eCMD Perl Module Version Development Reference Manual

Generated by Doxygen 1.3.5

Mon Oct 10 14:29:19 2005

Contents

1 eCMD Perl Module Version Development Main Page **1**

1.1 eCMD Perl API 1

1.2 Related Information 1

1.3 Perl Version 2

1.4 Using eCMD Perl Extensions 3

1.5 Data Passing 3

1.6 Chip/Object targeting 4

1.7 Using Lists 4

1.8 Using Vectors 5

1.9 Perl API Usage 6

1.10 Example Perl Script 6

2 eCMD Perl Module Version Development Hierarchical Index **7**

2.1 eCMD Perl Module Version Development Class Hierarchy 7

3 eCMD Perl Module Version Development Class Index **9**

3.1 eCMD Perl Module Version Development Class List 9

4 eCMD Perl Module Version Development File Index **11**

4.1 eCMD Perl Module Version Development File List 11

5 eCMD Perl Module Version Development Class Documentation **13**

5.1 ecmdArrayData Struct Reference 13

5.2 ecmdArrayEntry Struct Reference 15

5.3 ecmdCageData Struct Reference 16

5.4 ecmdChipData Struct Reference 17

5.5 ecmdChipTarget Struct Reference 20

5.6 ecmdCoreData Struct Reference 23

5.7 ecmdDataBuffer Class Reference 24

5.8 ecmdDataBufferImplementationHelper Class Reference 68

5.9 ecmdDllInfo Struct Reference 69
5.10 ecmdIndexEntry Struct Reference 71
5.11 ecmdIndexVectorEntry Struct Reference 72
5.12 ecmdLatchData Struct Reference 73
5.13 ecmdLatchEntry Struct Reference 75
5.14 ecmdLooperData Struct Reference 77
5.15 ecmdMemoryEntry Struct Reference 80
5.16 ecmdNameEntry Struct Reference 81
5.17 ecmdNameVectorEntry Struct Reference 82
5.18 ecmdNodeData Struct Reference 83
5.19 ecmdOptimizableDataBuffer Class Reference 84
5.20 ecmdProcRegisterInfo Struct Reference 85
5.21 ecmdQueryData Struct Reference 86
5.22 ecmdRingData Struct Reference 87
5.23 ecmdScomData Struct Reference 89
5.24 ecmdSimModelInfo Struct Reference 90
5.25 ecmdSlotData Struct Reference 91
5.26 ecmdSpyData Struct Reference 92
5.27 ecmdSpyGroupData Struct Reference 94
5.28 ecmdThreadData Struct Reference 95
5.29 ecmdTraceArrayData Struct Reference 96

6 eCMD Perl Module Version Development File Documentation 99

6.1 cipClientPerlapi.H File Reference 99
6.2 cmdClientPerlapi.H File Reference 111
6.3 croClientPerlapi.H File Reference 113
6.4 ecmdClientPerlapi.H File Reference 124
6.5 ecmdDataBuffer.H File Reference 202
6.6 ecmdSharedUtils.H File Reference 208
6.7 ecmdStructs.H File Reference 213
6.8 ecmdUtils.H File Reference 225
6.9 gipClientPerlapi.H File Reference 229
6.10 zseClientPerlapi.H File Reference 236

Chapter 1

eCMD Perl Module Version Development Main Page

1.1 eCMD Perl API

In addition to a C/C++ API, eCMD provides a Perl API. To do this, Perl's DynaLoader module handles the loading of a special eCMD shared object and the Perl XS interface handles the function calls between eCMD C code and Perl.

The API interface is implemented as a "ecmdClientPerlapi" object. The ecmdClientPerlapi class has methods to initialize eCMD, make standard function calls.

1.2 Related Information

NOTE : Some of this documentation may look like C/C++ files/headers but that is a side effect of the tool we are using to generated the PerlAPI and documentation. Treat as Perl functions and especially look at any usage information associated with each function description

- **ecmdClientPerlapi.H**(p. 124)
- **ecmdDataBuffer**(p. 24)
- **ecmdStructs.H**(p. 213)
- **ecmdUtils.H**(p. 225)
- **ecmdSharedUtils.H**(p. 208)

1.2.1 Api Extensions

CIP (Cronus/IP) Extension

- **cipClientPerlapi.H**(p. 99)

CMD (Command Line) Extension

- **cmdClientPerlapi.H**(p. 111)

GIP (GFW IP Series) Extension

- **gipClientPerlapi.H**(p. 229)

CRO (Cronus) Extension

- **croClientPerlapi.H**(p. 113)

Z Series Extension

- **zseClientPerlapi.H**(p. 236)

1.3 Perl Version

eCMD Scripts need Perl version 5.8.1 which is installed as part of the CTE package on all supported sites. All scripts using the perl module should start with the following:

In an IP Series LAB environment

```
TBD ...
```

In a CTE only environment

```
#!/bin/ksh
#! -*- perl -*-

eval '
if [ "X$CTEPATH" = "X" ]; then echo "CTEPATH env var is not set."; exit 1; fi
export CTEPERLBIN=$CTEPATH/tools/perl/5.8.1/bin/perl;
export CTEPERLPATH=$CTEPATH/tools/perl/5.8.1;
export CTEPERLLIB=$CTEPERLPATH/lib/5.8.1:$CTEPATH/tools/ecmd/$ECMD_RELEASE/perlapi/`uname`/:$CTEPERLLIB;

exec $CTEPERLBIN -x -S $0 ${1+"$@"}
'
if 0;

use strict;
use ecmd;

# Initialize the Plugin - has to be first thing done
# Here we pass in that we know this script supports eCMD Releases 1.x if we know we support more we could pass
if (ecmdLoadDll("", "ver1")) { die "Fatal errors initializing DLL"; }
```

1.4 Using eCMD Perl Extensions

If you need functionality from an eCMD Extension that provides the Perl Api, it is initialized like below. This is a 'cip' extension example but all extensions work the same way just replace the prefix with the extension of choice.

```
# Initialize the Plugin - has to be first thing done
if (ecmdLoadDll("", "ver1")) { die "Fatal errors initializing DLL"; }

# Now initialize the extension as well - notice I use the $cp variable that points to my extension
if (cipInitExtension("ver1")) { die "Fatal errors initializing CIP Extension"; }
```

1.5 Data Passing

1.5.1 ecmdDataBuffer's

Data is passed between the client Perl script and the eCMD shared object in the form of the same **ecmdDataBuffer**(p. 24) that is part of the C-Api. The function set is the identical. For additional documentation on the **ecmdDataBuffer**(p. 24) see the C/C++ Api Documentation.

Here is an example usage of the **ecmdDataBuffer**(p. 24):

```
# Create a pointer to an DataBuffer class
my $data = new ecmd::ecmdDataBuffer();

# Set the size of my buffer
$data->$setBitLength(32);

# Set the first word of data in this class
$data->$setWord(0,0xFEEDBEEF);

# Read data from the chip
my $rc = 0;
$rc = getRing($target, "idreg", $data);

# What is in the first word
printf("Data : %.08X",$data->$getWord(0));

#Change the value
$data->$setWord(0,0xAAAA5555);
printf("Data : %.08X",$data->$getWord(0));

# Write my new value to the chip
$rc = putRing($target, "idreg", $data);
```

WARNING: The '=' operator DOES NOT work between ecmdDataBuffer's or any other structure like you would expect it to in C. See 'Perl API Usage' below

1.5.2 uint64_t (64 bit data)

Perl doesn't support 64 bit variables, so all 64 bit data is converted to a string.

```
my $addr = "0012a33e884bb338";
putMemProc($target, $addr, $data);
```

1.6 Chip/Object targeting

Perl functions use the same **ecmdChipTarget**(p. 20) structure as the C-Api.

```
my $target = new ecmd::ecmdChipTarget();
$target->${chipType} = "pu";
$target->${cage} = 0;
$target->${node} = 0;
$target->${slot} = 0;
$target->${pos} = 1;
$target->${core} = 0;


$rc = putRing($target, "idreg", $data);
```

WARNING: The '=' operator DOES NOT work between ecmdChipTarget's or any other structure like you would expect it to in C. See 'Perl API Usage' below

1.7 Using Lists

Some eCMD functions return data to Perl in the form of a list of elements. This is the same way this data is handled on the C-Api.

Functions available on a Perl list :

$list->size()	Returns the number of entries in the list
$list->empty()	Returns True/False as to wether the list is empty
$list->clear()	Empty the list
$list->push_back($entry)	Push a new entry onto the back of the list
$list->push_front($entry)	Push a new entry onto the front of the list
$entry = $list->pop_back()	Pop an entry from the back of the list and return
$entry = $list->pop_front()	Pop an entry from the front of the list and return
$list->begin()	Returns an iterator that points to the beginning of list
$list->end()	Returns an iterator that points to end of list

Iterators are pointers to a position in the list, they can be used to walk a list and view all elements.

Functions available on a Perl list iterator :

$entry = $listIter->getValue()	Return the entry that this iterator points to
$listIter->setValue($entry)	Overwrite the value at this position with the new entry
$listIter++	Increment the iterator to next entry
$listIter–	Decrement the iterator to previous entry
$listIter == $listIter2	Check if two iterators point to the same entry
$listIter != $listIter2	Check if two iterators point to different entries
$listIter->setIter($list->begin())	Set the iterator pointer to an position in the list
$listIter2->setIter($listIter->getIter())	Retrieve the iterator pointer to assign to another iterator

Loading data into a list :

```
my $isteps = new ecmd::listString();
$isteps->$push_back("proc_cfaminit");
$isteps->$push_back("proc_runiap");
$isteps->$push_back("asic_cfaminit");
iStepsByNameMultiple($isteps);
```

Displaying data on a list :

```
my $istepIter = new ecmd::listStringIterator();
while ($istepIter != $isteps->$end()) {
  printf(" name: %s", $istepIter->$getValue());
  $istepIter++;
}
```

1.8 Using Vectors

Some eCMD functions return data to Perl in the form of a vector(array) of elements. This is the same way this data is handled on the C-Api.

Functions available on a Perl vector :

$vector->size()	Returns the number of entries in the vector
$vector->empty()	Returns True/False as to wether the vector is empty
$vector->clear()	Empty the vector
$vector->push_back($entry)	Push a new entry onto the back of the vector
$entry = $vector->pop_back()	Pop an entry from the back of the vector and return
$entry = $vector->get($index)	Retrieve entry at specified index
$vector->set($index, $entry)	Overwrite entry at specified index
$vector->begin()	Returns an iterator that points to the beginning of vector
$vector->end()	Returns an iterator that points to end of vector

Iterators are pointers to a position in the vector, they can be used to walk a vector and view all elements.

Functions available on a Perl vector iterator :

$entry = $vectorIter->getValue()	Return the entry that this iterator points to
$vectorIter->setValue($entry)	Overwrite the value at this position with the new entry
$vectorIter++	Increment the iterator to next entry
$vectorIter–	Decrement the iterator to previous entry
$vectorIter == $vectorIter2	Check if two iterators point to the same entry
$vectorIter != $vectorIter2	Check if two iterators point to different entries

Usage of a vector is identical to a list like above.

1.9 Perl API Usage

The following should be observed when using the Perl API.

1. The **ecmdLoadDll()**(p. 137) function should ALWAYS be the first function called.
2. The **ecmdUnloadDll()**(p. 138) function should be called just before script exit
3. Perl should be picked up as shown in the example below to grab the eCMD supported version

1.9.1 The '=' operator

WARNING: Because of the behavior of Perl the following statements results in two variables pointing to the same data:

```
my $data1 = new ecmd::ecmdDataBuffer();
$data1.setBitLength(32);
$data1.setWord(0,0xFEEDBEEF);

# Use the = operator
my $data2 = $data1;

printf("Data 1 : %.08X", $data1.getWord(0));        # This will echo 'Data 1 : FEEDBEEF'
printf("Data 2 : %.08X", $data2.getWord(0));        # This will echo 'Data 2 : FEEDBEEF'

# Now just set Data 2
$data2.setWord(0,0xAAAA5555);

printf("Data 1 : %.08X", $data1.getWord(0));        # This will echo 'Data 1 : AAAA5555'
printf("Data 2 : %.08X", $data2.getWord(0));        # This will echo 'Data 2 : AAAA5555'
```

NOTE : Since both $data1 and $data2 point to the same memory space when one is modified they are both modified. This behavior is the same for the **ecmdDataBuffer**(p. 24) and all eCMD structures found in **ecmdStructs.H**(p. 213)

1.10 Example Perl Script

Please see the eCMD web page under 'Use eCMD' for an example Perl script at : `http://rhea.rchland.ibm.com/eCMD/`

Chapter 2

eCMD Perl Module Version Development Hierarchical Index

2.1 eCMD Perl Module Version Development Class Hierarchy

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

ecmdArrayData . . . 13
ecmdArrayEntry . . . 15
ecmdCageData . . . 16
ecmdChipData . . . 17
ecmdChipTarget . . . 20
ecmdCoreData . . . 23
ecmdDataBuffer . . . 24
    ecmdOptimizableDataBuffer . . . 84
ecmdDataBufferImplementationHelper . . . 68
ecmdDllInfo . . . 69
ecmdIndexEntry . . . 71
ecmdIndexVectorEntry . . . 72
ecmdLatchData . . . 73
ecmdLatchEntry . . . 75
ecmdLooperData . . . 77
ecmdMemoryEntry . . . 80
ecmdNameEntry . . . 81
ecmdNameVectorEntry . . . 82
ecmdNodeData . . . 83
ecmdProcRegisterInfo . . . 85
ecmdQueryData . . . 86
ecmdRingData . . . 87
ecmdScomData . . . 89
ecmdSimModelInfo . . . 90
ecmdSlotData . . . 91
ecmdSpyData . . . 92
ecmdSpyGroupData . . . 94
ecmdThreadData . . . 95
ecmdTraceArrayData . . . 96

Chapter 3

eCMD Perl Module Version Development Class Index

3.1 eCMD Perl Module Version Development Class List

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

ecmdArrayData (Used for the ecmdQueryArray function to return array info) 13
ecmdArrayEntry (Used by the getArrayMultiple function to pass data) 15
ecmdCageData (Used for the ecmdQueryConfig function to return cage data) 16
ecmdChipData (Used for the ecmdQueryConfig function to return chip data) 17
ecmdChipTarget (Structure used to designate which cec object/chip you would like the function to operate on) 20
ecmdCoreData (Used for the ecmdQueryConfig function to return core data) 23
ecmdDataBuffer (Provides a means to handle data from the eCMD C API) 24
ecmdDataBufferImplementationHelper (This is used to help low-level implementation of the **ecmdDataBuffer**(p. 24), this CAN NOT be used by any eCMD client or data corruption will occur) 68
ecmdDllInfo (This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with) 69
ecmdIndexEntry (Used by get/put Gpr/Fpr Multiple function to pass data) 71
ecmdIndexVectorEntry (Used by ???? function to pass data) 72
ecmdLatchData (Used for the ecmdQueryLatch function to return latch info) 73
ecmdLatchEntry (Used by getlatch function to return data) 75
ecmdLooperData (Used internally by ecmdConfigLooper to store looping state information) 77
ecmdMemoryEntry (Used by ecmdReadDcard) 80
ecmdNameEntry (Used by get/putSprMultiple function to pass data) 81
ecmdNameVectorEntry (Used by getTraceArrayMultiple function to pass data) . . 82
ecmdNodeData (Used for the ecmdQueryConfig function to return node data) 83
ecmdOptimizableDataBuffer 84
ecmdProcRegisterInfo (Used by ecmdQueryProcRegisterInfo function to return data about a Architected register) 85
ecmdQueryData (Used by the ecmdQueryConfig function to return data) 86
ecmdRingData (Used for the ecmdQueryRing function to return ring info) 87
ecmdScomData (Used for the ecmdQueryScom function to return scom info) 89
ecmdSimModelInfo (Used by simGetModelInfo) 90
ecmdSlotData (Used for the ecmdQueryConfig function to return slot data) 91

ecmdSpyData (Used for the ecmdQuerySpy function to return spy info) 92
ecmdSpyGroupData (Used by get/putspy function to create the return data from a group) 94
ecmdThreadData (Used for the ecmdQueryConfig function to return thread data) . 95
ecmdTraceArrayData (Used for the ecmdQueryTraceArray function to return trace array info) 96

Chapter 4

eCMD Perl Module Version Development File Index

4.1 eCMD Perl Module Version Development File List

Here is a list of all files with brief descriptions:

cipClientPerlapi.H (Cronus & IP eCMD Perlapi Extension) 99
cmdClientPerlapi.H (Command Line eCMD Perlapi Extension) 111
croClientPerlapi.H (Cronus eCMD Perlapi Extension) 113
ecmdClientPerlapi.H (ECMD Perl API Usage :) 124
ecmdDataBuffer.H (Provides a means to handle data from the eCMD C API) 202
ecmdSharedUtils.H (Useful functions for use throughout the ecmd C API and Plugin) 208
ecmdStructs.H (All the Structures required for the eCMD Capi) 213
ecmdUtils.H (Useful functions for use throughout the ecmd C API) 225
gipClientPerlapi.H (IP eCMD Perlapi Extension) 229
zseClientPerlapi.H (Z Series Perlapi Extension) 236

Chapter 5

eCMD Perl Module Version Development Class Documentation

5.1 ecmdArrayData Struct Reference

Used for the ecmdQueryArray function to return array info.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **arrayName**
  (Detail: Low) Names used to reference this array
- int **readAddressLength**
  (Detail: Low) Bit length of read address
- int **writeAddressLength**
  (Detail: Low) Bit length of write address
- int **length**
  (Detail: Low) Length of array (number of entries)
- int **width**
  (Detail: Low) Bit width of array entry
- bool **isCoreRelated**
  (Detail: Low) This array is related to the core level of a chip
- std::string **clockDomain**
  (Detail: High) Clock domain this array belongs to
- **ecmdClockState_t clockState**
  (Detail: High) Required clock state to access this array

5.1.1 Detailed Description

Used for the ecmdQueryArray function to return array info.

5.1.2 Member Data Documentation

5.1.2.1 std::string ecmdArrayData::arrayName

(Detail: Low) Names used to reference this array

5.1.2.2 int ecmdArrayData::readAddressLength

(Detail: Low) Bit length of read address

5.1.2.3 int ecmdArrayData::writeAddressLength

(Detail: Low) Bit length of write address

5.1.2.4 int ecmdArrayData::length

(Detail: Low) Length of array (number of entries)

5.1.2.5 int ecmdArrayData::width

(Detail: Low) Bit width of array entry

5.1.2.6 bool ecmdArrayData::isCoreRelated

(Detail: Low) This array is related to the core level of a chip

5.1.2.7 std::string ecmdArrayData::clockDomain

(Detail: High) Clock domain this array belongs to

5.1.2.8 ecmdClockState_t ecmdArrayData::clockState

(Detail: High) Required clock state to access this array

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

- **ecmdStructs.H**

5.2 ecmdArrayEntry Struct Reference

Used by the getArrayMultiple function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdDataBuffer address**
  Array address/element to access.
- **ecmdDataBuffer buffer**
  Array data from address.
- uint32_t **rc**
  Error code in retrieving this entry.

5.2.1 Detailed Description

Used by the getArrayMultiple function to pass data.

5.2.2 Member Data Documentation

5.2.2.1 ecmdDataBuffer ecmdArrayEntry::address

Array address/element to access.

5.2.2.2 ecmdDataBuffer ecmdArrayEntry::buffer

Array data from address.

5.2.2.3 uint32_t ecmdArrayEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.3 ecmdCageData Struct Reference

Used for the ecmdQueryConfig function to return cage data.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **cageId**
  (Detail: Low) Cage number of this entry
- uint32_t **unitId**
  (Detail: High) Unit Id of this entry
- std::list< **ecmdNodeData** > **nodeData**
  (Detail: Low) List of all nodes requested in this cage - in numerical order by nodeId

5.3.1 Detailed Description

Used for the ecmdQueryConfig function to return cage data.

Operators Supported : <

5.3.2 Member Data Documentation

5.3.2.1 uint32_t ecmdCageData::cageId

(Detail: Low) Cage number of this entry

5.3.2.2 uint32_t ecmdCageData::unitId

(Detail: High) Unit Id of this entry

5.3.2.3 std::list<ecmdNodeData> ecmdCageData::nodeData

(Detail: Low) List of all nodes requested in this cage - in numerical order by nodeId

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

- **ecmdStructs.H**

5.4 ecmdChipData Struct Reference

Used for the ecmdQueryConfig function to return chip data.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **chipType**

  (Detail: Low) Full name of chip , ie. p6, enterprise, corona

- std::string **chipShortType**

  (Detail: Low) Short name of chip, ie. p6, ent, cor (should be 3chars or less)

- std::string **chipCommonType**

  (Detail: Low) common name of chip, ie. pu, iohub, l3cache

- uint32_t **pos**

  (Detail: Low) Position of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

- uint8_t **numProcCores**

  (Detail: Low) Number of cores this entry supports - only valid for Processor compute cores

- uint32_t **chipEc**

  (Detail: High) EC level of this chip, (ec read from 'jtag' chip id or CFAM id)

- uint32_t **simModelEc**

  (Detail: High) Model EC level of this chip

- **ecmdChipInterfaceType_t interfaceType**

  (Detail: High) Interface Macro used by the chip

- uint32_t **chipFlags**

  (Detail: High) Various additional info about the chip - bitmask of defines

- std::list< **ecmdCoreData** > **coreData**

  (Detail: Low) List of all cores requested in this chip - only valid for Processor compute cores - in numerical order by coreId

5.4.1 Detailed Description

Used for the ecmdQueryConfig function to return chip data.

Operators Supported : <

5.4.2 Member Data Documentation

5.4.2.1 std::string ecmdChipData::chipType

(Detail: Low) Full name of chip , ie. p6, enterprise, corona

5.4.2.2 std::string ecmdChipData::chipShortType

(Detail: Low) Short name of chip, ie. p6, ent, cor (should be 3chars or less)

5.4.2.3 std::string ecmdChipData::chipCommonType

(Detail: Low) common name of chip, ie. pu, iohub, l3cache

5.4.2.4 uint32_t ecmdChipData::pos

(Detail: Low) Position of this entry

5.4.2.5 uint32_t ecmdChipData::unitId

(Detail: High) Unit Id of this entry

5.4.2.6 uint8_t ecmdChipData::numProcCores

(Detail: Low) Number of cores this entry supports - only valid for Processor compute cores

5.4.2.7 uint32_t ecmdChipData::chipEc

(Detail: High) EC level of this chip, (ec read from 'jtag' chip id or CFAM id)

5.4.2.8 uint32_t ecmdChipData::simModelEc

(Detail: High) Model EC level of this chip

5.4.2.9 ecmdChipInterfaceType_t ecmdChipData::interfaceType

(Detail: High) Interface Macro used by the chip

5.4.2.10 uint32_t ecmdChipData::chipFlags

(Detail: High) Various additional info about the chip - bitmask of defines

5.4.2.11 std::list<ecmdCoreData> ecmdChipData::coreData

(Detail: Low) List of all cores requested in this chip - only valid for Processor compute cores - in numerical order by coreId

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

- ecmdStructs.H

5.5 ecmdChipTarget Struct Reference

Structure used to designate which cec object/chip you would like the function to operate on.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **cage**
  cage that contains node with chip
- uint32_t **node**
  node that contains chip
- uint32_t **slot**
  Card Slot/Fru to target.
- std::string **chipType**
  name of chip to access , either actual or common name
- uint32_t **pos**
  position of chip within node
- uint8_t **core**
  which core on chip to access, if chip is multi-core
- uint8_t **thread**
  which thread on chip to access, if chip is multi-threaded
- uint32_t **unitId**
  This is an optional field if unitid's are used to specify the target, the above info still needs to be filled in.
- **ecmdChipTargetState_t cageState**
  cage field state
- **ecmdChipTargetState_t nodeState**
  node field state
- **ecmdChipTargetState_t slotState**
  slot field state
- **ecmdChipTargetState_t chipTypeState**
  chipType field state
- **ecmdChipTargetState_t posState**
  pos field state
- **ecmdChipTargetState_t coreState**
  core field state

- **ecmdChipTargetState_t threadState**

  thread field state

- **ecmdChipTargetState_t unitIdState**

  unitId field state

5.5.1 Detailed Description

Structure used to designate which cec object/chip you would like the function to operate on.

- The state bits are used by D/A functions to tell the calling function what level of granularity the function operates on Ex. putmem/getmem display memory through the processor, they are only dependent on cage/node/pos because they do not use the cores to perform their function However put/getspr display architected registers from the processor, they will signify that cage/node/pos/core and depending on the particular spr referenced threads may be valid
- The state bits are used slightly differently for the queryFunctions they are used there to signify what data coming in is valid to refine a query

5.5.2 Member Data Documentation

5.5.2.1 uint32_t ecmdChipTarget::cage

cage that contains node with chip

5.5.2.2 uint32_t ecmdChipTarget::node

node that contains chip

5.5.2.3 uint32_t ecmdChipTarget::slot

Card Slot/Fru to target.

5.5.2.4 std::string ecmdChipTarget::chipType

name of chip to access , either actual or common name

5.5.2.5 uint32_t ecmdChipTarget::pos

position of chip within node

5.5.2.6 uint8_t ecmdChipTarget::core

which core on chip to access, if chip is multi-core

5.5.2.7 uint8_t ecmdChipTarget::thread

which thread on chip to access, if chip is multi-threaded

5.5.2.8 uint32_t ecmdChipTarget::unitId

This is an optional field if unitid's are used to specify the target, the above info still needs to be filled in.

5.5.2.9 ecmdChipTargetState_t ecmdChipTarget::cageState

cage field state

5.5.2.10 ecmdChipTargetState_t ecmdChipTarget::nodeState

node field state

5.5.2.11 ecmdChipTargetState_t ecmdChipTarget::slotState

slot field state

5.5.2.12 ecmdChipTargetState_t ecmdChipTarget::chipTypeState

chipType field state

5.5.2.13 ecmdChipTargetState_t ecmdChipTarget::posState

pos field state

5.5.2.14 ecmdChipTargetState_t ecmdChipTarget::coreState

core field state

5.5.2.15 ecmdChipTargetState_t ecmdChipTarget::threadState

thread field state

5.5.2.16 ecmdChipTargetState_t ecmdChipTarget::unitIdState

unitId field state

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

- **ecmdStructs.H**

5.6 ecmdCoreData Struct Reference

Used for the ecmdQueryConfig function to return core data.

`#include <ecmdStructs.H>`

Public Attributes

- uint8_t **coreId**
  (Detail: Low) core number of this entry
- uint8_t **numProcThreads**
  (Detail: Low) Number of threads per core this entry supports - only valid for Processors
- uint32_t **unitId**
  (Detail: High) Unit Id of this entry
- std::list< **ecmdThreadData** > **threadData**
  (Detail: Low) List of all threads avaliable for this chip - only valid for Processor compute cores - in numerical order

5.6.1 Detailed Description

Used for the ecmdQueryConfig function to return core data.

Operators Supported : <

5.6.2 Member Data Documentation

5.6.2.1 uint8_t ecmdCoreData::coreId

(Detail: Low) core number of this entry

5.6.2.2 uint8_t ecmdCoreData::numProcThreads

(Detail: Low) Number of threads per core this entry supports - only valid for Processors

5.6.2.3 uint32_t ecmdCoreData::unitId

(Detail: High) Unit Id of this entry

5.6.2.4 std::list<ecmdThreadData> ecmdCoreData::threadData

(Detail: Low) List of all threads avaliable for this chip - only valid for Processor compute cores - in numerical order

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

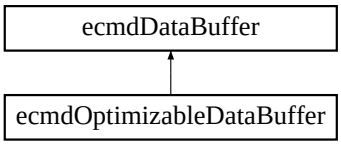
- **ecmdStructs.H**

5.7 ecmdDataBuffer Class Reference

Provides a means to handle data from the eCMD C API.

`#include <ecmdDataBuffer.H>`

Inheritance diagram for ecmdDataBuffer::



Public Member Functions

ecmdDataBuffer Constructors

- **ecmdDataBuffer** ()
  Default Constructor.
- **ecmdDataBuffer** (uint32_t i_numBits)
  Constructor.
- **ecmdDataBuffer** (const **ecmdDataBuffer** &other)
  Copy Constructor.
- virtual ~**ecmdDataBuffer** ()
  Default Destructor.

Buffer Size Functions

- uint32_t **clear** ()
  Called by the destructor, available to user to reset buffer to default constructor state.
- uint32_t **getWordLength** () const
  Return the length of the buffer in words.
- uint32_t **getByteLength** () const
  Return the length of the buffer in bytes.
- uint32_t **getBitLength** () const
  Return the length of the buffer in bits.
- uint32_t **getCapacity** () const
  Return the actual capacity of the internal buffer in words.
- uint32_t **setWordLength** (uint32_t i_newNumWords)
  Reinitialize the Buffer to specified length.
- uint32_t **setByteLength** (uint32_t i_newNumBytes)
  Reinitialize the Buffer to specified length.

- uint32_t **setBitLength** (uint32_t i_newNumBits)
  Reinitialize the Buffer to specified length.
- uint32_t **setCapacity** (uint32_t i_newNumWords)
  Reinitialize the internal buffer to specified length.
- uint32_t **shrinkBitLength** (uint32_t i_newNumBits)
  Shrink buffer size to a new bit size.
- uint32_t **growBitLength** (uint32_t i_newNumBits)
  Expand buffer size to a new bit size maintaining current data.
- virtual bool **isBufferOptimizable** (void)
  Returns value of iv_ BufferOptimizable.

Bit/Word Manipulation Functions

- uint32_t **setBit** (uint32_t i_bit)
  Turn on a bit in buffer.
- uint32_t **setBit** (uint32_t i_bit, uint32_t i_len)
  Turn on a bit in buffer.
- uint32_t **writeBit** (uint32_t i_bit, uint32_t i_value)
  Write a bit to specified value in buffer.
- uint32_t **setWord** (uint32_t i_wordoffset, uint32_t i_value)
  Set a word of data in buffer.
- uint32_t **getWord** (uint32_t i_wordoffset) const
  Fetch a word from ecmdDataBuffer.
- uint32_t **setByte** (uint32_t i_byteoffset, uint8_t i_value)
  Set a byte of data in buffer.
- uint8_t **getByte** (uint32_t i_byteoffset) const
  Fetch a byte from ecmdDataBuffer.
- uint32_t **setHalfWord** (uint32_t i_halfwordoffset, uint16_t i_value)
  Set a halfword of data in buffer.
- uint16_t **getHalfWord** (uint32_t i_halfwordoffset) const
  Fetch a halfword from ecmdDataBuffer.
- uint32_t **setDoubleWord** (uint32_t i_doublewordoffset, uint64_t i_value)
  Set a doubleword of data in buffer.
- uint64_t **getDoubleWord** (uint32_t i_doublewordoffset) const
  Fetch a doubleword from ecmdDataBuffer.
- uint32_t **clearBit** (uint32_t i_bit)
  Clear a bit in buffer.
- uint32_t **clearBit** (uint32_t i_bit, uint32_t i_len)

Clear multiple bits in buffer.

- uint32_t **flipBit** (uint32_t i_bit)
  Invert bit.
- uint32_t **flipBit** (uint32_t i_bit, uint32_t i_len)
  Invert multiple bits.
- bool **isBitSet** (uint32_t i_bit) const
  Test if bit is set.
- bool **isBitSet** (uint32_t i_bit, uint32_t i_len) const
  Test if multiple bits are set.
- bool **isBitClear** (uint32_t i_bit) const
  Test if bit is clear.
- bool **isBitClear** (uint32_t i_bit, uint32_t i_len) const
  Test if multiple bits are clear.
- uint32_t **getNumBitsSet** (uint32_t i_bit, uint32_t i_len) const
  Count number of bits set in a range.

Buffer Manipulation Functions

- uint32_t **shiftRight** (uint32_t i_shiftnum)
  Shift data to right.
- uint32_t **shiftLeft** (uint32_t i_shiftnum)
  Shift data to left.
- uint32_t **shiftRightAndResize** (uint32_t i_shiftnum)
  Shift data to right - resizing buffer.
- uint32_t **shiftLeftAndResize** (uint32_t i_shiftnum)
  Shift data to left - resizing buffer.
- uint32_t **rotateRight** (uint32_t i_rotatenum)
  Rotate data to right.
- uint32_t **rotateLeft** (uint32_t i_rotatenum)
  Rotate data to left.
- uint32_t **flushTo0** ()
  Clear entire buffer to 0's.
- uint32_t **flushTo1** ()
  Set entire buffer to 1's.
- uint32_t **invert** ()
  Invert entire buffer.
- uint32_t **reverse** ()
  Bit reverse entire buffer.

- uint32_t **applyInversionMask** (const uint32_t *i_invMask, uint32_t i_invByteLen)
  Apply an inversion mask to data inside buffer.

- uint32_t **applyInversionMask** (const **ecmdDataBuffer** &i_invMaskBuffer, uint32_t i_invByteLen)
  Apply an inversion mask to data inside buffer Just a wrapper that takes in a ecmdDataBuffer and calls uint32_t applyInversionMask.

- uint32_t **insert** (const **ecmdDataBuffer** &i_bufferIn, uint32_t i_targetStart, uint32_t i_len, uint32_t i_sourceStart=0)
  Copy part of another DataBuffer into this one.

- uint32_t **insert** (const uint32_t *i_datain, uint32_t i_targetStart, uint32_t i_len, uint32_t i_sourceStart=0)
  Copy part of a uint32_t array into this DataBuffer.

- uint32_t **insert** (uint32_t i_datain, uint32_t i_targetStart, uint32_t i_len, uint32_t i_sourceStart=0)
  Copy part of a uint32_t into the DataBuffer.

- uint32_t **insertFromRight** (const uint32_t *i_datain, uint32_t i_start, uint32_t i_len)
  Copy a right aligned (decimal) uint32_t array into this DataBuffer.

- uint32_t **insertFromRight** (uint32_t i_datain, uint32_t i_start, uint32_t i_len)
  Copy a right aligned (decimal) uint32_t into the DataBuffer.

- uint32_t **extract** (**ecmdDataBuffer** &o_bufferOut, uint32_t i_start, uint32_t i_len) const
  Copy data from this DataBuffer into another.

- uint32_t **extract** (uint32_t *o_data, uint32_t i_start, uint32_t i_len) const
  Copy data from this DataBuffer into another.

- uint32_t **extractPreserve** (**ecmdDataBuffer** &o_bufferOut, uint32_t i_start, uint32_t i_len, uint32_t i_targetStart=0) const
  Copy data from this buffer into another at a given offset, preserving the size and other data in the output buffer.

- uint32_t **extractPreserve** (uint32_t *o_data, uint32_t i_start, uint32_t i_len, uint32_t i_targetStart=0) const
  Copy data from this DataBuffer into a generic output buffer at a given offset.

- uint32_t **extractToRight** (**ecmdDataBuffer** &o_bufferOut, uint32_t i_start, uint32_t i_len) const
  Copy data from this DataBuffer into another DataBuffer and right justify.

- uint32_t **extractToRight** (uint32_t *o_data, uint32_t i_start, uint32_t i_len) const
  Copy data from this DataBuffer into a uint32_t buffer.

- uint32_t **concat** (const **ecmdDataBuffer** &i_buf0, const **ecmdDataBuffer** &i_buf1)
  Concatenate 2 DataBuffers into in this one.

- uint32_t **concat** (const **ecmdDataBuffer** &i_buf0, const **ecmdDataBuffer** &i_buf1, const **ecmdDataBuffer** &i_buf2)
  Concatenate 3 DataBuffers into in this one.

- uint32_t **setOr** (const **ecmdDataBuffer** &i_bufferIn, uint32_t i_startbit, uint32_t i_len)
  OR data into DataBuffer.

- uint32_t **setOr** (const uint32_t *i_datain, uint32_t i_startbit, uint32_t i_len)
  OR data into DataBuffer.

- uint32_t **setOr** (uint32_t i_datain, uint32_t i_startbit, uint32_t i_len)
  OR data into DataBuffer.

- uint32_t **merge** (const **ecmdDataBuffer** &i_bufferIn)
  OR data into DataBuffer.

- uint32_t **setXor** (const **ecmdDataBuffer** &i_bufferIn, uint32_t i_startbit, uint32_t i_len)
  XOR data into DataBuffer.

- uint32_t **setXor** (const uint32_t *i_datain, uint32_t i_startbit, uint32_t i_len)
  XOR data into DataBuffer.

- uint32_t **setXor** (uint32_t i_datain, uint32_t i_startbit, uint32_t i_len)
  XOR data into DataBuffer.

- uint32_t **setAnd** (const **ecmdDataBuffer** &i_bufferIn, uint32_t i_startbit, uint32_t i_len)
  AND data into DataBuffer.

- uint32_t **setAnd** (const uint32_t *i_datain, uint32_t i_startbit, uint32_t i_len)
  AND data into DataBuffer.

- uint32_t **setAnd** (uint32_t i_datain, uint32_t i_startbit, uint32_t i_len)
  AND data into DataBuffer.

- uint32_t **copy** (**ecmdDataBuffer** &o_copyBuffer) const
  Copy entire contents of this ecmdDataBuffer into o_copyBuffer.

- uint32_t **memCopyIn** (const uint32_t *i_buf, uint32_t i_bytes)
  Copy buffer into this ecmdDataBuffer.

- uint32_t **memCopyOut** (uint32_t *o_buf, uint32_t i_bytes) const
  Copy DataBuffer into supplied uint32_t buffer.

- uint32_t **flatten** (uint8_t *o_data, uint32_t i_len) const
  Flatten all the object data into a uint8_t buffer.

- uint32_t **unflatten** (const uint8_t *i_data, uint32_t i_len)
  Unflatten object data from a uint8_t buffer into this DataBuffer.

- uint32_t **flattenSize** (void) const
  Return number of bytes needed for a buffer to flatten the object.

Parity Functions

- uint32_t **oddParity** (uint32_t i_start, uint32_t i_stop) const
  Generate odd parity over a range of bits.
- uint32_t **evenParity** (uint32_t i_start, uint32_t i_stop) const
  Generate even parity over a range of bits.
- uint32_t **oddParity** (uint32_t i_start, uint32_t i_stop, uint32_t i_insertpos)
  Generate odd parity over a range of bits and insert into DataBuffer.
- uint32_t **evenParity** (uint32_t i_start, uint32_t i_stop, uint32_t i_insertpos)
  Generate even parity over a range of bits and insert into DataBuffer.

Buffer Character Conversion Functions

- std::string **genHexLeftStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a hex left aligned char string.
- std::string **genHexRightStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a hex right aligned char string.
- std::string **genBinStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a binary char string.
- std::string **genAsciiStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as an ASCII char string. If it's out of range, a . is printed.
- std::string **genHexLeftStr** () const
  Return entire buffer as a hex left aligned char string.
- std::string **genHexRightStr** () const
  Return entire buffer as a hex right aligned char string.
- std::string **genBinStr** () const
  Return entire buffer as a binary char string.
- std::string **genAsciiStr** () const
  Return Data as an ASCII char string. If it's out of range, a . is printed.
- std::string **genXstateStr** (uint32_t i_start, uint32_t i_bitlen) const
  Retrieve a section of the Xstate Data.
- std::string **genXstateStr** () const
  Retrieve entire Xstate Data buffer.

String to Data conversion functions

- uint32_t **insertFromHexLeft** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_length=0)
  Convert data from a hex left-aligned string and insert it into this data buffer.

- uint32_t **insertFromHexLeftAndResize** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_length=0)

  Convert data from a hex left-aligned string and insert it into this data buffer - and set's buffer length to size of data.

- uint32_t **insertFromHexRight** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_expectedLength=0)

  Convert data from a hex right-aligned string and insert it into this data buffer.

- uint32_t **insertFromHexRightAndResize** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_expectedLength=0)

  Convert data from a hex right-aligned string and insert it into this data buffer - and set's buffer length to size of data.

- uint32_t **insertFromBin** (const char *i_binChars, uint32_t i_start=0)

  Convert data from a binary string and insert it into this data buffer.

- uint32_t **insertFromBinAndResize** (const char *i_binChars, uint32_t i_start=0)

  Convert data from a binary string and insert it into this data buffer - and set's buffer length to size of data.

Simulation Buffer Functions

- uint32_t **enableXstateBuffer** ()

  Initializes the X-state buffer, from then on all changes are reflected in Xstate.

- uint32_t **disableXstateBuffer** ()

  Removes the X-state buffer, from then on no changes are made to Xstate.

- bool **isXstateEnabled** () const

  Query to find out if this buffer has X-states enabled.

- uint32_t **flushToX** (char i_value)

  Load entire buffer with an X-state value.

- bool **hasXstate** () const

  Check Entire buffer for any X-state values.

- bool **hasXstate** (uint32_t i_start, uint32_t i_length) const

  Check section of buffer for any X-state values.

- char **getXstate** (uint32_t i_bit) const

  Retrieve an Xstate value from the buffer.

- uint32_t **setXstate** (uint32_t i_bit, char i_value)

  Set an Xstate value in the buffer.

- uint32_t **setXstate** (uint32_t i_bit, char i_value, uint32_t i_length)

  Set an Xstate value in the buffer.

- uint32_t **setXstate** (uint32_t i_bitoffset, const char *i_datastr)

  Set a range of Xstate values in buffer.

- uint32_t **memCopyInXstate** (const char *i_buf, uint32_t i_bytes)

Copy buffer into the Xstate data of this ecmdDataBuffer.

- uint32_t **memCopyOutXstate** (char *o_buf, uint32_t i_bytes) const
  Copy DataBuffer into supplied char buffer from Xstate data.

Misc Functions

- uint32_t **writeFile** (const char *i_filename, **ecmdFormatType_t** i_format, const char *i_facName=NULL)
  Write buffer out into a file in the format specified.

- uint32_t **writeFileMultiple** (const char *i_filename, **ecmdFormatType_t** i_format, **ecmdWriteMode_t** i_mode, uint32_t &o_dataNumber, const char *i_property=NULL)
  Writes/Appends buffer out into a file in the format specified.

- uint32_t **writeFileStream** (std::ostream &o_filestream)
  Write buffer out into the stream in ECMD_SAVE_FORMAT_BINARY_DATA format.

- uint32_t **readFile** (const char *i_filename, **ecmdFormatType_t** i_format, std::string *o_property=NULL)
  Read data from the file into the buffer.

- uint32_t **readFileMultiple** (const char *i_filename, **ecmdFormatType_t** i_format, uint32_t i_dataNumber=0, std::string *o_property=NULL)
  Read data from the file into the buffer.

- uint32_t **queryNumOfBuffers** (const char *i_filename, **ecmdFormatType_t** i_format, uint32_t &o_num)
  Get the number of databuffers stored in the file created by writeFile/writeFileMultiple.

- uint32_t **readFileStream** (std::istream &i_filestream, uint32_t i_bitlength)
  Read data from the stream (in ECMD_SAVE_FORMAT_BINARY_DATA format) into the buffer.

- uint32_t **shareBuffer** (**ecmdDataBuffer** *i_sharingBuffer)
  This function will take the passed in buffer, delete any current data it holds, and point its data var to that which is owned by the one being called with. It will not have iv_UserOwned flag set, so it should not delete the buffer it points to, nor resize it, but it can alter the data. The use of this function is for caching data for reads.

- void **queryErrorState** (uint32_t &o_errorState)
  This function returns the stored error state that could have been caused by any number of previous operations on the buffer.

Operator overloads

- int **operator==** (const **ecmdDataBuffer** &other) const
  Overload the == operator.

- int **operator!=** (const **ecmdDataBuffer** &other) const
  Overload the != operator.

- **ecmdDataBuffer operator &** (const **ecmdDataBuffer** &other) const

  Overload the & operator.

- **ecmdDataBuffer operator|** (const **ecmdDataBuffer** &other) const

  Overload the | operator.

Protected Member Functions

- uint32_t **fillDataStr** (char fillChar)

Protected Attributes

- uint32_t **iv_Capacity**

  Actual buffer capacity - always >= iv_NumWords.

- uint32_t **iv_NumWords**

  Specified buffer size rounded to next word.

- uint32_t **iv_NumBits**

  Specified buffer size in bits.

- uint32_t * **iv_Data**

  Pointer to buffer inside iv_RealData.

- uint32_t * **iv_RealData**

  Real buffer - with header and tail.

- bool **iv_UserOwned**

  Whether or not this buffer owns the data.

- bool **iv_BufferOptimizable**

  Whether or not this is an optimizable buffer.

- char * **iv_DataStr**
- bool **iv_XstateEnabled**

Friends

- class **ecmdDataBufferImplementationHelper**

5.7.1 Detailed Description

Provides a means to handle data from the eCMD C API.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ecmdDataBuffer::ecmdDataBuffer ()

Default Constructor.

Postcondition:
buffer is not allocated, can be allocated later with setWordLength, setCapacity or setBitLength

5.7.2.2 ecmdDataBuffer::ecmdDataBuffer (uint32_t *i_numBits*)

Constructor.

Parameters:
i_numBits Size of data in bits to initialize

Postcondition:
ecmdDataBuffer is initialized and zero'd out

5.7.2.3 ecmdDataBuffer::ecmdDataBuffer (const ecmdDataBuffer & *other*)

Copy Constructor.

Parameters:
other Buffer to copy

5.7.2.4 virtual ecmdDataBuffer::~ecmdDataBuffer () [virtual]

Default Destructor.

5.7.3 Member Function Documentation

5.7.3.1 uint32_t ecmdDataBuffer::clear ()

Called by the destructor, available to user to reset buffer to default constructor state.

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_NOT_OWNER when called on buffer not owned
nonzero on failure

Postcondition:
Memory deallocated and size set to 0

5.7.3.2 uint32_t ecmdDataBuffer::getWordLength () const

Return the length of the buffer in words.

Return values:
Buffer length in words rounded up

5.7.3.3 uint32_t ecmdDataBuffer::getByteLength () const

Return the length of the buffer in bytes.

Return values:
Buffer length in bytes rounded up

5.7.3.4 uint32_t ecmdDataBuffer::getBitLength () const

Return the length of the buffer in bits.

Return values:
Buffer length in bits

5.7.3.5 uint32_t ecmdDataBuffer::getCapacity () const

Return the actual capacity of the internal buffer in words.

Return values:
Actual capacity in words of internal buffer

5.7.3.6 uint32_t ecmdDataBuffer::setWordLength (uint32_t *i_newNumWords*)

Reinitialize the Buffer to specified length.

Parameters:
i_newNumWords Length of new buffer in words

Postcondition:
Buffer is reinitialized and zero'd out

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_INIT_FAIL failure occurred setting new length
ECMD_DBUF_NOT_OWNER when called on buffer not owned

NOTE : Capacity will be adjusted to fit new size if neccesary CAUTION : All data stored in buffer will be lost

5.7.3.7 uint32_t ecmdDataBuffer::setByteLength (uint32_t *i_newNumBytes*)

Reinitialize the Buffer to specified length.

Parameters:
i_newNumBytes Length of new buffer in bytes

Postcondition:
Buffer is reinitialized and zero'd out

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_INIT_FAIL failure occurred setting new length
ECMD_DBUF_NOT_OWNER when called on buffer not owned

NOTE : Capacity will be adjusted to fit new size if neccesary CAUTION : All data stored in buffer will be lost

5.7.3.8 uint32_t ecmdDataBuffer::setBitLength (uint32_t *i_newNumBits*)

Reinitialize the Buffer to specified length.

Parameters:
i_newNumBits Length of new buffer in bits

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_INIT_FAIL failure occurred setting new length
ECMD_DBUF_NOT_OWNER when called on buffer not owned

Postcondition:
Buffer is reinitialized and zero'd out

NOTE : Capacity will be adjusted to fit new size if neccesary CAUTION : All data stored in buffer will be lost

5.7.3.9 uint32_t ecmdDataBuffer::setCapacity (uint32_t *i_newNumWords*)

Reinitialize the internal buffer to specified length.

Parameters:
i_newNumWords length of internal data buffer in words

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_INIT_FAIL failure occurred setting new length
ECMD_DBUF_NOT_OWNER when called on buffer not owned

Postcondition:
Internal buffer is reinitialized and zero'd out. Requests to decrease the capacity are ignored

CAUTION : All data stored in buffer will be lost

5.7.3.10 uint32_t ecmdDataBuffer::shrinkBitLength (uint32_t *i_newNumBits*)

Shrink buffer size to a new bit size.

Parameters:
i_newNumBits New bit length for buffer (must be <= current buffer length)

Return values:
ECMD_DBUF_SUCCESS on success

Postcondition:
Internal buffer size is reset but data inside new size is not lost

5.7.3.11 uint32_t ecmdDataBuffer::growBitLength (uint32_t *i_newNumBits*)

Expand buffer size to a new bit size maintaining current data.

Parameters:
i_newNumBits New bit length for buffer

Return values:
ECMD_DBUF_SUCCESS on success

Postcondition:
Internal buffer size is reset but data inside is not lost

NOTE : Capacity will be adjusted to fit new size if neccesary

5.7.3.12 virtual bool ecmdDataBuffer::isBufferOptimizable (void) [inline, virtual]

Returns value of iv_BufferOptimizable.

5.7.3.13 uint32_t ecmdDataBuffer::setBit (uint32_t *i_bit*)

Turn on a bit in buffer.

Parameters:
i_bit Bit in buffer to turn on

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.14 uint32_t ecmdDataBuffer::setBit (uint32_t *i_bit*, uint32_t *i_len*)

Turn on a bit in buffer.

Parameters:

i_bit start bit in buffer to turn on

i_len Number of consecutive bits from start bit to turn on

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.15 uint32_t ecmdDataBuffer::writeBit (uint32_t *i_bit*, uint32_t *i_value*)

Write a bit to specified value in buffer.

Parameters:

i_bit Bit in buffer to turn on

i_value Value to write

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.16 uint32_t ecmdDataBuffer::setWord (uint32_t *i_wordoffset*, uint32_t *i_value*)

Set a word of data in buffer.

Parameters:

i_wordoffset Offset of word to set

i_value 32 bits of data to put into word

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_wordoffset is not contained in the size of this buffer

5.7.3.17 uint32_t ecmdDataBuffer::getWord (uint32_t *i_wordoffset*) const

Fetch a word from ecmdDataBuffer.

Parameters:

i_wordoffset Offset of word to fetch

Return values:

Value of word requested

5.7.3.18 uint32_t ecmdDataBuffer::setByte (uint32_t *i_byteoffset*, uint8_t *i_value*)

Set a byte of data in buffer.

Parameters:

i_byteoffset Offset of byte to set

i_value 8 bits of data to put into byte

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_byteoffset is not contained in the size of this buffer

5.7.3.19 uint8_t ecmdDataBuffer::getByte (uint32_t *i_byteoffset*) const

Fetch a byte from ecmdDataBuffer.

Parameters:

i_byteoffset Offset of byte to fetch

Return values:

Value of byte requested

NOTE : If offset > buffer length retval = 0 and error printed

5.7.3.20 uint32_t ecmdDataBuffer::setHalfWord (uint32_t *i_halfwordoffset*, uint16_t *i_value*)

Set a halfword of data in buffer.

Parameters:

i_halfwordoffset Offset of halfword to set

i_value 16 bits of data to put into halfword

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_halfwordoffset is not contained in the size of this buffer

5.7.3.21 uint16_t ecmdDataBuffer::getHalfWord (uint32_t *i_halfwordoffset*) const

Fetch a halfword from ecmdDataBuffer.

Parameters:

i_halfwordoffset Offset of halfword to fetch

Return values:

Value of halfword requested

5.7.3.22 uint32_t ecmdDataBuffer::setDoubleWord (uint32_t *i_doublewordoffset*, uint64_t *i_value*)

Set a doubleword of data in buffer.

Parameters:

i_doublewordoffset Offset of doubleword to set

i_value 64 bits of data to put into doubleword

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_doublewordoffset is not contained in the size of this buffer

5.7.3.23 uint64_t ecmdDataBuffer::getDoubleWord (uint32_t *i_doublewordoffset*) const

Fetch a doubleword from ecmdDataBuffer.

Parameters:

i_doublewordoffset Offset of doubleword to fetch

Return values:

Value of doubleword requested

5.7.3.24 uint32_t ecmdDataBuffer::clearBit (uint32_t *i_bit*)

Clear a bit in buffer.

Parameters:

i_bit Bit in buffer to turn off

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.25 uint32_t ecmdDataBuffer::clearBit (uint32_t *i_bit*, uint32_t *i_len*)

Clear multiple bits in buffer.

Parameters:

i_bit Start bit in buffer to turn off

i_len Number of consecutive bits from start bit to off

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.26 uint32_t ecmdDataBuffer::flipBit (uint32_t *i_bit*)

Invert bit.

Parameters:
i_bit Bit in buffer to invert

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.27 uint32_t ecmdDataBuffer::flipBit (uint32_t *i_bit*, uint32_t *i_len*)

Invert multiple bits.

Parameters:
i_bit Start bit in buffer to invert
i_len Number of consecutive bits to invert

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

5.7.3.28 bool ecmdDataBuffer::isBitSet (uint32_t *i_bit*) const

Test if bit is set.

Parameters:
i_bit Bit to test

Return values:
true if bit is set - false if bit is clear

5.7.3.29 bool ecmdDataBuffer::isBitSet (uint32_t *i_bit*, uint32_t *i_len*) const

Test if multiple bits are set.

Parameters:
i_bit Start bit to test
i_len Number of consecutive bits to test

Return values:
true if all bits in range are set - false if any bit is clear

5.7.3.30 bool ecmdDataBuffer::isBitClear (uint32_t *i_bit*) const

Test if bit is clear.

Parameters:
i_bit Bit to test

Return values:
true if bit is clear - false if bit is set

5.7.3.31 bool ecmdDataBuffer::isBitClear (uint32_t *i_bit*, uint32_t *i_len*) const

Test if multiple bits are clear.

Parameters:
i_bit Start bit to test

i_len Number of consecutive bits to test

Return values:
true if all bits in range are clear - false if any bit is set

5.7.3.32 uint32_t ecmdDataBuffer::getNumBitsSet (uint32_t *i_bit*, uint32_t *i_len*) const

Count number of bits set in a range.

Parameters:
i_bit Start bit to test

i_len Number of consecutive bits to test

Return values:
Number of bits set in range

5.7.3.33 uint32_t ecmdDataBuffer::shiftRight (uint32_t *i_shiftnum*)

Shift data to right.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to right by specified number of bits - data is shifted off the end
Buffer size is unchanged

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.34 uint32_t ecmdDataBuffer::shiftLeft (uint32_t *i_shiftnum*)

Shift data to left.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to left by specified number of bits - data is shifted off the beginning
Buffer size is unchanged

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.35 uint32_t ecmdDataBuffer::shiftRightAndResize (uint32_t *i_shiftnum*)

Shift data to right - resizing buffer.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to right by specified number of bits
Buffer size is resized to accomodate shift

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_NOT_OWNER when called on buffer not owned

5.7.3.36 uint32_t ecmdDataBuffer::shiftLeftAndResize (uint32_t *i_shiftnum*)

Shift data to left - resizing buffer.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to left by specified number of bits - data is shifted off the beginning
Buffer size is resized to accomodate shift

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_NOT_OWNER when called on buffer not owned

5.7.3.37 uint32_t ecmdDataBuffer::rotateRight (uint32_t *i_rotatenum*)

Rotate data to right.

Parameters:
i_rotatenum Number of bits to rotate

Postcondition:
Bits in buffer are rotated to the right by specified number of bits - data is rotated to the beginning

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.38 uint32_t ecmdDataBuffer::rotateLeft (uint32_t *i_rotatenum*)

Rotate data to left.

Parameters:
i_rotatenum Number of bits to rotate

Postcondition:
Bits in buffer are rotated to the left by specified number of bits - data is rotated to the end

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.39 uint32_t ecmdDataBuffer::flushTo0 ()

Clear entire buffer to 0's.

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.40 uint32_t ecmdDataBuffer::flushTo1 ()

Set entire buffer to 1's.

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.41 uint32_t ecmdDataBuffer::invert ()

Invert entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.42 uint32_t ecmdDataBuffer::reverse ()

Bit reverse entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.43 uint32_t ecmdDataBuffer::applyInversionMask (const uint32_t * *i_invMask*, uint32_t *i_invByteLen*)

Apply an inversion mask to data inside buffer.

Parameters:
i_invMask Buffer that stores inversion mask
i_invByteLen Buffer length provided in bytes

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.44 uint32_t ecmdDataBuffer::applyInversionMask (const ecmdDataBuffer & *i_invMaskBuffer*, uint32_t *i_invByteLen*)

Apply an inversion mask to data inside buffer Just a wrapper that takes in a ecmdDataBuffer and calls uint32_t applyInversionMask.

Parameters:
i_invMaskBuffer Buffer that stores inversion mask
i_invByteLen Buffer length provided in bytes

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.45 uint32_t ecmdDataBuffer::insert (const ecmdDataBuffer & *i_bufferIn*, uint32_t *i_targetStart*, uint32_t *i_len*, uint32_t *i_sourceStart* = 0)

Copy part of another DataBuffer into this one.

Parameters:
i_bufferIn DataBuffer to copy data from - data is taken left aligned
i_targetStart Start bit to insert to
i_len Length of bits to insert
i_sourceStart Start bit in i_bufferIn - default value is zero

Precondition:
DataBuffer must be pre-allocated

Postcondition:
Data is copied from i_bufferIn to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.46 uint32_t ecmdDataBuffer::insert (const uint32_t * *i_datain*, uint32_t *i_targetStart*, uint32_t *i_len*, uint32_t *i_sourceStart* = 0)

Copy part of a uint32_t array into this DataBuffer.

Parameters:

i_datain uint32_t array to copy into this DataBuffer - data is taken left aligned

i_targetStart Start bit to insert into

i_len Length of bits to insert

i_sourceStart Start bit in i_datain - default value is zero

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Data is copied from i_datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.47 uint32_t ecmdDataBuffer::insert (uint32_t *i_datain*, uint32_t *i_targetStart*, uint32_t *i_len*, uint32_t *i_sourceStart* = 0)

Copy part of a uint32_t into the DataBuffer.

Parameters:

i_datain uint32_t value to copy into DataBuffer - data is taken left aligned

i_targetStart Start bit to insert into

i_len Length of bits to insert (must be <= 32)

i_sourceStart Start bit in i_datain - default value is zero

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Data is copied from bufferIn to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.48 uint32_t ecmdDataBuffer::insertFromRight (const uint32_t * *i_ datain*, uint32_t *i_ start*, uint32_t *i_ len*)

Copy a right aligned (decimal) uint32_t array into this DataBuffer.

Parameters:

i_ datain uint32_t array to copy into this DataBuffer - data is taken right aligned

i_ start Start bit to insert into

i_ len Length of bits to insert

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Data is copied from datain into this DataBuffer at specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

NOTE : Data is assumed to be aligned on the word boundary of i_len

5.7.3.49 uint32_t ecmdDataBuffer::insertFromRight (uint32_t *i_ datain*, uint32_t *i_ start*, uint32_t *i_ len*)

Copy a right aligned (decimal) uint32_t into the DataBuffer.

Parameters:

i_ datain uint32_t value to copy into DataBuffer - data is taken right aligned

i_ start Start bit to insert into

i_ len Length of bits to insert (must be <= 32)

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Data is copied from datain into this DataBuffer at specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.50 uint32_t ecmdDataBuffer::extract (ecmdDataBuffer & *o_ bufferOut*, uint32_t *i_ start*, uint32_t *i_ len*) const

Copy data from this DataBuffer into another.

Parameters:

o_ bufferOut DataBuffer to copy into - data is placed left aligned

i_ start Start bit of data in this DataBuffer to copy

i_ len Length of consecutive bits to copy

Postcondition:
Data is copied from specified location in this DataBuffer to bufferOut

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

NOTE : The o_bufferOut buffer is resized to the extract length and any data in the buffer is lost

5.7.3.51 uint32_t ecmdDataBuffer::extract (uint32_t * *o_ data*, uint32_t *i_ start*, uint32_t *i_ len*) const

Copy data from this DataBuffer into another.

Parameters:
o_ data uint32_t buffer to copy into - data is placed left aligned - must be pre-allocated

i_ start Start bit of data in DataBuffer to copy

i_ len Length of consecutive bits to copy

Postcondition:
Data is copied from specified location in this DataBuffer to o_data

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.52 uint32_t ecmdDataBuffer::extractPreserve (ecmdDataBuffer & *o_ bufferOut*, uint32_t *i_ start*, uint32_t *i_ len*, uint32_t *i_ targetStart* = 0) const

Copy data from this buffer into another at a given offset, preserving the size and other data in the output buffer.

Parameters:
o_ bufferOut Target data buffer where data is copied into

i_ start Start bit in this DataBuffer to begin copy

i_ len Length of consecutive bits to copy

i_ targetStart Start bit in output buffer where data is copied defaults to zero

Postcondition:
Data is copied from offset in this buffer to offset in out buffer

Return values:
ECMD_ DBUF_ SUCCESS on success

EMCD_ DBUF_ BUFFER_ OVERFLOW data requested is out of range in one of the 2 buffers

5.7.3.53 uint32_t ecmdDataBuffer::extractPreserve (uint32_t * *o_ data*, uint32_t *i_ start*, uint32_t *i_ len*, uint32_t *i_ targetStart* = 0) const

Copy data from this DataBuffer into a generic output buffer at a given offset.

Parameters:

o_ data Array of data to write into, must be pre-allocated

i_ start Start bit in this DataBuffer to begin the copy

i_ len Length of consecutive bits to copy

i_ targetStart Starting bit in output data to place extracted data, defaults to zero

Postcondition:

Data is copied from offset in this DataBuffer to offset in output buffer

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ INIT_ FAIL unable to allocate databuffer

ECMD_ DBUF BUFFER_OVERFLOW request is out of range for this DataBuffer, output buffer is NOT checked for overflow

5.7.3.54 uint32_t ecmdDataBuffer::extractToRight (ecmdDataBuffer & *o_ bufferOut*, uint32_t *i_ start*, uint32_t *i_ len*) const

Copy data from this DataBuffer into another DataBuffer and right justify.

Parameters:

o_ bufferOut DataBuffer to copy into - data is placed right aligned

i_ start Start bit of data in DataBuffer to copy

i_ len Length of consecutive bits to copy

Postcondition:

Data is copied from specified location in this DataBuffer to o_bufferOut, right aligned. Data is only right aligned if i_len < 32

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.55 uint32_t ecmdDataBuffer::extractToRight (uint32_t * *o_ data*, uint32_t *i_ start*, uint32_t *i_ len*) const

Copy data from this DataBuffer into a uint32_t buffer.

Parameters:

o_ data uint32_t buffer to copy into - data is placed right aligned - must be pre-allocated

i_ start Start bit of data in DataBuffer to copy

i_ len Length of consecutive bits to copy

Postcondition:
Data is copied from specified location in this DataBuffer to o_data, right aligned. Data is only right aligned if i_len < 32

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.56 uint32_t ecmdDataBuffer::concat (const ecmdDataBuffer & *i_buf0*, const ecmdDataBuffer & *i_buf1*)

Concatenate 2 DataBuffers into in this one.

Parameters:
i_buf0 First DataBuffer to concatenate; copied to beginning of this buffer
i_buf1 Second DataBuffer to concatenate; copied to this buffer after the first buffer

Postcondition:
Space is allocated, and data from the 2 DataBuffers is concatenated and copied to this buffer

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.57 uint32_t ecmdDataBuffer::concat (const ecmdDataBuffer & *i_buf0*, const ecmdDataBuffer & *i_buf1*, const ecmdDataBuffer & *i_buf2*)

Concatenate 3 DataBuffers into in this one.

Parameters:
i_buf0 First DataBuffer to concatenate; copied to beginning of this buffer
i_buf1 Second DataBuffer to concatenate; copied to this buffer after the first buffer
i_buf2 Third DataBuffer to concatenate; copied to this buffer after the second buffer

Postcondition:
Space is allocated, and data from the 3 DataBuffers is concatenated and copied to this buffer

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.58 uint32_t ecmdDataBuffer::setOr (const ecmdDataBuffer & *i_bufferIn*, uint32_t *i_startbit*, uint32_t *i_len*)

OR data into DataBuffer.

Parameters:
i_bufferIn DataBuffer to OR data from - data is taken left aligned

i_ startbit Start bit to OR to

i_ len Length of bits to OR

Postcondition:
Data is ORed from i_bufferIn to this DataBuffer in specified location

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.59 uint32_t ecmdDataBuffer::setOr (const uint32_t * *i_ datain*, uint32_t *i_ startbit*, uint32_t *i_ len*)

OR data into DataBuffer.

Parameters:
i_ datain uint32_t buffer to OR data from - data is taken left aligned

i_ startbit Start bit to OR to

i_ len Length of bits to OR

Postcondition:
Data is ORed from datain to this DataBuffer in specified location

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.60 uint32_t ecmdDataBuffer::setOr (uint32_t *i_ datain*, uint32_t *i_ startbit*, uint32_t *i_ len*)

OR data into DataBuffer.

Parameters:
i_ datain uint32_t to OR data from - data is taken left aligned

i_ startbit Start bit to OR to

i_ len Length of bits to OR (must be <= 32)

Postcondition:
Data is ORed from datain to this DataBuffer in specified location

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.61 uint32_t ecmdDataBuffer::merge (const ecmdDataBuffer & *i_bufferIn*)

OR data into DataBuffer.

Parameters:
i_bufferIn DataBuffer to OR data from - data is taken left aligned

Postcondition:
Entire data is ORed from bufferIn to this DataBuffer

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.62 uint32_t ecmdDataBuffer::setXor (const ecmdDataBuffer & *i_bufferIn*, uint32_t *i_startbit*, uint32_t *i_len*)

XOR data into DataBuffer.

Parameters:
i_bufferIn DataBuffer to XOR data from - data is taken left aligned
i_startbit Start bit to XOR to
i_len Length of bits to XOR

Postcondition:
Data is XORed from i_bufferIn to this DataBuffer in specified location

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.63 uint32_t ecmdDataBuffer::setXor (const uint32_t * *i_datain*, uint32_t *i_startbit*, uint32_t *i_len*)

XOR data into DataBuffer.

Parameters:
i_datain uint32_t buffer to XOR data from - data is taken left aligned
i_startbit Start bit to XOR to
i_len Length of bits to XOR

Postcondition:
Data is XORed from datain to this DataBuffer in specified location

Return values:
ECMD_DBUF_SUCCESS on success
ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.64 uint32_t ecmdDataBuffer::setXor (uint32_t *i_ datain*, uint32_t *i_ startbit*, uint32_t *i_ len*)

XOR data into DataBuffer.

Parameters:

i_ datain uint32_t to XOR data from - data is taken left aligned

i_ startbit Start bit to XOR to

i_ len Length of bits to XOR (must be <= 32)

Postcondition:

Data is XORed from datain to this DataBuffer in specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.65 uint32_t ecmdDataBuffer::setAnd (const ecmdDataBuffer & *i_ bufferIn*, uint32_t *i_ startbit*, uint32_t *i_ len*)

AND data into DataBuffer.

Parameters:

i_ bufferIn Bitvector to AND data from - data is taken left aligned

i_ startbit Start bit to AND to

i_ len Length of bits to AND

Postcondition:

Data is ANDed from bufferIn to this DataBuffer in specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.66 uint32_t ecmdDataBuffer::setAnd (const uint32_t * *i_ datain*, uint32_t *i_ startbit*, uint32_t *i_ len*)

AND data into DataBuffer.

Parameters:

i_ datain uint32_t buffer to AND data from - data is taken left aligned

i_ startbit Start bit to AND to

i_ len Length of bits to AND

Postcondition:

Data is ANDed from datain to this DataBuffer in specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.67 uint32_t ecmdDataBuffer::setAnd (uint32_t *i_ datain*, uint32_t *i_ startbit*, uint32_t *i_ len*)

AND data into DataBuffer.

Parameters:

i_ datain uint32_t to AND data from - data is taken left aligned

i_ startbit Start bit to AND to

i_ len Length of bits to AND (must be <= 32)

Postcondition:

Data is ANDed from datain to this DataBuffer in specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.68 uint32_t ecmdDataBuffer::copy (ecmdDataBuffer & *o_ copyBuffer*) const

Copy entire contents of this ecmdDataBuffer into o_copyBuffer.

Parameters:

o_ copyBuffer DataBuffer to copy data into

Postcondition:

copyBuffer is allocated, is an exact duplicate of this DataBuffer

Return values:

ECMD_ DBUF_ SUCCESS on success

5.7.3.69 uint32_t ecmdDataBuffer::memCopyIn (const uint32_t * *i_ buf*, uint32_t *i_ bytes*)

Copy buffer into this ecmdDataBuffer.

Parameters:

i_ buf Buffer to copy from

i_ bytes Byte length to copy

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Xstate and Raw buffer are set to value in i_buf for smaller of i_bytes or buffer capacity

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.70 uint32_t ecmdDataBuffer::memCopyOut (uint32_t * *o_ buf*, uint32_t *i_ bytes*) const

Copy DataBuffer into supplied uint32_t buffer.

Parameters:

o_ buf Buffer to copy into - must be pre-allocated

i_ bytes Byte length to copy

Postcondition:

o_buf has contents of databuffer for smaller of i_bytes or buffer capacity

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.71 uint32_t ecmdDataBuffer::flatten (uint8_t * *o_ data*, uint32_t *i_ len*) const

Flatten all the object data into a uint8_t buffer.

Parameters:

o_ data Byte buffer to write the flattened data to - should

i_ len Number of bytes in the o_data buffer

Postcondition:

o_data buffer has a flattened version of the DataBuffer - must be pre-allocated Data format (all in network byte order): First Word: iv_Capacity*32 (in bits) Second Word: iv_NumBits Remaining Words: Buffer data

5.7.3.72 uint32_t ecmdDataBuffer::unflatten (const uint8_t * *i_ data*, uint32_t *i_ len*)

Unflatten object data from a uint8_t buffer into this DataBuffer.

Parameters:

i_ data Byte buffer to read the flattened data from

i_ len Number of bytes in the i_data buffer

Postcondition:

This DataBuffer is allocated and initialized with the unflattened version of i_data Data format (all in network byte order): First Word: iv_Capacity*32 (in bits) Second Word: iv_NumBits Remaining Words: Buffer data

5.7.3.73 uint32_t ecmdDataBuffer::flattenSize (void) const

Return number of bytes needed for a buffer to flatten the object.

Return values:

Number of bytes needed

5.7.3.74 uint32_t ecmdDataBuffer::oddParity (uint32_t *i_start*, uint32_t *i_stop*) const

Generate odd parity over a range of bits.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

Return values:

0 or 1 depending on parity of range

5.7.3.75 uint32_t ecmdDataBuffer::evenParity (uint32_t *i_start*, uint32_t *i_stop*) const

Generate even parity over a range of bits.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

Return values:

0 or 1 depending on parity of range

5.7.3.76 uint32_t ecmdDataBuffer::oddParity (uint32_t *i_start*, uint32_t *i_stop*, uint32_t *i_insertpos*)

Generate odd parity over a range of bits and insert into DataBuffer.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_insertpos Bit position to insert parity

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.77 uint32_t ecmdDataBuffer::evenParity (uint32_t *i_start*, uint32_t *i_stop*, uint32_t *i_insertpos*)

Generate even parity over a range of bits and insert into DataBuffer.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_ insertpos Bit position to insert parity

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.78 std::string ecmdDataBuffer::genHexLeftStr (uint32_t *i_ start*, uint32_t *i_ bitlen*) const

Return Data as a hex left aligned char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

5.7.3.79 std::string ecmdDataBuffer::genHexRightStr (uint32_t *i_ start*, uint32_t *i_ bitlen*) const

Return Data as a hex right aligned char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

5.7.3.80 std::string ecmdDataBuffer::genBinStr (uint32_t *i_ start*, uint32_t *i_ bitlen*) const

Return Data as a binary char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

5.7.3.81 std::string ecmdDataBuffer::genAsciiStr (uint32_t *i_start*, uint32_t *i_bitlen*) const

Return Data as an ASCII char string. If it's out of range, a . is printed.

Parameters:

i_start Start bit of data to convert

i_bitlen Number of consecutive bits to convert

Return values:

String containing requested data

5.7.3.82 std::string ecmdDataBuffer::genHexLeftStr () const

Return entire buffer as a hex left aligned char string.

Return values:

String containing requested data

5.7.3.83 std::string ecmdDataBuffer::genHexRightStr () const

Return entire buffer as a hex right aligned char string.

Return values:

String containing requested data

5.7.3.84 std::string ecmdDataBuffer::genBinStr () const

Return entire buffer as a binary char string.

Return values:

String containing requested data

5.7.3.85 std::string ecmdDataBuffer::genAsciiStr () const

Return Data as an ASCII char string. If it's out of range, a . is printed.

Return values:

String containing requested data

5.7.3.86 std::string ecmdDataBuffer::genXstateStr (uint32_t *i_start*, uint32_t *i_bitlen*) const

Retrieve a section of the Xstate Data.

Parameters:

i_start Start bit of data to retrieve

i_ bitlen Number of consecutive bits to retrieve

Return values:

String containing requested data

5.7.3.87 std::string ecmdDataBuffer::genXstateStr () const

Retrieve entire Xstate Data buffer.

Return values:

String containing requested data

5.7.3.88 uint32_t ecmdDataBuffer::insertFromHexLeft (const char * *i_ hexChars*, uint32_t *i_ start* = 0, uint32_t *i_ length* = 0)

Convert data from a hex left-aligned string and insert it into this data buffer.

Parameters:

i_ hexChars Hex Left-aligned string of data to insert

i_ start Starting position in data buffer to insert to, 0 by default

i_ length Length of data to insert, defaults to length of i_hexChars, zeroes are padded or data dropped from right if necessary

Return values:

ECMD_ DBUF_ INVALID_ DATA_ FORMAT if non-hex chars detected in i_hexChars

ECMD_ SUCCESS on success

non-zero on failure

5.7.3.89 uint32_t ecmdDataBuffer::insertFromHexLeftAndResize (const char * *i_ hexChars*, uint32_t *i_ start* = 0, uint32_t *i_ length* = 0)

Convert data from a hex left-aligned string and insert it into this data buffer - and set's buffer length to size of data.

Parameters:

i_ hexChars Hex Left-aligned string of data to insert

i_ start Starting position in data buffer to insert to, 0 by default

i_ length Length of data to insert, defaults to length of i_hexChars, zeroes are padded or data dropped from right if necessary

Return values:

ECMD_ DBUF_ INVALID_ DATA_ FORMAT if non-hex chars detected in i_hexChars

ECMD_ SUCCESS on success

non-zero on failure

5.7.3.90 uint32_t ecmdDataBuffer::insertFromHexRight (const char * *i_ hexChars*, uint32_t *i_ start* = 0, uint32_t *i_ expectedLength* = 0)

Convert data from a hex right-aligned string and insert it into this data buffer.

Parameters:

i_ hexChars Hex Right-aligned string of data to insert

i_ expectedLength The expected length of the string data, zeros are padded or data dropped from the left if necessary

i_ start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

5.7.3.91 uint32_t ecmdDataBuffer::insertFromHexRightAndResize (const char * *i_ hexChars*, uint32_t *i_ start* = 0, uint32_t *i_ expectedLength* = 0)

Convert data from a hex right-aligned string and insert it into this data buffer - and set's buffer length to size of data.

Parameters:

i_ hexChars Hex Right-aligned string of data to insert

i_ expectedLength The expected length of the string data, zeros are padded or data dropped from the left if necessary

i_ start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

5.7.3.92 uint32_t ecmdDataBuffer::insertFromBin (const char * *i_ binChars*, uint32_t *i_ start* = 0)

Convert data from a binary string and insert it into this data buffer.

Return values:

0 on success- non-zero on failure

Parameters:

i_ binChars String of 0's and 1's to insert

i_ start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-binary chars detected in i_binChars

ECMD_SUCCESS on success

non-zero on failure

5.7.3.93 uint32_t ecmdDataBuffer::insertFromBinAndResize (const char * *i_binChars*, uint32_t *i_start* = 0)

Convert data from a binary string and insert it into this data buffer - and set's buffer length to size of data.

Return values:

0 on success- non-zero on failure

Parameters:

i_binChars String of 0's and 1's to insert

i_start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-binary chars detected in i_binChars

ECMD_SUCCESS on success

non-zero on failure

5.7.3.94 uint32_t ecmdDataBuffer::enableXstateBuffer ()

Initializes the X-state buffer, from then on all changes are reflected in Xstate.

Postcondition:

Xstate buffer is created and initialized to value of current raw buffer

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_INIT_FAIL failure occurred allocating X-state array

ECMD_DBUF_NOT_OWNER when called on buffer not owned

5.7.3.95 uint32_t ecmdDataBuffer::disableXstateBuffer ()

Removes the X-state buffer, from then on no changes are made to Xstate.

Postcondition:

Xstate buffer is deallocated

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_NOT_OWNER when called on buffer not owned

5.7.3.96 bool ecmdDataBuffer::isXstateEnabled () const

Query to find out if this buffer has X-states enabled.

Return values:
true if the Xstate buffer is active
false if the Xstate buffer is not active

5.7.3.97 uint32_t ecmdDataBuffer::flushToX (char *i_value*)

Load entire buffer with an X-state value.

Parameters:
i_value Value to load into buffer

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.98 bool ecmdDataBuffer::hasXstate () const

Check Entire buffer for any X-state values.

Return values:
true if xstate found false if none

5.7.3.99 bool ecmdDataBuffer::hasXstate (uint32_t *i_start*, uint32_t *i_length*) const

Check section of buffer for any X-state values.

Parameters:
i_start Start bit to test
i_length Number of consecutive bits to test

Return values:
true if xstate found false if none

5.7.3.100 char ecmdDataBuffer::getXstate (uint32_t *i_bit*) const

Retrieve an Xstate value from the buffer.

Parameters:
i_bit Bit to retrieve

NOTE - To retrieve multiple bits use genXstateStr

5.7.3.101 uint32_t ecmdDataBuffer::setXstate (uint32_t *i_ bit*, char *i_ value*)

Set an Xstate value in the buffer.

Parameters:

i_ bit Bit to set

i_ value Xstate value to set

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.102 uint32_t ecmdDataBuffer::setXstate (uint32_t *i_ bit*, char *i_ value*, uint32_t *i_ length*)

Set an Xstate value in the buffer.

Parameters:

i_ bit Bit to set

i_ value Xstate value to set

i_ length Number of consecutive bits to set to i_value

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.103 uint32_t ecmdDataBuffer::setXstate (uint32_t *i_ bitoffset*, const char * *i_ datastr*)

Set a range of Xstate values in buffer.

Parameters:

i_ bitoffset bit in buffer to start inserting

i_ datastr Character value to set bit - can be "0XX0", "1", "X"

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

5.7.3.104 uint32_t ecmdDataBuffer::memCopyInXstate (const char * *i_ buf*, uint32_t *i_ bytes*)

Copy buffer into the Xstate data of this ecmdDataBuffer.

Parameters:

i_ buf Buffer to copy from

i_bytes Byte length to copy (char length)

Postcondition:
Xstate and Raw buffer are set to value in i_buf for smaller of i_bytes or buffer capacity

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.105 uint32_t ecmdDataBuffer::memCopyOutXstate (char * *o_buf*, uint32_t *i_bytes*) const

Copy DataBuffer into supplied char buffer from Xstate data.

Parameters:
o_buf Buffer to copy into - must be pre-allocated

i_bytes Byte length to copy (char length)

Postcondition:
o_buf has contents of databuffer for smaller of i_bytes or buffer capacity

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

5.7.3.106 uint32_t ecmdDataBuffer::writeFile (const char * *i_filename*, ecmdFormatType_t *i_format*, const char * *i_facName* = NULL)

Write buffer out into a file in the format specified.

Parameters:
i_filename file to write to

i_format format to write in

i_property name(len <= 200 chars) associated with the databuffer(e.g. ringname/spyname)-default is NULL

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FOPEN_FAIL Unable to open the file for write

ECMD_DBUF_XSTATE_ERROR If Xstate values are detected on non-Xstate format request

5.7.3.107 uint32_t ecmdDataBuffer::writeFileMultiple (const char * *i_filename*, ecmdFormatType_t *i_format*, ecmdWriteMode_t *i_mode*, uint32_t & *o_dataNumber*, const char * *i_property* = NULL)

Writes/Appends buffer out into a file in the format specified.

Parameters:

i_filename file to write to

i_format format to write in

i_mode mode to open the file in

o_dataNumber the sequence number for this data, used by readFileMultiple to pick the right databuffer

i_property name(len <= 200 chars) associated with the databuffer(e.g. ringname/spyname)-default is NULL

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FOPEN_FAIL Unable to open the file for write

ECMD_DBUF_XSTATE_ERROR If Xstate values are detected on non-Xstate format request ECMD_SAVE_FORMAT_BINARY_DATA not accepted when ecmdWriteMode_t is ECMD_APPEND_MODE

5.7.3.108 uint32_t ecmdDataBuffer::writeFileStream (std::ostream & *o_filestream*)

Write buffer out into the stream in ECMD_SAVE_FORMAT_BINARY_DATA format.

Parameters:

o_filestream output stream to write to

Return values:

ECMD_DBUF_SUCCESS on success

non-zero on failure

5.7.3.109 uint32_t ecmdDataBuffer::readFile (const char * *i_filename*, ecmdFormatType_t *i_format*, std::string * *o_property* = NULL)

Read data from the file into the buffer.

Parameters:

i_filename to read from

i_format data format to expect in the file

o_property string associated with the databuffer read from the file(if present)

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FILE_FORMAT_MISMATCH specified format not found in the file

ECMD_DBUF_FOPEN_FAIL Unable to open the file for read

ECMD_DBUF_XSTATE_ERROR If XState format is requested when XState is not defined for the configuration

5.7.3.110 uint32_t ecmdDataBuffer::readFileMultiple (const char * *i_filename*, ecmdFormatType_t *i_format*, uint32_t *i_dataNumber* = 0, std::string * *o_property* = NULL)

Read data from the file into the buffer.

Parameters:

i_filename to read from

i_format data format to expect in the file

i_dataNumber data requested in case of multiple databuffers

o_property string associated with the databuffer read from the file(if present)

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FILE_FORMAT_MISMATCH specified format not found in the file

ECMD_DBUF_FOPEN_FAIL Unable to open the file for read

ECMD_DBUF_XSTATE_ERROR If XState format is requested when XState is not defined for the configuration

ECMD_DBUF_DATANUMBER_NOT_FOUND If requested i_dataNumber is not available in file ECMD_SAVE_FORMAT_BINARY_DATA not accepted when i_dataNumber != 0

5.7.3.111 uint32_t ecmdDataBuffer::queryNumOfBuffers (const char * *i_filename*, ecmdFormatType_t *i_format*, uint32_t & *o_num*)

Get the number of databuffers stored in the file created by writeFile/writeFileMultiple.

Parameters:

i_filename to read from

i_format data format to expect in the file

o_num number of data buffers stored in the file

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FILE_FORMAT_MISMATCH specified format not found in the file

ECMD_DBUF_FOPEN_FAIL Unable to open the file for read

5.7.3.112 uint32_t ecmdDataBuffer::readFileStream (std::istream & *i_filestream*, uint32_t *i_bitlength*)

Read data from the stream (in ECMD_SAVE_FORMAT_BINARY_DATA format) into the buffer.

Parameters:

i_filestream input stream to read from

i_bitlength used to est. the number of bytes to read from the stream

Return values:

ECMD_DBUF_SUCCESS on success

non-zero on failure

5.7.3.113 uint32_t ecmdDataBuffer::shareBuffer (ecmdDataBuffer * *i_sharingBuffer*)

This function will take the passed in buffer, delete any current data it holds, and point its data var to that which is owned by the one being called with. It will not have iv_UserOwned flag set, so it should not delete the buffer it points to, nor resize it, but it can alter the data. The use of this function is for caching data for reads.

Parameters:
i_sharingBuffer input buffer

Return values:
ECMD_DBUF_SUCCESS on success

5.7.3.114 void ecmdDataBuffer::queryErrorState (uint32_t & *o_errorState*)

This function returns the stored error state that could have been caused by any number of previous operations on the buffer.

Parameters:
o_errorState Stored Error state

5.7.3.115 int ecmdDataBuffer::operator== (const ecmdDataBuffer & *other*) const

Overload the == operator.

5.7.3.116 int ecmdDataBuffer::operator!= (const ecmdDataBuffer & *other*) const

Overload the != operator.

5.7.3.117 ecmdDataBuffer ecmdDataBuffer::operator & (const ecmdDataBuffer & *other*) const

Overload the & operator.

5.7.3.118 ecmdDataBuffer ecmdDataBuffer::operator| (const ecmdDataBuffer & *other*) const

Overload the | operator.

5.7.3.119 uint32_t ecmdDataBuffer::fillDataStr (char *fillChar*) [protected]

5.7.4 Friends And Related Function Documentation

5.7.4.1 friend class ecmdDataBufferImplementationHelper [friend]

5.7.5 Member Data Documentation

5.7.5.1 uint32_t ecmdDataBuffer::iv_Capacity [protected]

Actual buffer capacity - always >= iv_NumWords.

5.7.5.2 uint32_t ecmdDataBuffer::iv_NumWords [protected]

Specified buffer size rounded to next word.

5.7.5.3 uint32_t ecmdDataBuffer::iv_NumBits [protected]

Specified buffer size in bits.

5.7.5.4 uint32_t* ecmdDataBuffer::iv_Data [protected]

Pointer to buffer inside iv_RealData.

5.7.5.5 uint32_t* ecmdDataBuffer::iv_RealData [protected]

Real buffer - with header and tail.

5.7.5.6 bool ecmdDataBuffer::iv_UserOwned [protected]

Whether or not this buffer owns the data.

5.7.5.7 bool ecmdDataBuffer::iv_BufferOptimizable [protected]

Whether or not this is an optimizable buffer.

5.7.5.8 char* ecmdDataBuffer::iv_DataStr [protected]

5.7.5.9 bool ecmdDataBuffer::iv_XstateEnabled [protected]

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

- **ecmdDataBuffer.H**

5.8 ecmdDataBufferImplementationHelper Class Reference

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 24), this CAN NOT be used by any eCMD client or data corruption will occur.

`#include <ecmdDataBuffer.H>`

Static Public Member Functions

- uint32_t * **getDataPtr** (void *i_buffer)
- void **applyRawBufferToXstate** (void *i_buffer)

5.8.1 Detailed Description

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 24), this CAN NOT be used by any eCMD client or data corruption will occur.

5.8.2 Member Function Documentation

5.8.2.1 uint32_t* ecmdDataBufferImplementationHelper::getDataPtr (void * *i_buffer*) [static]

5.8.2.2 void ecmdDataBufferImplementationHelper::applyRawBufferToXstate (void * *i_buffer*) [static]

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

- **ecmdDataBuffer.H**

5.9 ecmdDllInfo Struct Reference

This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with.

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdDllType_t dllType**
  Dll instance type running.
- **ecmdDllProduct_t dllProduct**
  Dll product supported.
- std::string **dllProductType**
  Dll product type currently configured.
- **ecmdDllEnv_t dllEnv**
  Dll environment (Simulation vs Hardware).
- std::string **dllBuildDate**
  Date the Dll was built.
- std::string **dllCapiVersion**
  should be set to ECMD_CAPI_VERSION
- std::string **dllBuildInfo**
  Any additional info the Dll/Plugin would like to pass.

5.9.1 Detailed Description

This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with.

5.9.2 Member Data Documentation

5.9.2.1 ecmdDllType_t ecmdDllInfo::dllType

Dll instance type running.

5.9.2.2 ecmdDllProduct_t ecmdDllInfo::dllProduct

Dll product supported.

5.9.2.3 std::string ecmdDllInfo::dllProductType

Dll product type currently configured.

5.9.2.4 ecmdDllEnv_t ecmdDllInfo::dllEnv

Dll environment (Simulation vs Hardware).

5.9.2.5 std::string ecmdDllInfo::dllBuildDate

Date the Dll was built.

5.9.2.6 std::string ecmdDllInfo::dllCapiVersion

should be set to ECMD_CAPI_VERSION

5.9.2.7 std::string ecmdDllInfo::dllBuildInfo

Any additional info the Dll/Plugin would like to pass.

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

- **ecmdStructs.H**

5.10 ecmdIndexEntry Struct Reference

Used by get/put Gpr/Fpr Multiple function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- int **index**

  Index of entry.

- **ecmdDataBuffer buffer**

  Data to/from entry.

- uint32_t **rc**

  Error code in retrieving this entry.

5.10.1 Detailed Description

Used by get/put Gpr/Fpr Multiple function to pass data.

5.10.2 Member Data Documentation

5.10.2.1 int ecmdIndexEntry::index

Index of entry.

5.10.2.2 ecmdDataBuffer ecmdIndexEntry::buffer

Data to/from entry.

5.10.2.3 uint32_t ecmdIndexEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.11 ecmdIndexVectorEntry Struct Reference

Used by ???? function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- int **index**

  Index of entry.

- std::vector< **ecmdDataBuffer** > **buffer**

  Vector of data to/from entry.

- uint32_t **rc**

  Error code in retrieving this entry.

5.11.1 Detailed Description

Used by ???? function to pass data.

5.11.2 Member Data Documentation

5.11.2.1 int ecmdIndexVectorEntry::index

Index of entry.

5.11.2.2 std::vector<ecmdDataBuffer> ecmdIndexVectorEntry::buffer

Vector of data to/from entry.

5.11.2.3 uint32_t ecmdIndexVectorEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.12 ecmdLatchData Struct Reference

Used for the ecmdQueryLatch function to return latch info.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **latchName**
  (Detail: Low) Latch Name
- std::string **ringName**
  (Detail: Low) Ring that this latch belongs to
- int **bitLength**
  (Detail: Low) length of latch, sum of all the parts
- bool **isCoreRelated**
  (Detail: Low) This latch is related to the core level of a chip
- std::string **clockDomain**
  (Detail: High) Clock domain this latch belongs to
- **ecmdClockState_t clockState**
  (Detail: High) Required clock state to access this latch

5.12.1 Detailed Description

Used for the ecmdQueryLatch function to return latch info.

5.12.2 Member Data Documentation

5.12.2.1 std::string ecmdLatchData::latchName

(Detail: Low) Latch Name

5.12.2.2 std::string ecmdLatchData::ringName

(Detail: Low) Ring that this latch belongs to

5.12.2.3 int ecmdLatchData::bitLength

(Detail: Low) length of latch, sum of all the parts

5.12.2.4 bool ecmdLatchData::isCoreRelated

(Detail: Low) This latch is related to the core level of a chip

5.12.2.5 std::string ecmdLatchData::clockDomain

(Detail: High) Clock domain this latch belongs to

5.12.2.6 ecmdClockState_t ecmdLatchData::clockState

(Detail: High) Required clock state to access this latch

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

- **ecmdStructs.H**

5.13 ecmdLatchEntry Struct Reference

Used by getlatch function to return data.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **latchName**
  Latch name of entry.
- std::string **ringName**
  Ring that latch came from.
- **ecmdDataBuffer buffer**
  Latch data.
- int **latchStartBit**
  Start bit of data inside latch.
- int **latchEndBit**
  End bit of data inside latch.
- uint32_t **rc**
  Error code in retrieving this entry.

5.13.1 Detailed Description

Used by getlatch function to return data.

5.13.2 Member Data Documentation

5.13.2.1 std::string ecmdLatchEntry::latchName

Latch name of entry.

5.13.2.2 std::string ecmdLatchEntry::ringName

Ring that latch came from.

5.13.2.3 ecmdDataBuffer ecmdLatchEntry::buffer

Latch data.

5.13.2.4 int ecmdLatchEntry::latchStartBit

Start bit of data inside latch.

5.13.2.5 int ecmdLatchEntry::latchEndBit

End bit of data inside latch.

5.13.2.6 uint32_t ecmdLatchEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.14 ecmdLooperData Struct Reference

Used internally by ecmdConfigLooper to store looping state information.

`#include <ecmdUtils.H>`

Public Attributes

- bool **ecmdLooperInitFlag**
  Is fresh ?
- bool **ecmdUseUnitid**
  This looper is looping on unitid targets not config data.
- **ecmdConfigLoopMode_t ecmdLoopMode**
  Is this a variable depth or static loop?
- **ecmdQueryData ecmdSystemConfigData**
  Config data queried from the system.
- std::list< **ecmdCageData** >::iterator **ecmdCurCage**
  Pointer to current Cage.
- std::list< **ecmdNodeData** >::iterator **ecmdCurNode**
  Pointer to current Node.
- std::list< **ecmdSlotData** >::iterator **ecmdCurSlot**
  Pointer to current Slot.
- std::list< **ecmdChipData** >::iterator **ecmdCurChip**
  Pointer to current Chip.
- std::list< **ecmdCoreData** >::iterator **ecmdCurCore**
  Pointer to current Core.
- std::list< **ecmdThreadData** >::iterator **ecmdCurThread**
  Pointer to current Thread.
- **ecmdChipTarget prevTarget**
  Pointer to previous target.
- std::list< **ecmdChipTarget** > **unitIdTargets**
  List of targets if looping on a unitid.
- std::list< **ecmdChipTarget** >::iterator **curUnitIdTarget**
  Pointer to current unitid target.

5.14.1 Detailed Description

Used internally by ecmdConfigLooper to store looping state information.

5.14.2 Member Data Documentation

5.14.2.1 bool ecmdLooperData::ecmdLooperInitFlag

Is fresh ?

5.14.2.2 bool ecmdLooperData::ecmdUseUnitid

This looper is looping on unitid targets not config data.

5.14.2.3 ecmdConfigLoopMode_t ecmdLooperData::ecmdLoopMode

Is this a variable depth or static loop?

5.14.2.4 ecmdQueryData ecmdLooperData::ecmdSystemConfigData

Config data queried from the system.

5.14.2.5 std::list<ecmdCageData>::iterator ecmdLooperData::ecmdCurCage

Pointer to current Cage.

5.14.2.6 std::list<ecmdNodeData>::iterator ecmdLooperData::ecmdCurNode

Pointer to current Node.

5.14.2.7 std::list<ecmdSlotData>::iterator ecmdLooperData::ecmdCurSlot

Pointer to current Slot.

5.14.2.8 std::list<ecmdChipData>::iterator ecmdLooperData::ecmdCurChip

Pointer to current Chip.

5.14.2.9 std::list<ecmdCoreData>::iterator ecmdLooperData::ecmdCurCore

Pointer to current Core.

5.14.2.10 std::list<ecmdThreadData>::iterator ecmdLooperData::ecmdCurThread

Pointer to current Thread.

5.14.2.11 ecmdChipTarget ecmdLooperData::prevTarget

Pointer to previous target.

5.14.2.12 std::list<ecmdChipTarget> ecmdLooperData::unitIdTargets

List of targets if looping on a unitid.

5.14.2.13 std::list<ecmdChipTarget>::iterator ecmdLooperData::curUnitIdTarget

Pointer to current unitid target.

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

- **ecmdUtils.H**

5.15 ecmdMemoryEntry Struct Reference

Used by ecmdReadDcard.

`#include <ecmdStructs.H>`

Public Attributes

- uint64_t **address**
- **ecmdDataBuffer data**
- **ecmdDataBuffer tags**

5.15.1 Detailed Description

Used by ecmdReadDcard.

5.15.2 Member Data Documentation

5.15.2.1 uint64_t ecmdMemoryEntry::address

5.15.2.2 ecmdDataBuffer ecmdMemoryEntry::data

5.15.2.3 ecmdDataBuffer ecmdMemoryEntry::tags

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

- **ecmdStructs.H**

5.16 ecmdNameEntry Struct Reference

Used by get/putSprMultiple function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **name**

  Name of entry.

- **ecmdDataBuffer buffer**

  Data to/from entry.

- uint32_t **rc**

  Error code in retrieving this entry.

5.16.1 Detailed Description

Used by get/putSprMultiple function to pass data.

5.16.2 Member Data Documentation

5.16.2.1 std::string ecmdNameEntry::name

Name of entry.

5.16.2.2 ecmdDataBuffer ecmdNameEntry::buffer

Data to/from entry.

5.16.2.3 uint32_t ecmdNameEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.17 ecmdNameVectorEntry Struct Reference

Used by getTraceArrayMultiple function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **name**

  Name of entry.

- std::vector< **ecmdDataBuffer** > **buffer**

  Vector of data to/from entry.

- uint32_t **rc**

  Error code in retrieving this entry.

5.17.1 Detailed Description

Used by getTraceArrayMultiple function to pass data.

5.17.2 Member Data Documentation

5.17.2.1 std::string ecmdNameVectorEntry::name

Name of entry.

5.17.2.2 std::vector<ecmdDataBuffer> ecmdNameVectorEntry::buffer

Vector of data to/from entry.

5.17.2.3 uint32_t ecmdNameVectorEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.18 ecmdNodeData Struct Reference

Used for the ecmdQueryConfig function to return node data.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **nodeId**
  (Detail: Low) Node number of this entry
- uint32_t **unitId**
  (Detail: High) Unit Id of this entry
- std::list< **ecmdSlotData** > **slotData**
  (Detail: Low) List of all slots requested in this node - in numerical order by slotId

5.18.1 Detailed Description

Used for the ecmdQueryConfig function to return node data.

Operators Supported : <

5.18.2 Member Data Documentation

5.18.2.1 uint32_t ecmdNodeData::nodeId

(Detail: Low) Node number of this entry

5.18.2.2 uint32_t ecmdNodeData::unitId

(Detail: High) Unit Id of this entry

5.18.2.3 std::list<ecmdSlotData> ecmdNodeData::slotData

(Detail: Low) List of all slots requested in this node - in numerical order by slotId

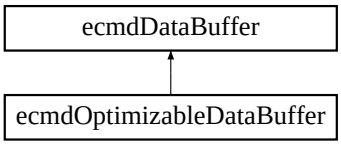
The documentation for this struct was generated from the following file:

- **ecmdStructs.H**

5.19 ecmdOptimizableDataBuffer Class Reference

`#include <ecmdDataBuffer.H>`

Inheritance diagram for ecmdOptimizableDataBuffer::



Public Member Functions

- **ecmdOptimizableDataBuffer** ()
  Default constructor for ecmdOptimizableDataBuffer class.
- **ecmdOptimizableDataBuffer** (uint32_t numBits)
  Constructor with bit length specified.
- **~ecmdOptimizableDataBuffer** ()
  Destructor for ecmdOptimizableDataBuffer class.

5.19.1 Constructor & Destructor Documentation

5.19.1.1 ecmdOptimizableDataBuffer::ecmdOptimizableDataBuffer ()

Default constructor for ecmdOptimizableDataBuffer class.

5.19.1.2 ecmdOptimizableDataBuffer::ecmdOptimizableDataBuffer (uint32_t *numBits*)

Constructor with bit length specified.

5.19.1.3 ecmdOptimizableDataBuffer::~ecmdOptimizableDataBuffer () [inline]

Destructor for ecmdOptimizableDataBuffer class.

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

- **ecmdDataBuffer.H**

5.20 ecmdProcRegisterInfo Struct Reference

Used by ecmdQueryProcRegisterInfo function to return data about a Architected register.

`#include <ecmdStructs.H>`

Public Attributes

- int **bitLength**
  Bit length of each entry.
- int **totalEntries**
  Total number of entries available.
- bool **threadReplicated**
  Register is replicated for each thread.

5.20.1 Detailed Description

Used by ecmdQueryProcRegisterInfo function to return data about a Architected register.

5.20.2 Member Data Documentation

5.20.2.1 int ecmdProcRegisterInfo::bitLength

Bit length of each entry.

5.20.2.2 int ecmdProcRegisterInfo::totalEntries

Total number of entries available.

5.20.2.3 bool ecmdProcRegisterInfo::threadReplicated

Register is replicated for each thread.

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

- **ecmdStructs.H**

5.21 ecmdQueryData Struct Reference

Used by the ecmdQueryConfig function to return data.

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdQueryDetail_t detailLevel**
  (Detail: Low) This is set to the detail level of the data contained within
- std::list< **ecmdCageData** > **cageData**
  (Detail: Low) List of all cages in the system - in nummerical order by cageId

5.21.1 Detailed Description

Used by the ecmdQueryConfig function to return data.

5.21.2 Member Data Documentation

5.21.2.1 ecmdQueryDetail_t ecmdQueryData::detailLevel

(Detail: Low) This is set to the detail level of the data contained within

5.21.2.2 std::list<ecmdCageData> ecmdQueryData::cageData

(Detail: Low) List of all cages in the system - in nummerical order by cageId

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

- **ecmdStructs.H**

5.22 ecmdRingData Struct Reference

Used for the ecmdQueryRing function to return ring info.

`#include <ecmdStructs.H>`

Public Attributes

- std::list< std::string > **ringNames**
  (Detail: Low) Names used to reference this ring
- uint32_t **address**
  (Detail: Low) Address modifier
- int **bitLength**
  (Detail: Low) length of ring
- bool **hasInversionMask**
  (Detail: High) Ring has an inversion mask applied before scanning
- bool **supportsBroadsideLoad**
  (Detail: High) This ring supports broadside load in simulation
- bool **isCheckable**
  (Detail: High) This ring can be run through the check_rings command
- bool **isCoreRelated**
  (Detail: Low) This ring is related to the core level of a chip
- std::string **clockDomain**
  (Detail: High) Clock domain this ring belongs to
- **ecmdClockState_t clockState**
  (Detail: High) Required clock state to access this ring

5.22.1 Detailed Description

Used for the ecmdQueryRing function to return ring info.

5.22.2 Member Data Documentation

5.22.2.1 std::list<std::string> ecmdRingData::ringNames

(Detail: Low) Names used to reference this ring

5.22.2.2 uint32_t ecmdRingData::address

(Detail: Low) Address modifier

5.22.2.3 int ecmdRingData::bitLength

(Detail: Low) length of ring

5.22.2.4 bool ecmdRingData::hasInversionMask

(Detail: High) Ring has an inversion mask applied before scanning

5.22.2.5 bool ecmdRingData::supportsBroadsideLoad

(Detail: High) This ring supports broadside load in simulation

5.22.2.6 bool ecmdRingData::isCheckable

(Detail: High) This ring can be run through the check_rings command

5.22.2.7 bool ecmdRingData::isCoreRelated

(Detail: Low) This ring is related to the core level of a chip

5.22.2.8 std::string ecmdRingData::clockDomain

(Detail: High) Clock domain this ring belongs to

5.22.2.9 ecmdClockState_t ecmdRingData::clockState

(Detail: High) Required clock state to access this ring
The documentation for this struct was generated from the following file:

- **ecmdStructs.H**

5.23 ecmdScomData Struct Reference

Used for the ecmdQueryScom function to return scom info.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **address**
  (Detail: Low) Scom Address
- bool **isCoreRelated**
  (Detail: Low) This scom is related to the core level of a chip
- std::string **clockDomain**
  (Detail: High) Clock domain this scom belongs to
- **ecmdClockState_t clockState**
  (Detail: High) Required clock state to access this scom

5.23.1 Detailed Description

Used for the ecmdQueryScom function to return scom info.

5.23.2 Member Data Documentation

5.23.2.1 uint32_t ecmdScomData::address

(Detail: Low) Scom Address

5.23.2.2 bool ecmdScomData::isCoreRelated

(Detail: Low) This scom is related to the core level of a chip

5.23.2.3 std::string ecmdScomData::clockDomain

(Detail: High) Clock domain this scom belongs to

5.23.2.4 ecmdClockState_t ecmdScomData::clockState

(Detail: High) Required clock state to access this scom

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

- **ecmdStructs.H**

5.24 ecmdSimModelInfo Struct Reference

Used by simGetModelInfo.

`#include <ecmdStructs.H>`

Public Attributes

- char **modelname** [255]
- char **modeldate** [255]
- char **modeltime** [255]
- char **multivalue**

  !=0 multivalue, ==0 2-state

5.24.1 Detailed Description

Used by simGetModelInfo.

5.24.2 Member Data Documentation

5.24.2.1 char ecmdSimModelInfo::modelname[255]

5.24.2.2 char ecmdSimModelInfo::modeldate[255]

5.24.2.3 char ecmdSimModelInfo::modeltime[255]

5.24.2.4 char ecmdSimModelInfo::multivalue

!=0 multivalue, ==0 2-state

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

- **ecmdStructs.H**

5.25 ecmdSlotData Struct Reference

Used for the ecmdQueryConfig function to return slot data.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **slotId**
  (Detail: Low) Slot number of this entry
- uint32_t **unitId**
  (Detail: High) Unit Id of this entry
- std::list< **ecmdChipData** > **chipData**
  (Detail: Low) List of all chips requested in this slot - in order by chipType and pos

5.25.1 Detailed Description

Used for the ecmdQueryConfig function to return slot data.

Operators Supported : <

5.25.2 Member Data Documentation

5.25.2.1 uint32_t ecmdSlotData::slotId

(Detail: Low) Slot number of this entry

5.25.2.2 uint32_t ecmdSlotData::unitId

(Detail: High) Unit Id of this entry

5.25.2.3 std::list<ecmdChipData> ecmdSlotData::chipData

(Detail: Low) List of all chips requested in this slot - in order by chipType and pos

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

- **ecmdStructs.H**

5.26 ecmdSpyData Struct Reference

Used for the ecmdQuerySpy function to return spy info.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **spyName**

  Names used to reference this spy.

- int **bitLength**

  length of spy

- **ecmdSpyType_t spyType**

  Type of spy.

- bool **isEccChecked**

  This spy affects some ECC groupings.

- bool **isEnumerated**

  This spy has enumerated values.

- bool **isCoreRelated**

  This spy is related to the core level of a chip.

- std::string **clockDomain**

  Clock domain this spy belongs to.

- **ecmdClockState_t clockState**

  Required clock state to access this spy.

- std::list< std::string > **enums**

  Possible enum values for Spy - I/P Can only provide this on a client, not on the FSP.

- std::list< std::string > **epCheckers**

  Possible epChecker names affected by this Spy.

5.26.1 Detailed Description

Used for the ecmdQuerySpy function to return spy info.

5.26.2 Member Data Documentation

5.26.2.1 std::string ecmdSpyData::spyName

Names used to reference this spy.

5.26.2.2 int ecmdSpyData::bitLength

length of spy

5.26.2.3 ecmdSpyType_t ecmdSpyData::spyType

Type of spy.

5.26.2.4 bool ecmdSpyData::isEccChecked

This spy affects some ECC groupings.

5.26.2.5 bool ecmdSpyData::isEnumerated

This spy has enumerated values.

5.26.2.6 bool ecmdSpyData::isCoreRelated

This spy is related to the core level of a chip.

5.26.2.7 std::string ecmdSpyData::clockDomain

Clock domain this spy belongs to.

5.26.2.8 ecmdClockState_t ecmdSpyData::clockState

Required clock state to access this spy.

5.26.2.9 std::list<std::string> ecmdSpyData::enums

Possible enum values for Spy - I/P Can only provide this on a client, not on the FSP.

5.26.2.10 std::list<std::string> ecmdSpyData::epCheckers

Possible epChecker names affected by this Spy.

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

- **ecmdStructs.H**

5.27 ecmdSpyGroupData Struct Reference

Used by get/putspy function to create the return data from a group.

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdDataBuffer extractBuffer**

  The data read from the ring buffer.

- **ecmdDataBuffer deadbitsMask**

  A mask of the bits that were deadbits in that buffer.

5.27.1 Detailed Description

Used by get/putspy function to create the return data from a group.

5.27.2 Member Data Documentation

5.27.2.1 ecmdDataBuffer ecmdSpyGroupData::extractBuffer

The data read from the ring buffer.

5.27.2.2 ecmdDataBuffer ecmdSpyGroupData::deadbitsMask

A mask of the bits that were deadbits in that buffer.

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

- **ecmdStructs.H**

5.28 ecmdThreadData Struct Reference

Used for the ecmdQueryConfig function to return thread data.

`#include <ecmdStructs.H>`

Public Attributes

- uint8_t **threadId**
  (Detail: Low) Thread number of this entry
- uint32_t **unitId**
  (Detail: High) Unit Id of this entry

5.28.1 Detailed Description

Used for the ecmdQueryConfig function to return thread data.

Operators Supported : <

5.28.2 Member Data Documentation

5.28.2.1 uint8_t ecmdThreadData::threadId

(Detail: Low) Thread number of this entry

5.28.2.2 uint32_t ecmdThreadData::unitId

(Detail: High) Unit Id of this entry

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

- **ecmdStructs.H**

5.29 ecmdTraceArrayData Struct Reference

Used for the ecmdQueryTraceArray function to return trace array info.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **traceArrayName**
  (Detail: Low) Name of Trace array
- int **length**
  (Detail: Low) Length of trace array (number of entries)
- int **width**
  (Detail: Low) Bit width of trace array entry
- bool **isCoreRelated**
  (Detail: Low) This tracearray is related to the core level of a chip
- std::string **clockDomain**
  (Detail: High) Clock domain this array belongs to
- **ecmdClockState_t clockState**
  (Detail: High) Required clock state to access this array

5.29.1 Detailed Description

Used for the ecmdQueryTraceArray function to return trace array info.

5.29.2 Member Data Documentation

5.29.2.1 std::string ecmdTraceArrayData::traceArrayName

(Detail: Low) Name of Trace array

5.29.2.2 int ecmdTraceArrayData::length

(Detail: Low) Length of trace array (number of entries)

5.29.2.3 int ecmdTraceArrayData::width

(Detail: Low) Bit width of trace array entry

5.29.2.4 bool ecmdTraceArrayData::isCoreRelated

(Detail: Low) This tracearray is related to the core level of a chip

5.29.2.5 std::string ecmdTraceArrayData::clockDomain

(Detail: High) Clock domain this array belongs to

5.29.2.6 ecmdClockState_t ecmdTraceArrayData::clockState

(Detail: High) Required clock state to access this array

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

- **ecmdStructs.H**

Chapter 6

eCMD Perl Module Version Development File Documentation

6.1 cipClientPerlapi.H File Reference

Cronus & IP eCMD Perlapi Extension.

```
#include <ecmdReturnCodes.H>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <cipStructs.H>
```

Load/Unload Functions

- int **cipInitExtension** (const char *i_clientVersion)
  Initialize eCMD CIP Extension DLL.

Processor Functions

- uint32_t **cipStartInstructions** (**ecmdChipTarget** &i_target)
  Start Instructions.
- uint32_t **cipStartAllInstructions** ()
  Start Instructions on all configured processors.
- uint32_t **cipStartInstructionsSreset** (**ecmdChipTarget** &i_target)
  Start Instructions using an S-Reset.
- uint32_t **cipStartAllInstructionsSreset** ()
  Start Instructions on all configured processors using an S-Reset.
- uint32_t **cipStopInstructions** (**ecmdChipTarget** &i_target)
  Stop Instructions.

- uint32_t **cipStopAllInstructions** ()

  Stop All Instructions.

- uint32_t **cipStepInstructions** (**ecmdChipTarget** &i_target, uint32_t i_steps)

  Step Instructions.

- uint32_t **cipSetBreakpoint** (**ecmdChipTarget** &i_target, uint64_t i_address, ecmdBreakpointType_t &i_type)

  Set a hardware breakpoint in Processor using a real address.

- uint32_t **cipClearBreakpoint** (**ecmdChipTarget** &i_target, uint64_t i_address, ecmdBreakpointType_t &i_type)

  Clear a hardware breakpoint from Processor using a real address.

- uint32_t **cipGetVr** (**ecmdChipTarget** &i_target, uint32_t i_vrNum, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected VMX Register (VR) into the data buffer.

- uint32_t **cipGetVrMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &io_entries)

  Reads the selected Processor Architected VMX Register (VR) into the data buffers.

- uint32_t **cipPutVr** (**ecmdChipTarget** &i_target, uint32_t i_vrNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected VMX Register (VR).

- uint32_t **cipPutVrMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &i_entries)

  Writes the data buffer into the selected Processor Architected VMX Register (VR).

Memory Functions

- uint32_t **cipGetMemProc** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_memoryData, **ecmdDataBuffer** &o_memoryTags, **ecmdDataBuffer** &o_memoryEcc, **ecmdDataBuffer** &o_memoryEccError)

  Reads System Mainstore through the processor chip using a real address.

- uint32_t **cipPutMemProc** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_memoryData, **ecmdDataBuffer** &i_memoryTags, **ecmdDataBuffer** &i_memoryErrorInject)

  Writes System Mainstore through the processor chip using a real address.

- uint32_t **cipGetMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_memoryData, **ecmdDataBuffer** &o_memoryTags, **ecmdDataBuffer** &o_memoryEcc, **ecmdDataBuffer** &o_memoryEccError, **ecmdDataBuffer** &o_memorySpareBits)

  Reads System Mainstore through the memory controller using a real address.

- uint32_t **cipPutMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_memoryData, **ecmdDataBuffer** &i_memoryTags, **ecmdDataBuffer** &i_memoryEcc, **ecmdDataBuffer** &i_memorySpareBits, **ecmdDataBuffer** &i_memoryErrorInject)

  Writes System Mainstore through the memory controller using a real address.

6.1.1 Detailed Description

Cronus & IP eCMD Perlapi Extension.

Extension Owner : Chris Engel

6.1.2 Function Documentation

6.1.2.1 int cipInitExtension (const char * *i_clientVersion*)

Initialize eCMD CIP Extension DLL.

Parameters:

i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:

eCMD CIP Extension is initialized and version checked

VERSIONS : eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE : if (cipInitExtension("ver3,ver4")) { die "Fatal errors initializing DLL"; }

6.1.2.2 uint32_t cipStartInstructions (ecmdChipTarget & *i_target*)

Start Instructions.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

TARGET DEPTH : Thread

TARGET STATES : Unused

6.1.2.3 uint32_t cipStartAllInstructions ()

Start Instructions on all configured processors.

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

6.1.2.4 uint32_t cipStartInstructionsSreset (ecmdChipTarget & *i_target*)

Start Instructions using an S-Reset.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

TARGET DEPTH : Thread

TARGET STATES : Unused

6.1.2.5 uint32_t cipStartAllInstructionsSreset ()

Start Instructions on all configured processors using an S-Reset.

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

6.1.2.6 uint32_t cipStopInstructions (ecmdChipTarget & *i_target*)

Stop Instructions.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

TARGET DEPTH : Thread

TARGET STATES : Unused

6.1.2.7 uint32_t cipStopAllInstructions ()

Stop All Instructions.

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

6.1.2.8 uint32_t cipStepInstructions (ecmdChipTarget & *i_target*, uint32_t *i_steps*)

Step Instructions.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_steps Number of steps to execute

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INSTRUCTIONS_IN_INVALID_STATE Current state of instructions is invalid to perform the operation

TARGET DEPTH : Thread

TARGET STATES : Unused

6.1.2.9 uint32_t cipSetBreakpoint (ecmdChipTarget & *i_target*, uint64_t *i_address*, ecmdBreakpointType_t & *i_type*)

Set a hardware breakpoint in Processor using a real address.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_address Address to set breakpoint at

i_type Type of breakpoint to set

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.10 uint32_t cipClearBreakpoint (ecmdChipTarget & *i_target*, uint64_t *i_address*, ecmdBreakpointType_t & *i_type*)

Clear a hardware breakpoint from Processor using a real address.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_address Address to clear breakpoint at

i_type Type of breakpoint to set

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.11 uint32_t cipGetVr (ecmdChipTarget & *i_target*, uint32_t *i_vrNum*, ecmdDataBuffer & *o_data*)

Reads the selected Processor Architected VMX Register (VR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vr number is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_vrNum Number of vr to read from

o_data DataBuffer object that holds data read from vr

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.12 uint32_t cipGetVrMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

Reads the selected Processor Architected VMX Register (VR) into the data buffers.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vr number is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 71) field must be filled in

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.13 uint32_t cipPutVr (ecmdChipTarget & *i_target*, uint32_t *i_vrNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected VMX Register (VR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disaled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_ vrNum Number of vr to write to

i_ data DataBuffer object that holds data to write into vr

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.14 uint32_t cipPutVrMultiple (ecmdChipTarget & *i_ target*, std::list< ecmdIndexEntry > & *i_ entries*)

Writes the data buffer into the selected Processor Architected VMX Register (VR).

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ ARGS Vr number is invalid

ECMD_ SUCCESS if successful

ECMD_ DATA_ OVERFLOW Too much data was provided for a write

ECMD_ DATA_ UNDERFLOW Too little data was provided to a write function

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information

i_ entries List of entries to write all **ecmdIndexEntry**(p. 71) fields must be filled in

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Core

TARGET STATES : Unused

6.1.2.15 uint32_t cipGetMemProc (ecmdChipTarget & *i_ target*, uint64_t *i_ address*, uint32_t *i_ bytes*, ecmdDataBuffer & *o_ memoryData*, ecmdDataBuffer & *o_ memoryTags*, ecmdDataBuffer & *o_ memoryEcc*, ecmdDataBuffer & *o_ memoryEccError*)

Reads System Mainstore through the processor chip using a real address.

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_MEMORY_ADDRESS Memory Address was not on a 8-byte boundary

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to read from

i_bytes Number of bytes to write

o_memoryData DataBuffer object that holds data read from memory

o_memoryTags 1 Tag bit for every 64 bits of memory data

o_memoryEcc 8 ECC bits for every 64 bits of memory data

o_memoryEccError 1 ECC Error bit for every 64 bits of memory data

NOTE : This function requires that the address be aligned on a 64 bit boundary

TARGET DEPTH : Pos

TARGET STATES : Unused

6.1.2.16 uint32_t cipPutMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_memoryData*, ecmdDataBuffer & *i_memoryTags*, ecmdDataBuffer & *i_memoryErrorInject*)

Writes System Mainstore through the processor chip using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_MEMORY_ADDRESS Memory Address was not on a 8-byte boundary

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to write to

i_bytes Number of bytes to write

i_memoryData DataBuffer object that holds data to write into memory

i_memoryTags 1 Tag bit for every 64 bits of memory data (If this has length of zero, the user wants the HW to generate this info; otherwise, use their their values.

i_memoryErrorInject 2 Error Inject bits for every 64 bits of memory data (If this has a length of zero, no error inject; otherwise 0b00, 0x11 no error inject, 0b01 single-bit error inject, 0b10 double-bit error inject.)

NOTE : This function requires that the address be aligned on a 64 bit boundary

TARGET DEPTH : Pos

TARGET STATES : Unused

6.1.2.17 uint32_t cipGetMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_memoryData*, ecmdDataBuffer & *o_memoryTags*, ecmdDataBuffer & *o_memoryEcc*, ecmdDataBuffer & *o_memoryEccError*, ecmdDataBuffer & *o_memorySpareBits*)

Reads System Mainstore through the memory controller using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_MEMORY_ADDRESS Memory Address was not on a 8-byte boundary

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to read from

i_bytes Number of bytes to write

o_memoryData DataBuffer object that holds data read from memory

o_memoryTags 2 Tag bits for every 32 bytes of memory data

o_memoryEcc 24 ECC bits for every 32 bytes of memory data

o_memoryEccError 1 ECC Error bit for every 32 bytes of memory data

o_memorySpareBits 2 Spare bits for every 32 bytes of memory data

NOTE : This function requires that the address be aligned on a 32-byte boundary

TARGET DEPTH : Cage

TARGET STATES : Unused

6.1.2.18 uint32_t cipPutMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_memoryData*, ecmdDataBuffer & *i_memoryTags*, ecmdDataBuffer & *i_memoryEcc*, ecmdDataBuffer & *i_memorySpareBits*, ecmdDataBuffer & *i_memoryErrorInject*)

Writes System Mainstore through the memory controller using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_MEMORY_ADDRESS Memory Address was not on a 8-byte boundary

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to write to

i_bytes Number of bytes to write

i_memoryData DataBuffer object that holds data to write into memory

i_memoryTags [Optional] 2 Tag bits for every 32 bytes of memory data

i_memoryEcc 24 ECC bits for every 32 bytes of memory data

i_memorySpareBits [Optional] 2 Spare bits for every 32 bytes of memory data

i_memoryErrorInject 2 Error Inject bits for every 64 bits of memory data (If this has a length of zero, no error inject; otherwise 0b00, 0x11 no error inject, 0b01 single-bit error inject, 0b10 double-bit error inject.)

NOTE : This function requires that the address be aligned on a 32-byte boundary

NOTE : If ecmdDataBuffers for i_memoryTags and i_memorySpareBIts have a length of zero, the user wants the HW to generate this info. If these ecmdDataBuffers have a length, the the user wants their values to be used.

TARGET DEPTH : Cage

TARGET STATES : Unused

6.2 cmdClientPerlapi.H File Reference

Command Line eCMD Perlapi Extension.

```
#include <ecmdReturnCodes.H>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <cmdStructs.H>
```

Load/Unload Functions

- int **cmdInitExtension** (const char *i_clientVersion)

  Initialize eCMD CMD Extension.

Functions

- uint32_t **cmdRunCommand** (std::string i_command)

  Run the command line command passed in , output goes to stdout and is not returned.

- uint32_t **cmdRunCommandCaptureOutput** (std::string i_command, std::string &o_output)

  Run the command line command passed in , stdout is captured and returned.

6.2.1 Detailed Description

Command Line eCMD Perlapi Extension.

Extension Owner : Chris Engel

6.2.2 Function Documentation

6.2.2.1 int cmdInitExtension (const char * *i_clientVersion*)

Initialize eCMD CMD Extension.

Parameters:

i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:

eCMD CIP Extension is initialized and version checked

VERSIONS : eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE : if (cmdInitExtension("ver3,ver4")) { die "Fatal errors initializing DLL"; }

6.2.2.2 uint32_t cmdRunCommand (std::string *i_command*)

Run the command line command passed in , output goes to stdout and is not returned.

Parameters:

i_command Command to pass to interpreter, ie 'ecmdquery version'

Return values:

ECMD_SUCCESS if successful load

nonzero if unsuccessful

6.2.2.3 uint32_t cmdRunCommandCaptureOutput (std::string *i_command*, std::string & *o_output*)

Run the command line command passed in , stdout is captured and returned.

Parameters:

i_command Command to pass to interpreter, ie 'ecmdquery version'

o_output Standard out from running command

Return values:

ECMD_SUCCESS if successful load

nonzero if unsuccessful

6.3 croClientPerlapi.H File Reference

Cronus eCMD Perlapi Extension.

```
#include <ecmdReturnCodes.H>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <croStructs.H>
```

Load/Unload Functions

- int **croInitExtension** (const char *i_clientVersion)

  Initialize eCMD CRO Extension DLL.

Misc Functions

- uint32_t **croReset** ()

  Reset Cronus internal state variables.

- uint32_t **croDisplayVersion** ()

  Display the Cronus version information to stdout.

- uint32_t **croSetDebug** (char i_major, char i_minor= '1')

  Set a Cronus debug flag -debug.

- uint32_t **croClearDebug** (char i_major, char i_minor= '1')

  Clear a Cronus debug flag -debug.

- bool **croIsDebugOn** (char i_major, char i_minor= '1')

  Checks whether Cronus debug flag is set.

JTAG Scan Functions

- uint32_t **croJtagScanRead** (**ecmdChipTarget** &i_target, int i_bitlength, **ecmdDataBuffer** &o_data, croJtagScanMode i_mode=CRO_JTAG_SHIFTDR)

  Perform a scan operation on the chip and sample TDO.

- uint32_t **croJtagScanWrite** (**ecmdChipTarget** &i_target, int i_bitlength, **ecmdDataBuffer** &i_data, croJtagScanMode i_mode=CRO_JTAG_SHIFTDR)

  Perform a scan operation on the chip and drive TDI.

- uint32_t **croJtagScanReadWrite** (**ecmdChipTarget** &i_target, int i_bitlength, **ecmdDataBuffer** &i_data, **ecmdDataBuffer** &o_data, croJtagScanMode i_mode=CRO_JTAG_SHIFTDR)

  Perform a scan operation on the chip and drive TDI and sample TDO on same cycle.

L2 Functions

- uint32_t **croFastLoadL2** (**ecmdChipTarget** &i_target, std::list< **ecmdMemoryEntry** > &i_data, const char *o_ringFile=NULL)

  Load the provided data into L2 using the fast load mechanism.

- uint32_t **croFastLoadL2RingImage** (**ecmdChipTarget** &i_target, const char *i_ringFile)

  Load the provided data into L2 using the fast load mechanism with a prebuilt ring image file.

- uint32_t **croRamInstruction** (**ecmdChipTarget** &i_target, const char *i_instructionDecode)

  Ram a particular instruction decode in Eclipz P6.

- uint32_t **croGetL2** (**ecmdChipTarget** &i_target, uint64_t i_address, croL2Fields &i_l2fields, **ecmdDataBuffer** &o_data)

  Displays cache lines from Level 2 processor cache.

- uint32_t **croGetL2Data** (**ecmdChipTarget** &i_target, croL2Fields &i_l2fields, **ecmdDataBuffer** &o_data)

  Displays cache lines from Level 2 processor cache data.

- uint32_t **croGetL2Dir** (**ecmdChipTarget** &i_target, croL2Fields &i_l2fields, **ecmdDataBuffer** &o_data)

  Displays Level 2 processor cache directory entry.

- uint32_t **croGetL2Tag** (**ecmdChipTarget** &i_target, uint64_t i_address, croL2Fields &i_l2fields, **ecmdDataBuffer** &o_data)

  Displays Level 2 processor cache tag bit per quadword basis.

- uint32_t **croPutL2** (**ecmdChipTarget** &i_target, uint64_t i_address, croL2Fields &i_l2fields, **ecmdDataBuffer** &i_data)

  Replace Level 2 processor cache line.

- uint32_t **croPutL2Data** (**ecmdChipTarget** &i_target, croL2Fields &i_l2fields, **ecmdDataBuffer** &i_data)

  Replace a full Level 2 processor data cache line.

- uint32_t **croPutL2Dir** (**ecmdChipTarget** &i_target, croL2Fields &i_l2fields, **ecmdDataBuffer** &i_data)

  Replace a full Level 2 processor directory cache entry.

- uint32_t **croPutL2Tag** (**ecmdChipTarget** &i_target, uint64_t i_address, croL2Fields &i_l2fields)

  Put the L2 tag bit for the quadword beginning at the given address.

Functions

- uint32_t **croGetConfiguration** (**ecmdChipTarget** &i_target, std::string i_name, **ecmdConfigValid_t** &o_validOutput, std::string &o_valueAlpha, uint32_t &o_valueNumeric)

Retrieve the value of a Configuration Setting - Cronus version.

- uint32_t **croSetConfiguration** (**ecmdChipTarget** &i_target, std::string i_name, **ecmdConfigValid_t** i_validInput, std::string i_valueAlpha, uint32_t i_valueNumeric)

  Set the value of a Configuration Setting - Cronus version.

6.3.1 Detailed Description

Cronus eCMD Perlapi Extension.

Extension Owner : Chris Engel

6.3.2 Function Documentation

6.3.2.1 int croInitExtension (const char * *i_clientVersion*)

Initialize eCMD CRO Extension DLL.

Parameters:

i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:

eCMD CRO Extension is initialized and version checked

VERSIONS : eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE : if (croInitExtension("ver3,ver4")) { die "Fatal errors initializing DLL"; }

6.3.2.2 uint32_t croReset ()

Reset Cronus internal state variables.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:

All internal Cronus data is cleared, config file is reread and Cronus is reinitialized

6.3.2.3 uint32_t croDisplayVersion ()

Display the Cronus version information to stdout.

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

NOTE : This is equivalent to 'croquery version'

6.3.2.4 uint32_t croSetDebug (char *i_major*, char *i_minor* = '1')

Set a Cronus debug flag -debug.

Parameters:
i_major Major debug char
i_minor Minor debug char

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

NOTE : To set all minor's of a particular major, set i_minor = '0'

NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

6.3.2.5 uint32_t croClearDebug (char *i_major*, char *i_minor* = '1')

Clear a Cronus debug flag -debug.

Parameters:
i_major Major debug char
i_minor Minor debug char

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

NOTE : To clear all minor's of a particular major, set i_minor = '0'

NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

6.3.2.6 bool croIsDebugOn (char *i_major*, char *i_minor* = '1')

Checks whether Cronus debug flag is set.

Parameters:
i_major Major debug char
i_minor Minor debug char

Return values:

true if debug is on

false if debug is off

NOTE : To check if any minor of a particular major is on, set i_minor = '0'

NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

6.3.2.7 uint32_t croJtagScanRead (ecmdChipTarget & *i_target*, int *i_bitlength*, ecmdDataBuffer & *o_data*, croJtagScanMode *i_mode* = CRO_JTAG_SHIFTDR)

Perform a scan operation on the chip and sample TDO.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

o_data Output from TDO

i_bitlength Bit Length to shift scan chain

i_mode Scan mode settings

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE : This function assumes the user has sent in the appropriate scan instruction

NOTE : The chain is shifted for the exact length of i_bitlength

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.8 uint32_t croJtagScanWrite (ecmdChipTarget & *i_target*, int *i_bitlength*, ecmdDataBuffer & *i_data*, croJtagScanMode *i_mode* = CRO_JTAG_SHIFTDR)

Perform a scan operation on the chip and drive TDI.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_data Input to drive on TDI

i_bitlength Bit Length to shift scan chain

i_mode Scan mode settings

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE : This function assumes the user has sent in the appropriate scan instruction

NOTE : The chain is shifted for the exact length of i_bitlength

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.9 uint32_t croJtagScanReadWrite (ecmdChipTarget & *i_target*, int *i_bitlength*, ecmdDataBuffer & *i_data*, ecmdDataBuffer & *o_data*, croJtagScanMode *i_mode* = CRO_JTAG_SHIFTDR)

Perform a scan operation on the chip and drive TDI and sample TDO on same cycle.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_data Input to drive on TDI

o_data Output from TDO

i_bitlength Bit Length to shift scan chain

i_mode Scan mode settings

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE : This function assumes the user has sent in the appropriate scan instruction

NOTE : The chain is shifted for the exact length of i_bitlength

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.10 uint32_t croGetConfiguration (ecmdChipTarget & *i_target*, std::string *i_name*, ecmdConfigValid_t & *o_validOutput*, std::string & *o_valueAlpha*, uint32_t & *o_valueNumeric*)

Retrieve the value of a Configuration Setting - Cronus version.

Parameters:

i_target struct that contains chip and cage/node/slot/position/core information if necessary

i_name Name of setting as defined by eCMD Api

o_validOutput Indicator if o_valueAlpha, o_valueNumeric (or both) are valid.

o_valueAlpha Alpha value of setting (if appropriate)

o_valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_INVALID_CONFIG_NAME Name specified is not valid

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

TARGET DEPTH : Thread

TARGET STATES : Unused

6.3.2.11 uint32_t croSetConfiguration (ecmdChipTarget & *i_target*, std::string *i_name*, ecmdConfigValid_t *i_validInput*, std::string *i_valueAlpha*, uint32_t *i_valueNumeric*)

Set the value of a Configuration Setting - Cronus version.

Parameters:

i_target struct that contains chip and cage/node/slot/position/core information if necessary

i_name Name of setting as defined by eCMD Api

i_validInput Indicator if i_valueAlpha, i_valueNumeric (or both) are valid.

i_valueAlpha Alpha value of setting (if appropriate)

i_valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT Value is not in correct format for specified configuration setting

ECMD_INVALID_CONFIG_NAME Name specified is not valid

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

TARGET DEPTH : Thread

TARGET STATES : Unused

6.3.2.12 uint32_t croFastLoadL2 (ecmdChipTarget & *i_target*, std::list< ecmdMemoryEntry > & *i_data*, const char * *o_ringFile* = NULL)

Load the provided data into L2 using the fast load mechanism.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_data List of DataBuffer objects that holds data to write into l2

o_ringFile Optional : Output filename to store ring images for later use with croFastLoadL2RingImage

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.13 uint32_t croFastLoadL2RingImage (ecmdChipTarget & *i_target*, const char * *i_ringFile*)

Load the provided data into L2 using the fast load mechanism with a prebuilt ring image file.

Parameters:

i_ target struct that contains chip and cage/node/slot/position information if necessary

i_ ringFile Filename to read ring images from, previously built from croFastLoadL2

Return values:

ECMD_ SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.14 uint32_t croRamInstruction (ecmdChipTarget & *i_ target*, const char * *i_ instructionDecode*)

Ram a particular instruction decode in Eclipz P6.

Parameters:

i_ target struct that contains chip and cage/node/slot/position information if necessary

i_ instructionDecode Decode of instruction to run ie 'addi G7,G3,0x0000'

Return values:

ECMD_ SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Core

TARGET STATES : Unused

6.3.2.15 uint32_t croGetL2 (ecmdChipTarget & *i_ target*, uint64_t *i_ address*, croL2Fields & *i_ l2fields*, ecmdDataBuffer & *o_ data*)

Displays cache lines from Level 2 processor cache.

Parameters:

i_ target struct that contains chip and cage/node/slot/position information if necessary

i_ address Real address(0:63) from which to get data

i_ l2fields Struct that holds L2 cache fields

o_ data DataBuffer object that holds cache line data

Return values:

ECMD_ SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.16 uint32_t croGetL2Data (ecmdChipTarget & *i_target*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *o_data*)

Displays cache lines from Level 2 processor cache data.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_l2fields Struct that holds L2 cache fields

o_data DataBuffer object that holds cache line data

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.17 uint32_t croGetL2Dir (ecmdChipTarget & *i_target*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *o_data*)

Displays Level 2 processor cache directory entry.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_l2fields Struct that holds L2 cache fields

o_data DataBuffer object that holds cache directory data

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.18 uint32_t croGetL2Tag (ecmdChipTarget & *i_target*, uint64_t *i_address*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *o_data*)

Displays Level 2 processor cache tag bit per quadword basis.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_address Starting real address(0:63) from which to get tag data

i_l2fields Struct that holds L2 cache fields

o_data DataBuffer object that holds tag bits

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.19 uint32_t croPutL2 (ecmdChipTarget & *i_target*, uint64_t *i_address*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *i_data*)

Replace Level 2 processor cache line.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_address Real address(0:63) to which to get write data

i_l2fields Struct that holds L2 cache fields

i_data DataBuffer object that holds cache data to be written

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.20 uint32_t croPutL2Data (ecmdChipTarget & *i_target*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *i_data*)

Replace a full Level 2 processor data cache line.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_l2fields Struct that holds L2 cache fields

i_data DataBuffer object that holds cache data to be written

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.21 uint32_t croPutL2Dir (ecmdChipTarget & *i_target*, croL2Fields & *i_l2fields*, ecmdDataBuffer & *i_data*)

Replace a full Level 2 processor directory cache entry.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_l2fields Struct that holds L2 cache fields

i_data DataBuffer object that holds cache data to be written

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.3.2.22 uint32_t croPutL2Tag (ecmdChipTarget & *i_target*, uint64_t *i_address*, croL2Fields & *i_l2fields*)

Put the L2 tag bit for the quadword beginning at the given address.

Parameters:

i_target struct that contains chip and cage/node/slot/position information if necessary

i_address Real address(0:63) (in hex) to put L2 tag data

i_l2fields Struct that holds L2 cache fields

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4 ecmdClientPerlapi.H File Reference

eCMD Perl API Usage :

```
#include <string>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <ecmdReturnCodes.H>
```

Load/Unload Functions

- int **ecmdLoadDll** (const char *i_dllName, const char *i_clientVersion)

  Initialize the eCMD DLL.

- void **ecmdUnloadDll** ()

  Clean up the Perl Module - MUST BE CALLED JUST BEFORE SCRIPT EXIT.

- int **ecmdCommandArgs** (char **io_argv)

  Pass any unknown command line paramaters to the DLL for processing (ex. -p#, Cronus -debug).

Error Handling Functions

- void **ecmdEnablePerlSafeMode** ()

  Enable Perl Error Handling Safe Mode - Croak on any eCMD error (default is enabled).

- void **ecmdDisablePerlSafeMode** ()

  Disable Perl Error Handling Safe Mode - Return eCMD errors to client (default is enabled).

Command Line Parsing Functions

- bool **ecmdParseOption** (char **io_argv, const char *i_option)

  Iterates over argv, looking for given option string, removes it if found.

- char * **ecmdParseOptionWithArgs** (char **io_argv, const char *i_option)

  Iterates over argv, looking for given option string, removes it if found.

Query Functions

- uint32_t **ecmdQueryDllInfo** (**ecmdDllInfo** &o_dllInfo)

  Query information about the Dll that is loaded.

- bool **ecmdQueryVersionGreater** (const char *version)

  Query to see if plugin is greater or equal to release specified.

- uint32_t **ecmdQueryConfig** (**ecmdChipTarget** &i_target, **ecmdQueryData** &o_queryData, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query configuration information from the DLL.

- uint32_t **ecmdQuerySelected** (**ecmdChipTarget** &io_target, **ecmdQueryData** &o_queryData, **ecmdConfigLoopType_t** i_looptype=ECMD_SELECTED_TARGETS_LOOP)

  Query User Selected Targeting information from the DLL, i.e (-p#,-c#,-t#).

- uint32_t **ecmdQueryRing** (**ecmdChipTarget** &i_target, std::list< **ecmdRingData** > &o_queryData, const char *i_ringName=NULL, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Ring information from the DLL.

- uint32_t **ecmdQueryLatch** (**ecmdChipTarget** &i_target, std::list< **ecmdLatchData** > &o_queryData, **ecmdLatchMode_t** i_mode, const char *i_latchName, const char *i_ringName=NULL, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Latch information from the DLL.

- uint32_t **ecmdQueryArray** (**ecmdChipTarget** &i_target, std::list< **ecmdArrayData** > &o_queryData, const char *i_arrayName=NULL, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Array information from the DLL.

- uint32_t **ecmdQuerySpy** (**ecmdChipTarget** &i_target, std::list< **ecmdSpyData** > &o_queryData, const char *i_spyName=NULL, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Spy information from the DLL.

- uint32_t **ecmdQueryScom** (**ecmdChipTarget** &i_target, std::list< **ecmdScomData** > &o_queryData, uint32_t i_address=0xFFFFFFFF, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Scom information from the DLL.

- uint32_t **ecmdQueryTraceArray** (**ecmdChipTarget** &i_target, std::list< **ecmdTraceArrayData** > &o_queryData, const char *i_traceArrayName=NULL, **ecmdQueryDetail_t** i_detail=ECMD_QUERY_DETAIL_HIGH)

  Query Trace Array information from the DLL.

- uint32_t **ecmdQueryFileLocation** (**ecmdChipTarget** &i_target, **ecmdFileType_t** i_fileType, std::string &o_fileLocation)

  Query the location of a specific file type for the selected target.

- bool **ecmdQueryTargetConfigured** (**ecmdChipTarget** i_target, **ecmdQueryData** *i_queryData=NULL)

  Query if a particular target is configured in the system.

Scan Functions

- uint32_t **getRing** (**ecmdChipTarget** &i_target, const char *i_ringName, **ecmdDataBuffer** &o_data)

Scans the ring from the selected chip into the data buffer.

- uint32_t **putRing** (**ecmdChipTarget** &i_target, const char *i_ringName, **ecmdDataBuffer** &i_data)

  Scans ring from the data buffer into the selected chip in the selected ring.

- uint32_t **getLatch** (**ecmdChipTarget** &i_target, const char *i_ringName, const char *i_latchName, std::list< **ecmdLatchEntry** > &o_data, **ecmdLatchMode_t** i_mode)

  Reads the selected spy into the data buffer.

- uint32_t **putLatch** (**ecmdChipTarget** &i_target, const char *i_ringName, const char *i_latchName, **ecmdDataBuffer** &i_data, uint32_t i_startBit, uint32_t i_numBits, uint32_t &o_matchs, **ecmdLatchMode_t** i_mode)

  Writes the data buffer into the all latches matching i_latchName.

- uint32_t **getRingWithModifier** (**ecmdChipTarget** &i_target, uint32_t i_address, uint32_t i_bitLength, **ecmdDataBuffer** &o_data)

  Scans the specified number of bits from the selected chip and ring address into the data buffer.

- uint32_t **putRingWithModifier** (**ecmdChipTarget** &i_target, uint32_t i_address, uint32_t i_bitLength, **ecmdDataBuffer** &i_data)

  Scans the specified number of bits from the data buffer into the selected chip in the selected ring address.

Scom Functions

- uint32_t **getScom** (**ecmdChipTarget** &i_target, uint32_t i_address, **ecmdDataBuffer** &o_data)

  Scoms bits from the selected address into the data buffer.

- uint32_t **putScom** (**ecmdChipTarget** &i_target, uint32_t i_address, **ecmdDataBuffer** &i_data)

  Scoms bits from the data buffer into the selected address.

Jtag Functions

- uint32_t **sendCmd** (**ecmdChipTarget** &i_target, uint32_t i_instruction, uint32_t i_modifier, **ecmdDataBuffer** &o_status)

  Send a JTAG instruction and modifier to the specified chip.

FSI Functions

- uint32_t **getCfamRegister** (**ecmdChipTarget** &i_target, uint32_t i_address, **ecmdDataBuffer** &o_data)

  Read data from the selected CFAM register address into the data buffer.

- uint32_t **putCfamRegister** (**ecmdChipTarget** &i_target, uint32_t i_address, **ecmdDataBuffer** &i_data)

  Write data into the selected CFAM register address.

Spy Functions

- uint32_t **getSpy** (**ecmdChipTarget** &i_target, const char *i_spyName, **ecmdDataBuffer** &o_data)

  Reads the selected spy into the data buffer.

- uint32_t **getSpyEnum** (**ecmdChipTarget** &i_target, const char *i_spyName, std::string &o_enumValue)

  Reads the selected spy and returns it's assocaiated enum.

- uint32_t **getSpyEpCheckers** (**ecmdChipTarget** &i_target, const char *i_spyEpCheckersName, **ecmdDataBuffer** &o_inLatchData, **ecmdDataBuffer** &o_outLatchData, **ecmdDataBuffer** &o_eccErrorMask)

  Read an ECC grouping and return the in and out bits as well as a error mask if any out bits are invalid.

- uint32_t **getSpyGroups** (**ecmdChipTarget** &i_target, const char *i_spyName, std::list< **ecmdSpyGroupData** > &o_groups)

  Reads the selected spy and load all the spy groups into provided list.

- uint32_t **putSpy** (**ecmdChipTarget** &i_target, const char *i_spyName, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected spy.

- uint32_t **putSpyEnum** (**ecmdChipTarget** &i_target, const char *i_spyName, const std::string i_enumValue)

  Writes the enum into the selected spy.

Ring Cache Functions

- void **ecmdEnableRingCache** ()

  Enables internal caching of read/writes of scan rings to the chip for functions like getring/getspy/getspr.

- uint32_t **ecmdDisableRingCache** ()

  Disable internal caching of reads/writes of scan rings.

- uint32_t **ecmdFlushRingCache** ()

  Flush all modified data from the internal cache to the hardware, then remove all rings from cache.

- bool **ecmdIsRingCacheEnabled** ()

  Returns true/false to signify if caching is currently enabled.

Array Functions

- uint32_t **getArray** (**ecmdChipTarget** &i_target, const char *i_arrayName, **ecmdDataBuffer** &i_address, **ecmdDataBuffer** &o_data)

  Reads bits from the selected array into the data buffer.

- uint32_t **getArrayMultiple** (**ecmdChipTarget** &i_target, const char *i_arrayName, std::list< **ecmdArrayEntry** > &io_entries)

  Reads bits from multiple array addresses/elements into the list of data buffers.

- uint32_t **putArray** (**ecmdChipTarget** &i_target, const char *i_arrayName, **ecmdDataBuffer** &i_address, **ecmdDataBuffer** &i_data)

  Writes bits from the data buffer into the selected array.

- uint32_t **putArrayMultiple** (**ecmdChipTarget** &i_target, const char *i_arrayName, std::list< **ecmdArrayEntry** > &i_entries)

  Writes bits from the list of entries into the selected array.

Clock Functions

- uint32_t **ecmdQueryClockState** (**ecmdChipTarget** &i_target, const char *i_clockDomain, **ecmdClockState_t** &o_clockState)

  Query the state of the clocks for a domain.

- uint32_t **startClocks** (**ecmdChipTarget** &i_target, const char *i_clockDomain, bool i_forceState=false)

  Start the clocks in the domain specified.

- uint32_t **stopClocks** (**ecmdChipTarget** &i_target, const char *i_clockDomain, bool i_forceState=false)

  Stop the clocks in the domain specified.

- uint32_t **ecmdSetClockSpeed** (**ecmdChipTarget** &i_target, **ecmdClockType_t** i_type, uint32_t i_speed, **ecmdClockSpeedType_t** i_speedType, **ecmdClockSetMode_t** i_mode, **ecmdClockRange_t** i_range)

  Change a system clock speed without adjusting system initialization settings using speed value.

- uint32_t **ecmdSetClockMultDiv** (**ecmdChipTarget** &i_target, **ecmdClockType_t** i_type, uint32_t i_multiplier, uint32_t i_divider)

  Change a system clock speed without adjusting system initialization settings using speed value.

iSteps Functions

- uint32_t **iStepsByNumber** (**ecmdDataBuffer** &i_steps)

  Run iSteps by number.

- uint32_t **iStepsByName** (std::string i_stepName)

  Run a single iStep by name.

- uint32_t **iStepsByNameMultiple** (std::list< std::string > i_stepNames)

  Run multiple iSteps by name.

- uint32_t **iStepsByNameRange** (std::string i_stepNameBegin, std::string i_stepNameEnd)

  Run all iSteps by name starting with i_stepNameBegin and ending with i_stepNameEnd.

Processor Functions

- uint32_t **ecmdQueryProcRegisterInfo** (**ecmdChipTarget** &i_target, const char *i_name, **ecmdProcRegisterInfo** &o_data)

  Query Information about a Processor Register (SPR/GPR/FPR).

- uint32_t **getSpr** (**ecmdChipTarget** &i_target, const char *i_sprName, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

- uint32_t **getSprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdNameEntry** > &io_entries)

  Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

- uint32_t **putSpr** (**ecmdChipTarget** &i_target, const char *i_sprName, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

- uint32_t **putSprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdNameEntry** > &i_entries)

  Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

- uint32_t **getGpr** (**ecmdChipTarget** &i_target, uint32_t i_gprNum, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

- uint32_t **getGprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &io_entries)

  Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

- uint32_t **putGpr** (**ecmdChipTarget** &i_target, uint32_t i_gprNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

- uint32_t **putGprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &i_entries)

  Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

- uint32_t **getFpr** (**ecmdChipTarget** &i_target, uint32_t i_fprNum, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

- uint32_t **getFprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &io_entries)

  Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

- uint32_t **putFpr** (**ecmdChipTarget** &i_target, uint32_t i_fprNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

- uint32_t **putFprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &i_entries)

  Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

- uint32_t **getSlb** (**ecmdChipTarget** &i_target, uint32_t i_slbNum, **ecmdDataBuffer** &o_data)

  Reads the selected Processor SLB Entry into the data buffer.

- uint32_t **getSlbMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &io_entries)

  Reads the selected Processor SLB Entry into the data buffer.

- uint32_t **putSlb** (**ecmdChipTarget** &i_target, uint32_t i_slbNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor SLB Entry.

- uint32_t **putSlbMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdIndexEntry** > &i_entries)

  Writes the data buffer into the selected Processor SLB Entry.

Trace Array Functions

- uint32_t **getTraceArray** (**ecmdChipTarget** &i_target, const char *i_name, bool i_doTraceStopStart, std::vector< **ecmdDataBuffer** > &o_data)

  Dump all entries of specified trace array.

- uint32_t **getTraceArrayMultiple** (**ecmdChipTarget** &i_target, bool i_doTraceStopStart, std::list< **ecmdNameVectorEntry** > &o_data)

  Dump all entries of specified trace array.

Memory Functions

- uint32_t **getMemProc** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Reads System Mainstore through the processor chip using a real address.

- uint32_t **putMemProc** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_data)

  Writes System Mainstore through the processor chip using a real address.

- uint32_t **getMemDma** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Reads System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

- uint32_t **putMemDma** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_data)

  Writes System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

- uint32_t **getMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Reads System Mainstore through the memory controller using a real address.

- uint32_t **putMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_data)

  Writes System Mainstore through the memory controller using a real address.

- uint32_t **ecmdCacheFlush** (**ecmdChipTarget** &i_target, **ecmdCacheType_t** i_cacheType)

  Cache Flush.

Simulation Functions

- uint32_t **simaet** (const char *i_function)

  Enable/Disable Simulation AET Logging.

- uint32_t **simcheckpoint** (const char *i_checkpoint)

  Store a checkpoint to specified file.

- uint32_t **simclock** (uint32_t i_cycles)

  Clock the model.

- uint32_t **simecho** (const char *i_message)

  Echo message to stdout and sim log.

- uint32_t **simexit** (uint32_t i_rc=0, const char *i_message=NULL)

  Close down the simulation model.

- uint32_t **simEXPECTFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_expect, uint32_t i_row=0, uint32_t i_offset=0)

  Perform expect on facility using name.

- uint32_t **simexpecttcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_expect, uint32_t i_row=0)

  Perform expect on TCFAC facility.

- uint32_t **simgetcurrentcycle** (uint64_t &o_cyclecount)

Fetch current model cycle count.

- uint32_t **simGETFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &o_data, uint32_t i_row=0, uint32_t i_offset=0)

  Retrieve a Facility using a name.

- uint32_t **simGETFACX** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &o_data, uint32_t i_row=0, uint32_t i_offset=0)

  Retrieve a Facility using a name - preserving Xstate.

- uint32_t **simgettcfac** (const char *i_tcfacname, **ecmdDataBuffer** &o_data, uint32_t i_row=0, uint32_t i_startbit=0, uint32_t i_bitlength=0)

  Retrieve a TCFAC facility.

- uint32_t **siminit** (const char *i_checkpoint)

  Initialize the simulation.

- uint32_t **simPOLLFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_expect, uint32_t i_row=0, uint32_t i_offset=0, uint32_t i_maxcycles=1, uint32_t i_pollinterval=1)

  Poll a facility waiting for expected value.

- uint32_t **simpolltcfac** (const char *i_tcfacname, **ecmdDataBuffer** &i_expect, uint32_t i_row=0, uint32_t i_startbit=0, uint32_t i_bitlength=0, uint32_t i_maxcycles=1, uint32_t i_pollinterval=1)

  Poll a TCFAC facility waiting for expected value.

- uint32_t **simPUTFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_offset=0)

  Write a Facility using a name.

- uint32_t **simPUTFACX** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_offset=0)

  Write a Facility using a name - preserving Xstate.

- uint32_t **simputtcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Write a TCFAC facility.

- uint32_t **simrestart** (const char *i_checkpoint)

  Load a checkpoint into model.

- uint32_t **simSTKFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_offset=0)

  Stick a Facility using a name.

- uint32_t **simstktcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Stick a TCFAC facility.

- uint32_t **simSUBCMD** (const char *i_command)

Run RTX SUBCMD.

- uint32_t **simtckinterval** (uint32_t i_tckinterval)

  Set TCK Interval setting in the model for JTAG Master.

- uint32_t **simUNSTICK** (const char *i_facname, uint32_t i_bitlength, uint32_t i_row=0, uint32_t i_offset=0)

  Unstick a Facility using a name.

- uint32_t **simunsticktcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Unstick a TCFAC facility.

- uint32_t **simGetHierarchy** (**ecmdChipTarget** &i_target, std::string &o_hierarchy)

  Fetch the hierarchy for the specified chip target relative to the latch names in the scandef.

- uint32_t **ecmdQueryChipSimModelVersion** (**ecmdChipTarget** &i_target, std::string &o_timestamp)

  Will retrieve the model timestamp from the simulation, in hardware mode "NA" is returned.

- uint32_t **ecmdQueryChipScandefVersion** (**ecmdChipTarget** &i_target, std::string &o_timestamp)

  Will retrieve the scandef timestamp from the scandef being used for the specified target.

- std::string **simCallFusionCommand** (const char *i_fusionObject, const char *i_replicaID, const char *i_command)

  Run a command on another Fusion module.

- uint32_t **simFusionRand32** (uint32_t i_min=0, uint32_t i_max=~0UL, const char *i_fusionRandObject=NULL)

  Returns a random 32 bit number in the range [min,max] using the Fusion MasterSeed; by default each client will have an own instance of Fusion's Random32BitNumber object, but the user can specify the object's name and reuse the same object across multiple clients; if no range is specified 0 and MAXINT32 will be used.

- uint32_t **simOutputFusionMessage** (const char *i_header, const char *i_message, **ecmdFusionSeverity_t** i_severity, **ecmdFusionMessageType_t** i_type, const char *i_file=NULL, uint32_t i_line=0)

  Echo Messages to Fusion logs.

- void **simSetFusionMessageFormat** (const char *i_format)

  Set Fusion Message Format.

- uint32_t **simPutDial** (const char *i_dialName, const std::string i_enumValue)

  Write a simulation dial with specified value.

- uint32_t **simGetDial** (const char *i_dialName, std::string &o_enumValue)

  Read a simulation dial.

- uint32_t **simGetOutFile** (const char *i_filename, std::string &o_absFilename)

Obtain absolute filename of a file that will be placed in the SIMOUT directory of the server / Fusion process and add new file to the bom information in the SUM file.

- uint32_t **simGetInFile** (const char *i_filename, std::string &o_absFilename)

  Resolve absolute filename of a file by searching the SIMIN paths and add new file to the bom information in the SUM file.

- uint32_t **simGetEnvironment** (const char *i_envName, std::string &o_envValue)

  Retrieve value of an environment variable on the server side.

- uint32_t **simGetModelInfo** (**ecmdSimModelInfo** &o_modelInfo)

  Query information about the model from the server.

Error Handling Functions

- std::string **ecmdGetErrorMsg** (uint32_t i_errorCode, bool i_parseReturnCode=true)

  Retrieve additional error information for errorcode.

- uint32_t **ecmdRegisterErrorMsg** (uint32_t i_errorCode, const char *i_whom, const char *i_message)

  Register an error message that has occured.

- void **ecmdFlushRegisteredErrorMsgs** ()

  Flush all registered messages, they are no long retrievable.

Output Functions

- void **ecmdOutputError** (const char *i_message)

  Output a message related to an error.

- void **ecmdOutputWarning** (const char *i_message)

  Output a message related to an warning.

- void **ecmdOutput** (const char *i_message)

  Output a message to the screen or logs.

Misc Functions

- uint32_t **ecmdGetGlobalVar** (**ecmdGlobalVarType_t** i_type)

  Retrieve the value of some ecmdGlobalVars.

- void **ecmdSetTraceMode** (**ecmdTraceType_t** i_type, bool i_enable)

  Enable/Disable a trace mode.

- bool **ecmdQueryTraceMode** (**ecmdTraceType_t** i_type)

  Query the state of a trace mode.

- uint32_t **ecmdDelay** (uint32_t i_simCycles, uint32_t i_msDelay)

  Function to delay a procedure either by running sim cycles or by doing a millisecond delay.

- uint32_t **makeSPSystemCall** (**ecmdChipTarget** &i_target, const std::string &i_command, std::string &o_stdout)

  Make a system call on the targetted Service Processor or Service Element.

Configuration Functions

- uint32_t **ecmdGetConfiguration** (**ecmdChipTarget** &i_target, std::string i_name, **ecmdConfigValid_t** &o_validOutput, std::string &o_valueAlpha, uint32_t &o_valueNumeric)

  Retrieve the value of a Configuration Setting.

- uint32_t **ecmdSetConfiguration** (**ecmdChipTarget** &i_target, std::string i_name, **ecmdConfigValid_t** i_validInput, std::string i_valueAlpha, uint32_t i_valueNumeric)

  Set the value of a Configuration Setting.

- uint32_t **ecmdDeconfigureTarget** (**ecmdChipTarget** &i_target)

  Deconfigure a target in the system.

- uint32_t **ecmdConfigureTarget** (**ecmdChipTarget** &i_target)

  Configure a target in the system - must be previously known to the system.

- uint32_t **ecmdTargetToUnitId** (**ecmdChipTarget** &io_target)

  Converts an eCmd (physical) Target to a HOM Unit Id.

- uint32_t **ecmdUnitIdStringToTarget** (std::string i_unitId, std::list< **ecmdChipTarget** > &o_targetList)

  Converts a Unit Id String to an eCmd (physical) Target.

- uint32_t **ecmdUnitIdToTarget** (uint32_t i_unitId, std::list< **ecmdChipTarget** > &o_targetList)

  Converts a Unit Id to an eCmd (physical) Target.

- uint32_t **ecmdUnitIdToString** (uint32_t i_unitId, std::string &o_unitIdStr)

  Converts a Unit Id into its String representation.

- uint32_t **ecmdSequenceIdToTarget** (uint32_t i_core_seq_num, **ecmdChipTarget** &io_target, uint32_t i_thread_seq_num=0)

  Sequence ID of Cores and Threads converted to **ecmdChipTarget**(p. 20) *struct.*

Module VPD Functions

- uint32_t **getModuleVpdKeyword** (**ecmdChipTarget** &i_target, const char *i_record_name, const char *i_keyword, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Read Module VPD Keyword Interface.

- uint32_t **putModuleVpdKeyword** (**ecmdChipTarget** &i_target, const char *i_record_name, const char *i_keyword, **ecmdDataBuffer** &i_data)

  Write Module VPD Keyword Interface.

- uint32_t **getModuleVpdImage** (**ecmdChipTarget** &i_target, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Read Module VPD Image Interface.

- uint32_t **putModuleVpdImage** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** &i_data)

  Write Module VPD Image Interface.

I2C Functions

- uint32_t **ecmdI2cReset** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_port)

  Resets the specified engine port.

- uint32_t **ecmdI2cRead** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_port, uint32_t i_slaveAddress, **ecmdI2cBusSpeed_t** i_busSpeed, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Read data from an I2C device.

- uint32_t **ecmdI2cReadOffset** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_port, uint32_t i_slaveAddress, **ecmdI2cBusSpeed_t** i_busSpeed, uint32_t i_offset, uint32_t i_offsetFieldSize, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Read data from an I2C device at the given offset.

- uint32_t **ecmdI2cWrite** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_port, uint32_t i_slaveAddress, **ecmdI2cBusSpeed_t** i_busSpeed, **ecmdDataBuffer** &i_data)

  Write the provided data into the I2C device.

- uint32_t **ecmdI2cWriteOffset** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_port, uint32_t i_slaveAddress, **ecmdI2cBusSpeed_t** i_busSpeed, uint32_t i_offset, uint32_t i_offsetFieldSize, **ecmdDataBuffer** &i_data)

  Write the provided data into the I2C device at the given offset.

GPIO Functions

- uint32_t **ecmdGpioConfigPin** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_pin, **ecmdDioMode_t** i_mode)

  Configures mode of pin.

- uint32_t **ecmdGpioReadPin** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_pin, uint32_t &o_state)

  Read the state of the specified pin (0/1).

- uint32_t **ecmdGpioReadLatch** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_pin, **ecmdDioMode_t** i_mode, uint32_t &o_state)

  Read the state of the latch.

- uint32_t **ecmdGpioWriteLatch** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_pin, **ecmdDioMode_t** i_mode, uint32_t i_state)

  Write value to the specified pin.

- uint32_t **ecmdGpioReadPins** (**ecmdChipTarget** &i_target, uint32_t i_engineId, uint32_t i_mask, uint32_t &o_value)

  Read the GPIO input register and AND with i_mask.

- uint32_t **ecmdGpioWriteLatches** (**ecmdChipTarget** &i_target, uint32_t i_engineId, **ecmdDioMode_t** i_mode, uint32_t i_mask, uint32_t i_value)

  Write several pins specified by i_mask.

6.4.1 Detailed Description

eCMD Perl API Usage :

```
use ecmd;
```

6.4.2 Function Documentation

6.4.2.1 int ecmdLoadDll (const char * *i_dllName*, const char * *i_clientVersion*)

Initialize the eCMD DLL.

Return values:
0 if successful load 1 if unsuccessful

Parameters:
i_dllName Full path and filename of the eCMD Dll to load
i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Precondition:
ecmdClientPerlapi constructor must have been called

- initializes the eCMD Dll.

VERSIONS :

eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple

versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE :

if (ecmdLoadDll("","ver3,ver4")) { die "Fatal errors initializing DLL"; }

6.4.2.2 void ecmdUnloadDll ()

Clean up the Perl Module - MUST BE CALLED JUST BEFORE SCRIPT EXIT.

6.4.2.3 int ecmdCommandArgs (char ** *io_argv*)

Pass any unknown command line paramaters to the DLL for processing (ex. -p#, Cronus -debug).

Return values:
ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:
io_argv Passed from Command line Arguments

Precondition:
initDll must have been called

Postcondition:
Global options (ex. -p#, -c#) will be removed from arg list

See also:
initDll

- argv gets passed to the eCMD DLL.
- Global options such as -p#, -c# will be parsed out.
- Target flags can be queried later with functions like ecmdQuerySelected

  NOTE : This function does not affect ring caching

USAGE :

ecmdCommandArgs(@ARGV);

6.4.2.4 void ecmdEnablePerlSafeMode ()

Enable Perl Error Handling Safe Mode - Croak on any eCMD error (default is enabled).

6.4.2.5 void ecmdDisablePerlSafeMode ()

Disable Perl Error Handling Safe Mode - Return eCMD errors to client (default is enabled).

6.4.2.6 bool ecmdParseOption (char ** *io_argv*, const char * *i_option*)

Iterates over argv, looking for given option string, removes it if found.

Return values:
1 if option found, 0 otherwise

Parameters:
io_argv Array of strings passed in from command line
i_option Option to look for

See also:
ecmdParseOptionWithArgs(p. 139)

6.4.2.7 char* ecmdParseOptionWithArgs (char ** *io_argv*, const char * *i_option*)

Iterates over argv, looking for given option string, removes it if found.

Return values:
Value of option arg if found, NULL otherwise

Parameters:
io_argv Array of strings passed in from command line
i_option Option to look for

See also:
ecmdParseOptionWithArgs(p. 139)

6.4.2.8 uint32_t ecmdQueryDllInfo (ecmdDllInfo & *o_dllInfo*)

Query information about the Dll that is loaded.

Parameters:
o_dllInfo Return data with data from the current dll loaded

Return values:
ECMD_SUCCESS if successful
nonzero on failure

This interface allows you to query what particular instance of the DLL is loaded (i.e Cronus/IP/Z), along with additional information.

NOTE : This function does not affect ring caching

6.4.2.9 bool ecmdQueryVersionGreater (const char * *version*)

Query to see if plugin is greater or equal to release specified.

Parameters:

version eCMD Release (ex "5.1", "6.3")

Return values:

true If the plugin release is >= version specified

false If the plugin release is < version specified

eCMD won't allow your code to run if the major numbers mismatch, but at times you may want to use a new function that was released in a minor release. This api let's you see if the plugin is at least or greater then the minor number where the new function was made available.

So if new function X was dropped in eCMD release v6.2 then if you include the following check in your code you can neatly handle if a user is trying to run a plugin that is 6.1 or less:

if (!ecmdQueryVersionGreater("6.2")) {

ecmdOutputWarning("Plugin doesn't support function X , skipping the new stuff\n");

}

6.4.2.10 uint32_t ecmdQueryConfig (ecmdChipTarget & *i_target*, ecmdQueryData & *o_queryData*, ecmdQueryDetail_t *i_detail* = ECMD_QUERY_DETAIL_HIGH)

Query configuration information from the DLL.

Parameters:

i_target Struct that contains partial information to limit query results

o_queryData Return data from query

i_detail Specify the level of detail that should be returned with the query

Return values:

ECMD_SUCCESS if successful

nonzero on failure

The Valid bits of the target are used to refine the query

The target paramater should be filled in with as much data as you know to limit the query, (including the chipType).

When a field state is set to ECMD_TARGET_FIELD_WILDCARD the query function will iterate on all possible values for that entry and return the relevant data.

When a field state is set to ECMD_TARGET_FIELD_UNUSED the query function will stop iterating at that level and below

Ex: to query what positions of the Nova chip are on cage 1, node 2:

cage = 1, node = 2, pos = 'wildcard', chipType = 'Nova', core = 'wildcard', thread = 'wildcard'

Ex: to query what positions of the Nova chip are in the entire system:

cage = 'wildcard', node = 'wildcard', pos = 'wildcard', chipType = 'Nova', core = 'wildcard', thread = 'wildcard'

Ex: to query all the chips on cage 3, node 0:

cage = 3, node = 0, pos = 'wildcard', chipType = 'wildcard', core = 'wildcard', thread = 'wildcard'

Ex: to query all the chips in the entire system:

cage = 'wildcard', node = 'wildcard', pos = 'wildcard', chipType = 'wildcard', core = 'wildcard', thread = 'wildcard'

Ex: to query the total nodes in a system:

cage = 'wildcard', node = 'wildcard', pos = 'ignore', chipType = 'ignore', core = 'ignore', thread = 'ignore'

NOTE : This function does not affect ring caching

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.4.2.11 uint32_t ecmdQuerySelected (ecmdChipTarget & *io_target*, ecmdQueryData & *o_queryData*, ecmdConfigLoopType_t *i_looptype* = ECMD_SELECTED_TARGETS_LOOP)

Query User Selected Targeting information from the DLL, i.e (-p#,-c#,-t#).

Parameters:

io_target Struct that contains partial information to limit query results - chipType is unused

o_queryData Return data from query

i_looptype (Optional) Used by config looper to specify different query modes

Return values:

ECMD_SUCCESS if successful

nonzero on failure

This function acts just like ecmdQueryConfig except it operates on what targets were selected by the user args -n#, -p#, -c#, -t#

Use of this function is the same as ecmdQueryConfig

When -talive is specified all threads configured will be returned in o_queryData and io_target.threadState will be set to ECMD_TARGET_THREAD_ALIVE.

NOTE : This function does not affect ring caching

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.4.2.12 uint32_t ecmdQueryRing (ecmdChipTarget & *i_target*, std::list< ecmdRingData > & *o_queryData*, const char * *i_ringName* = NULL, ecmdQueryDetail_t *i_detail* = ECMD_QUERY_DETAIL_HIGH)

Query Ring information from the DLL.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of chip to use

o_queryData Return list from query

i_ringName if != NULL used to refine query to a single ring

i_detail Specify the level of detail that should be returned with the query

Return values:

ECMD_INVALID_RING if i_ringName is not valid for target

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.13 uint32_t ecmdQueryLatch (ecmdChipTarget & *i_target*, std::list< ecmdLatchData > & *o_queryData*, ecmdLatchMode_t *i_mode*, const char * *i_latchName*, const char * *i_ringName* = NULL, ecmdQueryDetail_t *i_detail* = ECMD_QUERY_DETAIL_HIGH)

Query Latch information from the DLL.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of chip to use

o_queryData Return list from query

i_ringName if != NULL used to refine query to a single ring

i_latchName if != NULL used to refine query to a single latch

i_mode LatchName search mode (full or partial names)

i_detail Specify the level of detail that should be returned with the query

Return values:

ECMD_INVALID_RING if i_ringName is not valid for target

ECMD_INVALID_LATCHNAME if latchname not found in scandef

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

NOTE : i_latchName or i_ringName MUST be used, they can't both be NULL

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.14 uint32_t ecmdQueryArray (ecmdChipTarget & *i_target*, std::list< ecmdArrayData > & *o_queryData*, const char * *i_arrayName* = NULL, ecmdQueryDetail_t *i_detail* = ECMD_QUERY_DETAIL_HIGH)

Query Array information from the DLL.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information of chip to use

o_ queryData Return list from query

i_ arrayName if != NULL used to refine query to a single array

i_ detail Specify the level of detail that should be returned with the query

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ ARRAY if i_arrayName is not valid for target

ECMD_ SUCCESS if successful

nonzero on failure

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.15 uint32_t ecmdQuerySpy (ecmdChipTarget & *i_ target*, std::list< ecmdSpyData > & *o_ queryData*, const char * *i_ spyName* = NULL, ecmdQueryDetail_t *i_ detail* = ECMD_QUERY_DETAIL_HIGH)

Query Spy information from the DLL.

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information of chip to use

o_ queryData Return data from query

i_ spyName if != NULL used to refine query to a single spy

i_ detail Specify the level of detail that should be returned with the query

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ SUCCESS if successful

ECMD_ INVALID_ SPY if spy name is not valid for target

nonzero on failure

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.16 uint32_t ecmdQueryScom (ecmdChipTarget & *i_ target*, std::list< ecmdScomData > & *o_ queryData*, uint32_t *i_ address* = 0xFFFFFFFF, ecmdQueryDetail_t *i_ detail* = ECMD_QUERY_DETAIL_HIGH)

Query Scom information from the DLL.

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information of chip to use

o_queryData Return data from query

i_address if != 0xFFFFFFFF used to refine query to a single scom

i_detail Specify the level of detail that should be returned with the query

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.17 uint32_t ecmdQueryTraceArray (ecmdChipTarget & *i_target*, std::list< ecmdTraceArrayData > & *o_queryData*, const char * *i_traceArrayName* = NULL, ecmdQueryDetail_t *i_detail* = ECMD_QUERY_DETAIL_HIGH)

Query Trace Array information from the DLL.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information of chip to use

o_queryData Return data from query

i_traceArrayName if != NULL used to refine query to a single trace array

i_detail Specify the level of detail that should be returned with the query

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.18 uint32_t ecmdQueryFileLocation (ecmdChipTarget & *i_target*, ecmdFileType_t *i_fileType*, std::string & *o_fileLocation*)

Query the location of a specific file type for the selected target.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_fileType Enum that specifies which type of file you are looking for scandef/spydef/arraydef

o_fileLocation Return string with full path and filename to location

Return values:

ECMD_SUCCESS if successful

ECMD_UNKNOWN_FILE if unable to find requested file

nonzero if unsuccessful

NOTE : This function does not affect ring caching

TARGET DEPTH : Pos (where applicable based on i_fileType)

TARGET STATES : Unused

6.4.2.19 bool ecmdQueryTargetConfigured (ecmdChipTarget *i_target*, ecmdQueryData * *i_queryData* = NULL)

Query if a particular target is configured in the system.

Parameters:

i_target Target to query in system configuration

i_queryData If specified this data will be used, otherwise a call to ecmdQueryConfig will be made

Return values:

true if Target is configured in system

false if Target is not configured in system

NOTE : This function calls ecmdQueryConfig and searchs for the specified target

NOTE : The target State fields must be filled in as either VALID or UNUSED

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.4.2.20 uint32_t getRing (ecmdChipTarget & *i_target*, const char * *i_ringName*, ecmdDataBuffer & *o_data*)

Scans the ring from the selected chip into the data buffer.

Return values:

ECMD_INVALID_RING if ringname is not valid for target

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of ring to read

i_ringName Name of ring to read from

o_ data DataBuffer object that holds data read from ring

See also:
putRing(p. 146)

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.21 uint32_t putRing (ecmdChipTarget & *i_ target*, const char * *i_ ringName*, ecmdDataBuffer & *i_ data*)

Scans ring from the data buffer into the selected chip in the selected ring.

Return values:
ECMD_ SUCCESS if successful

nonzero if unsuccessful

ECMD_ DATA_ OVERFLOW Too much data was provided for a write

ECMD_ DATA_ UNDERFLOW Too little data was provided to a write function

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ RING if ringname is not valid for target

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:
i_ target Struct that contains chip and cage/node/slot/position/core information of ring to write

i_ ringName Name of ring to write to

i_ data DataBuffer object that holds data to write into ring

See also:
getRing(p. 145)

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.22 uint32_t getLatch (ecmdChipTarget & *i_ target*, const char * *i_ ringName*, const char * *i_ latchName*, std::list< ecmdLatchEntry > & *o_ data*, ecmdLatchMode_t *i_ mode*)

Reads the selected spy into the data buffer.

Return values:
ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read
ECMD_UNABLE_TO_OPEN_SCANDEF eCMD was unable to open the scandef
ECMD_INVALID_RING if ringname is not valid for target
ECMD_INVALID_LATCHNAME if latchname not found in scandef
nonzero if unsuccessful

Parameters:
i_target Struct that contains chip and cage/node/slot/position/core information
i_latchName Name of latch to read (can be a partial or full name based on i_mode)
o_data list of Entries containing all latches found matching i_latchName
i_ringName Name of ring to search for latch if == NULL, entire scandef is searched
i_mode LatchName search mode (full or partial names)

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.23 uint32_t putLatch (ecmdChipTarget & *i_target*, const char * *i_ringName*, const char * *i_latchName*, ecmdDataBuffer & *i_data*, uint32_t *i_startBit*, uint32_t *i_numBits*, uint32_t & *o_matchs*, ecmdLatchMode_t *i_mode*)

Writes the data buffer into the all latches matching i_latchName.

Return values:
ECMD_TARGET_NOT_CONFIGURED if target is not available in the system
ECMD_SUCCESS if successful
ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation
ECMD_UNABLE_TO_OPEN_SCANDEF eCMD was unable to open the scandef
ECMD_INVALID_RING if ringname is not valid for target
ECMD_INVALID_LATCHNAME if latchname not found in scandef
ECMD_DATA_OVERFLOW Too much data was provided for a write
ECMD_DATA_UNDERFLOW Too little data was provided to a write function
nonzero if unsuccessful

Parameters:
i_target Struct that contains chip and cage/node/slot/position/core information
i_latchName Name of latch to write (can be a partial or full name based on i_mode)
i_data DataBuffer object that holds data to write into latch
i_ringName Name of ring to search for latch if == NULL, entire scandef is searched
i_mode LatchName search mode
i_startBit Startbit in latchname to insert data
i_numBits Number of bits to insert from startbit
o_matchs Number of latchs found that matched your name and data was inserted

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.24 uint32_t getRingWithModifier (ecmdChipTarget & *i_target*, uint32_t *i_address*, uint32_t *i_bitLength*, ecmdDataBuffer & *o_data*)

Scans the specified number of bits from the selected chip and ring address into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of ring to read

i_address Address of ring to read from

i_bitLength Bit Length to scan for

o_data DataBuffer object that holds data read from ring

See also:

putRingWithModifier(p. 148)

NOTE : This is a debug interface and should not be used in normal situations

NOTE : This function does not handle processor cores for you, the i_address will be taken and used with no modifications so you are responsible for specifying the correct core address

NOTE : This function will only scan for the length provided, if this length doesn't match the actual length of the ring corruption may occur

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.25 uint32_t putRingWithModifier (ecmdChipTarget & *i_target*, uint32_t *i_address*, uint32_t *i_bitLength*, ecmdDataBuffer & *i_data*)

Scans the specified number of bits from the data buffer into the selected chip in the selected ring address.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_ target Struct that contains chip and cage/node/slot/position information of ring to write

i_ address Address of ring to write to

i_ bitLength Bit Length to scan for

i_ data DataBuffer object that holds data to write into ring

See also:

getRingWithModifier(p. 148)

NOTE : This is a debug interface and should not be used in normal situations

NOTE : This function does not handle processor cores for you, the i_address will be taken and used with no modifications so you are responsible for specifying the correct core address

NOTE : This function will only scan for the length provided, if this length doesn't match the actual length of the ring corruption may occur

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.26 uint32_t getScom (ecmdChipTarget & *i_ target*, uint32_t *i_ address*, ecmdDataBuffer & *o_ data*)

Scoms bits from the selected address into the data buffer.

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core information of scom address to read

i_ address Scom address to read from

o_ data DataBuffer object that holds data read from address

See also:

putScom(p. 150)

NOTE : For processor cores, only "core0 only" addresses are supported, other core addresses cause a failure

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.27 uint32_t putScom (ecmdChipTarget & *i_target*, uint32_t *i_address*, ecmdDataBuffer & *i_data*)

Scoms bits from the data buffer into the selected address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of scom address to write

i_address Scom address to write to

i_data DataBuffer object that holds data to write into address

See also:

getScom(p. 149)

NOTE : For processor cores, only "core0 only" addresses are supported, other core addresses cause a failure

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.28 uint32_t sendCmd (ecmdChipTarget & *i_target*, uint32_t *i_instruction*, uint32_t *i_modifier*, ecmdDataBuffer & *o_status*)

Send a JTAG instruction and modifier to the specified chip.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of scom address to write

i_instruction Right aligned instruction to send to chip

i_modifier Right aligned instruction modifier to send

o_status Instruction status register value retrieved

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_NON_JTAG_CHIP Chip Target is a non-jtag attached chip

nonzero if unsuccessful

NOTE : Proper parity will be generated on the command and modifier

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.29 uint32_t getCfamRegister (ecmdChipTarget & *i_target*, uint32_t *i_address*, ecmdDataBuffer & *o_data*)

Read data from the selected CFAM register address into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_NON_FSI_CHIP Targetted chip is not attached via FSI

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address CFAM address to read from

o_data DataBuffer object that holds data read from address

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.30 uint32_t putCfamRegister (ecmdChipTarget & *i_target*, uint32_t *i_address*, ecmdDataBuffer & *i_data*)

Write data into the selected CFAM register address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

ECMD_NON_FSI_CHIP Targetted chip is not attached via FSI

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address CFAM address to write to

i_ data DataBuffer object that holds data to write into address

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.31 uint32_t getSpy (ecmdChipTarget & *i_ target*, const char * *i_ spyName*, ecmdDataBuffer & *o_ data*)

Reads the selected spy into the data buffer.

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ SPY_ FAILED_ ECC_ CHECK if invalid ECC detected on Spy read

ECMD_ INVALID_ SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ SPY_ IS_ EDIAL Spy is an edial have to use getSpyEnum

ECMD_ SPY_ GROUP_ MISMATCH A mismatch was found reading a group spy not all groups set the same

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_ spyName Name of spy to read from

o_ data DataBuffer object that holds data read from spy

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.32 uint32_t getSpyEnum (ecmdChipTarget & *i_ target*, const char * *i_ spyName*, std::string & *o_ enumValue*)

Reads the selected spy and returns it's assocaiated enum.

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ SPY_ FAILED_ ECC_ CHECK if invalid ECC detected on Spy read - valid Spy Data still returned

ECMD_ INVALID_ SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_ INVALID_ SPY_ ENUM if value in hardware doesn't map to a valid enum

ECMD_ SPY_ NOT_ ENUMERATED Spy is not enumerated must use getSpy

ECMD_ SPY_ GROUP_ MISMATCH A mismatch was found reading a group spy not all groups set the same

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyName Name of spy to read from

o_enumValue Enum value read from the spy

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.33 uint32_t getSpyEpCheckers (ecmdChipTarget & *i_target*, const char * *i_spyEpCheckersName*, ecmdDataBuffer & *o_inLatchData*, ecmdDataBuffer & *o_outLatchData*, ecmdDataBuffer & *o_eccErrorMask*)

Read an ECC grouping and return the in and out bits as well as a error mask if any out bits are invalid.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is not an ECC Grouping

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read - valid Spy Data still returned

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyEpCheckersName Name of spy to read from

o_inLatchData Return the data for the input to the eccGroup

o_outLatchData Return the Ecc data associated with the outbits of the eccGroup

o_eccErrorMask Return a mask for the Ecc data a 1 in the mask means the associated eccData was in error

Return values:

nonzero if unsuccessful

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.34 uint32_t getSpyGroups (ecmdChipTarget & *i_target*, const char * *i_spyName*, std::list< ecmdSpyGroupData > & *o_groups*)

Reads the selected spy and load all the spy groups into provided list.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SPY_IS_EDIAL Spy is an edial have to use getSpyEnum

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyName Name of spy to read from

o_groups List of structures containing the group data and deadbits mask

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.35 uint32_t putSpy (ecmdChipTarget & *i_target*, const char * *i_spyName*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected spy.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SPY_IS_EDIAL Spy is an edial have to use putSpyEnum

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to write

i_spyName Name of spy to write to

i_data DataBuffer object that holds data to write into spy

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.36 uint32_t putSpyEnum (ecmdChipTarget & *i_target*, const char * *i_spyName*, const std::string *i_enumValue*)

Writes the enum into the selected spy.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping 2retval ECMD_SPY_NOT_ENUMERATED Spy is not enumerated must use putSpy

ECMD_INVALID_SPY_ENUM if enum value specified is not valid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to write

i_spyName Name of spy to write to

i_enumValue String enum value to load into the spy

NOTE : This function is ring cache enabled

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.37 void ecmdEnableRingCache ()

Enables internal caching of read/writes of scan rings to the chip for functions like getring/getspy/getspr.

Postcondition:

Ring caching is enabled on cache enabled functions

- Functions that support caching are documented in the detailed description of the function
- Functions that do not affect the state of the cache are documented in the detailed description of the function
- Any non-cache enabled function will force a flush of the cache before performing the operation
- Some Dll's may not support ring caching, they will not fail on these functions but you will not see the performance gains

6.4.2.38 uint32_t ecmdDisableRingCache ()

Disable internal caching of reads/writes of scan rings.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE: A Flush of the cache is performed before disabling the cache

6.4.2.39 uint32_t ecmdFlushRingCache ()

Flush all modified data from the internal cache to the hardware, then remove all rings from cache.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

6.4.2.40 bool ecmdIsRingCacheEnabled ()

Returns true/false to signify if caching is currently enabled.

Return values:

true if ring caching is enabled

false if ring caching is disabled

6.4.2.41 uint32_t getArray (ecmdChipTarget & *i_target*, const char * *i_arrayName*, ecmdDataBuffer & *i_address*, ecmdDataBuffer & *o_data*)

Reads bits from the selected array into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to read

i_arrayName Name of array to read from

o_data DataBuffer object that holds data read from address

i_address Array Address to read from - length of DataBuffer should be set to length of valid address data

See also:

putArray(p. 157)
getArrayMultiple(p. 157)

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.42 uint32_t getArrayMultiple (ecmdChipTarget & *i_target*, const char * *i_arrayName*, std::list< ecmdArrayEntry > & *io_entries*)

Reads bits from multiple array addresses/elements into the list of data buffers.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to read

i_arrayName Name of array to read from

io_entries list of array entries to fetch

See also:

putArray(p. 157)
getArray(p. 156)

NOTE : To use this function the io_entries list should be pre-loaded with the addresses to fetch, the associated dataBuffers will be loaded upon return

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.43 uint32_t putArray (ecmdChipTarget & *i_target*, const char * *i_arrayName*, ecmdDataBuffer & *i_address*, ecmdDataBuffer & *i_data*)

Writes bits from the data buffer into the selected array.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to write

i_arrayName Name of array to write to

i_data DataBuffer object that holds data to write into array

i_address Array Address to write to - length of DataBuffer should be set to length of valid address data

See also:

getArray(p. 156)

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.44 uint32_t putArrayMultiple (ecmdChipTarget & *i_target*, const char * *i_arrayName*, std::list< ecmdArrayEntry > & *i_entries*)

Writes bits from the list of entries into the selected array.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to write

i_arrayName Name of array to write to

i_entries List of addresses and data to write to chip

See also:

getArray(p. 156)

NOTE : i_entries should be pre-loaded with address and data

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.45 uint32_t ecmdQueryClockState (ecmdChipTarget & *i_target*, const char * *i_clockDomain*, ecmdClockState_t & *o_clockState*)

Query the state of the clocks for a domain.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_INVALID_CLOCK_DOMAIN An invalid clock domain name was specified

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_clockDomain Clock domain to query - as defined in scandef - use "ALL" to check all domains

o_clockState State of clocks for that domain

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.46 uint32_t startClocks (ecmdChipTarget & *i_target*, const char * *i_clockDomain*, bool *i_forceState* = false)

Start the clocks in the domain specified.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_INVALID_CLOCK_DOMAIN An invalid clock domain name was specified

ECMD_CLOCKS_ALREADY_ON The clocks in the specified domain are already on

ECMD_CLOCKS_IN_INVALID_STATE The clock in the specified domain are in an unknown state (not all on/off)

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_clockDomain Clock domain to start - as defined in scandef - use "ALL" to start all domains

i_forceState Force the clocks into the appropriate state - ignore if not in correct state to start

NOTE : If i_target refers to a particular chip object the i_clockDomain has to be "ALL" or a clock domain as defined in the scandef If i_target refers to a Cage/node then i_clockDomain has to be "ALL" or one of the predefined convenience clock domains as documented in the eCMD system spec for your particular product.

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.47 uint32_t stopClocks (ecmdChipTarget & *i_target*, const char * *i_clockDomain*, bool *i_forceState* = false)

Stop the clocks in the domain specified.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_INVALID_CLOCK_DOMAIN An invalid clock domain name was specified

ECMD_CLOCKS_ALREADY_OFF The clocks in the specified domain are already off

ECMD_CLOCKS_IN_INVALID_STATE The clock in the specified domain are in an unknown state (not all on/off)

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_clockDomain Clock domain to stop - as defined in scandef - use "ALL" to stop all domains

i_forceState Force the clocks into the appropriate state - ignore if not in correct state to start

NOTE : If i_target refers to a particular chip object the i_clockDomain has to be "ALL" or a clock domain as defined in the scandef If i_target refers to a Cage/node then i_clockDomain has to be "ALL" or one of the predefined convenience clock domains as documented in the eCMD system spec for your particular product.

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.48 uint32_t ecmdSetClockSpeed (ecmdChipTarget & *i_target*, ecmdClockType_t *i_type*, uint32_t *i_speed*, ecmdClockSpeedType_t *i_speedType*, ecmdClockSetMode_t *i_mode*, ecmdClockRange_t *i_range*)

Change a system clock speed without adjusting system initialization settings using speed value.

Parameters:

i_target Struct that contains chip and cage information

i_type Clock type to change in system

i_speed New speed value, specified in Mhz or micro-seconds based on i_speedType

i_speedType Specifies if i_speed is provided in frequency or cycle time

i_mode Do adjustment in one step or steer the clock to the new value

i_range Adjustments for the clock steer procedure

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Cage

TARGET STATES : Unused

6.4.2.49 uint32_t ecmdSetClockMultDiv (ecmdChipTarget & *i_target*, ecmdClockType_t *i_type*, uint32_t *i_multiplier*, uint32_t *i_divider*)

Change a system clock speed without adjusting system initialization settings using speed value.

Parameters:

i_target Struct that contains chip and cage information

i_type Clock type to change in system

i_multiplier Multiplier to use to program clock

i_divider Divider value to use to program clock

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Cage

TARGET STATES : Unused

6.4.2.50 uint32_t iStepsByNumber (ecmdDataBuffer & *i_steps*)

Run iSteps by number.

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ISTEPS_INVALID_STEP An invalid step number was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:

iSteps specified are complete

Parameters:

i_steps Bit mask defining which steps to run

NOTE - function returns on first failure and remaining steps are not run

6.4.2.51 uint32_t iStepsByName (std::string *i_stepName*)

Run a single iStep by name.

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ISTEPS_INVALID_STEP An invalid step name was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iStep specified is complete

Parameters:
i_ stepName List of iStep names to run

6.4.2.52 uint32_t iStepsByNameMultiple (std::list< std::string > *i_ stepNames*)

Run multiple iSteps by name.

Return values:
ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ ISTEPS_ INVALID_ STEP An invalid step name was provided

ECMD_ SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iSteps specified are complete

Parameters:
i_ stepNames List of iStep names to run

NOTE - Steps are run in order as is appropriate for proper system configuration, not by order provided in list

NOTE - function returns on first failure and remaining steps are not run

6.4.2.53 uint32_t iStepsByNameRange (std::string *i_ stepNameBegin*, std::string *i_ stepNameEnd*)

Run all iSteps by name starting with i_stepNameBegin and ending with i_stepNameEnd.

Return values:
ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ ISTEPS_ INVALID_ STEP An invalid step name was provided

ECMD_ SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iSteps specified are complete

Parameters:
i_ stepNameBegin Starting iStep to run

i_ stepNameEnd Ending iStep to run

NOTE - function returns on first failure and remaining steps are not run

6.4.2.54 uint32_t ecmdQueryProcRegisterInfo (ecmdChipTarget & *i_ target*, const char * *i_ name*, ecmdProcRegisterInfo & *o_ data*)

Query Information about a Processor Register (SPR/GPR/FPR).

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information

i_ name Name of the Register to fetch data about (can be either a specific SPR or GPR/FPR)

o_ data Data retrieved about the register

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ SPR Spr name is invalid

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.55 uint32_t getSpr (ecmdChipTarget & *i_ target*, const char * *i_ sprName*, ecmdDataBuffer & *o_ data*)

Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ SPR Spr name is invalid

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information

i_ sprName Name of spr to read from

o_ data DataBuffer object that holds data read from spr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.56 uint32_t getSprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdNameEntry > & *io_entries*)

Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_SPR Spr name is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

io_entries List of entries to fetch **ecmdNameEntry.name**(p. 81) field must be filled in

- NOTE : There are special keywords that can be specified to fetch groups of entries, they are used by adding only an entry to io_entries and setting **ecmdNameEntry.name**(p. 81) = -keyword-
  - "ALLTHREADED" : To fetch all threaded (replicated) SPR's for particular target
  - "ALLSHARED" : To fetch all non-threaded SPR's for particular target

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.57 uint32_t putSpr (ecmdChipTarget & *i_target*, const char * *i_sprName*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPR Spr name is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_sprName Name of spr to write to

i_data DataBuffer object that holds data to write into spr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.58 uint32_t putSprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdNameEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPR Spr name is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_entries List of entries to write all **ecmdNameEntry**(p. 81) fields must be filled in

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.59 uint32_t getGpr (ecmdChipTarget & *i_target*, uint32_t *i_gprNum*, ecmdDataBuffer & *o_data*)

Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_gprNum Number of gpr to read from

o_data DataBuffer object that holds data read from gpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.60 uint32_t getGprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 71) field must be filled in

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.61 uint32_t putGpr (ecmdChipTarget & *i_target*, uint32_t *i_gprNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_gprNum Number of gpr to write to

i_data DataBuffer object that holds data to write into gpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.62 uint32_t putGprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_ entries List of entries to write all **ecmdIndexEntry**(p. 71) fields must be filled in

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.63 uint32_t getFpr (ecmdChipTarget & *i_ target*, uint32_t *i_ fprNum*, ecmdDataBuffer & *o_ data*)

Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ INVALID_ FPR Fpr number is invalid

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position/core/thread information

i_ fprNum Number of fpr to read from

o_ data DataBuffer object that holds data read from fpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.64 uint32_t getFprMultiple (ecmdChipTarget & *i_ target*, std::list< ecmdIndexEntry > & *io_ entries*)

Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ FPR Fpr number is invalid

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 71) field must be filled in

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.65 uint32_t putFpr (ecmdChipTarget & *i_target*, uint32_t *i_fprNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_FPR Fpr number is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_fprNum Number of fpr to write to

i_data DataBuffer object that holds data to write into fpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.66 uint32_t putFprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_FPR Fpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_entries List of entries to write all **ecmdIndexEntry**(p. 71) fields must be filled in

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.67 uint32_t getSlb (ecmdChipTarget & *i_target*, uint32_t *i_slbNum*, ecmdDataBuffer & *o_data*)

Reads the selected Processor SLB Entry into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_ENTRY_REQUESTED Slb number is invalid

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_slbNum Number of fpr to read from

o_data DataBuffer object that holds data read from fpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.68 uint32_t getSlbMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

Reads the selected Processor SLB Entry into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ENTRY_REQUESTED Slb number is invalid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 71) field must be filled in with slb number

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.69 uint32_t putSlb (ecmdChipTarget & *i_target*, uint32_t *i_slbNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor SLB Entry.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_ENTRY_REQUESTED Slb number is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_slbNum Number of fpr to write to

i_data DataBuffer object that holds data to write into fpr

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.70 uint32_t putSlbMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

Writes the data buffer into the selected Processor SLB Entry.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ENTRY_REQUESTED Slb number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_entries List of entries to write all **ecmdIndexEntry**(p. 71) fields must be filled in with slb number

The return value of this function is set to the first non-zero return code found when writing multiple entries. The function will NOT continue through all subsequent entries.

TARGET DEPTH : Thread

TARGET STATES : Unused

6.4.2.71 uint32_t getTraceArray (ecmdChipTarget & *i_target*, const char * *i_name*, bool *i_doTraceStopStart*, std::vector< ecmdDataBuffer > & *o_data*)

Dump all entries of specified trace array.

Parameters:

i_target Target info to specify what to configure (target states must be set)

i_name Name of trace array - names may vary for each product/chip

i_doTraceStopStart If true disable trace arrays before logging and renable after completion, if false client is in control

o_data Vector of trace array data retrieved

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY Invalid trace array name specified

ECMD_SUCCESS if successful

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.72 uint32_t getTraceArrayMultiple (ecmdChipTarget & *i_target*, bool *i_doTraceStopStart*, std::list< ecmdNameVectorEntry > & *o_data*)

Dump all entries of specified trace array.

Parameters:

i_target Target info to specify what to configure (target states must be set)

i_doTraceStopStart If true disable trace arrays before logging and renable after completion, if false client is in control

o_data List of trace array data retrieved

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY Invalid trace array name specified

ECMD_SUCCESS if successful

The return value of this function is set to the first non-zero return code found when retrieving multiple entries. The entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

- NOTE : to fetch all Trace Arrays available add only one entry to io_entries and set **ecmdNameVectorEntry.name**(p. 82) = "ALL"

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.73 uint32_t getMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Reads System Mainstore through the processor chip using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to read from

i_bytes Number of bytes to write

o_data DataBuffer object that holds data read from memory

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.74 uint32_t putMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_data*)

Writes System Mainstore through the processor chip using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to write to

i_bytes Number of bytes to write

i_data DataBuffer object that holds data to write into memory

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.75 uint32_t getMemDma (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Reads System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains cage/node information

i_address Starting address to read from

i_bytes Number of bytes to write

o_data DataBuffer object that holds data read from memory

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.76 uint32_t putMemDma (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_data*)

Writes System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains cage/node information

i_address Starting address to write to

i_bytes Number of bytes to write

i_data DataBuffer object that holds data to write into memory

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.77 uint32_t getMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Reads System Mainstore through the memory controller using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to read from

i_bytes Number of bytes to write

o_data DataBuffer object that holds data read from memory

WARNING : This operation is typically not cache-coherent

TARGET DEPTH : Cage

TARGET STATES : Unused

6.4.2.78 uint32_t putMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_data*)

Writes System Mainstore through the memory controller using a real address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to write to

i_bytes Number of bytes to write

i_data DataBuffer object that holds data to write into memory

WARNING : This operation is typically not cache-coherent

TARGET DEPTH : Cage

TARGET STATES : Unused

6.4.2.79 uint32_t ecmdCacheFlush (ecmdChipTarget & *i_target*, ecmdCacheType_t *i_cacheType*)

Cache Flush.

Parameters:

i_target ecmdChipTarget_t struct defines what chip(s) to flush

i_cacheType ecmdCacheType_t struct defines what type of cache gets flushed

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Core

TARGET STATES : Unused

6.4.2.80 uint32_t simaet (const char * *i_function*)

Enable/Disable Simulation AET Logging.

Parameters:

i_function Should be either 'on'/'off'/'flush'

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.81 uint32_t simcheckpoint (const char * *i_checkpoint*)

Store a checkpoint to specified file.

Parameters:

i_checkpoint Name of checkpoint to write to

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.82 uint32_t simclock (uint32_t *i_cycles*)

Clock the model.

Parameters:

i_cycles Number of cycles to clock model

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.83 uint32_t simecho (const char * *i_message*)

Echo message to stdout and sim log.

Parameters:

i_message Message to echo

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.84 uint32_t simexit (uint32_t *i_rc* = 0, const char * *i_message* = NULL)

Close down the simulation model.

Parameters:

i_rc [Optional] Send a testcase failure return code to the simulation

i_message [Optional[Send a testcase failure message to the simulation

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.85 uint32_t simEXPECTFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Perform expect on facility using name.

Parameters:

i_facname Facility name

i_expect Value to expect on facility

i_bitlength Length of data to expect

i_row Optional: Array Facility row

i_offset Optional: Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.86 uint32_t simexpecttcfac (const char * *i_tcfacname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0)

Perform expect on TCFAC facility.

Parameters:

i_tcfacname Facility name

i_expect Value to expect on facility

i_bitlength Length of data to expect

i_row Optional: Array Facility row

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.87 uint32_t simgetcurrentcycle (uint64_t & *o_cyclecount*)

Fetch current model cycle count.

Parameters:

o_cyclecount Current model cycle count

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.88 uint32_t simGETFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *o_data*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Retrieve a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to read from facility

o_data Data read from facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.89 uint32_t simGETFACX (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *o_data*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Retrieve a Facility using a name - preserving Xstate.

Parameters:

i_facname Facility name

i_bitlength Bit length to read from facility

o_data Data read from facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.90 uint32_t simgettcfac (const char * *i_tcfacname*, ecmdDataBuffer & *o_data*, uint32_t *i_row* = 0, uint32_t *i_startbit* = 0, uint32_t *i_bitlength* = 0)

Retrieve a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

o_data Value read

i_row Optional: Array Facility row

i_startbit Optional: Startbit to read

i_bitlength Optional: Length of data to read

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.91 uint32_t siminit (const char * *i_checkpoint*)

Initialize the simulation.

Parameters:

i_checkpoint Checkpoint to load : 'none' to skip

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.92 uint32_t simPOLLFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0, uint32_t *i_maxcycles* = 1, uint32_t *i_pollinterval* = 1)

Poll a facility waiting for expected value.

Parameters:

i_facname Facility name

i_bitlength Bit length to expect

i_expect Data to expect in facility

i_row Optional: Array row

i_offset Optional : Facility offset

i_maxcycles Optional : Maximum number of cycles to run

i_pollinterval Option : Number of clock cycles to run between each poll

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_POLLING_FAILURE Polling completed without reaching expected value

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.93 uint32_t simpolltcfac (const char * *i_tcfacname*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0, uint32_t *i_startbit* = 0, uint32_t *i_bitlength* = 0, uint32_t *i_maxcycles* = 1, uint32_t *i_pollinterval* = 1)

Poll a TCFAC facility waiting for expected value.

Parameters:

i_tcfacname Facility name

i_bitlength Bit length to expect

i_expect Data to expect in facility

i_row Optional: Array row

i_startbit Optional : Facility startbit

i_maxcycles Optional : Maximum number of cycles to run

i_pollinterval Option : Number of clock cycles to run between each poll

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_POLLING_FAILURE Polling completed without reaching expected value

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.94 uint32_t simPUTFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Write a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to write to facility

i_data Data to write

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.95 uint32_t simPUTFACX (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Write a Facility using a name - preserving Xstate.

Parameters:

i_facname Facility name

i_bitlength Bit length to write to facility

i_data Data to write

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.96 uint32_t simputtcfac (const char * *i_tcfacname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_numrows* = 0)

Write a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to write

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to write

i_bitlength Bit length to write to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.97 uint32_t simrestart (const char * *i_checkpoint*)

Load a checkpoint into model.

Parameters:

i_checkpoint Name of checkpoint

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.98 uint32_t simSTKFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Stick a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to stick to facility

i_data Data to stick

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.99 uint32_t simstktcfac (const char * *i_tcfacname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_numrows* = 0)

Stick a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to stick

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to stick

i_bitlength Bit length to write to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.100 uint32_t simSUBCMD (const char * *i_command*)

Run RTX SUBCMD.

Parameters:

i_command Command

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.101 uint32_t simtckinterval (uint32_t *i_tckinterval*)

Set TCK Interval setting in the model for JTAG Master.

Parameters:

i_tckinterval new setting for tck interval when using JTAG

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.102 uint32_t simUNSTICK (const char * *i_facname*, uint32_t *i_bitlength*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0)

Unstick a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to unstick to facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

6.4.2.103 uint32_t simunsticktcfac (const char * *i_tcfacname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_data*, uint32_t *i_row* = 0, uint32_t *i_numrows* = 0)

Unstick a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to unstick to

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to unstick

i_bitlength Bit length to unstick to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

6.4.2.104 uint32_t simGetHierarchy (ecmdChipTarget & *i_target*, std::string & *o_hierarchy*)

Fetch the hierarchy for the specified chip target relative to the latch names in the scandef.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information

o_hierarchy Return the model hierarchy for this target

NOTE - To retrieve the hierarchy of a processor core the core field must be set and the state set to ECMD_TARGET_FIELD_VALID

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.105 uint32_t ecmdQueryChipSimModelVersion (ecmdChipTarget & *i_target*, std::string & *o_timestamp*)

Will retrieve the model timestamp from the simulation, in hardware mode "NA" is returned.

Parameters:

i_target Target to query for information

o_timestamp Timestamp value from simulation model

Return values:

ECMD_SUCCESS on success

non-zero on failure

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.106 uint32_t ecmdQueryChipScandefVersion (ecmdChipTarget & *i_target*, std::string & *o_timestamp*)

Will retrieve the scandef timestamp from the scandef being used for the specified target.

Parameters:

i_target Target to query for information

o_timestamp Timestamp value from scandef

Return values:

ECMD_SUCCESS on success

non-zero on failure

TARGET DEPTH : Pos

TARGET STATES : Unused

6.4.2.107 std::string simCallFusionCommand (const char * *i_fusionObject*, const char * *i_replicaID*, const char * *i_command*)

Run a command on another Fusion module.

Parameters:

i_fusionObject Name of Fusion module to run against

i_replicaID Id of Fusion module

i_command Command to run

Return values:

Results of command

NOTE - The fusion module has to provide the appropriate api for this call to be functional

6.4.2.108 uint32_t simFusionRand32 (uint32_t *i_min* = 0, uint32_t *i_max* = ~0UL, const char * *i_fusionRandObject* = NULL)

Returns a random 32 bit number in the range [min,max] using the Fusion MasterSeed; by default each client will have an own instance of Fusion's Random32BitNumber object, but the user can specify the object's name and reuse the same object across multiple clients; if no range is specified 0 and MAXINT32 will be used.

Parameters:

i_min lower bound for random number

i_max upper bound for random number

i_fusionRandObject name of Fusion random number object to use; if not specified, each this client will get a unique instance of the class

Return values:

Random number

6.4.2.109 uint32_t simOutputFusionMessage (const char * *i_header*, const char * *i_message*, ecmdFusionSeverity_t *i_severity*, ecmdFusionMessageType_t *i_type*, const char * *i_file* = NULL, uint32_t *i_line* = 0)

Echo Messages to Fusion logs.

Parameters:

i_header Message header

i_message Message text

i_severity Severity

i_type Message type

i_file File where message originated

i_line Line number where message originated

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.110 void simSetFusionMessageFormat (const char * *i_format*)

Set Fusion Message Format.

Parameters:

i_format New Format

6.4.2.111 uint32_t simPutDial (const char * *i_dialName*, const std::string *i_enumValue*)

Write a simulation dial with specified value.

Parameters:

i_dialName Fully qualified dial name

i_enumValue Value to set dial to either enum or numeric (ie 0b11 or 0xFE)

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.112 uint32_t simGetDial (const char * *i_dialName*, std::string & *o_enumValue*)

Read a simulation dial.

Parameters:

i_dialName Fully qualified dial name

o_enumValue Value read from model

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.113 uint32_t simGetOutFile (const char * *i_filename*, std::string & *o_absFilename*)

Obtain absolute filename of a file that will be placed in the SIMOUT directory of the server / Fusion process and add new file to the bom information in the SUM file.

Parameters:

i_filename filename (w/o path information) of the file to create / lookup in the SIMOUT directory

o_absFilename will contain the absolute filename upon successful return from the call

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.114 uint32_t simGetInFile (const char * *i_filename*, std::string & *o_absFilename*)

Resolve absolute filename of a file by searching the SIMIN paths and add new file to the bom information in the SUM file.

Parameters:

i_filename name (w/o path information) of the file to lookup in the SIMIN directories

o_absFilename will contain the absolute filename upon successful return from the call

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.115 uint32_t simGetEnvironment (const char * *i_envName*, std::string & *o_envValue*)

Retrieve value of an environment variable on the server side.

Parameters:

i_envName name of environment variable to retrieve

o_envValue will contain the envvar's value upon successful return from the call

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.116 uint32_t simGetModelInfo (ecmdSimModelInfo & *o_modelInfo*)

Query information about the model from the server.

Parameters:

o_modelInfo pointer to a user-provided sd_model_info struct; SDAPI will fill the members of this struct upon successful return from the call

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.117 std::string ecmdGetErrorMsg (uint32_t *i_errorCode*, bool *i_parseReturnCode* = true)

Retrieve additional error information for errorcode.

Parameters:

i_errorCode Error code to lookup up message for

i_parseReturnCode If true will search through return codes definitions to return define name of error code

Return values:

point to NULL terminated string containing error data, NULL if error occurs

6.4.2.118 uint32_t ecmdRegisterErrorMsg (uint32_t *i_errorCode*, const char * *i_whom*, const char * *i_message*)

Register an error message that has occured.

6.4.2.119 void ecmdFlushRegisteredErrorMsgs ()

Flush all registered messages, they are no long retrievable.

6.4.2.120 void ecmdOutputError (const char * *i_message*)

Output a message related to an error.

Parameters:
i_message String to output

6.4.2.121 void ecmdOutputWarning (const char * *i_message*)

Output a message related to an warning.

Parameters:
i_message String to output

6.4.2.122 void ecmdOutput (const char * *i_message*)

Output a message to the screen or logs.

Parameters:
i_message String to output

6.4.2.123 uint32_t ecmdGetGlobalVar (ecmdGlobalVarType_t *i_type*)

Retrieve the value of some ecmdGlobalVars.

Parameters:
i_type Specifies which global var you are looking for

Return values:
Value of global var

6.4.2.124 void ecmdSetTraceMode (ecmdTraceType_t *i_type*, bool *i_enable*)

Enable/Disable a trace mode.

Parameters:
i_type Specifies which trace mode to enable
i_enable Enable or disable

6.4.2.125 bool ecmdQueryTraceMode (ecmdTraceType_t *i_type*)

Query the state of a trace mode.

Parameters:

i_type Specifies which trace mode to query

Return values:

Value of trace mode enable

6.4.2.126 uint32_t ecmdDelay (uint32_t *i_simCycles*, uint32_t *i_msDelay*)

Function to delay a procedure either by running sim cycles or by doing a millisecond delay.

Parameters:

i_simCycles Number of sim cycles to run in simulation mode

i_msDelay Number of milliseconds to delay in hardware mode

Return values:

ECMD_SUCCESS on success

non-zero on failure

6.4.2.127 uint32_t makeSPSystemCall (ecmdChipTarget & *i_target*, const std::string & *i_command*, std::string & *o_stdout*)

Make a system call on the targetted Service Processor or Service Element.

Parameters:

i_target SP to run command on

i_command Command line call to make

o_stdout Standard out captured by running command

TARGET DEPTH : Node TARGET STATES : Unused

6.4.2.128 uint32_t ecmdGetConfiguration (ecmdChipTarget & *i_target*, std::string *i_name*, ecmdConfigValid_t & *o_validOutput*, std::string & *o_valueAlpha*, uint32_t & *o_valueNumeric*)

Retrieve the value of a Configuration Setting.

Parameters:

i_target struct that contains chip and cage/node/slot/position/core information if necessary

i_name Name of setting as defined by eCMD Api

o_validOutput Indicator if o_valueAlpha, o_valueNumeric (or both) are valid.

o_valueAlpha Alpha value of setting (if appropriate)

o_valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_INVALID_CONFIG_NAME Name specified is not valid

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

TARGET DEPTH : Thread (depending on value of i_name)

TARGET STATES : Unused

6.4.2.129 uint32_t ecmdSetConfiguration (ecmdChipTarget & *i_target*, std::string *i_name*, ecmdConfigValid_t *i_validInput*, std::string *i_valueAlpha*, uint32_t *i_valueNumeric*)

Set the value of a Configuration Setting.

Parameters:

i_target struct that contains chip and cage/node/slot/position/core information if necessary

i_name Name of setting as defined by eCMD Api

i_validInput Indicator if i_valueAlpha, i_valueNumeric (or both) are valid.

i_valueAlpha Alpha value of setting (if appropriate)

i_valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT Value is not in correct format for specified configuration setting

ECMD_INVALID_CONFIG_NAME Name specified is not valid

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

TARGET DEPTH : Thread (depending on value of i_name)

TARGET STATES : Unused

6.4.2.130 uint32_t ecmdDeconfigureTarget (ecmdChipTarget & *i_target*)

Deconfigure a target in the system.

Parameters:

i_target Target info to specify what to deconfigure (target states must be set)

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero on failure

NOTE - lowest state that is valid is level that is deconfigured.

ex - if coreState is VALID the core selected is deconfigured

ex - if coreState is UNUSED and posState is VALID then the pos is deconfigured

This interface allows you to deconfigure all levels cages, nodes, slots, pos's, cores

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.131 uint32_t ecmdConfigureTarget (ecmdChipTarget & *i_target*)

Configure a target in the system - must be previously known to the system.

Parameters:

i_target Target info to specify what to configure (target states must be set)

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system, or was not previously deconfigured

ECMD_SUCCESS if successful

nonzero on failure

NOTE - lowest state that is valid is level that is configured.

ex - if coreState is VALID the core selected is configured

ex - if coreState is UNUSED and posState is VALID then the pos is configured

This interface allows you to configure all levels cages, nodes, slots, pos's, cores

TARGET DEPTH : Core

TARGET STATES : Must be Initialized

6.4.2.132 uint32_t ecmdTargetToUnitId (ecmdChipTarget & *io_target*)

Converts an eCmd (physical) Target to a HOM Unit Id.

Parameters:

io_target an **ecmdChipTarget**(p. 20) struct representing a specific eCmd target

Return values:

ECMD_SUCCESS if conversion successful

ECMD_INVALID_ARGS if unsuccessful in finding a matching Unit ID

Postcondition:

HOM Unit Ids in **ecmdChipTarget**(p. 20) struct are set and valid

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.4.2.133 uint32_t ecmdUnitIdStringToTarget (std::string *i_unitId*, std::list< ecmdChipTarget > & *o_targetList*)

Converts a Unit Id String to an eCmd (physical) Target.

Parameters:

i_ unitId a string representing the name of a unitId

o_ targetList a list of targets that match the input unitId string

Return values:

ECMD_ SUCCESS if conversion successful

ECMD_ INVALID_ ARGS if unsuccessful in matching the string to a target

Postcondition:

There will be a list ecmdChipTargets that represent the passed in unitId string

6.4.2.134 uint32_t ecmdUnitIdToTarget (uint32_t *i_ unitId*, std::list< ecmdChipTarget > & *o_ targetList*)

Converts a Unit Id to an eCmd (physical) Target.

Parameters:

i_ unitId a uint32_t representing an unitID

o_ targetList a list of targets that match the unitId input

Return values:

ECMD_ SUCCESS if conversion successful

ECMD_ INVALID_ ARGS if unsuccessful in matching the string to a target

Postcondition:

ecmdChipTarget(p. 20) Fields are set and represent the passed in unitId string

6.4.2.135 uint32_t ecmdUnitIdToString (uint32_t *i_ unitId*, std::string & *o_ unitIdStr*)

Converts a Unit Id into its String representation.

Parameters:

i_ unitId a uint32_t representing an unitID

o_ unitIdStr a string to match the unitId input

Return values:

ECMD_ SUCCESS if conversion successful

ECMD_ INVALID_ ARGS if unsuccessful in matching the unitID to a String

Postcondition:

HOM Unit Id String is set and represents the passed in uint32_t unitId

6.4.2.136 uint32_t ecmdSequenceIdToTarget (uint32_t *i_core_seq_num*, ecmdChipTarget & *io_target*, uint32_t *i_thread_seq_num* = 0)

Sequence ID of Cores and Threads converted to **ecmdChipTarget**(p. 20) struct.

Parameters:

i_core_seq_num Sequence ID number of the core

io_target **ecmdChipTarget**(p. 20) struct set to the result of the conversion

i_thread_seq_num (OPTIONAL, default to 0) Sequence ID number of thread relative to the core parm

Return values:

ECMD_INVALID_ARGS the inputs could not be mapped to an **ecmdChipTarget**(p. 20) struct

ECMD_SUCCESS if successful

non-zero if unsuccessful

Precondition:

io_target must have states defined to either core or thread level
io_target must have an acceptable processor chipType

Postcondition:

io_target fields are set accordingly

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.4.2.137 uint32_t getModuleVpdKeyword (ecmdChipTarget & *i_target*, const char * *i_record_name*, const char * *i_keyword*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Read Module VPD Keyword Interface.

Parameters:

i_target Struct that contains cage/node/slot/position of module vpd to access

i_record_name Name of VPD Record for given keyword

i_keyword Name of VPD Keyword to Read

i_bytes Byte length to read

o_data Data buffer to copy data to

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS the inputs could not be mapped to a keyword

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.138 uint32_t putModuleVpdKeyword (ecmdChipTarget & *i_target*, const char * *i_record_name*, const char * *i_keyword*, ecmdDataBuffer & *i_data*)

Write Module VPD Keyword Interface.

Parameters:

i_target Struct that contains cage/node/slot/position of module vpd to access

i_record_name Name of VPD Record for given keyword

i_keyword Name of VPD Keyword to Write

i_data Data buffer of data to write

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS the inputs could not be mapped to a keyword

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.139 uint32_t getModuleVpdImage (ecmdChipTarget & *i_target*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Read Module VPD Image Interface.

Parameters:

i_target Struct that contains cage/node/slot/position of module vpd to access

i_bytes Byte length to read

o_data Data buffer of data read from module

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS the inputs could not be mapped to a module

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.140 uint32_t putModuleVpdImage (ecmdChipTarget & *i_target*, ecmdDataBuffer & *i_data*)

Write Module VPD Image Interface.

Parameters:

i_target Struct that contains cage/node/slot/position of module vpd to access

i_ data Data buffer of data to write

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ INVALID_ ARGS the inputs could not be mapped to a module

ECMD_ DATA_ OVERFLOW Too much data was provided for a write

ECMD_ DATA_ UNDERFLOW Too little data was provided to a write function

ECMD_ SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.141 uint32_t ecmdI2cReset (ecmdChipTarget & *i_ target*, uint32_t *i_ engineId*, uint32_t *i_ port*)

Resets the specified engine port.

Parameters:

i_ target Struct that contains cage/node/slot/position of device to access

i_ engineId I2c engine to use

i_ port I2C port to use

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ TARGET_ INVALID_ TYPE if target doesn't support I2c

ECMD_ INVALID_ ARGS Invalid argument values found

ECMD_ SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.142 uint32_t ecmdI2cRead (ecmdChipTarget & *i_ target*, uint32_t *i_ engineId*, uint32_t *i_ port*, uint32_t *i_ slaveAddress*, ecmdI2cBusSpeed_t *i_ busSpeed*, uint32_t *i_ bytes*, ecmdDataBuffer & *o_ data*)

Read data from an I2C device.

Parameters:

i_ target Struct that contains cage/node/slot/position of device to access

i_ engineId I2c engine to use

i_ port I2C port to use

i_ slaveAddress I2C slave device address to use

i_ busSpeed I2C Bus speed to use

i_ bytes Byte length to read

o_ data Data read from device

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ TARGET_ INVALID_ TYPE if target doesn't support I2c

ECMD_ INVALID_ ARGS Invalid argument values found

ECMD_ SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.143 uint32_t ecmdI2cReadOffset (ecmdChipTarget & *i_ target*, uint32_t *i_ engineId*, uint32_t *i_ port*, uint32_t *i_ slaveAddress*, ecmdI2cBusSpeed_t *i_ busSpeed*, uint32_t *i_ offset*, uint32_t *i_ offsetFieldSize*, uint32_t *i_ bytes*, ecmdDataBuffer & *o_ data*)

Read data from an I2C device at the given offset.

Parameters:

i_ target Struct that contains cage/node/slot/position of device to access

i_ engineId I2c engine to use

i_ port I2C port to use

i_ slaveAddress I2C slave device address to use

i_ busSpeed I2C Bus speed to use

i_ offset Byte offset in the device

i_ offsetFieldSize Specifies the field size used in the I2C protocol of the slave device

i_ bytes Byte length to read

o_ data Data read from device

Return values:

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ TARGET_ INVALID_ TYPE if target doesn't support I2c

ECMD_ INVALID_ ARGS Invalid argument values found

ECMD_ SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.144 uint32_t ecmdI2cWrite (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_port*, uint32_t *i_slaveAddress*, ecmdI2cBusSpeed_t *i_busSpeed*, ecmdDataBuffer & *i_data*)

Write the provided data into the I2C device.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access
i_engineId I2c engine to use
i_port I2C port to use
i_slaveAddress I2C slave device address to use
i_busSpeed I2C Bus speed to use
i_data Data to write to device

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system
ECMD_TARGET_INVALID_TYPE if target doesn't support I2c
ECMD_INVALID_ARGS Invalid argument values found
ECMD_SUCCESS if successful
non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.145 uint32_t ecmdI2cWriteOffset (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_port*, uint32_t *i_slaveAddress*, ecmdI2cBusSpeed_t *i_busSpeed*, uint32_t *i_offset*, uint32_t *i_offsetFieldSize*, ecmdDataBuffer & *i_data*)

Write the provided data into the I2C device at the given offset.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access
i_engineId I2c engine to use
i_port I2C port to use
i_slaveAddress I2C slave device address to use
i_busSpeed I2C Bus speed to use
i_offset Byte offset in the device
i_offsetFieldSize Specifies the field size used in the I2C protocol of the slave device
i_data Data to write to device

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system
ECMD_TARGET_INVALID_TYPE if target doesn't support I2c
ECMD_INVALID_ARGS Invalid argument values found
ECMD_SUCCESS if successful
non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.146 uint32_t ecmdGpioConfigPin (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_pin*, ecmdDioMode_t *i_mode*)

Configures mode of pin.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_pin Pin to use

i_mode Mode to configure pin

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

NOTE : Configuring a pin explicitly as output is not necessary since the write latch commands implicitly perform the required settings.

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.147 uint32_t ecmdGpioReadPin (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_pin*, uint32_t & *o_state*)

Read the state of the specified pin (0/1).

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_pin Pin to use

o_state State read on pin (0/1)

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.148 uint32_t ecmdGpioReadLatch (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_pin*, ecmdDioMode_t *i_mode*, uint32_t & *o_state*)

Read the state of the latch.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_pin Pin to use

i_mode Mode to configure pin

o_state State read on pin (0/1)

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.149 uint32_t ecmdGpioWriteLatch (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_pin*, ecmdDioMode_t *i_mode*, uint32_t *i_state*)

Write value to the specified pin.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_pin Pin to use

i_mode Mode to configure pin

i_state State to write to pin

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.150 uint32_t ecmdGpioReadPins (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, uint32_t *i_mask*, uint32_t & *o_value*)

Read the GPIO input register and AND with i_mask.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_mask Mask to apply to pins

o_value Value read from pins with mask applied

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.4.2.151 uint32_t ecmdGpioWriteLatches (ecmdChipTarget & *i_target*, uint32_t *i_engineId*, ecmdDioMode_t *i_mode*, uint32_t *i_mask*, uint32_t *i_value*)

Write several pins specified by i_mask.

Parameters:

i_target Struct that contains cage/node/slot/position of device to access

i_engineId Gpio engine to use

i_mask Mask to apply to pins

i_mode Mode to configure pin

i_value Value to write to pins with mask applied

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_TARGET_INVALID_TYPE if target doesn't support Gpio

ECMD_INVALID_ARGS Invalid argument values found

ECMD_SUCCESS if successful

non-zero if unsuccessful

TARGET DEPTH : Chip

TARGET STATES : Unused

6.5 ecmdDataBuffer.H File Reference

Provides a means to handle data from the eCMD C API.

```
#include <string>
```

```
#include <inttypes.h>
```

Classes

- class **ecmdDataBufferImplementationHelper**

  This is used to help low-level implementation of the **ecmdDataBuffer**(p. 24)*, this CAN NOT be used by any eCMD client or data corruption will occur.*

- class **ecmdDataBuffer**

  Provides a means to handle data from the eCMD C API.

- class **ecmdOptimizableDataBuffer**

Defines

- #define **ECMD_DBUF_SUCCESS** 0x0

  DataBuffer returned successfully.

- #define **ECMD_DBUF_INIT_FAIL** (0x01000000 | 0x2000)

  Initialization of the DataBuffer failed.

- #define **ECMD_DBUF_BUFFER_OVERFLOW** (0x01000000 | 0x2010)

  Attempt to read/write data beyond the length of the DataBuffer.

- #define **ECMD_DBUF_XSTATE_ERROR** (0x01000000 | 0x2020)

  An 'X' character occured where it was not expected.

- #define **ECMD_DBUF_UNDEFINED_FUNCTION** (0x01000000 | 0x2030)

  Function not included in this version of DataBuffer.

- #define **ECMD_DBUF_INVALID_ARGS** (0x01000000 | 0x2040)

  Args provided to dataBuffer were invalid.

- #define **ECMD_DBUF_INVALID_DATA_FORMAT** (0x01000000 | 0x2041)

  String data didn't match expected input format.

- #define **ECMD_DBUF_FOPEN_FAIL** (0x01000000 | 0x2050)

  File open on file for reading or writing the data buffer failed.

- #define **ECMD_DBUF_FILE_FORMAT_MISMATCH** (0x01000000 | 0x2051)

  In readFile specified format not found in the data file.

- #define **ECMD_DBUF_DATANUMBER_NOT_FOUND** (0x01000000 | 0x2052)

In readFileMultiple specified data number not found in file.

- #define **ECMD_DBUF_FILE_OPERATION_FAIL** (0x01000000 | 0x2053)

  File operation failed.

- #define **ECMD_DBUF_NOT_OWNER** (0x01000000 | 0x2060)

  Don't own this buffer so can't do this operation.

- #define **ECMD_DBUF_XSTATE_NOT_ENABLED** (0x01000000 | 0x2062)

  Xstate function called on a buffer that doesn't have xstates enabled.

- #define **ETRAC0**(fmt) printf("%s> ETRC: " fmt "\n", __FUNCTION__);
- #define **ETRAC1**(fmt, arg1) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1);
- #define **ETRAC2**(fmt, arg1, arg2) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2);
- #define **ETRAC3**(fmt, arg1, arg2, arg3) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3);
- #define **ETRAC4**(fmt, arg1, arg2, arg3, arg4) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4);
- #define **ETRAC5**(fmt, arg1, arg2, arg3, arg4, arg5) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5);
- #define **ETRAC6**(fmt, arg1, arg2, arg3, arg4, arg5, arg6) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6);
- #define **ETRAC7**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7);
- #define **ETRAC8**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
- #define **ETRAC9**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);

Enumerations

- enum **ecmdFormatType_t** { **ECMD_SAVE_FORMAT_BINARY**, **ECMD_SAVE_FORMAT_BINARY_DATA**, **ECMD_SAVE_FORMAT_ASCII**, **ECMD_SAVE_FORMAT_XSTATE** }

  This is the different formats in which the output file will be written.

- enum **ecmdWriteMode_t** { **ECMD_WRITE_MODE**, **ECMD_APPEND_MODE** }

  This is the different write modes for writing databuffer into a file.

6.5.1 Detailed Description

Provides a means to handle data from the eCMD C API.

DataBuffers handle and store data in a Big Endian fashion with Bit 0 being the MSB

6.5.2 Define Documentation

6.5.2.1 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

6.5.2.2 #define ECMD_DBUF_INIT_FAIL (0x01000000 | 0x2000)

Initialization of the DataBuffer failed.

6.5.2.3 #define ECMD_DBUF_BUFFER_OVERFLOW (0x01000000 | 0x2010)

Attempt to read/write data beyond the length of the DataBuffer.

6.5.2.4 #define ECMD_DBUF_XSTATE_ERROR (0x01000000 | 0x2020)

An 'X' character occured where it was not expected.

6.5.2.5 #define ECMD_DBUF_UNDEFINED_FUNCTION (0x01000000 | 0x2030)

Function not included in this version of DataBuffer.

6.5.2.6 #define ECMD_DBUF_INVALID_ARGS (0x01000000 | 0x2040)

Args provided to dataBuffer were invalid.

6.5.2.7 #define ECMD_DBUF_INVALID_DATA_FORMAT (0x01000000 | 0x2041)

String data didn't match expected input format.

6.5.2.8 #define ECMD_DBUF_FOPEN_FAIL (0x01000000 | 0x2050)

File open on file for reading or writing the data buffer failed.

6.5.2.9 #define ECMD_DBUF_FILE_FORMAT_MISMATCH (0x01000000 | 0x2051)

In readFile specified format not found in the data file.

6.5.2.10 #define ECMD_DBUF_DATANUMBER_NOT_FOUND (0x01000000 | 0x2052)

In readFileMultiple specified data number not found in file.

6.5.2.11 #define ECMD_DBUF_FILE_OPERATION_FAIL (0x01000000 | 0x2053)

File operation failed.

6.5.2.12 #define ECMD_DBUF_NOT_OWNER (0x01000000 | 0x2060)

Don't own this buffer so can't do this operation.

6.5.2.13 #define ECMD_DBUF_XSTATE_NOT_ENABLED (0x01000000 | 0x2062)

Xstate function called on a buffer that doesn't have xstates enabled.

6.5.2.14 #define ETRAC0(fmt) printf("%s> ETRC: " fmt "\n", __FUNCTION__);

6.5.2.15 #define ETRAC1(fmt, arg1) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1);

6.5.2.16 #define ETRAC2(fmt, arg1, arg2) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2);

6.5.2.17 #define ETRAC3(fmt, arg1, arg2, arg3) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3);

6.5.2.18 #define ETRAC4(fmt, arg1, arg2, arg3, arg4) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4);

6.5.2.19 #define ETRAC5(fmt, arg1, arg2, arg3, arg4, arg5) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5);

6.5.2.20 #define ETRAC6(fmt, arg1, arg2, arg3, arg4, arg5, arg6) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6);

6.5.2.21 #define ETRAC7(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7);

6.5.2.22 #define ETRAC8(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);

6.5.2.23 #define ETRAC9(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);

6.5.3 Enumeration Type Documentation

6.5.3.1 enum ecmdFormatType_t

This is the different formats in which the output file will be written.

Enumeration values:

ECMD_SAVE_FORMAT_BINARY binary file with header with info like bit length, format etc

ECMD_SAVE_FORMAT_BINARY_DATA binary file with data only - will NOT work with scan ring data as length is rounded up to next byte

ECMD_SAVE_FORMAT_ASCII ascii text file with header having same info like binary hdr

ECMD_SAVE_FORMAT_XSTATE xstate text file with header having same info like binary hdr

6.5.3.2 enum ecmdWriteMode_t

This is the different write modes for writing databuffer into a file.

Enumeration values:

ECMD_WRITE_MODE Overrwrite the data if the file already exists.

ECMD_APPEND_MODE Add databuffer to the end of the file.

6.6 ecmdSharedUtils.H File Reference

Useful functions for use throughout the ecmd C API and Plugin.

```
#include <string>
#include <vector>
#include <inttypes.h>
#include <ecmdDataBuffer.H>
#include <ecmdStructs.H>
```

Gen Functions

- std::string **ecmdGenEbcdic** (**ecmdDataBuffer** &i_data, int start, int bitLen)

  Turns the data in the buffer into ebcdic text.

- uint32_t **ecmdGenB32FromHex** (uint32_t *o_numPtr, const char *i_hexChars, int startPos)

  Default function for converting hex strings to unsigned int arrays.

- uint32_t **ecmdGenB32FromHexLeft** (uint32_t *o_numPtr, const char *i_hexChars)

  Convert a string of left-aligned Hex chars into a left-aligned unsigned int array.

- uint32_t **ecmdGenB32FromHexRight** (uint32_t *o_numPtr, const char *i_hexChars, int i_expectBits=0)

  Convert a string of right-aligned Hex chars into a left-aligned unsigned int array.

Enumerations

- enum **ecmdTargetDepth_t** {
  ECMD_DEPTH_CAGE = 0, **ECMD_DEPTH_NODE**, **ECMD_DEPTH_SLOT**, **ECMD_DEPTH_CHIP**,
  ECMD_DEPTH_CORE, **ECMD_DEPTH_THREAD** }

  Used by ecmdSetTargetDepth.

- enum **ecmdTargetDisplayMode_t** { **ECMD_DISPLAY_TARGET_DEFAULT** }

  Used by ecmdWriteTarget to specify displayMode.

Functions

- void **ecmdParseTokens** (std::string line, const char *seperators, std::vector< std::string > &tokens)

  Breaks the string line into tokens based on all chars in seperators.

- uint32_t **ecmdHexToUInt32** (const char *i_str)

  Converts strings to unsigned int values. The input format is 0xABCDEF.

- uint32_t **ecmdHashString32** (const char *i_str, uint32_t i_c)

  Calculates a 32bit hash value for a given string.

- uint32_t **ecmdSetTargetDepth** (**ecmdChipTarget** &io_target, **ecmdTargetDepth_t** i_depth)

  Sets State Fields of Chip Target based on depth.

- uint32_t **ecmdReadDcard** (const char *i_filename, std::list< **ecmdMemoryEntry** > &o_data)

  Sets Fields of ecmdMemoryEntry_t from the DCard data in the file.

- std::string **ecmdWriteTarget** (**ecmdChipTarget** &i_target, **ecmdTargetDisplayMode_t** i_displayMode=ECMD_DISPLAY_TARGET_DEFAULT)

  Returns a formatted string containing the data in the given **ecmdChipTarget**(p. 20).

6.6.1 Detailed Description

Useful functions for use throughout the ecmd C API and Plugin.

6.6.2 Enumeration Type Documentation

6.6.2.1 enum ecmdTargetDepth_t

Used by ecmdSetTargetDepth.

Enumeration values:

ECMD_DEPTH_CAGE
ECMD_DEPTH_NODE
ECMD_DEPTH_SLOT
ECMD_DEPTH_CHIP
ECMD_DEPTH_CORE
ECMD_DEPTH_THREAD

6.6.2.2 enum ecmdTargetDisplayMode_t

Used by ecmdWriteTarget to specify displayMode.

Enumeration values:

ECMD_DISPLAY_TARGET_DEFAULT Default mode.

6.6.3 Function Documentation

6.6.3.1 void ecmdParseTokens (std::string *line*, const char * *seperators*, std::vector< std::string > & *tokens*)

Breaks the string line into tokens based on all chars in seperators.

Parameters:

line String to tokenize

seperators String of characters to use as seperators

tokens Vector of strings that contain all the tokens

6.6.3.2 std::string ecmdGenEbcdic (ecmdDataBuffer & *i_ data*, int *start*, int *bitLen*)

Turns the data in the buffer into ebcdic text.

Parameters:

i_ data Data to convert

start Bit to start at

bitLen Number of bits

6.6.3.3 uint32_t ecmdGenB32FromHex (uint32_t * *o_ numPtr*, const char * *i_ hexChars*, int *startPos*)

Default function for converting hex strings to unsigned int arrays.

Return values:

First element of the parsed data, or 0xFFFFFFFF if error

Parameters:

o_ numPtr The array that stores the data parsed from the input string

i_ hexChars input string of hex data- alignment stuff handled by Left and Right functions

startPos

See also:

ecmdGenB32FromHexRight(p. 211)
ecmdGenB32FromHexLeft(p. 210)

6.6.3.4 uint32_t ecmdGenB32FromHexLeft (uint32_t * *o_ numPtr*, const char * *i_ hexChars*)

Convert a string of left-aligned Hex chars into a left-aligned unsigned int array.

Return values:

The first element of the parsed string data, or 0xFFFFFFFF if error

Parameters:

o_ numPtr The array that stores the data parsed from the input string

i_ hexChars A string of hex characters

See also:

ecmdGenB32FromHexRight(p. 211)
ecmdGenB32FromHex(p. 210)

6.6.3.5 uint32_t ecmdGenB32FromHexRight (uint32_t * *o_numPtr*, const char * *i_hexChars*, int *i_expectBits* = 0)

Convert a string of right-aligned Hex chars into a left-aligned unsigned int array.

Return values:
The first element of the parsed string data, or 0xFFFFFFFF if error

Parameters:
o_numPtr The array that stores the data parsed from the input string

i_hexChars A string of hex characters

i_expectBits The number of defined bits in the o_numPtr array returned

See also:
ecmdGenB32FromHex(p. 210)
ecmdGenB32FromHexLeft(p. 210)

6.6.3.6 uint32_t ecmdHexToUInt32 (const char * *i_str*)

Converts strings to unsigned int values. The input format is 0xABCDEF.

Parameters:
i_str String in hexadecimal notation

Date:
Tue Sep 21 13:22:33 2004

Return values:
uint32_t value of converted input string

6.6.3.7 uint32_t ecmdHashString32 (const char * *i_str*, uint32_t *i_c*)

Calculates a 32bit hash value for a given string.

LICENSE: By Bob Jenkins, 1996. `bob_jenkins@burtleburtle.net`. You may use this code any way you wish, private, educational, or commercial. It's free. See `http://burtleburtle.net/bob/hash/doobs.html`

Parameters:
i_str String to convert to hash

i_c Start value for hash.

Return values:
Hash value

6.6.3.8 uint32_t ecmdSetTargetDepth (ecmdChipTarget & *io_target*, ecmdTargetDepth_t *i_depth*)

Sets State Fields of Chip Target based on depth.

Parameters:

io_target an **ecmdChipTarget**(p. 20) struct representing a specific eCmd target

i_depth an ecmdTargetDepth_t enum representing depth to be set valid

Return values:

ECMD_SUCCESS if setting successful

ECMD_INVALID_ARGS if unsuccessful in finding a matching depth

Postcondition:

State Fields of Chip Target are set to either ECMD_TARGET_FIELD VALID or ECMD_-TARGET_FIELD_UNUSED

TARGET DEPTH : Input To This Function TARGET STATES : Gets Set By This Function

6.6.3.9 uint32_t ecmdReadDcard (const char * *i_filename*, std::list< ecmdMemoryEntry > & *o_data*)

Sets Fields of ecmdMemoryEntry_t from the DCard data in the file.

Parameters:

i_filename file to read the d-card data from

o_data list to be updated with the d-card data

Return values:

ECMD_SUCCESS if setting successful

ECMD_INVALID_ARGS if unsuccessful in finding a matching depth

6.6.3.10 std::string ecmdWriteTarget (ecmdChipTarget & *i_target*, ecmdTargetDisplayMode_t *i_displayMode* = ECMD_DISPLAY_TARGET_DEFAULT)

Returns a formatted string containing the data in the given **ecmdChipTarget**(p. 20).

Returns:

String with formatted target data

Parameters:

i_target **ecmdChipTarget**(p. 20) containing data to format into string

i_displayMode Mode to format data

TARGET DEPTH : Thread TARGET STATES : Must be Initialized

6.7 ecmdStructs.H File Reference

All the Structures required for the eCMD Capi.

```
#include <inttypes.h>
#include <list>
#include <vector>
#include <string>
#include <ecmdDataBuffer.H>
```

Classes

- struct **ecmdDllInfo**

  This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with.

- struct **ecmdChipTarget**

  Structure used to designate which cec object/chip you would like the function to operate on.

- struct **ecmdThreadData**

  Used for the ecmdQueryConfig function to return thread data.

- struct **ecmdCoreData**

  Used for the ecmdQueryConfig function to return core data.

- struct **ecmdChipData**

  Used for the ecmdQueryConfig function to return chip data.

- struct **ecmdSlotData**

  Used for the ecmdQueryConfig function to return slot data.

- struct **ecmdNodeData**

  Used for the ecmdQueryConfig function to return node data.

- struct **ecmdCageData**

  Used for the ecmdQueryConfig function to return cage data.

- struct **ecmdQueryData**

  Used by the ecmdQueryConfig function to return data.

- struct **ecmdRingData**

  Used for the ecmdQueryRing function to return ring info.

- struct **ecmdArrayData**

  Used for the ecmdQueryArray function to return array info.

- struct **ecmdTraceArrayData**

  Used for the ecmdQueryTraceArray function to return trace array info.

- struct **ecmdScomData**

  Used for the ecmdQueryScom function to return scom info.

- struct **ecmdLatchData**

  Used for the ecmdQueryLatch function to return latch info.

- struct **ecmdArrayEntry**

  Used by the getArrayMultiple function to pass data.

- struct **ecmdSpyGroupData**

  Used by get/putspy function to create the return data from a group.

- struct **ecmdNameEntry**

  Used by get/putSprMultiple function to pass data.

- struct **ecmdNameVectorEntry**

  Used by getTraceArrayMultiple function to pass data.

- struct **ecmdIndexVectorEntry**

  Used by ???? function to pass data.

- struct **ecmdIndexEntry**

  Used by get/put Gpr/Fpr Multiple function to pass data.

- struct **ecmdLatchEntry**

  Used by getlatch function to return data.

- struct **ecmdProcRegisterInfo**

  Used by ecmdQueryProcRegisterInfo function to return data about a Architected register.

- struct **ecmdSpyData**

  Used for the ecmdQuerySpy function to return spy info.

- struct **ecmdMemoryEntry**

  Used by ecmdReadDcard.

- struct **ecmdSimModelInfo**

  Used by simGetModelInfo.

Defines

- #define **ECMD_CAPI_VERSION** "6.0"

  eCMD API Version

- #define **QD_HDR_MAGIC** 0xFFFFFFF1
- #define **CAGE_HDR_MAGIC** 0xFFFFFF2F
- #define **NODE_HDR_MAGIC** 0xFFFFF3FF
- #define **SLOT_HDR_MAGIC** 0xFFFF4FFF
- #define **CHIP_HDR_MAGIC** 0xFFF5FFFF

- #define **CORE_HDR_MAGIC** 0xFF6FFFFF
- #define **THREAD_HDR_MAGIC** 0xF7FFFFFF
- #define **ECMD_CHIPT_PROCESSOR** "pu"
- #define **ECMD_CHIPT_MEM_BUF** "memb"
- #define **ECMD_CHIPT_MEM_CNTRL** "memc"
- #define **ECMD_CHIPT_MEM_L2CACHE** "l2cache"
- #define **ECMD_CHIPT_MEM_L3CACHE** "l3cache"
- #define **ECMD_CHIPT_IOBDG** "iobdg"
- #define **ECMD_CHIPT_IOHUB** "iohub"
- #define **ECMD_CHIPT_MUX** "mux"
- #define **ECMD_CHIPT_SERVICE_PROCESSOR** "sp"
- #define **ECMD_CHIPFLAG_BUSMASK** 0xC0000000
- #define **ECMD_CHIPFLAG_RSVDBUS1** 0x00000000

  This is reserved for later expansion (should not be used).

- #define **ECMD_CHIPFLAG_JTAG** 0x40000000
- #define **ECMD_CHIPFLAG_FSI** 0x80000000
- #define **ECMD_CHIPFLAG_RSVDBUS2** 0xC0000000

  This is reserved for later expansion (should not be used).

Enumerations

- enum **ecmdCacheType_t** {

  ECMD_CACHE_UNKNOWN = 0, **ECMD_CACHE_LEVEL1D**, **ECMD_CACHE_LEVEL1I**, **ECMD_CACHE_LEVEL2**,

  ECMD_CACHE_LEVEL3, **ECMD_CACHE_LEVEL4** }

  Used by ecmdCacheFlush to specify which level of cache to flush.

- enum **ecmdChipInterfaceType_t** { **ECMD_INTERFACE_ACCESS**, **ECMD_INTERFACE_CFAM**, **ECMD_INTERFACE_UNKNOWN** }

  Used in **ecmdChipData**(p. 17) *to describe the interface macro used by the chip.*

- enum **ecmdChipTargetState_t** {

  ECMD_TARGET_UNKNOWN_STATE, **ECMD_TARGET_FIELD_VALID**, **ECMD_TARGET_FIELD_UNUSED**, **ECMD_TARGET_FIELD_WILDCARD**,

  ECMD_TARGET_THREAD_ALIVE }

  Used by **ecmdChipTarget**(p. 20) *to describe the value in the state fields.*

- enum **ecmdClockRange_t** {

  ECMD_CLOCK_RANGE_DEFAULT, **ECMD_CLOCK_RANGE_LOWEST**, **ECMD_CLOCK_RANGE_LOW**, **ECMD_CLOCK_RANGE_MIDDLE**,

  ECMD_CLOCK_RANGE_HIGH, **ECMD_CLOCK_RANGE_HIGHEST** }

  Used by SetClockSpeed interfaces to adjust clock steering procedure.

- enum **ecmdClockSetMode_t** { **ECMD_CLOCK_ONE_STEP**, **ECMD_CLOCK_STEER** }

Used by SetClockSpeed interfaces to specify to do adjustment in one operation or to steer to new value.

- enum **ecmdClockSpeedType_t** { **ECMD_CLOCK_FREQUENCY_SPEC**, **ECMD_CLOCK_CYCLETIME_SPEC** }

  Used by SetClockSpeed interfaces to specify what notation the speed is provided in.

- enum **ecmdClockState_t** { **ECMD_CLOCKSTATE_UNKNOWN**, **ECMD_CLOCKSTATE_ON**, **ECMD_CLOCKSTATE_OFF**, **ECMD_CLOCKSTATE_NA** }

  Used by Ring/Array/Spy Query functions to return a required clock state.

- enum **ecmdClockType_t** { **ECMD_PROC_REFCLOCK**, **ECMD_MEMCTRL_REFCLOCK** }

  Used by SetClockSpeed interfaces to specify the clock to control.

- enum **ecmdConfigLoopMode_t** { **ECMD_STATIC_DEPTH_LOOP** }

  Used by ecmdConfigLooperInit to enable/disable variable depth looping.

- enum **ecmdConfigLoopType_t** {

  ECMD_SELECTED_TARGETS_LOOP, **ECMD_SELECTED_TARGETS_LOOP_DEFALL**, **ECMD_SELECTED_TARGETS_LOOP_VD**, **ECMD_SELECTED_TARGETS_LOOP_VD_DEFALL**,

  ECMD_ALL_TARGETS_LOOP }

  Used by ecmdConfigLooperInit function to specify what type of data to loop on.

- enum **ecmdConfigValid_t** { **ECMD_CONFIG_VALID_FIELD_NONE**, **ECMD_CONFIG_VALID_FIELD_ALPHA**, **ECMD_CONFIG_VALID_FIELD_NUMERIC**, **ECMD_CONFIG_VALID_FIELD_BOTH** }

  Used by the get/set configuration functions to specify what data is good.

- enum **ecmdDioMode_t** { **ECMD_DIO_INPUT**, **ECMD_DIO_OPEN_DRAIN**, **ECMD_DIO_OPEN_SOURCE**, **ECMD_DIO_PUSH_PULL** }

  Used by the GPIO functions to specify the different modes for the GPIO pin.

- enum **ecmdDllEnv_t** { **ECMD_DLL_ENV_HW**, **ECMD_DLL_ENV_SIM** }

  This is used by ecmdQueryDllInfo to return what environment the dll is designed to run in (i.e Simulation vs Hardware).

- enum **ecmdDllProduct_t** { **ECMD_DLL_PRODUCT_UNKNOWN**, **ECMD_DLL_PRODUCT_ECLIPZ** }

  This is used by ecmdQueryDllInfo to return what product the dll supports.

- enum **ecmdDllType_t** {

  ECMD_DLL_UNKNOWN, **ECMD_DLL_STUB**, **ECMD_DLL_CRONUS**, **ECMD_DLL_IPSERIES**,

  ECMD_DLL_ZSERIES, **ECMD_DLL_SCAND** }

  This is used by ecmdQueryDllInfo to return who's dll you are actually running against.

- enum **ecmdFileType_t** {

  ECMD_FILE_SCANDEF, **ECMD_FILE_SPYDEF**, **ECMD_FILE_ARRAYDEF**, **ECMD_FILE_HELPTEXT**,

  ECMD_FILE_SCOMDATA, **ECMD_FILE_SPYDEFHASH**, **ECMD_FILE_SCANDEFHASH** }

  Used for the ecmdQueryFileLocation function to specify the file type you are looking for.

- enum **ecmdFusionMessageType_t** {

  ECMD_SIM_MSG_EXCEPTION = 1, **ECMD_SIM_MSG_TESTCASE** = 2, **ECMD_SIM_MSG_CMD_RS** = 4, **ECMD_SIM_MSG_CMD_EXE** = 8,

  ECMD_SIM_MSG_DEBUG = 16, **ECMD_SIM_MSG_BROADSIDE** = 32, **ECMD_SIM_MSG_END_SD_MSGS** = 0x000000ff, **ECMD_SIM_MSG_ALWAYS** = 0xff }

  Used by simOutputFusionMessage function to specify message type Values copied from globals.h of SimDispatcher delivery.

- enum **ecmdFusionSeverity_t** { **ECMD_SIM_ERROR** = 0x1, **ECMD_SIM_WARNING** = 0x2, **ECMD_SIM_INFO** = 0x3, **ECMD_SIM_PLAIN** = ~0 }

  Used by simOutputFusionMessage function to specify message severity Values copied from globals.h of SimDispatcher delivery.

- enum **ecmdGlobalVarType_t** { **ECMD_GLOBALVAR_DEBUG**, **ECMD_GLOBALVAR_QUIETMODE** }

  Used by ecmdGetGlobalVar to specify what variable you are looking for.

- enum **ecmdI2cBusSpeed_t** { **ECMD_I2C_BUSSPEED_50KHZ**, **ECMD_I2C_BUSSPEED_100KHZ**, **ECMD_I2C_BUSSPEED_400KHZ** }

  Used by I2C functions to specify bus speed.

- enum **ecmdLatchMode_t** { **ECMD_LATCHMODE_FULL**, **ECMD_LATCHMODE_PARTIAL** }

  Used by get/putLatch functions to specify what mode should be used to find latches in the scandef.

- enum **ecmdQueryDetail_t** { **ECMD_QUERY_DETAIL_LOW**, **ECMD_QUERY_DETAIL_HIGH** }

  Used by ecmdQueryConfig to specify detail level of query.

- enum **ecmdSpyType_t** { **ECMD_SPYTYPE_ALIAS**, **ECMD_SPYTYPE_IDIAL**, **ECMD_SPYTYPE_EDIAL**, **ECMD_SPYTYPE_ECCGROUP** }

  Used for the ecmdQuerySpy function to specify which type of spy we have.

- enum **ecmdTraceType_t** { **ECMD_TRACE_SCAN**, **ECMD_TRACE_PROCEDURE** }

  Used by ecmdSetTraceMode to specify which trace to control.

6.7.1 Detailed Description

All the Structures required for the eCMD Capi.

6.7.2 Define Documentation

6.7.2.1 #define ECMD_CAPI_VERSION "6.0"

eCMD API Version

6.7.2.2 #define QD_HDR_MAGIC 0xFFFFFFF1

6.7.2.3 #define CAGE_HDR_MAGIC 0xFFFFFF2F

6.7.2.4 #define NODE_HDR_MAGIC 0xFFFFF3FF

6.7.2.5 #define SLOT_HDR_MAGIC 0xFFFF4FFF

6.7.2.6 #define CHIP_HDR_MAGIC 0xFFF5FFFF

6.7.2.7 #define CORE_HDR_MAGIC 0xFF6FFFFF

6.7.2.8 #define THREAD_HDR_MAGIC 0xF7FFFFFF

6.7.2.9 #define ECMD_CHIPT_PROCESSOR "pu"

Predefined common chip names for **ecmdChipData.chipCommonType**(p. 18)

6.7.2.10 #define ECMD_CHIPT_MEM_BUF "memb"

6.7.2.11 #define ECMD_CHIPT_MEM_CNTRL "memc"

6.7.2.12 #define ECMD_CHIPT_MEM_L2CACHE "l2cache"

6.7.2.13 #define ECMD_CHIPT_MEM_L3CACHE "l3cache"

6.7.2.14 #define ECMD_CHIPT_IOBDG "iobdg"

6.7.2.15 #define ECMD_CHIPT_IOHUB "iohub"

6.7.2.16 #define ECMD_CHIPT_MUX "mux"

6.7.2.17 #define ECMD_CHIPT_SERVICE_PROCESSOR "sp"

6.7.2.18 #define ECMD_CHIPFLAG_BUSMASK 0xC0000000

Defines for the **ecmdChipData**(p. 17) chipFlags field

6.7.2.19 #define ECMD_CHIPFLAG_RSVDBUS1 0x00000000

This is reserved for later expansion (should not be used).

6.7.2.20 #define ECMD_CHIPFLAG_JTAG 0x40000000

6.7.2.21 #define ECMD_CHIPFLAG_FSI 0x80000000

6.7.2.22 #define ECMD_CHIPFLAG_RSVDBUS2 0xC0000000

This is reserved for later expansion (should not be used).

6.7.3 Enumeration Type Documentation

6.7.3.1 enum ecmdCacheType_t

Used by ecmdCacheFlush to specify which level of cache to flush.

Enumeration values:

ECMD_CACHE_UNKNOWN Unknown Cache Type.

ECMD_CACHE_LEVEL1D L1 Data Cache.

ECMD_CACHE_LEVEL1I L1 Instruction Cache.

ECMD_CACHE_LEVEL2 L2 Cache.

ECMD_CACHE_LEVEL3 L3 Cache.

ECMD_CACHE_LEVEL4 L4 Cache.

6.7.3.2 enum ecmdChipInterfaceType_t

Used in **ecmdChipData**(p. 17) to describe the interface macro used by the chip.

Enumeration values:

ECMD_INTERFACE_ACCESS Standard Jtag Access Macro.

ECMD_INTERFACE_CFAM CommonFirmwareAccessMacro.

ECMD_INTERFACE_UNKNOWN Unknown Interface.

6.7.3.3 enum ecmdChipTargetState_t

Used by **ecmdChipTarget**(p. 20) to describe the value in the state fields.

Enumeration values:

ECMD_TARGET_UNKNOWN_STATE State field has not been initialized.

ECMD_TARGET_FIELD_VALID Associated State Field is set to a valid value.

ECMD_TARGET_FIELD_UNUSED Associated State Field is unused and should be ignored.

ECMD_TARGET_FIELD_WILDCARD Associated State Field is a wildcard and should be iterated on in query functions.

ECMD_TARGET_THREAD_ALIVE Used when calling thread dependent functions tell the function to check for the thread to be alive before running.

6.7.3.4 enum ecmdClockRange_t

Used by SetClockSpeed interfaces to adjust clock steering procedure.

Enumeration values:

ECMD_CLOCK_RANGE_DEFAULT
ECMD_CLOCK_RANGE_LOWEST
ECMD_CLOCK_RANGE_LOW
ECMD_CLOCK_RANGE_MIDDLE
ECMD_CLOCK_RANGE_HIGH
ECMD_CLOCK_RANGE_HIGHEST

6.7.3.5 enum ecmdClockSetMode_t

Used by SetClockSpeed interfaces to specify to do adjustment in one operation or to steer to new value.

Enumeration values:

ECMD_CLOCK_ONE_STEP Change to new frequency in one operation.
ECMD_CLOCK_STEER Steer to new frequency.

6.7.3.6 enum ecmdClockSpeedType_t

Used by SetClockSpeed interfaces to specify what notation the speed is provided in.

Enumeration values:

ECMD_CLOCK_FREQUENCY_SPEC Clock speed is specified in Mhz.
ECMD_CLOCK_CYCLETIME_SPEC Clock speed is specified in cycle time.

6.7.3.7 enum ecmdClockState_t

Used by Ring/Array/Spy Query functions to return a required clock state.

Enumeration values:

ECMD_CLOCKSTATE_UNKNOWN Unable to determine a required clock state.
ECMD_CLOCKSTATE_ON Chip clocks must be on to access.
ECMD_CLOCKSTATE_OFF Chip clocks must be off to access.
ECMD_CLOCKSTATE_NA Chip clocks can be in any state to access.

6.7.3.8 enum ecmdClockType_t

Used by SetClockSpeed interfaces to specify the clock to control.

Enumeration values:

ECMD_PROC_REFCLOCK Processor reference clock.
ECMD_MEMCTRL_REFCLOCK Memory Controller reference clock.

6.7.3.9 enum ecmdConfigLoopMode_t

Used by ecmdConfigLooperInit to enable/disable variable depth looping.

Enumeration values:

ECMD_STATIC_DEPTH_LOOP Use the state specified on init, don't allow plugin to change.

6.7.3.10 enum ecmdConfigLoopType_t

Used by ecmdConfigLooperInit function to specify what type of data to loop on.

Enumeration values:

ECMD_SELECTED_TARGETS_LOOP Loop on only targets in the system the user specified with -p# -c# -n#, etc. if not specified default to 0.

ECMD_SELECTED_TARGETS_LOOP_DEFALL Loop on only targets in the system the user specified with -p# -c# -n#, etc. if not specified default to all.

ECMD_SELECTED_TARGETS_LOOP_VD Loop only on targets in the system to the depth user specified on command line (ie if user said only '-n0' then -s and below are unused) if not specified default to 0.

ECMD_SELECTED_TARGETS_LOOP_VD_DEFALL Loop only on targets in the system to the depth user specified on command line (ie if user said only '-n0' then -s and below are unused) if not specified default to all.

ECMD_ALL_TARGETS_LOOP Loop on all valid targets in the system.

6.7.3.11 enum ecmdConfigValid_t

Used by the get/set configuration functions to specify what data is good.

Enumeration values:

ECMD_CONFIG_VALID_FIELD_NONE No field is valid, must have been an error.

ECMD_CONFIG_VALID_FIELD_ALPHA The string field contains valid data.

ECMD_CONFIG_VALID_FIELD_NUMERIC The numeric field contains valid data.

ECMD_CONFIG_VALID_FIELD_BOTH Bothe the string and numeric fields contain valid data.

6.7.3.12 enum ecmdDioMode_t

Used by the GPIO functions to specify the different modes for the GPIO pin.

| High (1) | Low (0) | | Output Enable | Output Register | Output Enable | Output Register | Push Pull | true | 1 | true | 0 | Open Drain | false | 0 | true | 0 | Open Source | true | 1 | false | 1 |

Enumeration values:

ECMD_DIO_INPUT Input pin.

ECMD_DIO_OPEN_DRAIN See detailed table.
ECMD_DIO_OPEN_SOURCE See detailed table.
ECMD_DIO_PUSH_PULL See detailed table.

6.7.3.13 enum ecmdDllEnv_t

This is used by ecmdQueryDllInfo to return what environment the dll is designed to run in (i.e Simulation vs Hardware).

Enumeration values:
ECMD_DLL_ENV_HW Hardware Environment.
ECMD_DLL_ENV_SIM Simulation Environment.

6.7.3.14 enum ecmdDllProduct_t

This is used by ecmdQueryDllInfo to return what product the dll supports.

Enumeration values:
ECMD_DLL_PRODUCT_UNKNOWN Unknown product.
ECMD_DLL_PRODUCT_ECLIPZ Eclipz.

6.7.3.15 enum ecmdDllType_t

This is used by ecmdQueryDllInfo to return who's dll you are actually running against.

Enumeration values:
ECMD_DLL_UNKNOWN This should never be encountered.
ECMD_DLL_STUB This is a stub version of the dll for client testing.
ECMD_DLL_CRONUS Running against the Cronus Dll.
ECMD_DLL_IPSERIES Running against I/P Series HOM.
ECMD_DLL_ZSERIES Running against Z Series HOM.
ECMD_DLL_SCAND Running against the ScanD dll owned by Meghna Paruthi.

6.7.3.16 enum ecmdFileType_t

Used for the ecmdQueryFileLocation function to specify the file type you are looking for.

Enumeration values:
ECMD_FILE_SCANDEF Scandef file type.
ECMD_FILE_SPYDEF Spy Definition file.
ECMD_FILE_ARRAYDEF Array Definition file.
ECMD_FILE_HELPTEXT eCMD Help Text file - target field of ecmdQueryFileLocation is not used for this and just a path is returned
ECMD_FILE_SCOMDATA eCMD ScanComm Parse data files, used by getscom - target field of ecmdQueryFileLocation is not used for this and just a path is returned
ECMD_FILE_SPYDEFHASH Hash file for spy definition.
ECMD_FILE_SCANDEFHASH Hash file for the scandef.

6.7.3.17 enum ecmdFusionMessageType_t

Used by simOutputFusionMessage function to specify message type Values copied from globals.h of SimDispatcher delivery.

Enumeration values:

ECMD_SIM_MSG_EXCEPTION
ECMD_SIM_MSG_TESTCASE
ECMD_SIM_MSG_CMD_RS
ECMD_SIM_MSG_CMD_EXE
ECMD_SIM_MSG_DEBUG
ECMD_SIM_MSG_BROADSIDE
ECMD_SIM_MSG_END_SD_MSGS
ECMD_SIM_MSG_ALWAYS

6.7.3.18 enum ecmdFusionSeverity_t

Used by simOutputFusionMessage function to specify message severity Values copied from globals.h of SimDispatcher delivery.

Enumeration values:

ECMD_SIM_ERROR
ECMD_SIM_WARNING
ECMD_SIM_INFO
ECMD_SIM_PLAIN

6.7.3.19 enum ecmdGlobalVarType_t

Used by ecmdGetGlobalVar to specify what variable you are looking for.

Enumeration values:

ECMD_GLOBALVAR_DEBUG Retrieve the value of the ecmd debug flag set by ECMD_DEBUG env var.

ECMD_GLOBALVAR_QUIETMODE Retrieve the value of the quiet mode debug flag = set by -quiet default = 0.

6.7.3.20 enum ecmdI2cBusSpeed_t

Used by I2C functions to specify bus speed.

Enumeration values:

ECMD_I2C_BUSSPEED_50KHZ Run I2c bus at 50Khz.

ECMD_I2C_BUSSPEED_100KHZ Run I2c bus at 100Khz.

ECMD_I2C_BUSSPEED_400KHZ Run I2c bus at 400Khz.

6.7.3.21 enum ecmdLatchMode_t

Used by get/putLatch functions to specify what mode should be used to find latches in the scandef.

Enumeration values:

ECMD_LATCHMODE_FULL Latch must match exactly.

ECMD_LATCHMODE_PARTIAL Latch can be a partial match.

6.7.3.22 enum ecmdQueryDetail_t

Used by ecmdQueryConfig to specify detail level of query.

Enumeration values:

ECMD_QUERY_DETAIL_LOW Only config info is returned.

ECMD_QUERY_DETAIL_HIGH All info is returned.

6.7.3.23 enum ecmdSpyType_t

Used for the ecmdQuerySpy function to specify which type of spy we have.

See also:

ecmdSpyData(p. 92)

Enumeration values:

ECMD_SPYTYPE_ALIAS Spy is an alias.

ECMD_SPYTYPE_IDIAL Spy is an iDial.

ECMD_SPYTYPE_EDIAL Spy is an eDial.

ECMD_SPYTYPE_ECCGROUP Spy is an eccGrouping.

6.7.3.24 enum ecmdTraceType_t

Used by ecmdSetTraceMode to specify which trace to control.

Enumeration values:

ECMD_TRACE_SCAN Scan Trace.

ECMD_TRACE_PROCEDURE Procedure Trace.

6.7.4 Function Documentation

6.7.4.1 std::string ecmdGetSharedLibVersion ()

Returns the version of the shared lib so it can be compared with the other versions.

6.8 ecmdUtils.H File Reference

Useful functions for use throughout the ecmd C API.

```
#include <inttypes.h>
#include <string>
#include <vector>
#include <ecmdClientCapi.H>
```

Classes

- struct **ecmdLooperData**

  Used internally by ecmdConfigLooper to store looping state information.

eCMD Utility Functions

- uint32_t **ecmdConfigLooperInit** (**ecmdChipTarget** &io_target, **ecmdConfigLoopType_t** i_looptype, **ecmdLooperData** &io_state, **ecmdConfigLoopMode_t** i_mode=ECMD_STATIC_DEPTH_LOOP)

  Initializes data structures and code to loop over configured and selected elements of the system.

- uint32_t **ecmdConfigLooperNext** (**ecmdChipTarget** &io_target, **ecmdLooperData** &io_state)

  Loops over configured and selected elements of the system, updating target to point to them.

- uint32_t **ecmdReadDataFormatted** (**ecmdDataBuffer** &o_data, const char *i_dataStr, std::string i_format, int i_expectedLength=0)

  Reads data from data string into data buffer based on a format type.

- uint32_t **decToUInt32** (const char *i_decstr)

  Converts decimal string to uint32_t.

- std::string **ecmdWriteDataFormatted** (**ecmdDataBuffer** &i_data, std::string i_format, uint64_t i_address=0)

  Formats data from data buffer into a string according to format flag and returns the string.

- std::string **ecmdBitsHeader** (int i_initCharOffset, int i_blockSize, int i_numCols, int i_maxBitWidth)

  Print the bits header used in the output formats.

- uint32_t **ecmdGetChipData** (**ecmdChipTarget** &i_target, **ecmdChipData** &o_data)

  Fetch the detailed chip data structure for the selected target.

- uint32_t **ecmdDisplayDllInfo** ()

  Function calls ecmdQueryDllInfo and displays the output to stdout.

- uint32_t **ecmdDisplayScomData** (**ecmdChipTarget** &i_target, uint32_t i_address, **ecmdDataBuffer** &i_data, const char *i_format)

  Display the Scom data with the bit descriptions.

6.8.1 Detailed Description

Useful functions for use throughout the ecmd C API.

6.8.2 Function Documentation

6.8.2.1 uint32_t ecmdConfigLooperInit (ecmdChipTarget & *io_target*, ecmdConfigLoopType_t *i_looptype*, ecmdLooperData & *io_state*, ecmdConfigLoopMode_t *i_mode* = ECMD_STATIC_DEPTH_LOOP)

Initializes data structures and code to loop over configured and selected elements of the system.

Parameters:

io_target Initial **ecmdChipTarget**(p. 20) that may contain information used in building the struct to loop over

i_looptype Specify type of all, all chips in system or all chips selected by user

i_mode Specify if you want to allow the plugin to change the depth of your loop or not

io_state Used internally by ConfigLooper to keep track of state, unique instance must be passed into each loop and must be passed to ecmdConfigLooperNext

Return values:

ECMD_SUCCESS if initialization succeeded, error code if otherwise

See also:

ecmdConfigLooperNext(p. 226)

TARGET DEPTH : Thread TARGET STATES : Must Be Initialized

6.8.2.2 uint32_t ecmdConfigLooperNext (ecmdChipTarget & *io_target*, ecmdLooperData & *io_state*)

Loops over configured and selected elements of the system, updating target to point to them.

Parameters:

io_target **ecmdChipTarget**(p. 20) that contains info about next target to process

io_state Used internally to keep track of state, must be passed from output of ecmdConfigLooperInit

Return values:

1 if io_target is valid, 0 if it is not

See also:

ecmdConfigLooperInit(p. 226)

TARGET DEPTH : Thread TARGET STATES : Must be Initialized (from ecmdConfigLooperInit)

6.8.2.3 uint32_t ecmdReadDataFormatted (ecmdDataBuffer & *o_ data*, const char * *i_ dataStr*, std::string *i_ format*, int *i_ expectedLength* = 0)

Reads data from data string into data buffer based on a format type.

Return values:
ECMD_SUCCESS if data is well-formatted, non-zero otherwise

Parameters:
o_ data **ecmdDataBuffer**(p. 24) where data from data string is placed.

i_ dataStr string of characters containing data

i_ format Flag that tells how to parse the data string, e.g., "b" = binary, "x" = hex left

i_ expectedLength If length of data is known before hand , should be passed is necessary for right aligned data that is not byte aligned lengths

6.8.2.4 uint32_t decToUInt32 (const char * *i_ decstr*)

Converts decimal string to uint32_t.

Return values:
uint32_t value of converted input string

Parameters:
i_ decstr string of characters containing data

6.8.2.5 std::string ecmdWriteDataFormatted (ecmdDataBuffer & *i_ data*, std::string *i_ format*, uint64_t *i_ address* = 0)

Formats data from data buffer into a string according to format flag and returns the string.

Returns:
String of formatted data

Parameters:
i_ data **ecmdDataBuffer**(p. 24) where data to format is stored

i_ format Flag that tells how to parse the data into a string, e.g., "b" = binary, "x" = hex left

i_ address A base address value that can be used in formating certain data- i.e., data from memory

6.8.2.6 std::string ecmdBitsHeader (int *i_ initCharOffset*, int *i_ blockSize*, int *i_ numCols*, int *i_ maxBitWidth*)

Print the bits header used in the output formats.

Parameters:
i_ initCharOffset char offset on screen to start printing

i_ blockSize Binary block size (ie. column char size)

i_ numCols Number of columns to display

i_ maxBitWidth Maximum number of bits to display - this is actual data valid so we don't display more columns then we need

Returns:
String of formatted data

6.8.2.7 uint32_t ecmdGetChipData (ecmdChipTarget & *i_ target*, ecmdChipData & *o_ data*)

Fetch the detailed chip data structure for the selected target.

Return values:
ECMD_ SUCCESS if chip data for target is found, non-zero otherwise

Parameters:
i_ target **ecmdChipTarget**(p. 20) that information is requested for

o_ data **ecmdChipData**(p. 17) struct that contains detailed info on chip ec level, etc.

TARGET DEPTH : Pos TARGET STATES : Unused

6.8.2.8 uint32_t ecmdDisplayDllInfo ()

Function calls ecmdQueryDllInfo and displays the output to stdout.

Return values:
ECMD_ SUCCESS if successful

nonzero on failure

6.8.2.9 uint32_t ecmdDisplayScomData (ecmdChipTarget & *i_ target*, uint32_t *i_ address*, ecmdDataBuffer & *i_ data*, const char * *i_ format*)

Display the Scom data with the bit descriptions.

Return values:
ECMD_ SUCCESS if scom lookup and display was successful, non-zero otherwise

Parameters:
i_ target target for which scom data needs to be displayed.

i_ address Scom Address for which the details are required

i_ data buffer that holds the scom data

i_ format posible values -v, -vs0, -vs1 for the information that needs to be displayed

6.9 gipClientPerlapi.H File Reference

IP eCMD Perlapi Extension.

```
#include <ecmdReturnCodes.H>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <gipStructs.H>
```

Load/Unload Functions

- int **gipInitExtension** (const char *i_clientVersion)

  Initialize eCMD GIP Extension DLL.

Chic Control Functions

- uint32_t **gipReserve** (chicReserveId i_reserveId, uint32_t i_waitTime)

  Requests that the cecserver's serial thread only run commands that are sent with the specified reserve id. (From chicclientlib.H/C).

- uint32_t **gipRelease** (chicReserveId i_reserveId)

  Release an active reserve. (From chicclientlib.H/C).

- uint32_t **gipAbort** (chicReserveId i_reserveId)

  Abort the currently executing serial command and any pending serial commands. (From chicclientlib.H/C).

- uint32_t **gipSetServAddr** (const char *i_servAddr, uint32_t nets_port)

  Set the the Client's IP Address and the port # for the server its connecting to.

Memory Functions

- uint32_t **gipGetMemProcVariousAddrType** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** i_address, uint32_t i_bytes, gipXlateVariables i_xlateVars, **ecmdDataBuffer** &o_memoryData, **ecmdDataBuffer** &o_memoryTags, **ecmdDataBuffer** &o_memoryEcc, **ecmdDataBuffer** &o_memoryEccError, **ecmdDataBuffer** &o_realAddress)

  Reads System Mainstore through the processor chip using an effective address.

- uint32_t **gipPutMemProcVariousAddrType** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** i_address, uint32_t i_bytes, gipXlateVariables i_xlateVars, **ecmdDataBuffer** &i_memoryData, **ecmdDataBuffer** &io_memoryTags, **ecmdDataBuffer** &o_realAddress)

  Writes System Mainstore through the processor chip using an effective address.

Breakpoint Functions

- uint32_t **gipSetSoftwareBreakpoint** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** i_address, gipXlateVariables i_xlateVars, **ecmdDataBuffer** &o_breakpointTable, **ecmdDataBuffer** &o_virtualAddress)

  Set a software breakpoint in Processor.

- uint32_t **gipClearSoftwareBreakpoint** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** i_address, gipXlateVariables i_xlateVars, **ecmdDataBuffer** &o_breakpointTable, **ecmdDataBuffer** &o_virtualAddress)

  Clear a software breakpoint in Processor.

- uint32_t **gipGetSoftwareBreakpoint** (**ecmdChipTarget** &i_target, **ecmdDataBuffer** i_address, gipXlateVariables i_xlateVars, **ecmdDataBuffer** &o_breakpointTable, **ecmdDataBuffer** &o_virtualAddress)

  Get software breakpoint table.

System Info Function

- uint32_t **gipGetSystemInfo** (cpcfSysInfo_t &o_systemInfo)

  Get the System Info structure.

Misc/Test Functions

- uint32_t **gipConnectionTest** (uint32_t i_options)

  Used to interface with connection test function.

6.9.1 Detailed Description

IP eCMD Perlapi Extension.

Extension Owner : Chris Engel

6.9.2 Function Documentation

6.9.2.1 int gipInitExtension (const char * *i_clientVersion*)

Initialize eCMD GIP Extension DLL.

Parameters:

i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:
eCMD GIP Extension is initialized and version checked

VERSIONS : eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE : if (gipInitExtension("ver3,ver4")) { die "Fatal errors initializing DLL"; }

6.9.2.2 uint32_t gipReserve (chicReserveId *i_reserveId*, uint32_t *i_waitTime*)

Requests that the cecserver's serial thread only run commands that are sent with the specified reserve id. (From chicclientlib.H/C).

Parameters:
i_reserveId The id that should be associated with the reserve if it is granted.
i_waitTime The number of milliseconds to wait for the reserve to be granted. If this is set to zero, it will wait without timing out.

Return values:
ECMD_SUCCESS if successful read
nonzero if unsuccessful

NOTE : See chicclientlib.C for more details

6.9.2.3 uint32_t gipRelease (chicReserveId *i_reserveId*)

Release an active reserve. (From chicclientlib.H/C).

Parameters:
i_reserveId The id of an active reserve which is to be released. Possible values can be found in chiccommon.H.

Returns:
Upon success, a NULL pointer will be returned. Otherwise, a pointer to an ErrlEntry with the return code field set to one of the values below will be returned.

Return values:
ECMD_SUCCESS if successful read
nonzero if unsuccessful

NOTE : See chicclientlib.C for more details

6.9.2.4 uint32_t gipAbort (chicReserveId *i_reserveId*)

Abort the currently executing serial command and any pending serial commands. (From chicclientlib.H/C).

Parameters:

i_reserveId The reserve Id that will be made active just before chicAbort completes.

Return values:

ECMD_SUCCESS if successful read

nonzero if unsuccessful NOTE : See chicclientlib.C for more details

6.9.2.5 uint32_t gipSetServAddr (const char * *i_servAddr*, uint32_t *nets_port*)

Set the the Client's IP Address and the port # for the server its connecting to.

Parameters:

i_servAddr constant string for the IP Address to be used

nets_port [optional] port number to be used on the server

Return values:

ECMD_SUCCESS if successful read

nonzero if unsuccessful NOTE : See chicclientlib.C for more details

6.9.2.6 uint32_t gipGetMemProcVariousAddrType (ecmdChipTarget & *i_target*, ecmdDataBuffer *i_address*, uint32_t *i_bytes*, gipXlateVariables *i_xlateVars*, ecmdDataBuffer & *o_memoryData*, ecmdDataBuffer & *o_memoryTags*, ecmdDataBuffer & *o_memoryEcc*, ecmdDataBuffer & *o_memoryEccError*, ecmdDataBuffer & *o_realAddress*)

Reads System Mainstore through the processor chip using an effective address.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_INVALID_MEMORY_ADDRESS Memory Address was not on a 8-byte boundary

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address Starting address to read from

i_bytes Number of bytes to write

i_xlateVars Scruct with numerous translation variables

o_memoryData DataBuffer object that holds data read from memory

o_memoryTags 1 Tag bit for every 64 bits of memory data

o_ memoryEcc 8 ECC bits for every 64 bits of memory data

o_ memoryEccError 1 ECC Error bit for every 64 bits of memory data

o_ realAddress Caculated Real Address

NOTE : This function requires that the address be aligned on an 8-byte boundary

TARGET DEPTH : Thread

TARGET STATES : Unused

6.9.2.7 uint32_t gipPutMemProcVariousAddrType (ecmdChipTarget & *i_ target*, ecmdDataBuffer *i_ address*, uint32_t *i_ bytes*, gipXlateVariables *i_ xlateVars*, ecmdDataBuffer & *i_ memoryData*, ecmdDataBuffer & *io_ memoryTags*, ecmdDataBuffer & *o_ realAddress*)

Writes System Mainstore through the processor chip using an effective address.

Return values:

ECMD_ TARGET_ INVALID_ TYPE if target is not a processor

ECMD_ TARGET_ NOT_ CONFIGURED if target is not available in the system

ECMD_ RING_ CACHE_ ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_ CLOCKS_ IN_ INVALID_ STATE Chip Clocks were in an invalid state to perform the operation

ECMD_ INVALID_ MEMORY_ ADDRESS Memory Address was not on a 8-byte boundary

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_ target Struct that contains chip and cage/node/slot/position information

i_ address Starting address to read from

i_ bytes Number of bytes to write

i_ xlateVars Scruct with numerous translation variables

i_ memoryData DataBuffer object that holds data read from memory

io_ memoryTags 1 Tag bit for every 64 bits of memory data (If this has length of zero, the user wants the HW to generate this info; otherwise, use their their values.)

o_ realAddress Caculated Real Address

NOTE : This function requires that the address be aligned on an 8-byte boundary

TARGET DEPTH : Thread

TARGET STATES : Unused

6.9.2.8 uint32_t gipSetSoftwareBreakpoint (ecmdChipTarget & *i_ target*, ecmdDataBuffer *i_ address*, gipXlateVariables *i_ xlateVars*, ecmdDataBuffer & *o_ breakpointTable*, ecmdDataBuffer & *o_ virtualAddress*)

Set a software breakpoint in Processor.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_address Address to set breakpoint at

i_xlateVars Scruct with numerous translation variables

o_breakpointTable DataBuffer object that holds breakpoint table

o_virtualAddress Caculated Virtual Address

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RETRY_WITH_VIRTUAL_ADDR Requests that the user retries operation with returned Virtual Address

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Thread

TARGET STATES : Unused

6.9.2.9 uint32_t gipClearSoftwareBreakpoint (ecmdChipTarget & *i_target*, ecmdDataBuffer *i_address*, gipXlateVariables *i_xlateVars*, ecmdDataBuffer & *o_breakpointTable*, ecmdDataBuffer & *o_virtualAddress*)

Clear a software breakpoint in Processor.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_address Address to clear breakpoint at

i_xlateVars Scruct with numerous translation variables

o_breakpointTable DataBuffer object that holds breakpoint table

o_virtualAddress Caculated Virtual Address

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RETRY_WITH_VIRTUAL_ADDR Requests that the user retries operation with returned Virtual Address

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Thread

TARGET STATES : Unused

6.9.2.10 uint32_t gipGetSoftwareBreakpoint (ecmdChipTarget & *i_target*, ecmdDataBuffer *i_address*, gipXlateVariables *i_xlateVars*, ecmdDataBuffer & *o_breakpointTable*, ecmdDataBuffer & *o_virtualAddress*)

Get software breakpoint table.

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core/thread information

i_address Address to get breakpoint at

i_xlateVars Scruct with numerous translation variables

o_breakpointTable DataBuffer object that holds breakpoint table

o_virtualAddress Caculated Virtual Address

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RETRY_WITH_VIRTUAL_ADDR Requests that the user retries operation with returned Virtual Address

ECMD_SUCCESS if successful

nonzero if unsuccessful

TARGET DEPTH : Thread

TARGET STATES : Unused

6.9.2.11 uint32_t gipGetSystemInfo (cpcfSysInfo_t & *o_systemInfo*)

Get the System Info structure.

Parameters:

o_systemInfo Buffer containing the SystemInfo struct

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

6.9.2.12 uint32_t gipConnectionTest (uint32_t *i_options*)

Used to interface with connection test function.

Parameters:

i_options Options for the call

Return values:

ECMD_SUCCESS if successful read

nonzero if unsuccessful NOTE : See chicclientlib.C for more details

6.10 zseClientPerlapi.H File Reference

Z Series Perlapi Extension.

```
#include <ecmdReturnCodes.H>
#include <ecmdStructs.H>
#include <ecmdDataBuffer.H>
#include <zseStructs.H>
```

Load/Unload Functions

- int **zseInitExtension** (const char *i_clientVersion)
  Initialize eCMD GIP Extension DLL.

6.10.1 Detailed Description

Z Series Perlapi Extension.

Extension Owner : Han-Joachim Hartmann

6.10.2 Function Documentation

6.10.2.1 int zseInitExtension (const char * *i_clientVersion*)

Initialize eCMD GIP Extension DLL.

Parameters:

i_clientVersion Comma seperated list of eCMD Perl api major numbers this script supports, see details

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:

eCMD GIP Extension is initialized and version checked

VERSIONS : eCMD at times has to make changes to add/remove functionality and parameters to functions. This could cause incompatability in your script if you used functions that have changed. The i_clientVersion string is used to tell eCMD which major releases you support such that your script will not continue execution if it encounters a version that is either not known about or not supported. This is similar to how the eCMD C-Api works except in Perl you can support multiple versions with one script as long as the changes that were made between the versions do not affect your script.

USAGE : if (zseInitExtension("ver3,ver4")) { die "Fatal errors initializing DLL"; }

Index

~ecmdDataBuffer
    ecmdDataBuffer, 33
~ecmdOptimizableDataBuffer
    ecmdOptimizableDataBuffer, 84

address
    ecmdArrayEntry, 15
    ecmdMemoryEntry, 80
    ecmdRingData, 87
    ecmdScomData, 89
applyInversionMask
    ecmdDataBuffer, 44
applyRawBufferToXstate
    ecmdDataBufferImplementationHelper, 68
arrayName
    ecmdArrayData, 14

bitLength
    ecmdLatchData, 73
    ecmdProcRegisterInfo, 85
    ecmdRingData, 87
    ecmdSpyData, 92
buffer
    ecmdArrayEntry, 15
    ecmdIndexEntry, 71
    ecmdIndexVectorEntry, 72
    ecmdLatchEntry, 75
    ecmdNameEntry, 81
    ecmdNameVectorEntry, 82

cage
    ecmdChipTarget, 21
CAGE_HDR_MAGIC
    ecmdStructs.H, 218
cageData
    ecmdQueryData, 86
cageId
    ecmdCageData, 16
cageState
    ecmdChipTarget, 22
CHIP_HDR_MAGIC
    ecmdStructs.H, 218
chipCommonType
    ecmdChipData, 18
chipData
    ecmdSlotData, 91
chipEc
    ecmdChipData, 18
chipFlags
    ecmdChipData, 18
chipShortType
    ecmdChipData, 18
chipType
    ecmdChipData, 18
    ecmdChipTarget, 21
chipTypeState
    ecmdChipTarget, 22
cipClearBreakpoint
    cipClientPerlapi.H, 104
cipClientPerlapi.H, 99
cipClientPerlapi.H
    cipClearBreakpoint, 104
    cipGetMemMemCtrl, 109
    cipGetMemProc, 107
    cipGetVr, 105
    cipGetVrMultiple, 105
    cipInitExtension, 101
    cipPutMemMemCtrl, 109
    cipPutMemProc, 108
    cipPutVr, 106
    cipPutVrMultiple, 107
    cipSetBreakpoint, 104
    cipStartAllInstructions, 102
    cipStartAllInstructionsSreset, 102
    cipStartInstructions, 101
    cipStartInstructionsSreset, 102
    cipStepInstructions, 103
    cipStopAllInstructions, 103
    cipStopInstructions, 103
cipGetMemMemCtrl
    cipClientPerlapi.H, 109
cipGetMemProc
    cipClientPerlapi.H, 107
cipGetVr
    cipClientPerlapi.H, 105
cipGetVrMultiple
    cipClientPerlapi.H, 105
cipInitExtension
    cipClientPerlapi.H, 101

cipPutMemMemCtrl
ecmdDataBuffer, 49
cipClientPerlapi.H, 109
cipPutMemProc
cipClientPerlapi.H, 108
cipPutVr
cipClientPerlapi.H, 106
cipPutVrMultiple
cipClientPerlapi.H, 107
cipSetBreakpoint
cipClientPerlapi.H, 104
cipStartAllInstructions
cipClientPerlapi.H, 102
cipStartAllInstructionsSreset
cipClientPerlapi.H, 102
cipStartInstructions
cipClientPerlapi.H, 101
cipStartInstructionsSreset
cipClientPerlapi.H, 102
cipStepInstructions
cipClientPerlapi.H, 103
cipStopAllInstructions
cipClientPerlapi.H, 103
cipStopInstructions
cipClientPerlapi.H, 103
clear
ecmdDataBuffer, 33
clearBit
ecmdDataBuffer, 39
clockDomain
ecmdArrayData, 14
ecmdLatchData, 73
ecmdRingData, 88
ecmdScomData, 89
ecmdSpyData, 93
ecmdTraceArrayData, 96
clockState
ecmdArrayData, 14
ecmdLatchData, 74
ecmdRingData, 88
ecmdScomData, 89
ecmdSpyData, 93
ecmdTraceArrayData, 97
cmdClientPerlapi.H, 111
cmdClientPerlapi.H
cmdInitExtension, 111
cmdRunCommand, 112
cmdRunCommandCaptureOutput, 112
cmdInitExtension
cmdClientPerlapi.H, 111
cmdRunCommand
cmdClientPerlapi.H, 112
cmdRunCommandCaptureOutput
cmdClientPerlapi.H, 112
concat
ecmdDataBuffer, 49
copy
ecmdDataBuffer, 53
core
ecmdChipTarget, 21
CORE_HDR_MAGIC
ecmdStructs.H, 218
coreData
ecmdChipData, 18
coreId
ecmdCoreData, 23
coreState
ecmdChipTarget, 22
croClearDebug
croClientPerlapi.H, 116
croClientPerlapi.H, 113
croClientPerlapi.H
croClearDebug, 116
croDisplayVersion, 115
croFastLoadL2, 119
croFastLoadL2RingImage, 119
croGetConfiguration, 118
croGetL2, 120
croGetL2Data, 120
croGetL2Dir, 121
croGetL2Tag, 121
croInitExtension, 115
croIsDebugOn, 116
croJtagScanRead, 117
croJtagScanReadWrite, 117
croJtagScanWrite, 117
croPutL2, 121
croPutL2Data, 122
croPutL2Dir, 122
croPutL2Tag, 122
croRamInstruction, 120
croReset, 115
croSetConfiguration, 118
croSetDebug, 116
croDisplayVersion
croClientPerlapi.H, 115
croFastLoadL2
croClientPerlapi.H, 119
croFastLoadL2RingImage
croClientPerlapi.H, 119
croGetConfiguration
croClientPerlapi.H, 118
croGetL2
croClientPerlapi.H, 120
croGetL2Data
croClientPerlapi.H, 120
croGetL2Dir
croClientPerlapi.H, 121
croGetL2Tag

croClientPerlapi.H, 121
croInitExtension
croClientPerlapi.H, 115
croIsDebugOn
croClientPerlapi.H, 116
croJtagScanRead
croClientPerlapi.H, 117
croJtagScanReadWrite
croClientPerlapi.H, 117
croJtagScanWrite
croClientPerlapi.H, 117
croPutL2
croClientPerlapi.H, 121
croPutL2Data
croClientPerlapi.H, 122
croPutL2Dir
croClientPerlapi.H, 122
croPutL2Tag
croClientPerlapi.H, 122
croRamInstruction
croClientPerlapi.H, 120
croReset
croClientPerlapi.H, 115
croSetConfiguration
croClientPerlapi.H, 118
croSetDebug
croClientPerlapi.H, 116
curUnitIdTarget
ecmdLooperData, 79

data
ecmdMemoryEntry, 80
deadbitsMask
ecmdSpyGroupData, 94
decToUInt32
ecmdUtils.H, 227
detailLevel
ecmdQueryData, 86
disableXstateBuffer
ecmdDataBuffer, 60
dllBuildDate
ecmdDllInfo, 70
dllBuildInfo
ecmdDllInfo, 70
dllCapiVersion
ecmdDllInfo, 70
dllEnv
ecmdDllInfo, 69
dllProduct
ecmdDllInfo, 69
dllProductType
ecmdDllInfo, 69
dllType
ecmdDllInfo, 69

ECMD_ALL_TARGETS_LOOP
ecmdStructs.H, 221
ECMD_APPEND_MODE
ecmdDataBuffer.H, 207
ECMD_CACHE_LEVEL1D
ecmdStructs.H, 219
ECMD_CACHE_LEVEL1I
ecmdStructs.H, 219
ECMD_CACHE_LEVEL2
ecmdStructs.H, 219
ECMD_CACHE_LEVEL3
ecmdStructs.H, 219
ECMD_CACHE_LEVEL4
ecmdStructs.H, 219
ECMD_CACHE_UNKNOWN
ecmdStructs.H, 219
ECMD_CAPI_VERSION
ecmdStructs.H, 218
ECMD_CHIPFLAG_BUSMASK
ecmdStructs.H, 218
ECMD_CHIPFLAG_FSI
ecmdStructs.H, 219
ECMD_CHIPFLAG_JTAG
ecmdStructs.H, 218
ECMD_CHIPFLAG_RSVDBUS1
ecmdStructs.H, 218
ECMD_CHIPFLAG_RSVDBUS2
ecmdStructs.H, 219
ECMD_CHIPT_IOBDG
ecmdStructs.H, 218
ECMD_CHIPT_IOHUB
ecmdStructs.H, 218
ECMD_CHIPT_MEM_BUF
ecmdStructs.H, 218
ECMD_CHIPT_MEM_CNTRL
ecmdStructs.H, 218
ECMD_CHIPT_MEM_L2CACHE
ecmdStructs.H, 218
ECMD_CHIPT_MEM_L3CACHE
ecmdStructs.H, 218
ECMD_CHIPT_MUX
ecmdStructs.H, 218
ECMD_CHIPT_PROCESSOR
ecmdStructs.H, 218
ECMD_CHIPT_SERVICE_PROCESSOR
ecmdStructs.H, 218
ECMD_CLOCK_CYCLETIME_SPEC
ecmdStructs.H, 220
ECMD_CLOCK_FREQUENCY_SPEC
ecmdStructs.H, 220
ECMD_CLOCK_ONE_STEP
ecmdStructs.H, 220
ECMD_CLOCK_RANGE_DEFAULT
ecmdStructs.H, 220

ECMD_CLOCK_RANGE_HIGH
ecmdStructs.H, 220
ECMD_CLOCK_RANGE_HIGHEST
ecmdStructs.H, 220
ECMD_CLOCK_RANGE_LOW
ecmdStructs.H, 220
ECMD_CLOCK_RANGE_LOWEST
ecmdStructs.H, 220
ECMD_CLOCK_RANGE_MIDDLE
ecmdStructs.H, 220
ECMD_CLOCK_STEER
ecmdStructs.H, 220
ECMD_CLOCKSTATE_NA
ecmdStructs.H, 220
ECMD_CLOCKSTATE_OFF
ecmdStructs.H, 220
ECMD_CLOCKSTATE_ON
ecmdStructs.H, 220
ECMD_CLOCKSTATE_UNKNOWN
ecmdStructs.H, 220
ECMD_CONFIG_VALID_FIELD_-
ALPHA
ecmdStructs.H, 221
ECMD_CONFIG_VALID_FIELD_BOTH
ecmdStructs.H, 221
ECMD_CONFIG_VALID_FIELD_NONE
ecmdStructs.H, 221
ECMD_CONFIG_VALID_FIELD_-
NUMERIC
ecmdStructs.H, 221
ECMD_DBUF_BUFFER_OVERFLOW
ecmdDataBuffer.H, 204
ECMD_DBUF_DATANUMBER_NOT_-
FOUND
ecmdDataBuffer.H, 204
ECMD_DBUF_FILE_FORMAT_-
MISMATCH
ecmdDataBuffer.H, 204
ECMD_DBUF_FILE_OPERATION_-
FAIL
ecmdDataBuffer.H, 204
ECMD_DBUF_FOPEN_FAIL
ecmdDataBuffer.H, 204
ECMD_DBUF_INIT_FAIL
ecmdDataBuffer.H, 204
ECMD_DBUF_INVALID_ARGS
ecmdDataBuffer.H, 204
ECMD_DBUF_INVALID_DATA_-
FORMAT
ecmdDataBuffer.H, 204
ECMD_DBUF_NOT_OWNER
ecmdDataBuffer.H, 205
ECMD_DBUF_SUCCESS
ecmdDataBuffer.H, 204
ECMD_DBUF_UNDEFINED_-
FUNCTION
ecmdDataBuffer.H, 204
ECMD_DBUF_XSTATE_ERROR
ecmdDataBuffer.H, 204
ECMD_DBUF_XSTATE_NOT_-
ENABLED
ecmdDataBuffer.H, 205
ECMD_DEPTH_CAGE
ecmdSharedUtils.H, 209
ECMD_DEPTH_CHIP
ecmdSharedUtils.H, 209
ECMD_DEPTH_CORE
ecmdSharedUtils.H, 209
ECMD_DEPTH_NODE
ecmdSharedUtils.H, 209
ECMD_DEPTH_SLOT
ecmdSharedUtils.H, 209
ECMD_DEPTH_THREAD
ecmdSharedUtils.H, 209
ECMD_DIO_INPUT
ecmdStructs.H, 221
ECMD_DIO_OPEN_DRAIN
ecmdStructs.H, 221
ECMD_DIO_OPEN_SOURCE
ecmdStructs.H, 222
ECMD_DIO_PUSH_PULL
ecmdStructs.H, 222
ECMD_DISPLAY_TARGET_DEFAULT
ecmdSharedUtils.H, 209
ECMD_DLL_CRONUS
ecmdStructs.H, 222
ECMD_DLL_ENV_HW
ecmdStructs.H, 222
ECMD_DLL_ENV_SIM
ecmdStructs.H, 222
ECMD_DLL_IPSERIES
ecmdStructs.H, 222
ECMD_DLL_PRODUCT_ECLIPZ
ecmdStructs.H, 222
ECMD_DLL_PRODUCT_UNKNOWN
ecmdStructs.H, 222
ECMD_DLL_SCAND
ecmdStructs.H, 222
ECMD_DLL_STUB
ecmdStructs.H, 222
ECMD_DLL_UNKNOWN
ecmdStructs.H, 222
ECMD_DLL_ZSERIES
ecmdStructs.H, 222
ECMD_FILE_ARRAYDEF
ecmdStructs.H, 222
ECMD_FILE_HELPTEXT
ecmdStructs.H, 222

ECMD_FILE_SCANDEF
ecmdStructs.H, 222
ECMD_FILE_SCANDEFHASH
ecmdStructs.H, 222
ECMD_FILE_SCOMDATA
ecmdStructs.H, 222
ECMD_FILE_SPYDEF
ecmdStructs.H, 222
ECMD_FILE_SPYDEFHASH
ecmdStructs.H, 222
ECMD_GLOBALVAR_DEBUG
ecmdStructs.H, 223
ECMD_GLOBALVAR_QUIETMODE
ecmdStructs.H, 223
ECMD_I2C_BUSSPEED_100KHZ
ecmdStructs.H, 223
ECMD_I2C_BUSSPEED_400KHZ
ecmdStructs.H, 223
ECMD_I2C_BUSSPEED_50KHZ
ecmdStructs.H, 223
ECMD_INTERFACE_ACCESS
ecmdStructs.H, 219
ECMD_INTERFACE_CFAM
ecmdStructs.H, 219
ECMD_INTERFACE_UNKNOWN
ecmdStructs.H, 219
ECMD_LATCHMODE_FULL
ecmdStructs.H, 224
ECMD_LATCHMODE_PARTIAL
ecmdStructs.H, 224
ECMD_MEMCTRL_REFCLOCK
ecmdStructs.H, 220
ECMD_PROC_REFCLOCK
ecmdStructs.H, 220
ECMD_QUERY_DETAIL_HIGH
ecmdStructs.H, 224
ECMD_QUERY_DETAIL_LOW
ecmdStructs.H, 224
ECMD_SAVE_FORMAT_ASCII
ecmdDataBuffer.H, 206
ECMD_SAVE_FORMAT_BINARY
ecmdDataBuffer.H, 206
ECMD_SAVE_FORMAT_BINARY_-
DATA
ecmdDataBuffer.H, 206
ECMD_SAVE_FORMAT_XSTATE
ecmdDataBuffer.H, 206
ECMD_SELECTED_TARGETS_LOOP
ecmdStructs.H, 221
ECMD_SELECTED_TARGETS_-
LOOP_DEFALL
ecmdStructs.H, 221
ECMD_SELECTED_TARGETS_-
LOOP_VD
ecmdStructs.H, 221
ECMD_SELECTED_TARGETS_-
LOOP_VD_DEFALL
ecmdStructs.H, 221
ECMD_SIM_ERROR
ecmdStructs.H, 223
ECMD_SIM_INFO
ecmdStructs.H, 223
ECMD_SIM_MSG_ALWAYS
ecmdStructs.H, 223
ECMD_SIM_MSG_BROADSIDE
ecmdStructs.H, 223
ECMD_SIM_MSG_CMD_EXE
ecmdStructs.H, 223
ECMD_SIM_MSG_CMD_RS
ecmdStructs.H, 223
ECMD_SIM_MSG_DEBUG
ecmdStructs.H, 223
ECMD_SIM_MSG_END_SD_MSGS
ecmdStructs.H, 223
ECMD_SIM_MSG_EXCEPTION
ecmdStructs.H, 223
ECMD_SIM_MSG_TESTCASE
ecmdStructs.H, 223
ECMD_SIM_PLAIN
ecmdStructs.H, 223
ECMD_SIM_WARNING
ecmdStructs.H, 223
ECMD_SPYTYPE_ALIAS
ecmdStructs.H, 224
ECMD_SPYTYPE_ECCGROUP
ecmdStructs.H, 224
ECMD_SPYTYPE_EDIAL
ecmdStructs.H, 224
ECMD_SPYTYPE_IDIAL
ecmdStructs.H, 224
ECMD_STATIC_DEPTH_LOOP
ecmdStructs.H, 221
ECMD_TARGET_FIELD_UNUSED
ecmdStructs.H, 219
ECMD_TARGET_FIELD_VALID
ecmdStructs.H, 219
ECMD_TARGET_FIELD_WILDCARD
ecmdStructs.H, 219
ECMD_TARGET_THREAD_ALIVE
ecmdStructs.H, 219
ECMD_TARGET_UNKNOWN_STATE
ecmdStructs.H, 219
ECMD_TRACE_PROCEDURE
ecmdStructs.H, 224
ECMD_TRACE_SCAN
ecmdStructs.H, 224
ECMD_WRITE_MODE
ecmdDataBuffer.H, 207

ecmdArrayData, 13
ecmdArrayData
    arrayName, 14
    clockDomain, 14
    clockState, 14
    isCoreRelated, 14
    length, 14
    readAddressLength, 14
    width, 14
    writeAddressLength, 14
ecmdArrayEntry, 15
ecmdArrayEntry
    address, 15
    buffer, 15
    rc, 15
ecmdBitsHeader
    ecmdUtils.H, 227
ecmdCacheFlush
    ecmdClientPerlapi.H, 176
ecmdCacheType_t
    ecmdStructs.H, 219
ecmdCageData, 16
ecmdCageData
    cageId, 16
    nodeData, 16
    unitId, 16
ecmdChipData, 17
ecmdChipData
    chipCommonType, 18
    chipEc, 18
    chipFlags, 18
    chipShortType, 18
    chipType, 18
    coreData, 18
    interfaceType, 18
    numProcCores, 18
    pos, 18
    simModelEc, 18
    unitId, 18
ecmdChipInterfaceType_t
    ecmdStructs.H, 219
ecmdChipTarget, 20
ecmdChipTarget
    cage, 21
    cageState, 22
    chipType, 21
    chipTypeState, 22
    core, 21
    coreState, 22
    node, 21
    nodeState, 22
    pos, 21
    posState, 22
    slot, 21
    slotState, 22
    thread, 21
    threadState, 22
    unitId, 22
    unitIdState, 22
ecmdChipTargetState_t
    ecmdStructs.H, 219
ecmdClientPerlapi.H, 124
ecmdClientPerlapi.H
    ecmdCacheFlush, 176
    ecmdCommandArgs, 138
    ecmdConfigureTarget, 192
    ecmdDeconfigureTarget, 191
    ecmdDelay, 190
    ecmdDisablePerlSafeMode, 138
    ecmdDisableRingCache, 155
    ecmdEnablePerlSafeMode, 138
    ecmdEnableRingCache, 155
    ecmdFlushRegisteredErrorMsgs, 189
    ecmdFlushRingCache, 155
    ecmdGetConfiguration, 190
    ecmdGetErrorMsg, 188
    ecmdGetGlobalVar, 189
    ecmdGpioConfigPin, 198
    ecmdGpioReadLatch, 199
    ecmdGpioReadPin, 199
    ecmdGpioReadPins, 200
    ecmdGpioWriteLatch, 200
    ecmdGpioWriteLatches, 201
    ecmdI2cRead, 196
    ecmdI2cReadOffset, 197
    ecmdI2cReset, 196
    ecmdI2cWrite, 197
    ecmdI2cWriteOffset, 198
    ecmdIsRingCacheEnabled, 156
    ecmdLoadDll, 137
    ecmdOutput, 189
    ecmdOutputError, 189
    ecmdOutputWarning, 189
    ecmdParseOption, 138
    ecmdParseOptionWithArgs, 139
    ecmdQueryArray, 142
    ecmdQueryChipScandefVersion, 185
    ecmdQueryChipSimModelVersion, 185
    ecmdQueryClockState, 158
    ecmdQueryConfig, 140
    ecmdQueryDllInfo, 139
    ecmdQueryFileLocation, 144
    ecmdQueryLatch, 142
    ecmdQueryProcRegisterInfo, 162
    ecmdQueryRing, 141
    ecmdQueryScom, 143
    ecmdQuerySelected, 141
    ecmdQuerySpy, 143

ecmdQueryTargetConfigured, 145
ecmdQueryTraceArray, 144
ecmdQueryTraceMode, 189
ecmdQueryVersionGreater, 139
ecmdRegisterErrorMsg, 188
ecmdSequenceIdToTarget, 193
ecmdSetClockMultDiv, 160
ecmdSetClockSpeed, 160
ecmdSetConfiguration, 191
ecmdSetTraceMode, 189
ecmdTargetToUnitId, 192
ecmdUnitIdStringToTarget, 192
ecmdUnitIdToString, 193
ecmdUnitIdToTarget, 193
ecmdUnloadDll, 138
getArray, 156
getArrayMultiple, 156
getCfamRegister, 151
getFpr, 168
getFprMultiple, 168
getGpr, 165
getGprMultiple, 166
getLatch, 146
getMemDma, 174
getMemMemCtrl, 175
getMemProc, 173
getModuleVpdImage, 195
getModuleVpdKeyword, 194
getRing, 145
getRingWithModifier, 147
getScom, 149
getSlb, 170
getSlbMultiple, 170
getSpr, 163
getSprMultiple, 163
getSpy, 152
getSpyEnum, 152
getSpyEpCheckers, 153
getSpyGroups, 153
getTraceArray, 172
getTraceArrayMultiple, 172
iStepsByName, 161
iStepsByNameMultiple, 162
iStepsByNameRange, 162
iStepsByNumber, 161
makeSPSystemCall, 190
putArray, 157
putArrayMultiple, 158
putCfamRegister, 151
putFpr, 169
putFprMultiple, 169
putGpr, 166
putGprMultiple, 167
putLatch, 147
putMemDma, 174
putMemMemCtrl, 175
putMemProc, 173
putModuleVpdImage, 195
putModuleVpdKeyword, 194
putRing, 146
putRingWithModifier, 148
putScom, 149
putSlb, 171
putSlbMultiple, 171
putSpr, 164
putSprMultiple, 165
putSpy, 154
putSpyEnum, 154
sendCmd, 150
simaet, 176
simCallFusionCommand, 185
simcheckpoint, 177
simclock, 177
simecho, 177
simexit, 177
simEXPECTFAC, 178
simexpecttcfac, 178
simFusionRand32, 186
simgetcurrentcycle, 178
simGetDial, 187
simGetEnvironment, 188
simGETFAC, 179
simGETFACX, 179
simGetHierarchy, 184
simGetInFile, 187
simGetModelInfo, 188
simGetOutFile, 187
simgettcfac, 179
siminit, 180
simOutputFusionMessage, 186
simPOLLFAC, 180
simpolltcfac, 181
simPutDial, 187
simPUTFAC, 181
simPUTFACX, 181
simputtcfac, 182
simrestart, 182
simSetFusionMessageFormat, 186
simSTKFAC, 182
simstktcfac, 183
simSUBCMD, 183
simtckinterval, 183
simUNSTICK, 184
simunsticktcfac, 184
startClocks, 159
stopClocks, 159
ecmdClockRange_t
ecmdStructs.H, 219

ecmdClockSetMode_t
ecmdStructs.H, 220
ecmdClockSpeedType_t
ecmdStructs.H, 220
ecmdClockState_t
ecmdStructs.H, 220
ecmdClockType_t
ecmdStructs.H, 220
ecmdCommandArgs
ecmdClientPerlapi.H, 138
ecmdConfigLooperInit
ecmdUtils.H, 226
ecmdConfigLooperNext
ecmdUtils.H, 226
ecmdConfigLoopMode_t
ecmdStructs.H, 220
ecmdConfigLoopType_t
ecmdStructs.H, 221
ecmdConfigureTarget
ecmdClientPerlapi.H, 192
ecmdConfigValid_t
ecmdStructs.H, 221
ecmdCoreData, 23
ecmdCoreData
coreId, 23
numProcThreads, 23
threadData, 23
unitId, 23
ecmdCurCage
ecmdLooperData, 78
ecmdCurChip
ecmdLooperData, 78
ecmdCurCore
ecmdLooperData, 78
ecmdCurNode
ecmdLooperData, 78
ecmdCurSlot
ecmdLooperData, 78
ecmdCurThread
ecmdLooperData, 78
ecmdDataBuffer, 24
ecmdDataBuffer, 33
ecmdDataBuffer
~ecmdDataBuffer, 33
applyInversionMask, 44
clear, 33
clearBit, 39
concat, 49
copy, 53
disableXstateBuffer, 60
ecmdDataBuffer, 33
ecmdDataBufferImplementationHelper, 67
enableXstateBuffer, 60
evenParity, 55
extract, 46, 47
extractPreserve, 47
extractToRight, 48
fillDataStr, 66
flatten, 54
flattenSize, 54
flipBit, 39, 40
flushTo0, 43
flushTo1, 43
flushToX, 61
genAsciiStr, 56, 57
genBinStr, 56, 57
genHexLeftStr, 56, 57
genHexRightStr, 56, 57
genXstateStr, 57, 58
getBitLength, 34
getByte, 38
getByteLength, 34
getCapacity, 34
getDoubleWord, 39
getHalfWord, 38
getNumBitsSet, 41
getWord, 37
getWordLength, 33
getXstate, 61
growBitLength, 36
hasXstate, 61
insert, 44, 45
insertFromBin, 59
insertFromBinAndResize, 60
insertFromHexLeft, 58
insertFromHexLeftAndResize, 58
insertFromHexRight, 58
insertFromHexRightAndResize, 59
insertFromRight, 45, 46
invert, 43
isBitClear, 40, 41
isBitSet, 40
isBufferOptimizable, 36
isXstateEnabled, 60
iv_BufferOptimizable, 67
iv_Capacity, 67
iv_Data, 67
iv_DataStr, 67
iv_NumBits, 67
iv_NumWords, 67
iv_RealData, 67
iv_UserOwned, 67
iv_XstateEnabled, 67
memCopyIn, 53
memCopyInXstate, 62
memCopyOut, 53
memCopyOutXstate, 63

merge, 50
oddParity, 54, 55
operator &, 66
operator!=, 66
operator==, 66
operator|, 66
queryErrorState, 66
queryNumOfBuffers, 65
readFile, 64
readFileMultiple, 64
readFileStream, 65
reverse, 43
rotateLeft, 43
rotateRight, 42
setAnd, 52
setBit, 36
setBitLength, 35
setByte, 37
setByteLength, 34
setCapacity, 35
setDoubleWord, 38
setHalfWord, 38
setOr, 49, 50
setWord, 37
setWordLength, 34
setXor, 51
setXstate, 61, 62
shareBuffer, 65
shiftLeft, 41
shiftLeftAndResize, 42
shiftRight, 41
shiftRightAndResize, 42
shrinkBitLength, 35
unflatten, 54
writeBit, 37
writeFile, 63
writeFileMultiple, 63
writeFileStream, 64
ecmdDataBuffer.H, 202
ECMD_APPEND_MODE, 207
ECMD_SAVE_FORMAT_ASCII, 206
ECMD_SAVE_FORMAT_BINARY, 206
ECMD_SAVE_FORMAT_BINARY_DATA, 206
ECMD_SAVE_FORMAT_XSTATE, 206
ECMD_WRITE_MODE, 207
ecmdDataBuffer.H
ECMD_DBUF_BUFFER_OVERFLOW, 204
ECMD_DBUF_DATANUMBER_NOT_FOUND, 204
ECMD_DBUF_FILE_FORMAT_MISMATCH, 204
ECMD_DBUF_FILE_OPERATION_FAIL, 204
ECMD_DBUF_FOPEN_FAIL, 204
ECMD_DBUF_INIT_FAIL, 204
ECMD_DBUF_INVALID_ARGS, 204
ECMD_DBUF_INVALID_DATA_FORMAT, 204
ECMD_DBUF_NOT_OWNER, 205
ECMD_DBUF_SUCCESS, 204
ECMD_DBUF_UNDEFINED_FUNCTION, 204
ECMD_DBUF_XSTATE_ERROR, 204
ECMD_DBUF_XSTATE_NOT_ENABLED, 205
ecmdFormatType_t, 206
ecmdWriteMode_t, 206
ETRAC0, 205
ETRAC1, 206
ETRAC2, 206
ETRAC3, 206
ETRAC4, 206
ETRAC5, 206
ETRAC6, 206
ETRAC7, 206
ETRAC8, 206
ETRAC9, 206
ecmdDataBufferImplementationHelper, 68
ecmdDataBuffer, 67
ecmdDataBufferImplementationHelper
applyRawBufferToXstate, 68
getDataPtr, 68
ecmdDeconfigureTarget
ecmdClientPerlapi.H, 191
ecmdDelay
ecmdClientPerlapi.H, 190
ecmdDioMode_t
ecmdStructs.H, 221
ecmdDisablePerlSafeMode
ecmdClientPerlapi.H, 138
ecmdDisableRingCache
ecmdClientPerlapi.H, 155
ecmdDisplayDllInfo
ecmdUtils.H, 228
ecmdDisplayScomData
ecmdUtils.H, 228
ecmdDllEnv_t
ecmdStructs.H, 222
ecmdDllInfo, 69
ecmdDllInfo
dllBuildDate, 70
dllBuildInfo, 70

dllCapiVersion, 70
dllEnv, 69
dllProduct, 69
dllProductType, 69
dllType, 69
ecmdDllProduct_t
ecmdStructs.H, 222
ecmdDllType_t
ecmdStructs.H, 222
ecmdEnablePerlSafeMode
ecmdClientPerlapi.H, 138
ecmdEnableRingCache
ecmdClientPerlapi.H, 155
ecmdFileType_t
ecmdStructs.H, 222
ecmdFlushRegisteredErrorMsgs
ecmdClientPerlapi.H, 189
ecmdFlushRingCache
ecmdClientPerlapi.H, 155
ecmdFormatType_t
ecmdDataBuffer.H, 206
ecmdFusionMessageType_t
ecmdStructs.H, 222
ecmdFusionSeverity_t
ecmdStructs.H, 223
ecmdGenB32FromHex
ecmdSharedUtils.H, 210
ecmdGenB32FromHexLeft
ecmdSharedUtils.H, 210
ecmdGenB32FromHexRight
ecmdSharedUtils.H, 210
ecmdGenEbcdic
ecmdSharedUtils.H, 210
ecmdGetChipData
ecmdUtils.H, 228
ecmdGetConfiguration
ecmdClientPerlapi.H, 190
ecmdGetErrorMsg
ecmdClientPerlapi.H, 188
ecmdGetGlobalVar
ecmdClientPerlapi.H, 189
ecmdGetSharedLibVersion
ecmdStructs.H, 224
ecmdGlobalVarType_t
ecmdStructs.H, 223
ecmdGpioConfigPin
ecmdClientPerlapi.H, 198
ecmdGpioReadLatch
ecmdClientPerlapi.H, 199
ecmdGpioReadPin
ecmdClientPerlapi.H, 199
ecmdGpioReadPins
ecmdClientPerlapi.H, 200
ecmdGpioWriteLatch
ecmdClientPerlapi.H, 200
ecmdGpioWriteLatches
ecmdClientPerlapi.H, 201
ecmdHashString32
ecmdSharedUtils.H, 211
ecmdHexToUInt32
ecmdSharedUtils.H, 211
ecmdI2cBusSpeed_t
ecmdStructs.H, 223
ecmdI2cRead
ecmdClientPerlapi.H, 196
ecmdI2cReadOffset
ecmdClientPerlapi.H, 197
ecmdI2cReset
ecmdClientPerlapi.H, 196
ecmdI2cWrite
ecmdClientPerlapi.H, 197
ecmdI2cWriteOffset
ecmdClientPerlapi.H, 198
ecmdIndexEntry, 71
ecmdIndexEntry
buffer, 71
index, 71
rc, 71
ecmdIndexVectorEntry, 72
ecmdIndexVectorEntry
buffer, 72
index, 72
rc, 72
ecmdIsRingCacheEnabled
ecmdClientPerlapi.H, 156
ecmdLatchData, 73
ecmdLatchData
bitLength, 73
clockDomain, 73
clockState, 74
isCoreRelated, 73
latchName, 73
ringName, 73
ecmdLatchEntry, 75
ecmdLatchEntry
buffer, 75
latchEndBit, 75
latchName, 75
latchStartBit, 75
rc, 76
ringName, 75
ecmdLatchMode_t
ecmdStructs.H, 223
ecmdLoadDll
ecmdClientPerlapi.H, 137
ecmdLooperData, 77
ecmdLooperData
curUnitIdTarget, 79

ecmdCurCage, 78
ecmdCurChip, 78
ecmdCurCore, 78
ecmdCurNode, 78
ecmdCurSlot, 78
ecmdCurThread, 78
ecmdLooperInitFlag, 78
ecmdLoopMode, 78
ecmdSystemConfigData, 78
ecmdUseUnitid, 78
prevTarget, 78
unitIdTargets, 78
ecmdLooperInitFlag
ecmdLooperData, 78
ecmdLoopMode
ecmdLooperData, 78
ecmdMemoryEntry, 80
ecmdMemoryEntry
address, 80
data, 80
tags, 80
ecmdNameEntry, 81
ecmdNameEntry
buffer, 81
name, 81
rc, 81
ecmdNameVectorEntry, 82
ecmdNameVectorEntry
buffer, 82
name, 82
rc, 82
ecmdNodeData, 83
ecmdNodeData
nodeId, 83
slotData, 83
unitId, 83
ecmdOptimizableDataBuffer, 84
ecmdOptimizableDataBuffer, 84
ecmdOptimizableDataBuffer
~ecmdOptimizableDataBuffer, 84
ecmdOptimizableDataBuffer, 84
ecmdOutput
ecmdClientPerlapi.H, 189
ecmdOutputError
ecmdClientPerlapi.H, 189
ecmdOutputWarning
ecmdClientPerlapi.H, 189
ecmdParseOption
ecmdClientPerlapi.H, 138
ecmdParseOptionWithArgs
ecmdClientPerlapi.H, 139
ecmdParseTokens
ecmdSharedUtils.H, 209
ecmdProcRegisterInfo, 85
ecmdProcRegisterInfo
bitLength, 85
threadReplicated, 85
totalEntries, 85
ecmdQueryArray
ecmdClientPerlapi.H, 142
ecmdQueryChipScandefVersion
ecmdClientPerlapi.H, 185
ecmdQueryChipSimModelVersion
ecmdClientPerlapi.H, 185
ecmdQueryClockState
ecmdClientPerlapi.H, 158
ecmdQueryConfig
ecmdClientPerlapi.H, 140
ecmdQueryData, 86
ecmdQueryData
cageData, 86
detailLevel, 86
ecmdQueryDetail_t
ecmdStructs.H, 224
ecmdQueryDllInfo
ecmdClientPerlapi.H, 139
ecmdQueryFileLocation
ecmdClientPerlapi.H, 144
ecmdQueryLatch
ecmdClientPerlapi.H, 142
ecmdQueryProcRegisterInfo
ecmdClientPerlapi.H, 162
ecmdQueryRing
ecmdClientPerlapi.H, 141
ecmdQueryScom
ecmdClientPerlapi.H, 143
ecmdQuerySelected
ecmdClientPerlapi.H, 141
ecmdQuerySpy
ecmdClientPerlapi.H, 143
ecmdQueryTargetConfigured
ecmdClientPerlapi.H, 145
ecmdQueryTraceArray
ecmdClientPerlapi.H, 144
ecmdQueryTraceMode
ecmdClientPerlapi.H, 189
ecmdQueryVersionGreater
ecmdClientPerlapi.H, 139
ecmdReadDataFormatted
ecmdUtils.H, 226
ecmdReadDcard
ecmdSharedUtils.H, 212
ecmdRegisterErrorMsg
ecmdClientPerlapi.H, 188
ecmdRingData, 87
ecmdRingData
address, 87
bitLength, 87

clockDomain, 88
clockState, 88
hasInversionMask, 88
isCheckable, 88
isCoreRelated, 88
ringNames, 87
supportsBroadsideLoad, 88
ecmdScomData, 89
ecmdScomData
address, 89
clockDomain, 89
clockState, 89
isCoreRelated, 89
ecmdSequenceIdToTarget
ecmdClientPerlapi.H, 193
ecmdSetClockMultDiv
ecmdClientPerlapi.H, 160
ecmdSetClockSpeed
ecmdClientPerlapi.H, 160
ecmdSetConfiguration
ecmdClientPerlapi.H, 191
ecmdSetTargetDepth
ecmdSharedUtils.H, 211
ecmdSetTraceMode
ecmdClientPerlapi.H, 189
ecmdSharedUtils.H, 208
ECMD_DEPTH_CAGE, 209
ECMD_DEPTH_CHIP, 209
ECMD_DEPTH_CORE, 209
ECMD_DEPTH_NODE, 209
ECMD_DEPTH_SLOT, 209
ECMD_DEPTH_THREAD, 209
ECMD_DISPLAY_TARGET_-
DEFAULT, 209
ecmdSharedUtils.H
ecmdGenB32FromHex, 210
ecmdGenB32FromHexLeft, 210
ecmdGenB32FromHexRight, 210
ecmdGenEbcdic, 210
ecmdHashString32, 211
ecmdHexToUInt32, 211
ecmdParseTokens, 209
ecmdReadDcard, 212
ecmdSetTargetDepth, 211
ecmdTargetDepth_t, 209
ecmdTargetDisplayMode_t, 209
ecmdWriteTarget, 212
ecmdSimModelInfo, 90
ecmdSimModelInfo
modeldate, 90
modelname, 90
modeltime, 90
multivalue, 90
ecmdSlotData, 91
ecmdSlotData
chipData, 91
slotId, 91
unitId, 91
ecmdSpyData, 92
ecmdSpyData
bitLength, 92
clockDomain, 93
clockState, 93
enums, 93
epCheckers, 93
isCoreRelated, 93
isEccChecked, 93
isEnumerated, 93
spyName, 92
spyType, 93
ecmdSpyGroupData, 94
ecmdSpyGroupData
deadbitsMask, 94
extractBuffer, 94
ecmdSpyType_t
ecmdStructs.H, 224
ecmdStructs.H, 213
ECMD_ALL_TARGETS_LOOP, 221
ECMD_CACHE_LEVEL1D, 219
ECMD_CACHE_LEVEL1I, 219
ECMD_CACHE_LEVEL2, 219
ECMD_CACHE_LEVEL3, 219
ECMD_CACHE_LEVEL4, 219
ECMD_CACHE_UNKNOWN, 219
ECMD_CLOCK_CYCLETIME_-
SPEC, 220
ECMD_CLOCK_FREQUENCY_-
SPEC, 220
ECMD_CLOCK_ONE_STEP, 220
ECMD_CLOCK_RANGE_-
DEFAULT, 220
ECMD_CLOCK_RANGE_HIGH, 220
ECMD_CLOCK_RANGE_-
HIGHEST, 220
ECMD_CLOCK_RANGE_LOW, 220
ECMD_CLOCK_RANGE_LOWEST,
220
ECMD_CLOCK_RANGE_MIDDLE,
220
ECMD_CLOCK_STEER, 220
ECMD_CLOCKSTATE_NA, 220
ECMD_CLOCKSTATE_OFF, 220
ECMD_CLOCKSTATE_ON, 220
ECMD_CLOCKSTATE_UNKNOWN,
220
ECMD_CONFIG_VALID_FIELD_-
ALPHA, 221

ECMD_CONFIG_VALID_FIELD_-
BOTH, 221
ECMD_CONFIG_VALID_FIELD_-
NONE, 221
ECMD_CONFIG_VALID_FIELD_-
NUMERIC, 221
ECMD_DIO_INPUT, 221
ECMD_DIO_OPEN_DRAIN, 221
ECMD_DIO_OPEN_SOURCE, 222
ECMD_DIO_PUSH_PULL, 222
ECMD_DLL_CRONUS, 222
ECMD_DLL_ENV_HW, 222
ECMD_DLL_ENV_SIM, 222
ECMD_DLL_IPSERIES, 222
ECMD_DLL_PRODUCT_ECLIPZ,
222
ECMD_DLL_PRODUCT_-
UNKNOWN, 222
ECMD_DLL_SCAND, 222
ECMD_DLL_STUB, 222
ECMD_DLL_UNKNOWN, 222
ECMD_DLL_ZSERIES, 222
ECMD_FILE_ARRAYDEF, 222
ECMD_FILE_HELPTEXT, 222
ECMD_FILE_SCANDEF, 222
ECMD_FILE_SCANDEFHASH, 222
ECMD_FILE_SCOMDATA, 222
ECMD_FILE_SPYDEF, 222
ECMD_FILE_SPYDEFHASH, 222
ECMD_GLOBALVAR_DEBUG, 223
ECMD_GLOBALVAR_-
QUIETMODE, 223
ECMD_I2C_BUSSPEED_100KHZ,
223
ECMD_I2C_BUSSPEED_400KHZ,
223
ECMD_I2C_BUSSPEED_50KHZ,
223
ECMD_INTERFACE_ACCESS, 219
ECMD_INTERFACE_CFAM, 219
ECMD_INTERFACE_UNKNOWN,
219
ECMD_LATCHMODE_FULL, 224
ECMD_LATCHMODE_PARTIAL,
224
ECMD_MEMCTRL_REFCLOCK,
220
ECMD_PROC_REFCLOCK, 220
ECMD_QUERY_DETAIL_HIGH,
224
ECMD_QUERY_DETAIL_LOW, 224
ECMD_SELECTED_TARGETS_-
LOOP, 221
ECMD_SELECTED_TARGETS_-
LOOP_DEFALL, 221
ECMD_SELECTED_TARGETS_-
LOOP_VD, 221
ECMD_SELECTED_TARGETS_-
LOOP_VD_DEFALL, 221
ECMD_SIM_ERROR, 223
ECMD_SIM_INFO, 223
ECMD_SIM_MSG_ALWAYS, 223
ECMD_SIM_MSG_BROADSIDE,
223
ECMD_SIM_MSG_CMD_EXE, 223
ECMD_SIM_MSG_CMD_RS, 223
ECMD_SIM_MSG_DEBUG, 223
ECMD_SIM_MSG_END_SD_-
MSGS, 223
ECMD_SIM_MSG_EXCEPTION,
223
ECMD_SIM_MSG_TESTCASE, 223
ECMD_SIM_PLAIN, 223
ECMD_SIM_WARNING, 223
ECMD_SPYTYPE_ALIAS, 224
ECMD_SPYTYPE_ECCGROUP, 224
ECMD_SPYTYPE_EDIAL, 224
ECMD_SPYTYPE_IDIAL, 224
ECMD_STATIC_DEPTH_LOOP,
221
ECMD_TARGET_FIELD_UNUSED,
219
ECMD_TARGET_FIELD_VALID,
219
ECMD_TARGET_FIELD_-
WILDCARD, 219
ECMD_TARGET_THREAD_-
ALIVE, 219
ECMD_TARGET_UNKNOWN_-
STATE, 219
ECMD_TRACE_PROCEDURE, 224
ECMD_TRACE_SCAN, 224
ecmdStructs.H
CAGE_HDR_MAGIC, 218
CHIP_HDR_MAGIC, 218
CORE_HDR_MAGIC, 218
ECMD_CAPI_VERSION, 218
ECMD_CHIPFLAG_BUSMASK, 218
ECMD_CHIPFLAG_FSI, 219
ECMD_CHIPFLAG_JTAG, 218
ECMD_CHIPFLAG_RSVDBUS1, 218
ECMD_CHIPFLAG_RSVDBUS2, 219
ECMD_CHIPT_IOBDG, 218
ECMD_CHIPT_IOHUB, 218
ECMD_CHIPT_MEM_BUF, 218
ECMD_CHIPT_MEM_CNTRL, 218

ECMD_CHIPT_MEM_L2CACHE, 218
ECMD_CHIPT_MEM_L3CACHE, 218
ECMD_CHIPT_MUX, 218
ECMD_CHIPT_PROCESSOR, 218
ECMD_CHIPT_SERVICE_PROCESSOR, 218
ecmdCacheType_t, 219
ecmdChipInterfaceType_t, 219
ecmdChipTargetState_t, 219
ecmdClockRange_t, 219
ecmdClockSetMode_t, 220
ecmdClockSpeedType_t, 220
ecmdClockState_t, 220
ecmdClockType_t, 220
ecmdConfigLoopMode_t, 220
ecmdConfigLoopType_t, 221
ecmdConfigValid_t, 221
ecmdDioMode_t, 221
ecmdDllEnv_t, 222
ecmdDllProduct_t, 222
ecmdDllType_t, 222
ecmdFileType_t, 222
ecmdFusionMessageType_t, 222
ecmdFusionSeverity_t, 223
ecmdGetSharedLibVersion, 224
ecmdGlobalVarType_t, 223
ecmdI2cBusSpeed_t, 223
ecmdLatchMode_t, 223
ecmdQueryDetail_t, 224
ecmdSpyType_t, 224
ecmdTraceType_t, 224
NODE_HDR_MAGIC, 218
QD_HDR_MAGIC, 218
SLOT_HDR_MAGIC, 218
THREAD_HDR_MAGIC, 218
ecmdSystemConfigData
ecmdLooperData, 78
ecmdTargetDepth_t
ecmdSharedUtils.H, 209
ecmdTargetDisplayMode_t
ecmdSharedUtils.H, 209
ecmdTargetToUnitId
ecmdClientPerlapi.H, 192
ecmdThreadData, 95
ecmdThreadData
threadId, 95
unitId, 95
ecmdTraceArrayData, 96
ecmdTraceArrayData
clockDomain, 96
clockState, 97
isCoreRelated, 96
length, 96
traceArrayName, 96
width, 96
ecmdTraceType_t
ecmdStructs.H, 224
ecmdUnitIdStringToTarget
ecmdClientPerlapi.H, 192
ecmdUnitIdToString
ecmdClientPerlapi.H, 193
ecmdUnitIdToTarget
ecmdClientPerlapi.H, 193
ecmdUnloadDll
ecmdClientPerlapi.H, 138
ecmdUseUnitid
ecmdLooperData, 78
ecmdUtils.H, 225
ecmdUtils.H
decToUInt32, 227
ecmdBitsHeader, 227
ecmdConfigLooperInit, 226
ecmdConfigLooperNext, 226
ecmdDisplayDllInfo, 228
ecmdDisplayScomData, 228
ecmdGetChipData, 228
ecmdReadDataFormatted, 226
ecmdWriteDataFormatted, 227
ecmdWriteDataFormatted
ecmdUtils.H, 227
ecmdWriteMode_t
ecmdDataBuffer.H, 206
ecmdWriteTarget
ecmdSharedUtils.H, 212
enableXstateBuffer
ecmdDataBuffer, 60
enums
ecmdSpyData, 93
epCheckers
ecmdSpyData, 93
ETRAC0
ecmdDataBuffer.H, 205
ETRAC1
ecmdDataBuffer.H, 206
ETRAC2
ecmdDataBuffer.H, 206
ETRAC3
ecmdDataBuffer.H, 206
ETRAC4
ecmdDataBuffer.H, 206
ETRAC5
ecmdDataBuffer.H, 206
ETRAC6
ecmdDataBuffer.H, 206
ETRAC7
ecmdDataBuffer.H, 206

ETRAC8
ecmdDataBuffer.H, 206
ETRAC9
ecmdDataBuffer.H, 206
evenParity
ecmdDataBuffer, 55
extract
ecmdDataBuffer, 46, 47
extractBuffer
ecmdSpyGroupData, 94
extractPreserve
ecmdDataBuffer, 47
extractToRight
ecmdDataBuffer, 48

fillDataStr
ecmdDataBuffer, 66
flatten
ecmdDataBuffer, 54
flattenSize
ecmdDataBuffer, 54
flipBit
ecmdDataBuffer, 39, 40
flushTo0
ecmdDataBuffer, 43
flushTo1
ecmdDataBuffer, 43
flushToX
ecmdDataBuffer, 61

genAsciiStr
ecmdDataBuffer, 56, 57
genBinStr
ecmdDataBuffer, 56, 57
genHexLeftStr
ecmdDataBuffer, 56, 57
genHexRightStr
ecmdDataBuffer, 56, 57
genXstateStr
ecmdDataBuffer, 57, 58
getArray
ecmdClientPerlapi.H, 156
getArrayMultiple
ecmdClientPerlapi.H, 156
getBitLength
ecmdDataBuffer, 34
getByte
ecmdDataBuffer, 38
getByteLength
ecmdDataBuffer, 34
getCapacity
ecmdDataBuffer, 34
getCfamRegister
ecmdClientPerlapi.H, 151
getDataPtr
ecmdDataBufferImplementationHelper, 68
getDoubleWord
ecmdDataBuffer, 39
getFpr
ecmdClientPerlapi.H, 168
getFprMultiple
ecmdClientPerlapi.H, 168
getGpr
ecmdClientPerlapi.H, 165
getGprMultiple
ecmdClientPerlapi.H, 166
getHalfWord
ecmdDataBuffer, 38
getLatch
ecmdClientPerlapi.H, 146
getMemDma
ecmdClientPerlapi.H, 174
getMemMemCtrl
ecmdClientPerlapi.H, 175
getMemProc
ecmdClientPerlapi.H, 173
getModuleVpdImage
ecmdClientPerlapi.H, 195
getModuleVpdKeyword
ecmdClientPerlapi.H, 194
getNumBitsSet
ecmdDataBuffer, 41
getRing
ecmdClientPerlapi.H, 145
getRingWithModifier
ecmdClientPerlapi.H, 147
getScom
ecmdClientPerlapi.H, 149
getSlb
ecmdClientPerlapi.H, 170
getSlbMultiple
ecmdClientPerlapi.H, 170
getSpr
ecmdClientPerlapi.H, 163
getSprMultiple
ecmdClientPerlapi.H, 163
getSpy
ecmdClientPerlapi.H, 152
getSpyEnum
ecmdClientPerlapi.H, 152
getSpyEpCheckers
ecmdClientPerlapi.H, 153
getSpyGroups
ecmdClientPerlapi.H, 153
getTraceArray
ecmdClientPerlapi.H, 172
getTraceArrayMultiple

ecmdClientPerlapi.H, 172
getWord
ecmdDataBuffer, 37
getWordLength
ecmdDataBuffer, 33
getXstate
ecmdDataBuffer, 61
gipAbort
gipClientPerlapi.H, 231
gipClearSoftwareBreakpoint
gipClientPerlapi.H, 234
gipClientPerlapi.H, 229
gipClientPerlapi.H
gipAbort, 231
gipClearSoftwareBreakpoint, 234
gipConnectionTest, 235
gipGetMemProcVariousAddrType, 232
gipGetSoftwareBreakpoint, 234
gipGetSystemInfo, 235
gipInitExtension, 230
gipPutMemProcVariousAddrType, 233
gipRelease, 231
gipReserve, 231
gipSetServAddr, 232
gipSetSoftwareBreakpoint, 233
gipConnectionTest
gipClientPerlapi.H, 235
gipGetMemProcVariousAddrType
gipClientPerlapi.H, 232
gipGetSoftwareBreakpoint
gipClientPerlapi.H, 234
gipGetSystemInfo
gipClientPerlapi.H, 235
gipInitExtension
gipClientPerlapi.H, 230
gipPutMemProcVariousAddrType
gipClientPerlapi.H, 233
gipRelease
gipClientPerlapi.H, 231
gipReserve
gipClientPerlapi.H, 231
gipSetServAddr
gipClientPerlapi.H, 232
gipSetSoftwareBreakpoint
gipClientPerlapi.H, 233
growBitLength
ecmdDataBuffer, 36

hasInversionMask
ecmdRingData, 88
hasXstate
ecmdDataBuffer, 61

index
ecmdIndexEntry, 71
ecmdIndexVectorEntry, 72
insert
ecmdDataBuffer, 44, 45
insertFromBin
ecmdDataBuffer, 59
insertFromBinAndResize
ecmdDataBuffer, 60
insertFromHexLeft
ecmdDataBuffer, 58
insertFromHexLeftAndResize
ecmdDataBuffer, 58
insertFromHexRight
ecmdDataBuffer, 58
insertFromHexRightAndResize
ecmdDataBuffer, 59
insertFromRight
ecmdDataBuffer, 45, 46
interfaceType
ecmdChipData, 18
invert
ecmdDataBuffer, 43
isBitClear
ecmdDataBuffer, 40, 41
isBitSet
ecmdDataBuffer, 40
isBufferOptimizable
ecmdDataBuffer, 36
isCheckable
ecmdRingData, 88
isCoreRelated
ecmdArrayData, 14
ecmdLatchData, 73
ecmdRingData, 88
ecmdScomData, 89
ecmdSpyData, 93
ecmdTraceArrayData, 96
isEccChecked
ecmdSpyData, 93
isEnumerated
ecmdSpyData, 93
iStepsByName
ecmdClientPerlapi.H, 161
iStepsByNameMultiple
ecmdClientPerlapi.H, 162
iStepsByNameRange
ecmdClientPerlapi.H, 162
iStepsByNumber
ecmdClientPerlapi.H, 161
isXstateEnabled
ecmdDataBuffer, 60
iv_BufferOptimizable
ecmdDataBuffer, 67
iv_Capacity

ecmdDataBuffer, 67
iv_Data
ecmdDataBuffer, 67
iv_DataStr
ecmdDataBuffer, 67
iv_NumBits
ecmdDataBuffer, 67
iv_NumWords
ecmdDataBuffer, 67
iv_RealData
ecmdDataBuffer, 67
iv_UserOwned
ecmdDataBuffer, 67
iv_XstateEnabled
ecmdDataBuffer, 67

latchEndBit
ecmdLatchEntry, 75
latchName
ecmdLatchData, 73
ecmdLatchEntry, 75
latchStartBit
ecmdLatchEntry, 75
length
ecmdArrayData, 14
ecmdTraceArrayData, 96

makeSPSystemCall
ecmdClientPerlapi.H, 190
memCopyIn
ecmdDataBuffer, 53
memCopyInXstate
ecmdDataBuffer, 62
memCopyOut
ecmdDataBuffer, 53
memCopyOutXstate
ecmdDataBuffer, 63
merge
ecmdDataBuffer, 50
modeldate
ecmdSimModelInfo, 90
modelname
ecmdSimModelInfo, 90
modeltime
ecmdSimModelInfo, 90
multivalue
ecmdSimModelInfo, 90

name
ecmdNameEntry, 81
ecmdNameVectorEntry, 82
node
ecmdChipTarget, 21
NODE_HDR_MAGIC
ecmdStructs.H, 218
nodeData
ecmdCageData, 16
nodeId
ecmdNodeData, 83
nodeState
ecmdChipTarget, 22
numProcCores
ecmdChipData, 18
numProcThreads
ecmdCoreData, 23

oddParity
ecmdDataBuffer, 54, 55
operator &
ecmdDataBuffer, 66
operator!=
ecmdDataBuffer, 66
operator==
ecmdDataBuffer, 66
operator|
ecmdDataBuffer, 66

pos
ecmdChipData, 18
ecmdChipTarget, 21
posState
ecmdChipTarget, 22
prevTarget
ecmdLooperData, 78
putArray
ecmdClientPerlapi.H, 157
putArrayMultiple
ecmdClientPerlapi.H, 158
putCfamRegister
ecmdClientPerlapi.H, 151
putFpr
ecmdClientPerlapi.H, 169
putFprMultiple
ecmdClientPerlapi.H, 169
putGpr
ecmdClientPerlapi.H, 166
putGprMultiple
ecmdClientPerlapi.H, 167
putLatch
ecmdClientPerlapi.H, 147
putMemDma
ecmdClientPerlapi.H, 174
putMemMemCtrl
ecmdClientPerlapi.H, 175
putMemProc
ecmdClientPerlapi.H, 173
putModuleVpdImage
ecmdClientPerlapi.H, 195

putModuleVpdKeyword
ecmdClientPerlapi.H, 194
putRing
ecmdClientPerlapi.H, 146
putRingWithModifier
ecmdClientPerlapi.H, 148
putScom
ecmdClientPerlapi.H, 149
putSlb
ecmdClientPerlapi.H, 171
putSlbMultiple
ecmdClientPerlapi.H, 171
putSpr
ecmdClientPerlapi.H, 164
putSprMultiple
ecmdClientPerlapi.H, 165
putSpy
ecmdClientPerlapi.H, 154
putSpyEnum
ecmdClientPerlapi.H, 154

QD_HDR_MAGIC
ecmdStructs.H, 218
queryErrorState
ecmdDataBuffer, 66
queryNumOfBuffers
ecmdDataBuffer, 65

rc
ecmdArrayEntry, 15
ecmdIndexEntry, 71
ecmdIndexVectorEntry, 72
ecmdLatchEntry, 76
ecmdNameEntry, 81
ecmdNameVectorEntry, 82
readAddressLength
ecmdArrayData, 14
readFile
ecmdDataBuffer, 64
readFileMultiple
ecmdDataBuffer, 64
readFileStream
ecmdDataBuffer, 65
reverse
ecmdDataBuffer, 43
ringName
ecmdLatchData, 73
ecmdLatchEntry, 75
ringNames
ecmdRingData, 87
rotateLeft
ecmdDataBuffer, 43
rotateRight
ecmdDataBuffer, 42

sendCmd
ecmdClientPerlapi.H, 150
setAnd
ecmdDataBuffer, 52
setBit
ecmdDataBuffer, 36
setBitLength
ecmdDataBuffer, 35
setByte
ecmdDataBuffer, 37
setByteLength
ecmdDataBuffer, 34
setCapacity
ecmdDataBuffer, 35
setDoubleWord
ecmdDataBuffer, 38
setHalfWord
ecmdDataBuffer, 38
setOr
ecmdDataBuffer, 49, 50
setWord
ecmdDataBuffer, 37
setWordLength
ecmdDataBuffer, 34
setXor
ecmdDataBuffer, 51
setXstate
ecmdDataBuffer, 61, 62
shareBuffer
ecmdDataBuffer, 65
shiftLeft
ecmdDataBuffer, 41
shiftLeftAndResize
ecmdDataBuffer, 42
shiftRight
ecmdDataBuffer, 41
shiftRightAndResize
ecmdDataBuffer, 42
shrinkBitLength
ecmdDataBuffer, 35
simaet
ecmdClientPerlapi.H, 176
simCallFusionCommand
ecmdClientPerlapi.H, 185
simcheckpoint
ecmdClientPerlapi.H, 177
simclock
ecmdClientPerlapi.H, 177
simecho
ecmdClientPerlapi.H, 177
simexit
ecmdClientPerlapi.H, 177
simEXPECTFAC
ecmdClientPerlapi.H, 178

simexpecttcfac
ecmdClientPerlapi.H, 178
simFusionRand32
ecmdClientPerlapi.H, 186
simgetcurrentcycle
ecmdClientPerlapi.H, 178
simGetDial
ecmdClientPerlapi.H, 187
simGetEnvironment
ecmdClientPerlapi.H, 188
simGETFAC
ecmdClientPerlapi.H, 179
simGETFACX
ecmdClientPerlapi.H, 179
simGetHierarchy
ecmdClientPerlapi.H, 184
simGetInFile
ecmdClientPerlapi.H, 187
simGetModelInfo
ecmdClientPerlapi.H, 188
simGetOutFile
ecmdClientPerlapi.H, 187
simgettcfac
ecmdClientPerlapi.H, 179
siminit
ecmdClientPerlapi.H, 180
simModelEc
ecmdChipData, 18
simOutputFusionMessage
ecmdClientPerlapi.H, 186
simPOLLFAC
ecmdClientPerlapi.H, 180
simpolltcfac
ecmdClientPerlapi.H, 181
simPutDial
ecmdClientPerlapi.H, 187
simPUTFAC
ecmdClientPerlapi.H, 181
simPUTFACX
ecmdClientPerlapi.H, 181
simputtcfac
ecmdClientPerlapi.H, 182
simrestart
ecmdClientPerlapi.H, 182
simSetFusionMessageFormat
ecmdClientPerlapi.H, 186
simSTKFAC
ecmdClientPerlapi.H, 182
simstktcfac
ecmdClientPerlapi.H, 183
simSUBCMD
ecmdClientPerlapi.H, 183
simtckinterval
ecmdClientPerlapi.H, 183
simUNSTICK
ecmdClientPerlapi.H, 184
simunsticktcfac
ecmdClientPerlapi.H, 184
slot
ecmdChipTarget, 21
SLOT_HDR_MAGIC
ecmdStructs.H, 218
slotData
ecmdNodeData, 83
slotId
ecmdSlotData, 91
slotState
ecmdChipTarget, 22
spyName
ecmdSpyData, 92
spyType
ecmdSpyData, 93
startClocks
ecmdClientPerlapi.H, 159
stopClocks
ecmdClientPerlapi.H, 159
supportsBroadsideLoad
ecmdRingData, 88

tags
ecmdMemoryEntry, 80
thread
ecmdChipTarget, 21
THREAD_HDR_MAGIC
ecmdStructs.H, 218
threadData
ecmdCoreData, 23
threadId
ecmdThreadData, 95
threadReplicated
ecmdProcRegisterInfo, 85
threadState
ecmdChipTarget, 22
totalEntries
ecmdProcRegisterInfo, 85
traceArrayName
ecmdTraceArrayData, 96

unflatten
ecmdDataBuffer, 54
unitId
ecmdCageData, 16
ecmdChipData, 18
ecmdChipTarget, 22
ecmdCoreData, 23
ecmdNodeData, 83
ecmdSlotData, 91
ecmdThreadData, 95

unitIdState
    ecmdChipTarget, 22
unitIdTargets
    ecmdLooperData, 78

width
    ecmdArrayData, 14
    ecmdTraceArrayData, 96
writeAddressLength
    ecmdArrayData, 14
writeBit
    ecmdDataBuffer, 37
writeFile
    ecmdDataBuffer, 63
writeFileMultiple
    ecmdDataBuffer, 63
writeFileStream
    ecmdDataBuffer, 64

zseClientPerlapi.H, 236
zseClientPerlapi.H
    zseInitExtension, 236
zseInitExtension
    zseClientPerlapi.H, 236