# Digital Signal Processing using CUDA

1.0

Generated by Doxygen 1.8.1.2

Mon Feb 3 2014 22:29:06

# Contents

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 DataReader Class Reference

### Public Member Functions

- **DataReader** (const std::string &filename, InputBuffer *buffer, int chunksize)
- void **readToBuffer** ()
- int **get_nSamp** ()
- int **get_nSeg** ()
- int **get_nWf** ()

### Static Public Member Functions

- static int **readHeader** (const std::string &filename, int &nSample, int &nSegment, int &nWaveform)

The documentation for this class was generated from the following file:

- DSP/src/DataReader.h

## 3.2 fitData Struct Reference

### Public Attributes

- float **param** [COUNTPARAM]
- float **startValue**
- float **endValue**
- float **extremumPos**
- float **extremumValue**
- float **euclidNormResidues**
- float **averageAbsResidues**
- int **status**

The documentation for this struct was generated from the following file:

- DSP/src/Types.h

## 3.3 Node Class Reference

`#include <Node.h>`

### Public Member Functions

- Node (int deviceIdentifier, InputBuffer *input, OutputBuffer *output)

  *Copy one chunk of data to the GPU and the result back to the output buffer.*

### 3.3.1 Detailed Description

Each installed device should be handled by its own thread. This class provides all functions to create a thread, copy data to and from the device and start the kernel on the device.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 Node::Node ( int *deviceIdentifier,* InputBuffer * *input,* OutputBuffer * *output* )

Copy one chunk of data to the GPU and the result back to the output buffer.

**Parameters**

| | |
|---|---|
| *texArray* | Location on the GPU, where the raw data will be copied to. |
| *fitData* | Location on the GPU, where the result will be written to.Basic constructor. |

Stats a new Thread. The new Thread reads data from the input buffer, copies them to the gpu and copy the result back to the output buffer.

**Parameters**

| | |
|---|---|
| *deviceIdentifier* | Number of the Device |
| *input* | Buffer which provides the raw input data. |
| *output* | Buffer which will be filled with the result data. |

The documentation for this class was generated from the following file:

- DSP/src/Node.h

## 3.4 OutputStream Class Reference

`#include <OutputStream.h>`

### Public Member Functions

- OutputStream (const std::string &file, int producer)

  *Basic constructor.*
- Ringbuffer< Output > * getBuffer ()

  *Returns a reference of the buffer.*
- void join ()

  *Waits until the writing thread to stops.*

### 3.4.1 Detailed Description

Class that provides all functions to write the results of the computation into a file.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 OutputStream::OutputStream ( const std::string & *file*, int *producer* )

Basic constructor.

Constructor opens a filestream, initialise the output buffer and start the thread, which takes elements from the buffers and writes them into the file.

**Parameters**

| | |
|---|---|
| *file* | Filename of the output file. |

The documentation for this class was generated from the following file:

- DSP/src/OutputStream.h

## 3.5 Ringbuffer< Type > Class Template Reference

A ringbuffer template supporting non-host consumers/producers.

`#include <Ringbuffer.h>`

### Public Member Functions

- Ringbuffer (const unsigned int bSize, int producer, Type defaultItem)
- Ringbuffer (const unsigned int bSize, int producer)
- int writeFromHost (Type &inputOnHost)
- int copyToHost (Type &outputOnHost)
- Type * reserveHead ()
- int freeHead ()
- Type * reserveTailTry ()
- int freeTail ()
- int getSize ()
- bool isEmpty ()
- bool isFinished ()
- void producerQuit ()

### 3.5.1 Detailed Description

**template<class Type>class Ringbuffer< Type >**

A ringbuffer template supporting non-host consumers/producers.

Ringbuffer Data is written to the head of the buffer and read from the tail. To enable reading to devices like graphic cards the tail of the buffer can be reserved. In the reserved state copy operations can be performed externally. After copying the head needs to be freed. The same mechanism is available for writing to the buffer from other devices. For data reading/writing from host to host classic write/read methods are available.

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 template<class Type> Ringbuffer< Type >::Ringbuffer ( const unsigned int *bSize*, int *producer*, Type *defaultItem* )

Constructor for dynamic size elements.

Reserves buffer memory. The buffer holds bSize items. The items consist of itemSize elements of type Type. These elements may be of a dynamic size type but they need to have the same size.

**Parameters**

| | |
|---|---|
| *bSize* | Amount of items the buffer can hold. |
| *producer* | number of producers feeding the buffer. |
| *defaultItem* | A default item to store in the buffer. This fixes the memory available for variable length types like std::vector. |

#### 3.5.2.2 template<class Type> Ringbuffer< Type >::Ringbuffer ( const unsigned int *bSize*, int *producer* )

Fixed size type Constructor.

For Type with fixed size no defaultItem is needed.

**Parameters**

| | |
|---|---|
| *bSize* | Amount of items the buffer can hold. |
| *producer* | Number of producers feeding the buffer. |

### 3.5.3 Member Function Documentation

#### 3.5.3.1 template<class Type> int Ringbuffer< Type >::copyToHost ( Type & *outputOnHost* )

Read data from the buffer to the host.

The call blocks until there is data available in the buffer. The call blocks if the buffer is already used by another thread.

**Parameters**

| | |
|---|---|
| *outputOnHost* | Pointer to host memory where buffer data is to be written. |

#### 3.5.3.2 template<class Type > int Ringbuffer< Type >::freeHead ( )

Unlock buffer after external write operation (using reserveHead) finished. All other calls to the buffer will block until freeHead() is called. Calling freeHead() wakes up other threads trying to read from an empty buffer.

#### 3.5.3.3 template<class Type > int Ringbuffer< Type >::freeTail ( )

Unlock buffer after external read operation (using reserveTail()) finished. All other calls to the buffer will block until freeTail() is called. Calling freeTail() wakes up other blocking threads trying to write to a full buffer.

#### 3.5.3.4 template<class Type > int Ringbuffer< Type >::getSize ( )

Get amount of items stored in buffer.

**Returns**

Number of items in buffer

#### 3.5.3.5 template<class Type > bool Ringbuffer< Type >::isEmpty ( )

Tell if buffer is empty.

**Returns**

True if no elements are in buffer. False otherwise.

#### 3.5.3.6 template<class Type > bool Ringbuffer< Type >::isFinished ( )

Tell if buffer is empty and will stay empty.

**Returns**

True if there are no elements in buffer and all producers announced that they stopped adding elements. False otherwise.

#### 3.5.3.7 template<class Type > void Ringbuffer< Type >::producerQuit ( )

Lets a producer announce that it is adding no more elements to the buffer. To be called only once per producer. This is not checked.

#### 3.5.3.8 template<class Type > Type * Ringbuffer< Type >::reserveHead ( )

Lock head position of buffer to perform write operations externally.

The call blocks until there is space available in the buffer.

Buffer is blocked until freeHead() is called.

**Returns**

Pointer to the head of the ringbuffer. One item of <Type> can be written here.

#### 3.5.3.9 template<class Type > Type * Ringbuffer< Type >::reserveTailTry ( )

Lock tail position of buffer to perform read/copy operation externally.

If there is no data in the buffer it returns NULL. The call blocks if another thread is using the buffer.

The buffer will block any other threads until freeTail() is called.

**Returns**

Pointer to data to be read or NULL if buffer is empty.

#### 3.5.3.10 template<class Type> int Ringbuffer< Type >::writeFromHost ( Type & *inputOnHost* )

Write data to the buffer from the host.

The call blocks if there is no space available on the buffer or if the buffer is already used by another thread.

**Parameters**

| | |
|---|---|
| *inputOnHost* | Needs to be on host memory. |

The documentation for this class was generated from the following file:

- DSP/src/Ringbuffer.h

# Chapter 4

# File Documentation

## 4.1 DSP/src/Constants.h File Reference

This File holds all configurations and constants.

```
#include <string>
```

### Variables

- const unsigned int SAMPLE_COUNT = 1000

  *Number of samples per event.*
- const unsigned int CHUNK_COUNT = 100

  *Number of events copied to the GPU in one step.*
- const unsigned int CHUNK_BUFFER_COUNT = 2048

  *Number of chunks in the input buffer.*
- const cudaTextureFilterMode FILTER_MODE = cudaFilterModeLinear

  *Interpolation mode.*
- const std::string **OUTPUT_FILENAME** = "results.txt"
- const std::string **FILENAME_TESTFILE** = "../data/Al_25keV-259.cdb"
- const unsigned int **SAMPLE_COUNT_TESTFILE** = 1000
- const unsigned int **SEGMENT_COUNT_TESTFILE** = 1
- const unsigned int **WAVEFORM_COUNT_TESTFILE** = 100000
- const unsigned int INTERPOLATION_COUNT = 20

  *Number of points that are averaged to on Datapoint. Higher Value decrease the resolution and increase the speed of the programm.*
- const unsigned int MAXCOUNTDATA = 1000

  *max. number of samples per event for compute capability 2.0 or higher - currently ca. 2450 is max. because (COUNTPARAM + 2) ∗ MAXCOUNTDATA ∗ sizeof(float) = 48 kB (= max. shared memory); for compute capability 1.x - currently ca. 800 is max. because (COUNTPARAM + 2) ∗ MAXCOUNTDATA ∗ sizeof(float) = 16 kB (= max. shared memory)*
- const unsigned int MAXCALL = 100

  *max. calls for Levenberg Marquardt until stops*
- const float FITVALUETHRESHOLD = 0.5

  *threshold between min (0.0) and max (1.0) value to define the data using interval to calculate the fit function*
- const float STARTENDPROPORTION = 0.01

  *proportion of countData for calculating the average of start/end value (e. g. 0.1 means average of the first 10% of data for start value and the last 10% for end value)*
- const unsigned int COUNTPARAM = 3

  *number of parameters for the fit function*

### 4.1.1 Detailed Description

This File holds all configurations and constants.

## 4.2 DSP/src/LevMarq.h File Reference

```
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <stdio.h>
#include "Types.h"
```

### Macros

- #define **CUDA**
- #define **GLOBAL** __global__
- #define **DEVICE** __device__
- #define **SHARED** __shared__
- #define **LM_MACHEP** FLT_EPSILON
- #define **LM_DWARF** FLT_MIN
- #define **LM_SQRT_DWARF** sqrt(FLT_MIN)
- #define **LM_SQRT_GIANT** sqrt(FLT_MAX)
- #define **LM_USERTOL** 30∗LM_MACHEP
- #define **MIN**(A, B) (((A) <= (B)) ? (A) : (B))
- #define **MAX**(A, B) (((A) >= (B)) ? (A) : (B))
- #define **SQR**(X) ((X) ∗ (X))

### Functions

- template<unsigned int tex>
  __device__ float getSample (float I, int INDEXDATASET)

  *getSample returns the y value of a given sample index*
- template<>
  __device__ float **getSample**< **0** > (float I, int INDEXDATASET)
- template<>
  __device__ float **getSample**< **1** > (float I, int INDEXDATASET)
- template<>
  __device__ float **getSample**< **2** > (float I, int INDEXDATASET)
- template<>
  __device__ float **getSample**< **3** > (float I, int INDEXDATASET)
- template<>
  __device__ float **getSample**< **4** > (float I, int INDEXDATASET)
- template<>
  __device__ float **getSample**< **5** > (float I, int INDEXDATASET)
- template<unsigned int tex>
  DEVICE void paramStartValue (int firstValue, int lastValue, int indexDataset, float ∗param)

  *paramStartValue returns the parameter start values for the fit-function calculation*
- DEVICE void fitFunction (float x, float ∗param, float ∗y)

  *fitFunction returns the y of a given x*
- DEVICE void fitFunctionExtremum (float ∗param, float ∗x)

  *fitFunctionExtremum returns the x of the min. or max. y value*

- template<unsigned int tex>
  DEVICE void evaluate (float *param, int countData, float *fvec, int indexDataset, int xOffset, float xStep)

  *evaluate calculates the residues between the given samples and the current fit-function*
- DEVICE void qrSolve (int n, float *r, int ldr, int *ipvt, float *diag, float *qtb, float *x, float *sdiag, float *wa)

  *qrSolve completes the solution of the problem if it is provided with the necessary information from the qr factorization, with column pivoting, of a*
- DEVICE void euclidNorm (int n, float *x, float *result)

  *euclidNorm calculates the euclidean norm of x*
- DEVICE void lmpar (int n, float *r, int ldr, int *ipvt, float *diag, float *qtb, float delta, float *par, float *x, float *sdiag, float *wa1, float *wa2)

  *lmpar determines a value for the parameter par such that x solves the system*
- DEVICE void qrFactorization (int m, int n, float *a, int pivot, int *ipvt, float *rdiag, float *acnorm, float *wa)

  *qrFactorization uses householder transformations with column pivoting (optional) to compute a qr factorization of the m by n matrix a*
- template<unsigned int tex>
  DEVICE void lmdif (int m, int n, float *x, float *fvec, float ftol, float xtol, float gtol, int maxfev, float epsfcn, float *diag, int mode, float factor, int *info, int *nfev, float *fjac, int *ipvt, float *qtf, float *wa1, float *wa2, float *wa3, float *wa4, int indexDataset, int xOffset, float xStep)

  *lmdif minimizes the sum of the squares of m nonlinear functions in n variables by a modification of the levenberg-marquardt algorithm*
- template<unsigned int tex>
  DEVICE void maxValue (int countData, int indexDataset, int *x, DATATYPE *y)

  *maxValue returns the x and y where y has the greatest value*
- template<unsigned int tex>
  DEVICE void averageValue (int start, int count, int indexDataset, float *y)

  *averageValue returns the average of all y values in a given range*
- template<unsigned int tex>
  DEVICE void xOfValue (int countData, int indexDataset, char fromDirection, DATATYPE minValue, int *x)

  *xOfValue returns the first x of a value y that is greater or equal of a given min. value*
- DEVICE void averageAbsResidues (int countResidues, float *residues, float *average)

  *averageAbsResidues returns the average of the residues absolute value*
- template<unsigned int tex>
  GLOBAL void kernel (int countData, float step, struct fitData *result)

  *kernel is the start method for calculation (you have to set the dataTexture (GPU mode) or data variable (CPU mode) before calling this method)*

## Variables

- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture0**
- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture1**
- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture2**
- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture3**
- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture4**
- texture< DATATYPE,
  2, cudaReadModeElementType > **dataTexture5**
- const char * **statusMessage** []

### 4.2.1 Detailed Description

### 4.2.2 Function Documentation

#### 4.2.2.1 DEVICE void averageAbsResidues ( int *countResidues*, float * *residues*, float * *average* )

averageAbsResidues returns the average of the residues absolute value

**Parameters**

| | |
|---|---|
| *countResidues* | number of residues |
| *residues* | array of length countResidues that contains the residues |
| *average* | the returned average |

#### 4.2.2.2 template<unsigned int tex> DEVICE void averageValue ( int *start*, int *count*, int *indexDataset*, float * *y* )

averageValue returns the average of all y values in a given range

**Parameters**

| | |
|---|---|
| *start* | first x for average calculation |
| *count* | number of values for average calculation |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *y* | the returned average |

#### 4.2.2.3 DEVICE void euclidNorm ( int *n*, float * *x*, float * *result* )

euclidNorm calculates the euclidean norm of x

**Parameters**

| | |
|---|---|
| *n* | length of array x |
| *x* | array for euclidean norm |
| *result* | euclidean norm of x |

calculation of norm

#### 4.2.2.4 template<unsigned int tex> DEVICE void evaluate ( float * *param*, int *countData*, float * *fvec*, int *indexDataset*, int *xOffset*, float *xStep* )

evaluate calculates the residues between the given samples and the current fit-function

**Parameters**

| | |
|---|---|
| *param* | parameters to define the concrete current fit-function |
| *countData* | number of samples |
| *fvec* | the returned residues |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *xOffset* | first x value that is used to calculate the fit-function |
| *xStep* | the distance between two x values that are used to calculate the fit-function (if decimal then y values will be interpolated) |

#### 4.2.2.5 DEVICE void fitFunction ( float *x*, float * *param*, float * *y* ) [inline]

fitFunction returns the y of a given x

**Parameters**

| | |
|---|---|
| *x* | given x value to calculate y |
| *param* | parameters to define the concrete current fit-function |
| *y* | the returned y value |

#### 4.2.2.6 DEVICE void fitFunctionExtremum ( float * *param*, float * *x* ) [inline]

fitFunctionExtremum returns the x of the min. or max. y value

**Parameters**

| | |
|---|---|
| *param* | parameters to define the concrete current fit-function |
| *x* | the returned x value |

#### 4.2.2.7 template<unsigned int tex> __device__ float getSample ( float *l*, int *INDEXDATASET* )

getSample returns the y value of a given sample index

**Parameters**

| | |
|---|---|
| *l* | sample index |
| *INDEXDATASET* | index of the current dataset (GPU mode) or not used (CPU mode) |

**Returns**

y value

#### 4.2.2.8 template<unsigned int tex> GLOBAL void kernel ( int *countData*, float *step*, struct fitData * *result* )

kernel is the start method for calculation (you have to set the dataTexture (GPU mode) or data variable (CPU mode) before calling this method)

**Parameters**

| | |
|---|---|
| *countData* | number of samples |
| *step* | the distance between two x values that are used to calculate the fit-function (if decimal then y values will be interpolated) |
| *result* | fit-function and other parameters, defined in fitData struct |

#### 4.2.2.9 template<unsigned int tex> DEVICE void lmdif ( int *m*, int *n*, float * *x*, float * *fvec*, float *ftol*, float *xtol*, float *gtol*, int *maxfev*, float *epsfcn*, float * *diag*, int *mode*, float *factor*, int * *info*, int * *nfev*, float * *fjac*, int * *ipvt*, float * *qtf*, float * *wa1*, float * *wa2*, float * *wa3*, float * *wa4*, int *indexDataset*, int *xOffset*, float *xStep* )

lmdif minimizes the sum of the squares of m nonlinear functions in n variables by a modification of the levenberg-marquardt algorithm

**Parameters**

| | |
|---|---|
| *m* | number of samples |
| *n* | number of parameters |
| *x* | input: must contain an initial estimate of the solution vector; output: contains the final estimate of the solution vector |
| *fvec* | an output array of length m which contains the functions evaluated at the output x |
| *ftol* | measures the relative error desired in the sum of squares |
| *xtol* | measures the relative error desired in the approximate solution |
| *gtol* | gtol measures the orthogonality desired between the function vector and the columns of the jacobian |
| *maxfev* | a integer input variable that is used to terminate when the number of calls is at least maxfev by the end of an iteration |
| *epsfcn* | an input variable used in determining a suitable step length for the forward-difference approximation |
| *mode* | if mode = 1 then the variables will be scaled internally, if mode = 2 then the scaling is specified by the input diag |
| *factor* | a input variable used in determining the initial step bound. This bound is set to the product of factor and the euclidean norm of diag∗x |
| *info* | an integer output variable that indicates the termination status of lmdif (see statusMessage) |
| *nfev* | an output variable set to the number of calls to the user-supplied routine ∗evaluate |
| *fjac* | an output m by n array. The upper n by n submatrix of fjac contains an upper triangular matrix r with diagonal elements of nonincreasing magnitude |
| *ipvt* | an integer output array of length n that defines a permutation matrix p such that jac∗p = q∗r |
| *qtf* | an output array of length n which contains the first n elements of the vector (q transpose)∗fvec |
| *wa1* | work array of length n |
| *wa2* | work array of length n |
| *wa3* | work array of length n |
| *wa4* | work array of length m |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *xOffset* | first x value that is used to calculate the fit-function |
| *xStep* | the distance between two x values that are used to calculate the fit-function (if decimal then y values will be interpolated) |

#### 4.2.2.10 DEVICE void lmpar ( int *n,* float ∗ *r,* int *ldr,* int ∗ *ipvt,* float ∗ *diag,* float ∗ *qtb,* float *delta,* float ∗ *par,* float ∗ *x,* float ∗ *sdiag,* float ∗ *wa1,* float ∗ *wa2* )

lmpar determines a value for the parameter par such that x solves the system

**Parameters**

| | |
|---|---|
| *n* | width and height of array r |
| *ldr* | a positive integer input variable not less than n which specifies the leading dimension of the array r |
| *ipvt* | an integer input array of length n which defines the permutation matrix p such that a∗p = q∗r |
| *diag* | an input array of length n which must contain the diagonal elements of the matrix d |
| *qtb* | an input array of length n which must contain the first n elements of the vector (q transpose)∗b |
| *delta* | a positive input variable which specifies an upper bound on the euclidean norm of d∗x |
| *par* | input: contains an initial estimate of the levenberg-marquardt parameter; output: contains the final estimate |
| *x* | an output array of length n which contains the least squares solution of the system a∗x = b, d∗x = 0 |
| *sdiag* | an output array of length n which contains the diagonal elements of the upper triangular matrix s |
| *wa1* | work array of length n |
| *wa2* | work array of length n |

#### 4.2.2.11 template<unsigned int tex> DEVICE void maxValue ( int *countData,* int * *x,* DATATYPE * *y* )

maxValue returns the x and y where y has the greatest value

**Parameters**

| | |
|---|---|
| *countData* | number of samples |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *x* | the returned x value |
| *y* | the returned y value |

#### 4.2.2.12 template<unsigned int tex> DEVICE void paramStartValue ( int *firstValue,* int *lastValue,* int *indexDataset,* float * *param* )

paramStartValue returns the parameter start values for the fit-function calculation

**Parameters**

| | |
|---|---|
| *firstValue* | first value of the data used for fit-function |
| *lastValue* | last value of the data used for fit-function |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *param* | the returned parameter start values |

#### 4.2.2.13 DEVICE void qrFactorization ( int *m,* int *n,* float * *a,* int *pivot,* int * *ipvt,* float * *rdiag,* float * *acnorm,* float * *wa* )

qrFactorization uses householder transformations with column pivoting (optional) to compute a qr factorization of the m by n matrix a

**Parameters**

| | |
|---|---|
| *m* | height of array a |
| *n* | width of array a |
| *a* | input: contains the matrix for which the qr factorization is to be computed; output: the strict upper trapezoidal part of a contains the strict upper trapezoidal part of r, and the lower trapezoidal part of a contains a factored form of q |
| *pivot* | if is set true then column pivoting is enforced; if is set false then no column pivoting is done |
| *ipvt* | defines the permutation matrix p such that a∗p = q∗r |
| *rdiag* | an output array of length n which contains the diagonal elements of r |
| *acnorm* | an output array of length n which contains the norms of the corresponding columns of the input matrix a |
| *wa* | work array of length n |

#### 4.2.2.14 DEVICE void qrSolve ( int *n,* float * *r,* int *ldr,* int * *ipvt,* float * *diag,* float * *qtb,* float * *x,* float * *sdiag,* float * *wa* )

qrSolve completes the solution of the problem if it is provided with the necessary information from the qr factorization, with column pivoting, of a

**Parameters**

| | |
|---|---|
| *n* | width and height of array r |
| *ldr* | a positive integer input variable not less than n which specifies the leading dimension of the array r |
| *ipvt* | an integer input array of length n which defines the permutation matrix p such that a∗p = q∗r |
| *diag* | an input array of length n which must contain the diagonal elements of the matrix d |
| *qtb* | an input array of length n which must contain the first n elements of the vector (q transpose)∗b |

| | |
|---|---|
| *x* | an output array of length n which contains the least squares solution of the system a∗x = b, d∗x = 0 |
| *sdiag* | an output array of length n which contains the diagonal elements of the upper triangular matrix s |
| *wa* | work array of length n |

#### 4.2.2.15 template<unsigned int tex> DEVICE void xOfValue ( int *countData,* int *indexDataset,* char *fromDirection,* DATATYPE *minValue,* int ∗ *x* )

xOfValue returns the first x of a value y that is greater or equal of a given min. value

**Parameters**

| | |
|---|---|
| *countData* | number of samples |
| *indexDataset* | index of the current dataset (GPU mode) or not used (CPU mode) |
| *fromDirection* | |
| *minValue* | min. y value |
| *x* | the returned x value, -1 if there is no x with a y greater or equal minValue |

### 4.2.3 Variable Documentation

#### 4.2.3.1 const char∗ statusMessage[ ]

**Initial value:**

```
{
   "fatal coding error (improper input parameters)",
   "success (the relative error in the sum of squares is at most tol)",
   "success (the relative error between x and the solution is at most tol)",
   "success (the relative errors in the sum of squares and between x and the
      solution are at most tol)",
   "trapped by degeneracy (fvec is orthogonal to the columns of the jacobian)"
     ,
   "timeout (number of calls to fcn has reached maxcall*(n+1))",
   "failure (ftol<tol: cannot reduce sum of squares any further)",
   "failure (xtol<tol: cannot improve approximate solution any further)",
   "failure (gtol<tol: cannot improve approximate solution any further)"
}
```

# Index