 16, and 32 bit signed and unsigned data types.

4.2.2 Enumeration Type Documentation

4.2.2.1 enum icp::_DATATYPE

Enumerator

DATATYPE_08U	8-bit unsigned
DATATYPE_08S	8-bit signed
DATATYPE_16U	16-bit unsigned
DATATYPE_16S	16-bit signed
DATATYPE_32U	32-bit unsigned
DATATYPE_32S	32-bit signed

Chapter 5

Class Documentation

5.1 _AcfProfilingInfo Struct Reference

```
#include <acf_process_apu.h>
```

Public Attributes

- int32_t host_start
- int32_t host_wait
- int32_t apu_total
- int32_t apu_init
- int32_t apu_processing
- int32_t apu_idle
- int32_t apu_misc

5.1.1 Detailed Description

AcfProfilingInfo is a struct containing acf profiling information

5.1.2 Member Data Documentation

5.1.2.1 int32_t _AcfProfilingInfo::apu_idle

apu time (us) spent waiting for data transfers to complete (if this is large, the process is likely bandwidth limited)

5.1.2.2 int32_t _AcfProfilingInfo::apu_init

apu time (us) spent on initialization

5.1.2.3 int32_t _AcfProfilingInfo::apu_misc

apu time (us) spent on misc. overhead (control flow, descriptor management, etc.)

5.1.2.4 int32_t _AcfProfilingInfo::apu_processing

apu time (us) spent on kernel execution + padding + circular buffer management

5.1.2.5 int32_t _AcfProfilingInfo::apu_total

total apu time (us) (includes init, processing, idle, and all other overhead)

5.1.2.6 int32_t _AcfProfilingInfo::host_start

host time (us) spent from the beginning of start() to the triggering of process execution on the APEX (this is 100% host overhead and includes apu loading, resource acquisition, etc.)

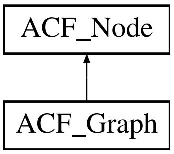
5.1.2.7 int32_t _AcfProfilingInfo::host_wait

host time (us) spent in the wait() call (this includes time spent waiting for the process execution to complete, plus a small amount of overhead)

5.2 ACF_Graph Class Reference

```
#include <acf_graph.hpp>
```

Inheritance diagram for ACF_Graph:



Public Member Functions

- virtual void Create ()=0
- void SetIdentifier (std::string lGraphIdentifier)
- void AddInputPort (std::string lPortIdentifier)
- void AddOutputPort (std::string lPortIdentifier)
- ACF_Port ∗ GraphPort (std::string lPortIdentifier)
- ACF_Port ∗ KernelPort (std::string lKernelIdentifier, std::string lPortIdentifier)
- void AddKernel (std::string lKernelIdentifier, std::string lKernelDatabaseIdentifier)
- void Connect (ACF_Port ∗lpSrcPort, ACF_Port ∗lpDstPort)

Friends

- class **ACF_Process_Desc**
- class **ACF_Process_Desc_APU**

5.2.1 Detailed Description

ACF_Graph is a base class designed to encapsulate an ACF graph. In order to create a graph, a user must derive from this class and implement the pure virtual Create() method.

5.2.2 Member Function Documentation

5.2.2.1 void ACF_Graph::AddInputPort (std::string *lPortIdentifier*)

Add an input port identified by "lPortIdentifier" to the graph. The total number of ports (input + output) must not exceed 50.

eg.

```
AddInputPort("GRAPH_INPUT_0");
AddInputPort("GRAPH_INPUT_1");
```

Parameters

in	*lPortIdentifier*	Input port identifier.

5.2.2.2 void ACF_Graph::AddKernel (std::string *lKernelIdentifier,* std::string *lKernelDatabaseIdentifier*)

Creates an instance of the kernel "lKernelDatabaseIdentifier" in the graph (this is the unique kernel identifier specified in the kernel metadata). and assigns the kernel instance the unique handle specified by "lKernelIdentifier". Note that if there are N instances of a kernel in a graph, that kernel must be 'instantiated' N times, each time with a unique "lKernelIdentifier". The number of kernels per graph must not exceed 100.

eg.

```
AddKernel("myAddKernel1",   "ADD"); //first instance of the ’ADD’ kernel
AddKernel("myAddKernel2",   "ADD"); //second instance of the ’ADD’ kernel
AddKernel("myFilterKernel", "FILTER");
```

Parameters

in	*lKernelIdentifier*	Identifier that acts as a local kernel handle (i.e. the identifier by which a kernel instance is referred to during graph construction)
in	*lKernelDatabaseIdentifier*	Identifier used to select kernel from the database. This is the identifier specified in the kernel metadata.

5.2.2.3 void ACF_Graph::AddOutputPort (std::string *lPortIdentifier*)

Add an output port identified by "lPortIdentifier" to the graph. The total number of ports (input + output) must not exceed 50.

eg.

```
AddOutputPort("GRAPH_OUTPUT_0");
AddOutputPort("GRAPH_OUTPUT_1");
```

Parameters

in	*lPortIdentifier*	Output port identifier.

5.2.2.4 void ACF_Graph::Connect (ACF_Port ∗ *lpSrcPort,* ACF_Port ∗ *lpDstPort*)

Connect the source port "lpSrcPort" to the destination port "lpDstPort". This is a forward-directed connection from source to destination. A source port may be connected to a maximum of 100 destination ports.

If a source port is connected to multiple destination ports, all destination ports must share the same fundamental attributes (i.e. e0, VEC/SCL, STATIC/NON-STATIC). For example, a graph input cannot be connected to a ACF_AT↩ TR_SCL_IN port and a ACF_ATTR_VEC_IN port; both must be VEC, or both must be SCL.

Parameters

in	*lpSrcPort*	Pointer to source ACF_Port.
in	*lpDstPort*	Pointer to destination ACF_Port.

5.2.2.5 virtual void ACF_Graph::Create ( ) [pure virtual]

This is a pure virtual method that must be implemented by the derived class. Use the graph construction methods of ACF_Graph (e.g. AddKernel, AddInputPort, AddOutputPort, Connect, etc.), to describe the graph.

5.2.2.6 ACF_Port∗ ACF_Graph::GraphPort (std::string *lPortIdentifier*)

Return a pointer to the graph port identified by "lPortIdentifier".

Parameters

in	*lPortIdentifier*	Graph port identifier.

Returns

Pointer to the graph port identified by "lPortIdentifier".

5.2.2.7 ACF_Port∗ ACF_Graph::KernelPort (std::string *lKernelIdentifier,* std::string *lPortIdentifier*)

Returns a pointer to the port "lPortIdentifier" belonging to the kernel "lKernelIdentifier".

Parameters

in	*lKernelIdentifier*	Kernel instance identifier.
in	*lPortIdentifier*	Kernel port identifier.

Returns

Pointer to the port "lPortIdentifier" belonging to the kernel "lKernelIdentifier".

5.2.2.8 void ACF_Graph::SetIdentifier (std::string *lGraphIdentifier*)

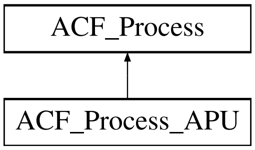
Set an identifier to uniquely identify the graph.

Parameters

in	*lGraphIdentifier*	Graph identifier.

5.3 ACF_Process Class Reference

Inheritance diagram for ACF_Process:



Public Member Functions

- int32_t ConnectIO (std::string lPortIdentifier, icp::DataDescriptor &lDataDesc)
- int32_t ConnectIO (std::string lPortIdentifier, const vsdk::SUMat &lUmat)
- int32_t ConnectIO_ROI (std::string lPortIdentifier, icp::DataDescriptor &lDataDesc, int32_t lROI_XOffset, int32_t lROI_YOffset, int32_t lROI_Width, int32_t lROI_Height)
- int32_t ConnectIO_ROI (std::string lPortIdentifier, const vsdk::SUMat &lUmat, int32_t lROI_XOffset, int32_t lROI_YOffset, int32_t lROI_Width, int32_t lROI_Height)
- int32_t ConnectIndirectInput (std::string lPortIdentifier, icp::DataDescriptor &lSrcData, icp::DataDescriptor &lChunkOffsetArray)
- int32_t ConnectIndirectInput (std::string lPortIdentifier, const vsdk::SUMat &lSrcData, const vsdk::SUMat &lChunkOffsetArray)
- int32_t SetRoiInfo (int32_t lRoiInfoL, int32_t lRoiInfoR, int32_t lRoiInfoT, int32_t lRoiInfoB)
- void CfgWaitTimeout (int32_t lTimeoutInUs)

5.3.1 Member Function Documentation

5.3.1.1 void ACF_Process::CfgWaitTimeout (int32_t *lTimeoutInUs*)

Specify Wait() timeout duration in microseconds. The default timeout duration is 1000000us (i.e 1 second).

Parameters

in	*lTimeoutInUs*	Desired timeout in microseconds (us).

5.3.1.2 int32_t ACF_Process::ConnectIndirectInput (std::string *lPortIdentifier,* icp::DataDescriptor & *lSrcData,* icp::DataDescriptor & *lChunkOffsetArray*)

Connect a 2D array of chunks specified by "lChunkOffsetArray" and "lSrcData" to graph input port "lPortIdentifier". "l↩ ChunkOffsetArray" is a 2D array of 32-bit offsets (in units of bytes) that when added to the start of "lSrcData" region, result in valid pointers to the top left corners of the desired chunks of data in memory. Note that chunk size is set via ACF_Process_Desc_APU::SetInputChunkSize(...) and all chunks are assumed to be the same size.

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lSrcData*	Reference to region of source data. The start of this memory region is effectively the 'base address' that will be used with the chunk offsets specified in "lChunkOffsetArray".
in	*lChunkOffsetArray*	Reference to 2D array of chunk offsets. Offsets should be relative to the start of the contiguous data region specified by "lSrcData", should address the upper left corners of the desired 2D chunks, and should be in units of bytes. NOTE: the number of offsets in the horizontal dimension must be a multiple of 4 (e.g. an offset array with dimensions 8x4 is allowed, but an array with dimensions 10x4 will result in an error).

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.3 int32_t ACF_Process::ConnectIndirectInput (std::string *lPortIdentifier,* const vsdk::SUMat & *lSrcData,* const vsdk::SUMat & *lChunkOffsetArray*)

Connect a 2D array of chunks specified by "lChunkOffsetArray" and "lSrcData" to graph input port "lPortIdentifier". "l↩ ChunkOffsetArray" is a 2D array of 32-bit offsets (in units of bytes) that when added to the start of "lSrcData" region, result in valid pointers to the top left corners of the desired chunks of data in memory. Note that chunk size is set via ACF_Process_Desc_APU::SetInputChunkSize(...) and all chunks are assumed to be the same size.

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lSrcData*	Reference to region of source data. The start of this memory region is effectively the 'base address' that will be used with the chunk offsets specified in "lChunkOffsetArray".
in	*lChunkOffsetArray*	Reference to 2D array of chunk offsets. Offsets should be relative to the start of the contiguous data region specified by "lSrcData", should address the upper left corners of the desired 2D chunks, and should be in units of bytes. NOTE: the number of offsets in the horizontal dimension must be a multiple of 4 (e.g. an offset array with dimensions 8x4 is allowed, but an array with dimensions 10x4 will result in an error).

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.4 int32_t ACF_Process::ConnectIO (std::string *lPortIdentifier,* icp::DataDescriptor & *lDataDesc*)

Connect the data region described by "lDataDesc" to graph port "lPortIdentifier".

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lDataDesc*	Description of contiguous data region.

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.5 int32_t ACF_Process::ConnectIO (std::string *lPortIdentifier,* const vsdk::SUMat & *lUmat*)

Connect the data region described by "lUmat" to graph port "lPortIdentifier".

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lUmat*	Description of contiguous data region.

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.6 int32_t ACF_Process::ConnectIO_ROI (std::string *lPortIdentifier,* icp::DataDescriptor & *lDataDesc,* int32_t *lROI_XOffset,* int32_t *lROI_YOffset,* int32_t *lROI_Width,* int32_t *lROI_Height*)

Connect the region of interest (ROI) described by "lDataDesc", "lROI_XOffset", "lROI_YOffset", "lROI_Width", and "l↩ROI_Height" to graph port "lPortIdentifier".

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lDataDesc*	Description of contiguous data region that 'contains' the ROI.
in	*lROI_XOffset*	The X offset of the top left corner of the ROI (relative to the top left corner of the region described by "lDataDesc").
in	*lROI_YOffset*	The Y offset of the top left corner of the ROI (relative to the top left corner of the region described by "lDataDesc").
in	*lROI_Width*	The width of the ROI.
in	*lROI_Height*	The height of the ROI.

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.7 int32_t ACF_Process::ConnectIO_ROI (std::string *lPortIdentifier,* const vsdk::SUMat & *lUmat,* int32_t *lROI_XOffset,* int32_t *lROI_YOffset,* int32_t *lROI_Width,* int32_t *lROI_Height*)

Connect the region of interest (ROI) described by "lUmat", "lROI_XOffset", "lROI_YOffset", "lROI_Width", and "lROI_↩ Height" to graph port "lPortIdentifier".

Parameters

`in`	*lPortIdentifier*	Graph port identifier.
`in`	*lUmat*	Description of contiguous data region that 'contains' the ROI.
`in`	*lROI_XOffset*	The X offset of the top left corner of the ROI (relative to the top left corner of the region described by "lUmat").
`in`	*lROI_YOffset*	The Y offset of the top left corner of the ROI (relative to the top left corner of the region described by "lUmat").
`in`	*lROI_Width*	The width of the ROI.
`in`	*lROI_Height*	The height of the ROI.

Returns

0 if successful, non-zero if an an error occurred.

5.3.1.8 int32_t ACF_Process::SetRoiInfo (int32_t *lRoiInfoL,* int32_t *lRoiInfoR,* int32_t *lRoiInfoT,* int32_t *lRoiInfoB*)

[DEPRECATED] Globally indicate how much data beyond 2D input borders should be taken into account for the region of interest (ROI) case. If these are set to non-zero values, the indicated data must be available on the borders of ALL applicable inputs.

Parameters

`in`	*lRoiInfoL*	Number of source elements available beyond the left border of the 2D input region. It must be a multiple of chunk width.
`in`	*lRoiInfoR*	Number of source elements available beyond the right border of the 2D input region. It must be a multiple of chunk width.
`in`	*lRoiInfoT*	Number of source elements available beyond the top border of the 2D input region. It must be a multiple of chunk height.
`in`	*lRoiInfoB*	Number of source elements available beyond the bottom border of the 2D input region. It must be a multiple of chunk height.

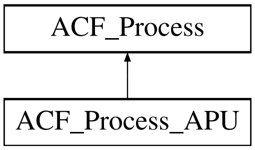
Returns

0 if successful, non-zero if an an error occurred.

5.4 ACF_Process_APU Class Reference

```
#include <acf_process_apu.h>
```

Inheritance diagram for ACF_Process_APU:



Public Member Functions

- virtual int32_t Initialize ()=0
- virtual int32_t Start ()
- virtual int32_t Wait ()
- int32_t SelectScenario (std::string lPortIdentifier, int32_t lChunkWidth, int32_t lChunkHeight)
- int32_t QueryPortChunkSize (std::string lPortIdentifier, int32_t &lChunkWidth, int32_t &lChunkHeight)
- int32_t SelectApuConfiguration (ACF_APU_CFG lApuConfig, int32_t lApexId)
- AcfProfilingInfo RetAcfProfilingInfo ()
- int32_t Start (void(*lpCallback)(void *lpParam, int32_t *lpRetVal), void *lpCallbackParam, int32_t *lpCallbackRetVal)
- int32_t ConnectIO (std::string lPortIdentifier, icp::DataDescriptor &lDataDesc)
- int32_t ConnectIO (std::string lPortIdentifier, const vsdk::SUMat &lUmat)
- int32_t ConnectIO_ROI (std::string lPortIdentifier, icp::DataDescriptor &lDataDesc, int32_t lROI_XOffset, int32_t lROI_YOffset, int32_t lROI_Width, int32_t lROI_Height)
- int32_t ConnectIO_ROI (std::string lPortIdentifier, const vsdk::SUMat &lUmat, int32_t lROI_XOffset, int32_t lROI_YOffset, int32_t lROI_Width, int32_t lROI_Height)
- int32_t ConnectIndirectInput (std::string lPortIdentifier, icp::DataDescriptor &lSrcData, icp::DataDescriptor &lChunkOffsetArray)
- int32_t ConnectIndirectInput (std::string lPortIdentifier, const vsdk::SUMat &lSrcData, const vsdk::SUMat &lChunkOffsetArray)
- int32_t SetRoiInfo (int32_t lRoiInfoL, int32_t lRoiInfoR, int32_t lRoiInfoT, int32_t lRoiInfoB)
- void CfgWaitTimeout (int32_t lTimeoutInUs)

5.4.1 Detailed Description

ACF_Process_APU is the base class from which an APU process is derived. It provides access to all the methods required for run-time configuration and execution of an APU process.

5.4.2 Member Function Documentation

5.4.2.1 void ACF_Process::CfgWaitTimeout (int32_t *lTimeoutInUs*) `[inherited]`

Specify Wait() timeout duration in microseconds. The default timeout duration is 1000000us (i.e 1 second).

Parameters

in	*lTimeoutInUs*	Desired timeout in microseconds (us).

5.4.2.2 int32_t ACF_Process::ConnectIndirectInput (std::string *lPortIdentifier,* icp::DataDescriptor & *lSrcData,* icp::DataDescriptor & *lChunkOffsetArray*) [inherited]

Connect a 2D array of chunks specified by "lChunkOffsetArray" and "lSrcData" to graph input port "lPortIdentifier". "l↩ChunkOffsetArray" is a 2D array of 32-bit offsets (in units of bytes) that when added to the start of "lSrcData" region, result in valid pointers to the top left corners of the desired chunks of data in memory. Note that chunk size is set via ACF_Process_Desc_APU::SetInputChunkSize(...) and all chunks are assumed to be the same size.

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lSrcData*	Reference to region of source data. The start of this memory region is effectively the 'base address' that will be used with the chunk offsets specified in "lChunkOffsetArray".
in	*lChunkOffsetArray*	Reference to 2D array of chunk offsets. Offsets should be relative to the start of the contiguous data region specified by "lSrcData", should address the upper left corners of the desired 2D chunks, and should be in units of bytes. NOTE: the number of offsets in the horizontal dimension must be a multiple of 4 (e.g. an offset array with dimensions 8x4 is allowed, but an array with dimensions 10x4 will result in an error).

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.3 int32_t ACF_Process::ConnectIndirectInput (std::string *lPortIdentifier,* const vsdk::SUMat & *lSrcData,* const vsdk::SUMat & *lChunkOffsetArray*) [inherited]

Connect a 2D array of chunks specified by "lChunkOffsetArray" and "lSrcData" to graph input port "lPortIdentifier". "l↩ChunkOffsetArray" is a 2D array of 32-bit offsets (in units of bytes) that when added to the start of "lSrcData" region, result in valid pointers to the top left corners of the desired chunks of data in memory. Note that chunk size is set via ACF_Process_Desc_APU::SetInputChunkSize(...) and all chunks are assumed to be the same size.

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lSrcData*	Reference to region of source data. The start of this memory region is effectively the 'base address' that will be used with the chunk offsets specified in "lChunkOffsetArray".
in	*lChunkOffsetArray*	Reference to 2D array of chunk offsets. Offsets should be relative to the start of the contiguous data region specified by "lSrcData", should address the upper left corners of the desired 2D chunks, and should be in units of bytes. NOTE: the number of offsets in the horizontal dimension must be a multiple of 4 (e.g. an offset array with dimensions 8x4 is allowed, but an array with dimensions 10x4 will result in an error).

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.4 int32_t ACF_Process::ConnectIO (std::string *lPortIdentifier,* icp::DataDescriptor & *lDataDesc*) `[inherited]`

Connect the data region described by "lDataDesc" to graph port "lPortIdentifier".

Parameters

`in`	*lPortIdentifier*	Graph port identifier.
`in`	*lDataDesc*	Description of contiguous data region.

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.5 int32_t ACF_Process::ConnectIO (std::string *lPortIdentifier,* const vsdk::SUMat & *lUmat*) `[inherited]`

Connect the data region described by "lUmat" to graph port "lPortIdentifier".

Parameters

`in`	*lPortIdentifier*	Graph port identifier.
`in`	*lUmat*	Description of contiguous data region.

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.6 int32_t ACF_Process::ConnectIO_ROI (std::string *lPortIdentifier,* icp::DataDescriptor & *lDataDesc,* int32_t *lROI_XOffset,* int32_t *lROI_YOffset,* int32_t *lROI_Width,* int32_t *lROI_Height*) `[inherited]`

Connect the region of interest (ROI) described by "lDataDesc", "lROI_XOffset", "lROI_YOffset", "lROI_Width", and "l↵ ROI_Height" to graph port "lPortIdentifier".

Parameters

`in`	*lPortIdentifier*	Graph port identifier.
`in`	*lDataDesc*	Description of contiguous data region that 'contains' the ROI.
`in`	*lROI_XOffset*	The X offset of the top left corner of the ROI (relative to the top left corner of the region described by "lDataDesc").
`in`	*lROI_YOffset*	The Y offset of the top left corner of the ROI (relative to the top left corner of the region described by "lDataDesc").
`in`	*lROI_Width*	The width of the ROI.
`in`	*lROI_Height*	The height of the ROI.

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.7 int32_t ACF_Process::ConnectIO_ROI (std::string *lPortIdentifier,* const vsdk::SUMat & *lUmat,* int32_t *lROI_XOffset,* int32_t *lROI_YOffset,* int32_t *lROI_Width,* int32_t *lROI_Height*) [inherited]

Connect the region of interest (ROI) described by "lUmat", "lROI_XOffset", "lROI_YOffset", "lROI_Width", and "lROI_↩ Height" to graph port "lPortIdentifier".

Parameters

in	*lPortIdentifier*	Graph port identifier.
in	*lUmat*	Description of contiguous data region that 'contains' the ROI.
in	*lROI_XOffset*	The X offset of the top left corner of the ROI (relative to the top left corner of the region described by "lUmat").
in	*lROI_YOffset*	The Y offset of the top left corner of the ROI (relative to the top left corner of the region described by "lUmat").
in	*lROI_Width*	The width of the ROI.
in	*lROI_Height*	The height of the ROI.

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.8 virtual int32_t ACF_Process_APU::Initialize ( ) [pure virtual]

Initialize the APU process. This must be invoked prior to any configuration or execution calls.

Returns

0 if successful, non-zero if an error occurred.

Implements ACF_Process.

5.4.2.9 int32_t ACF_Process_APU::QueryPortChunkSize (std::string *lPortIdentifier,* int32_t & *lChunkWidth,* int32_t & *lChunkHeight*)

Return the the chunk width and height associated with port "lPortIdentifier". It is only meaningful to call this method after a successful call to SelectScenario(...).

Parameters

in	*lPortIdentifier*	Graph port identifier.
out	*lChunkWidth*	Chunk width associated with port "lPortIdentifier"
out	*lChunkHeight*	Chunk height associated with port "lPortIdentifier"

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.10 AcfProfilingInfo ACF_Process_APU::RetAcfProfilingInfo ( )

Return profiling information associated with the last process execution. It should be called only after Start() and Wait() have completed.

```
myProcess.Start();
myProcess.Wait();
AcfProfilingInfo profInfo = myProcess.RetAcfProfilingInfo();
```

Returns

Returns an AcfProfilingInfo struct populated with the results of the last process execution.

5.4.2.11 int32_t ACF_Process_APU::SelectApuConfiguration (ACF_APU_CFG *lApuConfig,* int32_t *lApexId*)

Select a specific APU configuration and a specific APEX on which to execute the process. This method allows for multiple processes to be executed simultaneously on the same APEX (assuming HW resource availability).

For example, given a single 642 APEX configuration, run myProcessA on APU0 w/ 32 CUs and run myProcessB on APU1 w/ 32 CUs:

```
if (0 == myProcessA.SelectApuConfiguration(ACF_APU_CFG__APU_0_CU_0_31_SMEM_0_1
      , 0) &&
    0 == myProcessB.SelectApuConfiguration(ACF_APU_CFG__APU_1_CU_32_63_SMEM_2_3
      , 0))
{
   lRetVal |= myProcessA.Start();
   lRetVal |= myProcessB.Start();

   lRetVal |= myProcessA.Wait();
   lRetVal |= myProcessB.Wait();
}
```

Parameters

in	*lApuConfig*	Desired APU configuration (see definition of ACF_APU_CFG for available options)
in	*lApexId*	The ID of the desired APEX (e.g if there are 2 APEXs, valid values for lApexId would be 0 and 1).

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.12 int32_t ACF_Process_APU::SelectScenario (std::string *lPortIdentifier,* int32_t *lChunkWidth,* int32_t *lChunkHeight*)

This method is used to force a specific scenario to be selected. A successful call to SelectScenario(...) will override the scenario selection that typically takes place when Start() is called. If this function is called, it is assumed that the user is in charge of explicit scenario selection for the duration of the object's life-span. The following examples demonstrate how it can be used:

```
//Usage example 1: for port "myPort", select the scenario where the chunk width is 8 and the chunk height
       is 4
A: SelectScenario("myPort", 8, 4);


//Usage example 2: for port "myPort", select the scenario where the chunk width is 16 and the chunk height
       is optimal (i.e. the largest available)
A: SelectScenario("myPort", 16, 0); //'0' indicates that the choice should be left to ACF


//Usage example 3: for port "myPort", select the scenario where the chunk height is 8, and the chunk width
       is optimal (i.e. chosen to maximize CU utilization)
A: SelectScenario("myPort", 0, 8); //'0' indicates that the choice should be left to ACF


//Usage example 4: select the 'ideal' scenario (i.e. first choose chunk width to maximize CU utilization,
       then choose the largest corresponding chunk height)
SelectScenario("", 0, 0); //'0' indicates that the choice should be left to ACF
```

Parameters

in	*lPortIdentifier*	Graph port identifier. This port must have all of the following properties: non-fixed & direct (i.e. not indirect) & non-static & vector.
in	*lChunkWidth*	Desired chunk width associated with port "lPortIdentifier" (or '0' if the choice should be left to ACF)
in	*lChunkHeight*	Desired chunk height associated with port "lPortIdentifier" (or '0' if the choice should be left to ACF)

Returns

0 if successful, non-zero if an an error occurred or if the desired scenario could not be found

5.4.2.13 int32_t ACF_Process::SetRoiInfo (int32_t *lRoiInfoL,* int32_t *lRoiInfoR,* int32_t *lRoiInfoT,* int32_t *lRoiInfoB*) [inherited]

[DEPRECATED] Globally indicate how much data beyond 2D input borders should be taken into account for the region of interest (ROI) case. If these are set to non-zero values, the indicated data must be available on the borders of ALL applicable inputs.

Parameters

in	*lRoiInfoL*	Number of source elements available beyond the left border of the 2D input region. It must be a multiple of chunk width.
in	*lRoiInfoR*	Number of source elements available beyond the right border of the 2D input region. It must be a multiple of chunk width.
in	*lRoiInfoT*	Number of source elements available beyond the top border of the 2D input region. It must be a multiple of chunk height.
in	*lRoiInfoB*	Number of source elements available beyond the bottom border of the 2D input region. It must be a multiple of chunk height.

Returns

0 if successful, non-zero if an an error occurred.

5.4.2.14 virtual int32_t ACF_Process_APU::Start ( ) [virtual]

Launch the process. This is a non-blocking call, and must (eventually) be paired with a Wait() call.

Returns

> 0 if successful, non-zero if an error occurred. The return value will correspond to one of the ACF error codes defined in acf_common.h.

Implements ACF_Process.

5.4.2.15 int32_t ACF_Process_APU::Start (void(∗)(void ∗lpParam, int32_t ∗lpRetVal) *lpCallback,* void ∗ *lpCallbackParam,* int32_t ∗ *lpCallbackRetVal*)

Launch the process with a user specified callback. The callback "lpCallback" will be invoked with the parameters defined by "lpCallbackParam" and "lpCallbackRetVal" when process execution has completed. This is a non-blocking call, and it must (eventually) be paired with a Wait() call. The following code fragment illustrates a simple callback example:

```
void MyCallback(void* lpParam, int32_t* lpRetVal)
{
   int32_t lRetVal = 0;
   MyStruct* lpMyStruct = (MyStruct*)lpParam;

   //<do something>

   if (0 != lpRetVal)
      *lpRetVal = lRetVal;
}

void StartWithCallbackExample()
{
   MyProcess lMyProcess;
   lMyProcess.Initialize();

   //<connect IOs to process>

   MyStruct lMyStruct;
   int32_t  lMyRetval;
   lMyProcess.Start(MyCallback, (void*)&lMyStruct, &lMyRetval);
   lMyProcess.Wait();
}
```

Parameters

in	*lpCallback*	Callback function that will be invoked upon process completion.
in	*lpCallbackParam*	Pointer to callback parameter (e.g. pointer to a value, struct, array, etc.). Can be 0 if the callback doesn't use it.
in	*lpCallbackRetVal*	Pointer to callback return value. Can be 0 if the callback doesn't use it. Please note that ACF will not examine or draw any conclusions based on the value of ∗lpCallbackRetVal (i.e. it is for use by the user).

Returns

> 0 if successful, non-zero if an error occurred.

5.4.2.16 virtual int32_t ACF_Process_APU::Wait ( ) [virtual]

Wait for a launched process to complete.

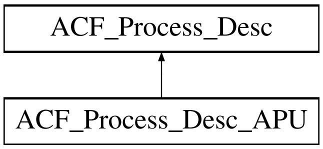
Returns

> 0 if successful, non-zero if an error occurred. The return value will correspond to one of the ACF error codes defined in acf_common.h. If a serious error like ACF_TIMEOUT_ERROR occurs, it may be necessary to call APEX_Reset(...) on the target APEX to recover (APEX_Reset is defined in apex.h). APEX_Reset(...) should only be called when the target APEX is not being used by any other ACF process.

Implements ACF_Process.

5.5 ACF_Process_Desc Class Reference

Inheritance diagram for ACF_Process_Desc:



Public Member Functions

- virtual void Create ()=0
- int32_t Initialize (ACF_Graph &lGraph, std::string lProcessIdentifier)
- int32_t SetInputChunkSize (std::string lInputPortIdentifier, int32_t lChunkWidth, int32_t lChunkHeight)
- int32_t FlagInputAsChunkBasedIndirect (std::string lInputPortIdentifier)

5.5.1 Member Function Documentation

5.5.1.1 virtual void ACF_Process_Desc::Create ( ) `[pure virtual]`

This is a pure virtual method that must be implemented by the derived class.

5.5.1.2 int32_t ACF_Process_Desc::FlagInputAsChunkBasedIndirect (std::string *lInputPortIdentifier*)

Indicate that the input will be a 2D table of pointers to chunks of data (instead of contiguous data). This allows for the processing of non-contiguous chunks of data. Use "SetInputChunkSize(...)" to select the input chunk size (all data chunks are assumed to be the same size).

Parameters

`in`	*lInputPortIdentifier*	Input port identifier. This must specify a non-static vector port with no spatial dependencies.

Returns

0 if successful, non-zero if port "lInputPortIdentifier" could not be found.

5.5.1.3 int32_t ACF_Process_Desc::Initialize (ACF_Graph & *lGraph,* std::string *lProcessIdentifier*)

Associate the graph "lGraph" with the process and give the process a unique identifier "lProcessIdentifier". The chosen process identifier will be used as a root name for generated output entities.

eg.

```
Initialize(mMyTestGraph, "MY_TEST_PROCESS");
```

Parameters

in	*lGraph*	Graph associated with the process.
in	*lProcessIdentifier*	Process identifier. Process identifier length should not exceed 64 characters.

Returns

0 if successful, non-zero if creation of "lGraph" failed.

5.5.1.4 int32_t ACF_Process_Desc::SetInputChunkSize (std::string *lInputPortIdentifier,* int32_t *lChunkWidth,* int32_t *lChunkHeight*)

Set the input chunk size (in units of e0) for port "lInputPortIdentifier" to "lChunkWidth" by "lChunkHeight".

eg.

```
SetInputChunkSize("GRAPH_INPUT_0", 8, 1);
```

Parameters

in	*lInputPortIdentifier*	Input port identifier.
in	*lChunkWidth*	Chunk width in units of e0.
in	*lChunkHeight*	Chunk height in units of e0.

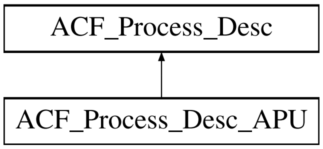
Returns

0 if successful, non-zero if port "lInputPortIdentifier" could not be found.

5.6 ACF_Process_Desc_APU Class Reference

`#include <acf_process_desc_apu.hpp>`

Inheritance diagram for ACF_Process_Desc_APU:



Public Member Functions

- virtual void Create ()=0
- int32_t Initialize (ACF_Graph &lGraph, std::string lProcessIdentifier)
- int32_t SetInputChunkSize (std::string lInputPortIdentifier, int32_t lChunkWidth, int32_t lChunkHeight)
- int32_t FlagInputAsChunkBasedIndirect (std::string lInputPortIdentifier)

Protected Member Functions

- void RtlSim_Init (int32_t lArrayWidth, int32_t lDmaChIn, int32_t lDmaChOut, int32_t lSmemAddrFromDmaPersp)
- void RtlSim_ConnectIO (std::string lPortIdentifier, int32_t lWidth, int32_t lHeight, int32_t lSpan, icp::DATATYPE lElementType, int32_t lElementDimX, int32_t lElementDimY, uint32_t lAddrPhys)
- void RtlSim_ConnectIndirectInput (std::string lPortIdentifier, int32_t lWidth, int32_t lHeight, int32_t lSpan, icp::DATATYPE lElementType, int32_t lElementDimX, int32_t lElementDimY, uint32_t lAddrPhys, int32_t lOffsetWidth, int32_t lOffsetHeight, int32_t lOffsetSpan, icp::DATATYPE lOffsetElementType, int32_t lOffsetElementDimX, int32_t lOffsetElementDimY, uint32_t lOffsetAddrPhys)

5.6.1 Detailed Description

ACF_Process_Desc_APU is a base class designed to encapsulate the configuration or 'description' of a process. It effectively links a graph with the APU processor and allows for any required APU specific configuration. In order to create an APU process description, a user must derive from this class and implement the pure virtual Create() method.

5.6.2 Member Function Documentation

5.6.2.1 virtual void ACF_Process_Desc::Create ( ) [pure virtual], [inherited]

This is a pure virtual method that must be implemented by the derived class.

5.6.2.2 int32_t ACF_Process_Desc::FlagInputAsChunkBasedIndirect (std::string *lInputPortIdentifier*) [inherited]

Indicate that the input will be a 2D table of pointers to chunks of data (instead of contiguous data). This allows for the processing of non-contiguous chunks of data. Use "SetInputChunkSize(...)" to select the input chunk size (all data chunks are assumed to be the same size).

Parameters

in	*lInputPortIdentifier*	Input port identifier. This must specify a non-static vector port with no spatial dependencies.

Returns

0 if successful, non-zero if port "lInputPortIdentifier" could not be found.

5.6.2.3 int32_t ACF_Process_Desc::Initialize (ACF_Graph & *lGraph*, std::string *lProcessIdentifier*) [inherited]

Associate the graph "lGraph" with the process and give the process a unique identifier "lProcessIdentifier". The chosen process identifier will be used as a root name for generated output entities.

eg.

```
Initialize(mMyTestGraph, "MY_TEST_PROCESS");
```

Parameters

in	*lGraph*	Graph associated with the process.
in	*lProcessIdentifier*	Process identifier. Process identifier length should not exceed 64 characters.

Returns

0 if successful, non-zero if creation of "lGraph" failed.

5.6.2.4 void ACF_Process_Desc_APU::RtlSim_ConnectIndirectInput (std::string *lPortIdentifier*, int32_t *lWidth*, int32_t *lHeight*, int32_t *lSpan*, icp::DATATYPE *lElementType*, int32_t *lElementDimX*, int32_t *lElementDimY*, uint32_t *lAddrPhys*, int32_t *lOffsetWidth*, int32_t *lOffsetHeight*, int32_t *lOffsetSpan*, icp::DATATYPE *lOffsetElementType*, int32_t *lOffsetElementDimX*, int32_t *lOffsetElementDimY*, uint32_t *lOffsetAddrPhys*) [protected]

This function is strictly used for configuring indirect inputs for RTL-SIM binary generation purposes.

Input data region:

Parameters

in	*lPortIdentifier*	Port identifier.
in	*lWidth*	Width (in elements) of the contiguous data region.
in	*lHeight*	Height (in elements) of the contiguous data region.
in	*lSpan*	Span is defined as the number of bytes required to jump from one line of bytes in memory to the 'next' line of bytes in memory. Note that span must be divisible by N where N = RetIcpDataTypeSizeInBytes (lElementDataType).
in	*lElementDataType*	The data type associated with an 'element' (i.e. the smallest unit of data)
in	*lElementDimX*	The 'x' dimension (i.e. width) of an element in units of "lElementDataType"
in	*lElementDimY*	The 'y' dimension (i.e. height) of an element in units of "lElementDataType"
in	*lAddrPhys*	Physical address of the start of the contiguous data region. This will depend on your HW setup and should correspond to general purpose or external memory.

For input offset array:

Parameters

Parameters

in	*lOffsetPortIdentifier*	Offset Port identifier.
in	*lOffsetWidth*	Width (in elements) of the contiguous data region.
in	*lOffsetHeight*	Height (in elements) of the contiguous data region.
in	*lOffsetSpan*	Span is defined as the number of bytes required to jump from one line of bytes in memory to the 'next' line of bytes in memory. Note that span must be divisible by N where N = RetIcpDataTypeSizeInBytes (lElementDataType).
in	*lOffsetElementDataType*	The data type associated with an 'element' (i.e. the smallest unit of data)
in	*lOffsetElementDimX*	The 'x' dimension (i.e. width) of an element in units of "lOffsetElementDataType"
in	*lOffsetElementDimY*	The 'y' dimension (i.e. height) of an element in units of "lOffsetElementDataType"
in	*lOffsetAddrPhys*	Physical address of the start of the contiguous data region. This will depend on your HW setup and should correspond to general purpose or external memory.

5.6.2.5 void ACF_Process_Desc_APU::RtlSim_ConnectIO (std::string *lPortIdentifier,* int32_t *lWidth,* int32_t *lHeight,* int32_t *lSpan,* icp::DATATYPE *lElementType,* int32_t *lElementDimX,* int32_t *lElementDimY,* uint32_t *lAddrPhys*) `[protected]`

This function is strictly used for configuring IOs for RTL-SIM binary generation purposes. By providing a basic IO configuration, it is possible to generate a fully configured process that is ready to execute in an RTL-SIM environment.

Parameters

in	*lPortIdentifier*	Port identifier.
in	*lWidth*	Width (in elements) of the contiguous data region.
in	*lHeight*	Height (in elements) of the contiguous data region.
in	*lSpan*	Span is defined as the number of bytes required to jump from one line of bytes in memory to the 'next' line of bytes in memory. Note that span must be divisible by N where N = RetIcpDataTypeSizeInBytes (lElementDataType).
in	*lElementDataType*	The data type associated with an 'element' (i.e. the smallest unit of data)
in	*lElementDimX*	The 'x' dimension (i.e. width) of an element in units of "lElementDataType"
in	*lElementDimY*	The 'y' dimension (i.e. height) of an element in units of "lElementDataType"
in	*lAddrPhys*	Physical address of the start of the contiguous data region. This will depend on your HW setup and should correspond to general purpose or external memory.

5.6.2.6 void ACF_Process_Desc_APU::RtlSim_Init (int32_t *lArrayWidth,* int32_t *lDmaChIn,* int32_t *lDmaChOut,* int32_t *lSmemAddrFromDmaPersp*) `[protected]`

Initialize various parameters required by the framework to generate an RTL-SIM ready binary.

Parameters

in	*lArrayWidth*	Desired number of CUs in the array (must be 32 or 64)
in	*lDmaChIn*	Input DMA channel (typically 0)
in	*lDmaChOut*	Output DMA channel (typically 1)

Parameters

in	*lSmemAddrFromDmaPersp*	The address of SMEM from the DMA perspective (i.e. from the host or FPGA perspective depending on your HW setup, not the APU perspective)

5.6.2.7 int32_t ACF_Process_Desc::SetInputChunkSize (std::string *lInputPortIdentifier,* int32_t *lChunkWidth,* int32_t *lChunkHeight*) [inherited]

Set the input chunk size (in units of e0) for port "lInputPortIdentifier" to "lChunkWidth" by "lChunkHeight".

eg.

```
SetInputChunkSize("GRAPH_INPUT_0", 8, 1);
```

Parameters

in	*lInputPortIdentifier*	Input port identifier.
in	*lChunkWidth*	Chunk width in units of e0.
in	*lChunkHeight*	Chunk height in units of e0.

Returns

0 if successful, non-zero if port "lInputPortIdentifier" could not be found.

Index