TimeTagger 2.2.0

Generated by Doxygen 1.8.14

# **Contents**

1	TimeTagg	jer	1
2	Deprecate	ed List	3
3	Module In	ndex	5
	3.1 Mod	lules	5
4	Hierarchi	cal Index	7
	4.1 Clas	ss Hierarchy	7
5	Class Ind	ex	9
	5.1 Clas	ss List	9
6	Module D	ocumentation	11
	6.1 Time	eTagger backend	11
	6.1.	1 Detailed Description	11
	6.2 base	e iterators	12
	6.2.	1 Detailed Description	13
	6.2.2		13
		6.2.2.1 Dump()	13
		6.2.2.2 TimeTagStream()	13

ii CONTENTS

7	Clas	s Docu	mentation		15
	7.1	_Iterate	or Class Refere	ence	. 15
		7.1.1	Detailed Desc	pription	. 17
		7.1.2	Constructor 8	Destructor Documentation	. 17
			7.1.2.1 _lte	erator()	. 17
			7.1.2.2 ~_	Iterator()	. 17
		7.1.3	Member Fund	ction Documentation	. 17
			7.1.3.1 cle	ar()	. 17
			7.1.3.2 cle	ar_impl()	. 17
			7.1.3.3 get	CaptureDuration()	. 18
			7.1.3.4 get	NewVirtualChannel()	. 18
			7.1.3.5 isR	unning()	. 18
			7.1.3.6 loc	k()	. 18
			7.1.3.7 nex	ct_impl()	. 18
			7.1.3.8 reg	isterChannel()	. 19
			7.1.3.9 sta	rt()	. 19
			7.1.3.10 sta	rtFor()	. 19
			7.1.3.11 sto	p()	. 20
			7.1.3.12 unl	ock()	. 20
			7.1.3.13 unr	egisterChannel()	. 20
		7.1.4	Member Data	Documentation	. 20
			7.1.4.1 cha	annels_registered	. 20
			7.1.4.2 run	ning	. 21
	7.2	Array<	T > Class Te	mplate Reference	. 21
	7.3	Coincid	lence Class Re	eference	. 21
		7.3.1	Detailed Desc	cription	. 22
		7.3.2	Constructor 8	Destructor Documentation	. 22
			7.3.2.1 Co	incidence()	. 22
		7.3.3	Member Fund	ction Documentation	. 22
			7.3.3.1 get	Channel()	. 22

CONTENTS

7.4	Coincid	dences Class Reference	23
	7.4.1	Detailed Description	23
	7.4.2	Constructor & Destructor Documentation	24
		7.4.2.1 Coincidences()	24
	7.4.3	Member Function Documentation	24
		7.4.3.1 getChannels()	24
		7.4.3.2 next_impl()	24
7.5	Combi	ner Class Reference	25
	7.5.1	Detailed Description	25
	7.5.2	Constructor & Destructor Documentation	26
		7.5.2.1 Combiner()	26
	7.5.3	Member Function Documentation	26
		7.5.3.1 clear_impl()	26
		7.5.3.2 GET_DATA_1D()	26
		7.5.3.3 getChannel()	27
		7.5.3.4 next_impl()	27
7.6	Correla	ation Class Reference	27
	7.6.1	Detailed Description	28
	7.6.2	Constructor & Destructor Documentation	28
		7.6.2.1 Correlation()	28
	7.6.3	Member Function Documentation	29
		7.6.3.1 clear_impl()	29
		7.6.3.2 GET_DATA_1D() [1/2]	29
		7.6.3.3 GET_DATA_1D() [2/2]	29
		7.6.3.4 next_impl()	30
7.7	Count	BetweenMarkers Class Reference	30
	7.7.1	Detailed Description	31
	7.7.2	Constructor & Destructor Documentation	31
		7.7.2.1 CountBetweenMarkers()	31
	7.7.3	Member Function Documentation	32

iv CONTENTS

		7.7.3.1	clear_impl()	 . 32
		7.7.3.2	GET_DATA_1D() [1/3]	 . 32
		7.7.3.3	GET_DATA_1D() [2/3]	 . 32
		7.7.3.4	GET_DATA_1D() [3/3]	 . 33
		7.7.3.5	next_impl()	 . 33
		7.7.3.6	ready()	 . 33
7.8	Counte	er Class Re	Reference	 . 34
	7.8.1	Detailed	Description	 . 34
	7.8.2	Construc	ctor & Destructor Documentation	 . 34
		7.8.2.1	Counter()	 . 35
	7.8.3	Member	Function Documentation	 . 36
		7.8.3.1	clear_impl()	 . 36
		7.8.3.2	GET_DATA_2D()	 . 36
		7.8.3.3	next_impl()	 . 36
		7.8.3.4	start()	 . 37
7.9	Countra	ate Class	Reference	 . 37
	7.9.1	Detailed	Description	 . 38
	7.9.2	Construc	ctor & Destructor Documentation	 . 38
		7.9.2.1	Countrate()	 . 38
	7.9.3	Member	Function Documentation	 . 38
		7.9.3.1	clear_impl()	 . 38
		7.9.3.2	GET_DATA_1D()	 . 39
		7.9.3.3	next_impl()	 . 39
		7.9.3.4	start()	 . 39
7.10	Delaye	dChannel	I Class Reference	 . 40
	7.10.1	Detailed	Description	 . 40
	7.10.2	Construc	ctor & Destructor Documentation	 . 40
		7.10.2.1	DelayedChannel()	 . 40
	7.10.3	Member	Function Documentation	 . 41
		7.10.3.1	getChannel()	 . 41

CONTENTS

		7.10.3.2 next_impl()	41
		7.10.3.3 setDelay()	42
		7.10.3.4 start()	42
7.11	Dump (	Class Reference	42
7	7.11.1	Detailed Description	43
7	7.11.2	Constructor & Destructor Documentation	43
		7.11.2.1 ~Dump()	43
7	7.11.3	Member Function Documentation	43
		7.11.3.1 clear_impl()	43
		7.11.3.2 next_impl()	43
7.12 E	Event S	Struct Reference	44
7.13 F	Flim Cl	lass Reference	44
7	7.13.1	Detailed Description	45
7	7.13.2	Constructor & Destructor Documentation	45
		7.13.2.1 Flim()	45
7.14 F	Freque	encyMultiplier Class Reference	45
7	7.14.1	Detailed Description	46
7	7.14.2	Constructor & Destructor Documentation	46
		7.14.2.1 FrequencyMultiplier()	46
7	7.14.3	Member Function Documentation	47
		7.14.3.1 next_impl()	47
7.15	GatedC	Channel Class Reference	47
7	7.15.1	Detailed Description	48
7	7.15.2	Constructor & Destructor Documentation	48
		7.15.2.1 GatedChannel()	48
7	7.15.3	Member Function Documentation	49
		7.15.3.1 getChannel()	49
		7.15.3.2 next_impl()	49
7.16 H	Histogr	ram Class Reference	49
7	7.16.1	Detailed Description	50

vi

	7.16.2	Constructor & Destructor Documentation	50
		7.16.2.1 Histogram()	50
7.17	Histogr	amLogBins Class Reference	51
	7.17.1	Detailed Description	51
	7.17.2	Constructor & Destructor Documentation	51
		7.17.2.1 HistogramLogBins()	52
	7.17.3	Member Function Documentation	52
		7.17.3.1 clear_impl()	52
		7.17.3.2 next_impl()	52
7.18	Iterator	Class Reference	53
	7.18.1	Detailed Description	54
	7.18.2	Constructor & Destructor Documentation	54
		7.18.2.1 Iterator()	54
	7.18.3	Member Function Documentation	54
		7.18.3.1 clear_impl()	54
		7.18.3.2 next()	54
		7.18.3.3 next_impl()	55
		7.18.3.4 size()	55
7.19	LED U	nion Reference	55
7.20	PRBS	Class Reference	56
7.21	Scope	Class Reference	56
	7.21.1	Constructor & Destructor Documentation	57
		7.21.1.1 Scope()	57
	7.21.2	Member Function Documentation	57
		7.21.2.1 clear_impl()	57
		7.21.2.2 next_impl()	58
		7.21.2.3 start()	58
7.22	StartSt	op Class Reference	58
	7.22.1	Detailed Description	59
	7.22.2	Constructor & Destructor Documentation	59

CONTENTS vii

		7.22.2.1 StartStop()	59
	7.22.3	Member Function Documentation	60
		7.22.3.1 clear_impl()	60
		7.22.3.2 next_impl()	60
		7.22.3.3 start()	60
7.23	Synchr	onizedMeasurments Class Reference	61
	7.23.1	Detailed Description	61
	7.23.2	Constructor & Destructor Documentation	61
		7.23.2.1 SynchronizedMeasurments()	61
	7.23.3	Member Function Documentation	62
		7.23.3.1 clear()	62
		7.23.3.2 registerMeasurement()	62
		7.23.3.3 start()	62
		7.23.3.4 stop()	62
7.24	Tag Str	ruct Reference	63
	7.24.1	Detailed Description	63
	7.24.2	Member Data Documentation	63
		7.24.2.1 channel	63
		7.24.2.2 overflow	63
		7.24.2.3 time	64
7.25	TimeDi	fferences Class Reference	64
	7.25.1	Detailed Description	65
	7.25.2	Constructor & Destructor Documentation	65
		7.25.2.1 TimeDifferences()	65
	7.25.3	Member Function Documentation	65
		7.25.3.1 clear_impl()	66
		7.25.3.2 GET_DATA_2D()	66
		7.25.3.3 getCounts()	66
		7.25.3.4 next_impl()	66
		7.25.3.5 ready()	67

viii CONTENTS

7.25.3.6 setMaxCounts()	67
7.25.3.7 start()	67
agger Class Reference	68
Detailed Description	70
? Constructor & Destructor Documentation	70
7.26.2.1 TimeTagger()	70
7.26.2.2 ~TimeTagger()	70
Member Function Documentation	70
7.26.3.1 autoCalibration()	70
7.26.3.2 clearOverflows()	71
7.26.3.3 GET_DATA_2D() [1/2]	71
7.26.3.4 GET_DATA_2D() [2/2]	71
7.26.3.5 getChannelNumberScheme()	71
7.26.3.6 getChannels()	72
7.26.3.7 getConditionalFilterFiltered()	72
7.26.3.8 getConditionalFilterTrigger()	72
7.26.3.9 getDACRange()	72
7.26.3.10 getDeadtime()	72
7.26.3.11 getEventDivider()	73
7.26.3.12 getFilter()	73
7.26.3.13 getHardwareBufferSize()	73
7.26.3.14 getInputDelay()	73
7.26.3.15 getInvertedChannel()	74
7.26.3.16 getModel()	74
7.26.3.17 getNormalization()	74
7.26.3.18 getOverflows()	74
7.26.3.19 getOverflowsAndClear()	74
7.26.3.20 getPcbVersion()	75
7.26.3.21 getPsPerClock()	75
7.26.3.22 getSensorData()	75
	7.25.3.7 start() agger Class Reference  Detailed Description  Constructor & Destructor Documentation  7.26.2.1 TimeTagger()  7.26.2.2 ~TimeTagger()  8 Member Function Documentation  7.26.3.1 autoCalibration()  7.26.3.2 clearOverflows()  7.26.3.3 GET_DATA_2D() [1/2]  7.26.3.4 GET_DATA_2D() [2/2]  7.26.3.5 getChannelNumberScheme()  7.26.3.6 getChannelNumberScheme()  7.26.3.7 getConditionalFilterFiltered()  7.26.3.8 getConditionalFilterFiltered()  7.26.3.10 getDacRange()  7.26.3.11 getEventDivider()  7.26.3.12 getFilter()  7.26.3.13 getHardwareBufferSize()  7.26.3.14 getInputDelay()  7.26.3.15 getInvertedChannel()  7.26.3.16 getModel()  7.26.3.17 getNormalization()  7.26.3.18 getCoverflows()  7.26.3.19 getOverflows()  7.26.3.19 getOverflows()  7.26.3.20 getPcbVersion()  7.26.3.21 getPsPerClock()

CONTENTS

	7.26.3.23 getSerial()	75
	7.26.3.24 getTestSignal()	75
	7.26.3.25 getTriggerLevel()	76
	7.26.3.26 isUnusedChannel()	76
	7.26.3.27 registerChannel()	76
	7.26.3.28 reset()	76
	7.26.3.29 runSynchronized()	77
	7.26.3.30 setConditionalFilter()	77
	7.26.3.31 setDeadtime()	77
	7.26.3.32 setEventDivider()	78
	7.26.3.33 setFilter()	78
	7.26.3.34 setHardwareBufferSize()	79
	7.26.3.35 setInputDelay()	79
	7.26.3.36 setLED()	79
	7.26.3.37 setNormalization()	79
	7.26.3.38 setTestSignal()	80
	7.26.3.39 setTestSignalDivider()	80
	7.26.3.40 setTriggerLevel()	80
	7.26.3.41 sync()	81
	7.26.3.42 unregisterChannel()	81
7.27	TimetaggerFPGA Class Reference	81
7.28	TimeTaggerModel Class Reference	82
7.29	TimeTagStream Class Reference	82
	7.29.1 Detailed Description	83
	7.29.2 Constructor & Destructor Documentation	83
	7.29.2.1 ~TimeTagStream()	83
	7.29.3 Member Function Documentation	83
	7.29.3.1 clear_impl()	83
	7.29.3.2 getCounts()	84
	7.29.3.3 next_impl()	84
7.30	TimeTagStreamBuffer Class Reference	84
Index		87

# TimeTagger

backend for TimeTagger, an OpalKelly based single photon counting library

Author

Markus Wick wickmarkus@web.de
Helmut Fedderhelmut.fedder@gmail.com

TimeTagger provides an easy to use and cost effective hardware solution for time-resolved single photon counting applications.

This document describes the C++ native interface to the TimeTagger device.

2 TimeTagger

# **Deprecated List**

Member TimeTagger::getChannels ()=0

Use getChannelList instead.

Member TimeTagger::getFilter ()=0

 $use\ getConditional Filter*$ 

Member TimeTagger::setFilter (bool state)=0

 $use \ set Conditional Filter$ 

Deprecated List

# **Module Index**

# 3.1 Modules

Here is a list of all modules:

TimeTagger backend		 		 													 				11
base iterators				 													 				12

6 Module Index

# **Hierarchical Index**

# 4.1 Class Hierarchy

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

_lterator	15
Coincidences	. 23
Coincidence	. 21
Combiner	. 25
Correlation	. 27
CountBetweenMarkers	. 30
Counter	. 34
Countrate	. 37
DelayedChannel	. 40
Dump	. 42
FrequencyMultiplier	. 45
GatedChannel	. 47
HistogramLogBins	. 51
Iterator	. 53
Scope	
StartStop	. 58
TimeDifferences	. 64
TimeTagStream	. 82
Array< T >	21
Event	44
Flim	44
Histogram	49
LED	55
PRBS	56
SynchronizedMeasurments	61
Tag	63
TimeTagger	68
TimetaggerFPGA	81
TimeTaggerModel	82
TimeTagStreamBuffer	84

8 Hierarchical Index

# **Class Index**

# 5.1 Class List

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

_lterator	
Base class for all iterators	15
Array < T >	21
Coincidence	
Coincidence monitor for one or more channel groups	21
Coincidences	
Coincidence monitor for one or more channel groups	23
Combiner	
Combine some channels in a virtual channel which has a tick for each tick in the input channels	25
Correlation	
Cross-correlation between two channels	27
Simple counter where external marker signals determine the bins	30
Counter	
Simple counter on one or more channels	34
Countrate	
Count rate on one or more channels	37
DelayedChannel	
Simple delayed queue	40
Dump	
Dump all time tags to a file	42
Event Flim	44
Fluorescence lifetime imaging	44
FrequencyMultiplier	
The signal of an input channel is scaled up to a higher frequency according to the multiplier	
passed as a parameter	45
GatedChannel	
An input channel is gated by a gate channel	47
Histogram	
Accumulate time differences into a histogram	49
HistogramLogBins	
Accumulate time differences into a histogram with logarithmic increasing bin sizes	51
Iterator	
Simple event queue	50

10 Class Index

ED	55
PRBS	
Scope	
StartStop	
Simple start-stop measurement	58
SynchronizedMeasurments	
Start, stop and clear several measurements synchronized	61
ag	
Single event on a channel	63
TimeDifferences	
Accumulates the time differences between clicks on two channels in one or more histograms .	64
imeTagger	
Backend for the TimeTagger	68
imetaggerFPGA	81
imeTaggerModel	
TimeTagStream	
Access the time tag stream	82
imeTagStreamBuffer	

# **Module Documentation**

# 6.1 TimeTagger backend

the timetagger kernel

### **Classes**

- class TimeTagger backend for the TimeTagger.
- struct Tag

a single event on a channel

· class \_lterator

Base class for all iterators.

# 6.1.1 Detailed Description

the timetagger kernel

12 Module Documentation

### 6.2 base iterators

base iterators for photon counting applications

#### **Classes**

· class Combiner

Combine some channels in a virtual channel which has a tick for each tick in the input channels.

class CountBetweenMarkers

a simple counter where external marker signals determine the bins

· class Counter

a simple counter on one or more channels

· class Coincidences

a coincidence monitor for one or more channel groups

class Coincidence

a coincidence monitor for one or more channel groups

· class Countrate

count rate on one or more channels

class DelayedChannel

a simple delayed queue

class GatedChannel

An input channel is gated by a gate channel.

class FrequencyMultiplier

The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.

· class Iterator

a simple event queue

· class StartStop

simple start-stop measurement

class TimeDifferences

Accumulates the time differences between clicks on two channels in one or more histograms.

· class Histogram

Accumulate time differences into a histogram.

• class HistogramLogBins

Accumulate time differences into a histogram with logarithmic increasing bin sizes.

· class Flim

Fluorescence lifetime imaging.

· class Correlation

cross-correlation between two channels

class Scope

### **Functions**

TimeTagStream::TimeTagStream (TimeTagger \*tagger, int n\_max\_events, const std::vector< channel\_t > &channels=std::vector< channel\_t >())

constructor of a TimeTagStream thread

Dump::Dump (TimeTagger \*tagger, std::string filename, size\_t max\_tags, const std::vector< channel\_t > &channels=std::vector< channel\_t >())

constructor of a Dump thread

6.2 base iterators

# 6.2.1 Detailed Description

base iterators for photon counting applications

# 6.2.2 Function Documentation

# 6.2.2.1 Dump()

### constructor of a Dump thread

#### **Parameters**

tagger	reference to a TimeTagger
filename	name of the file to dump to
max_tags	stop after this number of tags has been dumped
channels	channels which are dumped to the file (when empty or not passed all active channels are dumped)

# 6.2.2.2 TimeTagStream()

constructor of a TimeTagStream thread

Gives access to the time tag stream

### **Parameters**

tagger	reference to a TimeTagger
n_max_events	maximum number of tags stored
channels	channels which are dumped to the file (when empty or not passed all active channels are dumped)

14 Module Documentation

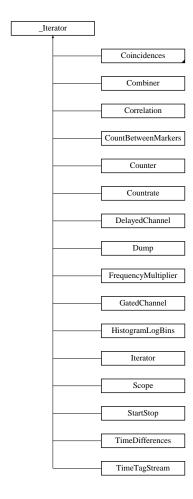
# **Class Documentation**

# 7.1 \_Iterator Class Reference

Base class for all iterators.

#include <TimeTagger.h>

Inheritance diagram for \_lterator:



16 **Class Documentation** 

virtual bool next\_impl (std::vector< Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_←

#### **Public Member Functions**

```
    virtual ~_Iterator ()

      destructor

    virtual void start ()

      start the iterator

    void startFor (timestamp_t capture_duration, bool clear=true)

      start the iterator, and stops it after the capture_duration
· virtual void stop ()
      stop the iterator
• void clear ()
      clear Iterator state.

    bool isRunning ()

      query the Iterator state.

    timestamp_t getCaptureDuration ()

      query the evaluation time
```

### **Protected Member Functions**

```
• _lterator (TimeTagger *tagger)
     standard constructor

    void registerChannel (channel_t channel)

     register a channel

    void unregisterChannel (channel_t channel)

     unregister a channel
· channel_t getNewVirtualChannel ()
     allocate a new virtual output channel for this iterator

    virtual void clear_impl ()

     clear Iterator state.
• void lock ()
     aquire update lock
· void unlock ()
```

release update lock

update iterator state

#### **Protected Attributes**

time)=0

```
• std::set< channel_t > channels_registered
     list of channels used by the iterator

    bool running
```

- TimeTagger \* tagger
- · timestamp\_t capture\_duration

running state of the iterator

# **Friends**

class \_TimeTagger

# 7.1.1 Detailed Description

Base class for all iterators.

### 7.1.2 Constructor & Destructor Documentation

```
7.1.2.1 _lterator()
```

standard constructor

will register with the TimeTagger backend.

```
7.1.2.2 \sim_lterator()
```

```
virtual _Iterator::~_Iterator ( ) [virtual]
```

destructor

will stop and unregister prior finalization.

### 7.1.3 Member Function Documentation

```
7.1.3.1 clear()
```

```
void _Iterator::clear ( )
```

clear Iterator state.

#### 7.1.3.2 clear\_impl()

```
virtual void _Iterator::clear_impl ( ) [inline], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented in Scope, Correlation, HistogramLogBins, TimeDifferences, StartStop, Dump, TimeTagStream, Iterator, Countrate, Counter, CountBetweenMarkers, and Combiner.

18 Class Documentation

#### 7.1.3.3 getCaptureDuration()

```
timestamp_t _Iterator::getCaptureDuration ( )
```

query the evaluation time

Query the total capture duration since the last call to . This might have a wrong amount of time if there were some overflows within this range.

Returns

capture duration of the data

#### 7.1.3.4 getNewVirtualChannel()

```
channel_t _Iterator::getNewVirtualChannel ( ) [protected]
```

allocate a new virtual output channel for this iterator

#### 7.1.3.5 isRunning()

```
bool _Iterator::isRunning ( )
```

query the Iterator state.

Fetches if this iterator is running.

### 7.1.3.6 lock()

```
void _Iterator::lock ( ) [protected]
```

aquire update lock

All mutable operations on a iterator are guarded with an update mutex. Implementers are adviced to lock() an iterator, whenever internal state is queried or changed.

### 7.1.3.7 next\_impl()

```
virtual bool _Iterator::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [protected], [pure virtual]
```

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implemented in Scope, Correlation, HistogramLogBins, TimeDifferences, StartStop, Dump, TimeTagStream, Iterator, FrequencyMultiplier, GatedChannel, DelayedChannel, Countrate, Coincidences, Counter, CountBetweenMarkers, and Combiner.

#### 7.1.3.8 registerChannel()

register a channel

Only channels registered by any iterator attached to a backend are delivered over the usb.

# **Parameters**

```
channel the channel
```

#### 7.1.3.9 start()

```
virtual void _Iterator::start ( ) [virtual]
```

start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented in Scope, TimeDifferences, StartStop, DelayedChannel, Countrate, and Counter.

#### 7.1.3.10 startFor()

start the iterator, and stops it after the capture\_duration

20 Class Documentation

#### **Parameters**

capture_duration	capture duration until the meassurement is stopped
clear	resets the data aquired

When the startFor is called before the previous measurement has ended and the clear parameter is set to false, then the passed capture\_duration will be added on top to the current max\_capture\_duration

```
7.1.3.11 stop()
virtual void _Iterator::stop ( ) [virtual]
stop the iterator
```

The iterator is put into the STOPPED state, but will still be registered with the backend.

```
7.1.3.12 unlock()
```

```
void _Iterator::unlock ( ) [protected]
```

release update lock

see lock()

# 7.1.3.13 unregisterChannel()

unregister a channel

#### **Parameters**

channel	the channel

### 7.1.4 Member Data Documentation

# 7.1.4.1 channels\_registered

```
std::set<channel_t> _Iterator::channels_registered [protected]
```

list of channels used by the iterator

#### 7.1.4.2 running

```
bool _Iterator::running [protected]
```

running state of the iterator

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

· TimeTagger.h

# 7.2 Array < T > Class Template Reference

### **Public Member Functions**

- Array (std::vector< int > dims)
- int getDims ()
- int getSize (int dim)
- size\_t getLength ()
- void get (T \*fixed\_output)
- T \* get ()
- Array (std::vector< int > dims)
- int getDims ()
- int getSize (int dim)
- size\_t getLength ()
- void get (T \*fixed\_output)
- T \* get ()

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

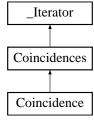
- · CSharpWrapper.h
- · JavaWrapper.h

# 7.3 Coincidence Class Reference

a coincidence monitor for one or more channel groups

```
#include <Iterators.h>
```

Inheritance diagram for Coincidence:



22 Class Documentation

#### **Public Member Functions**

• Coincidence (TimeTagger \*tagger, std::vector< channel\_t > channels, timestamp\_t coincidence 
Window=1000)

construct a coincidence

channel\_t getChannel ()

virtual channel which contains the coincidences

#### **Additional Inherited Members**

#### 7.3.1 Detailed Description

a coincidence monitor for one or more channel groups

Monitor coincidences for a given channel groups passed by the constructor. A coincidence is event is detected when all slected channels have a click within the given coincidenceWindow [ps] The coincidence will create a virtual events on a virtual channel with the channel number provided by getChannel(). For multiple coincidence channel combinations use the class Coincidences which outperformes multiple instances of Conincdence.

#### 7.3.2 Constructor & Destructor Documentation

# 7.3.2.1 Coincidence()

### construct a coincidence

#### **Parameters**

tagger	reference to a TimeTagger
channels	vector of channels to match
window	max distance between all clicks for a coincidence [ps]

# 7.3.3 Member Function Documentation

#### 7.3.3.1 getChannel()

```
channel_t Coincidence::getChannel ( ) [inline]
```

virtual channel which contains the coincidences

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

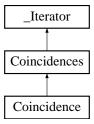
· Iterators.h

### 7.4 Coincidences Class Reference

a coincidence monitor for one or more channel groups

```
#include <Iterators.h>
```

Inheritance diagram for Coincidences:



#### **Public Member Functions**

Coincidences (TimeTagger \*tagger, std::vector< std::vector< channel\_t >> coincidenceGroups, timestamp\_t coincidenceWindow)

construct a Coincidences

std::vector< channel\_t > getChannels ()

fetches the block of virtual channels for those coincidence groups

void setCoincidenceWindow (timestamp\_t coincidenceWindow)

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

#### **Additional Inherited Members**

# 7.4.1 Detailed Description

a coincidence monitor for one or more channel groups

Monitor coincidences for given coincidence groups passed by the constructor. A coincidence is hereby defined as for a given coincidence group a) the incoming is part of this group b) at least tag arrived within the coincidence  $\leftarrow$  Window [ps] for all other channels of this coincidence group Each coincidence will create a virtual event. The block of event IDs for those coincidence group can be fetched.

24 Class Documentation

# 7.4.2 Constructor & Destructor Documentation

### 7.4.2.1 Coincidences()

#### construct a Coincidences

#### **Parameters**

tagger	reference to a TimeTagger
coincidenceGroups	a vector of channels defining the coincidences
coincidenceWindow	the size of the coincidence window in picoseconds

### 7.4.3 Member Function Documentation

### 7.4.3.1 getChannels()

```
std::vector< channel_t > Coincidences::getChannels ( )
```

fetches the block of virtual channels for those coincidence groups

#### 7.4.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events	
start_time	earliest event in the block	
end_time	start_time of the next block, not including in this block	

#### Returns

if the content of this block was modified

Implements \_lterator.

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

- · Iterators.h
- · Coincidence.cpp

# 7.5 Combiner Class Reference

Combine some channels in a virtual channel which has a tick for each tick in the input channels.

```
#include <Iterators.h>
```

Inheritance diagram for Combiner:



# **Public Member Functions**

- Combiner (TimeTagger \*tagger, std::vector< channel\_t > channels)
  - construct a combiner
- GET\_DATA\_1D (getData, long long, array\_out,)
  - get sum of counts
- channel t getChannel ()

the new virtual channel

# **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

void clear\_impl () override

clear Iterator state.

# **Additional Inherited Members**

# 7.5.1 Detailed Description

Combine some channels in a virtual channel which has a tick for each tick in the input channels.

This iterator can be used to get aggregation channels, eg if you want to monitor the countrate of the sum of two channels.

# 7.5.2 Constructor & Destructor Documentation

# 7.5.2.1 Combiner()

#### construct a combiner

#### **Parameters**

tagger	reference to a TimeTagger
channels	vector of channels to combine

# 7.5.3 Member Function Documentation

# 7.5.3.1 clear\_impl()

```
void Combiner::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

# 7.5.3.2 GET\_DATA\_1D()

# get sum of counts

For reference, this iterators sums up how much ticks are generated because of which input channel. So this functions returns an array with one value per input channel.

### 7.5.3.3 getChannel()

```
channel_t Combiner::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

#### 7.5.3.4 next\_impl()

```
bool Combiner::next_impl (
          std::vector< Tag > & incoming_tags,
          timestamp_t begin_time,
          timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

## Returns

if the content of this block was modified

Implements \_lterator.

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

- · Iterators.h
- · Combiner.cpp

# 7.6 Correlation Class Reference

cross-correlation between two channels

```
#include <Iterators.h>
```

Inheritance diagram for Correlation:



#### **Public Member Functions**

 Correlation (TimeTagger \*tagger, channel\_t channel\_t channel\_t channel\_t channel\_2=CHANNEL\_UNUSED, timestamp\_t binwidth=1000, int n\_bins=1000)

constructor of a correlation measurement

GET\_DATA\_1D (getData, int, array\_out,)

get result data

• GET\_DATA\_1D (getDataNormalized, double, array\_out,)

get result data - normalized such that a perfectly uncorrelated signals would be flat at a hight of one

GET\_DATA\_1D (getIndex, timestamp\_t, array\_out,)

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

· void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

#### 7.6.1 Detailed Description

cross-correlation between two channels

Accumulates time differences between clicks on two channels into a histogram, where all ticks are considered both as start and stop clicks and both positive and negative time differences are considered. The histogram is determined by the number of bins and the binwidth, which are used both for the positive and the negative histogram range (i.e., length of the histogram is 2\*n bins+1).

#### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 Correlation()

# constructor of a correlation measurement

#### **Parameters**

tagger	reference to a TimeTagger	
channel←	first click channel	
_1		
channel⊷	second click channel	Generated by Doxygen
_2		
binwidth	width of one histogram bin in ps	
n bins	the number of bins in the resulting histogram is 2*n bins+1	

# 7.6.3 Member Function Documentation

# 7.6.3.1 clear\_impl()

```
void Correlation::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

# **7.6.3.2 GET\_DATA\_1D()** [1/2]

get result data

### **Parameters**

ARGOUTVIEWM_ARRAY2	pointer receiving pointer to data
DIM1	pointer receiving first dimension
DIM2	pointer receiving second dimension

# **7.6.3.3 GET\_DATA\_1D()** [2/2]

get result data - normalized such that a perfectly uncorrelated signals would be flat at a hight of one

#### **Parameters**

ARGOUTVIEWM_ARRAY2	pointer receiving pointer to data	
DIM1	pointer receiving first dimension	
DIM2	pointer receiving second dimension	

### 7.6.3.4 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

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

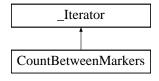
- · Iterators.h
- TimeDifferences.cpp

# 7.7 CountBetweenMarkers Class Reference

a simple counter where external marker signals determine the bins

```
#include <Iterators.h>
```

Inheritance diagram for CountBetweenMarkers:



#### **Public Member Functions**

```
• CountBetweenMarkers (TimeTagger *tagger, channel_t click_channel, channel_t begin_channel, channel_t end_channel=CHANNEL_UNUSED, int n_values=1000)
```

```
constructor of CountBetweenMarkers
```

• bool ready ()

tbd

• GET\_DATA\_1D (getData, int, array\_out,)

tbd

• GET\_DATA\_1D (getBinWidths, timestamp\_t, array\_out,)

fetches the widths of each bins

• GET\_DATA\_1D (getIndex, timestamp\_t, array\_out,)

fetches the starting time of each bin

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

· void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

# 7.7.1 Detailed Description

a simple counter where external marker signals determine the bins

Counter with external signals that trigger beginning and end of each counter accumulation. This can be used to implement counting triggered by a pixel clock and gated counting. The thread waits for the first time tag on the 'begin\_channel', then begins counting time tags on the 'click\_channel'. It ends counting when a tag on the 'end\_channel' is detected.

# 7.7.2 Constructor & Destructor Documentation

# 7.7.2.1 CountBetweenMarkers()

constructor of CountBetweenMarkers

#### **Parameters**

tagger	reference to a TimeTagger
click_channel	channel that increases the count
begin_channel	channel that triggers beginning of counting and stepping to the next value
end_channel	channel that triggers end of counting
n_values	the number of counter values to be stored

# 7.7.3 Member Function Documentation

# 7.7.3.1 clear\_impl()

```
void CountBetweenMarkers::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_Iterator.

# **7.7.3.2 GET\_DATA\_1D()** [1/3]

```
CountBetweenMarkers::GET_DATA_1D (
          getData ,
          int ,
          array_out )
```

tbd

# **Parameters**

ARGOUTVIEWM_ARRAY1	
DIM1	

# **7.7.3.3 GET\_DATA\_1D()** [2/3]

fetches the widths of each bins

fetches the starting time of each bin

# 7.7.3.5 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

### Returns

if the content of this block was modified

Implements \_Iterator.

## 7.7.3.6 ready()

```
bool CountBetweenMarkers::ready ( )
```

tbd

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

- · Iterators.h
- · CountBetweenMarkers.cpp

# 7.8 Counter Class Reference

a simple counter on one or more channels

```
#include <Iterators.h>
```

Inheritance diagram for Counter:



#### **Public Member Functions**

 Counter (TimeTagger \*tagger, std::vector< channel\_t > channels, timestamp\_t binwidth=1000000000, int n\_values=1)

construct a counter

• void start () override

start the iterator

GET\_DATA\_2D (getData, int, array\_out,)

get counts

GET\_DATA\_1D (getIndex, timestamp\_t, array\_out,)

# **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

• void clear\_impl () override

clear Iterator state.

# **Additional Inherited Members**

# 7.8.1 Detailed Description

a simple counter on one or more channels

Counter with fixed binwidth and circular buffer output. This class is suitable to generate a time trace of the count rate on one or more channels. The thread repeatedly counts clicks on a single channel over a given time interval and stores the results in a two-dimensional array. The array is treated as a circular buffer. I.e., once the array is full, each new value shifts all previous values one element to the left.

# 7.8.2 Constructor & Destructor Documentation

# 7.8.2.1 Counter()

construct a counter

#### **Parameters**

tagger	reference to a TimeTagger	
channels	channels to count on	
binwidth	th counts are accumulated for binwidth picoseconds	
n_values	number of counter values stored (for each channel)	

# 7.8.3 Member Function Documentation

```
7.8.3.1 clear_impl()
```

```
void Counter::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

# 7.8.3.2 GET\_DATA\_2D()

get counts

the counts are copied to a newly allocated allocated memory, an the pointer to this location is returned.

# 7.8.3.3 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

# Returns

if the content of this block was modified

Implements \_Iterator.

#### 7.8.3.4 start()

```
void Counter::start ( ) [override], [virtual]
```

start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from <u>\_lterator</u>.

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

- · Iterators.h
- · Counter.cpp

# 7.9 Countrate Class Reference

count rate on one or more channels

```
#include <Iterators.h>
```

Inheritance diagram for Countrate:



## **Public Member Functions**

- Countrate (TimeTagger \*tagger, std::vector < channel\_t > channels)
   constructor of Countrate
- void start () override

start the iterator

• GET\_DATA\_1D (getData, double, array\_out,)

get the count rates

# **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

· void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

## 7.9.1 Detailed Description

count rate on one or more channels

Measures the average count rate on one or more channels. Specifically, it counts incoming clicks and determines the time between the initial click and the latest click. The number of clicks divided by the time corresponds to the average countrate since the initial click.

# 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 Countrate()

# constructor of Countrate

#### **Parameters**

tagger	reference to a TimeTagger
channels	the channels to count on

#### 7.9.3 Member Function Documentation

#### 7.9.3.1 clear\_impl()

```
void Countrate::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from Iterator.

### 7.9.3.2 GET\_DATA\_1D()

### get the count rates

the count rates are copied to a newly allocated allocated memory, an the pointer to this location is returned.

#### 7.9.3.3 next\_impl()

#### update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_lterator.

#### 7.9.3.4 start()

```
void Countrate::start ( ) [override], [virtual]
```

# start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from \_lterator.

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

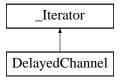
- · Iterators.h
- Counter.cpp

# 7.10 DelayedChannel Class Reference

a simple delayed queue

```
#include <Iterators.h>
```

Inheritance diagram for DelayedChannel:



#### **Public Member Functions**

• DelayedChannel (TimeTagger \*tagger, channel\_t input\_channel, unsigned long long delay)

constructor of a DelayedChannel

• void start () override

start the iterator

channel\_t getChannel ()

the new virtual channel

• void setDelay (unsigned long long delay)

set the delay time delay for the cloned tags in the virtual channel  $0 \le t \le 2^{63}$ 

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

### **Additional Inherited Members**

# 7.10.1 Detailed Description

a simple delayed queue

A simple first-in first-out queue of delayed event timestamps.

# 7.10.2 Constructor & Destructor Documentation

# 7.10.2.1 DelayedChannel()

constructor of a DelayedChannel

#### **Parameters**

tagger	reference to a TimeTagger
input_channel	channel which is delayed
delay	amount of time to delay, must be positive

### 7.10.3 Member Function Documentation

#### 7.10.3.1 getChannel()

```
channel_t DelayedChannel::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

# 7.10.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each  ${\sf Tag}$  on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

### 7.10.3.3 setDelay()

```
void DelayedChannel::setDelay (
          unsigned long long delay )
```

set the delay time delay for the cloned tags in the virtual channel 0 <=  $t < 2^{63}$ 

Note: When the delay is the same or greater than the previous value all incoming tags will be visible at virtual channel. By applying a shorter delay time, the tags stored in the local buffer will be flushed and won't be visible in the virtual channel.

#### 7.10.3.4 start()

```
void DelayedChannel::start ( ) [override], [virtual]
```

start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from <u>\_lterator</u>.

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

- · Iterators.h
- · Combiner.cpp

# 7.11 Dump Class Reference

dump all time tags to a file

```
#include <Iterators.h>
```

Inheritance diagram for Dump:



### **Public Member Functions**

Dump (TimeTagger \*tagger, std::string filename, size\_t max\_tags, const std::vector< channel\_t > &channels=std::vector< channel\_t >())

```
constructor of a Dump thread
```

• ~Dump ()

tbd

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

· void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

# 7.11.1 Detailed Description

dump all time tags to a file

#### 7.11.2 Constructor & Destructor Documentation

```
7.11.2.1 \sim Dump()

Dump::\simDump ( )

tbd
```

### 7.11.3 Member Function Documentation

```
7.11.3.1 clear_impl()
void Dump::clear_impl ( ) [override], [protected], [virtual]
clear Iterator state.
```

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from <u>\_lterator</u>.

#### 7.11.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### **Returns**

if the content of this block was modified

Implements \_Iterator.

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

- · Iterators.h
- · Iterator.cpp

# 7.12 Event Struct Reference

# **Public Attributes**

- · timestamp t time
- · State state

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

· Iterators.h

# 7.13 Flim Class Reference

Fluorescence lifetime imaging.

#include <Iterators.h>

### **Public Member Functions**

• Flim (TimeTagger \*tagger, channel\_t click\_channel, channel\_t start\_channel, channel\_t next\_channel, timestamp\_t binwidth=1000, int n\_bins=1000, int n\_pixels=1)

constructor of a FLIM measurement

- · void start ()
- void startFor (timestamp\_t capture\_duration, bool clear=true)
- void stop ()
- void clear ()
- timestamp\_t getCaptureDuration ()
- **GET\_DATA\_2D** (getData, int, array\_out,)
- **GET\_DATA\_1D** (getIndex, timestamp\_t, array\_out,)

# 7.13.1 Detailed Description

Fluorescence lifetime imaging.

Successively acquires n histograms (one for each pixel in the image), where each histogram is determined by the number of bins and the binwidth. Clicks that fall outside the histogram range are ignored.

Fluorescence-lifetime imaging microscopy or FLIM is an imaging technique for producing an image based on the differences in the exponential decay rate of the fluorescence from a fluorescent sample.

Fluorescence lifetimes can be determined in the time domain by using a pulsed source. When a population of fluorophores is excited by an ultrashort or delta pulse of light, the time-resolved fluorescence will decay exponentially.

#### 7.13.2 Constructor & Destructor Documentation

# 7.13.2.1 Flim()

#### constructor of a FLIM measurement

#### **Parameters**

tagger	reference to a TimeTagger
click_channel	channel that increments the count in a bin
start_channel	channel that sets start times relative to which clicks on the click channel are measured
next_channel	channel that increments the pixel
binwidth	width of one histogram bin in ps
n_bins	number of bins in each histogram
n_pixels	number of pixels

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

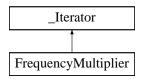
- · Iterators.h
- TimeDifferences.cpp

# 7.14 FrequencyMultiplier Class Reference

The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.

```
#include <Iterators.h>
```

Inheritance diagram for FrequencyMultiplier:



#### **Public Member Functions**

- FrequencyMultiplier (TimeTagger \*tagger, channel\_t input\_channel, int multiplier)
   constructor of a FrequencyMultiplier
- channel\_t getChannel ()
- int getMultiplier ()

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

#### **Additional Inherited Members**

# 7.14.1 Detailed Description

The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.

The FrequencyMultiplier inserts copies the original input events from the input\_channel and adds additional events to match the upscaling factor. The algorithm used assumes a constant frequency and calculates out of the last two incoming events linearly the intermediate timestamps to match the upscaled frequency given by the multiplier parameter.

The FrequencyMultiplier can be used to restore the actual frequency applied to an input\_channel which was reduces via the EventDivider to lower the effective data rate. For example a 80 MHz laser sync signal can be scaled down via setEventDivider(..., 80) to 1 MHz (hardware side) and an 80 MHz signal can be restored via FrequencyMultiplier(..., 80) on the software side with some loss in precision. The FrequencyMultiplier is an alternative way to reduce the data rate in comparison to the EventFilter, which has a higher precision but can be more difficult to use.

### 7.14.2 Constructor & Destructor Documentation

# 7.14.2.1 FrequencyMultiplier()

constructor of a FrequencyMultiplier

#### **Parameters**

tagger	reference to a TimeTagger
input_channel	channel on which the upscaling of the frequency is based on
multiplier	frequency upscaling factor

#### 7.14.3 Member Function Documentation

# 7.14.3.1 next\_impl()

```
bool FrequencyMultiplier::next_impl (
          std::vector< Tag > & incoming_tags,
          timestamp_t begin_time,
          timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

# Parameters

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

# Returns

if the content of this block was modified

Implements \_Iterator.

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

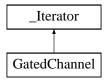
- Iterators.h
- · Combiner.cpp

# 7.15 GatedChannel Class Reference

An input channel is gated by a gate channel.

```
#include <Iterators.h>
```

Inheritance diagram for GatedChannel:



# **Public Member Functions**

• GatedChannel (TimeTagger \*tagger, channel\_t input\_channel, channel\_t gate\_start\_channel, channel\_ 

t gate\_stop\_channel)

constructor of a GatedChannel

channel\_t getChannel ()

the new virtual channel

#### **Protected Member Functions**

bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) over-ride

update iterator state

# **Additional Inherited Members**

# 7.15.1 Detailed Description

An input channel is gated by a gate channel.

Note: The gate is edge sensitive and not level sensitive. That means that the gate will transfer data only when an appropriate level change is detected on the gate\_start\_channel.

# 7.15.2 Constructor & Destructor Documentation

### 7.15.2.1 GatedChannel()

# constructor of a GatedChannel

#### **Parameters**

tagger	reference to a TimeTagger
input_channel	channel which is gated
_gate_start_channel	channel on which a signal detected will start the transmission of the input_channel through the gate  Generated by Doxygen
gate_stop_channel	channel on which a signal detected will stop the transmission of the input_channel through the gate

# 7.15.3 Member Function Documentation

# 7.15.3.1 getChannel()

```
channel_t GatedChannel::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

#### 7.15.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

# **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

# Returns

if the content of this block was modified

Implements \_Iterator.

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

- · Iterators.h
- · Combiner.cpp

# 7.16 Histogram Class Reference

Accumulate time differences into a histogram.

```
#include <Iterators.h>
```

#### **Public Member Functions**

• Histogram (TimeTagger \*tagger, channel\_t click\_channel, channel\_t start\_channel=CHANNEL\_UNUSED, timestamp\_t binwidth=1000, int n\_bins=1000)

constructor of a Histogram measurement

- void start ()
- void startFor (timestamp\_t capture\_duration, bool clear=true)
- · void stop ()
- · void clear ()
- timestamp\_t getCaptureDuration ()
- bool isRunning ()
- GET\_DATA\_1D (getData, int, array\_out,)
- GET\_DATA\_1D (getIndex, timestamp\_t, array\_out,)

# 7.16.1 Detailed Description

Accumulate time differences into a histogram.

This is a simple multiple start, multiple stop measurement. This is a special case of the more general 'TimeDifferences' measurement. Specifically, the thread waits for clicks on a first channel, the 'start channel', then measures the time difference between the last start click and all subsequent clicks on a second channel, the 'click channel', and stores them in a histogram. The histogram range and resolution is specified by the number of bins and the binwidth. Clicks that fall outside the histogram range are ignored. Data accumulation is performed independently for all start clicks. This type of measurement is frequently referred to as 'multiple start, multiple stop' measurement and corresponds to a full auto- or cross-correlation measurement.

#### 7.16.2 Constructor & Destructor Documentation

# 7.16.2.1 Histogram()

## constructor of a Histogram measurement

### **Parameters**

tagger	reference to a TimeTagger
click_channel	channel that increments the count in a bin
start_channel	channel that sets start times relative to which clicks on the click channel are measured
binwidth	width of one histogram bin in ps
n_bins	number of bins in the histogram

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

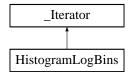
- · Iterators.h
- · TimeDifferences.cpp

# 7.17 HistogramLogBins Class Reference

Accumulate time differences into a histogram with logarithmic increasing bin sizes.

```
#include <Iterators.h>
```

Inheritance diagram for HistogramLogBins:



#### **Public Member Functions**

• HistogramLogBins (TimeTagger \*tagger, channel\_t click\_channel, channel\_t start\_channel, double exp\_start, double exp\_stop, int n\_bins)

constructor of a HistogramLogBins measurement

- **GET\_DATA\_1D** (getData, unsigned long long, array\_out,)
- **GET\_DATA\_1D** (getDataNormalized, double, array\_out,)
- **GET\_DATA\_1D** (getBinEdges, timestamp\_t, array\_out,)

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

void clear\_impl () override

clear Iterator state.

# **Additional Inherited Members**

# 7.17.1 Detailed Description

Accumulate time differences into a histogram with logarithmic increasing bin sizes.

This is a multiple start, multiple stop measurement, and works the very same way as the histogram measurement but with logarithmic increasing bin widths. After initializing the measurement (or after an overflow) no data is accumulated in the histogram until the full histogram duration has passed to ensure a balanced count accumulation over the full histogram.

### 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 HistogramLogBins()

constructor of a HistogramLogBins measurement

#### **Parameters**

tagger	reference to a TimeTagger
click_channel	channel that increments the count in a bin
start_channel	channel that sets start times relative to which clicks on the click channel are measured
exp_start	exponent for the lowest time diffrences in the histogram: $10^{\text{exp\_start}}$ s, lowest exp_start: -12 => 1ps
exp_stop	exponent for the highest time diffrences in the histogram: 10^exp_stop s
n_bins	total number of bins in the histogram

# 7.17.3 Member Function Documentation

# 7.17.3.1 clear\_impl()

```
void HistogramLogBins::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_Iterator.

### 7.17.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

# Returns

if the content of this block was modified

Implements \_Iterator.

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

- · Iterators.h
- · TimeDifferences.cpp

# 7.18 Iterator Class Reference

a simple event queue

```
#include <Iterators.h>
```

Inheritance diagram for Iterator:



# **Public Member Functions**

- Iterator (TimeTagger \*tagger, channel\_t channel)
- standard constructortimestamp\_t next ()

get next timestamp

• int size ()

get queue size

#### **Protected Member Functions**

bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) over-ride

update iterator state

• void clear\_impl () override

clear Iterator state.

# **Additional Inherited Members**

# 7.18.1 Detailed Description

a simple event queue

A simple Iterator, just keeping a first-in first-out queue of event timestamps.

# 7.18.2 Constructor & Destructor Documentation

# 7.18.2.1 Iterator()

standard constructor

#### **Parameters**

tagger	the backend
channel	the channel to get events from

#### 7.18.3 Member Function Documentation

```
7.18.3.1 clear_impl()
```

```
void Iterator::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

```
7.18.3.2 next()
```

```
timestamp_t Iterator::next ( )
```

get next timestamp

get the next timestamp from the queue.

7.19 LED Union Reference 55

### 7.18.3.3 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

## 7.18.3.4 size()

```
int Iterator::size ( )
```

get queue size

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

- Iterators.h
- · Iterator.cpp

# 7.19 LED Union Reference

# **Public Attributes**

•

```
struct {
    uint32_t statusR: 1
    uint32_t statusB: 1
    uint32_t statusG: 1
    uint32_t powerR: 1
    uint32_t powerB: 1
    uint32_t powerG: 1
    uint32_t clockR: 1
    uint32_t clockB: 1
    uint32_t clockG: 1
};
```

• uint32\_t ledStatus

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

TimeTaggerTest.cpp

# 7.20 PRBS Class Reference

**Public Member Functions** 

• uint32\_t get ()

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

• PRBS.h

# 7.21 Scope Class Reference

Inheritance diagram for Scope:



# **Public Member Functions**

• Scope (TimeTagger \*tagger, std::vector< channel\_t > event\_channels, channel\_t trigger\_channel, timestamp\_t window\_size=1000000000, int n\_traces=1, int n\_max\_events=1000)

constructor of a Scope measurement

- bool ready ()
- int triggered ()
- · void start () override

start the iterator

- std::vector< std::vector< Event >> getData ()
- timestamp\_t getWindowSize ()

# **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

· void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

# 7.21.1 Constructor & Destructor Documentation

#### 7.21.1.1 Scope()

# constructor of a Scope measurement

## **Parameters**

tagger	reference to a TimeTagger
event_channels	channels which are captured
trigger_channel	channel that starts a new trace
window_size	window time of each trace
n_traces	amount of traces (n_traces < 1, automatic retrigger)
n_max_events	maximum number of tags in each trace

## 7.21.2 Member Function Documentation

```
7.21.2.1 clear_impl()
```

```
void Scope::clear_impl () [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

### 7.21.2.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

# 7.21.2.3 start()

```
void Scope::start ( ) [override], [virtual]
```

start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from \_lterator.

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

- · Iterators.h
- · Scope.cpp

# 7.22 StartStop Class Reference

simple start-stop measurement

```
#include <Iterators.h>
```

Inheritance diagram for StartStop:



#### **Public Member Functions**

• StartStop (TimeTagger \*tagger, channel\_t click\_channel, channel\_t start\_channel=CHANNEL\_UNUSED, timestamp\_t binwidth=1000)

```
constructor of StartStop
```

· void start () override

start the iterator

• GET\_DATA\_2D (getData, timestamp t, array out,)

## **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

void clear\_impl () override

clear Iterator state.

# **Additional Inherited Members**

# 7.22.1 Detailed Description

simple start-stop measurement

This class performs a start-stop measurement between two channels and stores the time differences in a histogram. The histogram resolution is specified beforehand (binwidth) but the histogram range is unlimited. It is adapted to the largest time difference that was detected. Thus all pairs of subsequent clicks are registered.

Be aware, on long-running measurements this may considerably slow down system performance and even crash the system entirely when attached to an unsuitable signal source.

#### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 StartStop()

#### constructor of StartStop

### **Parameters**

tagger	reference to a TimeTagger
click_channel	channel for stop clicks
start_channel	channel for start clicks
binwidth Generated by Doxyger	width of one histogram bin in ps

# 7.22.3 Member Function Documentation

# 7.22.3.1 clear\_impl()

```
void StartStop::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_lterator.

# 7.22.3.2 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

# 7.22.3.3 start()

```
void StartStop::start ( ) [override], [virtual]
```

# start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from \_lterator.

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

- · Iterators.h
- · StartStop.cpp

# 7.23 SynchronizedMeasurments Class Reference

start, stop and clear several measurements synchronized

```
#include <Iterators.h>
```

## **Public Member Functions**

SynchronizedMeasurments (TimeTagger \*tagger)

construct a SyncronizedMeasurments object - if you have an better idea how to call it please let me know

void registerMeasurement (\_Iterator \*measurement)

register a measurement (iterator) to the SynchronizedMeasurements-group.

• void clear ()

clear all registered measurements synchronously

• void start ()

start all registered measurements synchronously

• void stop ()

stop all registered measurements synchronously

## 7.23.1 Detailed Description

start, stop and clear several measurements synchronized

For the case that several measurements should be started, stopped or cleared at the very same time, a SynchronizedMeasrements object can be create to which all the measurements (also called iterators) can be registered with .registerMeasurement(measurement). Calling .stop(), .start() or .clear() on the Synchronized Measurements object will call the respective method on each of the registered measurements at the very same time. That means that all measurements taking part will have processed the very same time tags.

#### 7.23.2 Constructor & Destructor Documentation

## 7.23.2.1 SynchronizedMeasurments()

```
\label{thm:constraint} Synchronized \texttt{Measurments::} Synchronized \texttt{Measurments::} \\ \texttt{TimeTagger} * tagger) \quad [inline]
```

construct a SyncronizedMeasurments object - if you have an better idea how to call it please let me know

#### **Parameters**

tagger reference to a TimeTagger

#### 7.23.3 Member Function Documentation

```
7.23.3.1 clear()
```

```
void SynchronizedMeasurments::clear ( ) [inline]
```

clear all registered measurements synchronously

#### 7.23.3.2 registerMeasurement()

```
void SynchronizedMeasurments::registerMeasurement (
    __Iterator * measurement ) [inline]
```

register a measurement (iterator) to the SynchronizedMeasurements-group.

All available methods called on the SynchronizedMeasurements will happen at the very same time for all the registered measurements.

```
7.23.3.3 start()
```

```
void SynchronizedMeasurments::start ( ) [inline]
```

start all registered measurements synchronously

#### 7.23.3.4 stop()

```
void SynchronizedMeasurments::stop ( ) [inline]
```

stop all registered measurements synchronously

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

· Iterators.h

# 7.24 Tag Struct Reference

a single event on a channel

```
#include <TimeTagger.h>
```

## **Public Attributes**

· bool overflow

when set, there was an overflow on the communication channel.

channel\_t channel

the channel number

• timestamp\_t time

the timestamp on the event, in picoseconds

# 7.24.1 Detailed Description

a single event on a channel

Channel events are passed from the backend to registered iterators by the \_lterator::next() callback function.

A Tag describes a single event on a channel.

## 7.24.2 Member Data Documentation

#### 7.24.2.1 channel

```
channel_t Tag::channel
```

the channel number

## 7.24.2.2 overflow

```
bool Tag::overflow
```

when set, there was an overflow on the communication channel.

#### 7.24.2.3 time

```
timestamp_t Tag::time
```

the timestamp on the event, in picoseconds

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

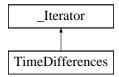
· TimeTagger.h

## 7.25 TimeDifferences Class Reference

Accumulates the time differences between clicks on two channels in one or more histograms.

```
#include <Iterators.h>
```

Inheritance diagram for TimeDifferences:



#### **Public Member Functions**

• TimeDifferences (TimeTagger \*tagger, channel\_t click\_channel, channel\_t start\_channel=CHANNEL\_ UNUSED, channel\_t next\_channel=CHANNEL\_UNUSED, channel\_t sync\_channel=CHANNEL\_UNUSED, timestamp\_t binwidth=1000, int n\_bins=1000, int n\_histograms=1)

constructor of a TimeDifferences measurement

· void start () override

start the iterator

GET\_DATA\_2D (getData, int, array\_out,)

get result data

- GET DATA 1D (getIndex, timestamp t, array out,)
- void setMaxCounts (int max\_counts)

set the number of sync/next clicks after which acquisition shall stop

• int getCounts ()

return the number of sync/next clicks

· bool ready ()

return 'true' if the maximum number of start clicks has been reached

## **Protected Member Functions**

bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) over-ride

update iterator state

• void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

#### 7.25.1 Detailed Description

Accumulates the time differences between clicks on two channels in one or more histograms.

Specifically, the thread waits for clicks on a first channel, the 'start channel', then measures the time difference between a start click and all subsequent clicks on a second channel, the 'click channel', and stores them in a histogram. The histogram range and resolution is specified by the number of bins and the binwidth. Clicks that fall outside the histogram range are ignored. Data accumulation is performed independently for all start clicks. This type of measurement is frequently referred to as 'multiple start, multiple stop' measurement and corresponds to a full auto- or cross-correlation measurement. The data obtained from subsequent start clicks can be accumulated into the same histogram (one-dimensional measurement) or into different histograms (two-dimensional measurement). In this way you can perform more complex two-dimensional time-difference measurements. Specifically, after each click on the next channel, the histogram index is incremented by one and reset to zero after the last histogram has been served. You can also provide an external synchronization trigger that resets the histogram index to zero.

#### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 TimeDifferences()

## constructor of a TimeDifferences measurement

#### **Parameters**

tagger	reference to a TimeTagger	
click_channel	channel that increments the count in a bin	
start_channel	channel that sets start times relative to which clicks on the click channel are measured	
next_channel	channel that increments the histogram index	
sync_channel	channel that resets the histogram index to zero	
binwidth	width width of one histogram bin in ps	
n_bins	number of bins in each histogram	
n_histograms	grams number of histograms	

## 7.25.3 Member Function Documentation

#### 7.25.3.1 clear\_impl()

```
void TimeDifferences::clear_impl ( ) [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from \_Iterator.

#### 7.25.3.2 GET\_DATA\_2D()

```
TimeDifferences::GET_DATA_2D (
    getData ,
    int ,
    array_out )
```

get result data

#### **Parameters**

ARGOUTVIEWM_ARRAY2	pointer receiving pointer to data
DIM1	pointer receiving first dimension
DIM2	pointer receiving second dimension

## 7.25.3.3 getCounts()

```
int TimeDifferences::getCounts ( )
```

return the number of sync/next clicks

## 7.25.3.4 next\_impl()

update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events	
start_time	earliest event in the block	
end_time	start_time of the next block, not including in this block	

#### Returns

if the content of this block was modified

Implements \_Iterator.

## 7.25.3.5 ready()

```
bool TimeDifferences::ready ( )
```

return 'true' if the maximum number of start clicks has been reached

#### 7.25.3.6 setMaxCounts()

set the number of sync/next clicks after which acquisition shall stop

## **Parameters**

counts	maximum number of sync/next clicks
Counts	maximum number of symomext clicks

## 7.25.3.7 start()

```
void TimeDifferences::start ( ) [override], [virtual]
```

start the iterator

The default behavior for iterators is to start automatically on creation.

Reimplemented from \_lterator.

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

- Iterators.h
- TimeDifferences.cpp

# 7.26 TimeTagger Class Reference

```
backend for the TimeTagger.
```

```
#include <TimeTagger.h>
```

# **Public Types**

- typedef std::function < void( lterator \*) > lteratorCallback
- typedef std::map< | Iterator \*, | IteratorCallbackMap</li>

#### **Public Member Functions**

virtual void reset ()=0

reset the TimeTagger object to default settings and detach all iterators

virtual void setTestSignalDivider (int divider)=0

set the divider for the frequency of the test signal

virtual void setTriggerLevel (channel t channel, double voltage)=0

set the trigger voltage threshold of a channel

• virtual double getTriggerLevel (channel\_t channel)=0

get the trigger voltage threshold of a channel

• virtual void setInputDelay (channel\_t channel, timestamp\_t delay)=0

set time delay on a channel

virtual timestamp\_t getInputDelay (channel\_t channel)=0

get time delay of a channel

- virtual void setConditionalFilter (std::vector< channel\_t > trigger, std::vector< channel\_t > filtered)=0
   configures the conditional filter
- virtual std::vector< channel\_t > getConditionalFilterTrigger ()=0

fetches the configuration of the conditional filter

virtual std::vector< channel t > getConditionalFilterFiltered ()=0

fetches the configuration of the conditional filter

virtual void setFilter (bool state)=0

enables or disables the filter on the FPGA board.

virtual bool getFilter ()=0

returns the filter state on the FPGA board

virtual void setNormalization (bool state)=0

enables or disables the normalization of the distribution.

• virtual bool getNormalization ()=0

returns the the normalization of the distribution.

• virtual void setHardwareBufferSize (int size)=0

sets the maximum USB buffer size

• virtual int getHardwareBufferSize ()=0

queries the size of the USB queue

• virtual timestamp\_t setDeadtime (channel\_t channel, timestamp\_t deadtime)=0

set the deadtime between two edges on the same channel.

virtual timestamp\_t getDeadtime (channel\_t channel)=0

get the deadtime between two edges on the same channel.

virtual void setEventDivider (channel\_t channel, unsigned int divider)=0

Divides the amount of transmitted edge per channel.

• virtual unsigned int getEventDivider (channel\_t channel)=0

Returns the factor of the dividing filter.

virtual void registerChannel (channel t channel)=0

register a FPGA channel.

virtual void unregisterChannel (channel\_t channel)=0

release a previously registered channel.

virtual void setTestSignal (channel t channel, bool enabled)=0

enable the calibration on a channel.

- virtual void setTestSignal (std::vector< channel t > channel, bool enabled)=0
- virtual bool getTestSignal (channel\_t channel)=0

fetch the status of the test signal generator

virtual void autoCalibration (bool verbose=true)=0

runs a calibrations based on the on-chip uncorrelated signal generator.

virtual std::string getSerial ()=0

identifies the hardware by serial number

virtual std::string getModel ()=0

identifies the hardware by Time Tagger Model

virtual int getChannelNumberScheme ()=0

Fetch the configured numbering scheme for this TimeTagger object.

virtual std::vector< double > getDACRange ()=0

returns the minumum and the maximum voltage of the DACs as a trigger reference

GET\_DATA\_2D (getDistributionCount, long long, array\_out, virtual)=0

get internal calibration data

GET\_DATA\_2D (getDistributionPSecs, timestamp\_t, array\_out, virtual)=0

get internal calibration data

virtual channel\_t getChannels ()=0

fetch the amount of channels

- virtual std::vector< channel\_t > getChannelList (int type=TT\_CHANNEL\_RISING\_AND\_FALLING\_ED
   GES)=0
- virtual channel\_t getInvertedChannel (channel\_t channel)=0

get the falling channel id for a raising channel and vice versa

• virtual bool isUnusedChannel (channel t channel)=0

compares the provided channel with CHANNEL\_UNUSED

virtual timestamp\_t getPsPerClock ()=0

fetch the duration of each clock cycle in picoseconds

• virtual long long getOverflows ()=0

get overflow count

• virtual void clearOverflows ()=0

clear overflow counter

• virtual long long getOverflowsAndClear ()=0

get and clear overflow counter

virtual void sync ()=0

Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.

virtual std::string getPcbVersion ()=0

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.

virtual std::string getSensorData ()=0

Show the status of the sensor data from the FPGA and peripherals on the console.

• virtual void setLED (uint32\_t bitmask)=0

Enforce a state to the LEDs 0: led\_status[R] 16: led\_status[R] - mux 1: led\_status[B] 17: led\_status[B] - mux 2: led\_status[G] 18: led\_status[G] - mux 3: led\_power[R] 19: led\_power[R] - mux 4: led\_power[B] 20: led\_power[B] - mux 5: led\_power[G] 21: led\_power[G] - mux 6: led\_clock[R] 22: led\_clock[R] - mux 7: led\_clock[B] 23: led\_clock[B] - mux 8: led\_clock[G] 24: led\_clock[G] - mux.

virtual void runSynchronized (const IteratorCallbackMap &callbacks, bool block=true)=0

Run synchronized callbacks for a list of iterators.

## **Protected Member Functions**

- TimeTagger ()
  - abstract interface class
- virtual ~TimeTagger ()
- TimeTagger (const TimeTagger &)=delete
- TimeTagger & operator= (const TimeTagger &)=delete

#### **Friends**

· class Iterator

## 7.26.1 Detailed Description

backend for the TimeTagger.

The TimeTagger class connects to the hardware, and handles the communication over the usb. There may be only one instance of the backend per physical device.

#### 7.26.2 Constructor & Destructor Documentation

```
7.26.2.1 TimeTagger()
TimeTagger::TimeTagger ( ) [inline], [protected]
abstract interface class

7.26.2.2 ~TimeTagger()
virtual TimeTagger::~TimeTagger ( ) [inline], [protected], [virtual]
destructor
```

## 7.26.3 Member Function Documentation

## 7.26.3.1 autoCalibration()

runs a calibrations based on the on-chip uncorrelated signal generator.

#### **Parameters**

```
verbose Verbose output on stdout
```

## 7.26.3.2 clearOverflows()

```
virtual void TimeTagger::clearOverflows ( ) [pure virtual]
```

clear overflow counter

Sets the overflow counter to zero

## 7.26.3.3 **GET\_DATA\_2D()** [1/2]

get internal calibration data

## 7.26.3.4 **GET\_DATA\_2D()** [2/2]

get internal calibration data

## 7.26.3.5 getChannelNumberScheme()

```
virtual int TimeTagger::getChannelNumberScheme ( ) [pure virtual]
```

Fetch the configured numbering scheme for this TimeTagger object.

Please see setTimeTaggerChannelNumberScheme() for details.

```
7.26.3.6 getChannels()
```

```
virtual channel_t TimeTagger::getChannels ( ) [pure virtual]
```

fetch the amount of channels

**Deprecated** Use getChannelList instead.

#### 7.26.3.7 getConditionalFilterFiltered()

```
virtual std::vector<channel_t> TimeTagger::getConditionalFilterFiltered ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see setConditionalFilter

## 7.26.3.8 getConditionalFilterTrigger()

```
virtual std::vector<channel_t> TimeTagger::getConditionalFilterTrigger ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see setConditionalFilter

## 7.26.3.9 getDACRange()

```
virtual std::vector<double> TimeTagger::getDACRange ( ) [pure virtual]
```

returns the minumum and the maximum voltage of the DACs as a trigger reference

## 7.26.3.10 getDeadtime()

get the deadtime between two edges on the same channel.

This function gets the user configureable deadtime.

#### **Parameters**

channel ch	nannel to be queried
------------	----------------------

#### Returns

the real configured deadtime

## 7.26.3.11 getEventDivider()

Returns the factor of the dividing filter.

See for further details.

**Parameters** 

```
channel channel to be queried
```

#### Returns

the configured divider

#### 7.26.3.12 getFilter()

```
virtual bool TimeTagger::getFilter ( ) [pure virtual]
```

returns the filter state on the FPGA board

## **Deprecated** use getConditionalFilter\*

The laserfilter disables transmission of nontriggered tags on channel 7. This is a deprecated specialization of getConditionalFilter\*.

## 7.26.3.13 getHardwareBufferSize()

```
virtual int TimeTagger::getHardwareBufferSize ( ) [pure virtual]
```

queries the size of the USB queue

See for more information.

Returns

the actual size of the USB queue in events

#### 7.26.3.14 getInputDelay()

get time delay of a channel

see setInputDelay

#### **Parameters**

	channel	the channel
--	---------	-------------

#### 7.26.3.15 getInvertedChannel()

get the falling channel id for a raising channel and vice versa

#### 7.26.3.16 getModel()

```
virtual std::string TimeTagger::getModel ( ) [pure virtual]
```

identifies the hardware by Time Tagger Model

## 7.26.3.17 getNormalization()

```
virtual bool TimeTagger::getNormalization ( ) [pure virtual]
```

returns the the normalization of the distribution.

Refer the Manual for a description of this function.

## 7.26.3.18 getOverflows()

```
virtual long long TimeTagger::getOverflows ( ) [pure virtual]
```

get overflow count

Get the number of communication overflows occured

## 7.26.3.19 getOverflowsAndClear()

```
virtual long long TimeTagger::getOverflowsAndClear ( ) [pure virtual]
```

get and clear overflow counter

Get the number of communication overflows occured and sets them to zero

## 7.26.3.20 getPcbVersion()

```
virtual std::string TimeTagger::getPcbVersion ( ) [pure virtual]
```

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.

## 7.26.3.21 getPsPerClock()

```
virtual timestamp_t TimeTagger::getPsPerClock ( ) [pure virtual]
```

fetch the duration of each clock cycle in picoseconds

## 7.26.3.22 getSensorData()

```
virtual std::string TimeTagger::getSensorData ( ) [pure virtual]
```

Show the status of the sensor data from the FPGA and peripherals on the console.

## 7.26.3.23 getSerial()

```
virtual std::string TimeTagger::getSerial ( ) [pure virtual]
```

identifies the hardware by serial number

# 7.26.3.24 getTestSignal()

fetch the status of the test signal generator

## **Parameters**

channel	the channel
CHAHILE	liie chaillei

## 7.26.3.25 getTriggerLevel()

get the trigger voltage threshold of a channel

## **Parameters**

```
channel the channel
```

## 7.26.3.26 isUnusedChannel()

compares the provided channel with CHANNEL\_UNUSED

But also keeps care about the channel number scheme and selects either CHANNEL\_UNUSED or CHANNEL\_
UNUSED OLD

# 7.26.3.27 registerChannel()

register a FPGA channel.

Only events on previously registered channels will be transfered over the communication channel.

#### **Parameters**

```
channel the channel
```

## 7.26.3.28 reset()

```
virtual void TimeTagger::reset ( ) [pure virtual]
```

reset the TimeTagger object to default settings and detach all iterators

## 7.26.3.29 runSynchronized()

Run synchronized callbacks for a list of iterators.

This method has a list of callbacks for a list of iterators. Those callbacks are called for a synchronized data set, but in parallel. They are called from an internal worker thread. As the data set is synchronized, this creates a bottleneck for one worker thread, so only fast and non-blocking callbacks are allowed.

#### **Parameters**

callbacks	Map of callbacks per iterator
block	Shall this method block until all callbacks are finished

#### 7.26.3.30 setConditionalFilter()

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are supressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

#### **Parameters**

trigger	the channels that sets the condition
filtered	the channels that are filtered by the condition

## 7.26.3.31 setDeadtime()

set the deadtime between two edges on the same channel.

This function sets the user configureable deadtime. The requested time will be rounded to the nearest multiple of the clock time. The deadtime will also be clamped to device specific limitations.

As the actual deadtime will be altered, the real value will be returned.

#### **Parameters**

channel	channel to be configured
deadtime	new deadtime

#### Returns

the real configured deadtime

## 7.26.3.32 setEventDivider()

Divides the amount of transmitted edge per channel.

This filter decimates the events on a given channel by a specified. factor. So for a divider n, every nth event is transmitted through the filter and n-1 events are skipped between consecutive transmitted events. If a conditional filter is also active, the event divider is applied after the conditional filter, so the conditional is applied to the complete event stream and only events which pass the conditional filter are forwarded to the divider.

As it is a hardware filter, it reduces the required USB bandwidth and CPU processing power, but it cannot be configured for virtual channels.

#### **Parameters**

channel	channel to be configured
divider	new divider, must be smaller than 65536

#### 7.26.3.33 setFilter()

enables or disables the filter on the FPGA board.

## **Deprecated** use setConditionalFilter

The filter disables transmission of nontriggered tags on channel 7. This is a deprecated specialization of set ← ConditionalFilter.

## 7.26.3.34 setHardwareBufferSize()

```
\begin{tabular}{ll} \begin{tabular}{ll} virtual void TimeTagger::setHardwareBufferSize ( \\ & int size ) & [pure virtual] \end{tabular}
```

sets the maximum USB buffer size

This option controls the maximum buffer size of the USB connection. This can be used to balance low input latency vs high (peak) throughput.

#### **Parameters**

num buffer size in	the maximum	size
--------------------	-------------	------

#### 7.26.3.35 setInputDelay()

set time delay on a channel

When set, every event on the channel is delayed by the given delay in picoseconds. Setting larger time delays consumes, dependend on input signal, a significant amount of memory. Implementers are adviced to keep the delay below 1 micro second.

## **Parameters**

channel	the channel to set
delay	the delay in picoseconds

## 7.26.3.36 setLED()

Enforce a state to the LEDs 0: led\_status[R] 16: led\_status[R] - mux 1: led\_status[B] 17: led\_status[B] - mux 2: led\_status[G] 18: led\_status[G] - mux 3: led\_power[R] 19: led\_power[R] - mux 4: led\_power[B] 20: led\_power[B] - mux 5: led\_power[G] 21: led\_power[G] - mux 6: led\_clock[R] 22: led\_clock[R] - mux 7: led\_clock[B] 23: led\_clock[B] - mux 8: led\_clock[G] 24: led\_clock[G] - mux.

#### 7.26.3.37 setNormalization()

enables or disables the normalization of the distribution.

Refer the Manual for a description of this function.

## 7.26.3.38 setTestSignal()

enable the calibration on a channel.

This will connect or disconnect the channel with the on-chip uncorrelated signal generator.

#### **Parameters**

channel	the channel
enabled	enabled / disabled flag

#### 7.26.3.39 setTestSignalDivider()

set the divider for the frequency of the test signal

The base clock of the test signal oscillator for the Time Tagger Ultra is running at 100.8 MHz sampled down by an factor of 2 to have a similar base clock as the Time Tagger 20 ( $\sim$ 50 MHz). The default divider is 63 ->  $\sim$ 800 kEvents/s

## **Parameters**

divider	frequency divisor of the oscillator

## 7.26.3.40 setTriggerLevel()

set the trigger voltage threshold of a channel

## **Parameters**

channel	the channel to set
voltage	voltage level [01]

```
7.26.3.41 sync()
virtual void TimeTagger::sync ( ) [pure virtual]
```

Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.

#### 7.26.3.42 unregisterChannel()

release a previously registered channel.

#### **Parameters**

channel
channel

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

· TimeTagger.h

# 7.27 TimetaggerFPGA Class Reference

**Public Member Functions** 

- TimetaggerFPGA (std::string serial="", int blocksize=1024)
- int configure ()
- bool configured ()
- void setTestSignalDivider (int divider)
- · void setTriggerLevel (int channel, double voltage)
- · bool sendDacCommand (int prefix, int control, int address, int data, int feature)
- int read (void \*buffer, int buffersize)
- bool open (std::string serial)
- std::string getSerial ()
- std::string getModel ()
- std::vector< double > getDACRange ()
- bool setWireIn (unsigned addr, int value)
- int getWireOut (unsigned addr)
- long WriteToPipeIn (int epAddr, long length, void \*data)
- bool ActivateTrigger (unsigned addr, unsigned bit)
- bool isTriggered (unsigned addr, unsigned bit)
- bool UpdateWireIns ()
- bool UpdateWireOuts ()
- bool UpdateTriggerOuts ()
- std::string getSensorData ()
- std::vector< char > SendToFlash (const char \*command, int input\_length, int output\_length=0, bool block-ing=false)
- void setLED (uint32\_t bitmask)

## **Static Public Member Functions**

- static std::vector< std::string > getDeviceList ()
- static std::string getTimeTaggerModel (std::string serial)

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

- · TimetaggerFPGA.h
- TimetaggerFPGA.cpp

# 7.28 TimeTaggerModel Class Reference

#### **Static Public Member Functions**

static std::string mapOkModelToTimeTaggerModel (OpalKellyLegacy::okCFrontPanel::BoardModel model)

#### **Static Public Attributes**

- static const std::string MODEL\_UNKNOWN = "unknown"
- static const std::string MODEL\_TIMETAGGER\_20 = "Time Tagger 20"
- static const std::string MODEL\_TIMETAGGER\_ULTRA = "Time Tagger Ultra"

## **Friends**

· class TimetaggerFPGA

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

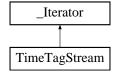
- · TimetaggerFPGA.h
- · TimetaggerFPGA.cpp

# 7.29 TimeTagStream Class Reference

access the time tag stream

```
#include <Iterators.h>
```

Inheritance diagram for TimeTagStream:



## **Public Member Functions**

TimeTagStream (TimeTagger \*tagger, int n\_max\_events, const std::vector< channel\_t > &channels=std ← ::vector< channel\_t >())

constructor of a TimeTagStream thread

∼TimeTagStream ()

thd

• size\_t getCounts ()

get incoming time tags

• void getData (TimeTagStreamBuffer &buffer)

#### **Protected Member Functions**

 bool next\_impl (std::vector < Tag > &incoming\_tags, timestamp\_t begin\_time, timestamp\_t end\_time) override

update iterator state

void clear\_impl () override

clear Iterator state.

#### **Additional Inherited Members**

## 7.29.1 Detailed Description

access the time tag stream

## 7.29.2 Constructor & Destructor Documentation

```
7.29.2.1 ~TimeTagStream()
```

```
\label{timeTagStream::} \texttt{TimeTagStream ( )}
```

tbd

# 7.29.3 Member Function Documentation

```
7.29.3.1 clear_impl()
```

```
void TimeTagStream::clear_impl () [override], [protected], [virtual]
```

clear Iterator state.

Each Iterator should implement the clear\_impl() method to reset its internal state. The clear\_impl() function is guarded by the update lock.

Reimplemented from <u>\_lterator</u>.

## 7.29.3.2 getCounts()

```
size_t TimeTagStream::getCounts ( )
```

#### get incoming time tags

All incoming time tags are stored in a buffer (max size: max\_tags). The buffer is cleared after retrieving the data with getData() return the number of stored tags

## 7.29.3.3 next\_impl()

#### update iterator state

Each Iterator must implement the next\_impl() method. The next\_impl() function is guarded by the update lock.

The backend delivers each Tag on each registered channel to this callback function.

#### **Parameters**

list	block of events
start_time	earliest event in the block
end_time	start_time of the next block, not including in this block

#### Returns

if the content of this block was modified

Implements \_Iterator.

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

- · Iterators.h
- · Iterator.cpp

# 7.30 TimeTagStreamBuffer Class Reference

**Public Member Functions** 

- GET\_DATA\_1D (getOverflows, unsigned char, array\_out,)
- **GET\_DATA\_1D** (getChannels, channel\_t, array\_out,)
- **GET\_DATA\_1D** (getTimestamps, timestamp\_t, array\_out,)

# **Public Attributes**

- std::vector < unsigned char > tagOverflows
- $std::vector < channel_t > tagChannels$
- std::vector< timestamp\_t > tagTimestamps
- int size
- bool hasOverflows
- timestamp\_t tStart
- timestamp\_t tGetData

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

· Iterators.h

# Index

_lterator, 15	HistogramLogBins, 52
lterator, 17	Iterator, 54
$\sim$ _Iterator, 17	Scope, 57
channels_registered, 20	StartStop, 60
clear, 17	TimeDifferences, 65
clear_impl, 17	TimeTagStream, 83
getCaptureDuration, 17	clearOverflows
getNewVirtualChannel, 18	TimeTagger, 71
isRunning, 18	Coincidence, 21
lock, 18	Coincidence, 22
next impl, 18	getChannel, 22
registerChannel, 19	Coincidences, 23
running, 20	Coincidences, 24
start, 19	getChannels, 24
startFor, 19	next_impl, 24
stop, 20	Combiner, 25
unlock, 20	clear_impl, 26
unregisterChannel, 20	Combiner, 26
~Dump	GET_DATA_1D, 26
Dump, 43	getChannel, 26
~TimeTagStream	next_impl, 27
TimeTagStream, 83	Correlation, 27
~TimeTagger	clear impl, 29
	Correlation, 28
TimeTagger, 70	GET_DATA_1D, 29
~_Iterator	next_impl, 30
_lterator, 17	·
Array < T >, 21	CountBetweenMarkers, 30
autoCalibration	clear_impl, 32
TimeTagger, 70	CountBetweenMarkers, 31 GET_DATA_1D, 32, 33
Time ragger, 70	
base iterators, 12	next_impl, 33
Dump, 13	ready, 33
TimeTagStream, 13	Counter, 34
Time ragotteam, 13	clear_impl, 36
channel	Counter, 34
Tag, 63	GET_DATA_2D, 36
channels_registered	next_impl, 36
Iterator, 20	start, 37
clear	Countrate, 37
Iterator, 17	clear_impl, 38
SynchronizedMeasurments, 62	Countrate, 38
•	GET_DATA_1D, 38
clear_impl	next_impl, 39
_lterator, 17	start, 39
Combiner, 26	Dalamad Obamad 40
Correlation, 29	DelayedChannel, 40
CountBetweenMarkers, 32	DelayedChannel, 40
Counter, 36	getChannel, 41
Countrate, 38	next_impl, 41
Dump, 43	setDelay, 41

88 INDEX

start, 42	getInvertedChannel
Dump, 42	TimeTagger, 74
$\sim$ Dump, 43	getModel
base iterators, 13	TimeTagger, 74
clear_impl, 43	getNewVirtualChannel
next impl, 43	•
next_impi, 43	_lterator, 18
Event 44	getNormalization
Event, 44	TimeTagger, 74
Flim 44	getOverflows
Flim, 44	TimeTagger, 74
Flim, 45	getOverflowsAndClear
FrequencyMultiplier, 45	TimeTagger, 74
FrequencyMultiplier, 46	getPcbVersion
next_impl, 47	TimeTagger, 74
	getPsPerClock
GET_DATA_1D	TimeTagger, 75
Combiner, 26	
Correlation, 29	getSensorData
CountBetweenMarkers, 32, 33	TimeTagger, 75
Countrate, 38	getSerial
GET DATA 2D	TimeTagger, 75
	getTestSignal
Counter, 36	TimeTagger, 75
TimeDifferences, 66	getTriggerLevel
TimeTagger, 71	TimeTagger, 75
GatedChannel, 47	1.39 - ,
GatedChannel, 48	Histogram, 49
getChannel, 49	Histogram, 50
next_impl, 49	HistogramLogBins, 51
getCaptureDuration	
Iterator, 17	clear_impl, 52
getChannel	HistogramLogBins, 51
Coincidence, 22	next_impl, 52
Combiner, 26	isRunning
DelayedChannel, 41	_lterator, 18
GatedChannel, 49	isUnusedChannel
getChannelNumberScheme	TimeTagger, 76
TimeTagger, 71	Iterator, 53
getChannels	clear_impl, 54
Coincidences, 24	Iterator, 54
TimeTagger, 71	next, 54
getConditionalFilterFiltered	next_impl, 54
TimeTagger, 72	size, 55
getConditionalFilterTrigger	3126, 33
TimeTagger, 72	LED, 55
getCounts	
	lock
TimeDifferences, 66	_lterator, 18
TimeTagStream, 83	
getDACRange	next
TimeTagger, 72	Iterator, 54
getDeadtime	next_impl
TimeTagger, 72	_lterator, 18
getEventDivider	Coincidences, 24
TimeTagger, 73	Combiner, 27
getFilter	Correlation, 30
TimeTagger, 73	CountBetweenMarkers, 33
getHardwareBufferSize	Counter, 36
TimeTagger, 73	Countrate, 39
getInputDelay	DelayedChannel, 41
TimeTagger, 73	Dump, 43

INDEX 89

FrequencyMultiplier, 47	TimeTagger, 80
GatedChannel, 49	size
HistogramLogBins, 52	Iterator, 55
Iterator, 54	start
Scope, 57	Iterator, 19
StartStop, 60	Counter, 37
TimeDifferences, 66	Countrate, 39
TimeTagStream, 84	DelayedChannel, 42
rimo ragotroam, o r	Scope, 58
overflow	StartStop, 60
Tag, 63	SynchronizedMeasurments, 62
rag, 00	TimeDifferences, 67
PRBS, 56	
11120,00	startFor
ready	_lterator, 19
CountBetweenMarkers, 33	StartStop, 58
	clear_impl, 60
TimeDifferences, 67	next_impl, 60
registerChannel	start, 60
_lterator, 19	StartStop, 59
TimeTagger, 76	stop
registerMeasurement	_lterator, 20
SynchronizedMeasurments, 62	SynchronizedMeasurments, 62
reset	sync
TimeTagger, 76	TimeTagger, 80
runSynchronized	SynchronizedMeasurments, 61
TimeTagger, 76	clear, 62
running	registerMeasurement, 62
_Iterator, 20	start, 62
	•
	stop, 62
Scope, 56	•
Scope, 56 clear impl. 57	SynchronizedMeasurments, 61
clear_impl, 57	SynchronizedMeasurments, 61
clear_impl, 57 next_impl, 57	SynchronizedMeasurments, 61 Tag, 63
clear_impl, 57 next_impl, 57 Scope, 57	SynchronizedMeasurments, 61  Tag, 63 channel, 63
clear_impl, 57 next_impl, 57 Scope, 57 start, 58	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts	SynchronizedMeasurments, 61  Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization	Tag, 63 channel, 63 overflow, 63 time, 63  time Tag, 63 TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84  TimeTagStreamBuffer, 84
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84  TimeTagStreamBuffer, 84  TimeTagger, 68
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79 setTestSignal	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84  TimeTagStreamBuffer, 84  TimeTagger, 68 ~TimeTagger, 70
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79 setTestSignal TimeTagger, 79	Tag, 63 channel, 63 overflow, 63 time Tag, 63 TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65 TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84 TimeTagStreamBuffer, 84 TimeTagger, 68 ~TimeTagger, 70 autoCalibration, 70
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79 setTestSignal TimeTagger, 79 setTestSignalDivider	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84  TimeTagStreamBuffer, 84  TimeTagger, 68 ~TimeTagger, 70 autoCalibration, 70 clearOverflows, 71
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79 setTestSignal TimeTagger, 79	Tag, 63 channel, 63 overflow, 63 time Tag, 63 TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65 TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84 TimeTagStreamBuffer, 84 TimeTagger, 68 ~TimeTagger, 70 autoCalibration, 70
clear_impl, 57 next_impl, 57 Scope, 57 start, 58 setConditionalFilter TimeTagger, 77 setDeadtime TimeTagger, 77 setDelay DelayedChannel, 41 setEventDivider TimeTagger, 78 setFilter TimeTagger, 78 setHardwareBufferSize TimeTagger, 78 setInputDelay TimeTagger, 79 setLED TimeTagger, 79 setMaxCounts TimeDifferences, 67 setNormalization TimeTagger, 79 setTestSignal TimeTagger, 79 setTestSignalDivider	Tag, 63 channel, 63 overflow, 63 time, 63  TimeDifferences, 64 clear_impl, 65 GET_DATA_2D, 66 getCounts, 66 next_impl, 66 ready, 67 setMaxCounts, 67 start, 67 TimeDifferences, 65  TimeTagStream, 82 ~TimeTagStream, 83 base iterators, 13 clear_impl, 83 getCounts, 83 next_impl, 84  TimeTagStreamBuffer, 84  TimeTagger, 68 ~TimeTagger, 70 autoCalibration, 70 clearOverflows, 71

90 INDEX

```
getChannels, 71
     getConditionalFilterFiltered, 72
    getConditionalFilterTrigger, 72
    getDACRange, 72
    getDeadtime, 72
    getEventDivider, 73
    getFilter, 73
    getHardwareBufferSize, 73
    getInputDelay, 73
     getInvertedChannel, 74
    getModel, 74
    getNormalization, 74
    getOverflows, 74
    getOverflowsAndClear, 74
    getPcbVersion, 74
    getPsPerClock, 75
     getSensorData, 75
     getSerial, 75
    getTestSignal, 75
    getTriggerLevel, 75
    isUnusedChannel, 76
     registerChannel, 76
     reset, 76
    runSynchronized, 76
     setConditionalFilter, 77
    setDeadtime, 77
    setEventDivider, 78
    setFilter, 78
     setHardwareBufferSize, 78
    setInputDelay, 79
    setLED, 79
    setNormalization, 79
     setTestSignal, 79
    setTestSignalDivider, 80
    setTriggerLevel, 80
    sync, 80
    TimeTagger, 70
     unregisterChannel, 81
TimeTagger backend, 11
TimeTaggerModel, 82
TimetaggerFPGA, 81
unlock
     _Iterator, 20
unregisterChannel
     _Iterator, 20
     TimeTagger, 81
```