The Fastest Verification

ZeBu™
C++ API Reference Manual

Version 6_3_1

May 2011

Contents

Chapter 1

ZeBu C++ API Namespace Index

1.1 ZeBu C++ API Namespace List

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

Chapter 2

ZeBu C++ API Hierarchical Index

2.1 ZeBu C++ API Class Hierarchy

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

Chapter 3

ZeBu C++ API Class Index

3.1 ZeBu C++ API Class List

Here are the classes, structs, unions and interfaces 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 Namespace Documentation

5.1 ZEBU Namespace Reference

5.1.1 Detailed Description

ZeBu C++ API namespace

```
using namespace ZEBU;
```

Classes

- class Board

  Implement public interface class for ZeBu board access.

- class Board::SignalIterator

  Implement public iterator on internal signals.

- class Board::MemoryIterator

  Implement public iterator on memories.

- class Board::DriverInfoIterator

  Implement public iterator on driver information.

- class Board::PortInfoIterator

  Implement public iterator on driver information.

- class CCall

  Allow controlling C_CALL or function calls.

- class Clock

  public interface class for ZeBu clocks.

- class Driver

  Implement public interface class for ZeBu drivers.

- class Driver::SignalIterator

  Implement public iterator on driver signals.

- class CDriver

  Implement ZeBu driver for simple C cosimulation.

- class Monitor

  Implement ZeBu monitor driver base class.

- class TraceMemory

  Implement ZeBu SRAM monitor driver. TraceMemory is used for dumping VCD with SRAM.

- class MckCDriver

  Implement a multi-clock driver.

- class PatternDriver

  multi-clocks pattern driver.

- class Events

  This class provides methods to register/unregister a user fonction that can be used to handle public events that are fired by ZeBu.

- class FastHardwareState

  Allow to capture fastly the hardware state and the software state of a ZeBu session and then to save it to disk.

- class Filter

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

- class FlexibleLocalProbeFile

Allows controlling a group of flexible local groups in ZeBu hardware and dumping their traces to disk.

- class LocalTracer

  Base class that allows controlling a local tracer in ZeBu hardware.

- class LocalTraceDumper

  Allows controlling a local tracer in ZeBu hardware and dumping its trace to disk.

- class LocalTraceReader

  Allows controlling a local tracer in ZeBu hardware and reading values of tracer's signals.

- class LocalTraceImporter

  Allows controlling a local tracer in ZeBu hardware and importing tracer's signal values in a DPI function.

- class LocalTracerGroup

  Base class that allows controlling a group of local tracers in ZeBu hardware.

- class LocalTraceDumperGroup

  Allows controlling a group of local tracers in ZeBu hardware and dumping their traces to disk.

- class LocalTraceReaderGroup

  Allows controlling a group of local tracers in ZeBu hardware and reading values of signals of all tracers in the group.

- class LocalTraceImporterGroup

  Allows controlling a group of local tracers in ZeBu hardware and importing signal values of all tracers of the group in a DPI function.

- class LogicAnalyzer

  interface for ZeBu logic analyer

- class APosterioriLoopDetector

  This class provides several methods to work with the combinationnal loop detector in the "a posteriori" detection mode.

- class APosterioriLoopDetector::Iterator

  Implements an iterator on oscillating loops.

- class InteractiveLoopDetector

This class provides several methods to work with the combinationnal loop detector in the "interactive" detection mode.

- class InteractiveLoopDetector::Iterator

  Implements an iterator on oscillating loops.

- class LoopBreak

  This class provides methods to deal with in-cycle oscillating loop breaking.

- class Memory

  Implement public interface class for Memory instances in the DUT.

- class **PartMemory**
- class PartMemoryBuilder

  Class used to create complex part memories.

- class **PartSignal**
- class Port

  interface for ZeBu port.

- class TxPort

  interface for ZeBu transmit port.

- class RxPort

  interface for ZeBu receive port.

- class Signal

  Implement public interface class for signals.

- class Sniffer

  Allow saving repetively the state of the DUT and to intercept continuously its input stream into a sniff folder.

- class SVA

  Allow controlling System Verilog Assertions.

- class Trigger

  Implement public interface class for triggers.

- class ValueChange

  This class provides several methods to work with the value change trigger.

- class ValueChange::Iterator

Implement public iterator on value change.

- class WaveFile

  Implement public interface class for readback waveform dump.

Chapter 6

ZeBu C++ API Class Documentation

6.1 _ZEBU_LocalTraceImporter_SignalUnion Union Reference

6.1.1 Detailed Description

Bit-vector union for inputs of local tracers passed to DPI import functions

Public Attributes

- const unsigned char * **bit**
- const unsigned int * **vector**

6.2 APosterioriLoopDetector Class Reference

6.2.1 Detailed Description

This class provides several methods to work with the combinationnal 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
driver->run(nbCycle);

// Test if an oscillation has been detected during the last run
if (ZEBU::APosterioriLoopDetector::checkDetectors(board))
{
    // Create a new oscillating loop iterator
    ZEBU::APosterioriLoopDetector::Iterator loopIterator(board);

    // Print the name of each oscillating loop
    for (loopIterator.goToFirst(); !loopIterator.isAtEnd(); loopIterator.goToNext()) {
        printf("oscillating loop name = %s\n", loopIterator.getName());
    }

    // Reset the detectors
    ZEBU::APosterioriLoopDetector::resetDetectors(board);
}
```

Static Public Member Functions

- bool checkDetectors (Board ∗board) throw (std::exception)

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

- void resetDetectors (Board ∗board) throw (std::exception)

  Resets the combinational loop detectors.

6.2.2 Member Function Documentation

6.2.2.1 bool checkDetectors (Board ∗ *board*) throw (std::exception) [static]

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

Parameters:
board A pointer to a `ZEBU::Board` object.

Returns:
A boolean value indicating whether an oscillating loop has been detected or not.

See also:
ZEBU::APosterioriLoopDetector::resetDetectors
ZEBU::APosterioriLoopDetector::Iterator

6.2.2.2 void resetDetectors (Board * *board*) throw (std::exception) [static]

Resets the combinational loop detectors.

Parameters:
board A pointer to a `ZEBU::Board` object.

See also:
ZEBU::APosterioriLoopDetector::checkDetectors

6.3 APosterioriLoopDetector::Iterator Class Reference

6.3.1 Detailed Description

Implements an iterator on oscillating loops.

```
ZEBU::APosterioriLoopDetector::Iterator loopIterator(zebu);
for (loopIterator.goToFirst(); !loopIterator.isAtEnd(); loopIterator.goToNext()) {
    printf("oscillating loop name = %s\n", loopIterator.getName());
}
```

See also:
ZEBU::APosterioriLoopDetector::checkDetectors

Public Member Functions

- Iterator (Board ∗board) throw (std::exception)

  Constructs and initializes a new `Zebu::APosterioriLoopDetector::Iterator` *instance.*

- ∼Iterator () throw (std::exception)

  Destroys a `Zebu::APosterioriLoopDetector::Iterator` *instance.*

- void goToFirst () throw (std::exception)

  Moves iterator to the first oscillating loop.

- void goToNext () throw (std::exception)

  Moves iterator to the next oscillating loop.

- bool isAtEnd () const throw (std::exception)

  Tests if iterator passed last oscillating loop.

- const char ∗ **getName** () const throw (std::exception)

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Iterator (Board ∗ *board*) throw (std::exception)

Constructs and initializes a new `Zebu::APosterioriLoopDetector::Iterator` instance.

Parameters:
board A pointer to a `ZEBU::Board` object.

6.3.2.2 ~Iterator () throw (std::exception)

Destroys a `Zebu::APosterioriLoopDetector::Iterator` instance.

6.3.3 Member Function Documentation

6.3.3.1 void goToFirst () throw (std::exception)

Moves iterator to the first oscillating loop.

See also:
Zebu::APosterioriLoopDetector::Iterator::goToNext
Zebu::APosterioriLoopDetector::Iterator::isAtEnd

6.3.3.2 void goToNext () throw (std::exception)

Moves iterator to the next oscillating loop.

See also:
Zebu::APosterioriLoopDetector::Iterator::goToFirst
Zebu::APosterioriLoopDetector::Iterator::isAtEnd

6.3.3.3 bool isAtEnd () const throw (std::exception)

Tests if iterator passed last oscillating loop.

Returns:
A boolean value indicating if the iterator is at the end.

See also:
Zebu::APosterioriLoopDetector::Iterator::goToFirst
Zebu::APosterioriLoopDetector::Iterator::goToNext

6.4 Board Class Reference

6.4.1 Detailed Description

Implement public interface class for ZeBu board access.

Public Member Functions

- bool setMsgVerboseMode (bool verbose) throw (std::exception)

  Set message verbose mode.

- unsigned int restoreLogicState (const char ∗filename, const Filter ∗filter=0, const char ∗initMemFile=0, const unsigned int severity=0) throw (std::exception)

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

- unsigned int init (const char ∗initMemFile, const char ∗logicStateFile, const unsigned int severity=0) throw (std::exception)

  initialize ZeBu board

- unsigned int init (const char ∗initMemFile=0) throw (std::exception)

  initialize ZeBu board

- bool isHDP () const throw (std::exception)

  test if the board is a Hardware Development Platform

- unsigned int close (const char ∗string=0) throw (std::exception)

  close ZeBu session

- void closeThread () throw (std::exception)

  close ZeBu thread, release locks acquired by the current thread

- const char ∗ getZebuWorkPath () const throw (std::exception)

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

- bool check (int severity, bool verbose=true) throw (std::exception)

  check ZeBu initialization

- bool check (bool verbose=true) throw (std::exception)

  check ZeBu initialization

- bool checkRACC (bool verbose=true) throw (std::exception)

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.

- void serviceLoop () throw (std::exception)

  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 proxies.

- bool serviceLoop (ServiceLoopHandler g, void ∗context, const long long unsigned int portGroupNumber) throw (std::exception)

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

- bool serviceLoop (ServiceLoopHandler g, void ∗context, const unsigned int portGroupNumber) throw (std::exception)

  Board::serviceLoop.

- bool serviceLoop (ServiceLoopHandler g, void ∗context) throw (std::exception)

  Board::serviceLoop.

- void loop () throw (std::exception)

  obselete

- void run () throw (std::exception)

  run a cycle for each driver which registered a callback

- bool isClockSystemEnabled () throw (std::exception)

  test if clock system is enabled

- unsigned int waitClockSystemEnable () throw (std::exception)

  wait until clock system is enabled

- void registerDriver (Driver ∗driver, void(∗callback)(void ∗data), void ∗data) throw (std::exception)

  associate a callback to a driver

- Clock ∗ getClock (const char ∗name) const throw (std::exception)

  get a clock handler

- Clock ∗ getClock (const char ∗name, const char ∗filename) const throw (std::exception)

get a clock handler

- Clock ∗ getClock (const char ∗name, const char ∗waveform, const char ∗mode, unsigned int frequency, const char ∗groupName) const throw (std::exception)

  get a clock handler

- void getDriverClockFrequency (unsigned int &frequency) throw (std::exception)

  get the Driver Clock Frequency in kHz

- Driver ∗ getDriver (const char ∗name) const throw (std::exception)

  get a driver handler

- PatternDriver ∗ getPatternDriver (const char ∗name) const throw (std::exception)

  get a pattern driver handler

- PatternDriver ∗ getFirstPatternDriver () const throw (std::exception)

  get the first pattern driver handler

- Signal ∗ getSignal (const char ∗name) const throw (std::exception)

  get a handler on an internal signal of the DUT.

- Memory ∗ getMemory (const char ∗name) const throw (std::exception)

  get a memory handler

- Trigger ∗ getTrigger (const char ∗name) const throw (std::exception)

  get a trigger handler

- LogicAnalyzer ∗ getLogicAnalyzer () const throw (std::exception)

  get the logic analyzer handler

- void getTriggerNameList (unsigned int &numberOfTriggers, const char ∗∗&triggerNameList) const throw (std::exception)

  get the trigger name list

- void getTriggerNameList (int &numberOfTriggers, const char ∗∗&triggerNameList) const throw (std::exception)

  absolete

- void getClockGroupNameList (unsigned int &numberOfClockGroup, const char ∗∗&clockGroupNameList) const throw (std::exception)

  get the clock group name list

- void getClockGroupNameList (int &numberOfClockGroup, char **&clockGroupNameList) const throw (std::exception)

  absolete

- void getClockNameList (const char *groupName, unsigned int &numberOfClocks, const char **&clockNameList) const throw (std::exception)

  get the clock groups name list

- void getClockNameList (char *groupName, int &numberOfClocks, char **&clockNameList) const throw (std::exception)

  absolete

- void dumpon () throw (std::exception)

  resume the dump

- void dumpoff () throw (std::exception)

  suspend the dump

- void dumpvars (Signal *signal=0) throw (std::exception)

  select internal register to dump

- void dumpvars (const char *name, int depth) throw (std::exception)

  select internal register to dump

- void dumpfile (const char *filename, int level=0) throw (std::exception)

  specify the name of a waveform file

- void dumpflush () throw (std::exception)

  fush the content of the waveform file open from ::dumpfile to the disk

- void dumpclosefile () throw (std::exception)

  close the waveform file open from ::dumpfile

- void closeDumpfile () throw (std::exception)

  absolete

- void writeRegisters () throw (std::exception)

  force register write

- void readRegisters () throw (std::exception)

  force register read

- unsigned int loadTriggers (const char ∗filename) throw (std::exception)

  load a trigger expression or a file containing a trigger expression

- void saveHardwareState (const char ∗filename, const Filter ∗filter) throw (std::exception)

  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

- void saveHardwareState (const char ∗filename) throw (std::exception)

  Board::saveHardwareState.

- void save (const char ∗filename) throw (std::exception)

  absolete

- void saveLogicState (const char ∗filename, const Filter ∗filter=0) throw (std::exception)

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

- void selectSignalsToRandomize (const char ∗signalList=0, const int invert=0) throw (std::exception)

  select signals to randomize

- void selectMemoriesToRandomize (const char ∗memoryList=0, const int invert=0) throw (std::exception)

  select memories to randomize

- void selectObjectsToRandomize (const Filter ∗filter) throw (std::exception)

  select signals and memories to randomize

- void randomize (const unsigned int seed) throw (std::exception)

  set signals and memories to a pseudo random value. Set all signals and memories if no selection has been done through the methods ::selectSignalsToRandomize, ::selectMemoriesToRandomize, or ::selectObjectsToRandomize

- void randomizeSignals (const unsigned int seed) throw (std::exception)

  set signals to a pseudo random value. Set all signals if no selection has been done through the methods ::selectSignalsToRandomize or ::selectObjectsToRandomize

- void randomizeMemories (const unsigned int seed) throw (std::exception)

  set memories to a pseudo random value. Set all memories if no selection has been done through the methods ::selectMemoriesToRandomize, or ::selectObjectsToRandomize

- ~Board () throw (std::exception)

  destructor

- unsigned int close (int code, const char *string=0) throw (std::exception)

  close ZeBu session

- unsigned int getBoardNumber () const

  private

Static Public Member Functions

- const char * getPlatformName () throw (std::exception)

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

- const char * getVersion () throw (std::exception)

  return the product version.

- const char * getLibraryName () throw (std::exception)

  return the name of the used library.

- Board * open (const char *zebuWorkPath="./zebu.work", const char *designFile=0, const char *processName="default_process") throw (std::exception)

  open ZeBu session

- Board * restoreHardwareState (const char *filename, const char *zebuWorkPath="./zebu.work", const char *designFile=0, const char *processName="default_process") throw (std::exception)

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

- Board * restore (const char *filename, const char *zebuWorkPath="./zebu.work", const char *designFile=0, const char *processName="default_process") throw (std::exception)

  absolete

Protected Member Functions

- Board (unsigned int board) throw (std::exception)

  protected constructor

Friends

- Board * **new_Board** (unsigned int)

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ~Board () throw (std::exception)

destructor

6.4.2.2 Board (unsigned int *board*) throw (std::exception) `[protected]`

protected constructor

6.4.3 Member Function Documentation

6.4.3.1 bool check (bool *verbose* = `true`) throw (std::exception)

check ZeBu initialization

See also:
Board::check

6.4.3.2 bool check (int *severity*, bool *verbose* = `true`) throw (std::exception)

check ZeBu initialization

Parameters:

verbose boolean value. If true, method displays error

severity integer value. The lower the severity is, the more defects are interpreted as errors.

Returns:
bool

Return values:

false failed.

true successful

Note:
do not run if status is false.

```
if(!zebuBoard->check()) {
    zebuBoard->close();
    exit(1);
}
```

See also:
Board::open
Board::init
Board::close

6.4.3.3 bool checkRACC (bool *verbose* = `true`) throw (std::exception)

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.

Parameters:
verbose boolean value. If true, method displays error

Returns:
bool

Return values:
false failed.

true successful

Note:
do not run if status is false.

```
if(!zebuBoard->checkRACC()) {
    zebuBoard->close();
    exit(1);
}
```

See also:
Board::open
Board::init
Board::close

6.4.3.4 unsigned int close (int *code*, const char * *string* = `0`) throw (std::exception)

close ZeBu session

Parameters:

code unused

string default 0. Message to display when closing.

Returns:

unsigned int

Return values:

0 if successfull.

```
zebuBoard->close(7);
```

Deprecated

You should use other close method.

See also:

Board::open
Board::init
Board::check

6.4.3.5 unsigned int close (const char * *string* = 0) throw (std::exception)

close ZeBu session

Parameters:

string default 0. Message to display when closing.

Returns:

unsigned int

Return values:

0 if successfull.

```
zebuBoard->close();
```

See also:

Board::open
Board::init
Board::check

6.4.3.6 void closeDumpfile () throw (std::exception)

absolete

See also:

Board::dumpclosefile

6.4.3.7 void closeThread () throw (std::exception)

close ZeBu thread, release locks acquired by the current thread

6.4.3.8 void dumpclosefile () throw (std::exception)

close the waveform file open from ::dumpfile

Note:
not supported in zTide environment

6.4.3.9 void dumpfile (const char * *filename*, int *level* = 0) throw (std::exception)

specify the name of a waveform file

Note:
not supported in zTide environment

Parameters:

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

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

See also:
Board::dumpclosefile
Board::dumpvars
Board::dumpon
Board::dumpoff

6.4.3.10 void dumpflush () throw (std::exception)

fush the content of the waveform file open from ::dumpfile to the disk

Note:
not supported in zTide environment

6.4.3.11 void dumpoff () throw (std::exception)

suspend the dump

Note:
not supported in zTide environment

switch partial readback waveform dump off. This is default

See also:
Board::dumpvars
Board::dumpon
Board::dumpfile

6.4.3.12 void dumpon () throw (std::exception)

resume the dump

Note:
not supported in zTide environment

switch partial readback waveform dump on

See also:
Board::dumpvars
Board::dumpfile
Board::dumpoff

6.4.3.13 void dumpvars (const char * *name*, int *depth*) throw (std::exception)

select internal register to dump

Note:
not supported in zTide environment

Parameters:
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:
Board::dumpfile
Board::dumpon
Board::dumpoff

6.4.3.14 void dumpvars (Signal * *signal* = 0) throw (std::exception)

select internal register to dump

Note:
not supported in zTide environment

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

Note:
no signal can be added after first run.

See also:
Board::dumpfile
Board::dumpon
Board::dumpoff

6.4.3.15 unsigned int getBoardNumber () const [inline]

private

6.4.3.16 Clock* getClock (const char * *name*, const char * *waveform*, const char * *mode*, unsigned int *frequency*, const char * *groupName*) const throw (std::exception)

get a clock handler

Note:
not supported in zTide environment

Parameters:
name clock name

waveform clock waveform

mode clock mode

frequency clock frequency

groupName name of the group of the clock

See also:
Board::getDriver
Board::getPatternDriver
Board::getMemory
Board::getSignal

6.4.3.17 Clock* getClock (const char * *name*, const char * *filename*) const throw (std::exception)

get a clock handler

Note:
not supported in zTide environment

Parameters:
name clock name

filename name of the file with parameters to initialize the clock

See also:
Board::getDriver
Board::getPatternDriver
Board::getMemory
Board::getSignal

6.4.3.18 Clock* getClock (const char * *name*) const throw (std::exception)

get a clock handler

Note:
not supported in zTide environment

Parameters:
name clock name

See also:
Board::getDriver
Board::getPatternDriver
Board::getMemory
Board::getSignal

6.4.3.19 void getClockGroupNameList (int & *numberOfClockGroup*, char ∗∗& *clockGroupNameList*) const throw (std::exception)

absolete

See also:
Board::getClockGroupNameList

6.4.3.20 void getClockGroupNameList (unsigned int & *numberOfClockGroup*, const char ∗∗& *clockGroupNameList*) const throw (std::exception)

get the clock group name list

Note:
not supported in zTide environment

Parameters:
numberOfClockGroup reference on the number of clock groups.

clockGroupNameList reference on a pointer on clock group names

See also:
Board::getTriggerNameList

6.4.3.21 void getClockNameList (char ∗ *groupName*, int & *numberOfClocks*, char ∗∗& *clockNameList*) const throw (std::exception)

absolete

See also:
Board::getClockNameList

6.4.3.22 void getClockNameList (const char ∗ *groupName*, unsigned int & *numberOfClocks*, const char ∗∗& *clockNameList*) const throw (std::exception)

get the clock groups name list

Note:
not supported in zTide environment

Parameters:

groupName clock group

numberOfClocks reference on the number of clocks

clockNameList reference on a pointer on clock names

See also:

Board::getTriggerNameList

6.4.3.23 Driver* getDriver (const char * *name*) const throw (std::exception)

get a driver handler

Note:

not supported in zTide environment

Parameters:

name driver's name as declared in dve file.

See also:

Board::getClock
Board::getPatternDriver
Board::getMemory
Board::getSignal
Board::getTrigger

6.4.3.24 void getDriverClockFrequency (unsigned int & *frequency*) throw (std::exception)

get the Driver Clock Frequency in kHz

Parameters:

frequency reference on the frequency.

See also:

Board::getClock
Clock::counter

6.4.3.25 PatternDriver* getFirstPatternDriver () const throw (std::exception)

get the first pattern driver handler

Note:

not supported in zTide environment

6.4.3.26 const char* getLibraryName () throw (std::exception) [static]

return the name of the used library.

Returns:
string "<libary name> <environment>".

6.4.3.27 LogicAnalyzer* getLogicAnalyzer () const throw (std::exception)

get the logic analyzer handler

Note:
not supported in zTide environment

See also:
LogicAnalyzer

6.4.3.28 Memory* getMemory (const char * *name*) const throw (std::exception)

get a memory handler

Note:
not supported in zTide environment

Parameters:
name full path name to the memory.

See also:
Board::getClock
Board::getDriver
Board::getPatternDriver
Board::getSignal
Board::getTrigger

6.4.3.29 PatternDriver* getPatternDriver (const char * *name*) const throw (std::exception)

get a pattern driver handler

Note:
not supported in zTide environment

Parameters:
name driver's name as declared in dve file.

See also:
Board::getClock
Board::getDriver
Board::getMemory
Board::getSignal
Board::getTrigger

6.4.3.30 const char∗ getPlatformName () throw (std::exception) [static]

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

6.4.3.31 Signal∗ getSignal (const char ∗ *name*) const throw (std::exception)

get a handler on an internal signal of the DUT.

Note:
not supported in zTide environment

Parameters:
name hierarchical name of the Signal from top of DUT.

```
Signal *sig  =  zebuBoard->getSignal("top.inst.sig");
Signal &vect  = *zebuBoard->getSignal("top.inst.vect");
Signal &vect_7 = *zebuBoard->getSignal("top.inst.vect[7]");
```

See also:
Board::getClock
Board::getDriver
Board::getPatternDriver
Board::getMemory
Board::getTrigger

6.4.3.32 Trigger∗ getTrigger (const char ∗ *name*) const throw (std::exception)

get a trigger handler

Note:
not completly supported in zTide environment

Parameters:
name trigger's name as declared in dve file.

Note:
it can be a static or a dynamic trigger

```
Trigger &trigger1 = *board->getTrigger("trigger1");
```

See also:
Board::getClock
Board::getDriver
Board::getPatternDriver
Board::getMemory
Board::getSignal

6.4.3.33 void getTriggerNameList (int & *numberOfTriggers*, const char ∗∗& *triggerNameList*) const throw (std::exception)

absolete

See also:
Board::getTriggerNameList

6.4.3.34 void getTriggerNameList (unsigned int & *numberOfTriggers*, const char ∗∗& *triggerNameList*) const throw (std::exception)

get the trigger name list

Note:
not supported in zTide environment

Parameters:
numberOfTriggers reference on the number of triggers.

triggerNameList reference on a pointer on trigger names

Note:
all types of triggers are inserted in the list

```
int numberOfTriggers = 0;
const char **trigNameList = NULL;
zebu->getTriggerNameList(numberOfTriggers, trigNameList);
for(int i=0;i<numberOfTriggers;i++) {
  cout << "trigger " << i << " => " <<  trigNameList[i] << endl;
}
Trigger &trigger1 = *board->getTrigger("trigger1");
```

See also:
Board::getTrigger

6.4.3.35 const char* getVersion () throw (std::exception) [static]

return the product version.

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

6.4.3.36 const char* getZebuWorkPath () const throw (std::exception)

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

See also:
Board::open
Board::restore

6.4.3.37 unsigned int init (const char * *initMemFile* = 0) throw (std::exception)

initialize ZeBu board

See also:
Board::init

6.4.3.38 unsigned int init (const char * *initMemFile*, const char * *logicStateFile*, const unsigned int *severity* = 0) throw (std::exception)

initialize ZeBu board

Note:
not supported in zTide environment

Parameters:

initMemFile default NULL. File name. This file give list of memories to initialize with file corresponding file.

logicStateFile default NULL. File name. This file give the logic state to restore before the starting of clocks.

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

Returns:
unsigned int

Return values:
0 if successfull.

```
if(zebuBoard->init()) {
    cerr << "Cannot init Board" << endl;
    zebuBoard->close();
    exit(1);
}
```

See also:
Board::open
Board::close
Board::check

6.4.3.39 bool isClockSystemEnabled () throw (std::exception)

test if clock system is enabled

Note:
not supported in zTide environment

See also:
Board::getClock
Board::waitClockSystemEnable

6.4.3.40 bool isHDP () const throw (std::exception)

test if the board is a Hardware Development Platform

Note:
board has to be opened to call this method

Return values:
true if the board is a Hardware Development Platform

6.4.3.41 unsigned int loadTriggers (const char * *filename*) throw (std::exception)

load a trigger expression or a file containing a trigger expression

Note:
not supported in zTide environment

Parameters:

filename name of definition string

```
zebu->loadTriggers("trig0 <= (output == 2'b10)");
```

See also:

Trigger::operator =

6.4.3.42 void loop () throw (std::exception)

obselete

See also:

serviceLoop

6.4.3.43 Board* open (const char * *zebuWorkPath* = "./zebu.work", const char * *designFile* = 0, const char * *processName* = "default_process") throw (std::exception) [static]

open ZeBu session

Parameters:

processName default "default_process"

designFile default "designFeatures"

zebuWorkPath default "./zebu.work"

Returns:

Board*

Return values:

handler on Board. NULL if open failed.

```
Board *zebuBoard = Board::open();
if(zebuBoard == NULL) {
    cerr << "Cannot open Board" << endl;
    exit(1);
}
```

See also:

Board::init
Board::close
Board::check

6.4.3.44 void randomize (const unsigned int *seed*) throw (std::exception)

set signals and memories to a pseudo random value. Set all signals and memories if no selection has been done through the methods ::selectSignalsToRandomize, ::selectMemoriesToRandomize, or ::selectObjectsToRandomize

Parameters:
seed seed of the sequence of value to set

6.4.3.45 void randomizeMemories (const unsigned int *seed*) throw (std::exception)

set memories to a pseudo random value. Set all memories if no selection has been done through the methods ::selectMemoriesToRandomize, or ::selectObjectsToRandomize

Parameters:
seed seed of the sequence of value to set

6.4.3.46 void randomizeSignals (const unsigned int *seed*) throw (std::exception)

set signals to a pseudo random value. Set all signals if no selection has been done through the methods ::selectSignalsToRandomize or ::selectObjectsToRandomize

Parameters:
seed seed of the sequence of value to set

6.4.3.47 void readRegisters () throw (std::exception)

force register read

Note:
not supported in zTide environment

6.4.3.48 void registerDriver (Driver * *driver*, void(*)(void *data) *callback*, void * *data*) throw (std::exception)

associate a callback to a driver

Note:
not supported in zTide environment

Parameters:

driver handler to the driver

callback pointer to the callback function

data argument of the callback function

6.4.3.49 Board∗ restore (const char ∗ *filename*, const char ∗ *zebuWorkPath* = `"./zebu.work"`, const char ∗ *designFile* = `0`, const char ∗ *processName* = `"default_process"`) throw (std::exception) `[static]`

absolete

See also:

Board::restoreHardwareState

6.4.3.50 Board∗ restoreHardwareState (const char ∗ *filename*, const char ∗ *zebuWorkPath* = `"./zebu.work"`, const char ∗ *designFile* = `0`, const char ∗ *processName* = `"default_process"`) throw (std::exception) `[static]`

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 ::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
not supported in zTide environment

Parameters:

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

processName default ”default_process”

designFile default ”designFeatures”

zebuWorkPath default ”./zebu.work”

Returns:

Board∗

Return values:

handler on Board. NULL if open failed.

```
Board *zebuBoard = Board::restoreHardwareState("saveDB.snr");
if(zebuBoard == NULL) {
    cerr << "Cannot restore Board" << endl;
    exit(1);
}
```

See also:

Board::saveHardwareState
Board::init
Board::close
Board::check

6.4.3.51 unsigned int restoreLogicState (const char * *filename*, const Filter * *filter* = 0, const char * *initMemFile* = 0, const unsigned int *severity* = 0) throw (std::exception)

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

Note:

the state must have been saved under the form of a logic state by means of the function ::saveLogicState. You can convert a hardware state into a logic state by means of the libary libZebuRestore.
not supported in zTide environment

Parameters:

filename name of the file of the state to restore

filter allow to filter the types of components of to load: internal signals, driver signals, internal and external memories, clocks ...

initMemFile default NULL. File name. This file give list of memories to initialize with memory file corresponding

Returns:

unsigned int

Parameters:

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

Return values:

0 if successfull.

```
Board *zebuBoard = Board::open();
if(zebuBoard == NULL) {
    cerr << "Cannot open Board" << endl;
    exit(1);
}
```

```
if(zebuBoard->init() != 0)  {
    cerr << "Cannot initialize Board" << endl;
    exit(1);
}

...

if(zebuBoard->saveLogicState("zebu.logic.state") != 0)  {
    cerr << "Cannot save logic state" << endl;
    exit(1);

...

if(zebuBoard->restoreLogicState("zebu.logic.state") != 0)  {
    cerr << "Cannot restore logic state" << endl;
    exit(1);
}
```

See also:
Board::saveLogicState
Board::init
Board::close
Board::check

6.4.3.52 void run () throw (std::exception)

run a cycle for each driver which registered a callback

Note:
not supported in zTide environment

See also:
Board::registerDriver

6.4.3.53 void save (const char * *filename*) throw (std::exception)

absolete

See also:
Board::saveHardwareState

6.4.3.54 void saveHardwareState (const char * *filename*) throw (std::exception)

Board::saveHardwareState.

6.4.3.55 void saveHardwareState (const char * *filename*, const Filter * *filter*) throw (std::exception)

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

filter allow to filter the types of components to save: internal signals, driver signals, internal and external memories, clocks ...

6.4.3.56 void saveLogicState (const char * *filename*, const Filter * *filter* = 0) throw (std::exception)

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

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

filter allow to filter the types of components to save: internal signals, driver signals, internal and external memories, clocks ...

6.4.3.57 void selectMemoriesToRandomize (const char * *memoryList* = 0, const int *invert* = 0) throw (std::exception)

select memories to randomize

Parameters:
memoryList filename that specifies the list of memories to randomize or not to 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 selected

invert if 0 set specified memories else set non specidied memories

6.4.3.58 void selectObjectsToRandomize (const Filter * *filter*) throw (std::exception)

select signals and memories to randomize

Parameters:

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

6.4.3.59 void selectSignalsToRandomize (const char * *signalList* = 0, const int *invert* = 0) throw (std::exception)

select signals to randomize

Parameters:

signalList filename that specifies the list of signals to randomize or not to 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 selected

invert if 0 set specified signals else set non specidied signals

6.4.3.60 bool serviceLoop (ServiceLoopHandler *g*, void * *context*) throw (std::exception)

Board::serviceLoop.

6.4.3.61 bool serviceLoop (ServiceLoopHandler *g*, void * *context*, const unsigned int *portGroupNumber*) throw (std::exception)

Board::serviceLoop.

6.4.3.62 bool serviceLoop (ServiceLoopHandler *g*, void * *context*, const long long unsigned int *portGroupNumber*) throw (std::exception)

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. 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:

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:

true if some messages arrived from the hardware side or new messages can be sent to the hardware side since the last call to serviceLoop()

See also:

Port::isPossibleToReceive
Port::isPossibleToSend.
TxPortQueue
RXPortQueue

6.4.3.63 void serviceLoop () throw (std::exception)

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 proxies.

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

See also:

Port::isPossibleToReceive
Port::isPossibleToSend.

6.4.3.64 bool setMsgVerboseMode (bool *verbose*) throw (std::exception)

Set message verbose mode.

Parameters:

verbose true (default) or false

6.4.3.65 unsigned int waitClockSystemEnable () throw (std::exception)

wait until clock system is enabled

Note:

not supported in zTide environment

See also:
Board::getClock
Board::isClockSystemEnabled

6.4.3.66 void writeRegisters () throw (std::exception)

force register write

Note:
not supported in zTide environment

6.5 Board::DriverInfoIterator Class Reference

Collaboration diagram for Board::DriverInfoIterator:

6.5.1 Detailed Description

Implement public iterator on driver information.

Public Member Functions

- DriverInfoIterator () throw (std::exception)
  constructor.
- ~DriverInfoIterator () throw (std::exception)
  destructor.
- void initialize (const Board *board) throw (std::exception)
  initialize the iterator.
- void goToFirst () throw (std::exception)
  move iterator to first driver.
- void goToNext () throw (std::exception)
  move iterator to next driver.
- bool isAtEnd () const throw (std::exception)
  test if iterator passed last driver.
- const char * getModelName () const throw (std::exception)
  return the model name of the driver
- const char * getInstanceName () const throw (std::exception)
  return the instance name of the driver
- const char * getProcessName () const throw (std::exception)
  return the name of the process from which the driver is accesssible
- ZEBU_Driver_Type getType () const throw (std::exception)
  return the type of the driver
- PortInfoIterator * getPortInfoIterator () const throw (std::exception)
  return an iterator on port information of the driver

6.5.2 Constructor & Destructor Documentation

6.5.2.1 DriverInfoIterator () throw (std::exception)

constructor.

6.5.2.2 ~DriverInfoIterator () throw (std::exception)

destructor.

6.5.3 Member Function Documentation

6.5.3.1 const char* getInstanceName () const throw (std::exception)

return the instance name of the driver

6.5.3.2 const char* getModelName () const throw (std::exception)

return the model name of the driver

6.5.3.3 PortInfoIterator* getPortInfoIterator () const throw (std::exception)

return an iterator on port information of the driver

6.5.3.4 const char* getProcessName () const throw (std::exception)

return the name of the process from which the driver is accesssible

6.5.3.5 ZEBU_Driver_Type getType () const throw (std::exception)

return the type of the driver

See also:
ZEBU_ROOT/include/Types.h

6.5.3.6 void goToFirst () throw (std::exception)

move iterator to first driver.

6.5.3.7 void goToNext () throw (std::exception)

move iterator to next driver.

6.5.3.8 void initialize (const Board * *board*) throw (std::exception)

initialize the iterator.

Parameters:

board handler on Board.

```
Board::DriverInfoIterator driverInfoIterator;
driverInfoIterator.initialize(board);
for(driverInfoIterator.goToFirst(); !driverInfoIterator.isAtEnd(); driverInfoIterator
    printf("Driver instance name = %s\n", driverInfoIterator.getInstanceName());
}
```

See also:

Board::open

6.5.3.9 bool isAtEnd () const throw (std::exception)

test if iterator passed last driver.

Return values:

true if at end.

6.6 Board::MemoryIterator Class Reference

Collaboration diagram for Board::MemoryIterator:

6.6.1 Detailed Description

Implement public iterator on memories.

Public Member Functions

- MemoryIterator () throw (std::exception)

  constructor.

- ~MemoryIterator () throw (std::exception)

  destructor.

- void initialize (const Board *board) throw (std::exception)

  initialize the iterator.

- void goToFirst () throw (std::exception)

  move iterator to first memory.

- void goToNext () throw (std::exception)

  move iterator to next memory.

- bool isAtEnd () const throw (std::exception)

  test if iterator passed last memory.

- const Memory & getMemory () const throw (std::exception)

  return the current memory. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from memory name by means of Board::getMemory to keep a handler on the memory

6.6.2 Constructor & Destructor Documentation

6.6.2.1 MemoryIterator () throw (std::exception)

constructor.

6.6.2.2 ~MemoryIterator () throw (std::exception)

destructor.

6.6.3 Member Function Documentation

6.6.3.1 const Memory& getMemory () const throw (std::exception)

return the current memory. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from memory name by means of Board::getMemory to keep a handler on the memory

6.6.3.2 void goToFirst () throw (std::exception)

move iterator to first memory.

6.6.3.3 void goToNext () throw (std::exception)

move iterator to next memory.

6.6.3.4 void initialize (const Board * *board*) throw (std::exception)

initialize the iterator.

Parameters:

board handler on Board.

```
Board::MemoryIterator memoryIterator;
memoryIterator.initialize(board);
for(memoryIterator.goToFirst(); !memoryIterator.isAtEnd(); memoryIterator.goToNext())
    printf("Memory name = %s\n", memoryIterator.getMemory().fullname());
}
```

See also:

Board::open

6.6.3.5 bool isAtEnd () const throw (std::exception)

test if iterator passed last memory.

Return values:

true if at end.

6.7 Board::PortInfoIterator Class Reference

6.7.1 Detailed Description

Implement public iterator on driver information.

Public Member Functions

- void goToFirst () throw (std::exception)
  move iterator to first memory.
- void goToNext () throw (std::exception)
  move iterator to next memory.
- bool isAtEnd () const throw (std::exception)
  test if iterator passed last memory.
- const char ∗ getPortName () const throw (std::exception)
  return the instance name of the port
- unsigned int getMessageSize () const throw (std::exception)
  return the message size in number of 32-bit words of the port
- ZEBU_Port_Direction getDirection () const throw (std::exception)
  Return the direction of the port.
- bool isConnected () const throw (std::exception)
  test if the port is connected

Protected Member Functions

- PortInfoIterator (const DriverInfoAbstract ∗driverInfo) throw (std::exception)
- ∼PortInfoIterator () throw (std::exception)

6.7.2 Constructor & Destructor Documentation

6.7.2.1 PortInfoIterator (const DriverInfoAbstract ∗ *driverInfo*) throw (std::exception) `[protected]`

private

6.7.2.2 ~PortInfoIterator () throw (std::exception) [protected]

private

6.7.3 Member Function Documentation

6.7.3.1 ZEBU_Port_Direction getDirection () const throw (std::exception)

Return the direction of the port.

See also:
ZEBU_ROOT/include/Types.h

6.7.3.2 unsigned int getMessageSize () const throw (std::exception)

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

6.7.3.3 const char* getPortName () const throw (std::exception)

return the instance name of the port

6.7.3.4 void goToFirst () throw (std::exception)

move iterator to first memory.

6.7.3.5 void goToNext () throw (std::exception)

move iterator to next memory.

6.7.3.6 bool isAtEnd () const throw (std::exception)

test if iterator passed last memory.

Return values:
true if at end.

6.7.3.7 bool isConnected () const throw (std::exception)

test if the port is connected

Return values:

true if the port is connected

6.8 Board::SignalIterator Class Reference

Collaboration diagram for Board::SignalIterator:

6.8.1 Detailed Description

Implement public iterator on internal signals.

Public Member Functions

- SignalIterator () throw (std::exception)

  constructor.

- ~SignalIterator () throw (std::exception)

  destructor.

- void initialize (const Board *board, const bool autoDeselect) throw (std::exception)

  initialize the iterator.

- void initialize (const Board *board) throw (std::exception)

  Board::initialize.

- void goToFirst () throw (std::exception)

  move iterator to first signal.

- void goToNext () throw (std::exception)

  move iterator to next signal.

- bool isAtEnd () const throw (std::exception)

  test if iterator passed last signal.

- const Signal & getSignal () const throw (std::exception)

  return the current signal. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from signal name by means of Board::getSignal to keep a handler on the signal

6.8.2 Constructor & Destructor Documentation

6.8.2.1 SignalIterator () throw (std::exception)

constructor.

6.8.2.2 ~SignalIterator () throw (std::exception)

destructor.

6.8.3 Member Function Documentation

6.8.3.1 const Signal& getSignal () const throw (std::exception)

return the current signal. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from signal name by means of Board::getSignal to keep a handler on the signal

6.8.3.2 void goToFirst () throw (std::exception)

move iterator to first signal.

6.8.3.3 void goToNext () throw (std::exception)

move iterator to next signal.

6.8.3.4 void initialize (const Board * *board*) throw (std::exception)

Board::initialize.

6.8.3.5 void initialize (const Board * *board*, const bool *autoDeselect*) throw (std::exception)

initialize the iterator.

Parameters:

board handler on Board.

autoDeselect specify if signal must be automatically deselected.

```
Board::SignalIterator signalIterator;
signalIterator.initialize(board);
for(signalIterator.goToFirst(); !signalIterator.isAtEnd(); signalIterator.
```

```
        printf("Signal name = %s\n", signalIterator.getSignal().fullname());
    }
```

Note:
to optimize access to signal value, you should get all signals that you need before to access to the value of the first signal.

See also:
Board::open

6.8.3.6 bool isAtEnd () const throw (std::exception)

test if iterator passed last signal.

Return values:
true if at end.

6.9 CCall Class Reference

6.9.1 Detailed Description

Allow controlling C_CALL or function calls.

Note:
Not supported in zTide environment.

Static Public Member Functions

- void SelectSamplingClockGroup (Board *board, const char *clockGroupName=NULL) throw (std::exception)

  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.

- void SelectSamplingClocks (Board *board, const char *clockExpression=NULL) throw (std::exception)

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

- void EnableSynchronization (Board *board) throw (std::exception)

  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.

- void DisableSynchronization (Board *board) throw (std::exception)

  disables the synchronization of function calls.

- void SetOnEvent (Board *board) throw (std::exception)

  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.

- void UnsetOnEvent (Board *board) throw (std::exception)

  deselect function calls on value change mode.

- void LoadDynamicLibrary (Board ∗board, const char ∗fullname) throw (std::exception)

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

- void Start (Board ∗board, const char ∗scope=NULL, const char ∗importName=NULL, const int callNumber=-1) throw (std::exception)

  enables a set of function calls.

- void Stop (Board ∗board, const char ∗scope=NULL, const char ∗importName=NULL, const int callNumber=-1) throw (std::exception)

  disables a set of function calls.

- void Start (const char ∗scopeExpression, Board ∗board, const bool invert=false, const bool ignoreCase=false, const char hierarchicalSeparator= '.', const char ∗importName=NULL, const int callNumber=-1) throw (std::exception)

  enables a set of function calls specified by a regular expession

- void Stop (const char ∗scopeExpression, Board ∗board, const bool invert=false, const bool ignoreCase=false, const char hierarchicalSeparator= '.', const char ∗importName=NULL, const int callNumber=-1) throw (std::exception)

  disables a set of function calls specified by a regular expession

- void Flush (Board ∗board) throw (std::exception)

  flushed the set of enabled function calls

6.9.2 Member Function Documentation

6.9.2.1 void DisableSynchronization (Board ∗ *board*) throw (std::exception) `[static]`

disables the synchronization of function calls.

Parameters:

board C++ handler on Board

Note:

this must be called before any function call start.

See also:

EnableSynchronization

6.9.2.2 void EnableSynchronization (Board * *board*) throw (std::exception) [static]

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:
DisableSynchronization

6.9.2.3 void Flush (Board * *board*) throw (std::exception) [static]

flushed the set of enabled function calls

Parameters:
board C++ handler on Board

6.9.2.4 void LoadDynamicLibrary (Board * *board*, const char * *fullname*) throw (std::exception) [static]

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 C++ handler on 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.

6.9.2.5 void SelectSamplingClockGroup (Board * *board*, const char * *clockGroupName* = NULL) throw (std::exception) [static]

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 C++ handler on 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.

6.9.2.6 void SelectSamplingClocks (Board * *board*, const char * *clockExpression* = NULL) throw (std::exception) [static]

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

Parameters:

board C++ handler on 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.

6.9.2.7 void SetOnEvent (Board * *board*) throw (std::exception) [static]

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 C++ handler on 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:
UnsetOnEvent

6.9.2.8 void Start (const char * *scopeExpression*, Board * *board*, const bool *invert* = `false`, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`, const char * *importName* = `NULL`, const int *callNumber* = `-1`) throw (std::exception) `[static]`

enables a set of function calls specified by a regular expession

Note:
all function calls are disabled by default.

Parameters:

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

board C++ handler on Board

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:
Stop

6.9.2.9 void Start (Board * *board*, const char * *scope* = `NULL`, const char * *importName* = `NULL`, const int *callNumber* = `-1`) throw (std::exception) `[static]`

enables a set of function calls.

Note:
all function calls are disabled by default.

Parameters:
board C++ handler on 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:
Stop

6.9.2.10 void Stop (const char * *scopeExpression*, Board * *board*, const bool *invert* = `false`, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`, const char * *importName* = `NULL`, const int *callNumber* = `-1`) throw (std::exception) `[static]`

disables a set of function calls specified by a regular expession

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

board C++ handler on Board

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:
Start

6.9.2.11 void Stop (Board * *board*, const char * *scope* = NULL, const char * *importName* = NULL, const int *callNumber* = -1) throw (std::exception) [static]

disables a set of function calls.

Parameters:

board C++ handler on 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:

Start

6.9.2.12 void UnsetOnEvent (Board * *board*) throw (std::exception) [static]

deselect function calls on value change mode.

Parameters:

board C++ handler on Board

See also:

SetOnEvent

6.10 CDriver Class Reference

Inheritance diagram for CDriver:Collaboration diagram for CDriver:

6.10.1 Detailed Description

Implement ZeBu driver for simple C cosimulation.

Public Member Functions

- virtual unsigned int run (unsigned int numCycles, bool block) const __attribute__((deprecated)) throw (std::exception)

  obsolete

- unsigned int wait (unsigned int triggers, unsigned int timeout=0xffffffff) const throw (std::exception)

  wait for a trigger event or timeout while running the clock

- void dumpfile (const char *filename, int compression=0) throw (std::exception)

  specify the name of the waveform file for the driver signals

- void dumpvars (Signal *signal=NULL) throw (std::exception)

  select driver signals to dump

- void dumpon () throw (std::exception)

  resume the dump. Switch driver signals waveform dump on

- void dumpoff () throw (std::exception)

  suspend the dump

- void dumpclosefile () throw (std::exception)

  close the waveform file open from CDriver::dumpfile

- void closeDumpfile () throw (std::exception)

  obsolete

- virtual unsigned int run (unsigned int numCycles) const throw (std::exception)

  run a number of cycles

- unsigned int connect () throw (std::exception)

connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char ∗ name () const throw (std::exception)

  get the driver's name

- virtual void update () const throw (std::exception)

  update IOs

- void registerCallback (void(∗)(void ∗callback), void ∗user) throw (std::exception)

  register a callback

- Signal ∗ getSignal (const char ∗name) const throw (std::exception)

  get a signal handler

- void dumpvars (const char ∗name) throw (std::exception)

  select driver signals to dump

- virtual void dumpon (char ∗clockName, char ∗edgeName="posedge") throw (std::exception)

  command the start the trace memory

- virtual void storeToFile () throw (std::exception)

  command the download and the dump of the trace memory

- virtual void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual void setPreTriggerRatio (float size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual bool isTraceMemoryDriver () throw (std::exception)

  returns true if the driver is a trace driver

6.10.2 Member Function Documentation

6.10.2.1 void closeDumpfile () throw (std::exception) `[virtual]`

obsolete

Reimplemented from Driver.

6.10.2.2 unsigned int connect () throw (std::exception) [inherited]

connect driver

Returns:
unsigned int

Return values:
0 OK

$>$***0*** KO

See also:
Driver::disconnect

6.10.2.3 void disconnect () throw (std::exception) [inherited]

disconnect driver

See also:
Driver::connect

6.10.2.4 void dumpclosefile () throw (std::exception) [virtual]

close the waveform file open from CDriver::dumpfile

Note:
not supported in zTide environment

Reimplemented from Driver.

6.10.2.5 void dumpfile (const char * *filename*, int *compression* = 0) throw (std::exception) [virtual]

specify the name of the waveform file for the driver signals

Parameters:
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

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

See also:
CDriver::dumpclosefile
CDriver::dumpvars
CDriver::dumpon
CDriver::dumpoff

Reimplemented from Driver.

6.10.2.6 void dumpoff () throw (std::exception) `[virtual]`

suspend the dump

switch driver signals waveform dump off. This is default.

See also:
CDriver::dumpvars
CDriver::dumpon
CDriver::dumpfile

Reimplemented from Driver.

6.10.2.7 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) `[virtual, inherited]`

command the start the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.10.2.8 void dumpon () throw (std::exception) `[virtual]`

resume the dump. Switch driver signals waveform dump on

See also:
CDriver::dumpvars
CDriver::dumpfile
CDriver::dumpoff

Reimplemented from Driver.

6.10.2.9 void dumpvars (const char * *name*) throw (std::exception) [inherited]

select driver signals to dump

Parameters:
name name of the signal to be dumped.

Note:
no signal can be added after first run.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.10.2.10 void dumpvars (Signal * *signal* = NULL) throw (std::exception) [virtual]

select driver signals to dump

Parameters:
signal handler to 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.

See also:
CDriver::dumpfile
CDriver::dumpon
CDriver::dumpoff

Reimplemented from Driver.

6.10.2.11 Signal* getSignal (const char * *name*) const throw (std::exception) [inherited]

get a signal handler

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

6.10.2.12 virtual bool isTraceMemoryDriver () throw (std::exception) [virtual, inherited]

returns true if the driver is a trace driver

Reimplemented in TraceMemory.

6.10.2.13 const char∗ name () const throw (std::exception) [inherited]

get the driver's name

Returns:
const char ∗

Return values:
NULL terminated C string containing driver's name

6.10.2.14 void registerCallback (void(∗)(void ∗callback), void ∗ *user*) throw (std::exception) [inherited]

register a callback

Parameters:
callback callback

user user data

6.10.2.15 virtual unsigned int run (unsigned int *numCycles*) const throw (std::exception) [virtual]

run a number of cycles

Parameters:
numCycles number of cycles

Returns:
int

Return values:
0 OK

>0 KO

Implements Driver.

6.10.2.16 virtual unsigned int run (unsigned int *numCycles*, bool *block*) const throw (std::exception) `[virtual]`

obsolete

Implements Driver.

6.10.2.17 virtual void setPreTriggerRatio (float *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.10.2.18 virtual void setPreTriggerSize (unsigned int *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.10.2.19 virtual void storeToFile () throw (std::exception) `[virtual, inherited]`

command the download and the dump of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.10.2.20 virtual void update () const throw (std::exception) `[virtual, inherited]`

update IOs

equivalent to `run(0)`

Returns:
void

Reimplemented in MckCDriver.

6.10.2.21 unsigned int wait (unsigned int *triggers*, unsigned int *timeout* = `0xffffffff`) const throw (std::exception) `[virtual]`

wait for a trigger event or timeout while running the clock

Parameters:

triggers to wait

- set bit i to 1 to stop on trigger i (on the 16 lsb)

timeout approximative timeout in number of cycles

Returns:
unsigned int

Return values:

0 if a timeout occurs

bit i set to 1 for trigger i

Reimplemented from Driver.

6.11 Clock Class Reference

6.11.1 Detailed Description

public interface class for ZeBu clocks.

Clock objects allow user to program and control design clocks.

See also:
Board::getClock

Public Member Functions

- ~Clock () throw (std::exception)
  destructor
- unsigned int enable (long long unsigned int cycles=0) const throw (std::exception)
  enable the clock
- unsigned int disable () const throw (std::exception)
  disable the clock
- bool isEnabled () const throw (std::exception)
  get the status of the clock (enabled or disabled)
- unsigned int counter (long long unsigned int &count) const throw (std::exception)
  get clock cycle counter value
- unsigned int reset () const throw (std::exception)
  reset clock cycle counter
- const char * name () const throw (std::exception)
  get the Clock name

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ~Clock () throw (std::exception)

destructor

6.11.3 Member Function Documentation

6.11.3.1 unsigned int counter (long long unsigned int & *count*) const throw (std::exception)

get clock cycle counter value

Parameters:
count return clock cycle counter value

Returns:
unsigned int

Return values:
0 if OK

positive if KO

6.11.3.2 unsigned int disable () const throw (std::exception)

disable the clock

Returns:
unsigned int

Return values:
0 if OK

positive if KO

6.11.3.3 unsigned int enable (long long unsigned int *cycles* = `0`) const throw (std::exception)

enable the clock

Parameters:
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(clk->enable()) {
    cerr << "Cannot enable clock for ever" << endl;
}
```

6.11.3.4 bool isEnabled () const throw (std::exception)

get the status of the clock (enabled or disabled)

Returns:
bool

Return values:
true if clock is enabled

false if clock is disabled

6.11.3.5 const char* name () const throw (std::exception)

get the Clock name

Returns:
const char *

Return values:
NULL terminated C string containing clock's name

6.11.3.6 unsigned int reset () const throw (std::exception)

reset clock cycle counter

Returns:
unsigned int

Return values:
0 if OK

positive if KO

6.12 Driver Class Reference

Inheritance diagram for Driver:

6.12.1 Detailed Description

Implement public interface class for ZeBu drivers.

Note:
Not supported in zTide environment.

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 ...

The Driver class is a pure virtual class and hence cannot be instantiated. It is a common interface for CDriver, MckCDriver, Monitor and PatternDriver.

See also:
Board::getDriver

Public Member Functions

- virtual ~Driver () throw (std::exception)

  destructor

- unsigned int connect () throw (std::exception)

  connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char * name () const throw (std::exception)

  get the driver's name

- virtual unsigned int run (unsigned int numCycles, bool block) const __attribute__((deprecated))=0 throw (std::exception)

  obsolete

- virtual unsigned int wait (unsigned int triggers, unsigned int timeOut=0xffffffff) const throw (std::exception)

  wait for a trigger event or timeout while running the clock

- virtual void update () const throw (std::exception)

  update IOs

- void registerCallback (void(∗)(void ∗callback), void ∗user) throw (std::exception)

  register a callback

- Signal ∗ getSignal (const char ∗name) const throw (std::exception)

  get a signal handler

- virtual void dumpfile (const char ∗filename, int compression=0)

  specify the name of a waveform file

- virtual void dumpvars (Signal ∗signal=NULL)

  select driver signals to dump

- void dumpvars (const char ∗name) throw (std::exception)

  select driver signals to dump

- virtual void dumpon ()

  resume the dump

- virtual void dumpoff () throw (std::exception)

  suspend the dump

- virtual void dumpon (char ∗clockName, char ∗edgeName="posedge") throw (std::exception)

  command the start the trace memory

- virtual void dumpclosefile ()

  close the waveform file open from ::dumpfile

- virtual void closeDumpfile ()

  obsolete

- virtual void storeToFile () throw (std::exception)

  command the download and the dump of the trace memory

- virtual void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual void setPreTriggerRatio (float size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual bool isTraceMemoryDriver () throw (std::exception)

  returns true if the driver is a trace driver

- virtual unsigned int run (unsigned int numCycles) const =0 throw (std::exception)

  run a number of cycles

Protected Member Functions

- Driver () throw (std::exception)

  constructor

- Driver (DriverAbstract *driver) throw (std::exception)

  constructor

6.12.2 Constructor & Destructor Documentation

6.12.2.1 virtual ~Driver () throw (std::exception) [virtual]

destructor

6.12.2.2 Driver () throw (std::exception) [protected]

constructor

6.12.2.3 Driver (DriverAbstract * *driver*) throw (std::exception) [protected]

constructor

6.12.3 Member Function Documentation

6.12.3.1 virtual void closeDumpfile () [inline, virtual]

obsolete

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.2 unsigned int connect () throw (std::exception)

connect driver

Returns:
unsigned int

Return values:
0 OK

>***0*** KO

See also:
Driver::disconnect

6.12.3.3 void disconnect () throw (std::exception)

disconnect driver

See also:
Driver::connect

6.12.3.4 virtual void dumpclosefile () `[inline, virtual]`

close the waveform file open from ::dumpfile

Note:
not supported in zTide environment

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.5 virtual void dumpfile (const char * *filename*, int *compression* = `0`) `[virtual]`

specify the name of a waveform file

Parameters:
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

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

See also:
Driver::dumpvars
Driver::dumpon
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.6 virtual void dumpoff () throw (std::exception) `[inline, virtual]`

suspend the dump

switch driver signals waveform dump off. This is default.

See also:
Driver::dumpvars
Driver::dumpon
Driver::dumpfile

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.7 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) `[virtual]`

command the start the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.12.3.8 virtual void dumpon () `[inline, virtual]`

resume the dump

switch driver signals waveform dump on

See also:
Driver::dumpvars
Driver::dumpfile
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.9 void dumpvars (const char * *name*) throw (std::exception)

select driver signals to dump

Parameters:
name name of the signal to be dumped.

Note:
no signal can be added after first run.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.12.3.10 virtual void dumpvars (Signal * *signal* = `NULL`) `[virtual]`

select driver signals to dump

Parameters:
signal handler to 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.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.12.3.11 Signal* getSignal (const char * *name*) const throw (std::exception)

get a signal handler

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

6.12.3.12 virtual bool isTraceMemoryDriver () throw (std::exception) [virtual]

returns true if the driver is a trace driver

Reimplemented in TraceMemory.

6.12.3.13 const char∗ name () const throw (std::exception)

get the driver's name

Returns:
const char ∗

Return values:
NULL terminated C string containing driver's name

6.12.3.14 void registerCallback (void(∗)(void ∗callback), void ∗ *user*) throw (std::exception)

register a callback

Parameters:
callback callback

user user data

6.12.3.15 virtual unsigned int run (unsigned int *numCycles*) const throw (std::exception) [pure virtual]

run a number of cycles

Parameters:
numCycles number of cycles

Returns:
int

Return values:
0 OK

$>$***0*** KO

Implemented in CDriver, Monitor, TraceMemory, MckCDriver, and PatternDriver.

6.12.3.16 virtual unsigned int run (unsigned int *numCycles*, bool *block*) const throw (std::exception) `[pure virtual]`

obsolete

Implemented in CDriver, Monitor, TraceMemory, MckCDriver, and PatternDriver.

6.12.3.17 virtual void setPreTriggerRatio (float *size*) throw (std::exception) `[virtual]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.12.3.18 virtual void setPreTriggerSize (unsigned int *size*) throw (std::exception) `[virtual]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.12.3.19 virtual void storeToFile () throw (std::exception) `[virtual]`

command the download and the dump of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.12.3.20 virtual void update () const throw (std::exception) `[virtual]`

update IOs

equivalent to `run(0)`

Returns:
void

Reimplemented in MckCDriver.

6.12.3.21 virtual unsigned int wait (unsigned int *triggers*, unsigned int *timeOut* = `0xffffffff`) const throw (std::exception) `[virtual]`

wait for a trigger event or timeout while running the clock

Parameters:

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 stopping.

Returns:

unsigned int

Return values:

0 if a timeout occurs

bit i set to 1 for trigger i

Reimplemented in CDriver, and MckCDriver.

6.13 Driver::SignalIterator Class Reference

Collaboration diagram for Driver::SignalIterator:

6.13.1 Detailed Description

Implement public iterator on driver signals.

Public Member Functions

- SignalIterator () throw (std::exception)

  constructor.

- ~SignalIterator () throw (std::exception)

  destructor.

- void initialize (const Driver *driver) throw (std::exception)

  initialize the iterator.

- void goToFirst () throw (std::exception)

  move iterator to first signal.

- void goToNext () throw (std::exception)

  move iterator to next signal.

- bool isAtEnd () const throw (std::exception)

  test if iterator passed last signal.

- const Signal & getSignal () const throw (std::exception)

  return the current signal. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from signal name by means of Board::getSignal to keep a handler on the signal

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SignalIterator () throw (std::exception)

constructor.

6.13.2.2 ~SignalIterator () throw (std::exception)

destructor.

6.13.3 Member Function Documentation

6.13.3.1 const Signal& getSignal () const throw (std::exception)

return the current signal. The returned reference is valid only as the iterator exists, and as long as only constant functions are called for it. Get a pointer from signal name by means of Board::getSignal to keep a handler on the signal

6.13.3.2 void goToFirst () throw (std::exception)

move iterator to first signal.

6.13.3.3 void goToNext () throw (std::exception)

move iterator to next signal.

6.13.3.4 void initialize (const Driver * *driver*) throw (std::exception)

initialize the iterator.

Parameters:

driver handler on Driver.

```
Driver::SignalIterator signalIterator;
signalIterator.initialize(driver);
for(signalIterator.goToFirst(); !signalIterator.isAtEnd(); signalIterator.
    printf("Signal name = %s\n", signalIterator.getSignal().name());
}
```

Note:

to optimize access to signal value, you should get all signals that you need before to access to the value of the first signal.

See also:

Board::open

6.13.3.5 bool isAtEnd () const throw (std::exception)

test if iterator passed last signal.

Return values:

true if at end.

6.14 Events Class Reference

6.14.1 Detailed Description

This class provides methods to register/unregister a user fonction that can be used to handle public events that are fired by ZeBu.

Public Types

- typedef void(∗ **handler_type**)(ZEBU_EventReason reason)

Static Public Member Functions

- void Register (Board ∗board, handler_type handler)

  Register a new global handler for ZEBU public events.

- void Unregister (Board ∗board, handler_type handler)

  Unregister a global callback.

6.14.2 Member Function Documentation

6.14.2.1 void Register (Board ∗ *board*, handler_type *handler*) [static]

Register a new global handler for ZEBU public events.

Parameters:
handler The handler to register.

6.14.2.2 void Unregister (Board ∗ *board*, handler_type *handler*) [static]

Unregister a global callback.

Parameters:
handler The handler to unregister.

6.15 FastHardwareState Class Reference

6.15.1 Detailed Description

Allow to capture fastly the hardware state and the software state of a ZeBu session and then to save it to disk.

Note:
Not supported in zTide environment.

Public Member Functions

- FastHardwareState ()
  constructor
- ~FastHardwareState ()
  destructor
- void initialize (Board *board) throw (std::exception)
  initialize the object
- void initialize (Board *board, const Filter *filter) throw (std::exception)
  initialize the object
- void capture () throw (std::exception)
  capture fastly the hardware state into memory
- void save (const char *filename, bool inParallel=false) throw (std::exception)
  write on disk the hardware state previouly captured
- bool isParallelSaveFinished () const throw (std::exception)
  test if parallel save is finished
- void clean () throw (std::exception)
  clean the hardware state previouly captured. Release used memory.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 FastHardwareState ()

constructor

Example:

FastHardwareState fastHardwareState; fastHardwareState.initialize(zebu); fastHardwareState.capture(); fastHardwareState.save("fastHardwareState"); fastHardwareState.clean();

6.15.2.2 ~FastHardwareState ()

destructor

6.15.3 Member Function Documentation

6.15.3.1 void capture () throw (std::exception)

capture fastly the hardware state into memory

6.15.3.2 void clean () throw (std::exception)

clean the hardware state previouly captured. Release used memory.

6.15.3.3 void initialize (Board * *board*, const Filter * *filter*) throw (std::exception)

initialize the object

Parameters:

board C++ handler on Board

filter allow to filter the types of components to save: internal signals, driver signals, internal and external memories, clocks ...

Note:

To do after board intialization

6.15.3.4 void initialize (Board * *board*) throw (std::exception)

initialize the object

Parameters:

board C++ handler on Board

6.15.3.5 bool isParallelSaveFinished () const throw (std::exception)

test if parallel save is finished

6.15.3.6 void save (const char * *filename*, bool *inParallel* = `false`) throw (std::exception)

write on disk the hardware state previouly captured

Note:

this method 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 z 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.
cannot be called after board closing
does not release memory used by capture

Parameters:

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

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

6.16 Filter Class Reference

6.16.1 Detailed Description

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

See also:
Board::saveLogicState
Board::restoreLogicState

Public Member Functions

- Filter (const unsigned int types=Z_ALL, const int numberOfHierarchicalLevels=-1, const char hierarchicalSeparator= '.', const char *regularExpression=0, const bool invert=false, const bool ignoreCase=false) throw (std::exception)

  constructor

- ~Filter ()

  destructor

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Filter (const unsigned int *types* = `Z_ALL`, const int *numberOfHierarchicalLevels* = `-1`, const char *hierarchicalSeparator* = `'.'`, const char * *regularExpression* = `0`, const bool *invert* = `false`, const bool *ignoreCase* = `false`) throw (std::exception)

constructor

Parameters:

types specify the types of components to enabled

numberOfHierarchicalLevels specify the number of hierarchical levels to enable. If -1 all levels are enabled

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

6.16.2.2 ∼Filter ()

destructor

6.17 FlexibleLocalProbeFile Class Reference

6.17.1 Detailed Description

Allows controlling a group of flexible local groups in ZeBu hardware and dumping their traces to disk.

Note:
Not supported in zTide environment.
Only one FlexibleLocalProbeFile instance can be used per group at the same time.

Public Types

- enum EdgeType { **POSEDGE** = 0, **NEGEDGE** }

  edge type

Public Member Functions

- FlexibleLocalProbeFile ()

  constructor

- ~FlexibleLocalProbeFile ()

  destructor.

- void initialize (Board *board, const int thread=-1) throw (std::exception)

  initialize the object

- void add (const char *groupname) throw (std::exception)

  add a group in the file

- void add (const char *regularExpression, const bool invert, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

  add some groups in the file from a regular expression. The regular expression specifies the names of groups to add in the files

- long long unsigned enable () throw (std::exception)

  enable all groups of the file. The trace of a group is sent by the hardware if its enable pin macro is asserted.

- long long unsigned disable () throw (std::exception)

  disable all groups of the file. No more trace is sent by the hardware even if the enable pins of the groups macro are asserted.

- long long unsigned dumpFile (const char *filename) throw (std::exception)

  dump trace sent by hardware to disk in concurrent thread progressively.

- long long unsigned flushFile () throw (std::exception)

  synchronize the dumped file, complete the last trace cycle already dumped in file.

- long long unsigned closeFile () throw (std::exception)

  dump acquired trace and close the dumped file. Disable the groups of the file temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the groups of the file. After closing the group is ready to dump trace to another file.

Static Public Member Functions

- void SelectSamplingClock (Board *board, const char *clockName, const EdgeType edgeType)

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

- void SelectSamplingClocks (Board *board, const char *clockExpression)

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

6.17.2 Member Enumeration Documentation

6.17.2.1 enum EdgeType

edge type

6.17.3 Constructor & Destructor Documentation

6.17.3.1 FlexibleLocalProbeFile ()

constructor

Example:

Initialize and enable the groups of the file FlexibleLocalProbeFile group; file.initialize(zebu); file.add(<groupname1>); file.add(<groupname2>); [...]

Start dumping to disk, run the test bench and then disable the groups and dumping and so on file.dumpFile(<filename 1>); <run the test bench generating value changes>

time = file.closeFile(); std::cout << ”close file at time=” << std::dec << time << std::endl;

file.dump(<filename 2>); <run the test bench generating value changes> time = file.closeFile(); std::cout << ”close file at time=” << std::dec << time << std::endl;

[...]

6.17.3.2 ~FlexibleLocalProbeFile ()

destructor.

Note:
disable the groups.

See also:
disable
dump

6.17.4 Member Function Documentation

6.17.4.1 void add (const char * *regularExpression*, const bool *invert*, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`) throw (std::exception)

add some groups in the file from a regular expression. The regular expression specifies the names of groups to add in the files

Parameters:

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 G1 L0.G2 T2.G0 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

6.17.4.2 void add (const char * *groupname*) throw (std::exception)

add a group in the file

Parameters:

groupname name of the group or just <RNAME> of the tracer if <RPATH> is undefined

6.17.4.3 long long unsigned closeFile () throw (std::exception)

dump acquired trace and close the dumped file. Disable the groups of the file temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the groups of the file. After closing the group is ready to dump trace to another file.

Returns:

long long unsigned

Return values:

last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing groups.

See also:

dump
abortDump

6.17.4.4 long long unsigned disable () throw (std::exception)

disable all groups of the file. No more trace is sent by the hardware even if the enable pins of the groups macro are asserted.

Returns:

long long unsigned

Return values:

timestamp at which the groups have been disabled. The timestamp is the number of cycles generated for the clock synchronizing groups.

6.17.4.5 long long unsigned dumpFile (const char * *filename*) throw (std::exception)

dump trace sent by hardware to disk in concurrent thread progressively.

Parameters:

filename name of the file in which must be dump values of group'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:

long long unsigned

Return values:

last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing groups.

See also:

flush
stopDump
abortDump

6.17.4.6 long long unsigned enable () throw (std::exception)

enable all groups of the file. The trace of a group is sent by the hardware if its enable pin macro is asserted.

Returns:

long long unsigned

Return values:

timestamp at which the groups have been enabled. The timestamp is the number of cycles generated for the clock synchronizing groups.

6.17.4.7 long long unsigned flushFile () throw (std::exception)

synchronize the dumped file, complete the last trace cycle already dumped in file.

Returns:
long long unsigned

Return values:
last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing groups.

See also:
dump

6.17.4.8 void initialize (Board * *board*, const int *thread* = -1) throw (std::exception)

initialize the object

Parameters:
board C++ handler on Board

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

6.17.4.9 void SelectSamplingClock (Board * *board*, const char * *clockName*, const EdgeType *edgeType*) [static]

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

Parameters:
board C++ handler on Board

clockName name of a controlled clock

edgeType sensitive clock edge

6.17.4.10 void SelectSamplingClocks (Board * *board*, const char * *clockExpression*) [static]

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

Parameters:
board C++ handler on 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

6.18 InteractiveLoopDetector Class Reference

6.18.1 Detailed Description

This class provides several methods to work with the combinationnal 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.

Static Public Member Functions

- void enable (Board ∗board, const char ∗loopPath=0) throw (std::exception)

  Enables an interactive combinational loop detector.

- void disable (Board ∗board, const char ∗loopPath=0) throw (std::exception)

  Disables an interactive combinational loop detector.

- int waitDriver (Board ∗board, Driver ∗driver, unsigned int timeout=0xffffffff) throw (std::exception)

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

- int waitDriver (Board ∗board, Driver ∗driver, unsigned int triggers, unsigned int &fired, unsigned int timeout=0xffffffff) throw (std::exception)

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

- void enableGlobalCallback (Board ∗board, Driver ∗driver) throw (std::exception)

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

- void disableGlobalCallback (Board ∗board, Driver ∗driver) throw (std::exception)

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

6.18.2 Member Function Documentation

6.18.2.1 void disable (Board * *board*, const char * *loopPath* = 0) throw (std::exception) [static]

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.

See also:

ZEBU::InteractiveLoopDetector::enable
ZEBU::InteractiveLoopDetector::waitDriver

6.18.2.2 void disableGlobalCallback (Board * *board*, Driver * *driver*) throw (std::exception) [static]

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.

See also:

ZEBU::InteractiveLoopDetector::enableGlobalCallback
ZEBU::Callback::Register

6.18.2.3 void enable (Board * *board*, const char * *loopPath* = 0) throw (std::exception) [static]

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.

See also:

ZEBU::InteractiveLoopDetector::disable
ZEBU::InteractiveLoopDetector::waitDriver

6.18.2.4 void enableGlobalCallback (Board * *board*, Driver * *driver*) throw (std::exception) [static]

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.

See also:

ZEBU::InteractiveLoopDetector::disableGlobalCallback
ZEBU::Callback::Register

6.18.2.5 int waitDriver (Board * *board*, Driver * *driver*, unsigned int *triggers*, unsigned int & *fired*, unsigned int *timeout* = `0xffffffff`) throw (std::exception) [static]

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).

→ ***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::Iterator

6.18.2.6 int waitDriver (Board * *board*, Driver * *driver*, unsigned int *timeout* = `0xffffffff`) throw (std::exception) `[static]`

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::Iterator

6.19 InteractiveLoopDetector::Iterator Class Reference

6.19.1 Detailed Description

Implements an iterator on oscillating loops.

```
ZEBU::InteractiveLoopDetector::Iterator loopIterator(zebu);
for (loopIterator.goToFirst(); !loopIterator.isAtEnd(); loopIterator.goToNext()) {
    printf("oscillating loop name = %s\n", loopIterator.getName());
}
```

See also:
ZEBU::InteractiveLoopDetector::enable
ZEBU::InteractiveLoopDetector::enableGlobalCallback
ZEBU::InteractiveLoopDetector::waitDriver

Public Member Functions

- Iterator (Board ∗board) throw (std::exception)

  Constructs and initializes a new `Zebu::InteractiveLoopDetector::Iterator` *instance.*

- ∼Iterator () throw (std::exception)

  Destroys a `Zebu::InteractiveLoopDetector::Iterator` *instance.*

- void goToFirst () throw (std::exception)

  Moves iterator to the first oscillating loop.

- void goToNext () throw (std::exception)

  Moves iterator to the next oscillating loop.

- bool isAtEnd () const throw (std::exception)

  Tests if iterator passed last oscillating loop.

- const char ∗ **getName** () const throw (std::exception)

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Iterator (Board ∗ *board*) throw (std::exception)

Constructs and initializes a new `Zebu::InteractiveLoopDetector::Iterator` instance.

Parameters:
board A pointer to a `ZEBU::Board` object.

6.19.2.2 ~Iterator () throw (std::exception)

Destroys a `Zebu::InteractiveLoopDetector::Iterator` instance.

6.19.3 Member Function Documentation

6.19.3.1 void goToFirst () throw (std::exception)

Moves iterator to the first oscillating loop.

See also:
Zebu::InteractiveLoopDetector::Iterator::goToNext
Zebu::InteractiveLoopDetector::Iterator::isAtEnd

6.19.3.2 void goToNext () throw (std::exception)

Moves iterator to the next oscillating loop.

See also:
Zebu::InteractiveLoopDetector::Iterator::goToFirst
Zebu::InteractiveLoopDetector::Iterator::isAtEnd

6.19.3.3 bool isAtEnd () const throw (std::exception)

Tests if iterator passed last oscillating loop.

Returns:
A boolean value indicating if the iterator is at the end.

See also:
Zebu::InteractiveLoopDetector::Iterator::goToFirst
Zebu::InteractiveLoopDetector::Iterator::goToNext

6.20 LocalTraceDumper Class Reference

Inheritance diagram for LocalTraceDumper:Collaboration diagram for LocalTraceDumper:

6.20.1 Detailed Description

Allows controlling a local tracer in ZeBu hardware and dumping its trace to disk.

Note:
Not supported in zTide environment.
Only one LocalTracer instance can be used per tracer at the same time.

Public Types

- enum EdgeType { **POSEDGE** = 0, **NEGEDGE** }

  edge type

Public Member Functions

- LocalTraceDumper ()

  constructor

- ~LocalTraceDumper ()

  destructor.

- void initialize (Board *board, const char *fullname, const int thread=-1) throw (std::exception)

  initialize the object

- void setInNoXDumpMode () throw (std::exception)

  set the tracer in special mode for disabling X dumps. It has to be called after initialize and before dumpFile.

- bool isNoXDumpModeStarted () throw (std::exception)

  get if the tracer in special mode for disabling X dumps.

- long long unsigned dumpFile (const char *filename, const long long int offset, const unsigned int ratio=1) throw (std::exception)

  dump trace sent by hardware to disk in concurrent thread progressively.

- long long unsigned dumpFile (const char *filename) throw (std::exception)

  LocalTraceDumper::dumpFile.

- long long unsigned flushFile () throw (std::exception)

  synchronize the dumped file, complete the last trace cycle already dumped in file.

- long long unsigned closeFile () throw (std::exception)

  dump acquired trace and close the dumped file. Disable tracer temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the tracer. After closing the tracer is ready to dump its trace to another file.

- const char * getPath () const throw (std::exception)

  get the tracer's path

- const char * getName () const throw (std::exception)

  get the tracer's name

- const char * getFullname () const throw (std::exception)

  get the tracer's fullname

- long long unsigned enable () throw (std::exception)

  enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable () throw (std::exception)

  disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled () const throw (std::exception)

  test if the tracer is enabled.

Static Public Member Functions

- void SelectSamplingClock (Board *board, const char *clockName, const EdgeType edgeType)

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

Protected Attributes

- LocalTracerAbstract * _tracerAbstract
  private

6.20.2 Member Enumeration Documentation

6.20.2.1 enum EdgeType [inherited]

edge type

6.20.3 Constructor & Destructor Documentation

6.20.3.1 LocalTraceDumper ()

constructor

Note:
do not enable the tracer

Example:

Initialize and enable the tracer LocalTraceDumper tracer; tracer.initialize(zebu, <fullname>); long long unsigned time = tracer.enable(); std::cout << "enable tracer=" << tracer->getFullname() << " at time=" << std::dec << time << std::endl;

Start dumping to disk, run the test bench and then disable the tracer and dumping and so on tracer.dumpFile(<filename 1>); <run the test bench generating value changes> time = tracer.closeFile(); std::cout << "close file at time=" << std::dec << time << std::endl;

tracer.dumpFile(<filename 2>); <run the test bench generating value changes> time = tracer.closeFile(); std::cout << "close file at time=" << std::dec << time << std::endl;

[...]

6.20.3.2 ~LocalTraceDumper ()

destructor.

Note:
disable the tracer.

See also:
disable
dump

6.20.4 Member Function Documentation

6.20.4.1 long long unsigned closeFile () throw (std::exception)

dump acquired trace and close the dumped file. Disable tracer temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the tracer. After closing the tracer is ready to dump its trace to another file.

Returns:
long long unsigned

Return values:
last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
dump
abortDump

6.20.4.2 long long unsigned disable () throw (std::exception) `[inherited]`

disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.20.4.3 long long unsigned dumpFile (const char * *filename*) throw (std::exception)

LocalTraceDumper::dumpFile.

6.20.4.4 long long unsigned dumpFile (const char * *filename*, const long long int *offset*, const unsigned int *ratio* = 1) throw (std::exception)

dump trace sent by hardware to disk in concurrent thread progressively.

Parameters:

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>]

offset positive of negative offset to apply to the timestamp. Default is 0

ratio ratio in number of sample clock cycles to apply to the sample clock cycle to obtain the timestamp. Default is 1

Returns:

long long unsigned

Return values:

last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:

flush
closeFile
abortDump

6.20.4.5 long long unsigned enable () throw (std::exception) [inherited]

enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

Returns:

long long unsigned

Return values:

timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.20.4.6 long long unsigned flushFile () throw (std::exception)

synchronize the dumped file, complete the last trace cycle already dumped in file.

Returns:
long long unsigned

Return values:
last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
dump

6.20.4.7 const char* getFullname () const throw (std::exception) [inherited]

get the tracer's fullname

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

6.20.4.8 const char* getName () const throw (std::exception) [inherited]

get the tracer's name

Returns:
const char *

Return values:
NULL terminated C string containing tracer's name

6.20.4.9 const char* getPath () const throw (std::exception) [inherited]

get the tracer's path

Returns:
const char *

Return values:
NULL terminated C string containing tracer's path

6.20.4.10 void initialize (Board * *board*, const char * *fullname*, const int *thread* = -1) throw (std::exception)

initialize the object

Parameters:

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.

6.20.4.11 bool isEnabled () const throw (std::exception) [inherited]

test if the tracer is enabled.

Note:

do not test if the enable pin of the tracer macro is asserted.

Returns:

bool

Return values:

true if tracer is enabled

6.20.4.12 bool isNoXDumpModeStarted () throw (std::exception)

get if the tracer in special mode for disabling X dumps.

Return values:

true if special mode for disabling X dumps is enabled

6.20.4.13 void SelectSamplingClock (Board * *board*, const char * *clockName*, const EdgeType *edgeType*) [static, inherited]

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

6.20.4.14 void setInNoXDumpMode () throw (std::exception)

set the tracer in special mode for disabling X dumps. It has to be called after initialize and before dumpFile.

6.20.5 Member Data Documentation

6.20.5.1 LocalTracerAbstract* _tracerAbstract [protected, inherited]

private

6.21 LocalTraceDumperGroup Class Reference

Inheritance diagram for LocalTraceDumperGroup:Collaboration diagram for LocalTraceDumperGroup:

6.21.1 Detailed Description

Allows controlling a group of local tracers in ZeBu hardware and dumping their traces to disk.

Note:
Not supported in zTide environment.
Only one LocalTracer or LocalTracerGroup instance can be used per tracer at the same time.

Public Member Functions

- LocalTraceDumperGroup ()

  constructor

- ~LocalTraceDumperGroup ()

  destructor.

- void initialize (Board *board, const int thread=-1) throw (std::exception)

  initialize the object

- void setInNoXDumpMode () throw (std::exception)

  set the tracer in special mode for disabling X dumps It has to be called after initialize and before dumpFile/

- bool isNoXDumpModeStarted () throw (std::exception)

  get if the tracer in special mode for disabling X dumps.

- long long unsigned dumpFile (const char *filename, const long long int offset, const unsigned int ratio=1) throw (std::exception)

  dump trace sent by hardware to disk in concurrent thread progressively.

- long long unsigned dumpFile (const char *filename) throw (std::exception)

  LocalTraceDumperGroup::dumpFile.

- long long unsigned flushFile () throw (std::exception)

  synchronize the dumped file, complete the last trace cycle already dumped in file.

- long long unsigned closeFile () throw (std::exception)

  dump acquired trace and close the dumped file. Disable the tracers of the group temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the tracers of the group. After closing the group is ready to dump trace to another file.

- int add (const char ∗fullname) throw (std::exception)

  add a tracer in the group

- void add (const char ∗regularExpression, const bool invert, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

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

- int getNumberOfTracers () const throw (std::exception)

  get the number of tracers added in the group

- int getIdentifier (const char ∗fullname) const throw (std::exception)

  get the identfier of a tracer added in the group

- const char ∗ getPath (const int tracerIdentifier) const throw (std::exception)

  get the path of a tracer

- const char ∗ getName (const int tracerIdentifier) const throw (std::exception)

  get the name of a tracer

- const char ∗ getFullname (const int tracerIdentifier) const throw (std::exception)

  get the fullname of a tracer

- long long unsigned enable (const int tracerIdentifier) throw (std::exception)

  enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable (const int tracerIdentifier) throw (std::exception)

  disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled (const int tracerIdentifier) const throw (std::exception)

  test if a tracer of the group is enabled.

- long long unsigned enableAll () throw (std::exception)

  enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

- long long unsigned disableAll () throw (std::exception)

  disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Protected Attributes

- LocalTracerGroupAbstract * _tracerGroupAbstract

  private

6.21.2 Constructor & Destructor Documentation

6.21.2.1 LocalTraceDumperGroup ()

constructor

Example:

Initialize and enable the tracers of the group LocalTraceDumperGroup group; group.initialize(zebu); group.add(<fullname1>); group.add(<fullname2>); [...]

Start dumping to disk, run the test bench and then disable the tracers of the group and dumping and so on group.dumpFile(<filename 1>); <run the test bench generating value changes> time = group.closeFile(); std::cout << "close file at time=" << std::dec << time << std::endl;

group.dump(<filename 2>); <run the test bench generating value changes> time = group.closeFile(); std::cout << "close file at time=" << std::dec << time << std::endl;

[...]

6.21.2.2 ~LocalTraceDumperGroup ()

destructor.

Note:
disable the tracers of the group.

See also:
disable

dump

6.21.3 Member Function Documentation

6.21.3.1 void add (const char * *regularExpression*, const bool *invert*, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`) throw (std::exception) `[inherited]`

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

Parameters:

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

6.21.3.2 int add (const char * *fullname*) throw (std::exception) `[inherited]`

add a tracer in the group

Parameters:

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:

int

Return values:

identifier of the tracer in the group

6.21.3.3 long long unsigned closeFile () throw (std::exception)

dump acquired trace and close the dumped file. Disable the tracers of the group temporary, wait for reception of all trace cycles acquired by hardware, dump all to file, close file and re-enable the tracers of the group. After closing the group is ready to dump trace to another file.

Returns:
long long unsigned

Return values:
last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
dump
abortDump

6.21.3.4 long long unsigned disable (const int *tracerIdentifier*) throw (std::exception) `[inherited]`

disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.21.3.5 long long unsigned disableAll () throw (std::exception) [inherited]

disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.21.3.6 long long unsigned dumpFile (const char * *filename*) throw (std::exception)

LocalTraceDumperGroup::dumpFile.

6.21.3.7 long long unsigned dumpFile (const char * *filename*, const long long int *offset*, const unsigned int *ratio* = 1) throw (std::exception)

dump trace sent by hardware to disk in concurrent thread progressively.

Parameters:
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>]

offset positive of negative offset to apply to the timestamp. Default is 0.

ratio ratio in number of sample clock cycles to apply to the sample clock cycle to obtain the timestamp. Default is 1.

Returns:
long long unsigned

Return values:

last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:

flush
stopDump
abortDump

6.21.3.8 long long unsigned enable (const int *tracerIdentifier*) throw (std::exception) `[inherited]`

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

Parameters:

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

Returns:

long long unsigned

Return values:

timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:

add
getIdentifier

6.21.3.9 long long unsigned enableAll () throw (std::exception) `[inherited]`

enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

Returns:

long long unsigned

Return values:

timestamp at which the tracers have been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.21.3.10 long long unsigned flushFile () throw (std::exception)

synchronize the dumped file, complete the last trace cycle already dumped in file.

Returns:
long long unsigned

Return values:
last timestamp written to disk. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
dump

6.21.3.11 const char* getFullname (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the fullname of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

See also:
add
getIdentifier

6.21.3.12 int getIdentifier (const char * *fullname*) const throw (std::exception) [inherited]

get the identfier of a tracer added in the group

Parameters:
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:
int

See also:
add

6.21.3.13 const char* getName (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the name of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's name

See also:
add
getIdentifier

6.21.3.14 int getNumberOfTracers () const throw (std::exception) [inherited]

get the number of tracers added in the group

Return values:
int

See also:
add

6.21.3.15 const char* getPath (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the path of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's path

See also:
add
getIdentifier

6.21.3.16 void initialize (Board * *board*, const int *thread* = -1) throw (std::exception)

initialize the object

Parameters:
board C++ handler on Board

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

6.21.3.17 bool isEnabled (const int *tracerIdentifier*) const throw (std::exception) [inherited]

test if a tracer of the group is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
bool

Return values:
true if tracer is enabled

6.21.3.18 bool isNoXDumpModeStarted () throw (std::exception)

get if the tracer in special mode for disabling X dumps.

Return values:
true if special mode for disabling X dumps is enabled

6.21.3.19 void setInNoXDumpMode () throw (std::exception)

set the tracer in special mode for disabling X dumps It has to be called after initialize and before dumpFile/

6.21.4 Member Data Documentation

6.21.4.1 LocalTracerGroupAbstract* _tracerGroupAbstract [protected, inherited]

private

6.22 LocalTraceImporter Class Reference

Inheritance diagram for LocalTraceImporter:Collaboration diagram for LocalTraceImporter:

6.22.1 Detailed Description

Allows controlling a local tracer in ZeBu hardware and importing tracer's signal values in a DPI function.

Note:
Not supported in zTide environment.
Only one LocalTracer instance can be used per tracer at the same time.

Public Types

- enum EdgeType { **POSEDGE** = 0, **NEGEDGE** }

  edge type

Public Member Functions

- const char ∗ getPath () const throw (std::exception)

  get the tracer's path

- const char ∗ getName () const throw (std::exception)

  get the tracer's name

- const char ∗ getFullname () const throw (std::exception)

  get the tracer's fullname

- long long unsigned enable () throw (std::exception)

  enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable () throw (std::exception)

  disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled () const throw (std::exception)

  test if the tracer is enabled.

Static Public Member Functions

- void SelectSamplingClock (Board *board, const char *clockName, const EdgeType edgeType)

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

Public Attributes

- long long unsigned **time** = tracer.enable()

Protected Attributes

- LocalTracerAbstract * _tracerAbstract

  private

6.22.2 Member Enumeration Documentation

6.22.2.1 enum EdgeType [inherited]

edge type

6.22.3 Member Function Documentation

6.22.3.1 long long unsigned disable () throw (std::exception) [inherited]

disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.22.3.2 long long unsigned enable () throw (std::exception) [inherited]

enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.22.3.3 const char* getFullname () const throw (std::exception) [inherited]

get the tracer's fullname

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

6.22.3.4 const char* getName () const throw (std::exception) [inherited]

get the tracer's name

Returns:
const char *

Return values:
NULL terminated C string containing tracer's name

6.22.3.5 const char* getPath () const throw (std::exception) [inherited]

get the tracer's path

Returns:
const char *

Return values:
NULL terminated C string containing tracer's path

6.22.3.6 bool isEnabled () const throw (std::exception) [inherited]

test if the tracer is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Returns:
bool

Return values:
true if tracer is enabled

6.22.3.7 void SelectSamplingClock (Board * *board*, const char * *clockName*, const EdgeType *edgeType*) [static, inherited]

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

6.22.4 Member Data Documentation

6.22.4.1 LocalTracerAbstract* _tracerAbstract [protected, inherited]

private

6.23 LocalTraceImporterGroup Class Reference

Inheritance diagram for LocalTraceImporterGroup:Collaboration diagram for LocalTraceImporterGroup:

6.23.1 Detailed Description

Allows controlling a group of local tracers in ZeBu hardware and importing signal values of all tracers of the group in a DPI function.

Note:
Not supported in zTide environment.
Only one LocalTracer instance can be used per tracer at the same time.

Public Member Functions

- int add (const char ∗fullname) throw (std::exception)

  add a tracer in the group

- void add (const char ∗regularExpression, const bool invert, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

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

- int getNumberOfTracers () const throw (std::exception)

  get the number of tracers added in the group

- int getIdentifier (const char ∗fullname) const throw (std::exception)

  get the identfier of a tracer added in the group

- const char ∗ getPath (const int tracerIdentifier) const throw (std::exception)

  get the path of a tracer

- const char ∗ getName (const int tracerIdentifier) const throw (std::exception)

  get the name of a tracer

- const char ∗ getFullname (const int tracerIdentifier) const throw (std::exception)

  get the fullname of a tracer

- long long unsigned enable (const int tracerIdentifier) throw (std::exception)

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable (const int tracerIdentifier) throw (std::exception)

  disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled (const int tracerIdentifier) const throw (std::exception)

  test if a tracer of the group is enabled.

- long long unsigned enableAll () throw (std::exception)

  enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

- long long unsigned disableAll () throw (std::exception)

  disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Public Attributes

- int **tracerIdentifier1** = group.add(<fullname1>)
- int **tracerIdentifier2** = group.add(<fullname2>)
- long long unsigned **time** = group.enableAll()

Protected Attributes

- LocalTracerGroupAbstract * _tracerGroupAbstract

  private

6.23.2 Member Function Documentation

6.23.2.1 void add (const char * *regularExpression*, const bool *invert*, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`) throw (std::exception) `[inherited]`

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

Parameters:

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

6.23.2.2 int add (const char * *fullname*) throw (std::exception) `[inherited]`

add a tracer in the group

Parameters:

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:

int

Return values:

identifier of the tracer in the group

6.23.2.3 long long unsigned disable (const int *tracerIdentifier*) throw (std::exception) `[inherited]`

disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Parameters:

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

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.23.2.4 long long unsigned disableAll () throw (std::exception) `[inherited]`

disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.23.2.5 long long unsigned enable (const int *tracerIdentifier*) throw (std::exception) `[inherited]`

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.23.2.6 long long unsigned enableAll () throw (std::exception) [inherited]

enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.23.2.7 const char* getFullname (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the fullname of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

See also:
add
getIdentifier

6.23.2.8 int getIdentifier (const char * *fullname*) const throw (std::exception) [inherited]

get the identfier of a tracer added in the group

Parameters:
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:
int

See also:
add

6.23.2.9 const char* getName (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the name of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's name

See also:
add
getIdentifier

6.23.2.10 int getNumberOfTracers () const throw (std::exception) [inherited]

get the number of tracers added in the group

Return values:
int

See also:
add

6.23.2.11 const char* getPath (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the path of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's path

See also:
add
getIdentifier

6.23.2.12 bool isEnabled (const int *tracerIdentifier*) const throw (std::exception) [inherited]

test if a tracer of the group is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
bool

Return values:
true if tracer is enabled

6.23.3 Member Data Documentation

6.23.3.1 LocalTracerGroupAbstract* _tracerGroupAbstract [protected, inherited]

private

6.24 LocalTracer Class Reference

Inheritance diagram for LocalTracer:

6.24.1 Detailed Description

Base class that allows controlling a local tracer in ZeBu hardware.

Public Types

- enum EdgeType { **POSEDGE** = 0, **NEGEDGE** }

  edge type

Public Member Functions

- virtual ~LocalTracer ()

  destructor.

- const char * getPath () const throw (std::exception)

  get the tracer's path

- const char * getName () const throw (std::exception)

  get the tracer's name

- const char * getFullname () const throw (std::exception)

  get the tracer's fullname

- long long unsigned enable () throw (std::exception)

  enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable () throw (std::exception)

  disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled () const throw (std::exception)

  test if the tracer is enabled.

Static Public Member Functions

- void SelectSamplingClock (Board *board, const char *clockName, const EdgeType edgeType)

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

Protected Member Functions

- LocalTracer ()

  constructor

Protected Attributes

- LocalTracerAbstract * _tracerAbstract

  private

6.24.2 Member Enumeration Documentation

6.24.2.1 enum EdgeType

edge type

6.24.3 Constructor & Destructor Documentation

6.24.3.1 virtual ~LocalTracer () [virtual]

destructor.

Note:
disable the tracer.

See also:
disable

6.24.3.2 LocalTracer () [protected]

constructor

6.24.4 Member Function Documentation

6.24.4.1 long long unsigned disable () throw (std::exception)

disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.24.4.2 long long unsigned enable () throw (std::exception)

enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.24.4.3 const char∗ getFullname () const throw (std::exception)

get the tracer's fullname

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's fullname

6.24.4.4 const char∗ getName () const throw (std::exception)

get the tracer's name

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's name

6.24.4.5 const char∗ getPath () const throw (std::exception)

get the tracer's path

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's path

6.24.4.6 bool isEnabled () const throw (std::exception)

test if the tracer is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Returns:
bool

Return values:
true if tracer is enabled

6.24.4.7 void SelectSamplingClock (Board ∗ *board*, const char ∗ *clockName*, const EdgeType *edgeType*) [static]

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

6.24.5 Member Data Documentation

6.24.5.1 LocalTracerAbstract∗ _tracerAbstract [protected]

private

6.25 LocalTraceReader Class Reference

Inheritance diagram for LocalTraceReader:Collaboration diagram for LocalTraceReader:

6.25.1 Detailed Description

Allows controlling a local tracer in ZeBu hardware and reading values of tracer's signals.

Note:
Not supported in zTide environment.
Only one LocalTracer instance can be used per tracer at the same time.

Public Types

- enum EdgeType { **POSEDGE** = 0, **NEGEDGE** }
  edge type

Public Member Functions

- LocalTraceReader ()
  constructor
- ~LocalTraceReader ()
  destructor.
- void initialize (Board *board, const char *fullname, const int thread=-1) throw (std::exception)
  initialize the object
- int step () throw (std::exception)
  read the next trace cycle. Block until the next trace cycle is received.
- int run (const long long unsigned &numberOfCycles) throw (std::exception)
  read the nth next trace cycles. Block until the nth next trace cycles are received.
- int waitNextChange () throw (std::exception)
  wait for next value change. Block until the nth next value change is received.
- int tryStep () throw (std::exception)

try to read the next trace cycle.

- int tryRun (const long long unsigned &numberOfCycles) throw (std::exception)

  try to read the nth next trace cycles.

- int getNextChange () throw (std::exception)

  try to get next value change.

- long long unsigned getTime () const throw (std::exception)

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

- void getTime (long long unsigned &time, bool &state) const throw (std::exception)

  get the current timestamp and the current state of the tracer. The timestamp is the number of cycles generated for the clock synchronizing all tracers. The state specifies if the tracer was enabled at the current timestamp.

- const Signal * getSignal (const char *name) throw (std::exception)

  get a signal handler to read its value

- const char * getPath () const throw (std::exception)

  get the tracer's path

- const char * getName () const throw (std::exception)

  get the tracer's name

- const char * getFullname () const throw (std::exception)

  get the tracer's fullname

- long long unsigned enable () throw (std::exception)

  enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable () throw (std::exception)

  disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled () const throw (std::exception)

  test if the tracer is enabled.

Static Public Member Functions

- void SelectSamplingClock (Board *board, const char *clockName, const EdgeType edgeType)

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

Protected Attributes

- LocalTracerAbstract * _tracerAbstract

  private

6.25.2 Member Enumeration Documentation

6.25.2.1 enum EdgeType [inherited]

edge type

6.25.3 Constructor & Destructor Documentation

6.25.3.1 LocalTraceReader ()

constructor

Note:
do not enable the tracer

Example:

Initialize and enable the tracer LocalTraceReader tracer; tracer.initialize(zebu, <fullname>); long long unsigned time = tracer.enable(); std::cout << "enable tracer=" << tracer->getFullname() << " at time=" << std::dec << time << std::endl;

Get signal handler Signal &myBus = *tracer.getSignal("myBus");

Read 10 trace cycles bool state = false; for(int cycle = 0; cycle < 10; ++cycle) { tracer.run(1); std::cout << std::dec << "time=" << tracer.getTime() << " myBus=" << std::hex << myBus << std::endl; }

Disable tracer and read acquired value changes until end of trace time = tracer.disable(); std::cout << "disable tracer=" << tracer->getFullname() << " at time=" << std::dec << time << std::endl; while(tracer.waitNextChange() == 0) { std::cout << std::dec << "time=" << tracer.getTime() << " myBus=" << std::hex << myBus; }

6.25.3.2 ~LocalTraceReader ()

destructor.

Note:
disable the tracer.

See also:
disable
dump

6.25.4 Member Function Documentation

6.25.4.1 long long unsigned disable () throw (std::exception) [inherited]

disable the tracer No more trace is sent by the hardware even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.25.4.2 long long unsigned enable () throw (std::exception) [inherited]

enable the tracer. The trace is sent by the hardware if the enable pin of the tracer macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.25.4.3 const char* getFullname () const throw (std::exception) [inherited]

get the tracer's fullname

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's fullname

6.25.4.4 const char∗ getName () const throw (std::exception) [inherited]

get the tracer's name

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's name

6.25.4.5 int getNextChange () throw (std::exception)

try to get next value change.

Returns:
int

Return values:
0 if sucessfull

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

1 if the next value change is not available yet

6.25.4.6 const char∗ getPath () const throw (std::exception) [inherited]

get the tracer's path

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's path

6.25.4.7 const Signal* getSignal (const char * *name*) throw (std::exception)

get a signal handler to read its value

Parameters:

name port name of the signal in the tracer. If the tracer was disabled at the current time, values returned by methods of Signal class are equal to a 0 in 2-state value or a X in 4-state value

Note:

use Signal::fetchValue to read the 4-state value of a signal.

See also:

Signal class

6.25.4.8 void getTime (long long unsigned & *time*, bool & *state*) const throw (std::exception)

get the current timestamp and the current state of the tracer. The timestamp is the number of cycles generated for the clock synchronizing all tracers. The state specifies if the tracer was enabled at the current timestamp.

Parameters:

time the current timestamp

state true if the tracer was enabled corresponding with the current values of tracer's signals.

6.25.4.9 long long unsigned getTime () const throw (std::exception)

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

6.25.4.10 void initialize (Board * *board*, const char * *fullname*, const int *thread* = -1) throw (std::exception)

initialize the object

Parameters:

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.

6.25.4.11 bool isEnabled () const throw (std::exception) `[inherited]`

test if the tracer is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Returns:
bool

Return values:
true if tracer is enabled

6.25.4.12 int run (const long long unsigned & *numberOfCycles*) throw (std::exception)

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

Returns:
int

Return values:
0 if sucessfull

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

6.25.4.13 void SelectSamplingClock (Board * *board*, const char * *clockName*, const EdgeType *edgeType*) `[static, inherited]`

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

6.25.4.14 int step () throw (std::exception)

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

Returns:
int

Return values:
0 if sucessfull

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

6.25.4.15 int tryRun (const long long unsigned & *numberOfCycles*) throw (std::exception)

try to read the nth next trace cycles.

Returns:
int

Return values:
the number of read cycles

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

6.25.4.16 int tryStep () throw (std::exception)

try to read the next trace cycle.

Returns:
int

Return values:
0 if sucessfull

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

1 if the next trace cycle is not available yet

6.25.4.17 int waitNextChange () throw (std::exception)

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

Returns:
bool

Return values:

0 if sucessfull

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

6.25.5 Member Data Documentation

6.25.5.1 LocalTracerAbstract* _tracerAbstract [protected, inherited]

private

6.26 LocalTraceReaderGroup Class Reference

Inheritance diagram for LocalTraceReaderGroup:Collaboration diagram for LocalTraceReaderGroup:

6.26.1 Detailed Description

Allows controlling a group of local tracers in ZeBu hardware and reading values of signals of all tracers in the group.

Note:
Not supported in zTide environment.
Only one LocalTracer or LocalTracerGroup instance can be used per tracer at the same time.

Public Member Functions

- LocalTraceReaderGroup ()
  constructor
- ~LocalTraceReaderGroup ()
  destructor.
- void initialize (Board *board, const int thread=-1) throw (std::exception)
  initialize the object
- int step () throw (std::exception)
  read the next trace cycle. Block until the next trace cycle is received.
- int run (const long long unsigned &numberOfCycles) throw (std::exception)
  read the nth next trace cycles. Block until the nth next trace cycles are received.
- int waitNextChange () throw (std::exception)
  wait for next value change. Block until the nth next value change is received.
- int tryStep () throw (std::exception)
  try to read the next trace cycle.
- int tryRun (const long long unsigned &numberOfCycles) throw (std::exception)
  try to read the nth next trace cycles.

- int getNextChange () throw (std::exception)

  try to get next value change.

- long long unsigned getTime () const throw (std::exception)

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

- void getTime (long long unsigned &time, bool &state) const throw (std::exception)

  get the current timestamp and the current state of the group. The timestamp is the number of cycles generated for the clock synchronizing all tracers. The state specifies if at least one tracer of the group was enabled at the current timestamp.

- const Signal * getSignal (const int tracerIdentifier, const char *name) throw (std::exception)

  get a signal handler to read its value

- int add (const char *fullname) throw (std::exception)

  add a tracer in the group

- void add (const char *regularExpression, const bool invert, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

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

- int getNumberOfTracers () const throw (std::exception)

  get the number of tracers added in the group

- int getIdentifier (const char *fullname) const throw (std::exception)

  get the identfier of a tracer added in the group

- const char * getPath (const int tracerIdentifier) const throw (std::exception)

  get the path of a tracer

- const char * getName (const int tracerIdentifier) const throw (std::exception)

  get the name of a tracer

- const char * getFullname (const int tracerIdentifier) const throw (std::exception)

  get the fullname of a tracer

- long long unsigned enable (const int tracerIdentifier) throw (std::exception)

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable (const int tracerIdentifier) throw (std::exception)

  disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled (const int tracerIdentifier) const throw (std::exception)

  test if a tracer of the group is enabled.

- long long unsigned enableAll () throw (std::exception)

  enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

- long long unsigned disableAll () throw (std::exception)

  disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Protected Attributes

- LocalTracerGroupAbstract * _tracerGroupAbstract

  private

6.26.2 Constructor & Destructor Documentation

6.26.2.1 LocalTraceReaderGroup ()

constructor

Example:

Initialize and enable all tracers of the group LocalTraceDumperGroup group; group.initialize(zebu); int tracerIdentifier1 = group.add(<fullname1>); int tracerIdentifier2 = group.add(<fullname2>); [...] long long unsigned time = group.enableAll(); std::cout << "enable group at time=" << std::dec << time << std::endl;

Get signal handler Signal &myBus1 = *group.getSignal(tracerIdentifier1, "myBus"); Signal &myBus2 = *group.getSignal(tracerIdentifier2, "myBus");

Read 10 trace cycles bool state = false; for(int cycle = 0; cycle < 10; ++cycle) { tracer.run(1); std::cout << std::dec << "time=" << group.getTime() << std::endl;

std::cout << " myBus1=" << std::hex << myBus1 << std::endl; std::cout << " myBus2=" << std::hex << myBus2 << std::endl; }

Disable all tracers of the group and read acquired value changes until end of trace time = group.disableAll(); std::cout << "disable group at time=" << std::dec << time << std::endl; while(group.waitNextChange() == 0) { std::cout << std::dec << "time=" << group.getTime() << std::endl; std::cout << " myBus1=" << std::hex << myBus1 << std::endl; std::cout << " myBus2=" << std::hex << myBus2 << std::endl; }

6.26.2.2 ~LocalTraceReaderGroup ()

destructor.

Note:
disable all tracers of the group.

See also:
disable
dump

6.26.3 Member Function Documentation

6.26.3.1 void add (const char * *regularExpression*, const bool *invert*, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`) throw (std::exception) `[inherited]`

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

Parameters:
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

6.26.3.2 int add (const char * *fullname*) throw (std::exception) [inherited]

add a tracer in the group

Parameters:

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:

int

Return values:

identifier of the tracer in the group

6.26.3.3 long long unsigned disable (const int *tracerIdentifier*) throw (std::exception) [inherited]

disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Parameters:

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

Returns:

long long unsigned

Return values:

timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:

add
getIdentifier

6.26.3.4 long long unsigned disableAll () throw (std::exception) [inherited]

disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.26.3.5 long long unsigned enable (const int *tracerIdentifier*) throw (std::exception) `[inherited]`

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.26.3.6 long long unsigned enableAll () throw (std::exception) `[inherited]`

enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.26.3.7 const char* getFullname (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the fullname of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

See also:
add
getIdentifier

6.26.3.8 int getIdentifier (const char * *fullname*) const throw (std::exception) [inherited]

get the identfier of a tracer added in the group

Parameters:
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:
int

See also:
add

6.26.3.9 const char* getName (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the name of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's name

See also:
add
getIdentifier

6.26.3.10 int getNextChange () throw (std::exception)

try to get next value change.

Returns:
int

Return values:
0 if sucessfull

-1 if all tracers of the group are disabled and if the end of trace has been reached

1 if the next value change is not available yet

6.26.3.11 int getNumberOfTracers () const throw (std::exception) [inherited]

get the number of tracers added in the group

Return values:
int

See also:
add

6.26.3.12 const char* getPath (const int *tracerIdentifier*) const throw (std::exception) [inherited]

get the path of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's path

See also:
add
getIdentifier

6.26.3.13 const Signal∗ getSignal (const int *tracerIdentifier*, const char ∗ *name*) throw (std::exception)

get a signal handler to read its value

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method
name port name of the signal in the specified tracer. If the tracer was disabled at the current time, values returned by methods of Signal class are equal to a 0 in 2-state value or a X in 4-state value

Note:
use Signal::fetchValue to read the 4-state value of a signal.

See also:
Signal class

6.26.3.14 void getTime (long long unsigned & *time*, bool & *state*) const throw (std::exception)

get the current timestamp and the current state of the group. The timestamp is the number of cycles generated for the clock synchronizing all tracers. The state specifies if at least one tracer of the group was enabled at the current timestamp.

Parameters:
time the current timestamp
state true if at least one tracer of the group was enabled corresponding with the current values of signals of all tracers in the group.

6.26.3.15 long long unsigned getTime () const throw (std::exception)

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

6.26.3.16 void initialize (Board * *board*, const int *thread* = -1) throw (std::exception)

initialize the object

Parameters:

board C++ handler on Board

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

6.26.3.17 bool isEnabled (const int *tracerIdentifier*) const throw (std::exception) [inherited]

test if a tracer of the group is enabled.

Note:

do not test if the enable pin of the tracer macro is asserted.

Parameters:

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

Returns:

bool

Return values:

true if tracer is enabled

6.26.3.18 int run (const long long unsigned & *numberOfCycles*) throw (std::exception)

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

Returns:

int

Return values:

0 if sucessfull

-1 if all tracers of the group are disabled and if the end of trace has been reached

6.26.3.19 int step () throw (std::exception)

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

Returns:
int

Return values:
0 if sucessfull

-1 if all tracers of the group are disabled and if the end of trace has been reached

6.26.3.20 int tryRun (const long long unsigned & *numberOfCycles*) throw (std::exception)

try to read the nth next trace cycles.

Returns:
int

Return values:
the number of read cycles

-1 if all tracers of the group are disabled and if the end of trace has been reached

6.26.3.21 int tryStep () throw (std::exception)

try to read the next trace cycle.

Returns:
int

Return values:
0 if sucessfull

-1 if all tracers of the group are disabled and if the end of trace has been reached

1 if the next trace cycle is not available yet

6.26.3.22 int waitNextChange () throw (std::exception)

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

Returns:
bool

Return values:

0 if sucessfull

-1 if all tracers of the group are disabled and if the end of trace has been reached

6.26.4 Member Data Documentation

6.26.4.1 LocalTracerGroupAbstract* _tracerGroupAbstract [protected, inherited]

private

6.27 LocalTracerGroup Class Reference

Inheritance diagram for LocalTracerGroup:

6.27.1 Detailed Description

Base class that allows controlling a group of local tracers in ZeBu hardware.

Note:
Not supported in zTide environment.
Only one LocalTracer or LocalTracerGroup instance can be used per tracer at the same time.

Public Member Functions

- virtual ~LocalTracerGroup ()

  destructor.

- int add (const char *fullname) throw (std::exception)

  add a tracer in the group

- void add (const char *regularExpression, const bool invert, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

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

- int getNumberOfTracers () const throw (std::exception)

  get the number of tracers added in the group

- int getIdentifier (const char *fullname) const throw (std::exception)

  get the identfier of a tracer added in the group

- const char * getPath (const int tracerIdentifier) const throw (std::exception)

  get the path of a tracer

- const char * getName (const int tracerIdentifier) const throw (std::exception)

  get the name of a tracer

- const char * getFullname (const int tracerIdentifier) const throw (std::exception)

  get the fullname of a tracer

- long long unsigned enable (const int tracerIdentifier) throw (std::exception)

  enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

- long long unsigned disable (const int tracerIdentifier) throw (std::exception)

  disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

- bool isEnabled (const int tracerIdentifier) const throw (std::exception)

  test if a tracer of the group is enabled.

- long long unsigned enableAll () throw (std::exception)

  enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

- long long unsigned disableAll () throw (std::exception)

  disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Protected Member Functions

- LocalTracerGroup ()

  constructor

Protected Attributes

- LocalTracerGroupAbstract * _tracerGroupAbstract

  private

6.27.2 Constructor & Destructor Documentation

6.27.2.1 virtual ~LocalTracerGroup () [virtual]

destructor.

Note:
disable the tracer.

See also:
disable

6.27.2.2 LocalTracerGroup () [protected]

constructor

6.27.3 Member Function Documentation

6.27.3.1 void add (const char * *regularExpression*, const bool *invert*, const bool *ignoreCase* = false, const char *hierarchicalSeparator* = '.') throw (std::exception)

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

Parameters:

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

6.27.3.2 int add (const char * *fullname*) throw (std::exception)

add a tracer in the group

Parameters:

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:
int

Return values:
identifier of the tracer in the group

6.27.3.3 long long unsigned disable (const int *tracerIdentifier*) throw (std::exception)

disable a tracer of the group. No more trace is sent by the hardware for the tracer even if the enable pin of the tracer macro is asserted. When a tracer is disabled, all tracers mapped on the same FPGA are flushed. Disabling can slowdown.local trace infrastructure.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.27.3.4 long long unsigned disableAll () throw (std::exception)

disable all tracers of the group. No more trace is sent by the hardware even if the enable pins of the tracers macro are asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been disabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.27.3.5 long long unsigned enable (const int *tracerIdentifier*) throw (std::exception)

enable a tracer of the group. The trace of the tracer is sent by the hardware if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
long long unsigned

Return values:
timestamp at which the tracer has been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

See also:
add
getIdentifier

6.27.3.6 long long unsigned enableAll () throw (std::exception)

enable all tracers of the group. The trace of a tracer is sent by the hardware if its enable pin macro is asserted.

Returns:
long long unsigned

Return values:
timestamp at which the tracers have been enabled. The timestamp is the number of cycles generated for the clock synchronizing tracers.

6.27.3.7 const char* getFullname (const int *tracerIdentifier*) const throw (std::exception)

get the fullname of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's fullname

See also:
add
getIdentifier

6.27.3.8 int getIdentifier (const char ∗ *fullname*) const throw (std::exception)

get the identfier of a tracer added in the group

Parameters:
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:
int

See also:
add

6.27.3.9 const char∗ getName (const int *tracerIdentifier*) const throw (std::exception)

get the name of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char ∗

Return values:
NULL terminated C string containing tracer's name

See also:
add
getIdentifier

6.27.3.10 int getNumberOfTracers () const throw (std::exception)

get the number of tracers added in the group

Return values:
int

See also:
add

6.27.3.11 const char* getPath (const int *tracerIdentifier*) const throw (std::exception)

get the path of a tracer

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
const char *

Return values:
NULL terminated C string containing tracer's path

See also:
add
getIdentifier

6.27.3.12 bool isEnabled (const int *tracerIdentifier*) const throw (std::exception)

test if a tracer of the group is enabled.

Note:
do not test if the enable pin of the tracer macro is asserted.

Parameters:
tracerIdentifier identifier of the tracer in the group returned by the add method

Returns:
bool

Return values:
true if tracer is enabled

6.27.4 Member Data Documentation

6.27.4.1 LocalTracerGroupAbstract* _tracerGroupAbstract [protected]

private

6.28 LogicAnalyzer Class Reference

6.28.1 Detailed Description

interface for ZeBu logic analyer

Logic analyzer is used to stop emulation when an event occurs on a trigger. It is possible too to program the trace monitor to start when an event occurs.

Public Member Functions

- void start (const char ∗clockName, const char ∗edgeName=”posedge”) throw (std::exception)

  start logic analyzing

- void stop () throw (std::exception)

  stop logic analyzing

- void stopOnTrigger (Trigger ∗trig) throw (std::exception)

  program logic analyzer to stop clocks on trigger event

- void stopOnTrigger (const char ∗triggerName) throw (std::exception)

  stop all ZeBu clocks

- void traceOnTrigger (Trigger ∗trig) throw (std::exception)

  start monitor trace in post-trigger part of the memory.

- void traceOnTrigger (const char ∗triggerName) throw (std::exception)

  start monitor trace in post-trigger part of the memory.

6.28.2 Member Function Documentation

6.28.2.1 void start (const char ∗ *clockName*, const char ∗ *edgeName* = "posedge") throw (std::exception)

start logic analyzing

Parameters:

clockName name of the clock used to change logic analyzer state

edgeName name of the edge used to change logic analyzer state could be ”posedge” or ”negedge”. Default is ”posedge”

See also:
LogicAnalyzer::stop

6.28.2.2 void stop () throw (std::exception)

stop logic analyzing

See also:
LogicAnalyzer::start

6.28.2.3 void stopOnTrigger (const char * *triggerName*) throw (std::exception)

stop all ZeBu clocks

Parameters:
triggerName name of the trigger on which clocks stop

See also:
LogicAnalyzer::traceOnTrigger
Clock

6.28.2.4 void stopOnTrigger (Trigger * *trig*) throw (std::exception)

program logic analyzer to stop clocks on trigger event

Parameters:
trig pointer to the trigger on which clocks stop

See also:
LogicAnalyzer::traceOnTrigger
Clock

6.28.2.5 void traceOnTrigger (const char * *triggerName*) throw (std::exception)

start monitor trace in post-trigger part of the memory.

Parameters:
triggerName name of the trigger on which trace begins

See also:
LogicAnalyzer::stopOnTrigger
TraceMemory

6.28.2.6 void traceOnTrigger (Trigger * *trig*) throw (std::exception)

start monitor trace in post-trigger part of the memory.

Parameters:

trig pointer to the trigger on which trace begins

See also:

LogicAnalyzer::stopOnTrigger
TraceMemory

6.29 LoopBreak Class Reference

6.29.1 Detailed Description

This class 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.

Static Public Member Functions

- void setInjectedValue (Board ∗board, const char ∗loopPath, unsigned int value) throw (std::exception)

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

- void alwaysBreak (Board ∗board, const char ∗loopPath, bool value) throw (std::exception)

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

6.29.2 Member Function Documentation

6.29.2.1 void alwaysBreak (Board ∗ *board*, const char ∗ *loopPath*, bool *value*) throw (std::exception) [static]

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.

6.29.2.2 void setInjectedValue (Board ∗ *board*, const char ∗ *loopPath*, unsigned int *value*) throw (std::exception) [static]

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.

6.30 MckCDriver Class Reference

Inheritance diagram for MckCDriver:Collaboration diagram for MckCDriver:

6.30.1 Detailed Description

Implement a multi-clock driver.

Public Member Functions

- void dumpfile (const char *filename, int compression=0) throw (std::exception)

  specify the name of the driver waveform file

- virtual void dumpvars (Signal *signal=NULL) throw (std::exception)

  select driver signals to dump

- virtual void dumpon () throw (std::exception)

  resume the dump. Switch partial driver waveform dump on

- virtual void dumpoff () throw (std::exception)

  suspend the dump

- virtual void dumpclosefile () throw (std::exception)

  close the waveform file open from MckCDriver::dumpfile

- virtual void closeDumpfile () throw (std::exception)

  obsolete

- virtual unsigned int run (unsigned int, bool) const __attribute__((deprecated)) throw (std::exception)

  obsolete

- virtual unsigned int wait (unsigned int triggers, unsigned int timeOut=0xffffffff) const throw (std::exception)

  no effect

- void update () const throw (std::exception)

  send input signals to the board and read output signals generates an exchange between the bench and the board.

- virtual unsigned int run (unsigned int) const throw (std::exception)

no effect

- unsigned int connect () throw (std::exception)

  connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char ∗ name () const throw (std::exception)

  get the driver's name

- void registerCallback (void(∗)(void ∗callback), void ∗user) throw (std::exception)

  register a callback

- Signal ∗ getSignal (const char ∗name) const throw (std::exception)

  get a signal handler

- void dumpvars (const char ∗name) throw (std::exception)

  select driver signals to dump

- virtual void dumpon (char ∗clockName, char ∗edgeName="posedge") throw (std::exception)

  command the start the trace memory

- virtual void storeToFile () throw (std::exception)

  command the download and the dump of the trace memory

- virtual void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual void setPreTriggerRatio (float size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual bool isTraceMemoryDriver () throw (std::exception)

  returns true if the driver is a trace driver

6.30.2 Member Function Documentation

6.30.2.1 virtual void closeDumpfile () throw (std::exception) `[virtual]`

obsolete

Reimplemented from Driver.

6.30.2.2 unsigned int connect () throw (std::exception) [inherited]

connect driver

Returns:
unsigned int

Return values:
0 OK

$>$***0*** KO

See also:
Driver::disconnect

6.30.2.3 void disconnect () throw (std::exception) [inherited]

disconnect driver

See also:
Driver::connect

6.30.2.4 virtual void dumpclosefile () throw (std::exception) [virtual]

close the waveform file open from MckCDriver::dumpfile

Note:
not supported in zTide environment

Reimplemented from Driver.

6.30.2.5 void dumpfile (const char * *filename*, int *compression* = 0) throw (std::exception) [virtual]

specify the name of the driver waveform file

Parameters:
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

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

See also:
MckCDriver::dumpclosefile
MckCDriver::dumpvars
MckCDriver::dumpon
MckCDriver::dumpoff

Reimplemented from Driver.

6.30.2.6 virtual void dumpoff () throw (std::exception) `[virtual]`

suspend the dump

switch partial driver waveform dump off. This is default.

See also:
MckCDriver::dumpvars
MckCDriver::dumpon
MckCDriver::dumpfile

Reimplemented from Driver.

6.30.2.7 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) `[virtual, inherited]`

command the start the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.30.2.8 virtual void dumpon () throw (std::exception) `[virtual]`

resume the dump. Switch partial driver waveform dump on

See also:
MckCDriver::dumpvars
MckCDriver::dumpfile
MckCDriver::dumpoff

Reimplemented from Driver.

6.30.2.9 void dumpvars (const char * *name*) throw (std::exception) [inherited]

select driver signals to dump

Parameters:
name name of the signal to be dumped.

Note:
no signal can be added after first run.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.30.2.10 virtual void dumpvars (Signal * *signal* = NULL) throw (std::exception) [virtual]

select driver signals to dump

Parameters:
signal handler to 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.

See also:
MckCDriver::dumpfile
MckCDriver::dumpon
MckCDriver::dumpoff

Reimplemented from Driver.

6.30.2.11 Signal* getSignal (const char * *name*) const throw (std::exception) [inherited]

get a signal handler

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

6.30.2.12 virtual bool isTraceMemoryDriver () throw (std::exception) [virtual, inherited]

returns true if the driver is a trace driver

Reimplemented in TraceMemory.

6.30.2.13 const char* name () const throw (std::exception) [inherited]

get the driver's name

Returns:
const char *

Return values:
NULL terminated C string containing driver's name

6.30.2.14 void registerCallback (void(*)(void *callback), void * *user*) throw (std::exception) [inherited]

register a callback

Parameters:
callback callback

user user data

6.30.2.15 virtual unsigned int run (unsigned *int*) const throw (std::exception) [virtual]

no effect

Implements Driver.

6.30.2.16 virtual unsigned int run (unsigned *int*, bool) const throw (std::exception) [virtual]

obsolete

Implements Driver.

6.30.2.17 virtual void setPreTriggerRatio (float *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.30.2.18 virtual void setPreTriggerSize (unsigned int *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.30.2.19 virtual void storeToFile () throw (std::exception) `[virtual, inherited]`

command the download and the dump of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.30.2.20 void update () const throw (std::exception) `[virtual]`

send input signals to the board and read output signals generates an exchange between the bench and the board.

Reimplemented from Driver.

6.30.2.21 virtual unsigned int wait (unsigned int *triggers*, unsigned int *timeOut* = `0xffffffff`) const throw (std::exception) `[virtual]`

no effect

Reimplemented from Driver.

6.31 Memory Class Reference

Inheritance diagram for Memory:

6.31.1 Detailed Description

Implement public interface class for Memory instances in the DUT.

Note:
Not completly supported in zTide environment.

See also:
Board::getMemory

Public Member Functions

- unsigned int depth () const throw (std::exception)

  get informations about the size of the memory

- unsigned int width () const throw (std::exception)
- const char * name () const throw (std::exception)

  get the memory instance name

- const char * fullname (char separator= '.') const throw (std::exception)

  get the hierarchical name of the memory instance

- unsigned int * newWordBuffer () throw (std::exception)
- int readWord (const unsigned int address, unsigned int *wordBuffer) throw (std::exception)

  get the value of a memory word

- int writeWord (const unsigned int address, const unsigned int *wordBuffer) throw (std::exception)

  set the value of a memory word

- unsigned int * newBuffer () throw (std::exception)

  create a buffer for read/write operations

- void storeTo (unsigned int *buffer) throw (std::exception)

  copy the content of the memory into the buffer

- void storeTo (unsigned int *buffer, const unsigned int firstAddress, const unsigned int lastAddress) throw (std::exception)

  copy the content of the memory into the buffer

- void loadFrom (const unsigned int *buffer) throw (std::exception)

  copy the content of the buffer into the memory

- void loadFrom (const unsigned int *buffer, const unsigned int firstAddress, const unsigned int lastAddress) throw (std::exception)

  copy the content of the buffer into the memory

- int storeTo (const char *fname, const unsigned int firstAddress, const unsigned int lastAddress, bool binary=false) throw (std::exception)

  copy the memory content into a file

- int storeTo (const char *fname, const bool binary=false) throw (std::exception)

  copy the memory content into a file

- int storeToHex (const char *filename, const unsigned int firstAddress, const unsigned int lastAddress, const bool useAddressRange=true, const bool dumpZero=true) throw (std::exception)

  copy the memory content into a hexadecimal format file

- int storeToRaw (const char *filename, const unsigned int firstAddress, const unsigned int lastAddress, const unsigned int compressionLevel=1) throw (std::exception)

  copy the memory content into a raw format file

- int loadFrom (const char *fname) throw (std::exception)

  load the content of a file into the memory

- int loadFromTxt (const char *filename, const char base) throw (std::exception)

  load the content of a human readable file into the memory

- int storeToTxt (const char *filename, const unsigned int firstAddress, const unsigned int lastAddress, const char base)

  Store the content of a memory to a human readable file.

- unsigned int readmemb (const char *filename)

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

- unsigned int readmemh (const char *filename)

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

- unsigned int writememb (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 writememh (const char ∗filename, unsigned int startAddress, unsigned int endAddress)

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

- void set (const unsigned int ∗wordBuffer, const unsigned int firstAddress, const unsigned int lastAddress) throw (std::exception)

  set a range of the memory to the same value

- void set (unsigned int pattern) throw (std::exception)

  set the whole memory to the same value

- void clear () throw (std::exception)

  clear the content of the memory

- void erase () throw (std::exception)

  erase the content of the memory

- ∼Memory () throw (std::exception)

  destructor

6.31.2 Constructor & Destructor Documentation

6.31.2.1 ∼Memory () throw (std::exception)

destructor

6.31.3 Member Function Documentation

6.31.3.1 void clear () throw (std::exception)

clear the content of the memory

See also:

Memory::set
Memory::erase

6.31.3.2 unsigned int depth () const throw (std::exception)

get informations about the size of the memory

Returns:
the number of words of the memory instance

See also:
Memory::width

6.31.3.3 void erase () throw (std::exception)

erase the content of the memory

See also:
Memory::set
Memory::clear

6.31.3.4 const char* fullname (char *separator* = '.') const throw (std::exception)

get the hierarchical name of the memory instance

Parameters:
separator hierarchical separator character

Return values:
NULL terminated C string containing the memory name

6.31.3.5 int loadFrom (const char * *fname*) throw (std::exception)

load the content of a file into the memory

Parameters:
fname name of the memory file

Returns:
int status

Return values:
0 OK

<>*0* KO

See also:
Memory::storeTo

6.31.3.6 void loadFrom (const unsigned int * *buffer*, const unsigned int *firstAddress*, const unsigned int *lastAddress*) throw (std::exception)

copy the content of the buffer into the memory

Parameters:

buffer unsigned int table allocated by Memory::newBuffer

firstAddress first address to be loaded

lastAddress last address to be loaded

See also:
Memory::storeTo
Memory::newBuffer

6.31.3.7 void loadFrom (const unsigned int * *buffer*) throw (std::exception)

copy the content of the buffer into the memory

Parameters:

buffer unsigned int table allocated by Memory::newBuffer

See also:
Memory::storeTo
Memory::newBuffer

6.31.3.8 int loadFromTxt (const char * *filename*, const char *base*) throw (std::exception)

load the content of a human readable file into the memory

Parameters:

filename name of the memory file

base base of the data of the memory file to load ('h' or 'H' for hexadecimal, 'b' or 'B' for binary)

Returns:
int status

Return values:
0 OK

<>***0*** KO

See also:
Memory::storeTo

6.31.3.9 const char* name () const throw (std::exception)

get the memory instance name

Return values:
NULL terminated C string containing the memory name

6.31.3.10 unsigned int* newBuffer () throw (std::exception)

create a buffer for read/write operations

Note:
delete buffer with delete[]

See also:
Memory::storeTo
Memory::loadFrom

6.31.3.11 unsigned int* newWordBuffer () throw (std::exception)

create a buffer for read/write operations (delete with delete[])

See also:
Memory::readWord
Memory::writeWord

6.31.3.12 unsigned int readmemb (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 retval >1 otherwise.

6.31.3.13 unsigned int readmemh (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.

6.31.3.14 int readWord (const unsigned int *address*, unsigned int * *wordBuffer*) throw (std::exception)

get the value of a memory word

Parameters:

address address to be accessed

wordBuffer buffer to keep the value of the memory word

Returns:

int status

Return values:

0 OK

<>0 KO

See also:

Memory::newWordBuffer
Memory::writeWord

6.31.3.15 void set (unsigned int *pattern*) throw (std::exception)

set the whole memory to the same value

Parameters:

pattern specify a memory word value to set

6.31.3.16 void set (const unsigned int * *wordBuffer*, const unsigned int *firstAddress*, const unsigned int *lastAddress*) throw (std::exception)

set a range of the memory to the same value

Parameters:

wordBuffer specify a memory word value to set

firstAddress first address to set

lastAddress last address to set

See also:

Memory::clear
Memory::erase

6.31.3.17 int storeTo (const char * *fname*, const bool *binary* = `false`) throw (std::exception)

copy the memory content into a file

Parameters:

fname name of the memory file

binary optional boolean parameter. Dump memory as a binary file if true. Default is false.

Returns:

int status

Return values:

0 OK

<>***0*** KO

See also:

Memory::loadFrom

6.31.3.18 int storeTo (const char * *fname*, const unsigned int *firstAddress*, const unsigned int *lastAddress*, bool *binary* = `false`) throw (std::exception)

copy the memory content into a file

Parameters:

fname name of the memory file

firstAddress first address to be dumped

lastAddress last address to be dumped

binary optional boolean parameter.

Returns:

int status

Return values:

0 OK

<>***0*** KO Dump memory as a binary file if true. Default is false.

See also:

Memory::loadFrom

6.31.3.19 void storeTo (unsigned int * *buffer*, const unsigned int *firstAddress*, const unsigned int *lastAddress*) throw (std::exception)

copy the content of the memory into the buffer

Parameters:

buffer unsigned int table allocated by Memory::newBuffer

firstAddress first address to be dumped

lastAddress last address to be dumped

See also:

Memory::newBuffer
Memory::loadFrom

6.31.3.20 void storeTo (unsigned int * *buffer*) throw (std::exception)

copy the content of the memory into the buffer

Parameters:

buffer unsigned int table allocated by Memory::newBuffer

See also:
Memory::newBuffer
Memory::loadFrom

6.31.3.21 int storeToHex (const char * *filename*, const unsigned int *firstAddress*, const unsigned int *lastAddress*, const bool *useAddressRange* = `true`, const bool *dumpZero* = `true`) throw (std::exception)

copy the memory content into a hexadecimal format file

Parameters:

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:
int status

Return values:

0 OK

See also:
Memory::loadFrom

6.31.3.22 int storeToRaw (const char * *filename*, const unsigned int *firstAddress*, const unsigned int *lastAddress*, const unsigned int *compressionLevel* = `1`) throw (std::exception)

copy the memory content into a raw format file

Parameters:

filename name of the memory file

firstAddress first address to be dumped

lastAddress last address to be dumped

compressionLevel level of compression

Returns:
int status

Return values:

0 OK

<>***0*** KO

See also:

Memory::loadFrom

6.31.3.23 int storeToTxt (const char * *filename*, const unsigned int *firstAddress*, const unsigned int *lastAddress*, const char *base*)

Store the content of a memory to a human readable file.

Parameters:

filename name of the memory file

firstAddress first address to be dumped

lastAddress last address to be dumped

base format of the data in the file ('h' or 'H' for hexadecimal, 'b' or 'B' for binary)

Returns:

int status

Return values:

0 OK

<>***0*** KO

6.31.3.24 unsigned int width () const throw (std::exception)

Returns:

the number of bits per memory word

See also:

Memory::depth

6.31.3.25 unsigned int writememb (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.

6.31.3.26 unsigned int writememh (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.

6.31.3.27 int writeWord (const unsigned int *address*, const unsigned int * *wordBuffer*) throw (std::exception)

set the value of a memory word

Parameters:

address address to be accessed

wordBuffer buffer to keep the value of the memory word

Returns:

int status

Return values:

0 OK

$<>$***0*** KO

See also:

Memory::newWordBuffer
Memory::readWord

6.32 Monitor Class Reference

Inheritance diagram for Monitor:Collaboration diagram for Monitor:

6.32.1 Detailed Description

Implement ZeBu monitor driver base class.

Public Member Functions

- Monitor (const char *filename) throw (std::exception)

  constructor

- virtual ~Monitor () throw (std::exception)

  destructor

- virtual unsigned int run (unsigned int numCycles, bool block) const __attribute__((deprecated)) throw (std::exception)

  obsolete

- virtual void dumpfile (const char *filename, int level=0) throw (std::exception)

  specify the name of a waveform file

- virtual void dumpvars (Signal *signal=NULL) throw (std::exception)

  select internal register to dump

- virtual void dumpon () throw (std::exception)

  resume the dump. Switch partial readback waveform dump on

- virtual void dumpoff () throw (std::exception)

  suspend the dump switch partial readback waveform dump off. This is default.

- virtual void dumpclosefile () throw (std::exception)

  close the waveform file open from Monitor::dumpfile

- virtual void closeDumpfile () throw (std::exception)

  obsolete

- virtual unsigned int run (unsigned int numCycles) const throw (std::exception)

  run a number of cycles

- unsigned int connect () throw (std::exception)

  connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char * name () const throw (std::exception)

  get the driver's name

- virtual unsigned int wait (unsigned int triggers, unsigned int timeOut=0xffffffff) const throw (std::exception)

  wait for a trigger event or timeout while running the clock

- virtual void update () const throw (std::exception)

  update IOs

- void registerCallback (void(*)(void *callback), void *user) throw (std::exception)

  register a callback

- Signal * getSignal (const char *name) const throw (std::exception)

  get a signal handler

- void dumpvars (const char *name) throw (std::exception)

  select driver signals to dump

- virtual void dumpon (char *clockName, char *edgeName="posedge") throw (std::exception)

  command the start the trace memory

- virtual void storeToFile () throw (std::exception)

  command the download and the dump of the trace memory

- virtual void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual void setPreTriggerRatio (float size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual bool isTraceMemoryDriver () throw (std::exception)

  returns true if the driver is a trace driver

Protected Member Functions

- Monitor (DriverAbstract *imp) throw (std::exception)
  constructor

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Monitor (const char * *filename*) throw (std::exception)

constructor

6.32.2.2 virtual ~Monitor () throw (std::exception) [virtual]

destructor

6.32.2.3 Monitor (DriverAbstract * *imp*) throw (std::exception) [protected]

constructor

6.32.3 Member Function Documentation

6.32.3.1 virtual void closeDumpfile () throw (std::exception) [virtual]

obsolete

Reimplemented from Driver.

6.32.3.2 unsigned int connect () throw (std::exception) [inherited]

connect driver

Returns:
unsigned int

Return values:
0 OK

>***0*** KO

See also:
Driver::disconnect

6.32.3.3 void disconnect () throw (std::exception) [inherited]

disconnect driver

See also:
Driver::connect

6.32.3.4 virtual void dumpclosefile () throw (std::exception) [virtual]

close the waveform file open from Monitor::dumpfile

Note:
not supported in zTide environment

Reimplemented from Driver.

6.32.3.5 virtual void dumpfile (const char * *filename*, int *level* = 0) throw (std::exception) [virtual]

specify the name of a waveform file

Parameters:

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

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

See also:
Monitor::dumpclosefile
Monitor::dumpvars
Monitor::dumpon
Monitor::dumpoff

Reimplemented from Driver.

6.32.3.6 virtual void dumpoff () throw (std::exception) [virtual]

suspend the dump switch partial readback waveform dump off. This is default.

switch waveform dump off

See also:
Monitor::dumpvars
Monitor::dumpon
Monitor::dumpfile

Reimplemented from Driver.

6.32.3.7 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) [virtual, inherited]

command the start the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.32.3.8 virtual void dumpon () throw (std::exception) [virtual]

resume the dump. Switch partial readback waveform dump on

See also:
Monitor::dumpvars
Monitor::dumpfile
Monitor::dumpoff

Reimplemented from Driver.

6.32.3.9 void dumpvars (const char * *name*) throw (std::exception) [inherited]

select driver signals to dump

Parameters:
name name of the signal to be dumped.

Note:
no signal can be added after first run.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.32.3.10 virtual void dumpvars (Signal * *signal* = NULL) throw (std::exception) [virtual]

select internal register to dump

Parameters:

signal handler to 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.

See also:

Monitor::dumpfile
Monitor::dumpon
Monitor::dumpoff

Reimplemented from Driver.

6.32.3.11 Signal* getSignal (const char * *name*) const throw (std::exception) [inherited]

get a signal handler

Parameters:

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

6.32.3.12 virtual bool isTraceMemoryDriver () throw (std::exception) [virtual, inherited]

returns true if the driver is a trace driver

Reimplemented in TraceMemory.

6.32.3.13 const char* name () const throw (std::exception) [inherited]

get the driver's name

Returns:

const char *

Return values:

NULL terminated C string containing driver's name

6.32.3.14 void registerCallback (void(∗)(void ∗callback), void ∗ *user*) throw (std::exception) [inherited]

register a callback

Parameters:

callback callback

user user data

6.32.3.15 virtual unsigned int run (unsigned int *numCycles*) const throw (std::exception) [virtual]

run a number of cycles

Parameters:

numCycles number of cycles

Returns:

int

Return values:

0 OK

>***0*** KO

Implements Driver.

6.32.3.16 virtual unsigned int run (unsigned int *numCycles*, bool *block*) const throw (std::exception) [virtual]

obsolete

Implements Driver.

6.32.3.17 virtual void setPreTriggerRatio (float *size*) throw (std::exception) [virtual, inherited]

set the preTrigger memory size of the trace memory

Note:

no effect on other drivers

Reimplemented in TraceMemory.

6.32.3.18 virtual void setPreTriggerSize (unsigned int *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.32.3.19 virtual void storeToFile () throw (std::exception) `[virtual, inherited]`

command the download and the dump of the trace memory

Note:
no effect on other drivers

Reimplemented in TraceMemory.

6.32.3.20 virtual void update () const throw (std::exception) `[virtual, inherited]`

update IOs

equivalent to `run(0)`

Returns:
void

Reimplemented in MckCDriver.

6.32.3.21 virtual unsigned int wait (unsigned int *triggers*, unsigned int *timeOut* = `0xffffffff`) const throw (std::exception) `[virtual, inherited]`

wait for a trigger event or timeout while running the clock

Parameters:

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 stopping.

Returns:
unsigned int

Return values:
0 if a timeout occurs

bit i set to 1 for trigger i

Reimplemented in CDriver, and MckCDriver.

6.33 PartMemoryBuilder Class Reference

6.33.1 Detailed Description

Class used to create complex part memories.

This class is a container of sub-memory descriptions and their mapping into the memory to be created. An instance of this class is initialized with the requested width and depth of the part memory to build. The methods `AddPart` are then used to define each part of existing memory to map and their location in the new memory instance. Finally, the new memory instance is created with the method `CreatePartMemory`.

```
// This sample code creates a 8x8 part memory from a 16x16 memory instance with the following mappi
//     8                          4                          0
//     +--------------------------+--------------------------+ 0
//     | top.mem16x16[11:8][11:8] |  top.mem16x16[3:0][3:0]  |
//     +--------------------------+--------------------------+ 4
//     | top.mem16x16[3:0][11:8]  |  top.mem16x16[11:8][3:0] |
//     +--------------------------+--------------------------+ 8

// First, create an instance of the builder in order to construct
// a 8x8 part memory.
PartMemoryBuilder partBuilder(8,8);

// then, define four pieces of the existing memory instance
// and their mapping into the new memory.
partBuilder.AddPart ("top.mem16x16[3:0][3:0]",   0, 0);
partBuilder.AddPart ("top.mem16x16[11:8][11:8]", 4, 0);
partBuilder.AddPart ("top.mem16x16[3:0][11:8]",  4, 4);
partBuilder.AddPart ("top.mem16x16[11:8][3:0]",  0, 4);

// The previous code would also have been written as:
//partBuilder.AddPart ("top.mem16x16", 0, 0, 4, 4, 0, 0);
//partBuilder.AddPart ("top.mem16x16", 8, 8, 4, 4, 4, 0);
//partBuilder.AddPart ("top.mem16x16", 8, 0, 4, 4, 4, 4);
//partBuilder.AddPart ("top.mem16x16", 0, 8, 4, 4, 0, 4);

// Finally, create the new memory instance
Memory* memory = partBuilder.CreatePartMemory(zebuBoard);
```

Warning:
Holes and overlaps are not supported and are not checked.

Public Member Functions

- PartMemoryBuilder (unsigned int partWidth, unsigned int partDepth)

  Create a new PartMemoryBuilder instance.

- ~PartMemoryBuilder ()

  Destroy a PartMemoryBuilder instance.

- void AddPart (const char ∗memoryName, unsigned int partBit, unsigned int partAddress)

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

- void AddPart (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.

- Memory ∗ CreatePartMemory (Board ∗board)

  Create a new memory instance.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 PartMemoryBuilder (unsigned int *partWidth*, unsigned int *partDepth*)

Create a new PartMemoryBuilder instance.

Parameters:

partWidth Width in bits of the memory to create.

partDepth Depth in number of words of the memory to create.

6.33.2.2 ∼PartMemoryBuilder ()

Destroy a PartMemoryBuilder instance.

6.33.3 Member Function Documentation

6.33.3.1 void AddPart (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:

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.

6.33.3.2 void AddPart (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:

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.

6.33.3.3 Memory* CreatePartMemory (Board * *board*)

Create a new memory instance.

6.34 PatternDriver Class Reference

Inheritance diagram for PatternDriver:Collaboration diagram for PatternDriver:

6.34.1 Detailed Description

multi-clocks pattern driver.

Public Member Functions

- void setParam (unsigned int nbPrintError, unsigned int firstCheck, unsigned int lastCheck) throw (std::exception)

  set optionnal parameters.

- unsigned int loadFile (const char *fileName) throw (std::exception)

  Load a pattern file.

- void runAll () throw (std::exception)

  Send all the patterns. Send all the patterns contained in the file to the board, and compares the received patterns with the references.

- virtual unsigned int run (unsigned int nbPattern, bool block) const __attribute__-((deprecated)) throw (std::exception)

  obsolete

- unsigned int error () const throw (std::exception)

  Returns the number of wrong patterns detected.

- virtual unsigned int run (unsigned int nbPattern) const throw (std::exception)

  Send a specified number of patterns.

- unsigned int connect () throw (std::exception)

  connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char * name () const throw (std::exception)

  get the driver's name

- virtual unsigned int wait (unsigned int triggers, unsigned int timeOut=0xffffffff) const throw (std::exception)

wait for a trigger event or timeout while running the clock

- virtual void update () const throw (std::exception)

  update IOs

- void registerCallback (void(∗)(void ∗callback), void ∗user) throw (std::exception)

  register a callback

- Signal ∗ getSignal (const char ∗name) const throw (std::exception)

  get a signal handler

- virtual void dumpfile (const char ∗filename, int compression=0)

  specify the name of a waveform file

- virtual void dumpvars (Signal ∗signal=NULL)

  select driver signals to dump

- void dumpvars (const char ∗name) throw (std::exception)

  select driver signals to dump

- virtual void dumpon ()

  resume the dump

- virtual void dumpon (char ∗clockName, char ∗edgeName="posedge") throw (std::exception)

  command the start the trace memory

- virtual void dumpoff () throw (std::exception)

  suspend the dump

- virtual void dumpclosefile ()

  close the waveform file open from ::dumpfile

- virtual void closeDumpfile ()

  obsolete

- virtual void storeToFile () throw (std::exception)

  command the download and the dump of the trace memory

- virtual void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size of the trace memory

- virtual void setPreTriggerRatio (float size) throw (std::exception)
  set the preTrigger memory size of the trace memory
- virtual bool isTraceMemoryDriver () throw (std::exception)
  returns true if the driver is a trace driver

6.34.2 Member Function Documentation

6.34.2.1 virtual void closeDumpfile () [inline, virtual, inherited]

obsolete

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.2 unsigned int connect () throw (std::exception) [inherited]

connect driver

Returns:
unsigned int

Return values:
0 OK
>***0*** KO

See also:
Driver::disconnect

6.34.2.3 void disconnect () throw (std::exception) [inherited]

disconnect driver

See also:
Driver::connect

6.34.2.4 virtual void dumpclosefile () [inline, virtual, inherited]

close the waveform file open from ::dumpfile

Note:
not supported in zTide environment

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.5 virtual void dumpfile (const char * *filename*, int *compression* = 0) [virtual, inherited]

specify the name of a waveform file

Parameters:

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

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

See also:

Driver::dumpvars
Driver::dumpon
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.6 virtual void dumpoff () throw (std::exception) [inline, virtual, inherited]

suspend the dump

switch driver signals waveform dump off. This is default.

See also:

Driver::dumpvars
Driver::dumpon
Driver::dumpfile

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.7 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) [virtual, inherited]

command the start the trace memory

Note:

no effect on other drivers

Reimplemented in TraceMemory.

6.34.2.8 virtual void dumpon () [inline, virtual, inherited]

resume the dump

switch driver signals waveform dump on

See also:
Driver::dumpvars
Driver::dumpfile
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.9 void dumpvars (const char * *name*) throw (std::exception) [inherited]

select driver signals to dump

Parameters:
name name of the signal to be dumped.

Note:
no signal can be added after first run.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.34.2.10 virtual void dumpvars (Signal * *signal* = NULL) [virtual, inherited]

select driver signals to dump

Parameters:
signal handler to 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.

See also:
Driver::dumpfile
Driver::dumpon
Driver::dumpoff

Reimplemented in CDriver, Monitor, TraceMemory, and MckCDriver.

6.34.2.11 unsigned int error () const throw (std::exception)

Returns the number of wrong patterns detected.

6.34.2.12 Signal* getSignal (const char * *name*) const throw (std::exception) [inherited]

get a signal handler

Parameters:

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

6.34.2.13 virtual bool isTraceMemoryDriver () throw (std::exception) [virtual, inherited]

returns true if the driver is a trace driver

Reimplemented in TraceMemory.

6.34.2.14 unsigned int loadFile (const char * *fileName*) throw (std::exception)

Load a pattern file.

Parameters:

fileName name of pattern file. The specified file is a .bin file dumped during a previous emulation.

See also:

Driver::dumpfile

6.34.2.15 const char* name () const throw (std::exception) [inherited]

get the driver's name

Returns:

const char *

Return values:

NULL terminated C string containing driver's name

6.34.2.16 void registerCallback (void(∗)(void ∗callback), void ∗ *user*) throw (std::exception) [inherited]

register a callback

Parameters:

callback callback

user user data

6.34.2.17 virtual unsigned int run (unsigned int *nbPattern*) const throw (std::exception) [virtual]

Send a specified number of patterns.

Parameters:

nbPattern number of pattern to send.

block blocking run or not

Returns:

int

Return values:

0 OK

>***0*** KO

Implements Driver.

6.34.2.18 virtual unsigned int run (unsigned int *nbPattern*, bool *block*) const throw (std::exception) [virtual]

obsolete

Implements Driver.

6.34.2.19 void runAll () throw (std::exception)

Send all the patterns. Send all the patterns contained in the file to the board, and compares the received patterns with the references.

6.34.2.20 void setParam (unsigned int *nbPrintError*, unsigned int *firstCheck*, unsigned int *lastCheck*) throw (std::exception)

set optionnal parameters.

Parameters:

nbPrintError maximum number of errors to display.

firstCheck begin to check values after this cycle.

lastCheck finish to check values after tis cycle.

See also:

Driver::dumpfile

6.34.2.21 virtual void setPreTriggerRatio (float *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:

no effect on other drivers

Reimplemented in TraceMemory.

6.34.2.22 virtual void setPreTriggerSize (unsigned int *size*) throw (std::exception) `[virtual, inherited]`

set the preTrigger memory size of the trace memory

Note:

no effect on other drivers

Reimplemented in TraceMemory.

6.34.2.23 virtual void storeToFile () throw (std::exception) `[virtual, inherited]`

command the download and the dump of the trace memory

Note:

no effect on other drivers

Reimplemented in TraceMemory.

6.34.2.24 virtual void update () const throw (std::exception) `[virtual, inherited]`

update IOs

equivalent to `run(0)`

Returns:
void

Reimplemented in MckCDriver.

6.34.2.25 virtual unsigned int wait (unsigned int *triggers*, unsigned int *timeOut* = `0xffffffff`) const throw (std::exception) `[virtual, inherited]`

wait for a trigger event or timeout while running the clock

Parameters:

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 stopping.

Returns:
unsigned int

Return values:

0 if a timeout occurs

bit i set to 1 for trigger i

Reimplemented in CDriver, and MckCDriver.

6.35 Port Class Reference

Inheritance diagram for Port:

6.35.1 Detailed Description

interface for ZeBu port.

You use Port class to allow communication between hardware side and software side in transaction based co-simulation. Port class is an abstract class. You use a TxPort to send data to HW, and RxPort to receive data from HW.

Public Member Functions

- virtual unsigned int connect (Board *board, const char *driverName)=0 throw (std::exception)

  connect port to ZeBu

- void disconnect () throw (std::exception)

  disconnect

- virtual bool isPossibleToReceive () const throw (std::exception)

  return true if port can receive data

- virtual bool isPossibleToSend () const throw (std::exception)

  returns true if port can send data

- virtual unsigned int * receiveMessage () throw (std::exception)

  return buffer with last read data

- virtual void sendMessage () throw (std::exception)

  send data to hardware side

- unsigned int size () const throw (std::exception)

  get message size in 32 bit word

- unsigned int * message () const throw (std::exception)

  get message handler

- virtual unsigned int read (unsigned int index) const throw (std::exception)

  get a word in read message

- virtual void write (unsigned int index, unsigned int value) throw (std::exception)

  write a word in sent message

- void registerCB (void(∗cb)(void ∗), void ∗user) throw (std::exception)

  register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

- virtual void waitToReceive () const throw (std::exception)

  wait to receive data

- virtual void waitToSend () const throw (std::exception)

  wait to send data

- virtual void setGroup (const unsigned int groupNumber)=0 throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

- virtual void flush () const throw (std::exception)

  flush messages to send

- virtual unsigned long long date () const throw (std::exception)

  returns date of last message.

- virtual void setGroup (const long long unsigned int groupNumber)=0 throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

Static Public Member Functions

- void WaitGroup (const Board ∗board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port ∗port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board ∗board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port ∗port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Board ∗board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port ∗port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board ∗board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port ∗port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

6.35.2 Member Function Documentation

6.35.2.1 virtual unsigned int connect (Board ∗ *board*, const char ∗ *driverName*) throw (std::exception) `[pure virtual]`

connect port to ZeBu

Parameters:

board handler on a Board

driverName driver's name

Returns:

unsigned int

Return values:
> 0 if error

See also:
Port::disconnect

Implemented in TxPort, and RxPort.

6.35.2.2 virtual unsigned long long date () const throw (std::exception) `[virtual]`

returns date of last message.

Bug
always return 0xffffffff

6.35.2.3 void disconnect () throw (std::exception)

disconnect

See also:
Port::connect

6.35.2.4 virtual void flush () const throw (std::exception) `[virtual]`

flush messages to send

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented in TxPort.

6.35.2.5 virtual bool isPossibleToReceive () const throw (std::exception) `[virtual]`

return true if port can receive data

Returns:
bool

Return values:

true port can receive data

false port cannot receive data

See also:

Port::isPossibleToSend
Port::waitToReceive
Port::receiveMessage

6.35.2.6 virtual bool isPossibleToSend () const throw (std::exception) `[virtual]`

returns true if port can send data

Returns:

bool

Return values:

true port can send data

false port cannot send data

See also:

Port::isPossibleToReceive
Port::waitToSend
Port::sendMessage

6.35.2.7 unsigned int* message () const throw (std::exception)

get message handler

Returns:

unsigned int *

6.35.2.8 virtual unsigned int read (unsigned int *index*) const throw (std::exception) `[virtual]`

get a word in read message

Parameters:

index index of the word to read

Returns:

unsigned int

6.35.2.9 virtual unsigned int∗ receiveMessage () throw (std::exception) [virtual]

return buffer with last read data

Returns:
unsigned int ∗

Return values:
array array with data receive from hardware side

See also:
Port::isPossibleToSend
Port::isPossibleToReceive
Port::sendMessage

6.35.2.10 void registerCB (void(∗)(void ∗) *cb*, void ∗ *user*) throw (std::exception)

register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

Parameters:
cb callback function pointer
user pointer to the argument of the callback

See also:
Board::serviceLoop

6.35.2.11 virtual void sendMessage () throw (std::exception) [virtual]

send data to hardware side

See also:
Port::isPossibleToSend
Port::isPossibleToReceive
Port::receiveMessage

6.35.2.12 virtual void setGroup (const long long unsigned int *groupNumber*) throw (std::exception) [pure virtual]

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:
groupNumber identifier of the group

See also:
Port::Wait

Implemented in TxPort, and RxPort.

6.35.2.13 virtual void setGroup (const unsigned int *groupNumber*) throw (std::exception) `[pure virtual]`

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:
groupNumber identifier of the group

See also:
Port::Wait

Implemented in TxPort, and RxPort.

6.35.2.14 unsigned int size () const throw (std::exception)

get message size in 32 bit word

Returns:
unsigned int

6.35.2.15 void WaitGroup (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = `0`) throw (std::exception) `[static]`

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.35.2.16 void WaitGroup (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:

Port::registerWaiter

6.35.2.17 void WaitGroup (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:

Port::registerWaiter

6.35.2.18 void WaitGroup (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.35.2.19 int WaitGroup2 (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:
Port::registerWaiter

6.35.2.20 int WaitGroup2 (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:
Port::registerWaiter

6.35.2.21 int WaitGroup2 (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:
Port::registerWaiter

6.35.2.22 int WaitGroup2 (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:
Port::registerWaiter

6.35.2.23 virtual void waitToReceive () const throw (std::exception) [virtual]

wait to receive data

See also:
Port::isPossibleToReceive
Port::receiveMessage

Reimplemented in RxPort.

6.35.2.24 virtual void waitToSend () const throw (std::exception) [virtual]

wait to send data

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented in TxPort.

6.35.2.25 virtual void write (unsigned int *index*, unsigned int *value*) throw (std::exception) [virtual]

write a word in sent message

Parameters:

index index of the word to write

value word value to write

Returns:
unsigned int

6.36 RxPort Class Reference

Inheritance diagram for RxPort:Collaboration diagram for RxPort:

6.36.1 Detailed Description

interface for ZeBu receive port.

You use RxPort class to allow communication between hardware side and software side in transaction based co-simulation. RxPort allow you to receive data from HW.

See also:
Port
TxPort

Public Member Functions

- RxPort (const char ∗name, unsigned int size=0) throw (std::exception)

  constructor

  ```
  RxPort txp("rxp");
  ```

- virtual ∼RxPort () throw (std::exception)

  destructor

- unsigned int connect (Board ∗board, const char ∗driverName) throw (std::exception)

  connect port to ZeBu

- virtual void waitToReceive () const throw (std::exception)

  wait to receive data

- virtual void setGroup (const unsigned int groupNumber) throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

- virtual void setGroup (const long long unsigned int groupNumber) throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

- void disconnect () throw (std::exception)

  disconnect

- virtual bool isPossibleToReceive () const throw (std::exception)

  return true if port can receive data

- virtual bool isPossibleToSend () const throw (std::exception)

  returns true if port can send data

- virtual unsigned int * receiveMessage () throw (std::exception)

  return buffer with last read data

- virtual void sendMessage () throw (std::exception)

  send data to hardware side

- unsigned int size () const throw (std::exception)

  get message size in 32 bit word

- unsigned int * message () const throw (std::exception)

  get message handler

- virtual unsigned int read (unsigned int index) const throw (std::exception)

  get a word in read message

- virtual void write (unsigned int index, unsigned int value) throw (std::exception)

  write a word in sent message

- void registerCB (void(*cb)(void *), void *user) throw (std::exception)

  register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

- virtual void waitToSend () const throw (std::exception)

  wait to send data

- virtual void flush () const throw (std::exception)

  flush messages to send

- virtual unsigned long long date () const throw (std::exception)

  returns date of last message.

Static Public Member Functions

- void WaitGroup (const Board *board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port *port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Board *board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port *port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board *board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port *port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board *board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port *port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 RxPort (const char * *name*, unsigned int *size* = 0) throw (std::exception)

constructor

```
RxPort txp("rxp");
```

6.36.2.2 virtual ~RxPort () throw (std::exception) [virtual]

destructor

6.36.3 Member Function Documentation

6.36.3.1 unsigned int connect (Board * *board*, const char * *driverName*) throw (std::exception) [virtual]

connect port to ZeBu

Parameters:

board handler on a Board

driverName driver's name

Returns:

unsigned int

Return values:

> 0 if error

See also:

Port::disconnect

```
rxp.connect("xtor");
```

Implements Port.

6.36.3.2 virtual unsigned long long date () const throw (std::exception) [virtual, inherited]

returns date of last message.

Bug

always return 0xffffffff

6.36.3.3 void disconnect () throw (std::exception) [inherited]

disconnect

See also:
Port::connect

6.36.3.4 virtual void flush () const throw (std::exception) [virtual, inherited]

flush messages to send

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented in TxPort.

6.36.3.5 virtual bool isPossibleToReceive () const throw (std::exception) [virtual, inherited]

return true if port can receive data

Returns:
bool

Return values:
true port can receive data
false port cannot receive data

See also:
Port::isPossibleToSend
Port::waitToReceive
Port::receiveMessage

6.36.3.6 virtual bool isPossibleToSend () const throw (std::exception) [virtual, inherited]

returns true if port can send data

Returns:
bool

Return values:

true port can send data

false port cannot send data

See also:

Port::isPossibleToReceive
Port::waitToSend
Port::sendMessage

6.36.3.7 unsigned int* message () const throw (std::exception) [inherited]

get message handler

Returns:

unsigned int *

6.36.3.8 virtual unsigned int read (unsigned int *index*) const throw (std::exception) [virtual, inherited]

get a word in read message

Parameters:

index index of the word to read

Returns:

unsigned int

6.36.3.9 virtual unsigned int* receiveMessage () throw (std::exception) [virtual, inherited]

return buffer with last read data

Returns:

unsigned int *

Return values:

array array with data receive from hardware side

See also:

Port::isPossibleToSend
Port::isPossibleToReceive
Port::sendMessage

6.36.3.10 void registerCB (void(∗)(void ∗) *cb*, void ∗ *user*) throw (std::exception) [inherited]

register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

Parameters:

cb callback function pointer

user pointer to the argument of the callback

See also:

Board::serviceLoop

6.36.3.11 virtual void sendMessage () throw (std::exception) [virtual, inherited]

send data to hardware side

See also:

Port::isPossibleToSend
Port::isPossibleToReceive
Port::receiveMessage

6.36.3.12 virtual void setGroup (const long long unsigned int *groupNumber*) throw (std::exception) [virtual]

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:

groupNumber identifier of the group, [-10:0] are forbidden

See also:

Port::Wait

Implements Port.

6.36.3.13 virtual void setGroup (const unsigned int *groupNumber*) throw (std::exception) [virtual]

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:

groupNumber identifier of the group, [-10:0] are forbidden

See also:

Port::Wait

Implements Port.

6.36.3.14 unsigned int size () const throw (std::exception) [inherited]

get message size in 32 bit word

Returns:

unsigned int

6.36.3.15 void WaitGroup (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:

Port::registerWaiter

6.36.3.16 void WaitGroup (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.36.3.17 void WaitGroup (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.36.3.18 void WaitGroup (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.36.3.19 int WaitGroup2 (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.36.3.20 int WaitGroup2 (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.36.3.21 int WaitGroup2 (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.36.3.22 int WaitGroup2 (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.36.3.23 virtual void waitToReceive () const throw (std::exception) [virtual]

wait to receive data

See also:
Port::isPossibleToReceive
Port::receiveMessage

Reimplemented from Port.

6.36.3.24 virtual void waitToSend () const throw (std::exception) [virtual, inherited]

wait to send data

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented in TxPort.

6.36.3.25 virtual void write (unsigned int *index*, unsigned int *value*) throw (std::exception) [virtual, inherited]

write a word in sent message

Parameters:

index index of the word to write

value word value to write

Returns:
unsigned int

6.37 Signal Class Reference

Inheritance diagram for Signal:

6.37.1 Detailed Description

Implement public interface class for signals.

Note:
Not supported in zTide environment.

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

See also:
Board::getSignal
Driver::getSignal

Public Member Functions

- ~Signal () throw (std::exception)

  destructor

- Signal & operator= (const Signal &signal) throw (std::exception)

  set the value of the signal

  ```
  vector1 = vector2;
  ```

- Signal & operator= (int value) throw (std::exception)

  set the value of the signal

  ```
  vector1 = 0;
  signal1 = 1;
  ```

- Signal & operator= (unsigned int value) throw (std::exception)

  set the signal to a value

  ```
  vector1 = 0U;
  signal1 = 1U;
  ```

- Signal & operator= (unsigned long value) throw (std::exception)

  set the signal to a value

  ```
  vector1 = 0U;
  signal1 = 1U;
  ```

- Signal & operator= (const char *str) throw (std::exception)

set the signal to a value

```
vector1 = "0xfeed"; // hexa
vector2 = "076543210"; // octal
vector2 = "0b010101010101010" // binary
```

- operator unsigned int () throw (std::exception)

  get the signal to a value

  ```
  unsigned int value = signal1;
  ```

- Signal & operator= (unsigned int *value) throw (std::exception)

  assign the signal to a value

- Signal & operator= (unsigned long *value) throw (std::exception)

  set the signal to a value

- Signal & operator[ ] (unsigned int bit) const throw (std::exception)

  get the value of a particular vector's bit

- void set (unsigned int index, unsigned int value) throw (std::exception)

  set the index-th word of the vector's value

- unsigned int get (unsigned int index) const throw (std::exception)

  get the index-th word of the vector's value

- unsigned int operator() () const throw (std::exception)

  get the value of the signal

- char * fetchValue (const char *format, ZEBU_Value *value=NULL) const throw (std::exception)

  get the value of the signal

- char * resolveValue (const char *format, ZEBU_Value *value=NULL) const throw (std::exception)

  get the software resolved value of a bi-directionnal signal

- int setValue (ZEBU_Value *value) throw (std::exception)

  set the value on a signal

- const char * name () const throw (std::exception)

  get the signal's name

- const char * fullname (char separator= '.') const throw (std::exception)

  get the signal's hierarchical name

- unsigned int size () const throw (std::exception)

  get the signal size

- unsigned int getLsbIndex () const throw (std::exception)

  get the lsb index

- bool isInternal () const throw (std::exception)

  test if the the signal is an internal signal

- bool isWritable () const throw (std::exception)

  test if the signal is writable

- bool isSelected () const throw (std::exception)

  test if the signal is selected

Static Public Member Functions

- bool IsForceable (const Board ∗board, const Signal ∗signal) throw (std::exception)

  test if a signal can be assigned for an indeterminate time

- bool IsForceable (const Board ∗board, const char ∗signalFullname) throw (std::exception)

  test if a signal can be assigned for an indeterminate time

- void Force (Board ∗board, Signal ∗signal, const unsigned int ∗value) throw (std::exception)

  assign a signal for an indeterminate time Board::writeRegisters has to be called after to apply modification

- void Force (Board ∗board, const char ∗signalFullname, const unsigned int ∗value) throw (std::exception)

  assign a signal for an indeterminate time Board::writeRegisters has to be called after to apply modification

- void Release (Board ∗board, Signal ∗signal) throw (std::exception)

  desassign a signal Board::writeRegisters has to be called after to apply modification

- void Release (Board ∗board, const char ∗signalFullname) throw (std::exception)

  desassign a signal Board::writeRegisters has to be called after to apply modification

Friends

- ostream & **operator**<< (ostream &output, Signal &signal) throw (std::exception)
- bool operator== (Signal &a, Signal &b) throw (std::exception)

  compare 2 signal values

- bool operator== (Signal &a, unsigned int i) throw (std::exception)

  compare a signal and an unsigned int

- bool operator== (Signal &a, int i) throw (std::exception)

  compare a signal and an int

- bool operator== (unsigned int i, Signal &a) throw (std::exception)

  compare an unsigned int and a signal

- bool operator== (int i, Signal &a) throw (std::exception)

  compare an int and a signal

- bool operator!= (Signal &a, Signal &b) throw (std::exception)

  compare 2 signals

- bool operator!= (Signal &a, unsigned int i) throw (std::exception)

  compare a signal and an unsigned int

- bool operator!= (unsigned int i, Signal &a) throw (std::exception)

  compare an unsigned int and a signal

- bool operator!= (Signal &a, int i) throw (std::exception)

  compare an int and a signal

- bool operator!= (int i, Signal &a) throw (std::exception)

  compare an int and a signal

6.37.2 Constructor & Destructor Documentation

6.37.2.1 ~Signal () throw (std::exception)

destructor

6.37.3 Member Function Documentation

6.37.3.1 char* fetchValue (const char * *format*, ZEBU_Value * *value* = NULL) const throw (std::exception)

get the value of the signal

Parameters:

format a literal string or a character string pointer with one of the following specifiers for formatting the return value: "%b" "%d" "%h" "%o" "%v" "%%"

value optional. Pointer to a structure with retrieved logic values and strength; used when format string is "%%"

Returns:

char* ointer to a character string. The returned value has to be duplicatedd by the user before a new call to this method.

See also:

setValue
ZEBU_Value

The method fetchValue shall return the logic and strength values in one of two ways :

- the value can be returned as a string
- the value can be returned as an aval/bval pair in predefined structure

The return value format shall be controlled by the format argument as shown below :

format specifier	return format	description
"%b"	binary	Value retrieved as a string, and a character pointer to the string shall be returned
"%d"	decimal	Not implemented
"%h"	hexadecimal	Value retrieved as a string, and a character pointer to the string shall be returned
"%o"	octal	Value retrieved as a string, and a character pointer to the string shall be returned
"%v"	strength	Not implemented
"%%"	aval/bval pair	Value retrieved and placed in a structure variable pointed to by the value argument

The use of the "%%" format argument to retrieve values to a structure requires the following steps:

- a structure variable shall be first declared of type ZEBU_Value
- the format field has to be set to a predefined constant. The format controls which field in the ZEBU_Value structure will be used when fetchValue returns the value.
- the structure variable has to be passed as the second argument to fetchValue

Note:
When using aval/bval pairs, the ZEBU_Value record and the appropriately sized ZEBU_vecval array must be allocated first
Setting `format` argument to "%%' and `value` argument to `NULL` is an error.

Bug
Decimal format does not work with vectors greater than 64 bits

6.37.3.2 void Force (Board * *board*, const char * *signalFullname*, const unsigned int * *value*) throw (std::exception) [static]

assign a signal for an indeterminate time Board::writeRegisters has to be called after to apply modification

Parameters:

board C++ handler on Board

signalFullname hierarchical name of the signal to force

value value to assign

6.37.3.3 void Force (Board * *board*, Signal * *signal*, const unsigned int * *value*) throw (std::exception) [static]

assign a signal for an indeterminate time Board::writeRegisters has to be called after to apply modification

Parameters:

board C++ handler on Board

signal signal to force

value value to assign

6.37.3.4 const char* fullname (char *separator* = '.') const throw (std::exception)

get the signal's hierarchical name

Parameters:
separator hierarchical sperarator character to use

Returns:
const char* character pointer to a string containing the signal's name

```
cout << "fullname : " << signal1.fullname() << endl;
```

6.37.3.5 unsigned int get (unsigned int *index*) const throw (std::exception)

get the index-th word of the vector's value

Parameters:
index

Returns:
unsigned int

See also:
Signal::set

```
unsigned int nbWord = (signal.size() - 1)/32 + 1;
unsigned int index;
for(index = 0; index < nbWord; ++index) {
    cout << "word " << index << " = " << signal.get(index) << endl;
}
```

6.37.3.6 unsigned int getLsbIndex () const throw (std::exception)

get the lsb index

Returns:
unsigned int

Return values:
bit index

```
unsigned int i = signal1.getLsbIndex();
```

6.37.3.7 bool IsForceable (const Board * *board*, const char * *signalFullname*) throw (std::exception) [static]

test if a signal can be assigned for an indeterminate time

Parameters:

board C++ handler on Board

signalFullname hierarchical name of the signal to test

Returns:

bool

Return values:

true if forceable

6.37.3.8 bool IsForceable (const Board * *board*, const Signal * *signal*) throw (std::exception) [static]

test if a signal can be assigned for an indeterminate time

Parameters:

board C++ handler on Board

signal signal to test

Returns:

bool

Return values:

true if forceable

6.37.3.9 bool isInternal () const throw (std::exception)

test if the the signal is an internal signal

Returns:

bool

Return values:

true if the signal is internal to the DUT

false if the signal belongs to a driver

```
if (signal->isInternal()) { };
```

6.37.3.10 bool isSelected () const throw (std::exception)

test if the signal is selected

Returns:
bool

Return values:
true if signal is selected

false if signal is not selected

```
if (signal->isSelected()) { };
```

6.37.3.11 bool isWritable () const throw (std::exception)

test if the signal is writable

Returns:
bool

Return values:
true if signal is writable

false if signal is not writable

```
if (signal->isWritable()) { };
```

6.37.3.12 const char* name () const throw (std::exception)

get the signal's name

Returns:
const char* character pointer to a string containing the signal's name

```
cout << "name : " << signal1.name() << endl;
```

6.37.3.13 operator unsigned int () throw (std::exception)

get the signal to a value

```
unsigned int value = signal1;
```

6.37.3.14 unsigned int operator() () const throw (std::exception)

get the value of the signal

Returns:
unsigned int

Return values:
signal's value

6.37.3.15 Signal& operator= (unsigned long * *value*) throw (std::exception) [inline]

set the signal to a value

6.37.3.16 Signal& operator= (unsigned int * *value*) throw (std::exception)

assign the signal to a value

6.37.3.17 Signal& operator= (const char * *str*) throw (std::exception)

set the signal to a value

```
vector1 = "0xfeed"; // hexa
vector2 = "076543210"; // octal
vector2 = "0b0101010101010101010" // binary
```

6.37.3.18 Signal& operator= (unsigned long *value*) throw (std::exception) [inline]

set the signal to a value

```
vector1 = 0U;
signal1 = 1U;
```

6.37.3.19 Signal& operator= (unsigned int *value*) throw (std::exception)

set the signal to a value

```
vector1 = 0U;
signal1 = 1U;
```

6.37.3.20 Signal& operator= (int *value*) throw (std::exception)

set the value of the signal

```
vector1 = 0;
signal1 = 1;
```

6.37.3.21 Signal& operator= (const Signal & *signal*) throw (std::exception)

set the value of the signal

```
vector1 = vector2;
```

6.37.3.22]

Signal& operator[ ] (unsigned int *bit*) const throw (std::exception)

get the value of a particular vector's bit

Parameters:

bit index of bit

Returns:

Signal&

```
Signal &signal1_2 = vector1[2];
```

6.37.3.23 void Release (Board * *board*, const char * *signalFullname*) throw (std::exception) [static]

desassign a signal Board::writeRegisters has to be called after to apply modification

Parameters:

board C++ handler on Board

signalFullname hierarchical name of the signal to release

6.37.3.24 void Release (Board * *board*, Signal * *signal*) throw (std::exception) [static]

desassign a signal Board::writeRegisters has to be called after to apply modification

Parameters:

board C++ handler on Board

signal signal to release

6.37.3.25 char∗ resolveValue (const char ∗ *format*, ZEBU_Value ∗ *value* = NULL) const throw (std::exception)

get the software resolved value of a bi-directionnal signal

Parameters:

format a literal string or a character string pointer with one of the following specifiers for formatting the return value: ”%b” ”%d” ”%h” ”%o” ”%v” ”%%”

value optional. Pointer to a structure with retrieved logic values and strength; used when format string is ”%%”

Returns:

char∗ Pointer to a character string. The returned value has to be duplicated by the user before a new call to this method.

See also:

fetchValue

Note:

for more details about resolveValue refer to the section Signal::fectchValue of this manual which details the options

6.37.3.26 void set (unsigned int *index*, unsigned int *value*) throw (std::exception)

set the index-th word of the vector’s value

Parameters:

index index of word to set

value value to set

Returns:

unsigned int

See also:

Signal::get

```
unsigned int nbWord = (signal.size() - 1)/32 + 1;
unsigned int index;
for(index = 0; index < nbWord; ++index) {
    signal.set(index, 0);
}
```

6.37.3.27 int setValue (ZEBU_Value * *value*) throw (std::exception)

set the value on a signal

Parameters:
value pointer to a structure containing the value to be set

Returns:
int

Return values:
0 if no errors

non zero if an error occured

See also:
fetchValue
ZEBU_Value

6.37.3.28 unsigned int size () const throw (std::exception)

get the signal size

Returns:
unsigned int

Return values:
number of bits

```
unsigned int s = signal1.size();
```

6.37.4 Friends And Related Function Documentation

6.37.4.1 operator!= (int *i*, Signal & *a*) throw (std::exception) [friend]

compare an int and a signal

Returns:
bool

Return values:
true signal `i` != `a`

false signal `i` == `a`

```
if(19 != vector1) {
    cout << "vector1 is not equal to 19" << endl;
}
```

6.37.4.2 operator!= (Signal & *a*, int *i*) throw (std::exception) [friend]

compare an int and a signal

Returns:
bool

Return values:
true signal `i` != `a`

false signal `i` == `a`

```
if(19 != vector1) {
    cout << "vector1 is not equal to 19" << endl;
}
```

6.37.4.3 operator!= (unsigned int *i*, Signal & *a*) throw (std::exception) [friend]

compare an unsigned int and a signal

Returns:
bool

Return values:
true signal `i` != `a`

false signal `i` == `a`

```
if(19 != vector1) {
    cout << "vector1 is not equal to 19" << endl;
}
```

6.37.4.4 operator!= (Signal & *a*, unsigned int *i*) throw (std::exception) [friend]

compare a signal and an unsigned int

Returns:
bool

Return values:
true signal `a` != `i`

false signal `a` == `i`

```
if(vector1 != 72) {
    cout << "vector1 is not equal to 72" << endl;
}
```

6.37.4.5 operator!= (Signal & *a*, Signal & *b*) throw (std::exception) [friend]

compare 2 signals

Returns:
bool

Return values:
true a != b

false a == b

```
if(signal1 != signal2) {
    cout << "signal1 and signal2 are differents" << endl;
}
```

6.37.4.6 operator== (int *i*, Signal & *a*) throw (std::exception) [friend]

compare an int and a signal

Returns:
bool

Return values:
true signal i == a

false signal i != a

```
if(0 == signal1) {
    cout << "signal1 is equal to 0" << endl;
}
```

6.37.4.7 operator== (unsigned int *i*, Signal & *a*) throw (std::exception) [friend]

compare an unsigned int and a signal

Returns:
bool

Return values:
true signal i == a

false signal i != a

```
if(0 == signal1) {
    cout << "signal1 is equal to 0" << endl;
}
```

6.37.4.8 operator== (Signal & *a*, int *i*) throw (std::exception) [friend]

compare a signal and an int

Returns:
bool

Return values:
true signal a == i

false signal a != i

```
if(signal1 == 1) {
    cout << "signal1 is equal to 1" << endl;
}
```

6.37.4.9 operator== (Signal & *a*, unsigned int *i*) throw (std::exception) [friend]

compare a signal and an unsigned int

Returns:
bool

Return values:
true signal a == i

false signal a != i

```
if(signal1 == 1) {
    cout << "signal1 is equal to 1" << endl;
}
```

6.37.4.10 operator== (Signal & *a*, Signal & *b*) throw (std::exception) [friend]

compare 2 signal values

Returns:
bool

Return values:
true a == b

false a != b

```
if(signal1 == signal2) {
    cout << "signal1 is equal to signal2" << endl;
}
```

6.38 Sniffer Class Reference

6.38.1 Detailed Description

Allow saving repetively the state of the DUT and to intercept continuously its input stream into a sniff folder.

Note:
Not supported in zTide environment.

Static Public Member Functions

- void Initialize (Board ∗board, const char ∗foldername, const char ∗clockName)
  initialize the sniffer
- long long unsigned int Start (Board ∗board)
  start the sniffer
- long long unsigned int CreateFrame (Board ∗board)
  force the sniffer to create a new frame
- long long unsigned int Stop (Board ∗board)
  stop the sniffer
- bool DeleteFrame (Board ∗board, unsigned int frameReference)
  delete a previously created frame
- bool DeleteSavedState (Board ∗board, unsigned int frameReference)
  delete the saved state of a previously created frame preventing restart from that frame
- long long unsigned int Disable (Board ∗board)
  disable the sniffer
- long long unsigned int Enable (Board ∗board)
  enable the sniffer

6.38.2 Member Function Documentation

6.38.2.1 long long unsigned int CreateFrame (Board ∗ *board*) [static]

force the sniffer to create a new frame

Parameters:

board C++ handler on Board retval the cycle number of the reference clock at which the sniffer has created a new frame

6.38.2.2 bool DeleteFrame (Board * *board*, unsigned int *frameReference*) [static]

delete a previously created frame

Parameters:

board C++ handler on Board retval boolean status

6.38.2.3 bool DeleteSavedState (Board * *board*, unsigned int *frameReference*) [static]

delete the saved state of a previously created frame preventing restart from that frame

Parameters:

board C++ handler on Board retval boolean status

6.38.2.4 long long unsigned int Disable (Board * *board*) [static]

disable the sniffer

Parameters:

board C++ handler on Board retval the cycle number of the reference clock at which the sniffer has been disabled

6.38.2.5 long long unsigned int Enable (Board * *board*) [static]

enable the sniffer

Parameters:

board C++ handler on Board retval the cycle number of the reference clock at which the sniffer has been enabled

6.38.2.6 void Initialize (Board * *board*, const char * *foldername*, const char * *clockName*) [static]

initialize the sniffer

Parameters:

board C++ handler on Board

board ZeBu board

foldername name of the directory in which must be saved sniffed data

clockName name of a reference clock of the sniffer

6.38.2.7 long long unsigned int Start (Board * *board*) [static]

start the sniffer

Parameters:

board C++ handler on Board retval the cycle number of the reference clock at which the sniffer has been started

6.38.2.8 long long unsigned int Stop (Board * *board*) [static]

stop the sniffer

Parameters:

board C++ handler on Board retval the cycle number of the reference clock at which the sniffer has been stopped

6.39 SVA Class Reference

6.39.1 Detailed Description

Allow controlling System Verilog Assertions.

Note:
Not supported in zTide environment.

Public Types

- enum EnableType { **DISABLE** = 0x0, **ENABLE_TRIGGER** = 0x1, **ENABLE_REPORT** = 0x2 }

  Enable types.

Static Public Member Functions

- void Start (Board *board, const char *clockName, const unsigned int enableTypes=SVA::ENABLE_REPORT, const bool timeInSVASamplingClockCycles=true) throw (std::exception)

  start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

- void Start (Board *board, const char *clockName, ZEBU_SVA_Report callback, void *context, const unsigned int enableTypes=SVA::ENABLE_REPORT, const bool timeInSVASamplingClockCycles=true) throw (std::exception)

  start System Verilog Assertion Assertion messages are uncompacted and are reported through a callback during emulation

- void Start (Board *board, const char *clockName, const char *filename, const unsigned int enableTypes=SVA::ENABLE_REPORT) throw (std::exception)

  start System Verilog Assertion Assertion messages are not uncompacted and reported during emulation but are saved in a .zsva file

- void Stop (Board *board) throw (std::exception)

  stop all System Verilog Assertions

- void Set (Board *board, const unsigned int types=SVA::ENABLE_REPORT, const char *regularExpression=NULL, const bool invert=0, const bool ignoreCase=false, const char hierarchicalSeparator= '.') throw (std::exception)

  enable or disable assertions

- void SelectReport (Board *board, const unsigned int severities=ZEBU_SVA_-Failed_Display) throw (std::exception)

  select the types of message to report to enable

- void EnableClockStoppingOnFailure (Board *board, ZEBU_SVA_OnStop callback, void *context) throw (std::exception)

  enables clock stopping on SVA failure. When clocks are stopped, the user callback specified is called

- void DisableClockStoppingOnFailure (Board *board) throw (std::exception)

  disables clock stopping on SVA failure.

- bool DesignHasSvaCompiled (Board *board) throw (std::exception)

  returns true if design has SVA compiled

6.39.2 Member Enumeration Documentation

6.39.2.1 enum EnableType

Enable types.

6.39.3 Member Function Documentation

6.39.3.1 bool DesignHasSvaCompiled (Board * *board*) throw (std::exception) [static]

returns true if design has SVA compiled

Parameters:
board C++ handler on Board

6.39.3.2 void DisableClockStoppingOnFailure (Board * *board*) throw (std::exception) [static]

disables clock stopping on SVA failure.

Parameters:
board C++ handler on Board

6.39.3.3 void EnableClockStoppingOnFailure (Board * *board*, ZEBU_SVA_OnStop *callback*, void * *context*) throw (std::exception) `[static]`

enables clock stopping on SVA failure. When clocks are stopped, the user callback specified is called

Parameters:

board C++ handler on Board

ZEBU_SVA_OnStop callback

callback context passed to the callback when called

6.39.3.4 void SelectReport (Board * *board*, const unsigned int *severities* = `ZEBU_SVA_Failed_Display`) throw (std::exception) `[static]`

select the types of message to report to enable

Parameters:

board C++ handler on Board

severities severities to report

6.39.3.5 void Set (Board * *board*, const unsigned int *types* = `SVA::ENABLE_REPORT`, const char * *regularExpression* = `NULL`, const bool *invert* = `0`, const bool *ignoreCase* = `false`, const char *hierarchicalSeparator* = `'.'`) throw (std::exception) `[static]`

enable or disable assertions

Parameters:

types specify actions to enable: DISABLE = disable the assertions ENABLE_TRIGGER = enable clock stop when assertions failed ENABLE_REPORT = enable assertion message report

board C++ handler on Board

regularExpression regular expression specifing the hierarchical names of assertions to enable

invert invert the sense of the regular expression

ignoreCase ignore case distinctions

hierarchicalSeparator hierarchical separator character

Example of regular expressions:

List of names A1 L0.A2 A2.L0 L0.L1.A2 L0.L1.L2.A2 L0.L1.L2.A3 L0.L1.L2.A4

Result of the regular expression = "A1" A1

Result of the regular expression = "A2" L0.A2 A2.L0 L0.L1.A2 L0.L1.L2.A2

Result of the regular expression = "A2$" L0.A2 L0.L1.A2 L0.L1.L2.A2

Result of the regular expression = "L0\\.L1\\.L2\\..*[^2]$" L0.L1.L2.A3 L0.L1.L2.A4

6.39.3.6 void Start (Board * *board*, const char * *clockName*, const char * *filename*, const unsigned int *enableTypes* = `SVA::ENABLE_REPORT`) throw (std::exception) `[static]`

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 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

6.39.3.7 void Start (Board * *board*, const char * *clockName*, ZEBU_SVA_Report *callback*, void * *context*, const unsigned int *enableTypes* = `SVA::ENABLE_REPORT`, const bool *timeInSVASamplingClockCycles* = `true`) throw (std::exception) `[static]`

start System Verilog Assertion Assertion messages are uncompacted and are reported through a callback during emulation

Parameters:

board C++ handler on 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

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 true report time in number of SVA clock cycles. Else report time in number of cycles of the reference controlled clock.

6.39.3.8 void Start (Board * *board*, const char * *clockName*, const unsigned int *enableTypes* = `SVA::ENABLE_REPORT`, const bool *timeInSVASamplingClockCycles* = `true`) throw (std::exception) `[static]`

start System Verilog Assertion Assertion messages are uncompacted and are reported on standard output during emulation

Parameters:

board C++ handler on 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 true report time in number of SVA clock cycles. Else report time in number of cycles of the reference controlled clock.

6.39.3.9 void Stop (Board * *board*) throw (std::exception) `[static]`

stop all System Verilog Assertions

6.40 TraceMemory Class Reference

Inheritance diagram for TraceMemory:Collaboration diagram for TraceMemory:

6.40.1 Detailed Description

Implement ZeBu SRAM monitor driver. TraceMemory is used for dumping VCD with SRAM.

Public Member Functions

- virtual ~TraceMemory () throw (std::exception)

  destructor

- bool isTraceMemoryDriver () throw (std::exception)

  returns true if the driver is a trace driver

- virtual void dumpfile (const char *filename, int level=0) throw (std::exception)

  specify the name of the driver waveform file

- virtual void dumpvars (Signal *signal=NULL) throw (std::exception)

  select signals of the trace driver to dump

- virtual void dumpon (char *clockName, char *edgeName="posedge") throw (std::exception)

  start trace

- virtual void dumpon () throw (std::exception)

  start trace. The clock and edge values for sampling are the previous used during the emulation.

- virtual void dumpoff () throw (std::exception)

  suspend the dump

- virtual void dumpclosefile () throw (std::exception)

  close the waveform file open from TraceMemory::dumpfile

- virtual void closeDumpfile () throw (std::exception)

  obsolete

- void storeToFile () throw (std::exception)

  command the download and the dump

- void setPreTriggerSize (unsigned int size) throw (std::exception)

  set the preTrigger memory size

- void setPreTriggerRatio (float ratio) throw (std::exception)

  set the preTrigger memory size in percent

- virtual unsigned int run (unsigned int numCycles, bool block=true) const __attribute__((deprecated)) throw (std::exception)

  obsolete

- virtual unsigned int run (unsigned int numCycles) const throw (std::exception)

  no effect

- unsigned int connect () throw (std::exception)

  connect driver

- void disconnect () throw (std::exception)

  disconnect driver

- const char * name () const throw (std::exception)

  get the driver's name

- virtual unsigned int wait (unsigned int triggers, unsigned int timeOut=0xffffffff) const throw (std::exception)

  wait for a trigger event or timeout while running the clock

- virtual void update () const throw (std::exception)

  update IOs

- void registerCallback (void(*)(void *callback), void *user) throw (std::exception)

  register a callback

- Signal * getSignal (const char *name) const throw (std::exception)

  get a signal handler

- void dumpvars (const char *name) throw (std::exception)

  select driver signals to dump

Protected Member Functions

- TraceMemory (SramMonitorAbstract *imp) throw (std::exception)

  constructor

Protected Attributes

- SramMonitorAbstract * _monitor

  pointer on the monitor implementation

6.40.2 Constructor & Destructor Documentation

6.40.2.1 virtual ~TraceMemory () throw (std::exception) [virtual]

destructor

6.40.2.2 TraceMemory (SramMonitorAbstract * *imp*) throw (std::exception) [protected]

constructor

6.40.3 Member Function Documentation

6.40.3.1 virtual void closeDumpfile () throw (std::exception) [virtual]

obsolete

Reimplemented from Driver.

6.40.3.2 unsigned int connect () throw (std::exception) [inherited]

connect driver

Returns:
unsigned int

Return values:
0 OK

>***0*** KO

See also:
Driver::disconnect

6.40.3.3 void disconnect () throw (std::exception) [inherited]

disconnect driver

See also:
Driver::connect

6.40.3.4 virtual void dumpclosefile () throw (std::exception) [virtual]

close the waveform file open from TraceMemory::dumpfile

Note:
not supported in zTide environment

Reimplemented from Driver.

6.40.3.5 virtual void dumpfile (const char * *filename*, int *level* = 0) throw (std::exception) [virtual]

specify the name of the driver waveform file

Parameters:

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 formatd

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

See also:
TraceMemory::dumpclosefile
TraceMemory::dumpvars
TraceMemory::dumpon
TraceMemory::dumpoff

Reimplemented from Driver.

6.40.3.6 virtual void dumpoff () throw (std::exception) [virtual]

suspend the dump

switch waveform dump off

switch off the SRAM trace.

Reimplemented from Driver.

6.40.3.7 virtual void dumpon () throw (std::exception) [virtual]

start trace. The clock and edge values for sampling are the previous used during the emulation.

Reimplemented from Driver.

6.40.3.8 virtual void dumpon (char * *clockName*, char * *edgeName* = "posedge") throw (std::exception) [virtual]

start trace

Parameters:

clockName name of the sample clock

edgeName name of the edge of the sample clock the value can be ”posedge” or ”negedge”, the default value is ”posedge”.

Reimplemented from Driver.

6.40.3.9 void dumpvars (const char * *name*) throw (std::exception) [inherited]

select driver signals to dump

Parameters:

name name of the signal to be dumped.

Note:

no signal can be added after first run.

See also:

Driver::dumpfile
Driver::dumpon
Driver::dumpoff

6.40.3.10 virtual void dumpvars (Signal * *signal* = NULL) throw (std::exception) [virtual]

select signals of the trace driver to dump

Parameters:

signal handler to 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.

See also:

TraceMemory::dumpfile
TraceMemory::dumpon
TraceMemory::dumpoff

Reimplemented from Driver.

6.40.3.11 Signal* getSignal (const char * *name*) const throw (std::exception) [inherited]

get a signal handler

Parameters:

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

6.40.3.12 bool isTraceMemoryDriver () throw (std::exception) [virtual]

returns true if the driver is a trace driver

Reimplemented from Driver.

6.40.3.13 const char* name () const throw (std::exception) [inherited]

get the driver's name

Returns:

const char *

Return values:

NULL terminated C string containing driver's name

6.40.3.14 void registerCallback (void(∗)(void ∗callback), void ∗ *user*) throw (std::exception) [inherited]

register a callback

Parameters:

callback callback

user user data

6.40.3.15 virtual unsigned int run (unsigned int *numCycles*) const throw (std::exception) [virtual]

no effect

Implements Driver.

6.40.3.16 virtual unsigned int run (unsigned int *numCycles*, bool *block* = true) const throw (std::exception) [virtual]

obsolete

Implements Driver.

6.40.3.17 void setPreTriggerRatio (float *ratio*) throw (std::exception) [virtual]

set the preTrigger memory size in percent

Parameters:

ratio corresponds to the ratio in percentage (0.0 to 100.0) of the full memory to be used as pre-trigger part

Note:

this cannot be set when the trace is dumping. Use dumpoff before.
if this method is not called, the defaut size of pre-trigger is 10% of the memory.

Reimplemented from Driver.

6.40.3.18 void setPreTriggerSize (unsigned int *size*) throw (std::exception) [virtual]

set the preTrigger memory size

Parameters:

size corresponds to the max number of samples to put in the pre-trigger memory part

Note:

this cannot be set when the trace is dumping. Use dumpoff before.
if this method is not called, the defaut size of pre-trigger is 10% of the memory.

Reimplemented from Driver.

6.40.3.19 void storeToFile () throw (std::exception) `[virtual]`

command the download and the dump

Reimplemented from Driver.

6.40.3.20 virtual void update () const throw (std::exception) `[virtual, inherited]`

update IOs

equivalent to `run(0)`

Returns:

void

Reimplemented in MckCDriver.

6.40.3.21 virtual unsigned int wait (unsigned int *triggers*, unsigned int *timeOut* = `0xffffffff`) const throw (std::exception) `[virtual, inherited]`

wait for a trigger event or timeout while running the clock

Parameters:

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 stopping.

Returns:

unsigned int

Return values:

0 if a timeout occurs

bit i set to 1 for trigger i

Reimplemented in CDriver, and MckCDriver.

6.40.4 Member Data Documentation

6.40.4.1 SramMonitorAbstract* _monitor [protected]

pointer on the monitor implementation

6.41 Trigger Class Reference

6.41.1 Detailed Description

Implement public interface class for triggers.

Note:
Not supported in zTide environment.

See also:
Board::getTrigger

Public Member Functions

- ~Trigger () throw (std::exception)

  destructor

- Trigger & operator= (const char *definition) throw (std::exception)

  Redifine Dynamic Trigger.

  ```
  trigger1 = "input1 == 0";
  trigger2 = "input1 == 1 && input2[7:0] == 8'hbe";
  ```

  .

- unsigned int operator| (const Trigger &trigger) const throw (std::exception)
- unsigned int operator| (unsigned int i) const throw (std::exception)
- unsigned int value () throw (std::exception)

  get the current value of the trigger.

  ```
  unsigned int val = trigger0->value();
  ```

- operator unsigned int () throw (std::exception)

  return the index of a trigger

Friends

- unsigned int operator| (unsigned int i, const Trigger &trigger) throw (std::exception)

6.41.2 Constructor & Destructor Documentation

6.41.2.1 ~Trigger () throw (std::exception)

destructor

6.41.3 Member Function Documentation

6.41.3.1 operator unsigned int () throw (std::exception)

return the index of a trigger

Returns:
unsigned int

Return values:
index of the trigger

See also:
Driver::wait

```
driver->wait(trigger0);
```

6.41.3.2 Trigger& operator= (const char * *definition*) throw (std::exception)

Redifine Dynamic Trigger.

```
trigger1 = "input1 == 0";
trigger2 = "input1 == 1 && input2[7:0] == 8'hbe";
```

.

6.41.3.3 unsigned int operator| (unsigned int *i*) const throw (std::exception)

```
driver->wait(trigger1 | trigger2);
driver->wait(trigger1 | trigger2 | trigger3);
```

See also:
Driver::wait

6.41.3.4 unsigned int operator| (const Trigger & *trigger*) const throw (std::exception)

```
driver->wait(trigger1 | trigger2);
driver->wait(trigger1 | trigger2 | trigger3);
```

See also:
Driver::wait

6.41.3.5 unsigned int value () throw (std::exception)

get the current value of the trigger.

```
unsigned int val = trigger0->value();
```

6.41.4 Friends And Related Function Documentation

6.41.4.1 unsigned int operator| (unsigned int *i*, const Trigger & *trigger*) throw (std::exception) [friend]

```
driver->wait(trigger1 | trigger2);
driver->wait(trigger1 | trigger2 | trigger3);
```

See also:
Driver::wait

6.42 TxPort Class Reference

Inheritance diagram for TxPort:Collaboration diagram for TxPort:

6.42.1 Detailed Description

interface for ZeBu transmit port.

You use TxPort class to allow communication between hardware side and software side in transaction based co-simulation. TxPort allow you to send data to HW.

See also:
Port
RxPort

Public Member Functions

- TxPort (const char ∗name, unsigned int size=0) throw (std::exception)

  constructor

  ```
  TxPort txp("txp");
  ```

- virtual ∼TxPort () throw (std::exception)

  destructor

- unsigned int connect (Board ∗board, const char ∗driverName) throw (std::exception)

  connect port to ZeBu

- virtual void waitToSend () const throw (std::exception)

  wait to send data

- virtual void flush () const throw (std::exception)

  flush messages to send

- virtual void setGroup (const unsigned int groupNumber) throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

- virtual void setGroup (const long long unsigned int groupNumber) throw (std::exception)

  set the group of the port. The group of the port must be declared to use Port::WaitGroup

- void disconnect () throw (std::exception)

  disconnect

- virtual bool isPossibleToReceive () const throw (std::exception)

  return true if port can receive data

- virtual bool isPossibleToSend () const throw (std::exception)

  returns true if port can send data

- virtual unsigned int ∗ receiveMessage () throw (std::exception)

  return buffer with last read data

- virtual void sendMessage () throw (std::exception)

  send data to hardware side

- unsigned int size () const throw (std::exception)

  get message size in 32 bit word

- unsigned int ∗ message () const throw (std::exception)

  get message handler

- virtual unsigned int read (unsigned int index) const throw (std::exception)

  get a word in read message

- virtual void write (unsigned int index, unsigned int value) throw (std::exception)

  write a word in sent message

- void registerCB (void(∗cb)(void ∗), void ∗user) throw (std::exception)

  register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

- virtual void waitToReceive () const throw (std::exception)

  wait to receive data

- virtual unsigned long long date () const throw (std::exception)

  returns date of last message.

Static Public Member Functions

- void WaitGroup (const Board *board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port *port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Board *board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup (const Port *port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board *board, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port *port, const unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Board *board, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  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 WaitGroup2 (const Port *port, const long long unsigned int groupNumber, const int timeout=0) throw (std::exception)

  wait that it was possible to receive a message or to send a message on at least one port declared in the specified group.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 TxPort (const char * *name*, unsigned int *size* = 0) throw (std::exception)

constructor

```
TxPort txp("txp");
```

6.42.2.2 virtual ~TxPort () throw (std::exception) [virtual]

destructor

6.42.3 Member Function Documentation

6.42.3.1 unsigned int connect (Board * *board*, const char * *driverName*) throw (std::exception) [virtual]

connect port to ZeBu

Parameters:

board handler on a Board

driverName driver's name

Returns:

unsigned int

Return values:

> 0 if error

See also:

Port::disconnect

```
txp.connect(board, "xtor");
```

Implements Port.

6.42.3.2 virtual unsigned long long date () const throw (std::exception) [virtual, inherited]

returns date of last message.

Bug

always return 0xffffffff

6.42.3.3 void disconnect () throw (std::exception) [inherited]

disconnect

See also:
Port::connect

6.42.3.4 virtual void flush () const throw (std::exception) [virtual]

flush messages to send

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented from Port.

6.42.3.5 virtual bool isPossibleToReceive () const throw (std::exception) [virtual, inherited]

return true if port can receive data

Returns:
bool

Return values:
true port can receive data

false port cannot receive data

See also:
Port::isPossibleToSend
Port::waitToReceive
Port::receiveMessage

6.42.3.6 virtual bool isPossibleToSend () const throw (std::exception) [virtual, inherited]

returns true if port can send data

Returns:
bool

Return values:

true port can send data

false port cannot send data

See also:

Port::isPossibleToReceive
Port::waitToSend
Port::sendMessage

6.42.3.7 unsigned int* message () const throw (std::exception) [inherited]

get message handler

Returns:

unsigned int *

6.42.3.8 virtual unsigned int read (unsigned int *index*) const throw (std::exception) [virtual, inherited]

get a word in read message

Parameters:

index index of the word to read

Returns:

unsigned int

6.42.3.9 virtual unsigned int* receiveMessage () throw (std::exception) [virtual, inherited]

return buffer with last read data

Returns:

unsigned int *

Return values:

array array with data receive from hardware side

See also:

Port::isPossibleToSend
Port::isPossibleToReceive
Port::sendMessage

6.42.3.10 void registerCB (void(∗)(void ∗) *cb*, void ∗ *user*) throw (std::exception) [inherited]

register a user callback on the port. The callback is used in Board::serviceLoop whenever is possible to receive a message or whenever it is possible to send a message.

Parameters:

cb callback function pointer

user pointer to the argument of the callback

See also:

Board::serviceLoop

6.42.3.11 virtual void sendMessage () throw (std::exception) [virtual, inherited]

send data to hardware side

See also:

Port::isPossibleToSend
Port::isPossibleToReceive
Port::receiveMessage

6.42.3.12 virtual void setGroup (const long long unsigned int *groupNumber*) throw (std::exception) [virtual]

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:

groupNumber identifier of the group, [-10:0] are forbidden

See also:

Port::Wait

Implements Port.

6.42.3.13 virtual void setGroup (const unsigned int *groupNumber*) throw (std::exception) [virtual]

set the group of the port. The group of the port must be declared to use Port::WaitGroup

Parameters:

groupNumber identifier of the group, [-10:0] are forbidden

See also:

Port::Wait

Implements Port.

6.42.3.14 unsigned int size () const throw (std::exception) [inherited]

get message size in 32 bit word

Returns:

unsigned int

6.42.3.15 void WaitGroup (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:

Port::registerWaiter

6.42.3.16 void WaitGroup (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.42.3.17 void WaitGroup (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.42.3.18 void WaitGroup (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

See also:
Port::registerWaiter

6.42.3.19 int WaitGroup2 (const Port * *port*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.42.3.20 int WaitGroup2 (const Board * *board*, const long long unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.42.3.21 int WaitGroup2 (const Port * *port*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 Port

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.42.3.22 int WaitGroup2 (const Board * *board*, const unsigned int *groupNumber*, const int *timeout* = 0) throw (std::exception) [static, inherited]

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 on a Board

groupNumber identifier of the group

timeout timeout in micro-seconds. If = 0 the default value is used. If = -1 no timeout is used

Return values:

0 if no error

-1 if timeout expired

See also:

Port::registerWaiter

6.42.3.23 virtual void waitToReceive () const throw (std::exception) [virtual, inherited]

wait to receive data

See also:
Port::isPossibleToReceive
Port::receiveMessage

Reimplemented in RxPort.

6.42.3.24 virtual void waitToSend () const throw (std::exception) [virtual]

wait to send data

See also:
Port::isPossibleToSend
Port::sendMessage

Reimplemented from Port.

6.42.3.25 virtual void write (unsigned int *index*, unsigned int *value*) throw (std::exception) [virtual, inherited]

write a word in sent message

Parameters:

index index of the word to write

value word value to write

Returns:
unsigned int

6.43 ValueChange Class Reference

6.43.1 Detailed Description

This class provides several methods to work with the value change trigger.

Static Public Member Functions

- std::vector< void * > enable (Board *board, const char *signalName=0, ZEBU_vcEdge edge=ZEBU_vcEdgeAny) throw (std::exception)

  Enables a signal of the value change trigger by its name.

- void disable (Board *board, const char *signalName=0, ZEBU_vcEdge edge=ZEBU_vcEdgeAny) throw (std::exception)

  Disables a signal of the value change trigger by its name.

- std::vector< void * > enable (Board *board, Signal *signal, ZEBU_vcEdge edge=ZEBU_vcEdgeAny) throw (std::exception)

  Enables a signal of the value change trigger.

- void disable (Board *board, Signal *signal, ZEBU_vcEdge edge=ZEBU_vcEdgeAny) throw (std::exception)

  Disables a signal of the value change trigger.

- int waitDriver (Board *board, Driver *driver, const unsigned int timeout=0xffffffff) throw (std::exception)

  Waits for a value change trigger event -or- timeout while running the clock through a cosimulation-driver.

- int waitDriver (Board *board, Driver *driver, unsigned int triggers, unsigned int &fired, const unsigned int timeout=0xffffffff) throw (std::exception)

  Waits for a value change trigger event -or- another trigger event -or- timeout while running the clock through a cosimulation-driver.

- int waitLogicAnalyzer (Board *board, const char *clockName, const char *edgeName, const unsigned int timeout=0xffffffff) throw (std::exception)

  Waits for a value change trigger event -or- a timeout through a logic analyzer.

- int waitLogicAnalyzer (Board *board, const char *clockName, const char *edgeName, unsigned int triggers, unsigned int &fired, const unsigned int timeout=0xffffffff) throw (std::exception)

  Waits for a value change trigger event -or- another trigger event -or- a timeout through a logic analyzer.

6.43.2 Member Function Documentation

6.43.2.1 void disable (Board * *board*, Signal * *signal*, ZEBU_vcEdge *edge* = ZEBU_vcEdgeAny) throw (std::exception) [static]

Disables a signal of the value change trigger.

Parameters:

board Handle to a ZEBU::Board

signal The signal to disable.

edge Edge on which the trigger should fire.

6.43.2.2 void disable (Board * *board*, const char * *signalName* = 0, ZEBU_vcEdge *edge* = ZEBU_vcEdgeAny) throw (std::exception) [static]

Disables a signal of the value change trigger by its name.

Parameters:

board Handle to a ZEBU::Board.

signalName Hierarchical name of the signal to disable. If NULL, all signals are disabled in the value change trigger.

edge Edge on which the trigger should fire.

6.43.2.3 std::vector<void*> enable (Board * *board*, Signal * *signal*, ZEBU_vcEdge *edge* = ZEBU_vcEdgeAny) throw (std::exception) [static]

Enables a signal of the value change trigger.

Parameters:

board Handle to a ZEBU::Board

signal The signal to enable.

edge Edge on which the trigger should fire.

6.43.2.4 std::vector<void*> enable (Board * *board*, const char * *signalName* = 0, ZEBU_vcEdge *edge* = ZEBU_vcEdgeAny) throw (std::exception) [static]

Enables a signal of the value change trigger by its name.

Parameters:

board Handle to a ZEBU::Board.

signalName Hierarchical name of the signal to enable. If NULL, all signals are enabled in the value change trigger.

edge Edge on which the trigger should fire.

6.43.2.5 int waitDriver (Board * *board*, Driver * *driver*, unsigned int *triggers*, unsigned int & *fired*, const unsigned int *timeout* = `0xffffffff`) throw (std::exception) `[static]`

Waits for a value change trigger event -or- another trigger event -or- timeout while running the clock through a cosimulation-driver.

Parameters:

board A pointer to the currently opened 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.

6.43.2.6 int waitDriver (Board * *board*, Driver * *driver*, const unsigned int *timeout* = `0xffffffff`) throw (std::exception) `[static]`

Waits for a value change trigger event -or- timeout while running the clock through a cosimulation-driver.

Parameters:

board A pointer to the currently opened 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

6.43.2.7 int waitLogicAnalyzer (Board * *board*, const char * *clockName*, const char * *edgeName*, unsigned int *triggers*, unsigned int & *fired*, const unsigned int *timeout* = `0xffffffff`) throw (std::exception) `[static]`

Waits for a value change trigger event -or- another trigger event -or- a timeout through a logic analyzer.

Parameters:

board A pointer to the currently opened 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.

6.43.2.8 int waitLogicAnalyzer (Board * *board*, const char * *clockName*, const char * *edgeName*, const unsigned int *timeout* = `0xffffffff`) throw (std::exception) `[static]`

Waits for a value change trigger event -or- a timeout through a logic analyzer.

Parameters:

board A pointer to the currently opened 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

6.44 ValueChange::Iterator Class Reference

6.44.1 Detailed Description

Implement public iterator on value change.

```
ValueChange::Iterator vcIterator;
vcIterator.initialize(zebu);
for (vcIterator.goToFirst(); !vcIterator.isAtEnd(); vcIterator.goToNext()) {
    printf("Signal name = %s\n", vcIterator.getName());
}
```

Public Member Functions

- Iterator () throw (std::exception)

  Constructor.

- ~Iterator () throw (std::exception)

  Destructor.

- void initialize (Board *board) throw (std::exception)

  Initialize the iterator.

- void goToFirst () throw (std::exception)

  Move iterator to first signal.

- void goToNext () throw (std::exception)

  Move iterator to next signal.

- bool isAtEnd () const throw (std::exception)

  Test if iterator passed last signal.

- const char * getName () const throw (std::exception)

  Return the gate-level hierarchical name of the current changed signal.

- const void * getHandle () const throw (std::exception)

  Return the handle of the current changed signal. This value must match one of those returned by `ZEBU::ValueChange::enable` *method.*

6.44.2 Constructor & Destructor Documentation

6.44.2.1 Iterator () throw (std::exception)

Constructor.

6.44.2.2 ~Iterator () throw (std::exception)

Destructor.

6.44.3 Member Function Documentation

6.44.3.1 const void* getHandle () const throw (std::exception)

Return the handle of the current changed signal. This value must match one of those returned by `ZEBU::ValueChange::enable` method.

6.44.3.2 const char* getName () const throw (std::exception)

Return the gate-level hierarchical name of the current changed signal.

6.44.3.3 void goToFirst () throw (std::exception)

Move iterator to first signal.

6.44.3.4 void goToNext () throw (std::exception)

Move iterator to next signal.

6.44.3.5 void initialize (Board * *board*) throw (std::exception)

Initialize the iterator.

Parameters:

board A pointer to the currently opened board.

6.44.3.6 bool isAtEnd () const throw (std::exception)

Test if iterator passed last signal.

Return values:

true if at end.

6.45 WaveFile Class Reference

6.45.1 Detailed Description

Implement public interface class for readback waveform dump.

Note:
supported only in thread safe version
not supported with zRun
not supported in zTide environment

Public Member Functions

- WaveFile (Board *board, const char *filename, const char *clockName, int level=0) throw (std::exception)

  constructor: open a waveform file

- ~WaveFile () throw (std::exception)

  destructor: close the waveform file open from the constructor

- void dumpvars (Signal *signal=0) throw (std::exception)

  select internal register to dump

- void dumpvars (const char *name, const int depth) throw (std::exception)

  select internal register to dump

- void dumpon () throw (std::exception)

  resume the dump

- void dumpoff () throw (std::exception)

  suspend the dump

- void flush () throw (std::exception)

  fush the content of the waveform file to the disk

- void close () throw (std::exception)

  close the waveform file open from the constructor

6.45.2 Constructor & Destructor Documentation

6.45.2.1 WaveFile (Board * *board*, const char * *filename*, const char * *clockName*, int *level* = 0) throw (std::exception)

constructor: 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 on Board returned by Board::open

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.

sa Board::open

6.45.2.2 ~WaveFile () throw (std::exception)

destructor: close the waveform file open from the constructor

Note:
not supported in zTide environment

6.45.3 Member Function Documentation

6.45.3.1 void close () throw (std::exception)

close the waveform file open from the constructor

6.45.3.2 void dumpoff () throw (std::exception)

suspend the dump

Note:
not supported in zTide environment

switch partial readback waveform dump off. This is default.

See also:
WaveFile::dumpvars
WaveFile::dumpon
WaveFile::dumpfile

6.45.3.3 void dumpon () throw (std::exception)

resume the dump

Note:
not supported in zTide environment

switch partial readback waveform dump on

See also:
WaveFile::dumpvars
WaveFile::dumpfile
WaveFile::dumpoff

6.45.3.4 void dumpvars (const char * *name*, const int *depth*) throw (std::exception)

select internal register to dump

Note:
not supported in zTide environment

Parameters:
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.

See also:
WaveFile::dumpfile
WaveFile::dumpon
WaveFile::dumpoff

6.45.3.5 void dumpvars (Signal * *signal* = 0) throw (std::exception)

select internal register to dump

Parameters:
signal handler to 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.

See also:
WaveFile::dumpfile
WaveFile::dumpon
WaveFile::dumpoff

6.45.3.6 void flush () throw (std::exception)

fush the content of the waveform file to the disk

6.46 ZEBU_Value Struct Reference

Collaboration diagram for ZEBU_Value:

6.46.1 Detailed Description

The format constants for the ZEBU_Value structure are defined below:

Format constant	Union field used
z_BinStrVal	str
z_OctStrVal	str
z_DecStrVal	str
z_HexStrVal	str
z_ScalarVal	scalar
z_IntVal	integer
z_RealVal	real
z_StringVal	str
z_VectorVal	vector

the array of ZEBU_vecval structures must contain a record for every 32 bits of the Signal, plus a record for any remaining bits. If a vector has n bits, then there must be ((n-1)/32+1) ZEBU_vecval records. The method size() of Signal can be used to retrieve the value of n.

Public Attributes

- int format
- union {
  char * str
  int scalar
  int integer
  double real
  ZEBU_vecval * vector
  } value

6.46.2 Member Data Documentation

6.46.2.1 int format

format of the value

6.46.2.2 int integer

value word

6.46.2.3 double real

real value word

6.46.2.4 int scalar

value bit

6.46.2.5 char* str

value string

6.46.2.6 union { ... } value

value of a signal

6.46.2.7 struct ZEBU_vecval* vector

vector value

6.47 ZEBU_vecval Struct Reference

6.47.1 Detailed Description

encoding of bits in ZEBU_vecval:

aval	bval	value
0	0	0
1	0	1
0	1	Z
1	1	X

Public Attributes

- int aval
- int bval

6.47.2 Member Data Documentation

6.47.2.1 int aval

4-state logic value

6.47.2.2 int bval

4-state logic value

Chapter 7

ZeBu C++ API Page Documentation

7.1 Deprecated List

Member close(int code, const char *string=0) You should use other close method.

7.2 Bug List

Member date() const always return 0xffffffff

Member fetchValue(const char *format, ZEBU_Value *value=NULL) const
Decimal format does not work with vectors greater than 64 bits

Index