The Fastest Verification

ZeBu™ C API Reference Manual

Version 6_3_1

May 2011

Contents

Chapter 1

Zebu C API

1.1 Legal Notice

1.2 About this manual

The ZeBu C API Reference Manual provides detailed information on C library, files, structures and functions necessary to write a C test bench to verify your design with the ZeBu verification platform.

Chapter 2

Zebu C API Class Index

2.1 Zebu C API Class List

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

Chapter 3

Zebu C API File Index

3.1 Zebu C API File List

Here is a list of all files with brief descriptions:

Chapter 4

Zebu C API Page Index

4.1 Zebu C API Related Pages

Here is a list of all related documentation pages:

Chapter 5

Zebu C API Class Documentation

5.1 ZEBU_Board Struct Reference

5.1.1 Detailed Description

board handler.

C type for ZEBU::Board

5.2 ZEBU_Clock Struct Reference

5.2.1 Detailed Description

clock handler

C type for ZEBU::Clock

5.3 ZEBU_Driver Struct Reference

5.3.1 Detailed Description

Implement public interface for Zebu drivers.

A driver is a mean to drive and monitor the interface to the design under test. Each driver offers specific behavior: it can be a link to a C/C++ test bench, a waveform monitor, a test vector driver ...

See also:

ZEBU_Board_getDriver

5.4 ZEBU_Filter Struct Reference

5.4.1 Detailed Description

Implement public interface for Zebu filter. Allow to filter components accessible from the ZeBu interface: internal signals, driver signals, internal and external memories, clocks ...

See also:

ZEBU_Board_saveLogicState
ZEBU_Board_restoreLogicState

5.5 ZEBU_FlexibleLocalProbeFile Struct Reference

5.5.1 Detailed Description

Implement public interface for Zebu flexible local probe file.

A flexible local probe file allows to dump the data connected to hardware flexible local probe groups in different formats: ZTDB, VCD or FSDB. ZTDB format offers best performances and can be converted off-line to other waveforms formats.

5.6 ZEBU_LocalTraceDumper Struct Reference

5.6.1 Detailed Description

Implement public interface for Zebu tracer dumper.

A tracer dumper allows to dump the data connected to a hardware local tracer in different formats: ZTDB, VCD or FSDB. ZTDB format offers best performances and can be converted off-line to other waveforms formats.

5.7 ZEBU_LocalTraceDumperGroup Struct Reference

5.7.1 Detailed Description

Implement public interface for Zebu tracer dumper.

A tracer dumper allows to dump the data connected to hardware local tracers in different formats: ZTDB, VCD or FSDB. ZTDB format offers best performances and can be converted off-line to other waveforms formats.

5.8 ZEBU_LocalTraceImporter Struct Reference

5.8.1 Detailed Description

Implement public interface for Zebu tracer importer.

A tracer reader allows to use DPI imported functions. local tracer with signals.

5.9 ZEBU_LocalTraceImporterGroup Struct Reference

5.9.1 Detailed Description

Implement public interface for Zebu tracer importer.

A tracer reader allows to use DPI imported functions. local tracer with signals.

5.10 ZEBU_LocalTraceReader Struct Reference

5.10.1 Detailed Description

Implement public interface for Zebu tracer reader.

A tracer reader allows to read the data connected to a hardware local tracer with signals.

5.11 ZEBU_LocalTraceReaderGroup Struct Reference

5.11.1 Detailed Description

Implement public interface for Zebu tracer reader.

A tracer reader allows to read the data connected to a hardware local tracer with signals.

5.12 ZEBU_LogicAnalyzer Struct Reference

5.12.1 Detailed Description

logic analyzer handler

C type for ZEBU::LogicAnalyzer

5.13 ZEBU_Memory Struct Reference

5.13.1 Detailed Description

memory ZEBU_memory handler

C type for ZEBU::Memory

5.14 ZEBU_Port Struct Reference

5.14.1 Detailed Description

Implement public interface for ports.

5.15 ZEBU_Signal Struct Reference

5.15.1 Detailed Description

Implement public interface for signals.

You use the Signal class to map C variable to ZeBu signals. ZeBu signals can be IOs of your DUT or internal state signals.

See also:

ZEBU_Board_getSignal
ZEBU_Driver_getSignal

5.16 ZEBU_Trigger Struct Reference

5.16.1 Detailed Description

Implement public interface for triggers.

See also:
ZEBU_Board_getTrigger

5.17 ZEBU_WaveFile Struct Reference

5.17.1 Detailed Description

wave file handler.

C type for ZEBU::WaveFile

Chapter 6

Zebu C API File Documentation

6.1 libZebu.h File Reference

Include dependency graph for libZebu.h:

Typedefs

- typedef ZEBU_Board ZEBU_Board
- typedef ZEBU_Board_DriverInfoIterator ZEBU_Board_DriverInfoIterator
- typedef ZEBU_Board_MemoryIterator ZEBU_Board_MemoryIterator
- typedef ZEBU_Board_PortInfoIterator ZEBU_Board_PortInfoIterator
- typedef ZEBU_Board_SignalIterator ZEBU_Board_SignalIterator
- typedef ZEBU_Clock ZEBU_Clock
- typedef ZEBU_Driver ZEBU_Driver
- typedef ZEBU_Driver_SignalIterator ZEBU_Driver_SignalIterator
- typedef ZEBU_FastHardwareState ZEBU_FastHardwareState
- typedef ZEBU_Filter ZEBU_Filter
- typedef ZEBU_FlexibleLocalProbeFile ZEBU_FlexibleLocalProbeFile
- typedef ZEBU_LocalTraceDumper ZEBU_LocalTraceDumper
- typedef ZEBU_LocalTraceDumperGroup ZEBU_LocalTraceDumperGroup
- typedef ZEBU_LocalTraceImporter ZEBU_LocalTraceImporter
- typedef ZEBU_LocalTraceImporterGroup ZEBU_LocalTraceImporterGroup
- typedef ZEBU_LocalTraceReader ZEBU_LocalTraceReader
- typedef ZEBU_LocalTraceReaderGroup ZEBU_LocalTraceReaderGroup
- typedef ZEBU_LogicAnalyzer ZEBU_LogicAnalyzer
- typedef ZEBU_Memory ZEBU_Memory
- typedef ZEBU_PartMemoryBuilder ZEBU_PartMemoryBuilder

- typedef ZEBU_Port ZEBU_Port

- typedef ZEBU_Signal ZEBU_Signal

- typedef ZEBU_Trigger ZEBU_Trigger

- typedef ZEBU_Value ZEBU_Value

- typedef ZEBU_ValueChange_Iterator ZEBU_ValueChange_Iterator

- typedef ZEBU_vecval ZEBU_vecval

- typedef ZEBU_WaveFile ZEBU_WaveFile

6.1.1 Typedef Documentation

6.1.1.1 typedef struct ZEBU_Board ZEBU_Board

6.1.1.2 typedef struct ZEBU_Board_DriverInfoIterator ZEBU_Board_DriverInfoIterator

6.1.1.3 typedef struct ZEBU_Board_MemoryIterator ZEBU_Board_MemoryIterator

6.1.1.4 typedef struct ZEBU_Board_PortInfoIterator ZEBU_Board_PortInfoIterator

6.1.1.5 typedef struct ZEBU_Board_SignalIterator ZEBU_Board_SignalIterator

6.1.1.6 typedef struct ZEBU_Clock ZEBU_Clock

6.1.1.7 typedef struct ZEBU_Driver ZEBU_Driver

6.1.1.8 typedef struct ZEBU_Driver_SignalIterator ZEBU_Driver_SignalIterator

6.1.1.9 typedef struct ZEBU_FastHardwareState ZEBU_FastHardwareState

6.1.1.10 typedef struct ZEBU_Filter ZEBU_Filter

6.1.1.11 typedef struct ZEBU_FlexibleLocalProbeFile ZEBU_FlexibleLocalProbeFile

6.1.1.12 typedef struct ZEBU_LocalTraceDumper ZEBU_LocalTraceDumper

6.1.1.13 typedef struct ZEBU_LocalTraceDumperGroup ZEBU_LocalTraceDumperGroup

6.1.1.14 typedef struct ZEBU_LocalTraceImporter ZEBU_LocalTraceImporter

6.1.1.15 typedef struct ZEBU_LocalTraceImporterGroup ZEBU_LocalTraceImporterGroup

6.1.1.16 typedef struct ZEBU_LocalTraceReader ZEBU_LocalTraceReader

6.1.1.17 typedef struct ZEBU_LocalTraceReaderGroup ZEBU_LocalTraceReaderGroup

6.1.1.18 typedef struct ZEBU_LogicAnalyzer ZEBU_LogicAnalyzer

6.1.1.19 typedef struct ZEBU_Memory ZEBU_Memory

6.1.1.20 typedef struct ZEBU_PartMemoryBuilder ZEBU_PartMemoryBuilder

6.1.1.21 typedef struct ZEBU_Port ZEBU_Port

6.1.1.22 typedef struct ZEBU_Signal ZEBU_Signal

6.2 ZEBU_Board.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- const char ∗ ZEBU_getPlatformName ()

  return the name of the platform for which is designed the product.

- const char ∗ ZEBU_getVersion ()

  return the product version.

- const char ∗ ZEBU_getLibraryName ()

  return the name of the used library.

- ZEBU_Board ∗ ZEBU_open (const char ∗zebuWorkPath, const char ∗designFile, const char ∗processName)

  open ZeBu session

- int ZEBU_setMsgVerboseMode (ZEBU_Board ∗, int verbose)
- ZEBU_Board ∗ ZEBU_restoreHardwareState (const char ∗filename, const char ∗zebuWorkPath, const char ∗designFile, const char ∗processName)

  restore fastly the hardware state of a ZeBu session on a hardware platform.

- ZEBU_Board ∗ ZEBU_restore (const char ∗filename, const char ∗zebuWorkPath, const char ∗designFile, const char ∗processName)
- unsigned int ZEBU_Board_destroy (ZEBU_Board ∗board)

  destroy the board

- unsigned int ZEBU_Board_restoreLogicState (ZEBU_Board ∗board, const char ∗logicStateFile, const ZEBU_Filter ∗filter, const char ∗initMemFile, unsigned int severity)

  restore the logic state of a ZeBu session on a any type and configuration platform.

- unsigned int ZEBU_Board_init (ZEBU_Board ∗board, const char ∗initMemFile)

  initialize ZeBu board

- unsigned int ZEBU_Board_init_and_restore (ZEBU_Board ∗board, const char ∗logicStateFile, const char ∗initMemFile, unsigned int severity)

  initialize ZeBu board

- unsigned int ZEBU_Board_isHDP (const ZEBU_Board ∗board, int ∗isHDP)

  test if the board is a Hardware Development Platform

- unsigned int ZEBU_Board_close (ZEBU_Board ∗board, int dummy, const char ∗message)

  free resources for ZEBU board. Free ressources for ZEBU board and print `message`

- unsigned int ZEBU_Board_closeThread (ZEBU_Board ∗board)

  close ZeBu thread, release locks acquired by the current thread.

- const char ∗ ZEBU_Board_getZebuWorkPath (const ZEBU_Board ∗board)

  get the <zebu.work> path set at opening or at restore

- int ZEBU_Board_check (ZEBU_Board ∗board, int verbose)

  check zebu initialization. Check zebu initialization. Return false if problem encountered. If verbose is different from 0 display problem.

- int ZEBU_Board_checkRACC (ZEBU_Board ∗board, int verbose)

  perform RACC: Runtime Asynchrous Communication Check. This method is provided for software compatibility with previous ZeBu series but is not actually supported by Zebu-Server. Return false if problem encountered. If verbose is different from 0 display problem.

- unsigned int ZEBU_Board_isClockSystemEnabled (ZEBU_Board ∗board)

  test if system clock is enabled

- unsigned int ZEBU_Board_waitClockSystemEnable (ZEBU_Board ∗board)

  wait until clock system is enabled

- ZEBU_Clock ∗ ZEBU_Board_createUserClock (ZEBU_Board ∗board, const char ∗name, const char ∗filename)

  create a clock handler

- ZEBU_Clock ∗ ZEBU_Board_getClock (ZEBU_Board ∗board, const char ∗name)

  create a clock handler

- void ZEBU_Board_getDriverClockFrequency (ZEBU_Board ∗board, unsigned int ∗frequency)
- ZEBU_Driver ∗ ZEBU_Board_getDriver (ZEBU_Board ∗board, const char ∗name)

  get a driver handler

- ZEBU_Memory ∗ ZEBU_Board_getMemory (ZEBU_Board ∗board, const char ∗name)

  get a memory handler

- ZEBU_Signal ∗ ZEBU_Board_getSignal (ZEBU_Board ∗board, const char ∗name)

  get a signal handler

- ZEBU_Trigger ∗ ZEBU_Board_getTrigger (ZEBU_Board ∗board, const char ∗name)

  get a trigger handler

- ZEBU_LogicAnalyzer ∗ ZEBU_Board_getLogicAnaLyzer (ZEBU_Board ∗board)

  get the logic analyzer handler

- void ZEBU_Board_getTriggerNameList (ZEBU_Board ∗board, unsigned int ∗nbTrigger, char ∗∗∗triggerNameList)

  get an array containing the trigger names

- void ZEBU_Board_getClockGroupNameList (ZEBU_Board ∗board, unsigned int ∗nbClockGroup, char ∗∗∗clockGroupNameList)

  get an array containing the clock group names

- void ZEBU_Board_getClockNameList (ZEBU_Board ∗board, const char ∗groupName, unsigned int ∗nbClock, char ∗∗∗clockNameList)

  get an array containing the trigger names

- unsigned int ZEBU_Board_run (ZEBU_Board ∗board)

  run a cycle for each driver which registered a callback

- unsigned int ZEBU_Board_register_Driver (ZEBU_Board ∗board, ZEBU_-Driver ∗driver, void(∗callback)(void ∗), void ∗data)

  associate a callback to a driver

- unsigned int ZEBU_Board_dumpfile (ZEBU_Board ∗board, const char ∗filename, unsigned int compressionLevel)

  specify the name of a waveform file

- unsigned int ZEBU_Board_dumpflush (ZEBU_Board ∗board)

  fush the content of the waveform file open from ZEBU_Board_dumpfile

- unsigned int ZEBU_Board_dumpclosefile (ZEBU_Board ∗board) *close the waveform file open from ZEBU_Board_dumpfile*

- void ZEBU_Board_closeDumpfile (ZEBU_Board ∗board)
- unsigned int ZEBU_Board_dumpvars (ZEBU_Board ∗board, ZEBU_Signal ∗signal)

  select internal register to dump

- unsigned int ZEBU_Board_dumpvarsdepth (ZEBU_Board ∗board, const char ∗name, int depth)

  select internal register to dump

- unsigned int ZEBU_Board_dumpon (ZEBU_Board ∗board)

  resume the dump

- unsigned int ZEBU_Board_dumpoff (ZEBU_Board ∗board)

  suspend the dump

- unsigned int ZEBU_Board_writeRegisters (ZEBU_Board ∗board)

  force dynamic probes write

- unsigned int ZEBU_Board_readRegisters (ZEBU_Board ∗board)

  force dynamic probes read

- unsigned int ZEBU_Board_saveHardwareState (ZEBU_Board ∗board, const char ∗filename)

  save the hardware state and the software state of a ZeBu session. Allows restoring fastly the state of the session on the same hardware platform by means of the fonction ::restorHardwareState

- unsigned int ZEBU_Board_saveHardwareState2 (ZEBU_Board ∗board, const char ∗filename, const ZEBU_Filter ∗filter)

  save the hardware state of a ZeBu session. Allows restoring fastly the state of the session on the same hardware platform by means of the fonction ::restorHardware-State

- unsigned int ZEBU_Board_save (ZEBU_Board ∗board, const char ∗filename)
- unsigned int ZEBU_Board_saveLogicState (ZEBU_Board ∗board, const char ∗filename, const ZEBU_Filter ∗filter)

  save the logic state of a ZeBu session, under a form indepedent on the type and the configuration of platform. Allows restoring the state of the session on any type and configuration of platform by means of the fonction ::restorLogicState

- unsigned int ZEBU_Board_selectSignalsToRandomize (ZEBU_Board ∗board, const char ∗signalList, int invert) *select signals to randomize*

- unsigned int ZEBU_Board_selectMemoriesToRandomize (ZEBU_Board ∗board, const char ∗memoryList, int invert)

  select memories to randomize

- unsigned int ZEBU_Board_selectObjectsToRandomize (ZEBU_Board ∗board, const ZEBU_Filter ∗filter)

  select signals and memories to randomize

- unsigned int ZEBU_Board_randomize (ZEBU_Board ∗board, unsigned int seed)

  set signals and memories to pseudo random values. Set all signals and memories if no selection has been done through the methods ZEBU_Board_selectSignals-ToRandomize, ZEBU_Board_selectMemoriesToRandomize, or ZEBU_Board_select-ObjectsToRandomize

- unsigned int ZEBU_Board_randomizeSignals (ZEBU_Board ∗board, unsigned int seed)

  set signals to pseudo random values. Set all signals if no selection has been done through the methods ZEBU_Board_selectSignalsToRandomize or ZEBU_Board_-selectObjectsToRandomize

- unsigned int ZEBU_Board_randomizeMemories (ZEBU_Board ∗board, unsigned int seed)

  set memories to pseudo random values. Set all memories if no selection has been done through the methods ZEBU_Board_selectSignalsToRandomize or ZEBU_Board_-selectObjectsToRandomize

- unsigned int ZEBU_Board_serviceLoop (ZEBU_Board ∗board)
- int ZEBU_Board_serviceLoop2 (ZEBU_Board ∗board, ServiceLoopHandler g, void ∗context)

  check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.

- int ZEBU_Board_serviceLoop3 (ZEBU_Board ∗board, ServiceLoopHandler g, void ∗context, const unsigned int portGroupNumber)

  check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.

- int ZEBU_Board_serviceLoop3L (ZEBU_Board ∗board, ServiceLoopHandler g, void ∗context, const long long unsigned int portGroupNumber) *check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.*

- unsigned int ZEBU_Board_loop (ZEBU_Board ∗board)

  obselete

- ZEBU_Board_SignalIterator ∗ ZEBU_Board_createSignalIterator (ZEBU_Board ∗board)

  create an iterator on internal signals

- ZEBU_Board_SignalIterator ∗ ZEBU_Board_createSignalIterator2 (ZEBU_Board ∗board, int autoDeselect)

  create an iterator on internal signals

- void ZEBU_Board_destroySignalIterator (ZEBU_Board_SignalIterator ∗signalIterator)

  destroy an iterator on internal signals

- void ZEBU_Board_SignalIterator_goToFirst (ZEBU_Board_SignalIterator ∗signalIterator)

  move iterator to first signal

- void ZEBU_Board_SignalIterator_goToNext (ZEBU_Board_SignalIterator ∗signalIterator)

  move iterator to next signal

- int ZEBU_Board_SignalIterator_isAtEnd (ZEBU_Board_SignalIterator ∗signalIterator)

  test if iterator passed last signal

- ZEBU_Signal ∗ ZEBU_Board_SignalIterator_getSignal (ZEBU_Board_SignalIterator ∗signalIterator)

  return the current signal

- ZEBU_Board_MemoryIterator ∗ ZEBU_Board_createMemoryIterator (ZEBU_Board ∗board)

  create an iterator on memories

- void ZEBU_Board_destroyMemoryIterator (ZEBU_Board_MemoryIterator ∗memoryIterator)

  destroy an iterator on memories

- void ZEBU_Board_MemoryIterator_goToFirst (ZEBU_Board_MemoryIterator ∗memoryIterator)

  move iterator to first memory

- void ZEBU_Board_MemoryIterator_goToNext (ZEBU_Board_MemoryIterator ∗memoryIterator)

  move iterator to next memory

- int ZEBU_Board_MemoryIterator_isAtEnd (ZEBU_Board_MemoryIterator ∗memoryIterator)

  test if iterator passed last memory

- ZEBU_Memory ∗ ZEBU_Board_MemoryIterator_getMemory (ZEBU_Board_-MemoryIterator ∗memoryIterator)

  return the current memory

- ZEBU_Board_DriverInfoIterator ∗ ZEBU_Board_createDriverInfoIterator (ZEBU_Board ∗board)

  create an iterator on driver information

- void ZEBU_Board_destroyDriverInfoIterator (ZEBU_Board_DriverInfoIterator ∗driverInfoIterator)

  destroy an iterator on driver information

- void ZEBU_Board_DriverInfoIterator_goToFirst (ZEBU_Board_DriverInfo-Iterator ∗driverInfoIterator)

  move iterator to first driver

- void ZEBU_Board_DriverInfoIterator_goToNext (ZEBU_Board_DriverInfo-Iterator ∗driverInfoIterator)

  move iterator to next driver

- int ZEBU_Board_DriverInfoIterator_isAtEnd (ZEBU_Board_DriverInfoIterator ∗driverInfoIterator)

  test if iterator passed last driver

- const char ∗ ZEBU_Board_DriverInfoIterator_getModelName (ZEBU_Board_-DriverInfoIterator ∗driverInfoIterator)

  return the current driver model name

- const char ∗ ZEBU_Board_DriverInfoIterator_getInstanceName (ZEBU_-Board_DriverInfoIterator ∗driverInfoIterator)

  return the current driver instance name

- const char ∗ ZEBU_Board_DriverInfoIterator_getProcessName (ZEBU_-Board_DriverInfoIterator ∗driverInfoIterator)

  return the process name from which the current driver is accesssible

- ZEBU_Driver_Type ZEBU_Board_DriverInfoIterator_getType (ZEBU_-Board_DriverInfoIterator ∗driverInfoIterator)

  return the current driver type

- ZEBU_Board_PortInfoIterator ∗ ZEBU_Board_DriverInfoIterator_getPortInfo-Iterator (ZEBU_Board_DriverInfoIterator ∗driverInfoIterator)

  return an iterator on port information of the current driver model name

- void ZEBU_Board_PortInfoIterator_goToFirst (ZEBU_Board_PortInfoIterator ∗portInfoIterator)

  move iterator to first port

- void ZEBU_Board_PortInfoIterator_goToNext (ZEBU_Board_PortInfoIterator ∗portInfoIterator)

  move iterator to next port

- int ZEBU_Board_PortInfoIterator_isAtEnd (ZEBU_Board_PortInfoIterator ∗portInfoIterator)

  test if iterator passed last port

- const char ∗ ZEBU_Board_PortInfoIterator_getPortName (ZEBU_Board_Port-InfoIterator ∗portInfoIterator)

  return the current port instance name

- unsigned int ZEBU_Board_PortInfoIterator_getMessageSize (ZEBU_Board_-PortInfoIterator ∗portInfoIterator)

  return the message size in number of 32-bit words of the current port

- ZEBU_Port_Direction ZEBU_Board_PortInfoIterator_getDirection (ZEBU_-Board_PortInfoIterator ∗portInfoIterator)

  return the current port direction

- int ZEBU_Board_PortInfoIterator_isConnected (ZEBU_Board_PortInfoIterator ∗portInfoIterator)

  test if the current port is connected

6.2.1 Function Documentation

6.2.1.1 int ZEBU_Board_check (ZEBU_Board ∗ *board*, int *verbose*)

check zebu initialization. Check zebu initialization. Return false if problem encountered. If verbose is different from 0 display problem.

Parameters:

board handler to a `ZEBU_Board`

verbose verbose mode true (!=0) or false(0)

Returns:

int

Return values:

0 failed

!=0 successfull

See also:

ZEBU_open
ZEBU_Board_init

6.2.1.2 int ZEBU_Board_checkRACC (ZEBU_Board ∗ *board*, int *verbose*)

perform RACC: Runtime Asynchrous Communication Check. This method is provided for software compatibility with previous ZeBu series but is not actually supported by Zebu-Server. Return false if problem encountered. If verbose is different from 0 display problem.

Parameters:

board handler to a `ZEBU_Board`

verbose verbose mode true (!=0) or false(0)

Returns:

int

Return values:

0 failed

!=0 successfull

See also:

ZEBU_open
ZEBU_Board_init

6.2.1.3 unsigned int ZEBU_Board_close (ZEBU_Board ∗ *board*, int *dummy*, const char ∗ *message*)

free resources for ZEBU board. Free ressources for ZEBU board and print `message`

Parameters:

board handler to a `ZEBU_Board`

dummy unused. Only for backward compatibility.

message printed message before program exit

See also:

ZEBU_open

6.2.1.4 void ZEBU_Board_closeDumpfile (ZEBU_Board ∗ *board*)

6.2.1.5 unsigned int ZEBU_Board_closeThread (ZEBU_Board ∗ *board*)

close ZeBu thread, release locks acquired by the current thread.

Parameters:

board handler to a `ZEBU_Board`

6.2.1.6 ZEBU_Board_DriverInfoIterator ZEBU_Board_createDriverInfoIterator (ZEBU_Board ∗ *board*)

create an iterator on driver information

Parameters:

board handler to a `ZEBU_Board`

Returns:

ZEBU_Board_DriverInfoIterator∗

Return values:

handler on the iterator. 0 if open failed.

```
ZEBU_Board_DriverInfoIterator *iterator = ZEBU_Board_createDriverInfoIterator(boar
if (iterator == 0) {
    printf("Cannot create driver info iterator\n");
    exit(1);
}
for (ZEBU_Board_DriverInfoIterator_goToFirst(iterator);
    !ZEBU_Board_DriverInfoIterator_isAtEnd(iterator);
    ZEBU_Board_DriverInfoIterator_goToNext(iterator)
```

```
) {
    printf("Driver instance name = %s\n", ZEBU_Board_DriverInfoIterator_getInstanceName(iterat
}
ZEBU_Board_destroyDriverInfoIterator(iterator);
```

See also:
ZEBU_Board_open
ZEBU_Board_destroyDriverInfoIterator

6.2.1.7 ZEBU_Board_MemoryIterator ZEBU_Board_createMemoryIterator (ZEBU_Board ∗ *board*)

create an iterator on memories

Parameters:
board handler to a `ZEBU_Board`

Returns:
ZEBU_Board_MemoryIterator∗

Return values:
handler on the iterator. 0 if open failed.

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if (iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(iteratorMemo
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:
ZEBU_Board_open
ZEBU_Board_destroyMemoryIterator

6.2.1.8 ZEBU_Board_SignalIterator ZEBU_Board_createSignalIterator (ZEBU_Board ∗ *board*)

create an iterator on internal signals

Parameters:

board handler to a `ZEBU_Board`

Returns:

ZEBU_Board_SignalIterator∗

Return values:

handler on the iterator. 0 if open failed.

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if (iterator == 0) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Board_SignalIterator_goToFirst(iterator);
    !ZEBU_Board_SignalIterator_isAtEnd(iterator);
    ZEBU_Board_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(i
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_open
ZEBU_Board_destroySignalIterator

6.2.1.9 ZEBU_Board_SignalIterator ZEBU_Board_createSignalIterator2 (ZEBU_Board ∗ *board*, int *autoDeselect*)

create an iterator on internal signals

Parameters:

board handler to a `ZEBU_Board`

autoDeselect specify if signals must be automacally deselected

Returns:

ZEBU_Board_SignalIterator∗

Return values:

handler on the iterator. 0 if open failed.

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if(iterator == 0) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Board_SignalIterator_goToFirst(iterator);
    !ZEBU_Board_SignalIterator_isAtEnd(iterator);
```

```
        ZEBU_Board_SignalIterator_goToNext(iterator)) {
        ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
        ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(iteratorSigna
        printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
    }
    ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_open
ZEBU_Board_destroySignalIterator

6.2.1.10 ZEBU_Clock ∗ ZEBU_Board_createUserClock (ZEBU_Board ∗ *board*, const char ∗ *name*, const char ∗ *filename*)

create a clock handler

Parameters:

board handler to a `ZEBU_Board`

name name of the clock

filename name of the file containing clock initialization parameters

Returns:

handler to a `ZEBU_Clock`

See also:

ZEBU_Clock_counter

6.2.1.11 unsigned int ZEBU_Board_destroy (ZEBU_Board ∗ *board*)

destroy the board

Parameters:

board ZEBU_Board handler

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.2.1.12 void ZEBU_Board_destroyDriverInfoIterator (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

destroy an iterator on driver information

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

```
ZEBU_Board_DriverInfoIterator *iterator = ZEBU_Board_createDriverInfoIterator(boar
if (iterator == 0) {
    printf("Cannot create driver information iterator\n");
    exit(1);
}
for (ZEBU_Board_DriverInfoIterator_goToFirst(iterator);
    !ZEBU_Board_DriverInfoIterator_isAtEnd(iterator);
    ZEBU_Board_DriverInfoIterator_goToNext(iterator)
) {
    printf("Driver instance name = %s\n", ZEBU_Board_DriverInfoIterator_getInstanc
}
ZEBU_Board_destroyDriverInfoIterator(iterator);
```

See also:

ZEBU_Board_createDriverInfoIterator

6.2.1.13 void ZEBU_Board_destroyMemoryIterator (ZEBU_Board_MemoryIterator ∗ *memoryIterator*)

destroy an iterator on memories

Parameters:

memoryIterator handler to a `ZEBU_Board_MemoryIterator`

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if(iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(i
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:

ZEBU_Board_createMemoryIterator

6.2.1.14 void ZEBU_Board_destroySignalIterator (ZEBU_Board_SignalIterator * *signalIterator*)

destroy an iterator on internal signals

Parameters:

signalIterator handler to a `ZEBU_Board_SignalIterator`

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if(iterator == 0) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Board_SignalIterator_goToFirst(iterator);
    !ZEBU_Board_SignalIterator_isAtEnd(iterator);
    ZEBU_Board_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(iteratorSigna
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_createSignalIterator

6.2.1.15 const char * ZEBU_Board_DriverInfoIterator_getInstanceName (ZEBU_Board_DriverInfoIterator * *driverInfoIterator*)

return the current driver instance name

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

6.2.1.16 const char * ZEBU_Board_DriverInfoIterator_getModelName (ZEBU_Board_DriverInfoIterator * *driverInfoIterator*)

return the current driver model name

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

6.2.1.17 ZEBU_Board_PortInfoIterator * ZEBU_Board_DriverInfoIterator_getPortInfoIterator (ZEBU_Board_DriverInfoIterator * *driverInfoIterator*)

return an iterator on port information of the current driver model name

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator` The returned value is owned by the iterator. Thus, the caller must neither modify nor free or delete the returned value. The returned pointer is valid only as the iterator exists, and as long as only constant functions are called for it.

6.2.1.18 const char ∗ ZEBU_Board_DriverInfoIterator_getProcessName (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

return the process name from which the current driver is accesssible

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

6.2.1.19 ZEBU_Driver_Type ZEBU_Board_DriverInfoIterator_getType (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

return the current driver type

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

See also:

ZEBU_ROOT/include/Types.h

6.2.1.20 void ZEBU_Board_DriverInfoIterator_goToFirst (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

move iterator to first driver

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

```
ZEBU_Board_DriverInfoIterator *iterator = ZEBU_Board_createDriverInfoIterator(boar
if (iterator == 0) {
    printf("Cannot create driver information iterator\n");
    exit(1);
}
for (ZEBU_Board_DriverInfoIterator_goToFirst(iterator);
    !ZEBU_Board_DriverInfoIterator_isAtEnd(iterator);
    ZEBU_Board_DriverInfoIterator_goToNext(iterator)
) {
    printf("Driver instance name = %s\n", ZEBU_Board_DriverInfoIterator_getInstanc
}
ZEBU_Board_destroyDriverInfoIterator(iterator);
```

See also:

ZEBU_Board_createDriverInfoIterator
ZEBU_Board_DriverInfoIterator_goToNext
ZEBU_Board_DriverInfoIterator_isAtEnd

6.2.1.21 void ZEBU_Board_DriverInfoIterator_goToNext (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

move iterator to next driver

Parameters:

driverInfoIterator handler to a `ZEBU_Board_destroyDriverInfoIterator`

```
ZEBU_Board_DriverInfoIterator *iterator = ZEBU_Board_createDriverInfoIterator(board);
if (iterator == 0) {
    printf("Cannot create driver information iterator\n");
    exit(1);
}
for (ZEBU_Board_DriverInfoIterator_goToFirst(iterator);
    !ZEBU_Board_DriverInfoIterator_isAtEnd(iterator);
    ZEBU_Board_DriverInfoIterator_goToNext(iterator)
) {
    printf("Driver instance name = %s\n", ZEBU_Board_DriverInfoIterator_getInstanceName(iterat
}
ZEBU_Board_destroyDriverInfoIterator(iterator);
```

See also:

ZEBU_Board_createDriverInfoIterator
ZEBU_Board_DriverInfoIterator_goToFirst
ZEBU_Board_DriverInfoIterator_isAtEnd

6.2.1.22 int ZEBU_Board_DriverInfoIterator_isAtEnd (ZEBU_Board_DriverInfoIterator ∗ *driverInfoIterator*)

test if iterator passed last driver

Parameters:

driverInfoIterator handler to a `ZEBU_Board_DriverInfoIterator`

Return values:

1 if at end else 0

```
ZEBU_Board_DriverInfoIterator *iterator = ZEBU_Board_createDriverInfoIterator(board);
if (iterator == 0) {
    printf("Cannot create driver information iterator\n");
    exit(1);
```

```
            }
            for (ZEBU_Board_DriverInfoIterator_goToFirst(iterator);
                !ZEBU_Board_DriverInfoIterator_isAtEnd(iterator);
                ZEBU_Board_DriverInfoIterator_goToNext(iterator)
            ) {
                printf("Driver instance name = %s\n", ZEBU_Board_DriverInfoIterator_getInstanc
            }
            ZEBU_Board_destroyDriverInfoIterator(iterator);
```

See also:

ZEBU_Board_createDriverInfoIterator
ZEBU_Board_DriverInfoIterator_goToFirst
ZEBU_Board_DriverInfoIterator_goToNext

6.2.1.23 unsigned int ZEBU_Board_dumpclosefile (ZEBU_Board ∗ *board*)

close the waveform file open from ZEBU_Board_dumpfile

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_dumpfile

6.2.1.24 unsigned int ZEBU_Board_dumpfile (ZEBU_Board ∗ *board*, const char ∗ *filename*, unsigned int *compressionLevel*)

specify the name of a waveform file

Parameters:

board handler to a `ZEBU_Board`

filename name of the waveform file

- if extension is ".bin", file is dumped in a proprietary binary format
- if extension is ".vcd", file is dumped in VCD format
- if extension is ".fsdb", file is dumped in FSDB format

compressionLevel compression level. Takes value between 0 and 9. 0 is fastest, and 9 is best. Default 0.

See also:

ZEBU_Board_dumpfile
ZEBU_Board_dumpvars
ZEBU_Board_dumpon
ZEBU_Board_dumpoff
ZEBU_open

6.2.1.25 unsigned int ZEBU_Board_dumpflush (ZEBU_Board ∗ *board*)

fush the content of the waveform file open from ZEBU_Board_dumpfile

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_dumpfile

6.2.1.26 unsigned int ZEBU_Board_dumpoff (ZEBU_Board ∗ *board*)

suspend the dump

Parameters:

board handler to a `ZEBU_Board`

switch partial readback waveform dump off. This is default.

See also:

ZEBU_Board_dumpvars
ZEBU_Board_dumpfile
ZEBU_Board_dumpon
ZEBU_open

6.2.1.27 unsigned int ZEBU_Board_dumpon (ZEBU_Board ∗ *board*)

resume the dump

Parameters:

board handler to a `ZEBU_Board`

switch partial readback waveform dump on

See also:

ZEBU_Board_dumpvars
ZEBU_Board_dumpfile
ZEBU_Board_dumpoff
ZEBU_open

6.2.1.28 unsigned int ZEBU_Board_dumpvars (ZEBU_Board ∗ *board*, ZEBU_Signal ∗ *signal*)

select internal register to dump

Parameters:

board handler to a `ZEBU_Board`

signal handler to the signal to be dumped. If no parameter is given, or `0`, all signals marked 'selected' into the DB are dumped.

Note:

no signal can be added after first run.

See also:

ZEBU_Board_dumpfile
ZEBU_Board_dumpon
ZEBU_Board_dumpoff
ZEBU_Board_dumpvarsdepth
ZEBU_open

6.2.1.29 unsigned int ZEBU_Board_dumpvarsdepth (ZEBU_Board ∗ *board*, const char ∗ *name*, int *depth*)

select internal register to dump

Note:

not supported in zTide environment

Parameters:

board handler to a `ZEBU_Board`

name path to an internal instance or signal. If no parameter is given, or `NULL`, all signals marked 'selected' into the DB are dumped.

depth number of hierarchy level to dump.

Note:

no signal can be added after first run.

See also:

ZEBU_Board_dumpfile
ZEBU_Board_dumpon
ZEBU_Board_dumpoff
ZEBU_Board_dumpvars
ZEBU_open

6.2.1.30 ZEBU_Clock ∗ ZEBU_Board_getClock (ZEBU_Board ∗ *board*, const char ∗ *name*)

create a clock handler

Parameters:

board handler to a `ZEBU_Board`

name name of the clock

Returns:

handler to a `ZEBU_Clock`

See also:

ZEBU_Clock_counter

6.2.1.31 void ZEBU_Board_getClockGroupNameList (ZEBU_Board ∗ *board*, unsigned int ∗ *nbClockGroup*, char ∗∗∗ *clockGroupNameList*)

get an array containing the clock group names

Parameters:

board handler to a `ZEBU_Board`

nbClockGroup reference to the number of clock groups.

clockGroupNameList reference an array of clock groups names

See also:

ZEBU_Board_getTrigger

6.2.1.32 void ZEBU_Board_getClockNameList (ZEBU_Board ∗ *board*, const char ∗ *groupName*, unsigned int ∗ *nbClock*, char ∗∗∗ *clockNameList*)

get an array containing the trigger names

Parameters:

board handler to a `ZEBU_Board`

groupName name of the clock group

nbClock reference to the number of clocks.

clockNameList reference an array of clocks names

See also:

ZEBU_Board_getTrigger

6.2.1.33 ZEBU_Driver ∗ ZEBU_Board_getDriver (ZEBU_Board ∗ *board*, const char ∗ *name*)

get a driver handler

Parameters:

board handler to a `ZEBU_Board`

name driver's name as declared in dve file.

See also:

ZEBU_Board_getClock
ZEBU_Board_createUserClock
ZEBU_Board_getMemory
ZEBU_Board_getSignal
ZEBU_open

6.2.1.34 void ZEBU_Board_getDriverClockFrequency (ZEBU_Board ∗ *board*, unsigned int ∗ *frequency*)

6.2.1.35 ZEBU_LogicAnalyzer ∗ ZEBU_Board_getLogicAnaLyzer (ZEBU_Board ∗ *board*)

get the logic analyzer handler

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_getClock
ZEBU_Board_createUserClock
ZEBU_Board_getDriver
ZEBU_Board_getSignal
ZEBU_open

6.2.1.36 ZEBU_Memory ∗ ZEBU_Board_getMemory (ZEBU_Board ∗ *board*, const char ∗ *name*)

get a memory handler

Parameters:

board handler to a `ZEBU_Board`

name memory's name.

See also:
ZEBU_Board_getClock
ZEBU_Board_createUserClock
ZEBU_Board_getDriver
ZEBU_Board_getSignal
ZEBU_open

6.2.1.37 ZEBU_Signal ∗ ZEBU_Board_getSignal (ZEBU_Board ∗ *board*, const char ∗ *name*)

get a signal handler

Parameters:
board handler to a `ZEBU_Board`

name signal's name.

See also:
ZEBU_Board_getClock
ZEBU_Board_createUserClock
ZEBU_Board_getDriver
ZEBU_Board_getSignal
ZEBU_open

6.2.1.38 ZEBU_Trigger ∗ ZEBU_Board_getTrigger (ZEBU_Board ∗ *board*, const char ∗ *name*)

get a trigger handler

Parameters:
board handler to a `ZEBU_Board`

name trigger's name.

See also:
ZEBU_Board_getClock
ZEBU_Board_createUserClock
ZEBU_Board_getDriver
ZEBU_Board_getSignal
ZEBU_open

6.2.1.39 void ZEBU_Board_getTriggerNameList (ZEBU_Board ∗ *board*, unsigned int ∗ *nbTrigger*, char ∗∗∗ *triggerNameList*)

get an array containing the trigger names

Parameters:

board handler to a `ZEBU_Board`

nbTrigger reference to the number of triggers.

triggerNameList reference an array of trigger names

See also:

ZEBU_Board_getTrigger

6.2.1.40 const char ∗ ZEBU_Board_getZebuWorkPath (const ZEBU_Board ∗ *board*)

get the <zebu.work> path set at opening or at restore

Parameters:

board handler to a `ZEBU_Board`

Returns:

const char ∗

Return values:

NULL failed

See also:

Board::open
Board::restore

6.2.1.41 unsigned int ZEBU_Board_init (ZEBU_Board ∗ *board*, const char ∗ *initMemFile*)

initialize ZeBu board

Parameters:

board handler to a `ZEBU_Board`

initMemFile File name. This file give list of memories to initialize with corresponding memory filenames. Use 0 if no memory init file to use.

See also:

ZEBU_open

6.2.1.42 unsigned int ZEBU_Board_init_and_restore (ZEBU_Board $*$ *board*, const char $*$ *logicStateFile*, const char $*$ *initMemFile*, unsigned int *severity*)

initialize ZeBu board

Parameters:

board handler to a `ZEBU_Board`

logicStateFile name of the file of the logic state to restore before the starting of clocks Use 0 if no logic state file to use.

initMemFile File name. This file give list of memories to initialize with corresponding memory filenames. Use 0 if no memory init file to use.

severity the lower the severity is, the more defects are interpreted as errors

See also:

ZEBU_open

6.2.1.43 unsigned int ZEBU_Board_isClockSystemEnabled (ZEBU_Board $*$ *board*)

test if system clock is enabled

Parameters:

board handler to a `ZEBU_Board`

Returns:

unsigned int

Return values:

0 false, clock system is not enabled

$>$***0*** true, clock system is enabled

See also:

ZEBU_open

6.2.1.44 bool ZEBU_Board_isHDP (const ZEBU_Board $*$ *board*, int $*$ *isHDP*)

test if the board is a Hardware Development Platform

Note:

board has to be opened to call this method

Parameters:

isHDP pointer on `integer`, set to 1 if board is a Hardware Development Platform, else set to 0

Return values:

0 if successful

6.2.1.45 unsigned int ZEBU_Board_loop (ZEBU_Board ∗ *board*)

obselete

See also:

ZEBU_Board_serviceLoop

6.2.1.46 ZEBU_Memory ∗ ZEBU_Board_MemoryIterator_getMemory (ZEBU_Board_MemoryIterator ∗ *memoryIterator*)

return the current memory

Parameters:

memoryIterator handler to a `ZEBU_Board_MemoryIterator` The returned value is owned by the iterator. Thus, the caller must neither modify nor free or delete the returned value. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Use ZEBU_Board_getMemory to get a memory from its name and keep its handler independently on the iterator status.

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if(iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(i
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:

ZEBU_Board_createMemoryIterator
ZEBU_Board_MemoryIterator_goToFirst
ZEBU_Board_MemoryIterator_goToNext
ZEBU_Board_MemoryIterator_isAtEnd
ZEBU_Board_getMemory

6.2.1.47 void ZEBU_Board_MemoryIterator_goToFirst (ZEBU_Board_MemoryIterator ∗ *memoryIterator*)

move iterator to first memory

Parameters:

memoryIterator handler to a `ZEBU_Board_MemoryIterator`

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if(iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(iteratorMemo
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:

ZEBU_Board_createMemoryIterator
ZEBU_Board_MemoryIterator_goToNext
ZEBU_Board_MemoryIterator_isAtEnd

6.2.1.48 void ZEBU_Board_MemoryIterator_goToNext (ZEBU_Board_MemoryIterator ∗ *memoryIterator*)

move iterator to next memory

Parameters:

memoryIterator handler to a `ZEBU_Board_destroyMemoryIterator`

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if(iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(iteratorMemo
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:

ZEBU_Board_createMemoryIterator

ZEBU_Board_MemoryIterator_goToFirst
ZEBU_Board_MemoryIterator_isAtEnd

6.2.1.49 int ZEBU_Board_MemoryIterator_isAtEnd (ZEBU_Board_MemoryIterator ∗ *memoryIterator*)

test if iterator passed last memory

Parameters:

memoryIterator handler to a `ZEBU_Board_MemoryIterator`

Return values:

1 if at end else 0

```
ZEBU_Board_MemoryIterator *iterator = ZEBU_Board_createMemoryIterator(board);
if(iterator == 0) {
    printf("Cannot create memory iterator\n");
    exit(1);
}
for (ZEBU_Board_MemoryIterator_goToFirst(iterator);
    !ZEBU_Board_MemoryIterator_isAtEnd(iterator);
    ZEBU_Board_MemoryIterator_goToNext(iterator)) {
    ZEBU_Memory *iteratorMemory = ZEBU_Board_MemoryIterator_getMemory(iterator);
    ZEBU_Memory *keepedMemory = ZEBU_Board_getMemory(board, ZEBU_Memory_fullname(i
    printf("Memory name = %s\n", ZEBU_Memory_fullname(keepedMemory));
}
ZEBU_Board_destroyMemoryIterator(iterator);
```

See also:

ZEBU_Board_createMemoryIterator
ZEBU_Board_MemoryIterator_goToFirst
ZEBU_Board_MemoryIterator_goToNext

6.2.1.50 ZEBU_Port_Direction ZEBU_Board_PortInfoIterator_getDirection (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

return the current port direction

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

See also:

ZEBU_ROOT/include/Types.h

6.2.1.51 unsigned int ZEBU_Board_PortInfoIterator_getMessageSize (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

return the message size in number of 32-bit words of the current port

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

6.2.1.52 const char ∗ ZEBU_Board_PortInfoIterator_getPortName (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

return the current port instance name

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

6.2.1.53 void ZEBU_Board_PortInfoIterator_goToFirst (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

move iterator to first port

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

```
for (ZEBU_Board_PortInfoIterator_goToFirst(iterator);
    !ZEBU_Board_PortInfoIterator_isAtEnd(iterator);
    ZEBU_Board_PortInfoIterator_goToNext(iterator)
) {
    printf("Port instance name = %s\n", ZEBU_Board_PortInfoIterator_getPortName(iterator));
}
ZEBU_Board_destroyPortInfoIterator(iterator);
```

See also:

ZEBU_Board_PortInfoIterator_goToNext
ZEBU_Board_PortInfoIterator_isAtEnd

6.2.1.54 void ZEBU_Board_PortInfoIterator_goToNext (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

move iterator to next port

Parameters:

portInfoIterator handler to a `ZEBU_Board_destroyPortInfoIterator`

```
for (ZEBU_Board_PortInfoIterator_goToFirst(iterator);
    !ZEBU_Board_PortInfoIterator_isAtEnd(iterator);
    ZEBU_Board_PortInfoIterator_goToNext(iterator)
) {
    printf("Driver instance name = %s\n", ZEBU_Board_PortInfoIterator_getInstanceN
}
ZEBU_Board_destroyPortInfoIterator(iterator);
```

See also:

ZEBU_Board_PortInfoIterator_goToFirst
ZEBU_Board_PortInfoIterator_isAtEnd

6.2.1.55 int ZEBU_Board_PortInfoIterator_isAtEnd (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

test if iterator passed last port

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

Return values:

1 if at end else 0

```
for (ZEBU_Board_PortInfoIterator_goToFirst(iterator);
    !ZEBU_Board_PortInfoIterator_isAtEnd(iterator);
    ZEBU_Board_PortInfoIterator_goToNext(iterator)
) {
    printf("Driver instance name = %s\n", ZEBU_Board_PortInfoIterator_getInstanceN
}
ZEBU_Board_destroyPortInfoIterator(iterator);
```

See also:

ZEBU_Board_createPortInfoIterator
ZEBU_Board_PortInfoIterator_goToFirst
ZEBU_Board_PortInfoIterator_goToNext

6.2.1.56 int ZEBU_Board_PortInfoIterator_isConnected (ZEBU_Board_PortInfoIterator ∗ *portInfoIterator*)

test if the current port is connected

Parameters:

portInfoIterator handler to a `ZEBU_Board_PortInfoIterator`

Return values:

1 if the port is connected else 0

6.2.1.57 unsigned int ZEBU_Board_randomize (ZEBU_Board ∗ *board*, unsigned int *seed*)

set signals and memories to pseudo random values. Set all signals and memories if no selection has been done through the methods ZEBU_Board_selectSignalsToRandomize, ZEBU_Board_selectMemoriesToRandomize, or ZEBU_Board_selectObjectsToRandomize

Parameters:

board handler to a `ZEBU_Board`

seed seed of the sequence of values to set

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_randomize(board, 11) != 0) {
    printf("Cannot randomize signals and memories\n");
    exit(1);
}
```

6.2.1.58 unsigned int ZEBU_Board_randomizeMemories (ZEBU_Board ∗ *board*, unsigned int *seed*)

set memories to pseudo random values. Set all memories if no selection has been done through the methods ZEBU_Board_selectSignalsToRandomize or ZEBU_Board_selectObjectsToRandomize

Parameters:

board handler to a `ZEBU_Board`

seed seed of the sequence of values to set

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_randomizeMemories(board, 11) != 0) {
    printf("Cannot randomize memories\n");
    exit(1);
}
```

6.2.1.59 unsigned int ZEBU_Board_randomizeSignals (ZEBU_Board ∗ *board*, unsigned int *seed*)

set signals to pseudo random values. Set all signals if no selection has been done through the methods ZEBU_Board_selectSignalsToRandomize or ZEBU_Board_-selectObjectsToRandomize

Parameters:

board handler to a `ZEBU_Board`

seed seed of the sequence of values to set

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_randomizeSignals(board, 11) != 0) {
    printf("Cannot randomize signals\n");
    exit(1);
}
```

6.2.1.60 unsigned int ZEBU_Board_readRegisters (ZEBU_Board ∗ *board*)

force dynamic probes read

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_readRegisters
ZEBU_open

6.2.1.61 unsigned int ZEBU_Board_register_Driver (ZEBU_Board ∗ *board*, ZEBU_Driver ∗ *driver*, void(∗)(void ∗) *callback*, void ∗ *data*)

associate a callback to a driver

Parameters:

board handler to a `ZEBU_Board`

driver handler to a `ZEBU_Driver`

callback pointer to the callback function

data parameters of the callback function

See also:

ZEBU_Board_run

6.2.1.62 unsigned int ZEBU_Board_restoreLogicState (ZEBU_Board ∗ *board*, const char ∗ *logicStateFile*, const ZEBU_Filter ∗ *filter*, const char ∗ *initMemFile*, unsigned int *severity*)

restore the logic state of a ZeBu session on a any type and configuration platform.

Note:

the state must have been saved under the form of a logic state by means of the function ZEBU_SaveLogicState. You can convert a hardware state into a logic state by means of the libary libZebuRestore.

Parameters:

board handler to a `ZEBU_Board`

logicStateFile name of the file of the logic state to restore

filter allow to filter the types of components of to restore: internal signals, driver signals, internal and external memories, clocks ... Use 0 if no filter to use.

initMemFile File name. This file give list of memories to initialize with file corresponding file. Use 0 if no memory init file to use.

severity the lower the severity is, the more defects are interpreted as errors

```
Board *zebuBoard = ZEBU_open("./zebu.work", "designFeature", "default_process");
if(zebuBoard == 0) {
    printf("Cannot open Board\n");
    exit(1);
}

if(ZEBU_Board_init(zebuBoard, 0) != 0) {
     printf("Cannot initializes Board\n");
    exit(1);
}

...

if(ZEBU_Board_saveLogicState(zebuBoard, "zebu.logic.state", 0) != 0) {
    printf("Cannot save the static state\n");
    exit(1);
}


if(ZEBU_Board_restoreLogicState(zebuBoard, "zebu.logic.state", 0) != 0) {
    printf("Cannot restore the static state\n");
    exit(1);
}
```

See also:

ZEBU_open
ZEBU_SaveLogicState
ZEBU_Board_init
ZEBU_Board_close
ZEBU_Board_check

6.2.1.63 unsigned int ZEBU_Board_run (ZEBU_Board ∗ *board*)

run a cycle for each driver which registered a callback

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_register_Driver

6.2.1.64 unsigned int ZEBU_Board_save (ZEBU_Board ∗ *board*, const char ∗ *filename*)

See also:

ZEBU_Board_saveHardwareState

6.2.1.65 unsigned int ZEBU_Board_saveHardwareState (ZEBU_Board ∗ *board*, const char ∗ *filename*)

save the hardware state and the software state of a ZeBu session. Allows restoring fastly the state of the session on the same hardware platform by means of the fonction ::restorHardwareState

Note:

not supported in zTide environment

Parameters:

board handler to a `ZEBU_Board`

filename name of the file in which must be saved the state

See also:

ZEBU_restoreHardwareState

6.2.1.66 unsigned int ZEBU_Board_saveHardwareState2 (ZEBU_Board ∗ *board*, const char ∗ *filename*, const ZEBU_Filter ∗ *filter*)

save the hardware state of a ZeBu session. Allows restoring fastly the state of the session on the same hardware platform by means of the fonction ::restorHardwareState

Note:

not supported in zTide environment

Parameters:

board handler to a `ZEBU_Board`

filename name of the file in which must be saved the state

filter allow to filter the types of components of to save: internal signals, driver signals, internal and external memories, clocks ... Use 0 if no filter to use.

See also:

ZEBU_restoreHardwareState

6.2.1.67 unsigned int ZEBU_Board_saveLogicState (ZEBU_Board ∗ *board*, const char ∗ *filename*, const ZEBU_Filter ∗ *filter*)

save the logic state of a ZeBu session, under a form indepedent on the type and the configuration of platform. Allows restoring the state of the session on any type and configuration of platform by means of the fonction ::restorLogicState

Parameters:

board handler to a `ZEBU_Board`

filename name of the file in which must be saved the state

filter allow to filter the types of components of to save: internal signals, driver signals, internal and external memories, clocks ... Use 0 if no filter to use.

See also:

ZEBU_restoreHardwareState

6.2.1.68 unsigned int ZEBU_Board_selectMemoriesToRandomize (ZEBU_Board ∗ *board*, const char ∗ *memoryList*, int *invert*)

select memories to randomize

Parameters:

board handler to a `ZEBU_Board`

memoryList filename that specifies the list of memories to randomize or to not randomize The specified file must contain the list of hierarchical nanes of memories separated by an "end of line" character. If the filename is NULL all sequential memories are randomized

invert if 0 set specified memories else set non specified memories

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_randomizeMemories(board, "memoryList", 0) != 0) {
    printf("Cannot select memories to randomize\n");
    exit(1);
}
```

6.2.1.69 unsigned int ZEBU_Board_selectObjectsToRandomize (ZEBU_Board ∗ *board*, const ZEBU_Filter ∗ *filter*)

select signals and memories to randomize

Parameters:

board handler to a `ZEBU_Board`

filter allow to filter components of to randomize: signals, internal and external memories, clocks ...

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_selectObjectsToRandomize(board, filter) != 0) {
    printf("Cannot select signals and memories to randomize\n");
    exit(1);
}
```

6.2.1.70 unsigned int ZEBU_Board_selectSignalsToRandomize (ZEBU_Board ∗ *board*, const char ∗ *signalList*, int *invert*)

select signals to randomize

Parameters:

board handler to a `ZEBU_Board`

signalList filename that specifies the list of signals to randomize.or to not randomize The specified file must contain the list of hierarchical nanes of signals separated by an "end of line" character. If the filename is NULL all sequential signals are randomized

invert if 0 set specified signals else set non specified signals

Returns:

integer error code

Return values:

0 if successfull.

```
if (ZEBU_Board_selectSignalsToRandomize(board, "signalList", 0) != 0) {
    printf("Cannot select signals to randomize\n");
    exit(1);
}
```

6.2.1.71 unsigned int ZEBU_Board_serviceLoop (ZEBU_Board * *board*)

6.2.1.72 int ZEBU_Board_serviceLoop2 (ZEBU_Board * *board*, ServiceLoopHandler *g*, void * *context*)

check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.

It can be used in alternation with polling functions of Port.

If g is NULL, return immediately after each polling cycle. If g is non-NULL, enter into a loop of performing polling cycle and calling 'g'. When 'g' returns 0 return from the loop. When 'g' is called, an indication of whether there is at least 1 message pending will be made with the 'pending' flag. You must minimize the number of returns from the loop by means of 'g' to maximize the frequency of multi-thread applications.

Parameters:

board handler to a `ZEBU_Board`

g pending callback

context user context object pointer passed straight to the 'g' function

Return values:

>0 if some messages arrived from the hardware side or new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

0 if no messages arrived from the hardware side and no new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

<0 if error

See also:

ZEBU_Port_isPossibleToReceive
ZEBU_Port_isPossibleToSend

6.2.1.73 int ZEBU_Board_serviceLoop3 (ZEBU_Board ∗ *board*, ServiceLoopHandler *g*, void ∗ *context*, const unsigned int *portGroupNumber*)

check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.

It can be used in alternation with polling functions of Port.

If g is NULL, return immediately after each polling cycle. If g is non-NULL, enter into a loop of performing polling cycle and calling 'g'. When 'g' returns 0 return from the loop. When 'g' is called, an indication of whether there is at least 1 message pending will be made with the 'pending' flag. You must minimize the number of returns from the loop by means of 'g' to maximize the frequency of multi-thread applications.

Parameters:

board handler to a `ZEBU_Board`

g pending callback

context user context object pointer passed straight to the 'g' function

portGroupNumber identifier of the group of ports to take into account. Execute only the registered callbacks of ports that belong to the specified group

Return values:

$>$***0*** if some messages arrived from the hardware side or new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

0 if no messages arrived from the hardware side and no new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

$<$***0*** if error

See also:

ZEBU_Port_isPossibleToReceive
ZEBU_Port_isPossibleToSend
ZEBU_Port_setGroup
ZEBU_Port_WaitGroup

6.2.1.74 int ZEBU_Board_serviceLoop3L (ZEBU_Board ∗ *board*, ServiceLoopHandler *g*, void ∗ *context*, const long long unsigned int *portGroupNumber*)

check for arriving messages or messages which are pending to be sent, call port callbacks when it is possible to receive a message or if it is possible to send a message and serve port queues.

It can be used in alternation with polling functions of Port.

If g is NULL, return immediately after each polling cycle. If g is non-NULL, enter into a loop of performing polling cycle and calling 'g'. When 'g' returns 0 return from the loop. When 'g' is called, an indication of whether there is at least 1 message pending will be made with the 'pending' flag. You must minimize the number of returns from the loop by means of 'g' to maximize the frequency of multi-thread applications.

Parameters:

board handler to a `ZEBU_Board`

g pending callback

context user context object pointer passed straight to the 'g' function

portGroupNumber identifier of the group of ports to take into account. Execute only the registered callbacks of ports that belong to the specified group

Return values:

$>$***0*** if some messages arrived from the hardware side or new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

0 if no messages arrived from the hardware side and no new messages can be sent to the hardware side since the last call to ZEBU_Board_serviceLoop()

$<$***0*** if error

See also:

ZEBU_Port_isPossibleToReceive
ZEBU_Port_isPossibleToSend
ZEBU_Port_setGroup
ZEBU_Port_WaitGroup

6.2.1.75 ZEBU_Signal * ZEBU_Board_SignalIterator_getSignal (ZEBU_Board_SignalIterator * *signalIterator*)

return the current signal

Parameters:

signalIterator handler to a `ZEBU_Board_SignalIterator` The returned value is owned by the iterator. Thus, the caller must neither modify nor free or delete the returned value. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Use ZEBU_Board_getSignal to get a signal from its name and keep its handler independently on the iterator status.

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if(iterator == 0) {
    printf("Cannot create signal iterator\n");
```

```
        exit(1);
    }
    for (ZEBU_Board_SignalIterator_goToFirst(iterator);
        !ZEBU_Board_SignalIterator_isAtEnd(iterator);
        ZEBU_Board_SignalIterator_goToNext(iterator)) {
        ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
        ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(i
        printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
    }
    ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_createSignalIterator
ZEBU_Board_SignalIterator_goToFirst
ZEBU_Board_SignalIterator_goToNext
ZEBU_Board_SignalIterator_isAtEnd
ZEBU_Board_getSignal

6.2.1.76 void ZEBU_Board_SignalIterator_goToFirst (ZEBU_Board_SignalIterator ∗ *signalIterator*)

move iterator to first signal

Parameters:

signalIterator handler to a `ZEBU_Board_SignalIterator`

```
    ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
    if(iterator == 0) {
        printf("Cannot create signal iterator\n");
        exit(1);
    }
    for (ZEBU_Board_SignalIterator_goToFirst(iterator);
        !ZEBU_Board_SignalIterator_isAtEnd(iterator);
        ZEBU_Board_SignalIterator_goToNext(iterator)) {
        ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
        ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(i
        printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
    }
    ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_createSignalIterator
ZEBU_Board_SignalIterator_goToNext
ZEBU_Board_SignalIterator_isAtEnd

6.2.1.77 void ZEBU_Board_SignalIterator_goToNext (ZEBU_Board_SignalIterator ∗ *signalIterator*)

move iterator to next signal

Parameters:

signalIterator handler to a `ZEBU_Board_destroySignalIterator`

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if(iterator == 0) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Board_SignalIterator_goToFirst(iterator);
    !ZEBU_Board_SignalIterator_isAtEnd(iterator);
    ZEBU_Board_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(iteratorSigna
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_createSignalIterator
ZEBU_Board_SignalIterator_goToFirst
ZEBU_Board_SignalIterator_isAtEnd

6.2.1.78 int ZEBU_Board_SignalIterator_isAtEnd (ZEBU_Board_SignalIterator ∗ *signalIterator*)

test if iterator passed last signal

Parameters:

signalIterator handler to a `ZEBU_Board_SignalIterator`

Return values:

1 if at end else 0

```
ZEBU_Board_SignalIterator *iterator = ZEBU_Board_createSignalIterator(board);
if(iterator == 0) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Board_SignalIterator_goToFirst(iterator);
    !ZEBU_Board_SignalIterator_isAtEnd(iterator);
    ZEBU_Board_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Board_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Board_getSignal(board, ZEBU_Signal_fullname(iteratorSigna
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Board_destroySignalIterator(iterator);
```

See also:

ZEBU_Board_createSignalIterator
ZEBU_Board_SignalIterator_goToFirst
ZEBU_Board_SignalIterator_goToNext

6.2.1.79 unsigned int ZEBU_Board_waitClockSystemEnable (ZEBU_Board ∗ *board*)

wait until clock system is enabled

Parameters:

board handler to a `ZEBU_Board`

Returns:

unsigned int

Return values:

0 OK

>*0* KO

See also:

ZEBU_open

6.2.1.80 unsigned int ZEBU_Board_writeRegisters (ZEBU_Board ∗ *board*)

force dynamic probes write

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_Board_readRegisters
ZEBU_open

6.2.1.81 const char ∗ ZEBU_getLibraryName ()

return the name of the used library.

Returns:

char∗

Return values:

string "<libary name> <environment>".

6.2.1.82 const char ∗ ZEBU_getPlatformName ()

return the name of the platform for which is designed the product.

Returns:

char∗

Return values:

name of the hardware platform

6.2.1.83 const char ∗ ZEBU_getVersion ()

return the product version.

Returns:

char∗

Return values:

string "<majorNum>.<minorNum>.<patchNum>"

6.2.1.84 ZEBU_Board ∗ ZEBU_open (const char ∗ *zebuWorkPath*, const char ∗ *designFile*, const char ∗ *processName*)

open ZeBu session

Parameters:

zebuWorkPath default "./zebu.work"

designFile default "designFeatures"

processName default "default_process"

Returns:

Board∗

Return values:

handler on Board. 0 if open failed.

```
Board *zebuBoard = ZEBU_open("./zebu.work", "designFeature", "default_process");
if(zebuBoard == 0) {
    printf("Cannot open Board\n");
    exit(1);
}
```

See also:

ZEBU_Board_init
ZEBU_Board_close
ZEBU_Board_check
ZEBU_restoreHardwareState

6.2.1.85 ZEBU_Board* ZEBU_restore (const char * *filename*, const char * *zebuWorkPath*, const char * *designFile*, const char * *processName*)

6.2.1.86 ZEBU_Board * ZEBU_restoreHardwareState (const char * *filename*, const char * *zebuWorkPath*, const char * *designFile*, const char * *processName*)

restore fastly the hardware state of a ZeBu session on a hardware platform.

Note:

the state must have been saved under the form of a hardware state by means of the function ZEBU_SaveHardwareState and must be have be done on the same type and the same configuration of hardware platform on which has been saved the state. You can also convert a hardware state on any type and configuration of hardware platform into a logic state by means of the libary libZebuRestore and load it on any type of platform

Parameters:

filename name of the file in which has been saved the state to restore

zebuWorkPath default "./zebu.work"

designFile default "designFeatures"

processName default "default_process"

Returns:

Board*

Return values:

handler on Board. 0 if open failed.

```
Board *zebuBoard = ZEBU_restoreHardwareState("zebu.hardware.state", "./zebu.work",
if(zebuBoard == 0) {
    printf("Cannot restore Board\n");
    exit(1);
}
```

See also:

ZEBU_open
ZEBU_SaveHardwareState
ZEBU_Board_init
ZEBU_Board_close
ZEBU_Board_check

6.2.1.87 int ZEBU_setMsgVerboseMode (ZEBU_Board *, int *verbose*)

6.3 ZEBU_CCall.h File Reference

Include dependency graph for ZEBU_CCall.h:

This graph shows which files directly or indirectly include this file:

Functions

- int ZEBU_CCall_SelectSamplingClockGroup (struct ZEBU_Board ∗board, const char ∗clockGroupName)

  selects the clock group on which the function calls are sampled on simulation/emulation side Function calls are sampled on all posedges and negedges of all clocks of the selected group.

- int ZEBU_CCall_SelectSamplingClocks (struct ZEBU_Board ∗board, const char ∗clockExpression)

  selects the set of clocks and sensitive edges on which the function calls are sampled on simulation/emulation side

- int ZEBU_CCall_EnableSynchronization (struct ZEBU_Board ∗board)

  enables the synchronization of function calls. This ensures functions are called in an determined order respecting their execution time and their call number. Execution time is the time point at which a function is executed on the simulation/emulation side. Call number allows specifying a call order for a set of functions in the same scope. Without this synchronization, each function call is executed in an increasing time order corresponding to its executions on the simulation/emulation side; but the software side does not coordinate function calls between themsleves, some time races can occur between several function calls.

- int ZEBU_CCall_DisableSynchronization (struct ZEBU_Board ∗board)

  disables the synchronization of function calls.

- int ZEBU_CCall_SetOnEvent (struct ZEBU_Board ∗board)

  selects function calls on value change mode. This ensures functions are called only when its sampled input value change between two executions on the simulation/emulation side.

- int ZEBU_CCall_UnsetOnEvent (struct ZEBU_Board ∗board)

  deselects function calls on value change mode.

- int ZEBU_CCall_LoadDynamicLibrary (struct ZEBU_Board ∗board, const char ∗fullname)

  loads a dynamic library containing the symbols of some C functions to import

- int ZEBU_CCall_Start (struct ZEBU_Board ∗board, const char ∗scope, const char ∗importName, const int callNumber)

  enables a set of function calls.

- int ZEBU_CCall_Stop (struct ZEBU_Board ∗board, const char ∗scope, const char ∗importName, const int callNumber)

  disables a set of function calls.

- int ZEBU_CCall_Start2 (struct ZEBU_Board ∗board, const char ∗scope-Expression, const int invert, const int ignoreCase, const char hierarchical-Separator, const char ∗importName, const int callNumber)

  enables a set of function calls specified by a regular expression

- int ZEBU_CCall_Stop2 (struct ZEBU_Board ∗board, const char ∗scope-Expression, const int invert, const int ignoreCase, const char hierarchical-Separator, const char ∗importName, const int callNumber)

  disables a set of function calls specified by a regular expression

- int ZEBU_CCall_Flush (struct ZEBU_Board ∗board)

  flushed the set of enabled function calls

6.3.1 Function Documentation

6.3.1.1 int ZEBU_CCall_DisableSynchronization (struct ZEBU_Board ∗ *board*)

disables the synchronization of function calls.

Parameters:

board handler to a `ZEBU_Board`

Note:

this must be called before any function call start.

See also:

ZEBU_CCall_EnableSynchronization

6.3.1.2 int ZEBU_CCall_EnableSynchronization (struct ZEBU_Board ∗ *board*)

enables the synchronization of function calls. This ensures functions are called in an determined order respecting their execution time and their call number. Execution time is the time point at which a function is executed on the simulation/emulation side. Call number allows specifying a call order for a set of functions in the same scope. Without this synchronization, each function call is executed in an increasing time order corresponding to its executions on the simulation/emulation side; but the software side does not coordinate function calls between themsleves, some time races can occur between several function calls.

Note:

the synchronization is disabled by default.
the synchronization decreases runtime performance.
this must be called before any function call start.

See also:

ZEBU_CCall_DisableSynchronization

6.3.1.3 int ZEBU_CCall_Flush (struct ZEBU_Board ∗ *board*)

flushed the set of enabled function calls

Parameters:

board handler to a `ZEBU_Board`

6.3.1.4 int ZEBU_CCall_LoadDynamicLibrary (struct ZEBU_Board ∗ *board*, const char ∗ *fullname*)

loads a dynamic library containing the symbols of some C functions to import

Note:

this can be called several times to load several libraries

Parameters:

board handler to a `ZEBU_Board`

name name of the dynamic library to load: [<path>/]<libary name="">.so. If no path is specified, the LD_LIBRARY_PATH environment variable must contain the path where the <libary name="">.so file is located.

Note:

this must be called before function call start.

6.3.1.5 int ZEBU_CCall_SelectSamplingClockGroup (struct ZEBU_Board ∗ *board*, const char ∗ *clockGroupName*)

selects the clock group on which the function calls are sampled on simulation/emulation side Function calls are sampled on all posedges and negedges of all clocks of the selected group.

Parameters:

board handler to a `ZEBU_Board`

clockGroupName name of a controlled clock group. If NULL, the first arbitrary group is selected.

Note:

this must be called before any function call start.

6.3.1.6 int ZEBU_CCall_SelectSamplingClocks (struct ZEBU_Board ∗ *board*, const char ∗ *clockExpression*)

selects the set of clocks and sensitive edges on which the function calls are sampled on simulation/emulation side

Parameters:

board handler to a `ZEBU_Board`

clockExpression clock sensitivity expression: "[posedge|negedge] <clock name> [or [posedge|negedge] <clock name>] and so on" If NULL all clocks and all edges are selected. For instance: "posedge clock1" => sampling on clock1's posedges "posedge clock1 or negedge clock2" => sampling on clock1's posedges and clock2's negedges "clock3" => sampling on clock3's posedges and clock3's negedges

Note:

this must be called before any function call start.

6.3.1.7 int ZEBU_CCall_SetOnEvent (struct ZEBU_Board ∗ *board*)

selects function calls on value change mode. This ensures functions are called only when its sampled input value change between two executions on the simulation/emulation side.

Parameters:

board handler to a `ZEBU_Board`

Note:

functions are called for each execution on the simulation/emulation side by default. this must be called before any function call start.

See also:

ZEBU_CCall_UnsetOnEvent

6.3.1.8 int ZEBU_CCall_Start (struct ZEBU_Board ∗ *board*, const char ∗ *scope*, const char ∗ *importName*, const int *callNumber*)

enables a set of function calls.

Note:

all function calls are disabled by default.

Parameters:

- ***board*** handler to a `ZEBU_Board`
- ***scope*** scope of the function calls to enabled. If null all function calls are enabled.
- ***importName*** name of the C function of calls to enable. If null all function calls of the scope are enabled.
- ***callNumber*** the call number in the scope of the function call to enable. If -1 all function calls of the scope are enabled.

Note:

this can be executed several times to start a set of function calls.

See also:

ZEBU_CCall_Stop

6.3.1.9 int ZEBU_CCall_Start2 (struct ZEBU_Board ∗ *board*, const char ∗ *scopeExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*, const char ∗ *importName*, const int *callNumber*)

enables a set of function calls specified by a regular expression

Note:

all function calls are disabled by default.

Parameters:

- ***board*** handler to a `ZEBU_Board`
- ***scopeExpression*** regular expression specifying the scopes of the function calls to enable.

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

importName name of the C function of calls to enable. If null all function calls of the scope are enabled.

callNumber the call number in the scope of the function call to enable. If -1 all function calls of the scope are enabled.

Note:
this method can be executed several times to start a set of function calls.

See also:
ZEBU_CCall_Stop2

6.3.1.10 int ZEBU_CCall_Stop (struct ZEBU_Board ∗ *board*, const char ∗ *scope*, const char ∗ *importName*, const int *callNumber*)

disables a set of function calls.

Parameters:

board handler to a `ZEBU_Board`

scope scope of the function calls. If null all enabled function calls are disabled.

importName name of the C function of calls to disable. If null all function calls of the scope are disabled.

callNumber the call number in the scope of the function call to disable. If -1 all function calls of the scope are disabled.

Note:
this can be executed several times to stop a set of function calls.

See also:
ZEBU_CCall_Start

6.3.1.11 int ZEBU_CCall_Stop2 (struct ZEBU_Board ∗ *board*, const char ∗ *scopeExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*, const char ∗ *importName*, const int *callNumber*)

disables a set of function calls specified by a regular expression

Parameters:

board handler to a `ZEBU_Board`

scopeExpression regular expression specifying the scopes of the function calls to disable.

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

importName name of the C function of calls to disable. If null all function calls of the scope are disabled.

callNumber the call number in the scope of the function call to disable. If -1 all function calls of the scope are disabled.

Note:

this method can be executed several times to start a set of function calls.

See also:

ZEBU_CCall_Start2

6.3.1.12 int ZEBU_CCall_UnsetOnEvent (struct ZEBU_Board ∗ *board*)

deselects function calls on value change mode.

Parameters:

board handler to a `ZEBU_Board`

See also:

ZEBU_CCall_SetOnEvent

6.4 ZEBU_Clock.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned int ZEBU_Clock_enable (ZEBU_Clock ∗clock, long long unsigned int cycles)

  enable the clock

- unsigned int ZEBU_Clock_disable (ZEBU_Clock ∗clock)

  disable the clock

- unsigned int ZEBU_Clock_isEnabled (ZEBU_Clock ∗clock)

  get the status of the clock (enabled or disabled)

- unsigned int ZEBU_Clock_reset (ZEBU_Clock ∗clock)

  reset clock cycle counter

- unsigned int ZEBU_Clock_counter (ZEBU_Clock ∗clock, long long unsigned int ∗count)

  get clock cycle counter value

- void ZEBU_Clock_delete (ZEBU_Clock ∗clock)

  destroy the clock

- char ∗ ZEBU_Clock_name (ZEBU_Clock ∗clock)

  get the Clock name

6.4.1 Function Documentation

6.4.1.1 unsigned int ZEBU_Clock_counter (ZEBU_Clock ∗ *clock*, long long unsigned int ∗ *count*)

get clock cycle counter value

Parameters:

clock ZEBU_Clock handler

count return clock cycle counter value

Returns:

unsigned int

Return values:

0 if OK

positive if KO

6.4.1.2 unsigned int ZEBU_Clock_delete (ZEBU_Clock ∗ *clock*)

destroy the clock

Parameters:

clock ZEBU_Clock handler

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.4.1.3 unsigned int ZEBU_Clock_disable (ZEBU_Clock ∗ *clock*)

disable the clock

Parameters:

clock ZEBU_Clock handler

Returns:

unsigned int

Return values:

0 if OK

positive if KO

6.4.1.4 unsigned int ZEBU_Clock_enable (ZEBU_Clock ∗ *clock*, long long unsigned int *cycles*)

enable the clock

Parameters:

clock ZEBU_Clock handler

cycles number of enabled cycles. If no value is given or 0, clock is enabled permanently.

Returns:

unsigned int

Return values:

0 if OK.

positive if KO

```
if(ZEBU_Clock_enable(clock0, 10)()) {
    printf("Cannot enable clock for ever\n);
}
```

6.4.1.5 unsigned int ZEBU_Clock_isEnabled (ZEBU_Clock ∗ *clock*)

get the status of the clock (enabled or disabled)

Parameters:

clock ZEBU_Clock handler

Returns:

unsigned int

Return values:

1 if clock is enabled

0 is clock is disabled

6.4.1.6 char ∗ ZEBU_Clock_name (ZEBU_Clock ∗ *clock*)

get the Clock name

Parameters:

clock ZEBU_Clock handler

Returns:

char ∗

Return values:

NULL terminated C string containing clock's name

6.4.1.7 unsigned int ZEBU_Clock_reset (ZEBU_Clock ∗ *clock*)

reset clock cycle counter

Parameters:

clock	ZEBU_Clock handler

Returns:

unsigned int

Return values:

0	if OK
positive	if KO

6.5 ZEBU_Driver.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define ZEBU_DefaultCompression -1
- #define ZEBU_NoCompression 0
- #define ZEBU_BestSpeed 1
- #define ZEBU_BestCompression 9

Functions

- unsigned int ZEBU_Driver_connect (ZEBU_Driver ∗driver)

  connect a driver to ZeBu

- unsigned int ZEBU_Driver_disconnect (ZEBU_Driver ∗driver)

  disconnect a driver to ZeBu Disconnect a driver to ZEBU board

- unsigned int ZEBU_Driver_run (ZEBU_Driver ∗driver, unsigned int nb-Cycles)

  enable the clock of the driver for a number of cycles

- unsigned int ZEBU_Driver_run_block (ZEBU_Driver ∗driver, unsigned int nb-Cycles, int block)

  enable the clock of the driver for a number of cycles

- unsigned int ZEBU_Driver_wait (ZEBU_Driver ∗driver, unsigned int triggers, unsigned int timeout)

  wait for a trigger event. Wait for the trigger event given in parameter while running the clock.

- unsigned int ZEBU_Driver_update (ZEBU_Driver ∗driver)

  update of the driver interface

- ZEBU_Signal ∗ ZEBU_Driver_getSignal (ZEBU_Driver ∗driver, const char ∗name)

  get a signal handler

- void ZEBU_Driver_delete (ZEBU_Driver ∗driver)

  destroy the driver

- unsigned int ZEBU_Driver_dumpfile (ZEBU_Driver ∗monitor, char ∗filename, int compressionLevel)

  create a waveform file. Create a waveform file. If file name extension is "bin" dump proprietary binary format. If extension is "vcd" dump VCD format If extension is "fsdb" dump FSDB format

- unsigned int ZEBU_Driver_dumpclosefile (ZEBU_Driver ∗monitor)

  close the waveform file open from ZEBU_Driver_dumpfile

- unsigned int ZEBU_Driver_closeDumpfile (ZEBU_Driver ∗monitor)

  absolete

- unsigned int ZEBU_Driver_dumpvars (ZEBU_Driver ∗monitor, ZEBU_Signal ∗signal)

  select signals to dump. Select signals to dump. If NULL passed, all signals are dumped.

- unsigned int ZEBU_Driver_dumpvarsBySignalName (ZEBU_Driver ∗monitor, const char ∗name)

  select signals to dump. Select signals to dump. If NULL passed, all signals are dumped.

- unsigned int ZEBU_Driver_dumpon (ZEBU_Driver ∗monitor)

  switch waveform dump on

- unsigned int ZEBU_Driver_dumpoff (ZEBU_Driver ∗monitor)

  Switch waveform dump off.

- unsigned int ZEBU_Driver_TraceDumpon (ZEBU_Driver ∗monitor, const char ∗clockName, const char ∗edgeName)

  switch the trace memory dump on

- unsigned int ZEBU_Driver_setPreTrigIntRatio (ZEBU_Driver ∗monitor, unsigned int ratio)

  set the pre trigger trace memory size in percent

- unsigned int ZEBU_Driver_setPreTrigFloatRatio (ZEBU_Driver ∗monitor, float ratio)

  set the Pre trigger size in percent

- unsigned int ZEBU_Driver_setPreTrigSize (ZEBU_Driver ∗monitor, unsigned int size)

  set the Pre trigger size in number of samples

- unsigned int ZEBU_Driver_storeToFile (ZEBU_Driver ∗monitor)

  store the trace memory into the dump file

- char ∗ ZEBU_Driver_name (ZEBU_Driver ∗driver)

  get the driver's name

- void ZEBU_Driver_registerCallback (ZEBU_Driver ∗driver, void(∗callback)(void ∗), void ∗user)

  register a callback

- ZEBU_Driver_SignalIterator ∗ ZEBU_Driver_createSignalIterator (ZEBU_-Driver ∗driver)

  create an iterator on driver signals

- void ZEBU_Driver_destroySignalIterator (ZEBU_Driver_SignalIterator ∗signalIterator)

  destroy an iterator on driver signals

- void ZEBU_Driver_SignalIterator_goToFirst (ZEBU_Driver_SignalIterator ∗signalIterator)

  move iterator to first signal

- void ZEBU_Driver_SignalIterator_goToNext (ZEBU_Driver_SignalIterator ∗signalIterator)

  move iterator to next signal

- int ZEBU_Driver_SignalIterator_isAtEnd (ZEBU_Driver_SignalIterator ∗signalIterator)

  test if iterator passed last signal

- ZEBU_Signal ∗ ZEBU_Driver_SignalIterator_getSignal (ZEBU_Driver_-SignalIterator ∗signalIterator)

  return the current signal. The returned value is owned by the iterator. Thus, the caller must neither modify nor free or delete the returned value. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Use ZEBU_Driver_getSignal to get a signal from its name and keep its handler independently on the iterator status.

6.5.1 Define Documentation

6.5.1.1 #define ZEBU_BestCompression 9

6.5.1.2 #define ZEBU_BestSpeed 1

6.5.1.3 #define ZEBU_DefaultCompression -1

6.5.1.4 #define ZEBU_NoCompression 0

6.5.2 Function Documentation

6.5.2.1 unsigned int ZEBU_Driver_closeDumpfile (ZEBU_Driver ∗ *monitor*)

absolete

See also:
ZEBU_Driver_dumpclosefile

6.5.2.2 unsigned int ZEBU_Driver_connect (ZEBU_Driver ∗ *driver*)

connect a driver to ZeBu

Parameters:
driver ZEBU_Driver handler

Returns:
status

Return values:
0 OK

$>$***0*** KO

connect a driver to ZEBU board

6.5.2.3 ZEBU_Driver_SignalIterator ZEBU_Driver_createSignalIterator (ZEBU_Driver ∗ *driver*)

create an iterator on driver signals

Parameters:
driver handler to a `ZEBU_Driver`

Returns:

ZEBU_Driver_SignalIterator*

Return values:

handler on th iterator. NULL if open failed.

```
ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
if(iterator == NULL) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
    !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
    ZEBU_Driver_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(ite
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Driver_destroySignalIterator(iterator);
```

See also:

ZEBU_Driver_open
ZEBU_Driver_destroySignalIterator

6.5.2.4 unsigned int ZEBU_Driver_delete (ZEBU_Driver ∗ *driver*)

destroy the driver

Parameters:

driver ZEBU_Driver handler

Returns:

unsigned int

Return values:

0 OK

>***0*** KO

6.5.2.5 void ZEBU_Driver_destroySignalIterator (ZEBU_Driver_SignalIterator ∗ *signalIterator*)

destroy an iterator on driver signals

Parameters:

signalIterator handler to a `ZEBU_Driver_SignalIterator`

```
ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
if(iterator == NULL) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
    !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
    ZEBU_Driver_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(iteratorSignal,
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Driver_destroySignalIterator(iterator);
```

See also:
ZEBU_Driver_createSignalIterator

6.5.2.6 void ZEBU_Driver_disconnect (ZEBU_Driver ∗ *driver*)

disconnect a driver to ZeBu Disconnect a driver to ZEBU board

Parameters:
driver ZEBU_Driver handler

6.5.2.7 unsigned int ZEBU_Driver_dumpclosefile (ZEBU_Driver ∗ *monitor*)

close the waveform file open from ZEBU_Driver_dumpfile

Parameters:
monitor ZEBU_Driver handler

See also:
ZEBU_Driver_dumpfile

6.5.2.8 void ZEBU_Driver_dumpfile (ZEBU_Driver ∗ *monitor*, char ∗ *filename*, int *compressionLevel*)

create a waveform file. Create a waveform file. If file name extension is "bin" dump proprietary binary format. If extension is "vcd" dump VCD format If extension is "fsdb" dump FSDB format

Parameters:
monitor ZEBU_Driver handler
filename waveform file name
compressionLevel compression level

6.5.2.9 void ZEBU_Driver_dumpoff (ZEBU_Driver ∗ *monitor*)

Switch waveform dump off.

Parameters:

monitor ZEBU_Driver handler

6.5.2.10 void ZEBU_Driver_dumpon (ZEBU_Driver ∗ *monitor*)

switch waveform dump on

Parameters:

monitor ZEBU_Driver handler

6.5.2.11 void ZEBU_Driver_dumpvars (ZEBU_Driver ∗ *monitor*, ZEBU_Signal ∗ *signal*)

select signals to dump. Select signals to dump. If NULL passed, all signals are dumped.

Parameters:

monitor ZEBU_Driver handler

signal handler to the signal to be dumped.

Note:

no signal can be added after first run.

6.5.2.12 void ZEBU_Driver_dumpvarsBySignalName (ZEBU_Driver ∗ *monitor*, const char ∗ *name*)

select signals to dump. Select signals to dump. If NULL passed, all signals are dumped.

Parameters:

monitor ZEBU_Driver handler

name name of the signal to be dumped. If no parameter is given, or `NULL`, all signals are dumped.

Note:

no signal can be added after first run.

6.5.2.13 ZEBU_Signal ∗ ZEBU_Driver_getSignal (ZEBU_Driver ∗ *driver*, const char ∗ *name*)

get a signal handler

Parameters:

driver handler

name name of the signal. Non hierarchical name as specified in .dve file or hierarchical name relative to the top of the DUT

Returns:

ZEBU_Signal ∗

Return values:

signal handler from signal's name

6.5.2.14 char ∗ ZEBU_Driver_name (ZEBU_Driver ∗ *driver*)

get the driver's name

Parameters:

driver handler

Returns:

const char ∗

Return values:

NULL terminated C string containing driver's name

6.5.2.15 void ZEBU_Driver_registerCallback (ZEBU_Driver ∗ *driver*, void(∗)(void ∗) *callback*, void ∗ *user*)

register a callback

Parameters:

driver handler

callback callback

user user data

6.5.2.16 void ZEBU_Driver_run (ZEBU_Driver ∗ *driver*, unsigned int *nbCycles*)

enable the clock of the driver for a number of cycles

Parameters:

driver ZEBU_Driver handler

nbCycles number of cycles

run `nbCycles` on ZEBU_Driver `driver`

6.5.2.17 void ZEBU_Driver_run_block (ZEBU_Driver ∗ *driver*, unsigned int *nbCycles*, int *block*)

enable the clock of the driver for a number of cycles

Parameters:

driver ZEBU_Driver handler

nbCycles number of cycles

block if null it leads to no blocking run else to a blocking run

run `nbCycles` on ZEBU_Driver `driver`

6.5.2.18 void ZEBU_Driver_setPreTrigFloatRatio (ZEBU_Driver ∗ *monitor*, float *ratio*)

set the Pre trigger size in percent

Parameters:

monitor ZEBU_Driver handler

ratio size ratio in percent

Note:

this function has to be used with a trace memory driver

6.5.2.19 void ZEBU_Driver_setPreTrigIntRatio (ZEBU_Driver ∗ *monitor*, unsigned int *ratio*)

set the pre trigger trace memory size in percent

Parameters:

monitor ZEBU_Driver handler

ratio size ratio in percent (0.0 to 100.0)

Note:

this function has to be used with a trace memory driver

6.5.2.20 void ZEBU_Driver_setPreTrigSize (ZEBU_Driver ∗ *monitor*, unsigned int *size*)

set the Pre trigger size in number of samples

Parameters:

monitor ZEBU_Driver handler

size size in number of samples

Note:

this function has to be used with a trace memory driver

6.5.2.21 ZEBU_Signal ∗ ZEBU_Driver_SignalIterator_getSignal (ZEBU_Driver_SignalIterator ∗ *signalIterator*)

return the current signal. The returned value is owned by the iterator. Thus, the caller must neither modify nor free or delete the returned value. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Use ZEBU_Driver_getSignal to get a signal from its name and keep its handler independently on the iterator status.

Parameters:

signalIterator handler to a `ZEBU_Driver_SignalIterator`.

```
ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
if(iterator == NULL) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
    !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
    ZEBU_Driver_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(iteratorSignal
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Driver_destroySignalIterator(iterator);
```

See also:

ZEBU_Driver_createSignalIterator

ZEBU_Driver_SignalIterator_goToFirst
ZEBU_Driver_SignalIterator_goToNext
ZEBU_Driver_SignalIterator_isAtEnd
ZEBU_Driver_getSignal

6.5.2.22 void ZEBU_Driver_SignalIterator_goToFirst (ZEBU_Driver_SignalIterator ∗ *signalIterator*)

move iterator to first signal

Parameters:

signalIterator handler to a `ZEBU_Driver_SignalIterator`

```
ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
if(iterator == NULL) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
    !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
    ZEBU_Driver_SignalIterator_goToNext(iterator)) {
    ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
    ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(ite
    printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
}
ZEBU_Driver_destroySignalIterator(iterator);
```

See also:

ZEBU_Driver_createSignalIterator
ZEBU_Driver_SignalIterator_goToNext
ZEBU_Driver_SignalIterator_isAtEnd

6.5.2.23 void ZEBU_Driver_SignalIterator_goToNext (ZEBU_Driver_SignalIterator ∗ *signalIterator*)

move iterator to next signal

Parameters:

signalIterator handler to a `ZEBU_Driver_destroySignalIterator`

```
ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
if(iterator == NULL) {
    printf("Cannot create signal iterator\n");
    exit(1);
}
for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
    !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
    ZEBU_Driver_SignalIterator_goToNext(iterator)) {
        ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
        ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(iteratorSignal
        printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
    }
    ZEBU_Driver_destroySignalIterator(iterator);
```

See also:

ZEBU_Driver_createSignalIterator
ZEBU_Driver_SignalIterator_goToFirst
ZEBU_Driver_SignalIterator_isAtEnd

6.5.2.24 int ZEBU_Driver_SignalIterator_isAtEnd (ZEBU_Driver_SignalIterator ∗ *signalIterator*)

test if iterator passed last signal

Parameters:

signalIterator handler to a `ZEBU_Driver_SignalIterator`

Return values:

1 if at end else 0

```
    ZEBU_Driver_SignalIterator *iterator = ZEBU_Driver_createSignalIterator(driver);
    if(iterator == NULL) {
        printf("Cannot create signal iterator\n");
        exit(1);
    }
    for (ZEBU_Driver_SignalIterator_goToFirst(iterator);
        !ZEBU_Driver_SignalIterator_isAtEnd(iterator);
        ZEBU_Driver_SignalIterator_goToNext(iterator)) {
        ZEBU_Signal *iteratorSignal = ZEBU_Driver_SignalIterator_getSignal(iterator);
        ZEBU_Signal *keepedSignal = ZEBU_Driver_getSignal(driver, ZEBU_Signal_name(iteratorSignal
        printf("Signal name = %s\n", ZEBU_Signal_fullname(keepedSignal));
    }
    ZEBU_Driver_destroySignalIterator(iterator);
```

See also:

ZEBU_Driver_createSignalIterator
ZEBU_Driver_SignalIterator_goToFirst
ZEBU_Driver_SignalIterator_goToNext

6.5.2.25 void ZEBU_Driver_storeToFile (ZEBU_Driver ∗ *monitor*)

store the trace memory into the dump file

Parameters:

monitor ZEBU_Driver handler

Note:
this function has to be used with a trace memory driver

6.5.2.26 void ZEBU_Driver_TraceDumpon (ZEBU_Driver ∗ *monitor*, const char ∗ *clockName*, const char ∗ *edgeName*)

switch the trace memory dump on

Parameters:
- ***monitor*** ZEBU_Driver handler
- ***clockName*** clock name
- ***edgeName*** edge name "posedge" or "negedge"

Note:
this function has to be used with a trace memory driver

6.5.2.27 unsigned int ZEBU_Driver_update (ZEBU_Driver ∗ *driver*)

update of the driver interface

Parameters:
- ***driver*** ZEBU_Driver

6.5.2.28 unsigned int ZEBU_Driver_wait (ZEBU_Driver ∗ *driver*, unsigned int *triggers*, unsigned int *timeout*)

wait for a trigger event. Wait for the trigger event given in parameter while running the clock.

Parameters:
- ***driver*** ZEBU_Driver handler
- ***triggers*** triggers to stop on
  - set bit i to 1 to stop on trigger i (on the 16 lsb)
- ***timeout*** maximum number of cycles before to stop

6.6 ZEBU_Events.h File Reference

Include dependency graph for ZEBU_Events.h:

Typedefs

- typedef void(∗ handler_type)(ZEBU_EventReason reason)

Functions

- unsigned int ZEBU_Events_Register (ZEBU_Board ∗board, void(∗handler)(ZEBU_EventReason reason))

  Register a new global handler for ZEBU public events.

- unsigned int ZEBU_Events_Unregister (ZEBU_Board ∗board, void(∗handler)(ZEBU_EventReason reason))

  Unregister a global callback.

6.6.1 Typedef Documentation

6.6.1.1 typedef void(∗ handler_type)(ZEBU_EventReason reason)

6.6.2 Function Documentation

6.6.2.1 unsigned int ZEBU_Events_Register (ZEBU_Board ∗ *board*, void(∗)(ZEBU_EventReason reason) *handler*)

Register a new global handler for ZEBU public events.

Parameters:

handler The handler to register.

See also:

ZEBU_Events_Unregister

6.6.2.2 unsigned int ZEBU_Events_Unregister (ZEBU_Board ∗ *board*, void(∗)(ZEBU_EventReason reason) *handler*)

Unregister a global callback.

Parameters:

handler The handler to unregister.

See also:

ZEBU_Events_Register

6.7 ZEBU_FastHardwareState.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_FastHardwareState ∗ ZEBU_FastHardwareState_create ()

  create a fast hardware state

- int ZEBU_FastHardwareState_destroy (ZEBU_FastHardwareState ∗fast-HardwareState)

  destroy the fast hardware state created from ZEBU_WaveFile_create

- int ZEBU_FastHardwareState_initialize (ZEBU_FastHardwareState ∗fast-HardwareState, ZEBU_Board ∗board)

  initialize a fast hardware state

- int ZEBU_FastHardwareState_initialize2 (ZEBU_FastHardwareState ∗fast-HardwareState, ZEBU_Board ∗board, const ZEBU_Filter ∗filter)

  initialize a fast hardware state

- int ZEBU_FastHardwareState_capture (ZEBU_FastHardwareState ∗fast-HardwareState)

  capture fastly a hardware state into memory

- int ZEBU_FastHardwareState_save (ZEBU_FastHardwareState ∗fastHardware-State, const char ∗filename, int inParallel)

  write on disk the hardware state previouly captured

- int ZEBU_FastHardwareState_isParallelSaveFinished (const ZEBU_Fast-HardwareState ∗fastHardwareState)

  test if parallel save is finished.

- int ZEBU_FastHardwareState_clean (ZEBU_FastHardwareState ∗fast-HardwareState)

  clean the hardware state previouly captured. Release used memory.

6.7.1 Function Documentation

6.7.1.1 int ZEBU_FastHardwareState_capture (ZEBU_FastHardwareState ∗ *fastHardwareState*)

capture fastly a hardware state into memory

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

Returns:

int

Return values:

0 OK

$>$***0*** KO

6.7.1.2 int ZEBU_FastHardwareState_clean (ZEBU_FastHardwareState ∗ *fastHardwareState*)

clean the hardware state previouly captured. Release used memory.

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

Returns:

int

Return values:

0 OK

$>$***0*** KO

6.7.1.3 ZEBU_FastHardwareState ∗ ZEBU_FastHardwareState_create ()

create a fast hardware state

Returns:

handler to the created `ZEBU_FastHardwareState`

6.7.1.4 int ZEBU_FastHardwareState_destroy (ZEBU_FastHardwareState ∗ *fastHardwareState*)

destroy the fast hardware state created from ZEBU_WaveFile_create

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

Returns:

int

Return values:

0 OK

$>$***0*** KO

Example:

ZEBU_FastHardwareState ∗fastHardwareState = ZEBU_FastHardwareState_create(); if (fastHardwareState == 0) { printf("Cannot create fast hardware state\n"); exit(1); }

if (ZEBU_FastHardwareState_initialize(fastHardwareState, zebu) != 0) { printf("Cannot initialize fast hardware state\n"); exit(1); }

if (ZEBU_FastHardwareState_capture(fastHardwareState) != 0) { printf("Cannot capture fast hardware state\n"); exit(1); }

if (ZEBU_FastHardwareState_save(fastHardwareState, "fastHardwareState", 0) != 0) { printf("Cannot capture fast hardware state\n"); exit(1); }

if (ZEBU_FastHardwareState_clean(fastHardwareState) != 0) { printf("Cannot clean fast hardware state\n"); exit(1); }

if (ZEBU_FastHardwareState_destroy(fastHardwareState) != 0) { printf("Cannot destroy fast hardware state\n"); exit(1); }

6.7.1.5 int ZEBU_FastHardwareState_initialize (ZEBU_FastHardwareState ∗ *fastHardwareState*, ZEBU_Board ∗ *board*)

initialize a fast hardware state

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

board handler to a `ZEBU_Board`

Returns:

int

Return values:

0 OK

>*0* KO

6.7.1.6 int ZEBU_FastHardwareState_initialize2 (ZEBU_FastHardwareState ∗ *fastHardwareState*, ZEBU_Board ∗ *board*, const ZEBU_Filter ∗ *filter*)

initialize a fast hardware state

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

board handler to a `ZEBU_Board`

filter allow to filter the types of components of to save: internal signals, driver signals, internal and external memories, clocks ... Use 0 if no filter to use.

Returns:

int

Return values:

0 OK

>*0* KO

6.7.1.7 int ZEBU_FastHardwareState_isParallelSaveFinished (const ZEBU_FastHardwareState ∗ *fastHardwareState*)

test if parallel save is finished.

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

Returns:

int

Return values:

>*0* parallel save is finished

6.7.1.8 int ZEBU_FastHardwareState_save (ZEBU_FastHardwareState ∗ *fastHardwareState*, const char ∗ *filename*, int *inParallel*)

write on disk the hardware state previouly captured

Note:

this function create a compact folder in which is saved the "fast hardware state". This state can be restored by means of ZEBU_Board_restoreHardwareState or read by zState as a "hardware state". At restore or read the "fast hardware state" will be converted automatically in the same folder into a "hardware state" which structure is close to <zebu.work> tree.
this function cannot be called after board closing
does not release memory used by capture

Parameters:

fastHardwareState handler to a `ZEBU_FastHardwareState_create`

filename name of the file in which must be saved the state

inParallel specify if the state must be saved in a parallel task

Returns:

int

Return values:

0 OK

>***0*** KO

6.8 ZEBU_Filter.h File Reference

Include dependency graph for ZEBU_Filter.h:

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_Filter ∗ ZEBU_createFilter (unsigned int types, int numberOfHierarchicalLevels, char hierarchicalSeparator, const char ∗regularExpression, int invert, int ignoreCase)

    create a ZEBU Filter

- void ZEBU_destroyFilter (ZEBU_Filter ∗filter)

    destroy a ZEBU Filter

6.8.1 Function Documentation

6.8.1.1 ZEBU_Filter ∗ZEBU_Filter ∗ ZEBU_createFilter (unsigned int *types*, int *numberOfHierarchicalLevels*, char *hierarchicalSeparator*, const char ∗ *regularExpression*, int *invert*, int *ignoreCase*)

create a ZEBU Filter

Parameters:

types specify the types of components to enabled. Use combinaiton of ZEBU_-Filter_Type

numberOfHierarchicalLevels number of hierarchical levels to enable

hierarchicalSeparator hierarchical separator character

regularExpression regular expression

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

Example of regular expressions:

List of names A A.BB A.BB.CCC A.BB.CCC.DDD A.AA.AAA.AAAA AAAA.AAA.AA.A AA.AA.AA.AA AAA.AAA.AAA.AAA AAA.AAA.AAA.BBB A[N:0] A.BB[N:0] A.BB.CCC[N:0] A.BB.CCC.DDD[N:0] A.BB.CCC[0].DDD[N:0] A.BB.CCC[0].DDD A(N:0) A/BB(N:0) A/BB/CCC(N:0) A/BB/CCC/DDD(N:0)

Result of the regular expression = "CCC" A.BB.CCC A.BB.CCC.DDD A.BB.CCC[N:0] A.BB.CCC.DDD[N:0] A.BB.CCC[0].DDD[N:0] A.BB.CCC[0].DDD A/BB/CCC(N:0) A/BB/CCC/DDD(N:0)

Result of the regular expression = "CCC$" A.BB.CCC

Result of the regular expression = "CCC\\[.∗\\]$" A.BB.CCC[N:0] A.BB.CCC[0].DDD[N:0]

Result of the regular expression = "CCC\\(\\[.∗\\]\\)∗$" A.BB.CCC A.BB.CCC[N:0] A.BB.CCC[0].DDD[N:0]

Result of the regular expression = "CCC(.∗)$" A/BB/CCC(N:0)

Result of the regular expression = "CCC\\((.∗)\\)∗$" A.BB.CCC A/BB/CCC(N:0)

Result of the regular expression = "\\(.∗\\.\\)AA\\.\\(.∗\\.\\).∗" A.AA.AAA.AAAA AA.AA.AA.AA

Result of the regular expression = "\\(.∗\\.\\).\\{2,3\\}\\.\\(.∗\\.\\).[^A].∗$" A.BB.CCC.DDD AAA.AAA.AAA.BBB A.BB.CCC.DDD[N:0] A.BB.CCC[0].DDD[N:0] A.BB.CCC[0].DDD

6.8.1.2 void ZEBU_destroyFilter (ZEBU_Filter ∗ *filter*)

destroy a ZEBU Filter

Parameters:

filter filter to destroy

6.9 ZEBU_FlexibleLocalProbeFile.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int ZEBU_FlexibleLocalProbeFile_selectSamplingClock (ZEBU_Board ∗board, const char ∗clockName, const int edgeType)

  select the name of the flexible local probe sampling clock and its sensitive edge

- int ZEBU_FlexibleLocalProbeFile_selectSamplingClocks (ZEBU_Board ∗board, const char ∗clockExpression)

  select the set of clocks and sensitive edges on which the flexible local probes must be sampled

- ZEBU_FlexibleLocalProbeFile ∗ ZEBU_FlexibleLocalProbeFile_new ()

  allocate a ZEBU_FlexibleLocalProbeFile

- int ZEBU_FlexibleLocalProbeFile_delete (ZEBU_FlexibleLocalProbeFile ∗file)

  delete a ZEBU_FlexibleLocalProbeFile

- int ZEBU_FlexibleLocalProbeFile_add (ZEBU_FlexibleLocalProbeFile ∗file, const char ∗groupname)

  add a group in the file

- int ZEBU_FlexibleLocalProbeFile_add_regexp (ZEBU_FlexibleLocalProbeFile ∗file, const char ∗regularExpression, const int invert, const int ignoreCase, const char hierarchicalSeparator)

  add tracers in the group from a regular expression. The regular expression specifies the names of thegroups to add in the file

- long long unsigned ZEBU_FlexibleLocalProbeFile_enable (ZEBU_Flexible-LocalProbeFile ∗file)
- long long unsigned ZEBU_FlexibleLocalProbeFile_disable (ZEBU_Flexible-LocalProbeFile ∗file)
- int ZEBU_FlexibleLocalProbeFile_initialize (ZEBU_FlexibleLocalProbeFile ∗file, ZEBU_Board ∗board, const int thread)

  initialize the object

- int ZEBU_FlexibleLocalProbeFile_dumpFile (ZEBU_FlexibleLocalProbeFile ∗file, const char ∗filename)

  open the file

- long long unsigned ZEBU_FlexibleLocalProbeFile_flushFile (ZEBU_FlexibleLocalProbeFile ∗file)

  flush the file

- long long unsigned ZEBU_FlexibleLocalProbeFile_closeFile (ZEBU_FlexibleLocalProbeFile ∗file)

  close the file

6.9.1 Function Documentation

6.9.1.1 int ZEBU_FlexibleLocalProbeFile_add (ZEBU_FlexibleLocalProbeFile ∗ *file*, const char ∗ *groupname*)

add a group in the file

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

groupname name of the group to add

Return values:

<***0*** KO

6.9.1.2 int ZEBU_FlexibleLocalProbeFile_add_regexp (ZEBU_FlexibleLocalProbeFile ∗ *file*, const char ∗ *regularExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*)

add tracers in the group from a regular expression. The regular expression specifies the names of thegroups to add in the file

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

regularExpression regular expression

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

Returns:

status

Return values:

0 OK

>*0* KO

Example of regular expressions:

List of names G1 L0.G2 G2.L0 L0.L1.G2 L0.L1.L2.G2 L0.L1.L2.G3 L0.L1.L2.G4

Result of the regular expression = "G1" G1

Result of the regular expression = "G2" L0.G2 G2.L0 L0.L1.G2 L0.L1.L2.G2

Result of the regular expression = "G2$" L0.G2 L0.L1.G2 L0.L1.L2.G2

Result of the regular expression = "L0\\.L1\\.L2\\..*[^2]$" L0.L1.L2.G3 L0.L1.L2.G4

6.9.1.3 long long unsigned ZEBU_FlexibleLocalProbeFile_closeFile (ZEBU_FlexibleLocalProbeFile * *file*)

close the file

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

6.9.1.4 void ZEBU_FlexibleLocalProbeFile_delete (ZEBU_FlexibleLocalProbeFile * *file*)

delete a ZEBU_FlexibleLocalProbeFile

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

Returns:

status

Return values:

0 OK

>*0* KO

6.9.1.5 long long unsigned ZEBU_FlexibleLocalProbeFile_disable (ZEBU_FlexibleLocalProbeFile * *file*)

6.9.1.6 int ZEBU_FlexibleLocalProbeFile_dumpFile (ZEBU_FlexibleLocalProbeFile * *file*, const char * *filename*)

open the file

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

filename name of the file in which must be dump values of tracer's signals

- if extension is ".vcd", file is dumped in VCD format
- if extension is ".vpd", file is dumped in VPD format
- if extension is ".fsdb", file is dumped in FSDB format
- if extension is ".ztdb", file is dumped in ZeBu Fast Trace Database format. ZTDB format file can be converted to:
  - a VCD format file by means of ztdb2vcd <-i .ztdb filename> [-o <.vcd filename>]
  - a VPD format file by means of ztdb2vpd <-i .ztdb filename> [-o <.vpd filename>]
  - a FSDB format file by means of ztdb2fsdb <-i .ztdb filename> [-o <.fsdb filename>]

Returns:

status

Return values:

0 OK

>***0*** KO

6.9.1.7 long long unsigned ZEBU_FlexibleLocalProbeFile_enable (ZEBU_FlexibleLocalProbeFile ∗ *file*)

6.9.1.8 long long unsigned ZEBU_FlexibleLocalProbeFile_flushFile (ZEBU_FlexibleLocalProbeFile ∗ *file*)

flush the file

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

6.9.1.9 int ZEBU_FlexibleLocalProbeFile_initialize (ZEBU_FlexibleLocalProbeFile ∗ *file*, ZEBU_Board ∗ *board*, const int *thread*)

initialize the object

Parameters:

file ZEBU_FlexibleLocalProbeFile handler

board C++ handler on Board

thread specify on which thread must be run the tracer. If -1 no specification.

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.9.1.10 ZEBU_FlexibleLocalProbeFile ∗ ZEBU_FlexibleLocalProbeFile_new ()

allocate a ZEBU_FlexibleLocalProbeFile

Return values:

ZEBU_FlexibleLocalProbeFile∗

6.9.1.11 int ZEBU_FlexibleLocalProbeFile_selectSamplingClock (ZEBU_Board ∗ *board*, const char ∗ *clockName*, const int *edgeType*)

select the name of the flexible local probe sampling clock and its sensitive edge

Parameters:

board C handler on ZEBU_Board

clockName name of a controlled clock

edgeType sensitive clock edge, 1 for posedge, 0 for negedge

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.9.1.12 int ZEBU_FlexibleLocalProbeFile_selectSamplingClocks (ZEBU_Board ∗ *board*, const char ∗ *clockExpression*)

select the set of clocks and sensitive edges on which the flexible local probes must be sampled

Parameters:

board C handler on ZEBU_Board

clockExpression clock sensitivity expression: "[posedge|negedge] <clock name> [or [posedge|negedge] <clock name>] and so on" For instance: "posedge clock1" => sampling on clock1's posedges "posedge clock1 or negedge clock2" => sampling on clock1's posedges and clock2's negedges "clock3" => sampling on clock3's posedges and clock3's negedges

Returns:

status

Return values:

0 OK

*>**0*** KO

6.10 ZEBU_LocalTracer.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_LocalTraceDumper ∗ ZEBU_LocalTraceDumper_new ()

  allocate a ZEBU_LocalTraceDumper

- int ZEBU_LocalTraceDumper_delete (ZEBU_LocalTraceDumper ∗tracer)

  delete a ZEBU_LocalTraceDumper

- const char ∗ ZEBU_LocalTraceDumper_getPath (ZEBU_LocalTraceDumper ∗tracer)

  get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceDumper_getName (ZEBU_LocalTraceDumper ∗tracer)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceDumper_getFullname (ZEBU_LocalTrace-Dumper ∗tracer)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceDumper_enable (ZEBU_LocalTrace-Dumper ∗tracer)

  enable the tracer

- long long unsigned ZEBU_LocalTraceDumper_disable (ZEBU_LocalTrace-Dumper ∗tracer)

  disable the tracer

- int ZEBU_LocalTraceDumper_isEnabled (ZEBU_LocalTraceDumper ∗tracer)

  get tracer state

- int ZEBU_LocalTraceDumper_initialize (ZEBU_LocalTraceDumper ∗tracer, ZEBU_Board ∗board, const char ∗fullname, const int thread)

  initialize the object

- int ZEBU_LocalTraceDumper_setInNoXDumpMode (ZEBU_LocalTrace-Dumper ∗tracer) *set the tracer in special mode for disabling X dumps*

- int ZEBU_LocalTraceDumper_isNoXDumpModeStarted (ZEBU_LocalTraceDumper ∗tracer)

  test if the tracer is in special mode for disabling X dumps

- int ZEBU_LocalTraceDumper_dumpFile (ZEBU_LocalTraceDumper ∗tracer, const char ∗filename)

  initialize the tracer

- int ZEBU_LocalTraceDumper_dumpFileWithOffsetAndRatio (ZEBU_LocalTraceDumper ∗tracer, const char ∗filename, const long long int offset, unsigned int ratio)

  initialize the tracer

- long long unsigned ZEBU_LocalTraceDumper_flushFile (ZEBU_LocalTraceDumper ∗tracer)

  flush the tracer

- long long unsigned ZEBU_LocalTraceDumper_closeFile (ZEBU_LocalTraceDumper ∗tracer)

  close the tracer dump file

- ZEBU_LocalTraceReader ∗ ZEBU_LocalTraceReader_new ()

  allocate a ZEBU_LocalTraceReader

- int ZEBU_LocalTraceReader_delete (ZEBU_LocalTraceReader ∗tracer)

  delete a ZEBU_LocalTraceReader

- const char ∗ ZEBU_LocalTraceReader_getPath (ZEBU_LocalTraceReader ∗tracer)

  get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceReader_getName (ZEBU_LocalTraceReader ∗tracer)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceReader_getFullname (ZEBU_LocalTraceReader ∗tracer)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceReader_enable (ZEBU_LocalTraceReader ∗tracer) *enable the tracer*

- long long unsigned ZEBU_LocalTraceReader_disable (ZEBU_LocalTraceReader ∗tracer)

  disable the tracer

- int ZEBU_LocalTraceReader_isEnabled (ZEBU_LocalTraceReader ∗tracer)

  get tracer state

- int ZEBU_LocalTraceReader_initialize (ZEBU_LocalTraceReader ∗tracer, ZEBU_Board ∗board, const char ∗fullname, const int thread)

  initialize the object

- int ZEBU_LocalTraceReader_step (ZEBU_LocalTraceReader ∗tracer)

  read the next trace cyclea Block until the next trace cycle is received.

- int ZEBU_LocalTraceReader_run (ZEBU_LocalTraceReader ∗tracer, const long long unsigned ∗numberOfCycles)

  read the nth next trace cycles. Block until the nth next trace cycles are received.

- int ZEBU_LocalTraceReader_waitNextChange (ZEBU_LocalTraceReader ∗tracer)

  wait for next value change Block until the nth next value change is received.

- int ZEBU_LocalTraceReader_tryStep (ZEBU_LocalTraceReader ∗tracer)

  try to read the next trace cycle

- int ZEBU_LocalTraceReader_tryRun (ZEBU_LocalTraceReader ∗tracer, const long long unsigned ∗numberOfCycles)

  try to read the nth next trace cycles

- int ZEBU_LocalTraceReader_getNextChange (ZEBU_LocalTraceReader ∗tracer)

  try to get next value change

- long long unsigned ZEBU_LocalTraceReader_getTime (ZEBU_LocalTraceReader ∗tracer)

  get the current timestamp of the tracer The timestamp is the number of cycles generated for the clock synchronizing all tracers corresponding with the current values of tracer's signals

- ZEBU_Signal ∗ ZEBU_LocalTraceReader_getSignal (ZEBU_LocalTraceReader ∗tracer, const char ∗name) *get a signal handler to read its value*

- ZEBU_LocalTraceImporter ∗ ZEBU_LocalTraceImporter_new ()

  allocate a ZEBU_LocalTraceImporter

- int ZEBU_LocalTraceImporter_delete (ZEBU_LocalTraceImporter ∗tracer)

  delete a ZEBU_LocalTraceImporter

- const char ∗ ZEBU_LocalTraceImporter_getPath (ZEBU_LocalTraceImporter ∗tracer)

  get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceImporter_getName (ZEBU_LocalTraceImporter ∗tracer)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceImporter_getFullname (ZEBU_LocalTrace-Importer ∗tracer)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceImporter_enable (ZEBU_LocalTrace-Importer ∗tracer)

  enable the tracer

- long long unsigned ZEBU_LocalTraceImporter_disable (ZEBU_LocalTrace-Importer ∗tracer)

  disable the tracer

- int ZEBU_LocalTraceImporter_isEnabled (ZEBU_LocalTraceImporter ∗tracer)

  get tracer state

- int ZEBU_LocalTraceImporter_initialize (ZEBU_LocalTraceImporter ∗tracer, ZEBU_Board ∗board, const char ∗fullname, const char ∗importName, const int onEvent, const int thread, const int synchronous)

  initialize the tracer

- int ZEBU_LocalTraceImporter_initialize_userData (ZEBU_LocalTrace-Importer ∗tracer, ZEBU_Board ∗board, const char ∗fullname, void ∗userData, const char ∗importName, const int onEvent, const int thread, const int synchronous)

  initialize the object with a user pointer

- int ZEBU_LocalTrace_selectSamplingClock (ZEBU_Board ∗board, const char ∗clockName, const int edgeType)

  select the name of the local trace sampling clock and its sensitive edge

6.10.1 Function Documentation

6.10.1.1 static int ZEBU_LocalTrace_selectSamplingClock (ZEBU_Board ∗ *board*, const char ∗ *clockName*, const int *edgeType*)

select the name of the local trace sampling clock and its sensitive edge

Parameters:

board C++ handler on Board

clockName name of a controlled clock

edgeType sensitive clock edge, 1 for posedge, 0 for negedge

Returns:

status

Return values:

0 OK

>***0*** KO

6.10.1.2 ZEBU_LocalTraceDumper_closeFile (ZEBU_LocalTraceDumper ∗ *tracer*)

close the tracer dump file

Parameters:

tracer ZEBU_LocalTraceDumper handler

6.10.1.3 int ZEBU_LocalTraceDumper_delete (ZEBU_LocalTraceDumper ∗ *tracer*)

delete a ZEBU_LocalTraceDumper

Parameters:

tracer ZEBU_LocalTraceDumper handler

Returns:

status

Return values:

0 OK

>0 KO

6.10.1.4 long long unsigned ZEBU_LocalTraceDumper_disable (ZEBU_LocalTraceDumper ∗ *tracer*)

disable the tracer

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

timestamp at which the tracer has been disabled

ULLONG_MAX on error

6.10.1.5 void ZEBU_LocalTraceDumper_dumpFile (ZEBU_LocalTraceDumper ∗ *tracer*, const char ∗ *filename*)

initialize the tracer

Parameters:

tracer ZEBU_LocalTraceDumper handler

filename name of the file in which must be dump values of tracer's signals

- if extension is ".vcd", file is dumped in VCD format
- if extension is ".vpd", file is dumped in VPD format
- if extension is ".fsdb", file is dumped in FSDB format
- if extension is ".ztdb", file is dumped in ZeBu Fast Trace Database format. ZTDB format file can be converted to:
  - a VCD format file by means of ztdb2vcd <-i .ztdb filename> [-o <.vcd filename>]
  - a VPD format file by means of ztdb2vpd <-i .ztdb filename> [-o <.vpd filename>]
  - a FSDB format file by means of ztdb2fsdb <-i .ztdb filename> [-o <.fsdb filename>]

Returns:

status

Return values:

0 OK

>***0*** KO

6.10.1.6 void ZEBU_LocalTraceDumper_dumpFileWithOffsetAndRatio (ZEBU_LocalTraceDumper ∗ *tracer*, const char ∗ *filename*, const long long int *offset*, unsigned int *ratio*)

initialize the tracer

Parameters:

tracer ZEBU_LocalTraceDumper handler

filename name of the file in which must be dump values of tracer's signals

- if extension is ".vcd", file is dumped in VCD format
- if extension is ".vpd", file is dumped in VPD format
- if extension is ".fsdb", file is dumped in FSDB format
- the extension is ".ztdb" is not supported

offset positive of negative offset to apply to the timestamp

ratio ratio in number of sample clock cycles to apply to the sample clock cycle to obtain the timestamp

Returns:

status

Return values:

0 OK

>***0*** KO

6.10.1.7 long long unsigned ZEBU_LocalTraceDumper_enable (ZEBU_LocalTraceDumper ∗ *tracer*)

enable the tracer

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

timestamp at which the tracer has been enabled

ULLONG_MAX on error

6.10.1.8 ZEBU_LocalTraceDumper_flushFile (ZEBU_LocalTraceDumper ∗ *tracer*)

flush the tracer

Parameters:

tracer ZEBU_LocalTraceDumper handler

6.10.1.9 const char ∗ ZEBU_LocalTraceDumper_getFullname (ZEBU_LocalTraceDumper ∗ *tracer*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

tracer full name

6.10.1.10 const char ∗ ZEBU_LocalTraceDumper_getName (ZEBU_LocalTraceDumper ∗ *tracer*)

get the tracer <RNAME>

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

tracer <RNANE>

6.10.1.11 const char ∗ ZEBU_LocalTraceDumper_getPath (ZEBU_LocalTraceDumper ∗ *tracer*)

get the tracer <RPATH>

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

tracer <RPATH>

6.10.1.12 int ZEBU_LocalTraceDumper_initialize (ZEBU_LocalTraceDumper ∗ *tracer*, ZEBU_Board ∗ *board*, const char ∗ *fullname*, const int *thread*)

initialize the object

Parameters:

tracer ZEBU_LocalTraceDumper handler

board C++ handler on Board

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

thread specify on which thread must be run the tracer. If -1 no specification.

Returns:

status

Return values:

0 OK

>***0*** KO

6.10.1.13 int ZEBU_LocalTraceDumper_isEnabled (ZEBU_LocalTraceDumper ∗ *tracer*)

get tracer state

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

1 if tracer is enabled, else 0

6.10.1.14 int ZEBU_LocalTraceDumper_isNoXDumpModeStarted (ZEBU_LocalTraceDumper ∗ *tracer*)

test if the tracer is in special mode for disabling X dumps

Parameters:

tracer ZEBU_LocalTraceDumper handler

Return values:

0 : X dump is disabled

>***0*** : X dump is enabled

-1 : error

6.10.1.15 ZEBU_LocalTraceDumper $*$ ZEBU_LocalTraceDumper_new ()

allocate a ZEBU_LocalTraceDumper

Return values:

LocalTraceDumper$*$

6.10.1.16 int ZEBU_LocalTraceDumper_setInNoXDumpMode (ZEBU_LocalTraceDumper $*$ *tracer*)

set the tracer in special mode for disabling X dumps

Parameters:

tracer ZEBU_LocalTraceDumper handler

6.10.1.17 void ZEBU_LocalTraceImporter_delete (ZEBU_LocalTraceImporter $*$ *tracer*)

delete a ZEBU_LocalTraceImporter

Parameters:

tracer ZEBU_LocalTraceImporter handler

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.10.1.18 long long unsigned ZEBU_LocalTraceImporter_disable (ZEBU_LocalTraceImporter $*$ *tracer*)

disable the tracer

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

timestamp at which the tracer has been disabled

ULLONG_MAX on error

6.10.1.19 long long unsigned ZEBU_LocalTraceImporter_enable (ZEBU_LocalTraceImporter ∗ *tracer*)

enable the tracer

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

timestamp at which the tracer has been enabled

ULLONG_MAX on error

6.10.1.20 const char ∗ ZEBU_LocalTraceImporter_getFullname (ZEBU_LocalTraceImporter ∗ *tracer*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

tracer full name

6.10.1.21 const char ∗ ZEBU_LocalTraceImporter_getName (ZEBU_LocalTraceImporter ∗ *tracer*)

get the tracer <RNAME>

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

tracer <RNAME>

6.10.1.22 const char ∗ ZEBU_LocalTraceImporter_getPath (ZEBU_LocalTraceImporter ∗ *tracer*)

get the tracer <RPATH>

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

tracer <RPATH>

6.10.1.23 int ZEBU_LocalTraceImporter_initialize (ZEBU_LocalTraceImporter ∗ *tracer*, ZEBU_Board ∗ *board*, const char ∗ *fullname*, const char ∗ *importName*, const int *onEvent*, const int *thread*, const int *synchronous*)

initialize the tracer

Parameters:

tracer ZEBU_LocalTraceImporter handler

board handler on Board

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

importName name of the C function to be called. If NULL, load the C function which name matchs with the RNAME of the tracer. It is possible to use the same C function for several tracers. The C function can call the followings System Verilog functions: svGetNameFromScope: return the <RPATH> of the tracer svPutUserData: save a user pointer for the tracer scope svGetUserData: return the user pointer saved for the tracer scope

onEvent if 0 imported C function is called for each trace cycle at which the tracer is enabled by software and hardware pin else if 1 imported C function is called only when software receive event on tracer's values

thread specify on which thread must be run the tracer. If -1 no specification.

synchronous if true imported C function calls are synchronized with other synchronous importers of the same thread according to timestamps of value changes

Returns:

status

Return values:

0 OK

>0 KO

6.10.1.24 int ZEBU_LocalTraceImporter_initialize_userData (ZEBU_LocalTraceImporter ∗ *tracer*, ZEBU_Board ∗ *board*, const char ∗ *fullname*, void ∗ *userData*, const char ∗ *importName*, const int *onEvent*, const int *thread*, const int *synchronous*)

initialize the object with a user pointer

Parameters:

board C++ handler on Board

tracer ZEBU_LocalTraceImporter handler

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

userData specify the user pointer.to the first argument of the C function. Tracer'signal values are set from the second argument and so on.

importName name of the C function to be called with the user data. If NULL, load the C function which name matchs with the RNAME of the tracer. It is possible to use the same C function for several tracers.

onEvent if 0 imported C function is called for each trace cycle at which the tracer is enabled by software and hardware pin else if 1 imported C function is called only when software receive event on tracer's values

thread specify on which thread must be run the tracer. If -1 no specification.

synchronous if true imported C function calls are synchronized with other synchronous importers of the same thread according to timestamps of value changes

Returns:

status

Return values:

0 OK

>***0*** KO

6.10.1.25 int ZEBU_LocalTraceImporter_isEnabled (ZEBU_LocalTraceImporter ∗ *tracer*)

get tracer state

Parameters:

tracer ZEBU_LocalTraceImporter handler

Return values:

1 if tracer is enabled, else 0

6.10.1.26 ZEBU_LocalTraceImporter ∗ ZEBU_LocalTraceImporter_new ()

allocate a ZEBU_LocalTraceImporter

Return values:

LocalTraceImporter∗

6.10.1.27 void ZEBU_LocalTraceReader_delete (ZEBU_LocalTraceReader ∗ *tracer*)

delete a ZEBU_LocalTraceReader

Parameters:

tracer ZEBU_LocalTraceReader handler

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.10.1.28 long long unsigned ZEBU_LocalTraceReader_disable (ZEBU_LocalTraceReader ∗ *tracer*)

disable the tracer

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

timestamp at which the tracer has been disabled

ULLONG_MAX on error

6.10.1.29 long long unsigned ZEBU_LocalTraceReader_enable (ZEBU_LocalTraceReader ∗ *tracer*)

enable the tracer

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

timestamp at which the tracer has been enabled

ULLONG_MAX on error

6.10.1.30 void ZEBU_LocalTraceReader_getFullname (ZEBU_LocalTraceReader ∗ *tracer*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

tracer full name

6.10.1.31 const char ∗ ZEBU_LocalTraceReader_getName (ZEBU_LocalTraceReader ∗ *tracer*)

get the tracer <RNAME>

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

tracer <RMANE:>

6.10.1.32 int ZEBU_LocalTraceReader_getNextChange (ZEBU_LocalTraceReader ∗ *tracer*)

try to get next value change

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

0 if sucessfull

1 if the next value change is not available yet

-1 if tracer is disabled and if the end of trace has been reached

6.10.1.33 const char ∗ ZEBU_LocalTraceReader_getPath (ZEBU_LocalTraceReader ∗ *tracer*)

get the tracer <RPATH>

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

tracer <RPATH>

6.10.1.34 ZEBU_Signal ∗ ZEBU_LocalTraceReader_getSignal (ZEBU_LocalTraceReader ∗ *tracer*, const char ∗ *name*)

get a signal handler to read its value

Parameters:

tracer ZEBU_LocalTraceReader handler

name signal name

Return values:

handler on a llocal tracer signal

6.10.1.35 long long unsigned ZEBU_LocalTraceReader_getTime (ZEBU_LocalTraceReader ∗ *tracer*)

get the current timestamp of the tracer The timestamp is the number of cycles generated for the clock synchronizing all tracers corresponding with the current values of tracer's signals

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

current timestamp of the tracer

6.10.1.36 int ZEBU_LocalTraceReader_initialize (ZEBU_LocalTraceReader ∗ *tracer*, ZEBU_Board ∗ *board*, const char ∗ *fullname*, const int *thread*)

initialize the object

Parameters:

tracer ZEBU_LocalTraceReader handler

board C++ handler on Board

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

thread specify on which thread must be run the tracer. If -1 no specification.

Returns:

status

Return values:

0 OK

>0 KO

6.10.1.37 int ZEBU_LocalTraceReader_isEnabled (ZEBU_LocalTraceReader ∗ *tracer*)

get tracer state

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

1 if tracer is enabled, else 0

6.10.1.38 ZEBU_LocalTraceReader ∗ ZEBU_LocalTraceReader_new ()

allocate a ZEBU_LocalTraceReader

Return values:

LocalTraceReader∗

6.10.1.39 int ZEBU_LocalTraceReader_run (ZEBU_LocalTraceReader ∗ *tracer*, const long long unsigned ∗ *numberOfCycles*)

read the nth next trace cycles. Block until the nth next trace cycles are received.

Parameters:

tracer ZEBU_LocalTraceReader handler

numberOfCycles number of cycles to run

Return values:

0 if sucessfull

-1 if tracer is disabled and if the end of trace has been reached

6.10.1.40 int ZEBU_LocalTraceReader_step (ZEBU_LocalTraceReader ∗ *tracer*)

read the next trace cyclea Block until the next trace cycle is received.

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

0 if sucessfull

-1 if tracer is disabled and if the end of trace has been reached

6.10.1.41 int ZEBU_LocalTraceReader_tryRun (ZEBU_LocalTraceReader ∗ *tracer*, const long long unsigned ∗ *numberOfCycles*)

try to read the nth next trace cycles

Parameters:

tracer ZEBU_LocalTraceReader handler

numberOfCycles number of cycles to try to run

Return values:

the number of read cycles

-1 if tracer is disabled and if the end of trace has been reached

6.10.1.42 int ZEBU_LocalTraceReader_tryStep (ZEBU_LocalTraceReader ∗ *tracer*)

try to read the next trace cycle

Parameters:

tracer ZEBU_LocalTraceReader handler

Return values:

0 if sucessfull

1 if the next trace cycle is not available yet

-1 if tracer is disabled and if the end of trace has been reached

6.10.1.43 int ZEBU_LocalTraceReader_waitNextChange (ZEBU_LocalTraceReader ∗ *tracer*)

wait for next value change Block until the nth next value change is received.

Parameters:

tracer	ZEBU_LocalTraceReader handler

Return values:

0	if sucessfull
-1	if tracer is disabled and if the end of trace has been reached

6.11 ZEBU_LocalTracerGroup.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_LocalTraceDumperGroup ∗ ZEBU_LocalTraceDumperGroup_new ()

    allocate a ZEBU_LocalTraceDumperGroup

- int ZEBU_LocalTraceDumperGroup_delete (ZEBU_LocalTraceDumperGroup ∗group)

    delete a ZEBU_LocalTraceDumperGroup

- int ZEBU_LocalTraceDumperGroup_add (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const char ∗fullname)

    add a tracer to the group

- int ZEBU_LocalTraceDumperGroup_add_regexp (ZEBU_LocalTraceDumper-Group ∗tracerGroup, const char ∗regularExpression, const int invert, const int ignoreCase, const char hierarchicalSeparator)

    add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

- int ZEBU_LocalTraceDumperGroup_getNumberOfTracers (ZEBU_Local-TraceDumperGroup ∗tracerGroup)

    get the number of tracers added in the group

- int ZEBU_LocalTraceDumperGroup_getIdentifier (ZEBU_LocalTraceDumper-Group ∗tracerGroup, const char ∗fullname)

    get identifier of the tracer

- long long unsigned ZEBU_LocalTraceDumperGroup_enableAll (ZEBU_Local-TraceDumperGroup ∗tracerGroup)

    enable all tracers of the group

- long long unsigned ZEBU_LocalTraceDumperGroup_disableAll (ZEBU_-LocalTraceDumperGroup ∗tracerGroup)

    diable all tracers of the group

- const char ∗ ZEBU_LocalTraceDumperGroup_getPath (ZEBU_LocalTrace-DumperGroup ∗tracerGroup, const int tracerIdentifier)

    get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceDumperGroup_getName (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceDumperGroup_getFullname (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceDumperGroup_enable (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const int tracerIdentifier)

  enable the tracer

- long long unsigned ZEBU_LocalTraceDumperGroup_disable (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const int tracerIdentifier)

  disable the tracer

- int ZEBU_LocalTraceDumperGroup_isEnabled (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const int tracerIdentifier)

  get tracer state

- int ZEBU_LocalTraceDumperGroup_initialize (ZEBU_LocalTraceDumperGroup ∗tracerGroup, ZEBU_Board ∗board, const int thread)

  initialize the object

- int ZEBU_LocalTraceDumperGroup_setInNoXDumpMode (ZEBU_LocalTraceDumperGroup ∗tracerGroup)

  set the group in special mode for disabling X dumps

- int ZEBU_LocalTraceDumperGroup_isNoXDumpModeStarted (ZEBU_LocalTraceDumperGroup ∗tracerGroup)

  get if the group is in special mode for disabling X dumps

- int ZEBU_LocalTraceDumperGroup_dumpFile (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const char ∗filename)

  open the file

- int ZEBU_LocalTraceDumperGroup_dumpFileWithOffsetAndRatio (ZEBU_LocalTraceDumperGroup ∗tracerGroup, const char ∗filename, const long long int offset, const unsigned int ratio)

  initialize the tracer

- long long unsigned ZEBU_LocalTraceDumperGroup_flushFile (ZEBU_LocalTraceDumperGroup ∗tracerGroup) *flush the tracer*

- long long unsigned ZEBU_LocalTraceDumperGroup_closeFile (ZEBU_LocalTraceDumperGroup ∗tracerGroup)

  close the tracer dump file

- ZEBU_LocalTraceReaderGroup ∗ ZEBU_LocalTraceReaderGroup_new ()

  allocate a ZEBU_LocalTraceReaderGroup

- int ZEBU_LocalTraceReaderGroup_delete (ZEBU_LocalTraceReaderGroup ∗group)

  delete a ZEBU_LocalTraceReaderGroup

- int ZEBU_LocalTraceReaderGroup_add (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const char ∗fullname)

  add a tracer to the group

- int ZEBU_LocalTraceReaderGroup_add_regexp (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const char ∗regularExpression, const int invert, const int ignoreCase, const char hierarchicalSeparator)

  add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

- int ZEBU_LocalTraceReaderGroup_getNumberOfTracers (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  get the number of tracers added in the group

- int ZEBU_LocalTraceReaderGroup_getIdentifier (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const char ∗fullname)

  get identifier of the tracer

- long long unsigned ZEBU_LocalTraceReaderGroup_enableAll (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  enable all tracers of the group

- long long unsigned ZEBU_LocalTraceReaderGroup_disableAll (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  diable all tracers of the group

- const char ∗ ZEBU_LocalTraceReaderGroup_getPath (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceReaderGroup_getName (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceReaderGroup_getFullname (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceReaderGroup_enable (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  enable the tracer

- long long unsigned ZEBU_LocalTraceReaderGroup_disable (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  disable the tracer

- int ZEBU_LocalTraceReaderGroup_isEnabled (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier)

  get tracer state

- int ZEBU_LocalTraceReaderGroup_initialize (ZEBU_LocalTraceReaderGroup ∗tracerGroup, ZEBU_Board ∗board, const int thread)

  initialize the object

- int ZEBU_LocalTraceReaderGroup_step (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  read the next trace cyclea Block until the next trace cycle is received.

- int ZEBU_LocalTraceReaderGroup_run (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const long long unsigned ∗numberOfCycles)

  read the nth next trace cycles. Block until the nth next trace cycles are received.

- int ZEBU_LocalTraceReaderGroup_waitNextChange (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  wait for next value change Block until the nth next value change is received.

- int ZEBU_LocalTraceReaderGroup_tryStep (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  try to read the next trace cycle

- int ZEBU_LocalTraceReaderGroup_tryRun (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const long long unsigned ∗numberOfCycles)

  try to read the nth next trace cycles

- int ZEBU_LocalTraceReaderGroup_getNextChange (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  try to get next value change

- long long unsigned ZEBU_LocalTraceReaderGroup_getTime (ZEBU_LocalTraceReaderGroup ∗tracerGroup)

  get the current timestamp of the tracer The timestamp is the number of cycles generated for the clock synchronizing all tracers corresponding with the current values of tracer's signals

- ZEBU_Signal ∗ ZEBU_LocalTraceReaderGroup_getSignal (ZEBU_LocalTraceReaderGroup ∗tracerGroup, const int tracerIdentifier, const char ∗name)

  get a signal handler to read its value

- int ZEBU_LocalTraceImporterGroup_add (ZEBU_LocalTraceImporterGroup ∗tracerGroup, const char ∗fullname)

  add a tracer to the group

- int ZEBU_LocalTraceImporterGroup_add_regexp (ZEBU_LocalTraceImporterGroup ∗tracerGroup, const char ∗regularExpression, const int invert, const int ignoreCase, const char hierarchicalSeparator)

  add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

- int ZEBU_LocalTraceImporterGroup_getNumberOfTracers (ZEBU_LocalTraceImporterGroup ∗tracerGroup)

  get the number of tracers added in the group

- int ZEBU_LocalTraceImporterGroup_getIdentifier (ZEBU_LocalTraceImporterGroup ∗tracerGroup, const char ∗fullname)

  get identifier of the tracer

- long long unsigned ZEBU_LocalTraceImporterGroup_enableAll (ZEBU_LocalTraceImporterGroup ∗tracerGroup)

  enable all tracers of the group

- long long unsigned ZEBU_LocalTraceImporterGroup_disableAll (ZEBU_LocalTraceImporterGroup ∗tracerGroup)

  diable all tracers of the group

- const char ∗ ZEBU_LocalTraceImporterGroup_getPath (ZEBU_LocalTraceImporterGroup ∗tracerGroup)

  get the tracer <RPATH>

- const char ∗ ZEBU_LocalTraceImporterGroup_getName (ZEBU_LocalTrace-ImporterGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer <RNAME>

- const char ∗ ZEBU_LocalTraceImporterGroup_getFullname (ZEBU_Local-TraceImporterGroup ∗tracerGroup, const int tracerIdentifier)

  get the tracer full name : <RPATH>.<RNAME>

- long long unsigned ZEBU_LocalTraceImporterGroup_enable (ZEBU_Local-TraceImporterGroup ∗tracerGroup, const int tracerIdentifier)

  enable the tracer

- long long unsigned ZEBU_LocalTraceImporterGroup_disable (ZEBU_Local-TraceImporterGroup ∗tracerGroup, const int tracerIdentifier)

  disable the tracer

- int ZEBU_LocalTraceImporterGroup_isEnabled (ZEBU_LocalTraceImporter-Group ∗tracerGroup, const int tracerIdentifier)

  get tracer state

- ZEBU_LocalTraceImporterGroup ∗ ZEBU_LocalTraceImporterGroup_new ()

  allocate a ZEBU_LocalTraceImporterGroup

- int ZEBU_LocalTraceImporterGroup_delete (ZEBU_LocalTraceImporter-Group ∗group)

  delete a ZEBU_LocalTraceImporterGroup

- int ZEBU_LocalTraceImporterGroup_initialize (ZEBU_LocalTraceImporter-Group ∗tracerGroup, ZEBU_Board ∗board, const char ∗importName, const int onEvent, const int thread, const int synchronous)

  initialize the tracer

- int ZEBU_LocalTraceImporterGroup_initialize_userData (ZEBU_LocalTrace-ImporterGroup ∗tracerGroup, ZEBU_Board ∗board, void ∗userData, const char ∗importName, const int onEvent, const int thread, const int synchronous)

  initialize the object with a user pointer

6.11.1 Function Documentation

6.11.1.1 int ZEBU_LocalTraceDumperGroup_add (ZEBU_-LocalTraceDumperGroup ∗ *tracerGroup*, const char ∗ *fullname*)

add a tracer to the group

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

Returns:

identifier of the tracer in the group

Return values:

<***0*** KO

6.11.1.2 int ZEBU_LocalTraceDumperGroup_add_regexp (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const char ∗ *regularExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*)

add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

regularExpression regular expression

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

Returns:

status

Return values:

0 OK

>***0*** KO

Example of regular expressions:

List of names T1 L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2 L0.L1.L2.T3 L0.L1.L2.T4

Result of the regular expression = "T1" T1

Result of the regular expression = "T2" L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "T2$" L0.T2 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "L0\\\\.L1\\\\.L2\\\\..∗[^2]$" L0.L1.L2.T3 L0.L1.L2.T4

6.11.1.3 ZEBU_LocalTraceDumperGroup_closeFile (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*)

close the tracer dump file

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

6.11.1.4 void ZEBU_LocalTraceDumperGroup_delete (ZEBU_LocalTraceDumperGroup ∗ *group*)

delete a ZEBU_LocalTraceDumperGroup

Parameters:

group ZEBU_LocalTraceDumperGroup handler

Returns:

status

Return values:

0 OK

>***0*** KO

6.11.1.5 void ZEBU_LocalTraceDumperGroup_disable (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

disable the tracer

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

Return values:

timestamp at which the tracer has been disabled

ULLONG_MAX on error

6.11.1.6 ZEBU_LocalTraceDumperGroup_disableAll (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*)

diable all tracers of the group

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

Return values:

timestamp at which the tracers have been disabled

ULLONG_MAX on error

6.11.1.7 void ZEBU_LocalTraceDumperGroup_dumpFile (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const char ∗ *filename*)

open the file

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

filename name of the file in which must be dump values of tracer's signals

- if extension is ".vcd", file is dumped in VCD format
- if extension is ".vpd", file is dumped in VPD format
- if extension is ".fsdb", file is dumped in FSDB format
- if extension is ".ztdb", file is dumped in ZeBu Fast Trace Database format. ZTDB format file can be converted to:
  – a VCD format file by means of ztdb2vcd <-i .ztdb filename> [-o <.vcd filename>]
  – a VPD format file by means of ztdb2vpd <-i .ztdb filename> [-o <.vpd filename>]
  – a FSDB format file by means of ztdb2fsdb <-i .ztdb filename> [-o <.fsdb filename>]

Returns:

status

Return values:

0 OK

$>$*0* KO

6.11.1.8 void ZEBU_LocalTraceDumperGroup_dumpFileWithOffsetAndRatio (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*, const char $*$ *filename*, const long long int *offset*, const unsigned int *ratio*)

initialize the tracer

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

filename name of the file in which must be dump values of tracer's signals

- if extension is ".vcd", file is dumped in VCD format
- if extension is ".vpd", file is dumped in VPD format
- if extension is ".fsdb", file is dumped in FSDB format
- the extension ".ztdb" is not supported

offset positive of negative offset to apply to the timestamp

ratio ratio in number of sample clock cycles to apply to the sample clock cycle to obtain the timestamp

Returns:

status

Return values:

0 OK

$>$*0* KO

6.11.1.9 void ZEBU_LocalTraceDumperGroup_enable (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*, const int *tracerIdentifier*)

enable the tracer

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier tracer identifier

Return values:

timestamp at which the tracer has been enabled

ULLONG_MAX on error

6.11.1.10 ZEBU_LocalTraceDumperGroup_enableAll (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*)

enable all tracers of the group

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

Return values:

timestamp at which the tracers have been enabled

ULLONG_MAX on error

6.11.1.11 ZEBU_LocalTraceDumperGroup_flushFile (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*)

flush the tracer

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

6.11.1.12 void ZEBU_LocalTraceDumperGroup_getFullname (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*, const int *tracerIdentifier*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

Return values:

tracer full name

6.11.1.13 ZEBU_LocalTraceDumperGroup_getIdentifier (ZEBU_LocalTraceDumperGroup $*$ *tracerGroup*, const char $*$ *fullname*)

get identifier of the tracer

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

fullname full name of the tracer

Return values:

int

6.11.1.14 void ZEBU_LocalTraceDumperGroup_getName (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer <RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

Return values:

tracer <RNAME>

6.11.1.15 ZEBU_LocalTraceDumperGroup_getNumberOfTracers (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*)

get the number of tracers added in the group

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

Return values:

int

6.11.1.16 void ZEBU_LocalTraceDumperGroup_getPath (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer <RPATH>

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

Return values:

tracer <RPATH>

6.11.1.17 int ZEBU_LocalTraceDumperGroup_initialize (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, ZEBU_Board ∗ *board*, const int *thread*)

initialize the object

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

board C++ handler on Board

thread specify on which thread must be run the tracer. If -1 no specification.

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.11.1.18 void ZEBU_LocalTraceDumperGroup_isEnabled (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get tracer state

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

Return values:

1 if tracer is enabled, else 0

6.11.1.19 int ZEBU_LocalTraceDumperGroup_isNoXDumpModeStarted (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*)

get if the group is in special mode for disabling X dumps

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

Return values:

0 : X dump is disabled

$>$***0*** : X dump is enabled

-1 : error

6.11.1.20 ZEBU_LocalTraceDumperGroup ∗ ZEBU_LocalTraceDumperGroup_new ()

allocate a ZEBU_LocalTraceDumperGroup

Return values:

ZEBU_LocalTraceDumperGroup∗

6.11.1.21 int ZEBU_LocalTraceDumperGroup_setInNoXDumpMode (ZEBU_LocalTraceDumperGroup ∗ *tracerGroup*)

set the group in special mode for disabling X dumps

Parameters:

tracerGroup ZEBU_LocalTraceDumperGroup handler

Returns:

status

Return values:

0 OK

>*0* KO

6.11.1.22 void ZEBU_LocalTraceImporterGroup_add (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const char ∗ *fullname*)

add a tracer to the group

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

Return values:

identifier of the tracer in the group

6.11.1.23 void ZEBU_LocalTraceImporterGroup_add_regexp (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const char ∗ *regularExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*)

add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

regularExpression regular expression

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

Example of regular expressions:

List of names T1 L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2 L0.L1.L2.T3 L0.L1.L2.T4

Result of the regular expression = "T1" T1

Result of the regular expression = "T2" L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "T2$" L0.T2 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "L0\\.L1\\.L2\\..∗[^2]$" L0.L1.L2.T3 L0.L1.L2.T4

6.11.1.24 int ZEBU_LocalTraceImporterGroup_delete (ZEBU_LocalTraceImporterGroup ∗ *group*)

delete a ZEBU_LocalTraceImporterGroup

Parameters:

group ZEBU_LocalTraceImporterGroup handler

Returns:

status

Return values:

0 OK

>0 KO

6.11.1.25 long long unsigned ZEBU_LocalTraceImporterGroup_disable (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

disable the tracer

Parameters:

tracerGroup	ZEBU_LocalTraceImporterGroup handler
tracerIdentifier	tracer identifier

Return values:

timestamp	at which the tracer has been disabled
ULLONG_MAX	on error

6.11.1.26 long long unsigned ZEBU_LocalTraceImporterGroup_disableAll (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*)

diable all tracers of the group

Parameters:

tracerGroup	ZEBU_LocalTraceImporterGroup handler

Return values:

timestamp	at which the tracers have been disabled
ULLONG_MAX	on error

6.11.1.27 long long unsigned ZEBU_LocalTraceImporterGroup_enable (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

enable the tracer

Parameters:

tracerGroup	ZEBU_LocalTraceImporterGroup handler
tracerIdentifier	tracer identifier

Return values:

timestamp	at which the tracer has been enabled
ULLONG_MAX	on error

6.11.1.28 long long unsigned ZEBU_LocalTraceImporterGroup_enableAll (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*)

enable all tracers of the group

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

Return values:

timestamp at which the tracers have been enabled

ULLONG_MAX on error

6.11.1.29 const char ∗ ZEBU_LocalTraceImporterGroup_getFullname (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

tracerIdentifier tracer identifier

Return values:

tracer full name

6.11.1.30 int ZEBU_LocalTraceImporterGroup_getIdentifier (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const char ∗ *fullname*)

get identifier of the tracer

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

fullname full name of the tracer

Return values:

int

6.11.1.31 const char ∗ ZEBU_LocalTraceImporterGroup_getName (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer <RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

tracerIdentifier tracer identifier

Return values:

tracer <RNAME>

6.11.1.32 int ZEBU_LocalTraceImporterGroup_getNumberOfTracers (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*)

get the number of tracers added in the group

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

Return values:

int

6.11.1.33 const char ∗ ZEBU_LocalTraceImporterGroup_getPath (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*)

get the tracer <RPATH>

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

Return values:

tracer <RPATH>

6.11.1.34 int ZEBU_LocalTraceImporterGroup_initialize (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, ZEBU_Board ∗ *board*, const char ∗ *importName*, const int *onEvent*, const int *thread*, const int *synchronous*)

initialize the tracer

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

board handler on Board

importName name of the C function to be called. If NULL, load the C function which name matchs with the RNAME of the tracer. It is possible to use the same C function for several tracers. The C function can call the followings System Verilog functions: svGetNameFromScope: return the <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined svPutUserData: save a user pointer for the tracer scope svGetUserData: return the user pointer saved for the tracer scope

onEvent if 0 imported C function is called for each trace cycle at which the tracer is enabled by software and hardware pin else if 1 imported C function is called only when software receive event on tracer's values

thread specify on which thread must be run the tracer. If -1 no specification.

synchronous if true imported C function calls are synchronized with other synchronous importers of the same thread according to timestamps of value changes

Returns:

status

Return values:

0 OK

>***0*** KO

6.11.1.35 int ZEBU_LocalTraceImporterGroup_initialize_userData (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, ZEBU_Board ∗ *board*, void ∗ *userData*, const char ∗ *importName*, const int *onEvent*, const int *thread*, const int *synchronous*)

initialize the object with a user pointer

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

board C++ handler on Board

userData specify the user pointer.to the first argument of the C function. Tracer'signal values are set from the second argument and so on.

importName name of the C function to be called with the user data. If NULL, load the C function which name matchs with the RNAME of the tracer. It is possible to use the same C function for several tracers.

onEvent if 1 imported C function is called for each trace cycle at which the tracer is enabled by software and hardware pin else if 0 imported C function is called only when software receive event on tracer's values

thread specify on which thread must be run the tracer. If -1 no specification.

synchronous if true imported C function calls are synchronized with other synchronous importers of the same thread according to timestamps of value changes

Returns:

status

Return values:

0 OK

*>**0*** KO

6.11.1.36 int ZEBU_LocalTraceImporterGroup_isEnabled (ZEBU_LocalTraceImporterGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get tracer state

Parameters:

tracerGroup ZEBU_LocalTraceImporterGroup handler

tracerIdentifier tracer identifier

Return values:

1 if tracer is enabled, else 0

6.11.1.37 ZEBU_LocalTraceImporterGroup ∗ ZEBU_LocalTraceImporterGroup_new ()

allocate a ZEBU_LocalTraceImporterGroup

Return values:

ZEBU_LocalTracImporterGroup∗

6.11.1.38 void ZEBU_LocalTraceReaderGroup_add (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const char ∗ *fullname*)

add a tracer to the group

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

fullname hierarchical instanciation name of the tracer by default or <RPATH>.<RNAME> of the tracer or just <RNAME> of the tracer if <RPATH> is undefined

Return values:

identifier of the tracer in the group

6.11.1.39 void ZEBU_LocalTraceReaderGroup_add_regexp (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const char ∗ *regularExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*)

add tracers in the group from a regular expression. The regular expression specifies the names of tracers to add in the group

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

regularExpression regular expression

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

Example of regular expressions:

List of names T1 L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2 L0.L1.L2.T3 L0.L1.L2.T4

Result of the regular expression = "T1" T1

Result of the regular expression = "T2" L0.T2 T2.L0 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "T2$" L0.T2 L0.L1.T2 L0.L1.L2.T2

Result of the regular expression = "L0\\.L1\\.L2\\..∗[^2]$" L0.L1.L2.T3 L0.L1.L2.T4

6.11.1.40 void ZEBU_LocalTraceReaderGroup_delete (ZEBU_LocalTraceReaderGroup ∗ *group*)

delete a ZEBU_LocalTraceReaderGroup

Parameters:

group ZEBU_LocalTraceReaderGroup handler

Returns:

status

Return values:

0 OK

>0 KO

6.11.1.41 void ZEBU_LocalTraceReaderGroup_disable (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

disable the tracer

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

timestamp at which the tracer has been disabled

ULLONG_MAX on error

6.11.1.42 ZEBU_LocalTraceReaderGroup_disableAll (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*)

diable all tracers of the group

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

timestamp at which the tracers have been disabled

ULLONG_MAX on error

6.11.1.43 long long unsigned ZEBU_LocalTraceReaderGroup_enable (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

enable the tracer

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

timestamp at which the tracer has been enabled

ULLONG_MAX on error

6.11.1.44 ZEBU_LocalTraceReaderGroup_enableAll (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*)

enable all tracers of the group

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

timestamp at which the tracers have been enabled

ULLONG_MAX on error

6.11.1.45 void ZEBU_LocalTraceReaderGroup_getFullname (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer full name : <RPATH>.<RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

tracer full name

6.11.1.46 ZEBU_LocalTraceReaderGroup_getIdentifier (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const char ∗ *fullname*)

get identifier of the tracer

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

fullname full name of the tracer

Return values:

int

6.11.1.47 const char * ZEBU_LocalTraceReaderGroup_getName (ZEBU_LocalTraceReaderGroup * *tracerGroup*, const int *tracerIdentifier*)

get the tracer <RNAME>

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

tracer <RNAME>

6.11.1.48 int ZEBU_LocalTraceReaderGroup_getNextChange (ZEBU_LocalTraceReaderGroup * *tracerGroup*)

try to get next value change

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

0 if sucessfull

1 if the next value change is not available yet

-1 if tracer is disabled and if the end of trace has been reached

6.11.1.49 ZEBU_LocalTraceReaderGroup_getNumberOfTracers (ZEBU_LocalTraceReaderGroup * *tracerGroup*)

get the number of tracers added in the group

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

int

6.11.1.50 const char ∗ ZEBU_LocalTraceReaderGroup_getPath (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const int *tracerIdentifier*)

get the tracer <RPATH>

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

tracer <RPATH>

6.11.1.51 ZEBU_Signal ∗ ZEBU_LocalTraceReaderGroup_getSignal (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const int *tracerIdentifier*, const char ∗ *name*)

get a signal handler to read its value

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier identifier of the tracer in the group returned by the add function

name of the signal

Return values:

handler on a llocal tracer signal

6.11.1.52 long long unsigned ZEBU_LocalTraceReaderGroup_getTime (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*)

get the current timestamp of the tracer The timestamp is the number of cycles generated for the clock synchronizing all tracers corresponding with the current values of tracer's signals

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

current timestamp of the tracer

6.11.1.53 int ZEBU_LocalTraceReaderGroup_initialize (ZEBU_LocalTraceReaderGroup $*$ *tracerGroup*, ZEBU_Board $*$ *board*, const int *thread*)

initialize the object

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

board C++ handler on Board

thread specify on which thread must be run the tracer. If -1 no specification.

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.11.1.54 void ZEBU_LocalTraceReaderGroup_isEnabled (ZEBU_LocalTraceReaderGroup $*$ *tracerGroup*, const int *tracerIdentifier*)

get tracer state

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

tracerIdentifier tracer identifier

Return values:

1 if tracer is enabled, else 0

6.11.1.55 ZEBU_LocalTraceReaderGroup $*$ ZEBU_LocalTraceReaderGroup_-new ()

allocate a ZEBU_LocalTraceReaderGroup

Return values:

ZEBU_LocalTracReaderGroup*$*$*

6.11.1.56 int ZEBU_LocalTraceReaderGroup_run (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const long long unsigned ∗ *numberOfCycles*)

read the nth next trace cycles. Block until the nth next trace cycles are received.

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

numberOfCycles number of cycles to run

Return values:

0 if sucessfull

-1 if tracer is disabled and if the end of trace has been reached

6.11.1.57 int ZEBU_LocalTraceReaderGroup_step (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*)

read the next trace cyclea Block until the next trace cycle is received.

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

0 if sucessfull

-1 if tracer is disabled and if the end of trace has been reached

6.11.1.58 int ZEBU_LocalTraceReaderGroup_tryRun (ZEBU_LocalTraceReaderGroup ∗ *tracerGroup*, const long long unsigned ∗ *numberOfCycles*)

try to read the nth next trace cycles

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

numberOfCycles number of cycles to try to run

Return values:

the number of read cycles

-1 if tracer is disabled and if the end of trace has been reached

6.11.1.59 int ZEBU_LocalTraceReaderGroup_tryStep (ZEBU_LocalTraceReaderGroup * *tracerGroup*)

try to read the next trace cycle

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

0 if sucessfull

1 if the next trace cycle is not available yet

-1 if tracer is disabled and if the end of trace has been reached

6.11.1.60 int ZEBU_LocalTraceReaderGroup_waitNextChange (ZEBU_LocalTraceReaderGroup * *tracerGroup*)

wait for next value change Block until the nth next value change is received.

Parameters:

tracerGroup ZEBU_LocalTraceReaderGroup handler

Return values:

0 if sucessfull

-1 if tracer is disabled and if the end of trace has been reached

6.12 ZEBU_LogicAnalyzer.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned int ZEBU_LogicAnalyzer_destroy (ZEBU_LogicAnalyzer ∗la)

  destroy the logic analyzer

- unsigned int ZEBU_LogicAnalyzer_start (ZEBU_LogicAnalyzer ∗la, const char ∗clockName, const char ∗edgeName)

  start the logic analyzer

- unsigned int ZEBU_LogicAnalyzer_stop (ZEBU_LogicAnalyzer ∗la)

  stop the logic analyzer

- unsigned int ZEBU_LogicAnalyzer_stopOnTrigger (ZEBU_LogicAnalyzer ∗la, const char ∗triggerName)

  program the logic analyzer to stop clocks on trigger

- unsigned int ZEBU_LogicAnalyzer_traceOnTrigger (ZEBU_LogicAnalyzer ∗la, const char ∗triggerName)

  program the logic analyzer to trace on trigger

6.12.1 Function Documentation

6.12.1.1 unsigned int ZEBU_LogicAnalyzer_destroy (ZEBU_LogicAnalyzer ∗ *la*)

destroy the logic analyzer

Parameters:

la ZEBU_LogicAnalyzer handler

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.12.1.2 unsigned int ZEBU_LogicAnalyzer_start (ZEBU_LogicAnalyzer ∗ *la*, const char ∗ *clockName*, const char ∗ *edgeName*)

start the logic analyzer

Parameters:

la logic analyzer handler

clockName name of the clock used to change logic analyzer state

edgeName name of the edge used to change logic analyzer state

See also:

ZEBU_LogicAnalyzer_stop
ZEBU_LogicAnalyzer_traceOnTrigger
ZEBU_LogicAnalyzer_stopOnTrigger

6.12.1.3 unsigned int ZEBU_LogicAnalyzer_stop (ZEBU_LogicAnalyzer ∗ *la*)

stop the logic analyzer

Parameters:

la logic analyzer handler

See also:

ZEBU_LogicAnalyzer_stop
ZEBU_LogicAnalyzer_traceOnTrigger
ZEBU_LogicAnalyzer_stopOnTrigger

6.12.1.4 unsigned int ZEBU_LogicAnalyzer_stopOnTrigger (ZEBU_LogicAnalyzer ∗ *la*, const char ∗ *triggerName*)

program the logic analyzer to stop clocks on trigger

Parameters:

la logic analyzer handler

triggerName name of the trigger on which to stop

See also:

ZEBU_LogicAnalyzer_stop
ZEBU_LogicAnalyzer_start

6.12.1.5 unsigned int ZEBU_LogicAnalyzer_traceOnTrigger (ZEBU_LogicAnalyzer ∗ *la*, const char ∗ *triggerName*)

program the logic analyzer to trace on trigger

Parameters:

la logic analyzer handler

triggerName name of the trigger on which to stop

See also:

ZEBU_LogicAnalyzer_stop
ZEBU_LogicAnalyzer_start

6.13 ZEBU_LoopDetector.h File Reference

This graph shows which files directly or indirectly include this file:

ZEBU_APosteriori A Posteriori Loop Detection

The ZEBU_APosteriori∗ API provides several methods to work with the combinational loop detector in the "a posteriori" detection mode.

Note:

All the functions implemented in this class should only be used if the "a posteriori" detection mode has been enabled at compile-time.

```
// Run nbCycles
ZEBU_Driver_run(driver, nbCycle);

// Test if an oscillation has been detected during the last run
if (ZEBU_APosterioriLoopDetector_checkDetectors(board) > 0)
{
    // Create a new oscillating loop iterator
    ZEBU_APosterioriIterator* loopIterator = ZEBU_APosterioriIterator_Create(board);

    // Print the name of each oscillating loop
    for (ZEBU_APosterioriIterator_goToFirst(loopIterator); ZEBU_APosterioriIterator_isAtEnd
        printf("oscillating loop name = %s\n", ZEBU_APosterioriIterator_getName(loopIterato
    }

    // Destroy the loop iterator
    ZEBU_APosterioriIterator_Destroy(loopIterator);

    // Reset the detectors
    ZEBU_APosterioriLoopDetector_resetDetectors(board);
}
```

- unsigned int ZEBU_APosterioriLoopDetector_checkDetectors (ZEBU_Board ∗board)

  Tests if an oscillating loop has been detected during the last run.

- unsigned int ZEBU_APosterioriLoopDetector_resetDetectors (ZEBU_Board ∗board)

  Resets the combinational loop detectors.

- ZEBU_APosterioriIterator ∗ ZEBU_APosterioriIterator_Create (ZEBU_Board ∗board)

  Constructs and initializes a new `ZEBU_APosterioriIterator` *instance.*

- unsigned int ZEBU_APosterioriIterator_Destroy (ZEBU_APosterioriIterator ∗iterator)

Destroys a `ZEBU_APosterioriIterator` *instance.*

- unsigned int ZEBU_APosterioriIterator_goToFirst (ZEBU_APosterioriIterator ∗iterator)

  Moves iterator to the first oscillating loop.

- unsigned int ZEBU_APosterioriIterator_goToNext (ZEBU_APosterioriIterator ∗iterator)

  Moves iterator to the next oscillating loop.

- unsigned int ZEBU_APosterioriIterator_isAtEnd (ZEBU_APosterioriIterator ∗iterator)

  Tests if iterator passed last oscillating loop.

- const char ∗ ZEBU_APosterioriIterator_getName (ZEBU_APosterioriIterator ∗iterator)

ZEBU_Interactive Interactive Loop Detection

The ZEBU_Interactive∗ API provides several methods to work with the combinational loop detector in the "interactive" detection mode.

Note:

All the functions implemented in this class should only be used if the "interactive" detection mode has been enabled at compile-time.

- unsigned int ZEBU_InteractiveLoopDetector_enable (ZEBU_Board ∗board, const char ∗loopPath)

  Enables an interactive combinational loop detector.

- unsigned int ZEBU_InteractiveLoopDetector_disable (ZEBU_Board ∗board, const char ∗loopPath)

  Disables an interactive combinational loop detector.

- int ZEBU_InteractiveLoopDetector_waitDriver (ZEBU_Board ∗board, ZEBU_-Driver ∗driver, unsigned int timeout)

  Waits for any interactive combinational loop detection -or- a timeout while running the clock through the C cosimulation-driver.

- int ZEBU_InteractiveLoopDetector_waitDriverEx (ZEBU_Board ∗board, ZEBU_Driver ∗driver, unsigned int triggers, unsigned int ∗fired, unsigned int timeout)

Waits for any interactive combinational loop detection -or- another trigger -or- a timeout while running the clock through the C cosimulation-driver.

- unsigned int ZEBU_InteractiveLoopDetector_enableGlobalCallback (ZEBU_-Board ∗board, ZEBU_Driver ∗driver)

  Tells the system to call the global callback registered through `ZEBU_-Callback::Register` *whenever an oscillating loop is detected.*

- unsigned int ZEBU_InteractiveLoopDetector_disableGlobalCallback (ZEBU_-Board ∗board, ZEBU_Driver ∗driver)

  Tells the system not to call the global callback registered through `ZEBU_-Callback::Register` *if an oscillating loop is detected.*

- ZEBU_InteractiveIterator ∗ ZEBU_InteractiveIterator_Create (ZEBU_Board ∗board)

  Constructs and initializes a new `ZEBU_InteractiveLoopDetector_-Iterator` *instance.*

- unsigned int ZEBU_InteractiveIterator_Destroy (ZEBU_InteractiveIterator ∗iterator)

  Destroys a `ZEBU_InteractiveLoopDetector_Iterator` *instance.*

- unsigned int ZEBU_InteractiveIterator_goToFirst (ZEBU_InteractiveIterator ∗iterator)

  Moves iterator to the first oscillating loop.

- void ZEBU_InteractiveIterator_goToNext (ZEBU_InteractiveIterator ∗iterator)

  Moves iterator to the next oscillating loop.

- unsigned int ZEBU_InteractiveIterator_isAtEnd (ZEBU_InteractiveIterator ∗iterator)

  Tests if iterator passed last oscillating loop.

- const char ∗ ZEBU_InteractiveIterator_getName (ZEBU_InteractiveIterator ∗iterator)

ZEBU_LoopBreak Interactive Loop Detection

The ZEBU_LoopBreak∗ API provides methods to deal with in-cycle oscillating loop breaking.

Note:

All the functions implemented in this class should only be used if the "loop break" mode has been enabled at compile-time.

- unsigned int ZEBU_LoopBreak_setInjectedValue (ZEBU_Board ∗board, const char ∗loopPath, unsigned int value)

  Sets the value used by the system to break an oscillating combinational loop.

- unsigned int ZEBU_LoopBreak_alwaysBreak (ZEBU_Board ∗board, const char ∗loopPath, unsigned int value)

  Sets the detector in a mode where the user value is always injected in the combinational loop, or only when an oscillation was detected.

Typedefs

- typedef ZEBU_APosterioriIterator ZEBU_APosterioriIterator
- typedef ZEBU_InteractiveIterator ZEBU_InteractiveIterator

6.13.1 Typedef Documentation

6.13.1.1 typedef struct ZEBU_APosterioriIterator ZEBU_APosterioriIterator

6.13.1.2 typedef struct ZEBU_InteractiveIterator ZEBU_InteractiveIterator

6.13.2 Function Documentation

6.13.2.1 ZEBU_APosterioriIterator∗ ZEBU_APosterioriIterator_Create (ZEBU_Board ∗ *board*)

Constructs and initializes a new `ZEBU_APosterioriIterator` instance.

Parameters:

board A pointer to a `ZEBU_Board` object.

Returns:

A handler to a `ZEBU_APosterioriIterator` object or NULL if an error has occured.

See also:

ZEBU_APosterioriIterator_Destroy

6.13.2.2 unsigned int ZEBU_APosterioriIterator_Destroy (ZEBU_APosterioriIterator ∗ *iterator*)

Destroys a `ZEBU_APosterioriIterator` instance.

Returns:
Error status

Return values:
- ***0*** The operation has completed successfully.
- ***>0*** An error has occured.

See also:
ZEBU_APosterioriIterator_Create

6.13.2.3 const char∗ ZEBU_APosterioriIterator_getName (ZEBU_APosterioriIterator ∗ *iterator*)

6.13.2.4 unsigned int ZEBU_APosterioriIterator_goToFirst (ZEBU_APosterioriIterator ∗ *iterator*)

Moves iterator to the first oscillating loop.

Returns:
Error status

Return values:
- ***0*** The operation has completed successfully.
- ***>0*** An error has occured.

See also:
ZEBU_APosterioriIterator_goToNext
ZEBU_APosterioriIterator_isAtEnd

6.13.2.5 unsigned int ZEBU_APosterioriIterator_goToNext (ZEBU_APosterioriIterator ∗ *iterator*)

Moves iterator to the next oscillating loop.

Returns:
Error status

Return values:
- ***0*** The operation has completed successfully.
- ***>0*** An error has occured.

See also:
ZEBU_APosterioriIterator_goToFirst
ZEBU_APosterioriIterator_isAtEnd

6.13.2.6 unsigned int ZEBU_APosterioriIterator_isAtEnd (ZEBU_APosterioriIterator ∗ *iterator*)

Tests if iterator passed last oscillating loop.

Returns:

An integer value indicating if the iterator is at the end.

See also:

ZEBU_APosterioriIterator_goToFirst
ZEBU_APosterioriIterator_goToNext

6.13.2.7 unsigned int ZEBU_APosterioriLoopDetector_checkDetectors (ZEBU_Board ∗ *board*)

Tests if an oscillating loop has been detected during the last run.

Parameters:

board A pointer to a `ZEBU_Board` object.

Returns:

A integer value indicating whether an oscillating loop has been detected or not.

Return values:

0 No oscillated loop detected.

!=0 At least one oscillated loop has been detected. Use a `ZEBU_-APosterioriIterator` object to enumerate the oscillated loops.

See also:

ZEBU_APosterioriLoopDetector_resetDetectors
ZEBU_APosterioriIterator_Create

6.13.2.8 unsigned int ZEBU_APosterioriLoopDetector_resetDetectors (ZEBU_Board ∗ *board*)

Resets the combinational loop detectors.

Parameters:

board A pointer to a `ZEBU_Board` object.

Returns:

Error status

Return values:

0 The operation has completed successfully.

>*0* An error has occured.

See also:

ZEBU_APosterioriLoopDetector_checkDetectors

6.13.2.9 ZEBU_InteractiveIterator$*$ ZEBU_InteractiveIterator_Create (ZEBU_Board $*$ *board*)

Constructs and initializes a new `ZEBU_InteractiveLoopDetector_-Iterator` instance.

Parameters:

board A pointer to a `ZEBU_Board` object.

Returns:

A handler to a `ZEBU_InteractiveIterator` object or NULL if an error has occured.

See also:

ZEBU_InteractiveIterator_Destroy

6.13.2.10 unsigned int ZEBU_InteractiveIterator_Destroy (ZEBU_InteractiveIterator $*$ *iterator*)

Destroys a `ZEBU_InteractiveLoopDetector_Iterator` instance.

Returns:

Error status

Return values:

0 The operation has completed successfully.

>*0* An error has occured.

See also:

ZEBU_InteractiveIterator_Create

6.13.2.11 const char∗ ZEBU_InteractiveIterator_getName (ZEBU_InteractiveIterator ∗ *iterator*)

6.13.2.12 unsigned int ZEBU_InteractiveIterator_goToFirst (ZEBU_InteractiveIterator ∗ *iterator*)

Moves iterator to the first oscillating loop.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_Iterator::goToNext
ZEBU_InteractiveLoopDetector_Iterator::isAtEnd

6.13.2.13 void ZEBU_InteractiveIterator_goToNext (ZEBU_InteractiveIterator ∗ *iterator*)

Moves iterator to the next oscillating loop.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_Iterator::goToFirst
ZEBU_InteractiveLoopDetector_Iterator::isAtEnd

6.13.2.14 unsigned int ZEBU_InteractiveIterator_isAtEnd (ZEBU_InteractiveIterator ∗ *iterator*)

Tests if iterator passed last oscillating loop.

Returns:

An integer value indicating if the iterator is at the end.

See also:

ZEBU_InteractiveLoopDetector_Iterator::goToFirst
ZEBU_InteractiveLoopDetector_Iterator::goToNext

6.13.2.15 unsigned int ZEBU_InteractiveLoopDetector_disable (ZEBU_Board ∗ *board*, const char ∗ *loopPath*)

Disables an interactive combinational loop detector.

Parameters:

board A pointer to a `ZEBU_Board` object.

loopPath The full path of the signal which identifies the interactive loop detector to enable. If NULL, all interactive loop detectors are disabled.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_enable
ZEBU_InteractiveLoopDetector_waitDriver
ZEBU_InteractiveLoopDetector_waitDriverEx

6.13.2.16 unsigned int ZEBU_InteractiveLoopDetector_disableGlobalCallback (ZEBU_Board ∗ *board*, ZEBU_Driver ∗ *driver*)

Tells the system not to call the global callback registered through `ZEBU_-Callback::Register` if an oscillating loop is detected.

Parameters:

board A pointer to a `ZEBU_Board` object.

driver A pointer to the `ZEBU_Driver` object that runs the clock.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_enableGlobalCallback
ZEBU_Events_Register

6.13.2.17 unsigned int ZEBU_InteractiveLoopDetector_enable (ZEBU_Board ∗ *board*, const char ∗ *loopPath*)

Enables an interactive combinational loop detector.

Parameters:

board A pointer to a `ZEBU_Board` object.

loopPath The full path of the signal which identifies the interactive loop detector to enable. If NULL, all interactive loop detectors are enabled.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_disable
ZEBU_InteractiveLoopDetector_waitDriver
ZEBU_InteractiveLoopDetector_waitDriverEx

6.13.2.18 unsigned int ZEBU_InteractiveLoopDetector_enableGlobalCallback (ZEBU_Board ∗ *board*, ZEBU_Driver ∗ *driver*)

Tells the system to call the global callback registered through `ZEBU_-Callback::Register` whenever an oscillating loop is detected.

Parameters:

board A pointer to a `ZEBU_Board` object.

driver A pointer to the `ZEBU_Driver` object that runs the clock.

Returns:

Error status

Return values:

0 The operation has completed successfully.

$>$***0*** An error has occured.

See also:

ZEBU_InteractiveLoopDetector_disableGlobalCallback
ZEBU_Events_Register

6.13.2.19 int ZEBU_InteractiveLoopDetector_waitDriver (ZEBU_Board $*$ *board*, ZEBU_Driver $*$ *driver*, unsigned int *timeout*)

Waits for any interactive combinational loop detection -or- a timeout while running the clock through the C cosimulation-driver.

Parameters:

board A pointer to a `ZEBU_Board` object.

driver A pointer to the `ZEBU_Driver` object that runs the clock.

timeout Maximum number of cycles to run.

Returns:

An integer indicating the status.

Return values:

0 The timeout has expired.

$>$***0*** At least one combinationnal loop has been detected.

See also:

ZEBU_InteractiveLoopDetector_enable
ZEBU_InteractiveLoopDetector_disable
ZEBU_InteractiveLoopDetector_waitDriverEx
ZEBU_InteractiveIterator_Create

6.13.2.20 int ZEBU_InteractiveLoopDetector_waitDriverEx (ZEBU_Board $*$ *board*, ZEBU_Driver $*$ *driver*, unsigned int *triggers*, unsigned int $*$ *fired*, unsigned int *timeout*)

Waits for any interactive combinational loop detection -or- another trigger -or- a time-out while running the clock through the C cosimulation-driver.

Parameters:

board A pointer to a `ZEBU_Board` object.

driver A pointer to the `ZEBU_Driver` object that runs the clock.

triggers Other triggers to stop on. Set bit i to stop on trigger i (on the 16 lsb).

$\rightarrow$ ***fired*** Fired triggers. Bit i is set for trigger i.

timeout Maximum number of cycles to run.

Returns:

An integer indicating the status.

Return values:

0 The timeout has expired.

>***0*** At least one combinationnal loop has been detected.

<***0*** Another trigger has fired.

Remarks:

Even if this method returns >0, which means that the loop detector trigger has fired, you should also check the `fired` out parameter for another fired trigger.

See also:

ZEBU_InteractiveLoopDetector_enable
ZEBU_InteractiveLoopDetector_disable
ZEBU_InteractiveLoopDetector_waitDriver
ZEBU_InteractiveIterator_Create

6.13.2.21 unsigned int ZEBU_LoopBreak_alwaysBreak (ZEBU_Board ∗ *board*, const char ∗ *loopPath*, unsigned int *value*)

Sets the detector in a mode where the user value is always injected in the combinational loop, or only when an oscillation was detected.

Parameters:

board A pointer to a `ZEBU_Board` object.

loopPath The full path of the signal which identifies the loop detector to modify. If NULL, all loop detectors are modified.

value Always inject the value in the loop.

Returns:

Error status

Return values:

0 The operation has completed successfully.

>***0*** An error has occured.

6.13.2.22 unsigned int ZEBU_LoopBreak_setInjectedValue (ZEBU_Board * *board*, const char * *loopPath*, unsigned int *value*)

Sets the value used by the system to break an oscillating combinational loop.

Parameters:

board A pointer to a `ZEBU_Board` object.

loopPath The full path of the signal which identifies the loop detector to modify. If NULL, all loop detectors are modified.

value The value that will be injected in the loop if an oscillation is detected.

Returns:

Error status

Return values:

0 The operation has completed successfully.

*>**0*** An error has occured.

6.14 ZEBU_Memory.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void ZEBU_Memory_set (ZEBU_Memory ∗memory, unsigned int pattern)

  set the content of the memory with a pattern

- void ZEBU_Memory_clear (ZEBU_Memory ∗memory)

  clear the content of the memory

- void ZEBU_Memory_erase (ZEBU_Memory ∗memory)

  erase the content of the memory

- char ∗ ZEBU_Memory_name (ZEBU_Memory ∗memory)

  get the name of the memory

- char ∗ ZEBU_Memory_fullname (ZEBU_Memory ∗memory, char separator)

  get the hierarchical name of the memory

- unsigned int ZEBU_Memory_depth (ZEBU_Memory ∗memory)

  get the depth of the memory

- unsigned int ZEBU_Memory_width (ZEBU_Memory ∗memory)

  get the width of the memory

- unsigned int ∗ ZEBU_Memory_newBuffer (ZEBU_Memory ∗memory)

  get a new buffer which has size of memory

- unsigned int ∗ ZEBU_Memory_newWordBuffer (ZEBU_Memory ∗memory)

  get a new buffer which has size of a memory word

- unsigned int ZEBU_Memory_loadFromFile (ZEBU_Memory ∗memory, const char ∗filename)

  load the content of a file into the memory

- unsigned int ZEBU_Memory_readmemb (ZEBU_Memory ∗memory, const char ∗filename)

  Load the content of a human readable binary file into the memory.

- unsigned int ZEBU_Memory_readmemh (ZEBU_Memory ∗memory, const char ∗filename)

    Load the content of a human readable hexadecimal file into the memory.

- unsigned int ZEBU_Memory_writememb (ZEBU_Memory ∗memory, const char ∗filename, unsigned int startAddress, unsigned int endAddress)

    Store the content of a memory to a human readable file in binary data format.

- unsigned int ZEBU_Memory_writememh (ZEBU_Memory ∗memory, const char ∗filename, unsigned int startAddress, unsigned int endAddress)

    Store the content of a memory to a human readable file in hexadecimal data format.

- unsigned int ZEBU_Memory_storeToFile (ZEBU_Memory ∗memory, const char ∗filename, unsigned int startAdd, unsigned int endAdd)

    copy the memory content into a file

- unsigned int ZEBU_Memory_storeToBinFile (ZEBU_Memory ∗memory, const char ∗filename, unsigned int startAdd, unsigned int endAdd)

    copy the memory content into a binary file

- unsigned int ZEBU_Memory_storeToHexFile (ZEBU_Memory ∗memory, const char ∗filename, const unsigned int firstAddress, const unsigned int lastAddress, const int useAddressRange, const int dumpZero)

    copy the memory content into a hexadecimal format file

- unsigned int ZEBU_Memory_storeToRawFile (ZEBU_Memory ∗memory, const char ∗filename, unsigned int startAdd, unsigned int endAdd, unsigned int compressionLevel)

    copy the memory content into a binary file

- unsigned int ZEBU_Memory_storeToBuffer (ZEBU_Memory ∗memory, unsigned int ∗buffer)

    copy the memory content into a buffer

- unsigned int ZEBU_Memory_storeToBuffer2 (ZEBU_Memory ∗memory, unsigned int ∗buffer, unsigned int firstAddress, unsigned int lastAddress)

    copy the memory content into a buffer

- unsigned int ZEBU_Memory_loadFromBuffer (ZEBU_Memory ∗memory, const unsigned int ∗buffer)

    copy the buffer into the memory

- unsigned int ZEBU_Memory_loadFromBuffer2 (ZEBU_Memory ∗memory, const unsigned int ∗buffer, unsigned int firstAddress, unsigned int lastAddress) *copy the buffer into the memory*

- unsigned int ZEBU_Memory_readWord (ZEBU_Memory ∗memory, unsigned int address, unsigned int ∗word_buffer)

  read a word of the memory into a buffer

- unsigned int ZEBU_Memory_writeWord (ZEBU_Memory ∗memory, unsigned int address, const unsigned int ∗word_buffer)

  write a buffer into a memory word

- void ZEBU_Memory_delete (ZEBU_Memory ∗memory)

  delete the memory object

6.14.1 Function Documentation

6.14.1.1 void ZEBU_Memory_clear (ZEBU_Memory ∗ *memory*)

clear the content of the memory

Parameters:

memory ZEBU_memory handler

See also:

ZEBU_Memory_set
ZEBU_Memory_erase

6.14.1.2 void ZEBU_Memory_delete (ZEBU_Memory ∗ *memory*)

delete the memory object

Parameters:

memory ZEBU_memory handler

6.14.1.3 unsigned int ZEBU_Memory_depth (ZEBU_Memory ∗ *memory*)

get the depth of the memory

Parameters:

memory ZEBU_memory handler

Returns:
unsigned int

Return values:

depth of the memory

6.14.1.4 void ZEBU_Memory_erase (ZEBU_Memory ∗ *memory*)

erase the content of the memory

Parameters:

memory ZEBU_memory handler

See also:
ZEBU_Memory_set
ZEBU_Memory_clear

6.14.1.5 char ∗ ZEBU_Memory_fullname (ZEBU_Memory ∗ *memory*, char *separator*)

get the hierarchical name of the memory

Parameters:

memory ZEBU_memory handler

separator hierarchical separator character

Returns:
char ∗

Return values:

name of the memory

6.14.1.6 unsigned int ZEBU_Memory_loadFromBuffer (ZEBU_Memory ∗ *memory*, const unsigned int ∗ *buffer*)

copy the buffer into the memory

Parameters:

memory ZEBU_memory handler

buffer pointer to the buffer

Returns:

unsigned int

Return values:

0	OK
$>$***0***	KO

See also:

ZEBU_Memory_newBuffer

6.14.1.7 unsigned int ZEBU_Memory_loadFromBuffer2 (ZEBU_Memory $*$ *memory*, const unsigned int $*$ *buffer*, unsigned int *firstAddress*, unsigned int *lastAddress*)

copy the buffer into the memory

Parameters:

memory	ZEBU_memory handler
buffer	pointer to the buffer
firstAddress	first address to be loaded
lastAddress	last address to be loaded

Returns:

unsigned int

Return values:

0	OK
$>$***0***	KO

See also:

ZEBU_Memory_newBuffer

6.14.1.8 unsigned int ZEBU_Memory_loadFromFile (ZEBU_Memory $*$ *memory*, const char $*$ *filename*)

load the content of a file into the memory

Parameters:

memory	ZEBU_memory handler
filename	name of file to load

Returns:

int

6.14.1.9 char ∗ ZEBU_Memory_name (ZEBU_Memory ∗ *memory*)

get the name of the memory

Parameters:

memory ZEBU_memory handler

Returns:

char ∗

Return values:

name of the memory

6.14.1.10 unsigned int ∗ ZEBU_Memory_newBuffer (ZEBU_Memory ∗ *memory*)

get a new buffer which has size of memory

Parameters:

memory ZEBU_memory handler the size of the buffer computed with formula : memory_word_width ∗ memory_depth

Returns:

unsigned int ∗

Return values:

buffer

6.14.1.11 unsigned int ∗ ZEBU_Memory_newWordBuffer (ZEBU_Memory ∗ *memory*)

get a new buffer which has size of a memory word

Parameters:

memory ZEBU_memory handler the size of the buffer computed with formula : memory_word_width ∗ 1

Returns:

unsigned int ∗

Return values:

buffer

6.14.1.12 unsigned int ZEBU_Memory_readmemb (ZEBU_Memory $*$ *memory*, const char $*$ *filename*)

Load the content of a human readable binary file into the memory.

Parameters:

memory ZEBU_memory handler

filename Path to the file to load.

Return values:

0 is success

$>$***1*** otherwise.

6.14.1.13 unsigned int ZEBU_Memory_readmemh (ZEBU_Memory $*$ *memory*, const char $*$ *filename*)

Load the content of a human readable hexadecimal file into the memory.

Parameters:

memory ZEBU_memory handler

filename Path to the file to load.

Return values:

0 is success

$>$***1*** otherwise.

6.14.1.14 unsigned int ZEBU_Memory_readWord (ZEBU_Memory $*$ *memory*, unsigned int *address*, unsigned int $*$ *word_buffer*)

read a word of the memory into a buffer

Parameters:

memory ZEBU_memory handler

address address to read

word_buffer pointer to the buffer

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

See also:

ZEBU_Memory_newWordBuffer
ZEBU_Memory_writeWord

6.14.1.15 void ZEBU_Memory_set (ZEBU_Memory * *memory*, unsigned int *pattern*)

set the content of the memory with a pattern

Parameters:

memory ZEBU_memory handler

pattern

See also:

ZEBU_Memory_clear
ZEBU_Memory_erase

6.14.1.16 unsigned int ZEBU_Memory_storeToBinFile (ZEBU_Memory * *memory*, const char * *filename*, unsigned int *startAdd*, unsigned int *endAdd*)

copy the memory content into a binary file

Parameters:

memory ZEBU_memory handler

filename name of the memory file

startAdd first address to be dumped

endAdd last address to be dumped

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.14.1.17 unsigned int ZEBU_Memory_storeToBuffer (ZEBU_Memory ∗ *memory*, unsigned int ∗ *buffer*)

copy the memory content into a buffer

Parameters:

memory ZEBU_memory handler

buffer pointer to the buffer

Returns:

unsigned int

Return values:

0 OK

>***0*** KO

See also:

ZEBU_Memory_newBuffer

6.14.1.18 unsigned int ZEBU_Memory_storeToBuffer2 (ZEBU_Memory ∗ *memory*, unsigned int ∗ *buffer*, unsigned int *firstAddress*, unsigned int *lastAddress*)

copy the memory content into a buffer

Parameters:

memory ZEBU_memory handler

buffer pointer to the buffer

firstAddress first address to be dumped

lastAddress last address to be dumped

Returns:

unsigned int

Return values:

0 OK

>***0*** KO

See also:

ZEBU_Memory_newBuffer

6.14.1.19 unsigned int ZEBU_Memory_storeToFile (ZEBU_Memory $*$ *memory*, const char $*$ *filename*, unsigned int *startAdd*, unsigned int *endAdd*)

copy the memory content into a file

Parameters:

memory	ZEBU_memory handler
filename	name of the memory file
startAdd	first address to be dumped
endAdd	last address to be dumped

Returns:

unsigned int

Return values:

0	OK
>***0***	KO

6.14.1.20 unsigned int ZEBU_Memory_storeToHexFile (ZEBU_Memory $*$ *memory*, const char $*$ *filename*, const unsigned int *firstAddress*, const unsigned int *lastAddress*, const int *useAddressRange*, const int *dumpZero*)

copy the memory content into a hexadecimal format file

Parameters:

memory	ZEBU_memory handler
filename	name of the memory file
firstAddress	first address to be dumped
lastAddress	last address to be dumped
useAddressRange	specify if address range can be used
dumpZero	specify if null memory words must be dumped

Returns:

unsigned int

Return values:

0	OK
>***0***	KO

6.14.1.21 unsigned int ZEBU_Memory_storeToRawFile (ZEBU_Memory ∗ *memory*, const char ∗ *filename*, unsigned int *startAdd*, unsigned int *endAdd*, unsigned int *compressionLevel*)

copy the memory content into a binary file

Parameters:

memory ZEBU_memory handler

filename name of the memory file

startAdd first address to be dumped

endAdd last address to be dumped

compressionLevel level of compression

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.14.1.22 unsigned int ZEBU_Memory_width (ZEBU_Memory ∗ *memory*)

get the width of the memory

Parameters:

memory ZEBU_memory handler

Returns:

unsigned int

Return values:

width of the memory

6.14.1.23 unsigned int ZEBU_Memory_writememb (ZEBU_Memory ∗ *memory*, const char ∗ *filename*, unsigned int *startAddress*, unsigned int *endAddress*)

Store the content of a memory to a human readable file in binary data format.

Parameters:

memory ZEBU_memory handler

filename Path to the file to create.

startAddress First address to be dumped

endAddress Last address to be dumped

Return values:

0 is success

$>$***1*** otherwise.

6.14.1.24 unsigned int ZEBU_Memory_writememh (ZEBU_Memory $*$ *memory*, const char $*$ *filename*, unsigned int *startAddress*, unsigned int *endAddress*)

Store the content of a memory to a human readable file in hexadecimal data format.

Parameters:

memory ZEBU_memory handler

filename Path to the file to create.

startAddress First address to be dumped

endAddress Last address to be dumped

Return values:

0 is success

$>$***1*** otherwise.

6.14.1.25 unsigned int ZEBU_Memory_writeWord (ZEBU_Memory $*$ *memory*, unsigned int *address*, const unsigned int $*$ *word_buffer*)

write a buffer into a memory word

Parameters:

memory ZEBU_memory handler

address address to write

word_buffer pointer to the buffer

Returns:

unsigned int

Return values:

0 OK

>0 KO

See also:

ZEBU_Memory_newWordBuffer
ZEBU_Memory_readWord

6.15 ZEBU_PartMemory.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_Memory ∗ ZEBU_PartMemory_Create (ZEBU_Board ∗board, unsigned int partWidth, unsigned int partDepth, const char ∗memoryName, unsigned int memoryBit, unsigned int memoryAddr)

  Construct a sub-memory of an existing memory instance. ! ! The memory part is defined with: ! - the name of the memory instance, ! - the address (w) and bit (b) of the part into the existing memory, ! - the width (Wp) and depth (Dp) of the constructed memory. ! This part will be mapped at address 0 and bit 0 of the constructed memory. ! !

  ```
  !        Wm <----------- 0        Wp <-- 0
  !      0  +----------b---+         #=====#  0
  !      |  |          |   |         "     "  |
  !      |  |    +-----+---w         "     "  |
  !      |  |    |   o~~~~~~~~~~~~~~~~>o   "  v
  !      |  |    |     |   |         #=====#  Dp
  !      |  |    |     |   |
  !      v  |    +-----+   |
  !      Dm +--------------+
  !
  !         Existing memory         PartMemory
  !
  ```
  ! !

- ZEBU_PartMemoryBuilder ∗ ZEBU_PartMemoryBuilder_create (unsigned int partWidth, unsigned int partDepth)

  Create a new ZEBU_PartMemoryBuilder instance. ! !

- void ZEBU_PartMemoryBuilder_destroy (ZEBU_PartMemoryBuilder ∗pmb)
- void ZEBU_PartMemoryBuilder_AddPartByName (ZEBU_PartMemoryBuilder ∗pmb, const char ∗memoryName, unsigned int partBit, unsigned int partAddress)

  Specify a sub-memory and its location into the created memory. ! ! The memory name has to follow the following syntax !

  ```
  !    <path of the memory>[addr1:addr2][bit1:bit2]
  !
  ```
  ! for example, "top.ins0.ins2.mem[7:3][31:0]". ! ! Address range ([addr1:addr2]) and bit range ([bit1:bit2]) are mandatory in the memory name. ! But the ordering of the ranges are not taken into account. The memory will always be read ! from the lowest address to the highest address and same thing for the bits. ! That means that all the following sub-parts are equivalent: ! "top.ins0.ins2.mem[7:3][31:0]" ! "top.ins0.ins2.mem[3:7][31:0]" ! "top.ins0.ins2.mem[3:7][0:31]" ! !

- void ZEBU_PartMemoryBuilder_AddPart (ZEBU_PartMemoryBuilder ∗pmb, const char ∗memoryBaseName, unsigned int memoryBit, unsigned int memoryAddr, unsigned int memoryWidth, unsigned int memoryDepth, unsigned int partBit, unsigned int partAddress)

  Specify a sub-memory and its location into the created memory. ! !

- ZEBU_Memory ∗ ZEBU_PartMemoryBuilder_CreatePartMemory (ZEBU_PartMemoryBuilder ∗pmb, ZEBU_Board ∗board)

  Create a new memory instance. ! !

6.15.1 Function Documentation

6.15.1.1 ZEBU_Memory ∗ ZEBU_PartMemory_Create (ZEBU_Board ∗ *board*, unsigned int *partWidth*, unsigned int *partDepth*, const char ∗ *memoryName*, unsigned int *memoryBit*, unsigned int *memoryAddr*)

Construct a sub-memory of an existing memory instance. ! ! The memory part is defined with: ! - the name of the memory instance, ! - the address (w) and bit (b) of the part into the existing memory, ! - the width (Wp) and depth (Dp) of the constructed memory. ! This part will be mapped at address 0 and bit 0 of the constructed memory. ! !

```
!        Wm <----------- 0        Wp <-- 0
!      0  +----------b---+         #=====#  0
!      |  |          |   |         "     "  |
!      |  |    +-----+---w         "     "  |
!      |  |    |   o~~~~~~~~~~~~~~~~>o   "  v
!      |  |    |     |   |         #=====#  Dp
!      |  |    |     |   |
!      v  |    +-----+   |
!      Dm +--------------+
!
!         Existing memory          PartMemory
!
```

! !

! !

Parameters:

board A pointer to the currently opened board !

partWidth Part memory width in number of bits !

partDepth Part memory depth in number of memory words !

memoryName Name of the existing memory !

memoryBit Bit of the part into the existing memory !

memoryAddr Address of the part into the existing memory

6.15.1.2 void ZEBU_PartMemoryBuilder_AddPart (ZEBU_PartMemoryBuilder ∗ *pmb*, const char ∗ *memoryBaseName*, unsigned int *memoryBit*, unsigned int *memoryAddr*, unsigned int *memoryWidth*, unsigned int *memoryDepth*, unsigned int *partBit*, unsigned int *partAddress*)

Specify a sub-memory and its location into the created memory. ! !

! !

Parameters:

pmb Handler to the `ZEBU_PartMemoryBuilder` instance. !

memoryName The full path of the existing memory instance. !

memoryBit Bit of the part into the existing memory !

memoryAddr Address of the part into the existing memory !

memoryWidth Width of the part into the existing memory !

memoryDepth Depth of the part into the existing memory !

partBit Bit of the part into the created memory. !

partAddress Address of this part into the created memory. ! !

See also:

ZEBU_PartMemoryBuilder_create

6.15.1.3 void ZEBU_PartMemoryBuilder_AddPartByName (ZEBU_PartMemoryBuilder ∗ *pmb*, const char ∗ *memoryName*, unsigned int *partBit*, unsigned int *partAddress*)

Specify a sub-memory and its location into the created memory. ! ! The memory name has to follow the following syntax !

```
!    <path of the memory>[addr1:addr2][bit1:bit2]
!
```

! for example, "top.ins0.ins2.mem[7:3][31:0]". ! ! Address range ([addr1:addr2]) and bit range ([bit1:bit2]) are mandatory in the memory name. ! But the ordering of the ranges are not taken into account. The memory will always be read ! from the lowest address to the highest address and same thing for the bits. ! That means that all the following sub-parts are equivalent: ! "top.ins0.ins2.mem[7:3][31:0]" ! "top.ins0.ins2.mem[3:7][31:0]" ! "top.ins0.ins2.mem[3:7][0:31]" ! !

! !

Parameters:

pmb Handler to the `ZEBU_PartMemoryBuilder` instance. !

memoryName The full description of the sub-memory. !

partBit Bit of the part into the created memory. !

partAddress Address of the part into the created memory. ! !

See also:

ZEBU_PartMemoryBuilder_create

6.15.1.4 ZEBU_PartMemoryBuilder ∗ ZEBU_PartMemoryBuilder_create (unsigned int *partWidth*, unsigned int *partDepth*)

Create a new ZEBU_PartMemoryBuilder instance. ! !

! !

Parameters:

pmb Handler to the `ZEBU_PartMemoryBuilder` instance. ! !

See also:

ZEBU_PartMemoryBuilder_create

6.15.1.5 ZEBU_Memory ∗ ZEBU_PartMemoryBuilder_CreatePartMemory (ZEBU_PartMemoryBuilder ∗ *pmb*, ZEBU_Board ∗ *board*)

Create a new memory instance. ! !

! !

Parameters:

pmb Handler to the `ZEBU_PartMemoryBuilder` instance. !

board Handler to the `ZEBU_Board` ! !

See also:

ZEBU_PartMemoryBuilder_create

6.15.1.6 void ZEBU_PartMemoryBuilder_destroy (ZEBU_PartMemoryBuilder ∗ *pmb*)

6.16 ZEBU_PartSignal.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_Signal ∗ ZEBU_PartSignal_Create (ZEBU_Board ∗board, const char ∗∗names)

  Contruct a signal from a list of parts of signals or memories. ! ! The constructed signal will have for length the sum of the lengths of all individual signals. ! The list is an array of const char∗ strings ended with a NULL pointer. ! The constructed signal LSB will be the last element of the list. ! The constructed signal MSB will be the first element of the list. ! ! Supported strings in the nameList: ! signal path "top.level0.sig" -or- "top.level0.vec[3]" ! vector path "top.level0.vec" -or- "top.level0.vec[5:3]" ! inverted signal path "∼top.level0.sig" -or- "∼top.level0.vec[3]" ! inverted vector path "∼top.level.vec" -or- "∼top.level0.vec[5:3]" ! constant "=0" -or- "=1" ! !

```
!   const char* nameList[] = {
!       "top.level0.vec[6]",
!       "top.level0.vec[5]",
!       "top.level0.vec[4]",
!       NULL
!   };
!
!   ZEBU_Signal* partSignal = ZEBU_PartSignal_Create(zebuBoard, nameList);
!   // This will result in a 3 bit signal
!   // partSignal[0] --> top.level0.vec[4]
!   // partSignal[1] --> top.level0.vec[5]
!   // partSignal[2] --> top.level0.vec[6]
!
```

  .

6.16.1 Function Documentation

6.16.1.1 ZEBU_Signal ∗ ZEBU_PartSignal_Create (ZEBU_Board ∗ *board*, const char ∗∗ *names*)

Contruct a signal from a list of parts of signals or memories. ! ! The constructed signal will have for length the sum of the lengths of all individual signals. ! The list is an array of const char∗ strings ended with a NULL pointer. ! The constructed signal LSB will be the last element of the list. ! The constructed signal MSB will be the first element of the list. ! ! Supported strings in the nameList: ! signal path "top.level0.sig" -or- "top.level0.vec[3]" ! vector path "top.level0.vec" -or- "top.level0.vec[5:3]" ! inverted signal path "∼top.level0.sig" -or- "∼top.level0.vec[3]" ! inverted vector path "∼top.level.vec" -or- "∼top.level0.vec[5:3]" ! constant "=0" -or- "=1" ! !

```
!   const char* nameList[] = {
```

```
!       "top.level0.vec[6]",
!       "top.level0.vec[5]",
!       "top.level0.vec[4]",
!       NULL
!   };
!
!   ZEBU_Signal* partSignal = ZEBU_PartSignal_Create(zebuBoard, nameList);
!   // This will result in a 3 bit signal
!   // partSignal[0] --> top.level0.vec[4]
!   // partSignal[1] --> top.level0.vec[5]
!   // partSignal[2] --> top.level0.vec[6]
!
```

.

! !

6.17 ZEBU_Port.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_Port ∗ ZEBU_getTxPort (const char ∗name, unsigned int size)

  get a new Tx Port

- ZEBU_Port ∗ ZEBU_getRxPort (char ∗name, unsigned int size)

  get a new Rx Port

- unsigned int ZEBU_Port_destroy (ZEBU_Port ∗port)

  destroy the port

- unsigned int ZEBU_Port_connect (ZEBU_Port ∗port, ZEBU_Board ∗board, const char ∗driverName)

  connect a port to a ZEBU Board

- unsigned int ZEBU_Port_disconnect (ZEBU_Port ∗port)

  disconnect a port

- unsigned int ZEBU_Port_isPossibleToReceive (ZEBU_Port ∗port)

  true if port can receive data

- unsigned int ZEBU_Port_isPossibleToSend (ZEBU_Port ∗port)

  true if port can send data

- void ZEBU_Port_waitToReceive (ZEBU_Port ∗port)

  wait to receive data

- void ZEBU_Port_waitToSend (ZEBU_Port ∗port)

  wait to send data

- unsigned int ∗ ZEBU_Port_receiveMessage (ZEBU_Port ∗port)

  receive a message from the hardware side

- unsigned int ZEBU_Port_sendMessage (ZEBU_Port ∗port)

  send a message to the hardware side

- unsigned int ZEBU_Port_size (ZEBU_Port ∗port)

  get the size of a port

- unsigned int ∗ ZEBU_Port_message (ZEBU_Port ∗port)

  get a pointer to the message buffer of the port

- unsigned int ZEBU_Port_read (ZEBU_Port ∗port, unsigned int index)

  get a word of the last received message

- void ZEBU_Port_write (ZEBU_Port ∗port, unsigned int word, unsigned int value)

  set a word of the next message to be send

- unsigned long long ZEBU_Port_date (ZEBU_Port ∗port)

  returns date of last message.

- void ZEBU_Port_registerCB (ZEBU_Port ∗port, void(∗cb)(void ∗), void ∗user)

  register a user callback on the port. The callback is used in ZEBU_Board_serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

- void ZEBU_Port_setGroup (ZEBU_Port ∗port, const unsigned int groupNumber)

  set the group of the port. The group of the port must be declared to use ZEBU_Port_WaitGroup and ZEBU_Board_serviceLoop3 functions.

- void ZEBU_Port_setGroupL (ZEBU_Port ∗port, const long long unsigned int groupNumber)

  set the group of the port. The group of the port must be declared to use ZEBU_Port_WaitGroup and ZEBU_Board_serviceLoop3_2 functions.

- void ZEBU_Port_WaitGroup (ZEBU_Board ∗board, const unsigned int groupNumber)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- void ZEBU_Port_WaitGroupL (ZEBU_Board ∗board, const long long unsigned int groupNumber)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- void ZEBU_Port_WaitGroup2 (ZEBU_Port ∗port, const unsigned int groupNumber)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- void ZEBU_Port_WaitGroup2L (ZEBU_Port ∗port, const long long unsigned int groupNumber)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- int ZEBU_Port_WaitGroup_timedwait (ZEBU_Board ∗board, const unsigned int groupNumber, const unsigned int timeout)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- int ZEBU_Port_WaitGroup_timedwaitL (ZEBU_Board ∗board, const long long unsigned int groupNumber, const unsigned int timeout)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- int ZEBU_Port_WaitGroup_timedwait2 (ZEBU_Port ∗port, const unsigned int groupNumber, const unsigned int timeout)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

- int ZEBU_Port_WaitGroup_timedwait2L (ZEBU_Port ∗port, const long long unsigned int groupNumber, const unsigned int timeout)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

6.17.1 Function Documentation

6.17.1.1 ZEBU_Port ∗ ZEBU_getRxPort (char ∗ *name*, unsigned int *size*)

get a new Rx Port

Parameters:

name name of the port

size size of the port in 32-bits words

Returns:

ZEBU_Port∗

Return values:

handler on Port. NULL if failed.

6.17.1.2 ZEBU_Port $*$ ZEBU_getTxPort (const char $*$ *name*, unsigned int *size*)

get a new Tx Port

Parameters:

name name of the port

size size of the port in 32-bits words

Returns:

ZEBU_Port$*$

Return values:

handler on Port. NULL if failed.

6.17.1.3 unsigned int ZEBU_Port_connect (ZEBU_Port $*$ *port*, ZEBU_Board $*$ *board*, const char $*$ *driverName*)

connect a port to a ZEBU Board

Parameters:

port handler on a `ZEBU_Port`

board handler on a `ZEBU_Board`

driverName name of the driver

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

See also:

ZEBU_Port_disconnect

6.17.1.4 unsigned long long ZEBU_Port_date (ZEBU_Port $*$ *port*)

returns date of last message.

Parameters:

port handler on a `ZEBU_Port`

Bug

always return 0xffffffff

6.17.1.5 unsigned int ZEBU_Port_destroy (ZEBU_Port * *port*)

destroy the port

Parameters:

port ZEBU_Port handler

Returns:

unsigned int

Return values:

0 OK

>*0* KO

6.17.1.6 unsigned int ZEBU_Port_disconnect (ZEBU_Port * *port*)

disconnect a port

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int

Return values:

0 OK

>*0* KO

See also:

ZEBU_Port_connect

6.17.1.7 unsigned int ZEBU_Port_isPossibleToReceive (ZEBU_Port * *port*)

true if port can receive data

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int

Return values:

0 false, the port cannot receive data

$>$*0* true, the port can receive data

See also:

ZEBU_Port_isPossibleToSend

6.17.1.8 unsigned int ZEBU_Port_isPossibleToSend (ZEBU_Port ∗ *port*)

true if port can send data

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int

Return values:

0 false, the port cannot send data

$>$*0* true, the port can send data

See also:

ZEBU_Port_isPossibleToReceive

6.17.1.9 unsigned int ∗ ZEBU_Port_message (ZEBU_Port ∗ *port*)

get a pointer to the message buffer of the port

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int ∗

Return values:

pointer to the buffer message

See also:

ZEBU_Port_write
ZEBU_Port_read
ZEBU_Port_message

6.17.1.10 unsigned int ZEBU_Port_read (ZEBU_Port ∗ *port*, unsigned int *index*)

get a word of the last received message

Parameters:

port handler on a `ZEBU_Port`

index index of the word

Returns:

unsigned int

Return values:

value of the specified word of the last received message

See also:

ZEBU_Port_write
ZEBU_Port_message

6.17.1.11 unsigned int ∗ ZEBU_Port_receiveMessage (ZEBU_Port ∗ *port*)

receive a message from the hardware side

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int ∗

Return values:

pointer to the received message

See also:

ZEBU_Port_read
ZEBU_Port_message
ZEBU_Port_sendMessage

6.17.1.12 void ZEBU_Port_registerCB (ZEBU_Port ∗ *port*, void(∗)(void ∗) *cb*, void ∗ *user*)

register a user callback on the port. The callback is used in ZEBU_Board_serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

Parameters:

port handler on a `ZEBU_Port`

cb callback function pointer

user pointer to the arguments of the callback

See also:

ZEBU_Board_serviceLoop

6.17.1.13 unsigned int ZEBU_Port_sendMessage (ZEBU_Port ∗ *port*)

send a message to the hardware side

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int

Return values:

0 OK

$>$*0* KO

See also:

ZEBU_Port_write
ZEBU_Port_message
ZEBU_Port_receiveMessage

6.17.1.14 void ZEBU_Port_setGroup (ZEBU_Port ∗ *port*, const unsigned int *groupNumber*)

set the group of the port. The group of the port must be declared to use ZEBU_Port_-WaitGroup and ZEBU_Board_serviceLoop3 functions.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

See also:

ZEBU_Port_WaitGroup
ZEBU_Board_serviceLoop3

6.17.1.15 void ZEBU_Port_setGroupL (ZEBU_Port ∗ *port*, const long long unsigned int *groupNumber*)

set the group of the port. The group of the port must be declared to use ZEBU_Port_-WaitGroup and ZEBU_Board_serviceLoop3_2 functions.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

See also:

ZEBU_Port_WaitGroup
ZEBU_Board_serviceLoop3L

6.17.1.16 unsigned int ZEBU_Port_size (ZEBU_Port ∗ *port*)

get the size of a port

Parameters:

port handler on a `ZEBU_Port`

Returns:

unsigned int

Return values:

size of the port in 32-bit words

See also:

ZEBU_Port_write
ZEBU_Port_read
ZEBU_Port_message

6.17.1.17 void ZEBU_Port_WaitGroup (ZEBU_Board ∗ *board*, const unsigned int *groupNumber*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

board handler to a `ZEBU_Board`

groupNumber identifier of the group

See also:

ZEBU_Port_setGroup
ZEBU_Board_serviceLoop3

6.17.1.18 void ZEBU_Port_WaitGroup2 (ZEBU_Port ∗ *port*, const unsigned int *groupNumber*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

See also:

ZEBU_Port_setGroup
ZEBU_Board_serviceLoop3

6.17.1.19 void ZEBU_Port_WaitGroup2L (ZEBU_Port ∗ *port*, const long long unsigned int *groupNumber*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

See also:

ZEBU_Port_setGroupL
ZEBU_Board_serviceLoop3L

6.17.1.20 int ZEBU_Port_WaitGroup_timedwait (ZEBU_Board ∗ *board*, const unsigned int *groupNumber*, const unsigned int *timeout*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

board handler to a `ZEBU_Board`

groupNumber identifier of the group

timeout timeout in micro-seconds.

Return values:

0 if no error

-1 if timeout expired

See also:

ZEBU_Port_setGroup
ZEBU_Board_serviceLoop3

6.17.1.21 int ZEBU_Port_WaitGroup_timedwait2 (ZEBU_Port ∗ *port*, const unsigned int *groupNumber*, const unsigned int *timeout*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

timeout timeout in micro-seconds.

Return values:

0 if no error

-1 if timeout expired

See also:

ZEBU_Port_setGroup
ZEBU_Board_serviceLoop3

6.17.1.22 int ZEBU_Port_WaitGroup_timedwait2L (ZEBU_Port ∗ *port*, const long long unsigned int *groupNumber*, const unsigned int *timeout*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

port handler on a `ZEBU_Port`

groupNumber identifier of the group

timeout timeout in micro-seconds.

Return values:

0 if no error

-1 if timeout expired

See also:

ZEBU_Port_setGroup
ZEBU_Board_serviceLoop3

6.17.1.23 int ZEBU_Port_WaitGroup_timedwaitL (ZEBU_Board ∗ *board*, const long long unsigned int *groupNumber*, const unsigned int *timeout*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

board handler to a `ZEBU_Board`

groupNumber identifier of the group

timeout timeout in micro-seconds.

Return values:

0 if no error

-1 if timeout expired

See also:

ZEBU_Port_setGroupL
ZEBU_Board_serviceLoop3L

6.17.1.24 void ZEBU_Port_WaitGroupL (ZEBU_Board ∗ *board*, const long long unsigned int *groupNumber*)

wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

Parameters:

board handler to a `ZEBU_Board`

groupNumber identifier of the group

See also:

ZEBU_Port_setGroupL
ZEBU_Board_serviceLoop3L

6.17.1.25 unsigned int ZEBU_Port_waitToReceive (ZEBU_Port ∗ *port*)

wait to receive data

Parameters:

port handler on a `ZEBU_Port`

See also:

ZEBU_Port_isPossibleToSend

6.17.1.26 unsigned int ZEBU_Port_waitToSend (ZEBU_Port ∗ *port*)

wait to send data

Parameters:

port handler on a `ZEBU_Port`

See also:

ZEBU_Port_isPossibleToSend

6.17.1.27 void ZEBU_Port_write (ZEBU_Port ∗ *port*, unsigned int *word*, unsigned int *value*)

set a word of the next message to be send

Parameters:

port handler on a `ZEBU_Port`

word index of the word

value value to write

See also:

ZEBU_Port_read
ZEBU_Port_message

6.18 ZEBU_Signal.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned int ZEBU_Signal_destroy (ZEBU_Signal ∗signal)

  destroy the signal

- unsigned int ZEBU_Signal_read (ZEBU_Signal ∗signal)

  read signal value

- char ∗ ZEBU_Signal_fetchValue (ZEBU_Signal ∗signal, const char ∗format, struct ZEBU_Value ∗value)

  get the logic or strength value of a Signal (or Vector)

- char ∗ ZEBU_Signal_resolveValue (ZEBU_Signal ∗signal, const char ∗format, struct ZEBU_Value ∗value)

  get the resolved logic or strength value of a Signal (or Vector)

- int ZEBU_Signal_setValue (ZEBU_Signal ∗signal, struct ZEBU_Value ∗value)

  set a value on a signal or a Vector

- unsigned int ZEBU_Signal_setString (ZEBU_Signal ∗signal, const char ∗string)

  write signal value

- unsigned int ZEBU_Signal_write (ZEBU_Signal ∗signal, unsigned int value)

  write a value in a signal.

- unsigned int ZEBU_Signal_set (ZEBU_Signal ∗signal, unsigned int index, unsigned int value)
- unsigned int ZEBU_Signal_get (ZEBU_Signal ∗signal, unsigned int index)
- ZEBU_Signal ∗ ZEBU_Signal_getBit (ZEBU_Signal ∗signal, unsigned int bit)

  get a handle on a bit of the vector

- unsigned int ZEBU_Signal_size (ZEBU_Signal ∗signal)

  get the size of a signal

- char ∗ ZEBU_Signal_name (ZEBU_Signal ∗signal) *get the name of a signal*

- char ∗ ZEBU_Signal_fullname (ZEBU_Signal ∗signal, char separator)

  get the name of a signal

- unsigned int ZEBU_Signal_isInternal (ZEBU_Signal ∗signal)

  test if the signal is internal

- unsigned int ZEBU_Signal_isWritable (ZEBU_Signal ∗signal)

  test if the signal is writable

- unsigned int ZEBU_Signal_isSelected (ZEBU_Signal ∗signal)

  test if the signal is selected

- unsigned int ZEBU_Signal_isForceable (const ZEBU_Board ∗board, const ZEBU_Signal ∗signal)

  test if a signal can be assigned for an indeterminate time

- unsigned int ZEBU_Signal_isForceable2 (const ZEBU_Board ∗board, const char ∗signalFullname)

  test if a signal can be assigned for an indeterminate time

- unsigned int ZEBU_Signal_force (ZEBU_Board ∗board, ZEBU_Signal ∗signal, const unsigned int ∗value)

  assign a signal for an indeterminate time ZEBU_Board_writeRegisters has to be called after to apply modification

- unsigned int ZEBU_Signal_force2 (ZEBU_Board ∗board, const char ∗signal-Fullname, const unsigned int ∗value)

  assign a signal for an indeterminate time ZEBU_Board_writeRegisters has to be called after to apply modification

- unsigned int ZEBU_Signal_release (ZEBU_Board ∗board, ZEBU_Signal ∗signal)

  desassign a signal ZEBU_Board_writeRegisters has to be called after to apply modification

- unsigned int ZEBU_Signal_release2 (ZEBU_Board ∗board, const char ∗signal-Fullname)

  desassign a signal ZEBU_Board_writeRegisters has to be called after to apply modification

6.18.1 Function Documentation

6.18.1.1 unsigned int ZEBU_Signal_destroy (ZEBU_Signal ∗ *signal*)

destroy the signal

Parameters:

signal ZEBU_Signal handler

Returns:

unsigned int

Return values:

0 OK

>*0* KO

6.18.1.2 char ∗ ZEBU_Signal_fetchValue (ZEBU_Signal ∗ *signal*, const char ∗ *format*, struct ZEBU_Value ∗ *value*)

get the logic or strength value of a Signal (or Vector)

Parameters:

signal signal handle

format C string with the following specifiers for formatting the return value

- `"%b"` binary
- `"%d"` decimal
- `"%h"` hexadecimal
- `"%o"` octal
- `"%%"` aval/bval pair

value pointer to a structure with the retrieved logic value and strength; used when format string is "%%"

Returns:

char∗ pointer to a char string

read signal value as Verilog PLI `acc_fetch_value` function.

See also:

ZEBU_Signal_setValue

6.18.1.3 void ZEBU_Signal_force (ZEBU_Board ∗ *board*, ZEBU_Signal ∗ *signal*, const unsigned int ∗ *value*)

assign a signal for an indeterminate time ZEBU_Board_writeRegisters has to be called after to apply modification

Parameters:

board handler to a `ZEBU_Board`

signal handler to the c\ ZEBU_Signal of the signal to force

value value to assign

Returns:

unsigned int

Return values:

0 OK

*>**0*** KO

6.18.1.4 void ZEBU_Signal_force2 (ZEBU_Board ∗ *board*, const char ∗ *signalFullname*, const unsigned int ∗ *value*)

assign a signal for an indeterminate time ZEBU_Board_writeRegisters has to be called after to apply modification

Parameters:

board handler to a `ZEBU_Board`

signalFullname hierarchical name of the signal to force

value value to assign

Returns:

unsigned int

Return values:

0 OK

*>**0*** KO

6.18.1.5 char ∗ ZEBU_Signal_fullname (ZEBU_Signal ∗ *signal*, char *separator*)

get the name of a signal

Parameters:

signal handle

separator hierarchical sperarator character to use

Returns:

char ∗

Return values:

name of the signal

6.18.1.6 unsigned int ZEBU_Signal_get (ZEBU_Signal ∗ *signal*, unsigned int *index*)

6.18.1.7 ZEBU_Signal ∗ ZEBU_Signal_getBit (ZEBU_Signal ∗ *signal*, unsigned int *bit*)

get a handle on a bit of the vector

Parameters:

signal handler

bit number of the bit to get

Returns:

ZEBU_Signal ∗

Return values:

handle on the signal #bit of the vector

6.18.1.8 bool ZEBU_Signal_isForceable (const ZEBU_Board ∗ *board*, const ZEBU_Signal ∗ *signal*)

test if a signal can be assigned for an indeterminate time

Parameters:

board signal's holder

signal handler to the c\ ZEBU_Signal of the signal to test

Returns:

unsigned int

Return values:

1 if signal is forcable

0 is signal is not forcable

6.18.1.9 bool ZEBU_Signal_isForceable2 (const ZEBU_Board ∗ *board*, const char ∗ *signalFullname*)

test if a signal can be assigned for an indeterminate time

Parameters:

board signal's holder

signalFullname hierarchical name of the signal to test

Returns:

unsigned int

Return values:

1 if signal is forcable

0 is signal is not forcable

6.18.1.10 unsigned int ZEBU_Signal_isInternal (ZEBU_Signal ∗ *signal*)

test if the signal is internal

Parameters:

signal ZEBU_Signal handler

Returns:

unsigned int

Return values:

1 if signal is internal

0 is signal is not internal

6.18.1.11 unsigned int ZEBU_Signal_isSelected (ZEBU_Signal ∗ *signal*)

test if the signal is selected

Parameters:

signal ZEBU_Signal handler

Returns:

unsigned int

Return values:

1 if signal is selected

0 is signal is not selected

6.18.1.12 unsigned int ZEBU_Signal_isWritable (ZEBU_Signal ∗ *signal*)

test if the signal is writable

Parameters:

signal ZEBU_Signal handler

Returns:

unsigned int

Return values:

1 if signal is writable

0 is signal is not writable

6.18.1.13 char ∗ ZEBU_Signal_name (ZEBU_Signal ∗ *signal*)

get the name of a signal

Parameters:

signal handle

Returns:

char ∗

Return values:

name of the signal

6.18.1.14 unsigned int ZEBU_Signal_read (ZEBU_Signal ∗ *signal*)

read signal value

Parameters:

signal signal handler

Returns:

unsigned int with signal value

Return values:

b0 0

b1 1

bZ high-Z

bz high-Z

read signal value. If signal is a handler on a ZEBU_Vector, return the value of the first 32 bits (lsb).

6.18.1.15 void ZEBU_Signal_release (ZEBU_Board ∗ *board*, ZEBU_Signal ∗ *signal*)

desassign a signal ZEBU_Board_writeRegisters has to be called after to apply modification

Parameters:

board handler to a `ZEBU_Board`

signal handler to the c\ ZEBU_Signal of the signal to release

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.18.1.16 void ZEBU_Signal_release2 (ZEBU_Board ∗ *board*, const char ∗ *signalFullname*)

desassign a signal ZEBU_Board_writeRegisters has to be called after to apply modification

Parameters:

board handler to a `ZEBU_Board`

signalFullname hierarchical name of the signal to release

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.18.1.17 char ∗ ZEBU_Signal_resolveValue (ZEBU_Signal ∗ *signal*, const char ∗ *format*, struct ZEBU_Value ∗ *value*)

get the resolved logic or strength value of a Signal (or Vector)

Parameters:

signal signal handle

format C string with the following specifiers for formatting the return value

- `"%b"` binary
- `"%d"` decimal
- `"%h"` hexadecimal
- `"%o"` octal
- `"%%"` aval/bval pair

value pointer to a structure with the retrieved logic value and strength; used when format string is "%%"

Returns:

char∗ pointer to a char string

read signal value as Verilog PLI `acc_fetch_value` function.

See also:

ZEBU_Signal_fetchValue

6.18.1.18 unsigned int ZEBU_Signal_set (ZEBU_Signal ∗ *signal*, unsigned int *index*, unsigned int *value*)

6.18.1.19 unsigned int ZEBU_Signal_setString (ZEBU_Signal ∗ *signal*, const char ∗ *string*)

write signal value

Parameters:

signal handler

string value as a NULL terminated C string

- str begins with `0b` for binary format
- str begins with `0x` for hexadecimal format
- str begins with `0` for oct format

Returns:

unsigned int

Return values:

0 OK

>***0*** KO

6.18.1.20 int ZEBU_Signal_setValue (ZEBU_Signal ∗ *signal*, struct ZEBU_Value ∗ *value*)

set a value on a signal or a Vector

Parameters:

signal signal handle

value value

Returns:

int

Return values:

0 no errors

$>$*0* error

write signal value as Verilog PLI `acc_set_value` function.

See also:

ZEBU_Signal_fetchValue

6.18.1.21 unsigned int ZEBU_Signal_size (ZEBU_Signal ∗ *signal*)

get the size of a signal

Parameters:

signal handle

Returns:

unsigned int

Return values:

size of the signal

6.18.1.22 unsigned int ZEBU_Signal_write (ZEBU_Signal ∗ *signal*, unsigned int *value*)

write a value in a signal.

Parameters:

signal handler

value value as an unsigned integer

Write a value in a signal. If signal is a single bit, value can be b0, b1, bZ or bz.

If signal is a vector, value is assigned to first 32 bits (lsb)

6.19 ZEBU_Sniffer.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int ZEBU_Sniffer_initialize (ZEBU_Board ∗board, const char ∗foldername, const char ∗clockName)

  initialize the sniffer

- long long unsigned int ZEBU_Sniffer_start (ZEBU_Board ∗board)

  start the sniffer

- long long unsigned int ZEBU_Sniffer_createFrame (ZEBU_Board ∗board)

  force the sniffer to create a new frame

- long long unsigned int ZEBU_Sniffer_stop (ZEBU_Board ∗board)

  stop the sniffer

- int ZEBU_Sniffer_deleteFrame (ZEBU_Board ∗board, unsigned int frame-Reference)

  delete a previously created frame

- int ZEBU_Sniffer_deleteSavedState (ZEBU_Board ∗board, unsigned int frame-Reference)

  delete the saved state of a previously created frame preventing restart from that frame

- long long unsigned int ZEBU_Sniffer_disable (ZEBU_Board ∗board)

  disable the sniffer

- long long unsigned int ZEBU_Sniffer_enable (ZEBU_Board ∗board)

  enable the sniffer

6.19.1 Function Documentation

6.19.1.1 long long unsigned int ZEBU_Sniffer_createFrame (ZEBU_Board ∗ *board*)

force the sniffer to create a new frame

Parameters:

board C handler on ZEBU_Board

Return values:

the cycle number of the reference clock at which the sniffer has been started

ULONG_MAX on error

6.19.1.2 int ZEBU_Sniffer_deleteFrame (ZEBU_Board ∗ *board*, unsigned int *frameReference*)

delete a previously created frame

Parameters:

board C handler on ZEBU_Board

Returns:

status

Return values:

0 OK

>*0* KO

6.19.1.3 int ZEBU_Sniffer_deleteSavedState (ZEBU_Board ∗ *board*, unsigned int *frameReference*)

delete the saved state of a previously created frame preventing restart from that frame

Parameters:

board C handler on ZEBU_Board

Returns:

status

Return values:

0 OK

>*0* KO

6.19.1.4 long long unsigned int ZEBU_Sniffer_disable (ZEBU_Board ∗ *board*)

disable the sniffer

Parameters:

board C handler on ZEBU_Board

Return values:

the cycle number of the reference clock at which the sniffer has been disabled

ULONG_MAX on error

6.19.1.5 long long unsigned int ZEBU_Sniffer_enable (ZEBU_Board ∗ *board*)

enable the sniffer

Parameters:

board C handler on ZEBU_Board

Return values:

the cycle number of the reference clock at which the sniffer has been enabled

ULONG_MAX on error

6.19.1.6 int ZEBU_Sniffer_initialize (ZEBU_Board ∗ *board*, const char ∗ *foldername*, const char ∗ *clockName*)

initialize the sniffer

Parameters:

board C handler on ZEBU_Board

foldername name of the directory in which must be saved sniffed data

clockName name of a reference clock of the sniffer

Returns:

status

Return values:

0 OK

>0 KO

6.19.1.7 long long unsigned int ZEBU_Sniffer_start (ZEBU_Board ∗ *board*)

start the sniffer

Parameters:

\param board C handler on ZEBU_Board

Return values:

the cycle number of the reference clock at which the sniffer has been started

ULONG_MAX on error

6.19.1.8 long long unsigned int ZEBU_Sniffer_stop (ZEBU_Board ∗ *board*)

stop the sniffer

Parameters:

board C handler on ZEBU_Board

Return values:

the cycle number of the reference clock at which the sniffer has been started

ULONG_MAX on error

6.20 ZEBU_SVA.h File Reference

Include dependency graph for ZEBU_SVA.h:

This graph shows which files directly or indirectly include this file:

Typedefs

- typedef Enable types enum _ZEBU_SVA_EnableType ZEBU_SVA_EnableType

Enumerations

- enum _ZEBU_SVA_EnableType { ZEBU_SVA_DISABLE = 0x0, ZEBU_SVA_ENABLE_TRIGGER = 0x1, ZEBU_SVA_ENABLE_REPORT = 0x2 }

Functions

- int ZEBU_SVA_Start (ZEBU_Board ∗board, const char ∗clockName, const unsigned int enableTypes)

  start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

- int ZEBU_SVA_Start2 (ZEBU_Board ∗board, const char ∗clockName, const unsigned int enableTypes, const int timeInSVASamplingClockCycles)

  start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

- int ZEBU_SVA_Start_callback (ZEBU_Board ∗board, const char ∗clockName, ZEBU_SVA_Report callback, void ∗context, const unsigned int enableTypes)

  start System Verilog Assertion Assertion messages are uncompacted and are reported through a callback during emulation

- int ZEBU_SVA_Start_callback2 (ZEBU_Board ∗board, const char ∗clockName, ZEBU_SVA_Report callback, void ∗context, const unsigned int enableTypes, const int timeInSVASamplingClockCycles)
- int ZEBU_SVA_Start_file (ZEBU_Board ∗board, const char ∗clockName, const char ∗filename, const unsigned int enableTypes)

  start System Verilog Assertion Assertion messages are not uncompacted and reported during emulation but are saved in a .zsva file

- int ZEBU_SVA_Stop (ZEBU_Board ∗board)

  stop all System Verilog Assertions

- int ZEBU_SVA_Set (ZEBU_Board ∗board, const unsigned int types, const char ∗regularExpression, const int invert, const int ignoreCase, const char hierarchicalSeparator)
- int ZEBU_SVA_SelectReport (ZEBU_Board ∗board, const ZEBU_SVA_-Severity severities)
- int ZEBU_SVA_enableClockStoppingOnFailure (ZEBU_Board ∗board, ZEBU_SVA_OnStop callback, void ∗context)
- int ZEBU_SVA_disableClockStoppingOnFailure (ZEBU_Board ∗board, ZEBU_SVA_OnStop callback, void ∗context)
- int ZEBU_SVA_DesignHasSvaCompiled (ZEBU_Board ∗board)

  returns if design has SVA compiled in it

6.20.1 Typedef Documentation

6.20.1.1 typedef Enable types enum _ZEBU_SVA_EnableType ZEBU_SVA_EnableType

6.20.2 Enumeration Type Documentation

6.20.2.1 enum _ZEBU_SVA_EnableType

Enumeration values:

ZEBU_SVA_DISABLE

ZEBU_SVA_ENABLE_TRIGGER

ZEBU_SVA_ENABLE_REPORT

6.20.3 Function Documentation

6.20.3.1 int ZEBU_SVA_DesignHasSvaCompiled (ZEBU_Board ∗ *board*)

returns if design has SVA compiled in it

Parameters:

board C handler on ZEBU_Board

Returns:

status

Return values:

0 OK, design has SVA

>***0*** KO

6.20.3.2 int ZEBU_SVA_disableClockStoppingOnFailure (ZEBU_Board $*$ *board*, ZEBU_SVA_OnStop *callback*, void $*$ *context*)

6.20.3.3 int ZEBU_SVA_enableClockStoppingOnFailure (ZEBU_Board $*$ *board*, ZEBU_SVA_OnStop *callback*, void $*$ *context*)

6.20.3.4 int ZEBU_SVA_SelectReport (ZEBU_Board $*$ *board*, const ZEBU_SVA_Severity *severities*)

6.20.3.5 int ZEBU_SVA_Set (ZEBU_Board $*$ *board*, const unsigned int *types*, const char $*$ *regularExpression*, const int *invert*, const int *ignoreCase*, const char *hierarchicalSeparator*)

6.20.3.6 int ZEBU_SVA_Start (ZEBU_Board $*$ *board*, const char $*$ *clockName*, const unsigned int *enableTypes*)

start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

Parameters:

board C handler on ZEBU_Board

clockName name of the reference controlled clock

enableTypes specify actions to enable: DISABLE = disable the assertions ENABLE_TRIGGER = enable clock stop when assertions failed ENABLE_-REPORT = enable assertion message report

Returns:

status

Return values:

0 OK

$>$***0*** KO

6.20.3.7 int ZEBU_SVA_Start2 (ZEBU_Board $*$ *board*, const char $*$ *clockName*, const unsigned int *enableTypes*, const int *timeInSVASamplingClockCycles*)

start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

Parameters:

board C handler on ZEBU_Board

clockName name of the reference controlled clock

enableTypes specify actions to enable: DISABLE = disable the assertions ENABLE_TRIGGER = enable clock stop when assertions failed ENABLE_-REPORT = enable assertion message report

timeInSVASamplingClockCycles selects the clock from which time is reported in assertion messages. ZeBu system implicitly creates a SVA sampling clock according to the clock group of the reference controlled clock. SVA sampling clock posedges correspond to posedges and negedges of any clock of the selected group as in the following waveform. controlled clock 1 : _—-___-_—-____—- controlled clock 2 : _——______——__ SVA sampling clock : _-___-_-_-___-___-_-_ By default, assertion messages are reported with time in number of SVA sampling clock cycles. If 1 report time in number of SVA clock cycles. Else report time in number of cycles of the reference controlled clock.

Returns:

status

Return values:

0 OK

>***0*** KO

6.20.3.8 int ZEBU_SVA_Start_callback (ZEBU_Board ∗ *board*, const char ∗ *clockName*, ZEBU_SVA_Report *callback*, void ∗ *context*, const unsigned int *enableTypes*)

start System Verilog Assertion Assertion messages are uncompacted and are reported through a callback during emulation

Parameters:

board C handler on ZEBU_Board

clockName name of the reference controlled clock

callback function to call to report assertion message

context pointer to pass by parameter in the callback

enableType specify actions to enable: DISABLE = disable the assertions ENABLE_TRIGGER = enable clock stop when assertions failed ENABLE_-REPORT = enable assertion message report

Returns:

status

Return values:

0 OK

>***0*** KO

6.20.3.9 int ZEBU_SVA_Start_callback2 (ZEBU_Board ∗ *board*, const char ∗ *clockName*, ZEBU_SVA_Report *callback*, void ∗ *context*, const unsigned int *enableTypes*, const int *timeInSVASamplingClockCycles*)

6.20.3.10 int ZEBU_SVA_Start_file (ZEBU_Board ∗ *board*, const char ∗ *clockName*, const char ∗ *filename*, const unsigned int *enableTypes*)

start System Verilog Assertion Assertion messages are not uncompacted and reported during emulation but are saved in a .zsva file

Parameters:

board C handler on ZEBU_Board

clockName name of the reference controlled clock

filename specify the name of a .zsva file in which must be dumped assertion message data. Then this file can be read by means of zsvaReport tool

enableTypes specify actions to enable: DISABLE = disable the assertions ENABLE_TRIGGER = enable clock stop when assertions failed ENABLE_-REPORT = enable assertion message report

Returns:

status

Return values:

0 OK

>0 KO

6.20.3.11 int ZEBU_SVA_Stop (ZEBU_Board ∗ *board*)

stop all System Verilog Assertions

Returns:

status

Return values:

0 OK

>0 KO

6.21 ZEBU_Trigger.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned int ZEBU_Trigger_destroy (ZEBU_Trigger ∗trigger)

  destroy the trigger

- unsigned int ZEBU_Trigger_define (ZEBU_Trigger ∗trigger, const char ∗definition)

  define the expression of a dynamic trigger

- unsigned int ZEBU_Trigger_index (ZEBU_Trigger ∗trigger)

  get the index of a trigger

- unsigned int ZEBU_Trigger_value (ZEBU_Trigger ∗trigger)

  get the value of a trigger

6.21.1 Function Documentation

6.21.1.1 unsigned int ZEBU_Trigger_define (ZEBU_Trigger ∗ *trigger*, const char ∗ *definition*)

define the expression of a dynamic trigger

Parameters:

trigger trigger handle

definition expression of the trigger

Returns:

unsigned int

Return values:

0 if the expression is correctly programmed

```
ZEBU_Trigger_define(trigger0, "input1 == 0 && input2[7:0] == 8’hbe");
```

6.21.1.2 unsigned int ZEBU_Trigger_destroy (ZEBU_Trigger ∗ *trigger*)

destroy the trigger

Parameters:

trigger ZEBU_Trigger handler

Returns:

unsigned int

Return values:

0 OK

$>$***0*** KO

6.21.1.3 unsigned int ZEBU_Trigger_index (ZEBU_Trigger ∗ *trigger*)

get the index of a trigger

Parameters:

trigger trigger handle

Returns:

unsigned int

Return values:

index of the trigger

See also:

ZEBU_Driver_wait

```
uint mask = 0;
mask = ZEBU_Trigger_index(trigger0) | (ZEBU_Trigger_index(trigger5));
ZEBU_Driver_wait(mask);
```

6.21.1.4 unsigned int ZEBU_Trigger_value (ZEBU_Trigger ∗ *trigger*)

get the value of a trigger

Parameters:

trigger trigger handle

Returns:

unsigned int

Return values:

value of the trigger

```
uint val = 0;
val = ZEBU_Trigger_value(trigger0);
```

6.22 ZEBU_ValueChange.h File Reference

Include dependency graph for ZEBU_ValueChange.h:

This graph shows which files directly or indirectly include this file:

Typedefs

- typedef ZEBU_ValueChange_HandleList ZEBU_ValueChange_HandleList

Functions

- unsigned int ZEBU_ValueChange_enableByName (ZEBU_Board ∗board, const char ∗signalName, ZEBU_vcEdge edge, ZEBU_ValueChange_HandleList ∗∗handles)

  Enables a signal of the value change trigger by its name.

- unsigned int ZEBU_ValueChange_disableByName (ZEBU_Board ∗board, const char ∗signalName, ZEBU_vcEdge edge)

  Disables a signal of the value change trigger by its name.

- unsigned int ZEBU_ValueChange_enable (ZEBU_Board ∗board, ZEBU_Signal ∗signal, ZEBU_vcEdge edge, ZEBU_ValueChange_HandleList ∗∗handles)

  Enables a signal of the value change trigger.

- unsigned int ZEBU_ValueChange_disable (ZEBU_Board ∗board, ZEBU_Signal ∗signal, ZEBU_vcEdge edge)

  Disables a signal of the value change trigger.

- int ZEBU_ValueChange_waitDriver (ZEBU_Board ∗board, ZEBU_Driver ∗driver, unsigned int timeout)

  Waits for a value change trigger event -or- timeout while running the clock through a cosimulation-driver.

- int ZEBU_ValueChange_waitDriverEx (ZEBU_Board ∗board, ZEBU_Driver ∗driver, unsigned int triggers, unsigned int ∗fired, unsigned int timeout)

  Waits for a value change trigger event -or- another trigger event -or- timeout while running the clock through a cosimulation-driver.

- int ZEBU_ValueChange_waitLogicAnalyzer (ZEBU_Board ∗board, const char ∗clockName, const char ∗edgeName, unsigned int timeout)

  Waits for a value change trigger event -or- a timeout through a logic analyzer.

- int ZEBU_ValueChange_waitLogicAnalyzerEx (ZEBU_Board ∗board, const char ∗clockName, const char ∗edgeName, unsigned int triggers, unsigned int ∗fired, unsigned int timeout)

  Waits for a value change trigger event -or- another trigger event -or- a timeout through a logic analyzer.

- ZEBU_ValueChange_Iterator ∗ ZEBU_ValueChange_createIterator (ZEBU_-Board ∗board)

  Create an iterator on changed signals.

- void ZEBU_ValueChange_destroyIterator (ZEBU_ValueChange_Iterator ∗iterator)

  Destroy an iterator on changed signals.

- void ZEBU_ValueChange_Iterator_goToFirst (ZEBU_ValueChange_Iterator ∗iterator)

  Move iterator to the first changed signal.

- void ZEBU_ValueChange_Iterator_goToNext (ZEBU_ValueChange_Iterator ∗iterator)

  Move iterator to the next changed signal.

- int ZEBU_ValueChange_Iterator_isAtEnd (ZEBU_ValueChange_Iterator ∗iterator)

  Test if iterator passed last changed signal.

- const char ∗ ZEBU_ValueChange_Iterator_getName (ZEBU_ValueChange_-Iterator ∗iterator)

  Return the hierarchical gate name of the current changed signal.

- void ∗ ZEBU_ValueChange_Iterator_getHandle (ZEBU_ValueChange_Iterator ∗iterator)

  Return the handle of the current changed signal.

- unsigned int ZEBU_ValueChange_HandleList_getSize (ZEBU_ValueChange_-HandleList ∗handles)

  Return the size of the handle list.

- void ∗ ZEBU_ValueChange_HandleList_getHandle (ZEBU_ValueChange_-HandleList ∗handles, unsigned int index)

  Return the handle at position `index` *in the handle list.*

- void ZEBU_ValueChange_HandleList_destroy (ZEBU_ValueChange_Handle-List ∗handles)

Destroy a `ZEBU_ValueChange_Handles` *handle list.*

6.22.1 Typedef Documentation

6.22.1.1 typedef struct ZEBU_ValueChange_HandleList ZEBU_ValueChange_HandleList

6.22.2 Function Documentation

6.22.2.1 ZEBU_ValueChange_Iterator∗ ZEBU_ValueChange_createIterator (ZEBU_Board ∗ *board*)

Create an iterator on changed signals.

Parameters:

board Handler to a `ZEBU_Board`.

Return values:

A handler on a `ZEBU_ValueChange_Iterator` iterator.

NULL if the creation failed.

```
// Create a new iterator on changed signal names
ZEBU_ValueChange_Iterator *iterator = ZEBU_ValueChange_createIterator(board);
if (iterator == 0) {
    printf("Cannot create changed signal iterator\n");
    exit(1);
}

// Iterate over each changed signal names
for (ZEBU_ValueChange_Iterator_goToFirst(iterator);
     !ZEBU_ValueChange_Iterator_isAtEnd(iterator);
     ZEBU_ValueChange_Iterator_goToNext(iterator) )
{
    printf("Changed signal name / handle: %s / %lu\n",
        ZEBU_ValueChange_Iterator_getName(iterator),
        (size_t)ZEBU_ValueChange_Iterator_getHandle(iterator)
    );
}

// Destroy the iterator
ZEBU_ValueChange_destroyIterator(iterator);
```

6.22.2.2 void ZEBU_ValueChange_destroyIterator (ZEBU_ValueChange_Iterator ∗ *iterator*)

Destroy an iterator on changed signals.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

See also:

ZEBU_ValueChange_createIterator

6.22.2.3 unsigned int ZEBU_ValueChange_disable (ZEBU_Board ∗ *board*, ZEBU_Signal ∗ *signal*, ZEBU_vcEdge *edge*)

Disables a signal of the value change trigger.

Parameters:

board Handler to a `ZEBU_Board`.

signal The signal to disable.

vcEdge Edge on which the trigger should fire. Accepted values are:

- vcEdgeRising
- vcEdgeFalling
- vcEdgeBoth
- vcEdgeAll

Return values:

0 if the operation has complete successfuly.

>*0* an error has occcurs and a message should have been printed.

6.22.2.4 unsigned int ZEBU_ValueChange_disableByName (ZEBU_Board ∗ *board*, const char ∗ *signalName*, ZEBU_vcEdge *edge*)

Disables a signal of the value change trigger by its name.

Parameters:

board Handler to a `ZEBU_Board`.

signalName Hierarchical name of the signal to disable. If NULL, all signals are disabled in the value change trigger.

vcEdge Edge on which the trigger should fire. Accepted values are:

- vcEdgeRising
- vcEdgeFalling
- vcEdgeBoth
- vcEdgeAll

Return values:

0 if the operation has complete successfuly.

>*0* an error has occcurs and a message should have been printed.

6.22.2.5 unsigned int ZEBU_ValueChange_enable (ZEBU_Board ∗ *board*, ZEBU_Signal ∗ *signal*, ZEBU_vcEdge *edge*, ZEBU_ValueChange_HandleList ∗∗ *handles*)

Enables a signal of the value change trigger.

Parameters:

board Handler to a `ZEBU_Board`.

signal The signal to enable.

vcEdge Edge on which the trigger should fire. Accepted values are:

- vcEdgeRising
- vcEdgeFalling
- vcEdgeBoth
- vcEdgeAll

handles A pointer to a pointer to an object of type `ZEBU_ValueChange_-HandleList` which is a list of handles that can be manipulated with `ZEBU_ValueChange_HandleList_∗` functions. This list must be destroyed after use. If NULL, the list is not created.

Remarks:

The list will be ordered from LSB to MSB. Each bit of the signal will match 1 handle if the edge is vcEdgeRising, vcEdgeFalling or vcEdgeBoth. If the edge argument is vcEdgeAll, each bit will be associated to 3 handles: the first for vc-EdgeRising, the second for vcEdgeFalling and the third for vcEdgeBoth. At least one of these handle must be non-NULL. In the first case, the list will have N element (N is the size of the signal). In the second case, the list will have 3∗N elements.

Return values:

0 if the operation has complete successfuly.

$>$***0*** an error has occcurs and a message should have been printed.

6.22.2.6 unsigned int ZEBU_ValueChange_enableByName (ZEBU_Board ∗ *board*, const char ∗ *signalName*, ZEBU_vcEdge *edge*, ZEBU_ValueChange_HandleList ∗∗ *handles*)

Enables a signal of the value change trigger by its name.

Parameters:

board Handler to a `ZEBU_Board`.

signalName Hierarchical name of the signal to enable. If NULL, all signals are enabled in the value change trigger.

vcEdge Edge on which the trigger should fire. Accepted values are:

- vcEdgeRising
- vcEdgeFalling
- vcEdgeBoth
- vcEdgeAll

handles A pointer to a pointer to an object of type `ZEBU_ValueChange_-HandleList` which is a list of handles that can be manipulated with `ZEBU_ValueChange_HandleList_*` functions. This list must be destroyed after use. If NULL, the list is not created.

Remarks:

The handle list will be ordered from LSB to MSB. Each bit of the signal will match 1 handle if the edge is `vcEdgeRising`, `vcEdgeFalling` or `vcEdgeBoth`. If the edge argument is `vcEdgeAll`, each bit will be associated to 3 handles: the first for `vcEdgeRising`, the second for `vcEdgeFalling` and the third for `vcEdgeBoth`. At least one of these handle must be non-NULL. In the first case, the list will have N element (N is the size of the signal). In the second case, the list will have 3*N elements.

Return values:

0 if the operation has complete successfuly.

>***0*** an error has occcurs and a message should have been printed.

6.22.2.7 void ZEBU_ValueChange_HandleList_destroy (ZEBU_ValueChange_HandleList * *handles*)

Destroy a `ZEBU_ValueChange_Handles` handle list.

Parameters:

handles Handler to a `ZEBU_ValueChange_HandleList`.

6.22.2.8 void* ZEBU_ValueChange_HandleList_getHandle (ZEBU_ValueChange_HandleList * *handles*, unsigned int *index*)

Return the handle at position `index` in the handle list.

Parameters:

handles Handler to a `ZEBU_ValueChange_HandleList`.

index Position of the element in the list. If this is greater than the vector size, a NULL handle is returned.

Return values:

A handle to the changed signal.

6.22.2.9 unsigned int ZEBU_ValueChange_HandleList_getSize (ZEBU_ValueChange_HandleList ∗ *handles*)

Return the size of the handle list.

Parameters:

handles Handler to a `ZEBU_ValueChange_HandleList`.

Return values:

The number of elements in the handle list.

6.22.2.10 void∗ ZEBU_ValueChange_Iterator_getHandle (ZEBU_ValueChange_Iterator ∗ *iterator*)

Return the handle of the current changed signal.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

Return values:

A handle to the changed signal

See also:

ZEBU_ValueChange_createIterator

6.22.2.11 const char∗ ZEBU_ValueChange_Iterator_getName (ZEBU_ValueChange_Iterator ∗ *iterator*)

Return the hierarchical gate name of the current changed signal.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

Return values:

A hierarchical get name.

See also:

ZEBU_ValueChange_createIterator

6.22.2.12 void ZEBU_ValueChange_Iterator_goToFirst (ZEBU_ValueChange_Iterator ∗ *iterator*)

Move iterator to the first changed signal.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

See also:

ZEBU_ValueChange_createIterator

6.22.2.13 void ZEBU_ValueChange_Iterator_goToNext (ZEBU_ValueChange_Iterator ∗ *iterator*)

Move iterator to the next changed signal.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

See also:

ZEBU_ValueChange_createIterator

6.22.2.14 int ZEBU_ValueChange_Iterator_isAtEnd (ZEBU_ValueChange_Iterator ∗ *iterator*)

Test if iterator passed last changed signal.

Parameters:

iterator Handler to a `ZEBU_ValueChange_Iterator`.

See also:

ZEBU_ValueChange_createIterator

6.22.2.15 int ZEBU_ValueChange_waitDriver (ZEBU_Board ∗ *board*, ZEBU_Driver ∗ *driver*, unsigned int *timeout*)

Waits for a value change trigger event -or- timeout while running the clock through a cosimulation-driver.

Parameters:

board Handler to a `ZEBU_Board`.

driver The cosimulation-driver to run clock.

timeout Maximum number of cycles before stopping.

Return values:

0 if a timeout expired

$>$***0*** if the value change trigger fired

6.22.2.16 int ZEBU_ValueChange_waitDriverEx (ZEBU_Board ∗ *board*, ZEBU_Driver ∗ *driver*, unsigned int *triggers*, unsigned int ∗ *fired*, unsigned int *timeout*)

Waits for a value change trigger event -or- another trigger event -or- timeout while running the clock through a cosimulation-driver.

Parameters:

board Handler to a `ZEBU_Board`.

driver The cosimulation-driver to run clock.

triggers Triggers to stop on. Set bit i to 1 to stop on trigger i (on the 16 lsb).

→ ***fired*** Fired triggers. Bit i set to 1 for trigger i.

timeout Maximum number of cycles before stopping.

Return values:

0 if a timeout expired

$>$***0*** if the value change trigger fired

$<$***0*** if some other triggers fired

Remarks:

Even if this method returns >0, which means that the value change trigger has fired, you should also check the `fired` out parameter for another fired trigger.

6.22.2.17 int ZEBU_ValueChange_waitLogicAnalyzer (ZEBU_Board ∗ *board*, const char ∗ *clockName*, const char ∗ *edgeName*, unsigned int *timeout*)

Waits for a value change trigger event -or- a timeout through a logic analyzer.

Parameters:

board Handler to a `ZEBU_Board`.

clockName Name of the clock used to change logic Analyzer state

edgeName Name of the edge used to change logic Analyzer state. Valid names are "posedge" or "negedge".

timeout Maximum number of cycles before aborting.

Return values:

0 if a timeout expired

>***0*** if the value change trigger fired

6.22.2.18 int ZEBU_ValueChange_waitLogicAnalyzerEx (ZEBU_Board ∗ *board*, const char ∗ *clockName*, const char ∗ *edgeName*, unsigned int *triggers*, unsigned int ∗ *fired*, unsigned int *timeout*)

Waits for a value change trigger event -or- another trigger event -or- a timeout through a logic analyzer.

Parameters:

board Handler to a `ZEBU_Board`.

clockName Name of the clock used to change logic Analyzer state

edgeName Name of the edge used to change logic Analyzer state. Valid names are "posedge" or "negedge".

triggers Triggers to stop on. Set bit i to 1 to stop on trigger i (on the 16 lsb).

→ ***fired*** Fired triggers. Bit i set to 1 for trigger i.

timeout Maximum number of cycles before aborting.

Return values:

0 if a timeout expired

>***0*** if the value change trigger fired

<***0*** if some other triggers fired

Remarks:

Even if this method returns >0, which means that the value change trigger has fired, you should also check the `fired` out parameter for another fired trigger.

6.23 ZEBU_WaveFile.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- ZEBU_WaveFile ∗ ZEBU_WaveFile_open (ZEBU_Board ∗board, const char ∗filename, const char ∗clockName, int level)

  open a waveform file

- int ZEBU_WaveFile_destroy (ZEBU_WaveFile ∗waveFile)

  destroy the waveform file open from ZEBU_WaveFile_open

- int ZEBU_WaveFile_dumpvars (ZEBU_WaveFile ∗waveFile, ZEBU_Signal ∗signal)

  select internal register to dump

- int ZEBU_WaveFile_dumpvars2 (ZEBU_WaveFile ∗waveFile, const char ∗name, const int depth)

  select internal register to dump

- int ZEBU_WaveFile_dumpon (ZEBU_WaveFile ∗waveFile)

  resume the dump

- int ZEBU_WaveFile_dumpoff (ZEBU_WaveFile ∗waveFile)
- int ZEBU_WaveFile_flush (ZEBU_WaveFile ∗waveFile)

  fush the content of the waveform file to the disk

- int ZEBU_WaveFile_close (ZEBU_WaveFile ∗waveFile)

  close the waveform file open from the constructor

6.23.1 Function Documentation

6.23.1.1 int ZEBU_WaveFile_close (ZEBU_WaveFile ∗ *waveFile*)

close the waveform file open from the constructor

Parameters:

waveFile handler to a `ZEBU_WaveFile`

Returns:

int

Return values:

0 OK

>*0* KO

6.23.1.2 int ZEBU_WaveFile_destroy (ZEBU_WaveFile ∗ *waveFile*)

destroy the waveform file open from ZEBU_WaveFile_open

Parameters:

waveFile handler to a `ZEBU_WaveFile`

Returns:

int

Return values:

0 OK

>*0* KO

6.23.1.3 int ZEBU_WaveFile_dumpoff (ZEBU_WaveFile ∗ *waveFile*)

Parameters:

waveFile handler to a `ZEBU_WaveFile`

switch partial readback waveform dump off. This is default.

Returns:

int

Return values:

0 OK

>*0* KO

See also:

ZEBU_WaveFile_dumpvars
ZEBU_WaveFile_dumpon
ZEBU_WaveFile_dumpfile

6.23.1.4 int ZEBU_WaveFile_dumpon (ZEBU_WaveFile * *waveFile*)

resume the dump

Parameters:

waveFile handler to a `ZEBU_WaveFile`

switch partial readback waveform dump on

Returns:

int

Return values:

0 OK

$>$***0*** KO

See also:

ZEBU_WaveFile_dumpvars
ZEBU_WaveFile_dumpfile
ZEBU_WaveFile_dumpoff

6.23.1.5 int ZEBU_WaveFile_dumpvars (ZEBU_WaveFile * *waveFile*, ZEBU_Signal * *signal*)

select internal register to dump

Parameters:

waveFile handler to a `ZEBU_WaveFile`

signal handler to a `ZEBU_Signal` to be dumped. If no parameter is given, or `NULL`, all signals are dumped.

Note:

no signal can be added after first run.

Returns:

int

Return values:

0 OK

$>$***0*** KO

See also:

ZEBU_WaveFile_dumpfile
ZEBU_WaveFile_dumpon
ZEBU_WaveFile_dumpoff

6.23.1.6 int ZEBU_WaveFile_dumpvars2 (ZEBU_WaveFile ∗ *waveFile*, const char ∗ *name*, const int *depth*)

select internal register to dump

Parameters:

waveFile handler to a `ZEBU_WaveFile`

name path to an internal instance or signal. If no parameter is given, or `NULL`, all signals are dumped.

depth number of hierarchy level to dump.

Note:

no signal can be added after first run.

Returns:

int

Return values:

0 OK

>***0*** KO

See also:

ZEBU_WaveFile_dumpfile
ZEBU_WaveFile_dumpon
ZEBU_WaveFile_dumpoff

6.23.1.7 int ZEBU_WaveFile_flush (ZEBU_WaveFile ∗ *waveFile*)

fush the content of the waveform file to the disk

Parameters:

waveFile handler to a `ZEBU_WaveFile`

Returns:

int

Return values:

0 OK

>***0*** KO

6.23.1.8 ZEBU_WaveFile ∗ ZEBU_WaveFile_open (ZEBU_Board ∗ *board*, const char ∗ *filename*, const char ∗ *clockName*, int *level*)

open a waveform file

Note:

not supported in zTide environment

The file must be opened after the openning of the ZeBu session and must be closed before the closing of the ZeBu session. If the file is not closed before the closing of the ZeBu session it will cause a deadlock.

It is impossible to access to dynamic probes that have not been selected until the file is open. It is impossible to access to clocks until the file is dumping. Such accesses can cause deadlocks.

Parameters:

board handler to a `ZEBU_Board`

filename name of the waveform file

- if extension is ".bin", file is dumped in a proprietary binary format
- if extension is ".vcd", file is dumped in VCD format
- if extension is ".fsdb", file is dumped in VCD format

clockName name of the signal sampling clock.

level compression level. Takes value between 0 and 9. 0 is fastest, and 9 is best. Default 0.

See also:

ZEBU_Board_open

Chapter 7

Zebu C API Page Documentation

7.1 Additional Resources

7.1.1 EVE ressources

- The ZeBu Reference Manual provides detailed information on program commands, memory models, libraries and files necessary to compile and verify your design with the ZeBu verification platform.

- The Zebu Installation Manual describes how to install the ZeBu system, including the software and hardware. This manual also provides information about how to run diagnostics to check the ZeBu board.

- The ZeBu User's Manual describes how to use the ZeBu platform for verifying a design under test.

- The ZeBu Tutorial gives several examples on how to use the ZeBu platform for verifying a design under test.

7.1.2 Other resources

- Using the GCC compiler collection, `http://gcc.gnu.org/onlinedocs/gcc-3.4.6/gcc`

7.2 Bug List

Member ZEBU_Port_date(ZEBU_Port ∗port) always return 0xffffffff

Index