eCMD C/C++ Dll Version Development Reference Manual

Generated by Doxygen 1.3.5

Mon Oct 10 14:28:35 2005

Contents

1 eCMD C/C++ Dll Version Development Main Page **1**

1.1 Introduction 1

1.2 eCMD Core Include Files 1

1.3 Link objects 1

1.4 eCMD Extensions 2

1.5 DLL Version 3

1.6 The ecmdDataBuffer class 3

1.7 Examples 3

2 eCMD C/C++ Dll Version Development Hierarchical Index **5**

2.1 eCMD C/C++ Dll Version Development Class Hierarchy 5

3 eCMD C/C++ Dll Version Development Class Index **7**

3.1 eCMD C/C++ Dll Version Development Class List 7

4 eCMD C/C++ Dll Version Development File Index **9**

4.1 eCMD C/C++ Dll Version Development File List 9

5 eCMD C/C++ Dll Version Development Class Documentation **11**

5.1 cpcfSysInfo_t Struct Reference 11

5.2 croEclipzL2Fields Struct Reference 13

5.3 croEclipzPlusL2Fields Struct Reference 14

5.4 croL2Fields Struct Reference 15

5.5 ecmdArrayData Struct Reference 16

5.6 ecmdArrayEntry Struct Reference 18

5.7 ecmdCageData Struct Reference 19

5.8 ecmdChipData Struct Reference 20

5.9 ecmdChipTarget Struct Reference 23

5.10 ecmdCoreData Struct Reference 26

5.11 ecmdDataBuffer Class Reference 27

5.12 ecmdDataBufferImplementationHelper Class Reference 72
5.13 ecmdDllInfo Struct Reference 73
5.14 ecmdIndexEntry Struct Reference 75
5.15 ecmdIndexVectorEntry Struct Reference 76
5.16 ecmdLatchData Struct Reference 77
5.17 ecmdLatchEntry Struct Reference 79
5.18 ecmdLooperData Struct Reference 81
5.19 ecmdMemoryEntry Struct Reference 84
5.20 ecmdNameEntry Struct Reference 85
5.21 ecmdNameVectorEntry Struct Reference 86
5.22 ecmdNodeData Struct Reference 87
5.23 ecmdOptimizableDataBuffer Class Reference 88
5.24 ecmdProcRegisterInfo Struct Reference 89
5.25 ecmdQueryData Struct Reference 90
5.26 ecmdRingData Struct Reference 91
5.27 ecmdScomData Struct Reference 93
5.28 ecmdSimModelInfo Struct Reference 94
5.29 ecmdSlotData Struct Reference 95
5.30 ecmdSpyData Struct Reference 96
5.31 ecmdSpyGroupData Struct Reference 98
5.32 ecmdThreadData Struct Reference 99
5.33 ecmdTraceArrayData Struct Reference 100
5.34 gipXlateVariables Struct Reference 102

6 eCMD C/C++ Dll Version Development File Documentation 105

6.1 cipClientCapi.H File Reference 105
6.2 cipStructs.H File Reference 117
6.3 cmdClientCapi.H File Reference 118
6.4 cmdStructs.H File Reference 120
6.5 croClientCapi.H File Reference 121
6.6 croStructs.H File Reference 132
6.7 ecmdClientCapi.H File Reference 133
6.8 ecmdDataBuffer.H File Reference 212
6.9 ecmdReturnCodes.H File Reference 218
6.10 ecmdSharedUtils.H File Reference 231
6.11 ecmdStructs.H File Reference 237
6.12 ecmdUtils.H File Reference 250

6.13 gipClientCapi.H File Reference 256
6.14 gipStructs.H File Reference 263
6.15 zseClientCapi.H File Reference 265
6.16 zseStructs.H File Reference 266

Chapter 1

eCMD C/C++ Dll Version Development Main Page

1.1 Introduction

Common Hardware Access Programming Interface (eCMD)
This is the documentation of the eCMD C/C++ Programming Api

1.2 eCMD Core Include Files

To compile client code to use the C++ API, the following header files are required:

- **ecmdClientCapi.H**(p. 133)
- **ecmdDataBuffer.H**(p. 212)
- **ecmdStructs.H**(p. 237)
- **ecmdReturnCodes.H**(p. 218)
- **ecmdUtils.H**(p. 250)
- **ecmdSharedUtils.H**(p. 231)

1.3 Link objects

To link the client code on AIX, the following is required:

- ecmdClientCapi_aix.a
- libecmd_aix.so
- xlC v6.0.0.8

To create Linux x86 binaries, the following is required:

- ecmdClientCapi_x86.a
- libecmd_x86.so
- g++ 3.2.3

1.4 eCMD Extensions

These are extensions to the core eCMD interface, not all eCMD Plugins support these extensions.

To use an eCMD extension you will need to link in the appropriate library, see the example Makefiles under 'Use eCMD' for help.

1.4.1 CMD Command line Extension

This extensions provides interfaces to call command line functions and have formatted data displayed to stdout or returned to the caller. It supports command from the core command line and also all extensions.

Include files :

- **cmdClientCapi.H**(p. 118)
- **cmdStructs.H**(p. 120)
- Library : cmdClientCapi_aix.a / cmdClientCapi_x86.a

1.4.2 CIP (Cronus/IP) Extension

This extensions provides interfaces to start/stop processor instructions and breakpoint handling.

Include files :

- **cipClientCapi.H**(p. 105)
- **cipStructs.H**(p. 117)
- Library : cipClientCapi_aix.a / cipClientCapi_x86.a

1.4.3 GIP GFW IP-Series Extension

This extensions provides IP Series GFW only interfaces.

Include files :

- **gipClientCapi.H**(p. 256)
- **gipStructs.H**(p. 263)
- Library : gipClientCapi_aix.a / gipClientCapi_x86.a

1.4.4 Cronus Extension

This extensions provides Cronus only interfaces.

Include files :

- **croClientCapi.H**(p. 121)
- **croStructs.H**(p. 132)
- Library : croClientCapi_aix.a / croClientCapi_x86.a

1.4.5 Z Series Extension

This extensions provides Z-Series only interfaces.

Include files :

- **zseClientCapi.H**(p. 265)
- **zseStructs.H**(p. 266)
- Library : zseClientCapi_aix.a / zseClientCapi_x86.a

1.5 DLL Version

The eCMD Capi client code is built with a ECMD_CAPI_VERSION that gets passed into the DLL with the initDll function. If the version passed in does not match the version compiled into the DLL, the init will fail. The programmer needs to get a new copy of the .a archive and rebuild there client to correct this problem.

1.6 The ecmdDataBuffer class

Data is passed between the client and the DLL with the **ecmdDataBuffer**(p. 27) class. The **ecmdDataBuffer**(p. 27) object is linked on both the client side and the DLL side.

The **ecmdDataBuffer**(p. 27) maintains data both as unsigned integers and as a character string. The class contains methods for accessing and modifying data as well as converting data to strings (e.g. hex, left-aligned). The **ecmdDataBuffer**(p. 27) class allocates the memory for the conversion-to-string routines and returns a char* pointer to the memory. The client should allocate its own memory and do a strcpy if the string is to be preserved upon the next **ecmdDataBuffer**(p. 27) conversion-to-string call.

1.7 Examples

For Makefile and Client examples please go to 'Use eCMD' on the eCMD web page : `http://rhea.rchland.ibm.com/eCMD/`

Chapter 2

eCMD C/C++ Dll Version Development Hierarchical Index

2.1 eCMD C/C++ Dll Version Development Class Hierarchy

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

cpcfSysInfo_t . . . 11
croEclipzL2Fields . . . 13
croEclipzPlusL2Fields . . . 14
croL2Fields . . . 15
ecmdArrayData . . . 16
ecmdArrayEntry . . . 18
ecmdCageData . . . 19
ecmdChipData . . . 20
ecmdChipTarget . . . 23
ecmdCoreData . . . 26
ecmdDataBuffer . . . 27
    ecmdOptimizableDataBuffer . . . 88
ecmdDataBufferImplementationHelper . . . 72
ecmdDllInfo . . . 73
ecmdIndexEntry . . . 75
ecmdIndexVectorEntry . . . 76
ecmdLatchData . . . 77
ecmdLatchEntry . . . 79
ecmdLooperData . . . 81
ecmdMemoryEntry . . . 84
ecmdNameEntry . . . 85
ecmdNameVectorEntry . . . 86
ecmdNodeData . . . 87
ecmdProcRegisterInfo . . . 89
ecmdQueryData . . . 90
ecmdRingData . . . 91
ecmdScomData . . . 93
ecmdSimModelInfo . . . 94
ecmdSlotData . . . 95

ecmdSpyData 96
ecmdSpyGroupData 98
ecmdThreadData 99
ecmdTraceArrayData 100
gipXlateVariables 102

Chapter 3

eCMD C/C++ Dll Version Development Class Index

3.1 eCMD C/C++ Dll Version Development Class List

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

cpcfSysInfo_t 11
croEclipzL2Fields (These are used by L2 Display/Alter functions) 13
croEclipzPlusL2Fields 14
croL2Fields 15
ecmdArrayData (Used for the ecmdQueryArray function to return array info) 16
ecmdArrayEntry (Used by the getArrayMultiple function to pass data) 18
ecmdCageData (Used for the ecmdQueryConfig function to return cage data) 19
ecmdChipData (Used for the ecmdQueryConfig function to return chip data) 20
ecmdChipTarget (Structure used to designate which cec object/chip you would like the function to operate on) 23
ecmdCoreData (Used for the ecmdQueryConfig function to return core data) 26
ecmdDataBuffer (Provides a means to handle data from the eCMD C API) 27
ecmdDataBufferImplementationHelper (This is used to help low-level implementation of the **ecmdDataBuffer**(p. 27), this CAN NOT be used by any eCMD client or data corruption will occur) 72
ecmdDllInfo (This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with) 73
ecmdIndexEntry (Used by get/put Gpr/Fpr Multiple function to pass data) 75
ecmdIndexVectorEntry (Used by ???? function to pass data) 76
ecmdLatchData (Used for the ecmdQueryLatch function to return latch info) 77
ecmdLatchEntry (Used by getlatch function to return data) 79
ecmdLooperData (Used internally by ecmdConfigLooper to store looping state information) 81
ecmdMemoryEntry (Used by ecmdReadDcard) 84
ecmdNameEntry (Used by get/putSprMultiple function to pass data) 85
ecmdNameVectorEntry (Used by getTraceArrayMultiple function to pass data) . . 86
ecmdNodeData (Used for the ecmdQueryConfig function to return node data) 87
ecmdOptimizableDataBuffer 88
ecmdProcRegisterInfo (Used by ecmdQueryProcRegisterInfo function to return data about a Architected register) 89
ecmdQueryData (Used by the ecmdQueryConfig function to return data) 90

ecmdRingData (Used for the ecmdQueryRing function to return ring info) 91
ecmdScomData (Used for the ecmdQueryScom function to return scom info) 93
ecmdSimModelInfo (Used by simGetModelInfo) 94
ecmdSlotData (Used for the ecmdQueryConfig function to return slot data) 95
ecmdSpyData (Used for the ecmdQuerySpy function to return spy info) 96
ecmdSpyGroupData (Used by get/putspy function to create the return data from a group) 98
ecmdThreadData (Used for the ecmdQueryConfig function to return thread data) . 99
ecmdTraceArrayData (Used for the ecmdQueryTraceArray function to return trace array info) 100
gipXlateVariables (Struct used for Translate variables in Mainstore Memory D/A and Breakpoint Interfaces) 102

Chapter 4

eCMD C/C++ Dll Version Development File Index

4.1 eCMD C/C++ Dll Version Development File List

Here is a list of all files with brief descriptions:

cipClientCapi.H (Cronus & IP eCMD Extension) 105
cipStructs.H (Cronus & IP eCMD Extension Structures) 117
cmdClientCapi.H (ECMD Command line extension) 118
cmdStructs.H (ECMD Command line Extension Structures) 120
croClientCapi.H (Cronus eCMD Extension) 121
croStructs.H (Cronus eCMD Extension Structures) 132
ecmdClientCapi.H (ECMD C/C++ Client Interface) 133
ecmdDataBuffer.H (Provides a means to handle data from the eCMD C API) 212
ecmdReturnCodes.H (All Return Codes for the eCmd Capi) 218
ecmdSharedUtils.H (Useful functions for use throughout the ecmd C API and Plugin) 231
ecmdStructs.H (All the Structures required for the eCMD Capi) 237
ecmdUtils.H (Useful functions for use throughout the ecmd C API) 250
gipClientCapi.H (GFW IP Series eCMD Extension) 256
gipStructs.H (GFW IP Series eCMD Extension Structures) 263
zseClientCapi.H (Z Series eCMD Extension) 265
zseStructs.H (Z Series eCMD Extension Structures) 266

Chapter 5

eCMD C/C++ Dll Version Development Class Documentation

5.1 cpcfSysInfo_t Struct Reference

`#include <gipStructs.H>`

Public Member Functions

- void **flatten** ()
- void **unflatten** ()

Public Attributes

- uint64_t **procExist**
- uint64_t **procFunct**
- char **boxType**
- char **reservedA**
- char **clkDomState** [2]
- uint64_t **puInstValid**
- uint64_t **puInstState**
- char **actThread** [64]
- char **reserved0** [8]
- char **iSeries**
- char **olcRcvd**
- char **errorState**
- char **reserved1** [9]
- uint32_t **cpuCtlsVersion**
- uint32_t **primaryProc**
- uint64_t **brkPtHit** [2]
- char **procType**
- char **iopType**
- char **reserved2** [10]

5.1.1 Member Function Documentation

5.1.1.1 void cpcfSysInfo_t::flatten ()

5.1.1.2 void cpcfSysInfo_t::unflatten ()

5.1.2 Member Data Documentation

5.1.2.1 uint64_t cpcfSysInfo_t::procExist

5.1.2.2 uint64_t cpcfSysInfo_t::procFunct

5.1.2.3 char cpcfSysInfo_t::boxType

5.1.2.4 char cpcfSysInfo_t::reservedA

5.1.2.5 char cpcfSysInfo_t::clkDomState[2]

5.1.2.6 uint64_t cpcfSysInfo_t::puInstValid

5.1.2.7 uint64_t cpcfSysInfo_t::puInstState

5.1.2.8 char cpcfSysInfo_t::actThread[64]

5.1.2.9 char cpcfSysInfo_t::reserved0[8]

5.1.2.10 char cpcfSysInfo_t::iSeries

5.1.2.11 char cpcfSysInfo_t::olcRcvd

5.1.2.12 char cpcfSysInfo_t::errorState

5.1.2.13 char cpcfSysInfo_t::reserved1[9]

5.1.2.14 uint32_t cpcfSysInfo_t::cpuCtlsVersion

5.1.2.15 uint32_t cpcfSysInfo_t::primaryProc

5.1.2.16 uint64_t cpcfSysInfo_t::brkPtHit[2]

5.1.2.17 char cpcfSysInfo_t::procType

5.1.2.18 char cpcfSysInfo_t::iopType

5.1.2.19 char cpcfSysInfo_t::reserved2[10]

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

- **gipStructs.H**

5.2 croEclipzL2Fields Struct Reference

These are used by L2 Display/Alter functions.

`#include <croStructs.H>`

Public Attributes

- char **i_slice**
- uint32_t **i_set**
- uint64_t **i_endaddress**
- uint32_t **i_classbits**
- uint32_t **i_startcongclass**
- uint32_t **i_endcongclass**
- uint32_t **i_mesibits**
- uint32_t **i_tagbits**

5.2.1 Detailed Description

These are used by L2 Display/Alter functions.

5.2.2 Member Data Documentation

5.2.2.1 char croEclipzL2Fields::i_slice

5.2.2.2 uint32_t croEclipzL2Fields::i_set

5.2.2.3 uint64_t croEclipzL2Fields::i_endaddress

5.2.2.4 uint32_t croEclipzL2Fields::i_classbits

5.2.2.5 uint32_t croEclipzL2Fields::i_startcongclass

5.2.2.6 uint32_t croEclipzL2Fields::i_endcongclass

5.2.2.7 uint32_t croEclipzL2Fields::i_mesibits

5.2.2.8 uint32_t croEclipzL2Fields::i_tagbits

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

- **croStructs.H**

5.3 croEclipzPlusL2Fields Struct Reference

`#include <croStructs.H>`

Public Attributes

- uint32_t **undefined**

5.3.1 Member Data Documentation

5.3.1.1 uint32_t croEclipzPlusL2Fields::undefined

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

- **croStructs.H**

5.4 croL2Fields Struct Reference

`#include <croStructs.H>`

Public Attributes

- **ecmdDllProduct_t dllProduct**

5.4.1 Member Data Documentation

5.4.1.1 ecmdDllProduct_t croL2Fields::dllProduct

5.4.1.2 struct croEclipzL2Fields croL2Fields::eclipz

5.4.1.3 struct croEclipzPlusL2Fields croL2Fields::eclipzplus

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

- **croStructs.H**

5.5 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

5.5.1 Detailed Description

Used for the ecmdQueryArray function to return array info.

5.5.2 Member Data Documentation

5.5.2.1 std::string ecmdArrayData::arrayName

(Detail: Low) Names used to reference this array

5.5.2.2 int ecmdArrayData::readAddressLength

(Detail: Low) Bit length of read address

5.5.2.3 int ecmdArrayData::writeAddressLength

(Detail: Low) Bit length of write address

5.5.2.4 int ecmdArrayData::length

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

5.5.2.5 int ecmdArrayData::width

(Detail: Low) Bit width of array entry

5.5.2.6 bool ecmdArrayData::isCoreRelated

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

5.5.2.7 std::string ecmdArrayData::clockDomain

(Detail: High) Clock domain this array belongs to

5.5.2.8 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**

5.6 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.

5.6.1 Detailed Description

Used by the getArrayMultiple function to pass data.

5.6.2 Member Data Documentation

5.6.2.1 ecmdDataBuffer ecmdArrayEntry::address

Array address/element to access.

5.6.2.2 ecmdDataBuffer ecmdArrayEntry::buffer

Array data from address.

5.6.2.3 uint32_t ecmdArrayEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.7 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

5.7.1 Detailed Description

Used for the ecmdQueryConfig function to return cage data.

Operators Supported : <

5.7.2 Member Data Documentation

5.7.2.1 uint32_t ecmdCageData::cageId

(Detail: Low) Cage number of this entry

5.7.2.2 uint32_t ecmdCageData::unitId

(Detail: High) Unit Id of this entry

5.7.2.3 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**

5.8 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

5.8.1 Detailed Description

Used for the ecmdQueryConfig function to return chip data.

Operators Supported : <

5.8.2 Member Data Documentation

5.8.2.1 std::string ecmdChipData::chipType

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

5.8.2.2 std::string ecmdChipData::chipShortType

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

5.8.2.3 std::string ecmdChipData::chipCommonType

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

5.8.2.4 uint32_t ecmdChipData::pos

(Detail: Low) Position of this entry

5.8.2.5 uint32_t ecmdChipData::unitId

(Detail: High) Unit Id of this entry

5.8.2.6 uint8_t ecmdChipData::numProcCores

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

5.8.2.7 uint32_t ecmdChipData::chipEc

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

5.8.2.8 uint32_t ecmdChipData::simModelEc

(Detail: High) Model EC level of this chip

5.8.2.9 ecmdChipInterfaceType_t ecmdChipData::interfaceType

(Detail: High) Interface Macro used by the chip

5.8.2.10 uint32_t ecmdChipData::chipFlags

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

5.8.2.11 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**

5.9 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

5.9.1 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

5.9.2 Member Data Documentation

5.9.2.1 uint32_t ecmdChipTarget::cage

cage that contains node with chip

5.9.2.2 uint32_t ecmdChipTarget::node

node that contains chip

5.9.2.3 uint32_t ecmdChipTarget::slot

Card Slot/Fru to target.

5.9.2.4 std::string ecmdChipTarget::chipType

name of chip to access , either actual or common name

5.9.2.5 uint32_t ecmdChipTarget::pos

position of chip within node

5.9.2.6 uint8_t ecmdChipTarget::core

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

5.9.2.7 uint8_t ecmdChipTarget::thread

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

5.9.2.8 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.

5.9.2.9 ecmdChipTargetState_t ecmdChipTarget::cageState

cage field state

5.9.2.10 ecmdChipTargetState_t ecmdChipTarget::nodeState

node field state

5.9.2.11 ecmdChipTargetState_t ecmdChipTarget::slotState

slot field state

5.9.2.12 ecmdChipTargetState_t ecmdChipTarget::chipTypeState

chipType field state

5.9.2.13 ecmdChipTargetState_t ecmdChipTarget::posState

pos field state

5.9.2.14 ecmdChipTargetState_t ecmdChipTarget::coreState

core field state

5.9.2.15 ecmdChipTargetState_t ecmdChipTarget::threadState

thread field state

5.9.2.16 ecmdChipTargetState_t ecmdChipTarget::unitIdState

unitId field state

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

- **ecmdStructs.H**

5.10 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

5.10.1 Detailed Description

Used for the ecmdQueryConfig function to return core data.

Operators Supported : <

5.10.2 Member Data Documentation

5.10.2.1 uint8_t ecmdCoreData::coreId

(Detail: Low) core number of this entry

5.10.2.2 uint8_t ecmdCoreData::numProcThreads

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

5.10.2.3 uint32_t ecmdCoreData::unitId

(Detail: High) Unit Id of this entry

5.10.2.4 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:

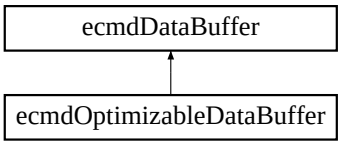
- **ecmdStructs.H**

5.11 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.

- **ecmdDataBuffer** & **operator=** (const **ecmdDataBuffer** &i_master)
  Copy Constructor.

- 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**

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ecmdDataBuffer::ecmdDataBuffer ()

Default Constructor.

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

5.11.2.2 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

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

Copy Constructor.

Parameters:
other Buffer to copy

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

Default Destructor.

5.11.3 Member Function Documentation

5.11.3.1 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

5.11.3.2 uint32_t ecmdDataBuffer::getWordLength () const

Return the length of the buffer in words.

Return values:
Buffer length in words rounded up

5.11.3.3 uint32_t ecmdDataBuffer::getByteLength () const

Return the length of the buffer in bytes.

Return values:
Buffer length in bytes rounded up

5.11.3.4 uint32_t ecmdDataBuffer::getBitLength () const

Return the length of the buffer in bits.

Return values:
Buffer length in bits

5.11.3.5 uint32_t ecmdDataBuffer::getCapacity () const

Return the actual capacity of the internal buffer in words.

Return values:
Actual capacity in words of internal buffer

5.11.3.6 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

5.11.3.7 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

5.11.3.8 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

5.11.3.9 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

5.11.3.10 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

5.11.3.11 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

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

Returns value of iv_BufferOptimizable.

5.11.3.13 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

5.11.3.14 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

5.11.3.15 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

5.11.3.16 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

5.11.3.17 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

5.11.3.18 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

5.11.3.19 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

5.11.3.20 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

5.11.3.21 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

5.11.3.22 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

5.11.3.23 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

5.11.3.24 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

5.11.3.25 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

5.11.3.26 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

5.11.3.27 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

5.11.3.28 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

5.11.3.29 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

5.11.3.30 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

5.11.3.31 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

5.11.3.32 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

5.11.3.33 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

5.11.3.34 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

5.11.3.35 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

5.11.3.36 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

5.11.3.37 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

5.11.3.38 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

5.11.3.39 uint32_t ecmdDataBuffer::flushTo0 ()

Clear entire buffer to 0's.

Return values:
ECMD_DBUF_SUCCESS on success

5.11.3.40 uint32_t ecmdDataBuffer::flushTo1 ()

Set entire buffer to 1's.

Return values:
ECMD_DBUF_SUCCESS on success

5.11.3.41 uint32_t ecmdDataBuffer::invert ()

Invert entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

5.11.3.42 uint32_t ecmdDataBuffer::reverse ()

Bit reverse entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

5.11.3.43 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

5.11.3.44 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

5.11.3.45 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

5.11.3.46 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

5.11.3.47 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

5.11.3.48 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

5.11.3.49 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

5.11.3.50 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

5.11.3.51 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

5.11.3.52 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

5.11.3.53 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

5.11.3.54 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

5.11.3.55 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

5.11.3.56 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

5.11.3.57 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

5.11.3.58 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

5.11.3.59 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

5.11.3.60 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

5.11.3.61 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

5.11.3.62 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

5.11.3.63 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

5.11.3.64 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

5.11.3.65 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

5.11.3.66 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

5.11.3.67 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

5.11.3.68 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

5.11.3.69 ecmdDataBuffer& ecmdDataBuffer::operator= (const ecmdDataBuffer & *i_master*)

Copy Constructor.

Parameters:

i_master DataBuffer to copy from

Postcondition:

this DataBuffer is allocated, is an exact duplicate of the other

5.11.3.70 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

5.11.3.71 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

5.11.3.72 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

5.11.3.73 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

5.11.3.74 uint32_t ecmdDataBuffer::flattenSize (void) const

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

Return values:

Number of bytes needed

5.11.3.75 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

5.11.3.76 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

5.11.3.77 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

5.11.3.78 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

5.11.3.79 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

5.11.3.80 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

5.11.3.81 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

5.11.3.82 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

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

Return entire buffer as a hex left aligned char string.

Return values:

String containing requested data

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

Return entire buffer as a hex right aligned char string.

Return values:

String containing requested data

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

Return entire buffer as a binary char string.

Return values:

String containing requested data

5.11.3.86 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

5.11.3.87 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

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

Retrieve entire Xstate Data buffer.

Return values:
String containing requested data

5.11.3.89 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

5.11.3.90 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

5.11.3.91 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

5.11.3.92 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

5.11.3.93 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

5.11.3.94 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

5.11.3.95 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

5.11.3.96 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

5.11.3.97 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

5.11.3.98 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

5.11.3.99 bool ecmdDataBuffer::hasXstate () const

Check Entire buffer for any X-state values.

Return values:
true if xstate found false if none

5.11.3.100 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

5.11.3.101 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

5.11.3.102 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

5.11.3.103 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

5.11.3.104 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

5.11.3.105 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

5.11.3.106 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

5.11.3.107 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

5.11.3.108 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

5.11.3.109 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

5.11.3.110 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

5.11.3.111 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

5.11.3.112 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

5.11.3.113 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

5.11.3.114 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

5.11.3.115 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

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

Overload the == operator.

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

Overload the != operator.

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

Overload the & operator.

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

Overload the | operator.

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

5.11.4 Friends And Related Function Documentation

5.11.4.1 friend class ecmdDataBufferImplementationHelper [friend]

5.11.5 Member Data Documentation

5.11.5.1 uint32_t ecmdDataBuffer::iv_Capacity [protected]

Actual buffer capacity - always >= iv_NumWords.

5.11.5.2 uint32_t ecmdDataBuffer::iv_NumWords [protected]

Specified buffer size rounded to next word.

5.11.5.3 uint32_t ecmdDataBuffer::iv_NumBits [protected]

Specified buffer size in bits.

5.11.5.4 uint32_t* ecmdDataBuffer::iv_Data [protected]

Pointer to buffer inside iv_RealData.

5.11.5.5 uint32_t* ecmdDataBuffer::iv_RealData [protected]

Real buffer - with header and tail.

5.11.5.6 bool ecmdDataBuffer::iv_UserOwned [protected]

Whether or not this buffer owns the data.

5.11.5.7 bool ecmdDataBuffer::iv_BufferOptimizable [protected]

Whether or not this is an optimizable buffer.

5.11.5.8 char* ecmdDataBuffer::iv_DataStr [protected]

5.11.5.9 bool ecmdDataBuffer::iv_XstateEnabled [protected]

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

- **ecmdDataBuffer.H**

5.12 ecmdDataBufferImplementationHelper Class Reference

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 27), 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)

5.12.1 Detailed Description

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 27), this CAN NOT be used by any eCMD client or data corruption will occur.

5.12.2 Member Function Documentation

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

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

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

- **ecmdDataBuffer.H**

5.13 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.

5.13.1 Detailed Description

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

5.13.2 Member Data Documentation

5.13.2.1 ecmdDllType_t ecmdDllInfo::dllType

Dll instance type running.

5.13.2.2 ecmdDllProduct_t ecmdDllInfo::dllProduct

Dll product supported.

5.13.2.3 std::string ecmdDllInfo::dllProductType

Dll product type currently configured.

5.13.2.4 ecmdDllEnv_t ecmdDllInfo::dllEnv

Dll environment (Simulation vs Hardware).

5.13.2.5 std::string ecmdDllInfo::dllBuildDate

Date the Dll was built.

5.13.2.6 std::string ecmdDllInfo::dllCapiVersion

should be set to ECMD_CAPI_VERSION

5.13.2.7 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**

5.14 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.

5.14.1 Detailed Description

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

5.14.2 Member Data Documentation

5.14.2.1 int ecmdIndexEntry::index

Index of entry.

5.14.2.2 ecmdDataBuffer ecmdIndexEntry::buffer

Data to/from entry.

5.14.2.3 uint32_t ecmdIndexEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.15 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.

5.15.1 Detailed Description

Used by ???? function to pass data.

5.15.2 Member Data Documentation

5.15.2.1 int ecmdIndexVectorEntry::index

Index of entry.

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

Vector of data to/from entry.

5.15.2.3 uint32_t ecmdIndexVectorEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.16 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

5.16.1 Detailed Description

Used for the ecmdQueryLatch function to return latch info.

5.16.2 Member Data Documentation

5.16.2.1 std::string ecmdLatchData::latchName

(Detail: Low) Latch Name

5.16.2.2 std::string ecmdLatchData::ringName

(Detail: Low) Ring that this latch belongs to

5.16.2.3 int ecmdLatchData::bitLength

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

5.16.2.4 bool ecmdLatchData::isCoreRelated

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

5.16.2.5 std::string ecmdLatchData::clockDomain

(Detail: High) Clock domain this latch belongs to

5.16.2.6 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**

5.17 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.

5.17.1 Detailed Description

Used by getlatch function to return data.

5.17.2 Member Data Documentation

5.17.2.1 std::string ecmdLatchEntry::latchName

Latch name of entry.

5.17.2.2 std::string ecmdLatchEntry::ringName

Ring that latch came from.

5.17.2.3 ecmdDataBuffer ecmdLatchEntry::buffer

Latch data.

5.17.2.4 int ecmdLatchEntry::latchStartBit

Start bit of data inside latch.

5.17.2.5 int ecmdLatchEntry::latchEndBit

End bit of data inside latch.

5.17.2.6 uint32_t ecmdLatchEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.18 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.

5.18.1 Detailed Description

Used internally by ecmdConfigLooper to store looping state information.

5.18.2 Member Data Documentation

5.18.2.1 bool ecmdLooperData::ecmdLooperInitFlag

Is fresh ?

5.18.2.2 bool ecmdLooperData::ecmdUseUnitid

This looper is looping on unitid targets not config data.

5.18.2.3 ecmdConfigLoopMode_t ecmdLooperData::ecmdLoopMode

Is this a variable depth or static loop?

5.18.2.4 ecmdQueryData ecmdLooperData::ecmdSystemConfigData

Config data queried from the system.

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

Pointer to current Cage.

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

Pointer to current Node.

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

Pointer to current Slot.

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

Pointer to current Chip.

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

Pointer to current Core.

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

Pointer to current Thread.

5.18.2.11 ecmdChipTarget ecmdLooperData::prevTarget

Pointer to previous target.

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

List of targets if looping on a unitid.

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

Pointer to current unitid target.

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

- **ecmdUtils.H**

5.19 ecmdMemoryEntry Struct Reference

Used by ecmdReadDcard.

`#include <ecmdStructs.H>`

Public Attributes

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

5.19.1 Detailed Description

Used by ecmdReadDcard.

5.19.2 Member Data Documentation

5.19.2.1 uint64_t ecmdMemoryEntry::address

5.19.2.2 ecmdDataBuffer ecmdMemoryEntry::data

5.19.2.3 ecmdDataBuffer ecmdMemoryEntry::tags

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

- **ecmdStructs.H**

5.20 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.

5.20.1 Detailed Description

Used by get/putSprMultiple function to pass data.

5.20.2 Member Data Documentation

5.20.2.1 std::string ecmdNameEntry::name

Name of entry.

5.20.2.2 ecmdDataBuffer ecmdNameEntry::buffer

Data to/from entry.

5.20.2.3 uint32_t ecmdNameEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.21 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.

5.21.1 Detailed Description

Used by getTraceArrayMultiple function to pass data.

5.21.2 Member Data Documentation

5.21.2.1 std::string ecmdNameVectorEntry::name

Name of entry.

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

Vector of data to/from entry.

5.21.2.3 uint32_t ecmdNameVectorEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

5.22 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

5.22.1 Detailed Description

Used for the ecmdQueryConfig function to return node data.

Operators Supported : <

5.22.2 Member Data Documentation

5.22.2.1 uint32_t ecmdNodeData::nodeId

(Detail: Low) Node number of this entry

5.22.2.2 uint32_t ecmdNodeData::unitId

(Detail: High) Unit Id of this entry

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

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

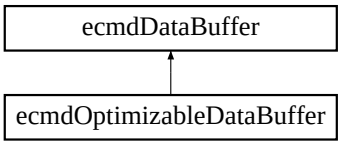
The documentation for this struct was generated from the following file:

- **ecmdStructs.H**

5.23 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.

5.23.1 Constructor & Destructor Documentation

5.23.1.1 ecmdOptimizableDataBuffer::ecmdOptimizableDataBuffer ()

Default constructor for ecmdOptimizableDataBuffer class.

5.23.1.2 ecmdOptimizableDataBuffer::ecmdOptimizableDataBuffer (uint32_t *numBits*)

Constructor with bit length specified.

5.23.1.3 ecmdOptimizableDataBuffer::~ecmdOptimizableDataBuffer () `[inline]`

Destructor for ecmdOptimizableDataBuffer class.

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

- **ecmdDataBuffer.H**

5.24 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.

5.24.1 Detailed Description

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

5.24.2 Member Data Documentation

5.24.2.1 int ecmdProcRegisterInfo::bitLength

Bit length of each entry.

5.24.2.2 int ecmdProcRegisterInfo::totalEntries

Total number of entries available.

5.24.2.3 bool ecmdProcRegisterInfo::threadReplicated

Register is replicated for each thread.

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

- **ecmdStructs.H**

5.25 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

5.25.1 Detailed Description

Used by the ecmdQueryConfig function to return data.

5.25.2 Member Data Documentation

5.25.2.1 ecmdQueryDetail_t ecmdQueryData::detailLevel

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

5.25.2.2 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**

5.26 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

5.26.1 Detailed Description

Used for the ecmdQueryRing function to return ring info.

5.26.2 Member Data Documentation

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

(Detail: Low) Names used to reference this ring

5.26.2.2 uint32_t ecmdRingData::address

(Detail: Low) Address modifier

5.26.2.3 int ecmdRingData::bitLength

(Detail: Low) length of ring

5.26.2.4 bool ecmdRingData::hasInversionMask

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

5.26.2.5 bool ecmdRingData::supportsBroadsideLoad

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

5.26.2.6 bool ecmdRingData::isCheckable

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

5.26.2.7 bool ecmdRingData::isCoreRelated

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

5.26.2.8 std::string ecmdRingData::clockDomain

(Detail: High) Clock domain this ring belongs to

5.26.2.9 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**

5.27 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

5.27.1 Detailed Description

Used for the ecmdQueryScom function to return scom info.

5.27.2 Member Data Documentation

5.27.2.1 uint32_t ecmdScomData::address

(Detail: Low) Scom Address

5.27.2.2 bool ecmdScomData::isCoreRelated

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

5.27.2.3 std::string ecmdScomData::clockDomain

(Detail: High) Clock domain this scom belongs to

5.27.2.4 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**

5.28 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

5.28.1 Detailed Description

Used by simGetModelInfo.

5.28.2 Member Data Documentation

5.28.2.1 char ecmdSimModelInfo::modelname[255]

5.28.2.2 char ecmdSimModelInfo::modeldate[255]

5.28.2.3 char ecmdSimModelInfo::modeltime[255]

5.28.2.4 char ecmdSimModelInfo::multivalue

!=0 multivalue, ==0 2-state

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

- **ecmdStructs.H**

5.29 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

5.29.1 Detailed Description

Used for the ecmdQueryConfig function to return slot data.

Operators Supported : <

5.29.2 Member Data Documentation

5.29.2.1 uint32_t ecmdSlotData::slotId

(Detail: Low) Slot number of this entry

5.29.2.2 uint32_t ecmdSlotData::unitId

(Detail: High) Unit Id of this entry

5.29.2.3 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**

5.30 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.

5.30.1 Detailed Description

Used for the ecmdQuerySpy function to return spy info.

5.30.2 Member Data Documentation

5.30.2.1 std::string ecmdSpyData::spyName

Names used to reference this spy.

5.30.2.2 int ecmdSpyData::bitLength

length of spy

5.30.2.3 ecmdSpyType_t ecmdSpyData::spyType

Type of spy.

5.30.2.4 bool ecmdSpyData::isEccChecked

This spy affects some ECC groupings.

5.30.2.5 bool ecmdSpyData::isEnumerated

This spy has enumerated values.

5.30.2.6 bool ecmdSpyData::isCoreRelated

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

5.30.2.7 std::string ecmdSpyData::clockDomain

Clock domain this spy belongs to.

5.30.2.8 ecmdClockState_t ecmdSpyData::clockState

Required clock state to access this spy.

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

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

5.30.2.10 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**

5.31 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.

5.31.1 Detailed Description

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

5.31.2 Member Data Documentation

5.31.2.1 ecmdDataBuffer ecmdSpyGroupData::extractBuffer

The data read from the ring buffer.

5.31.2.2 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**

5.32 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

5.32.1 Detailed Description

Used for the ecmdQueryConfig function to return thread data.

Operators Supported : <

5.32.2 Member Data Documentation

5.32.2.1 uint8_t ecmdThreadData::threadId

(Detail: Low) Thread number of this entry

5.32.2.2 uint32_t ecmdThreadData::unitId

(Detail: High) Unit Id of this entry

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

- **ecmdStructs.H**

5.33 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

5.33.1 Detailed Description

Used for the ecmdQueryTraceArray function to return trace array info.

5.33.2 Member Data Documentation

5.33.2.1 std::string ecmdTraceArrayData::traceArrayName

(Detail: Low) Name of Trace array

5.33.2.2 int ecmdTraceArrayData::length

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

5.33.2.3 int ecmdTraceArrayData::width

(Detail: Low) Bit width of trace array entry

5.33.2.4 bool ecmdTraceArrayData::isCoreRelated

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

5.33.2.5 std::string ecmdTraceArrayData::clockDomain

(Detail: High) Clock domain this array belongs to

5.33.2.6 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**

5.34 gipXlateVariables Struct Reference

Struct used for Translate variables in Mainstore Memory D/A and Breakpoint Interfaces.

`#include <gipStructs.H>`

Public Attributes

- bool **tagsActive**

  1 = Tags Active Mode; 0 = Tags Inactive Mode

- bool **littleEndian**

  1 = Little Endian ; 0 = Big Endian

- bool **mode32bit**

  1 = 32 bit mode; 0 = 64 bit mode

- bool **writeECC**

  1 = write ECC using ECC buffer as input; 0 = have ECC calculated

- bool **manualXlateFlag**

  1 = Manual Translation Needed; 0 = Manual Translation Not Needed

- **gipMainstoreAddrType_t addrType**

  Type of Mainstore Address being used.

- uint32_t **partitionId**

  Id of the partition to be acted on.

5.34.1 Detailed Description

Struct used for Translate variables in Mainstore Memory D/A and Breakpoint Interfaces.

5.34.2 Member Data Documentation

5.34.2.1 bool gipXlateVariables::tagsActive

1 = Tags Active Mode; 0 = Tags Inactive Mode

5.34.2.2 bool gipXlateVariables::littleEndian

1 = Little Endian ; 0 = Big Endian

5.34.2.3 bool gipXlateVariables::mode32bit

1 = 32 bit mode; 0 = 64 bit mode

5.34.2.4 bool gipXlateVariables::writeECC

1 = write ECC using ECC buffer as input; 0 = have ECC calculated

5.34.2.5 bool gipXlateVariables::manualXlateFlag

1 = Manual Translation Needed; 0 = Manual Translation Not Needed

5.34.2.6 gipMainstoreAddrType_t gipXlateVariables::addrType

Type of Mainstore Address being used.

5.34.2.7 uint32_t gipXlateVariables::partitionId

Id of the partition to be acted on.

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

- **gipStructs.H**

Chapter 6

eCMD C/C++ Dll Version Development File Documentation

6.1 cipClientCapi.H File Reference

Cronus & IP eCMD Extension.

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

Load/Unload Functions

- uint32_t **cipInitExtension** ()
  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 Extension.

Extension Owner : Chris Engel

6.1.2 Function Documentation

6.1.2.1 uint32_t cipInitExtension ()

Initialize eCMD CIP Extension DLL.

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

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. 75) 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. 75) 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 cipStructs.H File Reference

Cronus & IP eCMD Extension Structures.

Defines

- #define **ECMD_CIP_CAPI_VERSION** "1.1d"
  eCMD CIP Extension version

Enumerations

- enum **ecmdBreakpointType_t** { **CIP_BREAKPOINT_IABR**, **CIP_BREAKPOINT_DABR**, **CIP_BREAKPOINT_CIABR** }
  Used by setBreakpoint to specify what type of breakpoint to set.

6.2.1 Detailed Description

Cronus & IP eCMD Extension Structures.

Extension Owner : Chris Engel

6.2.2 Define Documentation

6.2.2.1 #define ECMD_CIP_CAPI_VERSION "1.1d"

eCMD CIP Extension version

6.2.3 Enumeration Type Documentation

6.2.3.1 enum ecmdBreakpointType_t

Used by setBreakpoint to specify what type of breakpoint to set.

Enumeration values:

CIP_BREAKPOINT_IABR Instruction Address Breakpoint.

CIP_BREAKPOINT_DABR Data Address Breakpoint.

CIP_BREAKPOINT_CIABR ?? Breakpoint

6.3 cmdClientCapi.H File Reference

eCMD Command line extension

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

Load/Unload Functions

- uint32_t **cmdInitExtension** ()

  Initialize eCMD CMD Extension DLL.

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.

6.3.1 Detailed Description

eCMD Command line extension

Extension Owner : Chris Engel

6.3.2 Function Documentation

6.3.2.1 uint32_t cmdInitExtension ()

Initialize eCMD CMD Extension DLL.

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

6.3.2.2 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

6.3.2.3 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

6.4 cmdStructs.H File Reference

eCMD Command line Extension Structures

Defines

- #define **ECMD_CMD_CAPI_VERSION** "1.1"
  eCMD CMD Extension version

6.4.1 Detailed Description

eCMD Command line Extension Structures

Extension Owner : Chris Engel

6.4.2 Define Documentation

6.4.2.1 #define ECMD_CMD_CAPI_VERSION "1.1"

eCMD CMD Extension version

6.5 croClientCapi.H File Reference

Cronus eCMD Extension.

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

Load/Unload Functions

- uint32_t **croInitExtension** ()

  Initialize eCMD Cronus 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.

6.5.1 Detailed Description

Cronus eCMD Extension.

Extension Owner : Chris Engel

6.5.2 Function Documentation

6.5.2.1 uint32_t croInitExtension ()

Initialize eCMD Cronus Extension DLL.

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 Cronus Extension is initialized and version checked

6.5.2.2 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

6.5.2.3 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'

6.5.2.4 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

6.5.2.5 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

6.5.2.6 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

6.5.2.7 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

6.5.2.8 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

6.5.2.9 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

6.5.2.10 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

6.5.2.11 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

6.5.2.12 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

6.5.2.13 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

6.5.2.14 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

6.5.2.15 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

6.5.2.16 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

6.5.2.17 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

6.5.2.18 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

6.5.2.19 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

6.5.2.20 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

6.5.2.21 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

6.5.2.22 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

6.6 croStructs.H File Reference

Cronus eCMD Extension Structures.

Classes

- struct **croEclipzL2Fields**

  These are used by L2 Display/Alter functions.

- struct **croEclipzPlusL2Fields**
- struct **croL2Fields**

Defines

- #define **ECMD_CRO_CAPI_VERSION** "1.1"

  eCMD Cronus Extension version

Enumerations

- enum **croJtagScanMode** { **CRO_JTAG_SHIFTDR**, **CRO_JTAG_NOTAPSTATEMOVE** }

  These are used by the Jtag Scan functions.

6.6.1 Detailed Description

Cronus eCMD Extension Structures.

Extension Owner : Chris Engel

6.6.2 Define Documentation

6.6.2.1 #define ECMD_CRO_CAPI_VERSION "1.1"

eCMD Cronus Extension version

6.6.3 Enumeration Type Documentation

6.6.3.1 enum croJtagScanMode

These are used by the Jtag Scan functions.

Enumeration values:

CRO_JTAG_SHIFTDR

CRO_JTAG_NOTAPSTATEMOVE

6.7 ecmdClientCapi.H File Reference

eCMD C/C++ Client Interface

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

Simulation Functions

- #define **simFusionOut**(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_PLAIN, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)
- #define **simFusionInfo**(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_INFO, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)
- #define **simFusionWarning**(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_WARNING, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)
- #define **simFusionError**(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_ERROR, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)
- 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.

- uint64_t **simFusionRand64** (uint64_t i_min=0, uint64_t i_max=~0ULL, const char *i_fusionRandObject=NULL)

  Returns a random 64 bit number in the range [min,max] using the Fusion MasterSeed; by default each client will have an own instance of Fusion's Random64BitNumber 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 MAXINT64 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.

Load/Unload Functions

- uint32_t **ecmdLoadDll** (std::string i_dllName)

  Load the eCMD DLL.

- uint32_t **ecmdUnloadDll** ()

  Unload the eCMD DLL.

- uint32_t **ecmdCommandArgs** (int *i_argc, char **i_argv[])

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

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.

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. 23) *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.

6.7.1 Detailed Description

eCMD C/C++ Client Interface

6.7.2 Define Documentation

6.7.2.1 #define simFusionOut(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_PLAIN, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)

6.7.2.2 #define simFusionInfo(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_INFO, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)

6.7.2.3 #define simFusionWarning(header, msg) simOutput-FusionMessage (header,msg, ECMD_SIM_WARNING, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)

6.7.2.4 #define simFusionError(header, msg) simOutputFusionMessage (header,msg, ECMD_SIM_ERROR, ECMD_SIM_MSG_TESTCASE, __FILE__, __LINE__)

6.7.3 Function Documentation

6.7.3.1 uint32_t ecmdLoadDll (std::string *i_dllName*)

Load the eCMD DLL.

Parameters:

i_dllName Specify the full path and name of the dll to load,

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

ECMD_INVALID_DLL_FILENAME if dllName and ECMD_DLL_FILE are not specified

ECMD_DLL_LOAD_FAILURE if failure occurs on call to dlopen
nonzero if unsuccessful

Postcondition:
eCMD DLL is loaded into memory and initialized

See also:
unloadDll

- This function loads the DLL based on dllName if specified, otherwise the env var ECMD_DLL_FILE is used
- Name limit of 255 characters.
- Errors in loading are printed to STDERR.

6.7.3.2 uint32_t ecmdUnloadDll ()

Unload the eCMD DLL.

Return values:
ECMD_SUCCESS if successful unload
ECMD_DLL_LOAD_FAILURE if failure occurs on call to dlclose
nonzero if failure on dll's unload

See also:
loadDll

- Errors in unloading are printed to STDERR

6.7.3.3 uint32_t ecmdCommandArgs (int * *i_argc*, char ** *i_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:
i_argc Passed from Command line Arguments
i_argv Passed from Command line Arguments

Precondition:
loadDll must have been called

Postcondition:
Global options (ex. -debug, -p#, -c#) will be removed from arg list

See also:
loadDll

- argc/argv get passed to the eCMD DLL.
- Global options such as -debug flags and -p#, -c# will be parsed out.
  NOTE : This function does not affect ring caching

6.7.3.4 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

6.7.3.5 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");

}

6.7.3.6 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

6.7.3.7 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

6.7.3.8 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

6.7.3.9 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

6.7.3.10 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

6.7.3.11 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

6.7.3.12 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

6.7.3.13 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

6.7.3.14 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

6.7.3.15 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

6.7.3.16 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. 154)

TARGET DEPTH : Core

TARGET STATES : Unused

6.7.3.17 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. 154)

TARGET DEPTH : Core

TARGET STATES : Unused

6.7.3.18 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

6.7.3.19 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

6.7.3.20 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. 157)

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

6.7.3.21 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. 156)

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

6.7.3.22 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. 158)

NOTE : For processor cores, only "core0 only" addresses are supported, other core addresses cause a failure

TARGET DEPTH : Core

TARGET STATES : Unused

6.7.3.23 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. 157)

NOTE : For processor cores, only "core0 only" addresses are supported, other core addresses cause a failure

TARGET DEPTH : Core

TARGET STATES : Unused

6.7.3.24 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

6.7.3.25 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

6.7.3.26 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

6.7.3.27 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

6.7.3.28 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

6.7.3.29 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

6.7.3.30 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

6.7.3.31 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

6.7.3.32 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

6.7.3.33 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

6.7.3.34 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

6.7.3.35 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

6.7.3.36 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

6.7.3.37 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. 166)
getArrayMultiple(p. 165)

TARGET DEPTH : Pos

TARGET STATES : Unused

6.7.3.38 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. 166)
getArray(p. 165)

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

6.7.3.39 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. 165)

TARGET DEPTH : Pos

TARGET STATES : Unused

6.7.3.40 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. 165)

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

6.7.3.41 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

6.7.3.42 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

6.7.3.43 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

6.7.3.44 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

6.7.3.45 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

6.7.3.46 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

6.7.3.47 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

6.7.3.48 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

6.7.3.49 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

6.7.3.50 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

6.7.3.51 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

6.7.3.52 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. 85) 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. 85) = -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

6.7.3.53 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

6.7.3.54 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. 85) 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

6.7.3.55 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

6.7.3.56 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. 75) 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

6.7.3.57 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

6.7.3.58 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. 75) 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

6.7.3.59 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

6.7.3.60 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. 75) 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

6.7.3.61 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

6.7.3.62 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. 75) 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

6.7.3.63 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

6.7.3.64 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. 75) 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

6.7.3.65 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

6.7.3.66 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. 75) 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

6.7.3.67 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

6.7.3.68 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. 86) = "ALL"

TARGET DEPTH : Core

TARGET STATES : Unused

6.7.3.69 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

6.7.3.70 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

6.7.3.71 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

6.7.3.72 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

6.7.3.73 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

6.7.3.74 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

6.7.3.75 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

6.7.3.76 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

6.7.3.77 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

6.7.3.78 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

6.7.3.79 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

6.7.3.80 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

6.7.3.81 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

6.7.3.82 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

6.7.3.83 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

6.7.3.84 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

6.7.3.85 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

6.7.3.86 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

6.7.3.87 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

6.7.3.88 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

6.7.3.89 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

6.7.3.90 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

6.7.3.91 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

6.7.3.92 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

6.7.3.93 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

6.7.3.94 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

6.7.3.95 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

6.7.3.96 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

6.7.3.97 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

6.7.3.98 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

6.7.3.99 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

6.7.3.100 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

6.7.3.101 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

6.7.3.102 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

6.7.3.103 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

6.7.3.104 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

6.7.3.105 uint64_t simFusionRand64 (uint64_t *i_min* = 0, uint64_t *i_max* = ~0ULL, const char * *i_fusionRandObject* = NULL)

Returns a random 64 bit number in the range [min,max] using the Fusion MasterSeed; by default each client will have an own instance of Fusion's Random64BitNumber 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 MAXINT64 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

6.7.3.106 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

6.7.3.107 void simSetFusionMessageFormat (const char * *i_format*)

Set Fusion Message Format.

Parameters:

i_format New Format

6.7.3.108 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

6.7.3.109 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

6.7.3.110 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

6.7.3.111 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

6.7.3.112 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

6.7.3.113 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

6.7.3.114 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

6.7.3.115 uint32_t ecmdRegisterErrorMsg (uint32_t *i_errorCode*, const char * *i_whom*, const char * *i_message*)

Register an error message that has occured.

6.7.3.116 void ecmdFlushRegisteredErrorMsgs ()

Flush all registered messages, they are no long retrievable.

6.7.3.117 void ecmdOutputError (const char * *i_message*)

Output a message related to an error.

Parameters:

i_message String to output

6.7.3.118 void ecmdOutputWarning (const char * *i_message*)

Output a message related to an warning.

Parameters:

i_message String to output

6.7.3.119 void ecmdOutput (const char * *i_message*)

Output a message to the screen or logs.

Parameters:
i_message String to output

6.7.3.120 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

6.7.3.121 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

6.7.3.122 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

6.7.3.123 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

6.7.3.124 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

6.7.3.125 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

6.7.3.126 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

6.7.3.127 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

6.7.3.128 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

6.7.3.129 uint32_t ecmdTargetToUnitId (ecmdChipTarget & *io_target*)

Converts an eCmd (physical) Target to a HOM Unit Id.

Parameters:

io_target an **ecmdChipTarget**(p. 23) 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. 23) struct are set and valid

TARGET DEPTH : Thread

TARGET STATES : Must be Initialized

6.7.3.130 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

6.7.3.131 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. 23) Fields are set and represent the passed in unitId string

6.7.3.132 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

6.7.3.133 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. 23) struct.

Parameters:

i_core_seq_num Sequence ID number of the core

io_target **ecmdChipTarget**(p. 23) 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. 23) 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

6.7.3.134 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

6.7.3.135 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

6.7.3.136 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

6.7.3.137 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

6.7.3.138 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

6.7.3.139 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

6.7.3.140 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

6.7.3.141 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

6.7.3.142 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

6.7.3.143 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

6.7.3.144 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

6.7.3.145 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

6.7.3.146 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

6.7.3.147 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

6.7.3.148 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

6.8 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. 27)*, 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.

6.8.1 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

6.8.2 Define Documentation

6.8.2.1 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

6.8.2.2 #define ECMD_DBUF_INIT_FAIL (0x01000000 | 0x2000)

Initialization of the DataBuffer failed.

6.8.2.3 #define ECMD_DBUF_BUFFER_OVERFLOW (0x01000000 | 0x2010)

Attempt to read/write data beyond the length of the DataBuffer.

6.8.2.4 #define ECMD_DBUF_XSTATE_ERROR (0x01000000 | 0x2020)

An 'X' character occured where it was not expected.

6.8.2.5 #define ECMD_DBUF_UNDEFINED_FUNCTION (0x01000000 | 0x2030)

Function not included in this version of DataBuffer.

6.8.2.6 #define ECMD_DBUF_INVALID_ARGS (0x01000000 | 0x2040)

Args provided to dataBuffer were invalid.

6.8.2.7 #define ECMD_DBUF_INVALID_DATA_FORMAT (0x01000000 | 0x2041)

String data didn't match expected input format.

6.8.2.8 #define ECMD_DBUF_FOPEN_FAIL (0x01000000 | 0x2050)

File open on file for reading or writing the data buffer failed.

6.8.2.9 #define ECMD_DBUF_FILE_FORMAT_MISMATCH (0x01000000 | 0x2051)

In readFile specified format not found in the data file.

6.8.2.10 #define ECMD_DBUF_DATANUMBER_NOT_FOUND (0x01000000 | 0x2052)

In readFileMultiple specified data number not found in file.

6.8.2.11 #define ECMD_DBUF_FILE_OPERATION_FAIL (0x01000000 | 0x2053)

File operation failed.

6.8.2.12 #define ECMD_DBUF_NOT_OWNER (0x01000000 | 0x2060)

Don't own this buffer so can't do this operation.

6.8.2.13 #define ECMD_DBUF_XSTATE_NOT_ENABLED (0x01000000 | 0x2062)

Xstate function called on a buffer that doesn't have xstates enabled.

6.8.2.14 #define ETRAC0(fmt) printf("%s> ETRC: " fmt "\n", __FUNCTION__);

6.8.2.15 #define ETRAC1(fmt, arg1) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1);

6.8.2.16 #define ETRAC2(fmt, arg1, arg2) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2);

6.8.2.17 #define ETRAC3(fmt, arg1, arg2, arg3) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3);

6.8.2.18 #define ETRAC4(fmt, arg1, arg2, arg3, arg4) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4);

6.8.2.19 #define ETRAC5(fmt, arg1, arg2, arg3, arg4, arg5) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5);

6.8.2.20 #define ETRAC6(fmt, arg1, arg2, arg3, arg4, arg5, arg6) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6);

6.8.2.21 #define ETRAC7(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7);

6.8.2.22 #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);

6.8.2.23 #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);

6.8.3 Enumeration Type Documentation

6.8.3.1 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

6.8.3.2 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.

6.9 ecmdReturnCodes.H File Reference

All Return Codes for the eCmd Capi.

Defines

- #define **ECMD_ERR_UNKNOWN** 0x00000000
  This error code wasn't flagged to which plugin it came from.
- #define **ECMD_ERR_ECMD** 0x01000000
  Error came from eCMD.
- #define **ECMD_ERR_CRONUS** 0x02000000
  Error came from Cronus.
- #define **ECMD_ERR_IP** 0x04000000
  Error came from IP GFW.
- #define **ECMD_ERR_Z** 0x08000000
  Error came from Z GFW.
- #define **ECMD_SUCCESS** 0x0
  API Returned Successfully.
- #define **ECMD_INVALID_DLL_VERSION** (ECMD_ERR_ECMD | 0x1000)
  Dll Version didn't match the Client version detected.
- #define **ECMD_INVALID_DLL_FILENAME** (ECMD_ERR_ECMD | 0x1001)
  Unable to find filename to load or file doesn't exist.
- #define **ECMD_DLL_LOAD_FAILURE** (ECMD_ERR_ECMD | 0x1002)
  Error occured on call to dlopen.
- #define **ECMD_DLL_UNLOAD_FAILURE** (ECMD_ERR_ECMD | 0x1003)
  Error occurred on call to dlclose.
- #define **ECMD_DLL_UNINITIALIZED** (ECMD_ERR_ECMD | 0x1004)
  A function was called before ecmdLoadDll was called.
- #define **ECMD_DLL_INVALID** (ECMD_ERR_ECMD | 0x1005)
  If we are unable to lookup a function in the Dll.
- #define **ECMD_FAILURE** (ECMD_ERR_ECMD | 0x1010)
  General Failure occurred in eCMD.
- #define **ECMD_TARGET_NOT_CONFIGURED** (ECMD_ERR_ECMD | 0x1011)
  Chip target provided was not configured in the system.

- #define **ECMD_FUNCTION_NOT_SUPPORTED** (ECMD_ERR_ECMD | 0x1012)

  Returned if a specific Dll instance doesn't support the function you called.

- #define **ECMD_UNKNOWN_FILE** (ECMD_ERR_ECMD | 0x1013)

  ecmdQueryFileLocation was unable to find the file you requested

- #define **ECMD_INVALID_ARGS** (ECMD_ERR_ECMD | 0x1020)

  Not enough arguments provided to the function.

- #define **ECMD_INVALID_SPY_ENUM** (ECMD_ERR_ECMD | 0x1021)

  getSpyEnum or putSpyEnum used an invalid enum

- #define **ECMD_SPY_FAILED_ECC_CHECK** (ECMD_ERR_ECMD | 0x1022)

  getSpy or getSpyEnum failed with invalid ECC detected in the hardware

- #define **ECMD_SPY_NOT_ENUMERATED** (ECMD_ERR_ECMD | 0x1023)

  getSpyEnum or putSpyEnum was called on a non-enumerated spy

- #define **ECMD_SPY_IS_EDIAL** (ECMD_ERR_ECMD | 0x1024)

  getSpy or Putspy was called on an edial

- #define **ECMD_INVALID_SPY** (ECMD_ERR_ECMD | 0x1025)

  Spy functions found an invalid Spy name or type.

- #define **ECMD_DATA_OVERFLOW** (ECMD_ERR_ECMD | 0x1026)

  Too much data was provided to a write function.

- #define **ECMD_DATA_UNDERFLOW** (ECMD_ERR_ECMD | 0x1027)

  Too little data was provided to a write function.

- #define **ECMD_INVALID_RING** (ECMD_ERR_ECMD | 0x1028)

  Invalid ring name was provided.

- #define **ECMD_INVALID_ARRAY** (ECMD_ERR_ECMD | 0x1029)

  Invalid array name was provided.

- #define **ECMD_INVALID_CONFIG** (ECMD_ERR_ECMD | 0x1030)

  There was an error processing the configuration information.

- #define **ECMD_CLOCKS_IN_INVALID_STATE** (ECMD_ERR_ECMD | 0x1031)

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

- #define **ECMD_NON_JTAG_CHIP** (ECMD_ERR_ECMD | 0x1032)

  JTag function called on non-jtag attached chip.

- #define **ECMD_NON_FSI_CHIP** (ECMD_ERR_ECMD | 0x1033)

  Fsi function called on non-fsi attached chip.

- #define **ECMD_INVALID_SPR** (ECMD_ERR_ECMD | 0x1034)
  Invalid SPR was specified to get/put spr functions.

- #define **ECMD_INVALID_GPR** (ECMD_ERR_ECMD | 0x1035)
  Invalid GPR number was specified to get/put gpr functions.

- #define **ECMD_INVALID_FPR** (ECMD_ERR_ECMD | 0x1036)
  Invalid FPR number was specified to get/put fpr functions.

- #define **ECMD_RING_CACHE_ENABLED** (ECMD_ERR_ECMD | 0x1037)
  Ring Cache enabled during call non-cache enabled function.

- #define **ECMD_INVALID_CONFIG_NAME** (ECMD_ERR_ECMD | 0x1038)
  An Invalid name was used to set/get a configuation setting.

- #define **ECMD_SPY_GROUP_MISMATCH** (ECMD_ERR_ECMD | 0x1039)
  A mismatch was found reading a group spy not all groups set the same.

- #define **ECMD_INVALID_CLOCK_DOMAIN** (ECMD_ERR_ECMD | 0x1040)
  An invalid clock domain name was specified.

- #define **ECMD_CLOCKS_ALREADY_OFF** (ECMD_ERR_ECMD | 0x1041)
  A stopclocks was requested when clocks are already off.

- #define **ECMD_CLOCKS_ALREADY_ON** (ECMD_ERR_ECMD | 0x1042)
  A startclocks was requested when clocks are already on.

- #define **ECMD_UNABLE_TO_OPEN_SCANDEF** (ECMD_ERR_ECMD | 0x1043)
  eCMD was unable to open the scandef

- #define **ECMD_INVALID_LATCHNAME** (ECMD_ERR_ECMD | 0x1044)
  eCMD was unable to find the specified latch in the scandef

- #define **ECMD_POLLING_FAILURE** (ECMD_ERR_ECMD | 0x1045)
  eCMD failed waiting for a poll to match expected value

- #define **ECMD_TARGET_INVALID_TYPE** (ECMD_ERR_ECMD | 0x1046)
  Target specified an object that was inappropriate for the function.

- #define **ECMD_EXTENSION_NOT_SUPPORTED** (ECMD_ERR_ECMD | 0x1047)
  The current plugin does not supported the requested extension.

- #define **ECMD_ISTEPS_INVALID_STEP** (ECMD_ERR_ECMD | 0x1048)
  An invalid step name was provided.

- #define **ECMD_UNABLE_TO_OPEN_SCANDEFHASH** (ECMD_ERR_-ECMD | 0x1049)
  eCMD was unable to open the scandefhash

- #define **ECMD_SCANDEFHASH_MULT_RINGS** (ECMD_ERR_ECMD | 0x1050)

  Multiple ring keys matching the same latchname found.

- #define **ECMD_UNABLE_TO_OPEN_SCOMDEF** (ECMD_ERR_ECMD | 0x1051)

  eCMD was unable to open scomdef file

- #define **ECMD_SCOMADDRESS_NOT_FOUND** (ECMD_ERR_ECMD | 0x1052)

  Scom Address not found in the ScomDef file.

- #define **ECMD_INVALID_ENTRY_REQUESTED** (ECMD_ERR_ECMD | 0x1053)

  An invalid entry was requested.

- #define **ECMD_INSTRUCTIONS_IN_INVALID_STATE** (ECMD_ERR_ECMD | 0x1054)

  Instructions were in an invalid state for operation.

- #define **ECMD_INVALID_MEMORY_ADDRESS** (ECMD_ERR_ECMD | 0x1055)

  Memory Address was not on a 8-byte boundary.

- #define **ECMD_RETRY_WITH_VIRTUAL_ADDR** (ECMD_ERR_IP | 0x1056)

  Requests that the user retries operation with returned Virtual Address.

- #define **ECMD_UNABLE_TO_MAP_HASHID** (ECMD_ERR_IP | 0x1057)

  Mapping function was unable to match the hashid given by the user.

- #define **ECMD_UNABLE_TO_OPEN_ARRAYDEF** (ECMD_ERR_ECMD | 0x1058)

  eCMD was unable to open arraydef file

- #define **ECMD_INVALID_RETURN_DATA** (ECMD_ERR_ECMD | 0x1059)

  Data returned (if any) is incomplete or invalid... caller beware.

- #define **ECMD_INT_UNKNOWN_COMMAND** (ECMD_ERR_ECMD | 0x1900)

  Command interpreter didn't understand command.

- #define **ECMD_EXPECT_FAILURE** (ECMD_ERR_ECMD | 0x1901)

  An expect was performed and a miscompare was found.

- #define **ECMD_SCANDEF_LOOKUP_FAILURE** (ECMD_ERR_ECMD | 0x1902)

  An Error occurred trying to process the scandef file.

- #define **ECMD_DATA_BOUNDS_OVERFLOW** (ECMD_ERR_ECMD | 0x1903)

  The user specified to get/put data that was larger then ECMD_MAX_DATA_BITS.

- #define **ECMD_DBUF_SUCCESS** 0x0

  DataBuffer returned successfully.

- #define **ECMD_DBUF_INIT_FAIL** (ECMD_ERR_ECMD | 0x2000)

  Initialization of the DataBuffer failed.

- #define **ECMD_DBUF_BUFFER_OVERFLOW** (ECMD_ERR_ECMD | 0x2010)

  Attempt to read/write data beyond the length of the DataBuffer.

- #define **ECMD_DBUF_XSTATE_ERROR** (ECMD_ERR_ECMD | 0x2020)

  An 'X' character occured where it was not expected.

- #define **ECMD_DBUF_UNDEFINED_FUNCTION** (ECMD_ERR_ECMD | 0x2030)

  Function not included in this version of DataBuffer.

- #define **ECMD_DBUF_INVALID_ARGS** (ECMD_ERR_ECMD | 0x2040)

  Args provided to dataBuffer were invalid.

- #define **ECMD_DBUF_INVALID_DATA_FORMAT** (ECMD_ERR_ECMD | 0x2041)

  String data didn't match expected input format.

- #define **ECMD_DBUF_FOPEN_FAIL** (ECMD_ERR_ECMD | 0x2050)

  File open on file for reading or writing the data buffer failed.

- #define **ECMD_DBUF_FILE_FORMAT_MISMATCH** (ECMD_ERR_ECMD | 0x2051)

  In readFile specified format not found in the data file.

- #define **ECMD_DBUF_DATANUMBER_NOT_FOUND** (ECMD_ERR_ECMD | 0x2052)

  In readFileMultiple specified data number not found in file.

- #define **ECMD_DBUF_FILE_OPERATION_FAIL** (ECMD_ERR_ECMD | 0x2053)

  File operation failed.

- #define **ECMD_DBUF_NOT_OWNER** (ECMD_ERR_ECMD | 0x2060)

  Don't own this buffer so can't do this operation.

- #define **ECMD_DBUF_XSTATE_NOT_ENABLED** (ECMD_ERR_ECMD | 0x2062)

  Xstate function called on a buffer that doesn't have xstates enabled.

6.9.1 Detailed Description

All Return Codes for the eCmd Capi.

6.9.2 Define Documentation

6.9.2.1 #define ECMD_ERR_UNKNOWN 0x00000000

This error code wasn't flagged to which plugin it came from.

6.9.2.2 #define ECMD_ERR_ECMD 0x01000000

Error came from eCMD.

6.9.2.3 #define ECMD_ERR_CRONUS 0x02000000

Error came from Cronus.

6.9.2.4 #define ECMD_ERR_IP 0x04000000

Error came from IP GFW.

6.9.2.5 #define ECMD_ERR_Z 0x08000000

Error came from Z GFW.

6.9.2.6 #define ECMD_SUCCESS 0x0

API Returned Successfully.

6.9.2.7 #define ECMD_INVALID_DLL_VERSION (ECMD_ERR_ECMD | 0x1000)

Dll Version didn't match the Client version detected.

6.9.2.8 #define ECMD_INVALID_DLL_FILENAME (ECMD_ERR_ECMD | 0x1001)

Unable to find filename to load or file doesn't exist.

6.9.2.9 #define ECMD_DLL_LOAD_FAILURE (ECMD_ERR_ECMD | 0x1002)

Error occured on call to dlopen.

6.9.2.10 #define ECMD_DLL_UNLOAD_FAILURE (ECMD_ERR_ECMD | 0x1003)

Error occurred on call to dlclose.

6.9.2.11 #define ECMD_DLL_UNINITIALIZED (ECMD_ERR_ECMD | 0x1004)

A function was called before ecmdLoadDll was called.

6.9.2.12 #define ECMD_DLL_INVALID (ECMD_ERR_ECMD | 0x1005)

If we are unable to lookup a function in the Dll.

6.9.2.13 #define ECMD_FAILURE (ECMD_ERR_ECMD | 0x1010)

General Failure occurred in eCMD.

6.9.2.14 #define ECMD_TARGET_NOT_CONFIGURED (ECMD_ERR_ECMD | 0x1011)

Chip target provided was not configured in the system.

6.9.2.15 #define ECMD_FUNCTION_NOT_SUPPORTED (ECMD_ERR_ECMD | 0x1012)

Returned if a specific Dll instance doesn't support the function you called.

6.9.2.16 #define ECMD_UNKNOWN_FILE (ECMD_ERR_ECMD | 0x1013)

ecmdQueryFileLocation was unable to find the file you requested

6.9.2.17 #define ECMD_INVALID_ARGS (ECMD_ERR_ECMD | 0x1020)

Not enough arguments provided to the function.

6.9.2.18 #define ECMD_INVALID_SPY_ENUM (ECMD_ERR_ECMD | 0x1021)

getSpyEnum or putSpyEnum used an invalid enum

6.9.2.19 #define ECMD_SPY_FAILED_ECC_CHECK (ECMD_ERR_ECMD | 0x1022)

getSpy or getSpyEnum failed with invalid ECC detected in the hardware

6.9.2.20 #define ECMD_SPY_NOT_ENUMERATED (ECMD_ERR_ECMD | 0x1023)

getSpyEnum or putSpyEnum was called on a non-enumerated spy

6.9.2.21 #define ECMD_SPY_IS_EDIAL (ECMD_ERR_ECMD | 0x1024)

getSpy or Putspy was called on an edial

6.9.2.22 #define ECMD_INVALID_SPY (ECMD_ERR_ECMD | 0x1025)

Spy functions found an invalid Spy name or type.

6.9.2.23 #define ECMD_DATA_OVERFLOW (ECMD_ERR_ECMD | 0x1026)

Too much data was provided to a write function.

6.9.2.24 #define ECMD_DATA_UNDERFLOW (ECMD_ERR_ECMD | 0x1027)

Too little data was provided to a write function.

6.9.2.25 #define ECMD_INVALID_RING (ECMD_ERR_ECMD | 0x1028)

Invalid ring name was provided.

6.9.2.26 #define ECMD_INVALID_ARRAY (ECMD_ERR_ECMD | 0x1029)

Invalid array name was provided.

6.9.2.27 #define ECMD_INVALID_CONFIG (ECMD_ERR_ECMD | 0x1030)

There was an error processing the configuration information.

6.9.2.28 #define ECMD_CLOCKS_IN_INVALID_STATE (ECMD_ERR_ECMD | 0x1031)

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

6.9.2.29 #define ECMD_NON_JTAG_CHIP (ECMD_ERR_ECMD | 0x1032)

JTag function called on non-jtag attached chip.

6.9.2.30 #define ECMD_NON_FSI_CHIP (ECMD_ERR_ECMD | 0x1033)

Fsi function called on non-fsi attached chip.

6.9.2.31 #define ECMD_INVALID_SPR (ECMD_ERR_ECMD | 0x1034)

Invalid SPR was specified to get/put spr functions.

6.9.2.32 #define ECMD_INVALID_GPR (ECMD_ERR_ECMD | 0x1035)

Invalid GPR number was specified to get/put gpr functions.

6.9.2.33 #define ECMD_INVALID_FPR (ECMD_ERR_ECMD | 0x1036)

Invalid FPR number was specified to get/put fpr functions.

6.9.2.34 #define ECMD_RING_CACHE_ENABLED (ECMD_ERR_ECMD | 0x1037)

Ring Cache enabled during call non-cache enabled function.

6.9.2.35 #define ECMD_INVALID_CONFIG_NAME (ECMD_ERR_ECMD | 0x1038)

An Invalid name was used to set/get a configuation setting.

6.9.2.36 #define ECMD_SPY_GROUP_MISMATCH (ECMD_ERR_ECMD | 0x1039)

A mismatch was found reading a group spy not all groups set the same.

6.9.2.37 #define ECMD_INVALID_CLOCK_DOMAIN (ECMD_ERR_ECMD | 0x1040)

An invalid clock domain name was specified.

6.9.2.38 #define ECMD_CLOCKS_ALREADY_OFF (ECMD_ERR_ECMD | 0x1041)

A stopclocks was requested when clocks are already off.

6.9.2.39 #define ECMD_CLOCKS_ALREADY_ON (ECMD_ERR_ECMD | 0x1042)

A startclocks was requested when clocks are already on.

6.9.2.40 #define ECMD_UNABLE_TO_OPEN_SCANDEF (ECMD_ERR_ECMD | 0x1043)

eCMD was unable to open the scandef

6.9.2.41 #define ECMD_INVALID_LATCHNAME (ECMD_ERR_ECMD | 0x1044)

eCMD was unable to find the specified latch in the scandef

6.9.2.42 #define ECMD_POLLING_FAILURE (ECMD_ERR_ECMD | 0x1045)

eCMD failed waiting for a poll to match expected value

6.9.2.43 #define ECMD_TARGET_INVALID_TYPE (ECMD_ERR_ECMD | 0x1046)

Target specified an object that was inappropriate for the function.

6.9.2.44 #define ECMD_EXTENSION_NOT_SUPPORTED (ECMD_ERR_ECMD | 0x1047)

The current plugin does not supported the requested extension.

6.9.2.45 #define ECMD_ISTEPS_INVALID_STEP (ECMD_ERR_ECMD | 0x1048)

An invalid step name was provided.

6.9.2.46 #define ECMD_UNABLE_TO_OPEN_SCANDEFHASH (ECMD_ERR_ECMD | 0x1049)

eCMD was unable to open the scandefhash

6.9.2.47 #define ECMD_SCANDEFHASH_MULT_RINGS (ECMD_ERR_ECMD | 0x1050)

Multiple ring keys matching the same latchname found.

6.9.2.48 #define ECMD_UNABLE_TO_OPEN_SCOMDEF (ECMD_ERR_ECMD | 0x1051)

eCMD was unable to open scomdef file

6.9.2.49 #define ECMD_SCOMADDRESS_NOT_FOUND (ECMD_ERR_ECMD | 0x1052)

Scom Address not found in the ScomDef file.

6.9.2.50 #define ECMD_INVALID_ENTRY_REQUESTED (ECMD_ERR_ECMD | 0x1053)

An invalid entry was requested.

6.9.2.51 #define ECMD_INSTRUCTIONS_IN_INVALID_STATE (ECMD_ERR_ECMD | 0x1054)

Instructions were in an invalid state for operation.

6.9.2.52 #define ECMD_INVALID_MEMORY_ADDRESS (ECMD_ERR_ECMD | 0x1055)

Memory Address was not on a 8-byte boundary.

6.9.2.53 #define ECMD_RETRY_WITH_VIRTUAL_ADDR (ECMD_ERR_IP | 0x1056)

Requests that the user retries operation with returned Virtual Address.

6.9.2.54 #define ECMD_UNABLE_TO_MAP_HASHID (ECMD_ERR_IP | 0x1057)

Mapping function was unable to match the hashid given by the user.

6.9.2.55 #define ECMD_UNABLE_TO_OPEN_ARRAYDEF (ECMD_ERR_ECMD | 0x1058)

eCMD was unable to open arraydef file

6.9.2.56 #define ECMD_INVALID_RETURN_DATA (ECMD_ERR_ECMD | 0x1059)

Data returned (if any) is incomplete or invalid... caller beware.

6.9.2.57 #define ECMD_INT_UNKNOWN_COMMAND (ECMD_ERR_ECMD | 0x1900)

Command interpreter didn't understand command.

6.9.2.58 #define ECMD_EXPECT_FAILURE (ECMD_ERR_ECMD | 0x1901)

An expect was performed and a miscompare was found.

6.9.2.59 #define ECMD_SCANDEF_LOOKUP_FAILURE (ECMD_ERR_ECMD | 0x1902)

An Error occurred trying to process the scandef file.

6.9.2.60 #define ECMD_DATA_BOUNDS_OVERFLOW (ECMD_ERR_ECMD | 0x1903)

The user specified to get/put data that was larger then ECMD_MAX_DATA_BITS.

6.9.2.61 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

6.9.2.62 #define ECMD_DBUF_INIT_FAIL (ECMD_ERR_ECMD | 0x2000)

Initialization of the DataBuffer failed.

6.9.2.63 #define ECMD_DBUF_BUFFER_OVERFLOW (ECMD_ERR_ECMD | 0x2010)

Attempt to read/write data beyond the length of the DataBuffer.

6.9.2.64 #define ECMD_DBUF_XSTATE_ERROR (ECMD_ERR_ECMD | 0x2020)

An 'X' character occured where it was not expected.

6.9.2.65 #define ECMD_DBUF_UNDEFINED_FUNCTION (ECMD_ERR_ECMD | 0x2030)

Function not included in this version of DataBuffer.

6.9.2.66 #define ECMD_DBUF_INVALID_ARGS (ECMD_ERR_ECMD | 0x2040)

Args provided to dataBuffer were invalid.

6.9.2.67 #define ECMD_DBUF_INVALID_DATA_FORMAT (ECMD_ERR_ECMD | 0x2041)

String data didn't match expected input format.

6.9.2.68 #define ECMD_DBUF_FOPEN_FAIL (ECMD_ERR_ECMD | 0x2050)

File open on file for reading or writing the data buffer failed.

6.9.2.69 #define ECMD_DBUF_FILE_FORMAT_MISMATCH (ECMD_ERR_ECMD | 0x2051)

In readFile specified format not found in the data file.

6.9.2.70 #define ECMD_DBUF_DATANUMBER_NOT_FOUND (ECMD_ERR_ECMD | 0x2052)

In readFileMultiple specified data number not found in file.

6.9.2.71 #define ECMD_DBUF_FILE_OPERATION_FAIL (ECMD_ERR_ECMD | 0x2053)

File operation failed.

6.9.2.72 #define ECMD_DBUF_NOT_OWNER (ECMD_ERR_ECMD | 0x2060)

Don't own this buffer so can't do this operation.

6.9.2.73 #define ECMD_DBUF_XSTATE_NOT_ENABLED (ECMD_ERR_ECMD | 0x2062)

Xstate function called on a buffer that doesn't have xstates enabled.

6.10 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>
```

Command Line Parsing Functions

- bool **ecmdParseOption** (int *io_argc, char **io_argv[], const char *i_option)

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

- char * **ecmdParseOptionWithArgs** (int *io_argc, char **io_argv[], const char *i_option)

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

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. 23).

6.10.1 Detailed Description

Useful functions for use throughout the ecmd C API and Plugin.

6.10.2 Enumeration Type Documentation

6.10.2.1 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

6.10.2.2 enum ecmdTargetDisplayMode_t

Used by ecmdWriteTarget to specify displayMode.

Enumeration values:

ECMD_DISPLAY_TARGET_DEFAULT Default mode.

6.10.3 Function Documentation

6.10.3.1 bool ecmdParseOption (int * *io_argc*, 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_argc Pointer to number of elements in io_argv array

io_argv Array of strings passed in from command line

i_option Option to look for

See also:
ecmdParseOptionWithArgs(p. 233)

6.10.3.2 char* ecmdParseOptionWithArgs (int * *io_argc*, 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_argc Pointer to number of elements in io_argv array

io_argv Array of strings passed in from command line

i_option Option to look for

See also:
ecmdParseOptionWithArgs(p. 233)

6.10.3.3 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

6.10.3.4 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

6.10.3.5 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. 235)
ecmdGenB32FromHexLeft(p. 234)

6.10.3.6 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. 235)
ecmdGenB32FromHex(p. 234)

6.10.3.7 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. 234)
ecmdGenB32FromHexLeft(p. 234)

6.10.3.8 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

6.10.3.9 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

6.10.3.10 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. 23) 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

6.10.3.11 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

6.10.3.12 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. 23).

Returns:

String with formatted target data

Parameters:

i_target **ecmdChipTarget**(p. 23) containing data to format into string

i_displayMode Mode to format data

TARGET DEPTH : Thread TARGET STATES : Must be Initialized

6.11 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. 20) *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. 23) *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.

6.11.1 Detailed Description

All the Structures required for the eCMD Capi.

6.11.2 Define Documentation

6.11.2.1 #define ECMD_CAPI_VERSION "6.0"

eCMD API Version

6.11.2.2 #define QD_HDR_MAGIC 0xFFFFFFF1

6.11.2.3 #define CAGE_HDR_MAGIC 0xFFFFFF2F

6.11.2.4 #define NODE_HDR_MAGIC 0xFFFFF3FF

6.11.2.5 #define SLOT_HDR_MAGIC 0xFFFF4FFF

6.11.2.6 #define CHIP_HDR_MAGIC 0xFFF5FFFF

6.11.2.7 #define CORE_HDR_MAGIC 0xFF6FFFFF

6.11.2.8 #define THREAD_HDR_MAGIC 0xF7FFFFFF

6.11.2.9 #define ECMD_CHIPT_PROCESSOR "pu"

Predefined common chip names for **ecmdChipData.chipCommonType**(p. 21)

6.11.2.10 #define ECMD_CHIPT_MEM_BUF "memb"

6.11.2.11 #define ECMD_CHIPT_MEM_CNTRL "memc"

6.11.2.12 #define ECMD_CHIPT_MEM_L2CACHE "l2cache"

6.11.2.13 #define ECMD_CHIPT_MEM_L3CACHE "l3cache"

6.11.2.14 #define ECMD_CHIPT_IOBDG "iobdg"

6.11.2.15 #define ECMD_CHIPT_IOHUB "iohub"

6.11.2.16 #define ECMD_CHIPT_MUX "mux"

6.11.2.17 #define ECMD_CHIPT_SERVICE_PROCESSOR "sp"

6.11.2.18 #define ECMD_CHIPFLAG_BUSMASK 0xC0000000

Defines for the **ecmdChipData**(p. 20) chipFlags field

6.11.2.19 #define ECMD_CHIPFLAG_RSVDBUS1 0x00000000

This is reserved for later expansion (should not be used).

6.11.2.20 #define ECMD_CHIPFLAG_JTAG 0x40000000

6.11.2.21 #define ECMD_CHIPFLAG_FSI 0x80000000

6.11.2.22 #define ECMD_CHIPFLAG_RSVDBUS2 0xC0000000

This is reserved for later expansion (should not be used).

6.11.3 Enumeration Type Documentation

6.11.3.1 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.

6.11.3.2 enum ecmdChipInterfaceType_t

Used in **ecmdChipData**(p. 20) 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.

6.11.3.3 enum ecmdChipTargetState_t

Used by **ecmdChipTarget**(p. 23) 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.

6.11.3.4 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

6.11.3.5 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.

6.11.3.6 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.

6.11.3.7 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.

6.11.3.8 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.

6.11.3.9 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.

6.11.3.10 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.

6.11.3.11 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.

6.11.3.12 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.

6.11.3.13 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.

6.11.3.14 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.

6.11.3.15 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.

6.11.3.16 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.

6.11.3.17 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

6.11.3.18 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

6.11.3.19 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.

6.11.3.20 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.

6.11.3.21 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.

6.11.3.22 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.

6.11.3.23 enum ecmdSpyType_t

Used for the ecmdQuerySpy function to specify which type of spy we have.

See also:

ecmdSpyData(p. 96)

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.

6.11.3.24 enum ecmdTraceType_t

Used by ecmdSetTraceMode to specify which trace to control.

Enumeration values:

ECMD_TRACE_SCAN Scan Trace.

ECMD_TRACE_PROCEDURE Procedure Trace.

6.11.4 Function Documentation

6.11.4.1 bool operator< (const ecmdCageData & *lhs*, const ecmdCageData & *rhs*)

Used to sort Cage entries in an **ecmdCageData**(p. 19) list.

6.11.4.2 bool operator< (const ecmdNodeData & *lhs*, const ecmdNodeData & *rhs*)

Used to sort Node entries in an **ecmdNodeData**(p. 87) list.

6.11.4.3 bool operator< (const ecmdSlotData & *lhs*, const ecmdSlotData & *rhs*)

Used to sort Slot entries in an **ecmdSlotData**(p. 95) list.

6.11.4.4 bool operator< (const ecmdChipData & *lhs*, const ecmdChipData & *rhs*)

Used to sort Chip entries (based on Pos) in an **ecmdChipData**(p. 20) list.

6.11.4.5 bool operator< (const ecmdCoreData & *lhs*, const ecmdCoreData & *rhs*)

Used to sort Core entries in an **ecmdCoreData**(p. 26) list.

6.11.4.6 bool operator< (const ecmdThreadData & *lhs*, const ecmdThreadData & *rhs*)

Used to sort Thread entries in an **ecmdThreadData**(p. 99) list.

6.11.4.7 std::string ecmdGetSharedLibVersion ()

Returns the version of the shared lib so it can be compared with the other versions.

6.12 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.

Defines

- #define **PTRAC0**(fmt)
- #define **PTRAC1**(fmt, arg1)
- #define **PTRAC2**(fmt, arg1, arg2)
- #define **PTRAC3**(fmt, arg1, arg2, arg3)
- #define **PTRAC4**(fmt, arg1, arg2, arg3, arg4)
- #define **PTRAC5**(fmt, arg1, arg2, arg3, arg4, arg5)
- #define **PTRAC6**(fmt, arg1, arg2, arg3, arg4, arg5, arg6)
- #define **PTRAC7**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7)
- #define **PTRAC8**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)
- #define **PTRAC9**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9)

6.12.1 Detailed Description

Useful functions for use throughout the ecmd C API.

6.12.2 Define Documentation

6.12.2.1 #define PTRAC0(fmt)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__); \
  ecmdOutput(buffer);}
```

6.12.2.2 #define PTRAC1(fmt, arg1)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1); \
  ecmdOutput(buffer);}
```

6.12.2.3 #define PTRAC2(fmt, arg1, arg2)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2); \
  ecmdOutput(buffer);}
```

6.12.2.4 #define PTRAC3(fmt, arg1, arg2, arg3)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3); \
  ecmdOutput(buffer);}
```

6.12.2.5 #define PTRAC4(fmt, arg1, arg2, arg3, arg4)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, \
  arg4); \
  ecmdOutput(buffer);}
```

6.12.2.6 #define PTRAC5(fmt, arg1, arg2, arg3, arg4, arg5)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, arg4, \
  arg5); \
  ecmdOutput(buffer);}
```

6.12.2.7 #define PTRAC6(fmt, arg1, arg2, arg3, arg4, arg5, arg6)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, arg4, \
  arg5, arg6); \
  ecmdOutput(buffer);}
```

6.12.2.8 #define PTRAC7(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, arg4, \
  arg5, arg6, arg7); \
  ecmdOutput(buffer);}
```

6.12.2.9 #define PTRAC8(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, arg4, \
  arg5, arg6, arg7, arg8); \
  ecmdOutput(buffer);}
```

6.12.2.10 #define PTRAC9(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9)

Value:

```
{char buffer [255]; \
  snprintf( buffer, 255, "%s> PTRC: "fmt, __FUNCTION__, arg1, arg2, arg3, arg4, \
  arg5, arg6, arg7, arg8, arg9); \
  ecmdOutput(buffer);}
```

6.12.3 Function Documentation

6.12.3.1 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. 23) 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. 253)

TARGET DEPTH : Thread TARGET STATES : Must Be Initialized

6.12.3.2 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. 23) 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. 253)

TARGET DEPTH : Thread TARGET STATES : Must be Initialized (from ecmdConfigLooperInit)

6.12.3.3 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. 27) 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

6.12.3.4 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

6.12.3.5 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. 27) 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

6.12.3.6 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

6.12.3.7 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. 23) that information is requested for

o_ data **ecmdChipData**(p. 20) struct that contains detailed info on chip ec level, etc.

TARGET DEPTH : Pos TARGET STATES : Unused

6.12.3.8 uint32_t ecmdDisplayDllInfo ()

Function calls ecmdQueryDllInfo and displays the output to stdout.

Return values:
ECMD_ SUCCESS if successful

nonzero on failure

6.12.3.9 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

6.13 gipClientCapi.H File Reference

GFW IP Series eCMD Extension.

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

Load/Unload Functions

- uint32_t **gipInitExtension** ()

  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.

6.13.1 Detailed Description

GFW IP Series eCMD Extension.

Extension Owner : Mike Baiocchi

6.13.2 Function Documentation

6.13.2.1 uint32_t gipInitExtension ()

Initialize eCMD gip Extension DLL.

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

6.13.2.2 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

6.13.2.3 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

6.13.2.4 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

6.13.2.5 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

6.13.2.6 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

6.13.2.7 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

6.13.2.8 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

6.13.2.9 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

6.13.2.10 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

6.13.2.11 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

6.13.2.12 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

6.14 gipStructs.H File Reference

GFW IP Series eCMD Extension Structures.

```
#include <chiccommon.H>
```

Classes

- struct **gipXlateVariables**
  Struct used for Translate variables in Mainstore Memory D/A and Breakpoint Interfaces.
- struct **cpcfSysInfo_t**

Defines

- #define **ECMD_GIP_CAPI_VERSION** "1.0"
  eCMD gip Extension version
- #define **HANDLE_SYSV_PRAGMA** 1

Enumerations

- enum **gipMainstoreAddrType_t** {
  GIP_MAINSTORE_REAL_ADDR, **GIP_MAINSTORE_EFFECTIVE_ADDR**, **GIP_MAINSTORE_VIRTUAL_ADDR**, **GIP_MAINSTORE_RMO_ADDR**,
  GIP_MAINSTORE_VRM_ADDR }
  Enum used to identify Mainstore Address Type.

6.14.1 Detailed Description

GFW IP Series eCMD Extension Structures.

Extension Owner : Mike Baiocchi

6.14.2 Define Documentation

6.14.2.1 #define ECMD_GIP_CAPI_VERSION "1.0"

eCMD gip Extension version

6.14.2.2 #define HANDLE_SYSV_PRAGMA 1

6.14.3 Enumeration Type Documentation

6.14.3.1 enum gipMainstoreAddrType_t

Enum used to identify Mainstore Address Type.

Enumeration values:

GIP_MAINSTORE_REAL_ADDR Real Address.

GIP_MAINSTORE_EFFECTIVE_ADDR Effective Address.

GIP_MAINSTORE_VIRTUAL_ADDR Virtual Address.

GIP_MAINSTORE_RMO_ADDR RMO Address.

GIP_MAINSTORE_VRM_ADDR Virtual Real Memory Address.

6.15 zseClientCapi.H File Reference

Z Series eCMD Extension.

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

Load/Unload Functions

- uint32_t **zseInitExtension** ()
  Initialize eCMD zse Extension DLL.

6.15.1 Detailed Description

Z Series eCMD Extension.

Extension Owner : Hans-Joachim Hartmann

6.15.2 Function Documentation

6.15.2.1 uint32_t zseInitExtension ()

Initialize eCMD zse Extension DLL.

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 zse Extension is initialized and version checked

6.16 zseStructs.H File Reference

Z Series eCMD Extension Structures.

Defines

- #define **ECMD_ZSE_CAPI_VERSION** "1.0"
  eCMD zse Extension version

6.16.1 Detailed Description

Z Series eCMD Extension Structures.

Extension Owner : Hans-Joachim Hartmann

6.16.2 Define Documentation

6.16.2.1 #define ECMD_ZSE_CAPI_VERSION "1.0"

eCMD zse Extension version

Index

~ecmdDataBuffer
    ecmdDataBuffer, 36
~ecmdOptimizableDataBuffer
    ecmdOptimizableDataBuffer, 88

actThread
    cpcfSysInfo_t, 12
address
    ecmdArrayEntry, 18
    ecmdMemoryEntry, 84
    ecmdRingData, 91
    ecmdScomData, 93
addrType
    gipXlateVariables, 103
applyInversionMask
    ecmdDataBuffer, 47
applyRawBufferToXstate
    ecmdDataBufferImplementationHelper, 72
arrayName
    ecmdArrayData, 16

bitLength
    ecmdLatchData, 77
    ecmdProcRegisterInfo, 89
    ecmdRingData, 91
    ecmdSpyData, 96
boxType
    cpcfSysInfo_t, 12
brkPtHit
    cpcfSysInfo_t, 12
buffer
    ecmdArrayEntry, 18
    ecmdIndexEntry, 75
    ecmdIndexVectorEntry, 76
    ecmdLatchEntry, 79
    ecmdNameEntry, 85
    ecmdNameVectorEntry, 86

cage
    ecmdChipTarget, 24
CAGE_HDR_MAGIC
    ecmdStructs.H, 242
cageData
    ecmdQueryData, 90
cageId
    ecmdCageData, 19
cageState
    ecmdChipTarget, 25
CHIP_HDR_MAGIC
    ecmdStructs.H, 242
chipCommonType
    ecmdChipData, 21
chipData
    ecmdSlotData, 95
chipEc
    ecmdChipData, 21
chipFlags
    ecmdChipData, 21
chipShortType
    ecmdChipData, 21
chipType
    ecmdChipData, 21
    ecmdChipTarget, 24
chipTypeState
    ecmdChipTarget, 25
CIP_BREAKPOINT_CIABR
    cipStructs.H, 117
CIP_BREAKPOINT_DABR
    cipStructs.H, 117
CIP_BREAKPOINT_IABR
    cipStructs.H, 117
cipClearBreakpoint
    cipClientCapi.H, 110
cipClientCapi.H, 105
cipClientCapi.H
    cipClearBreakpoint, 110
    cipGetMemMemCtrl, 114
    cipGetMemProc, 113
    cipGetVr, 111
    cipGetVrMultiple, 111
    cipInitExtension, 107
    cipPutMemMemCtrl, 115
    cipPutMemProc, 114
    cipPutVr, 112
    cipPutVrMultiple, 112
    cipSetBreakpoint, 110
    cipStartAllInstructions, 107
    cipStartAllInstructionsSreset, 108
    cipStartInstructions, 107

cipStartInstructionsSreset, 108
cipStepInstructions, 109
cipStopAllInstructions, 109
cipStopInstructions, 108
cipGetMemMemCtrl
cipClientCapi.H, 114
cipGetMemProc
cipClientCapi.H, 113
cipGetVr
cipClientCapi.H, 111
cipGetVrMultiple
cipClientCapi.H, 111
cipInitExtension
cipClientCapi.H, 107
cipPutMemMemCtrl
cipClientCapi.H, 115
cipPutMemProc
cipClientCapi.H, 114
cipPutVr
cipClientCapi.H, 112
cipPutVrMultiple
cipClientCapi.H, 112
cipSetBreakpoint
cipClientCapi.H, 110
cipStartAllInstructions
cipClientCapi.H, 107
cipStartAllInstructionsSreset
cipClientCapi.H, 108
cipStartInstructions
cipClientCapi.H, 107
cipStartInstructionsSreset
cipClientCapi.H, 108
cipStepInstructions
cipClientCapi.H, 109
cipStopAllInstructions
cipClientCapi.H, 109
cipStopInstructions
cipClientCapi.H, 108
cipStructs.H, 117
CIP_BREAKPOINT_CIABR, 117
CIP_BREAKPOINT_DABR, 117
CIP_BREAKPOINT_IABR, 117
cipStructs.H
ECMD_CIP_CAPI_VERSION, 117
ecmdBreakpointType_t, 117
clear
ecmdDataBuffer, 36
clearBit
ecmdDataBuffer, 42
clkDomState
cpcfSysInfo_t, 12
clockDomain
ecmdArrayData, 17
ecmdLatchData, 77
ecmdRingData, 92
ecmdScomData, 93
ecmdSpyData, 97
ecmdTraceArrayData, 100
clockState
ecmdArrayData, 17
ecmdLatchData, 78
ecmdRingData, 92
ecmdScomData, 93
ecmdSpyData, 97
ecmdTraceArrayData, 101
cmdClientCapi.H, 118
cmdClientCapi.H
cmdInitExtension, 118
cmdRunCommand, 118
cmdRunCommandCaptureOutput, 119
cmdInitExtension
cmdClientCapi.H, 118
cmdRunCommand
cmdClientCapi.H, 118
cmdRunCommandCaptureOutput
cmdClientCapi.H, 119
cmdStructs.H, 120
cmdStructs.H
ECMD_CMD_CAPI_VERSION, 120
concat
ecmdDataBuffer, 52
copy
ecmdDataBuffer, 56
core
ecmdChipTarget, 24
CORE_HDR_MAGIC
ecmdStructs.H, 242
coreData
ecmdChipData, 21
coreId
ecmdCoreData, 26
coreState
ecmdChipTarget, 25
cpcfSysInfo_t, 11
cpcfSysInfo_t
actThread, 12
boxType, 12
brkPtHit, 12
clkDomState, 12
cpuCtlsVersion, 12
errorState, 12
flatten, 12
iopType, 12
iSeries, 12
olcRcvd, 12
primaryProc, 12
procExist, 12
procFunct, 12

procType, 12
puInstState, 12
puInstValid, 12
reserved0, 12
reserved1, 12
reserved2, 12
reservedA, 12
unflatten, 12
cpuCtlsVersion
cpcfSysInfo_t, 12
CRO_JTAG_NOTAPSTATEMOVE
croStructs.H, 132
CRO_JTAG_SHIFTDR
croStructs.H, 132
croClearDebug
croClientCapi.H, 124
croClientCapi.H, 121
croClientCapi.H
croClearDebug, 124
croDisplayVersion, 123
croFastLoadL2, 127
croFastLoadL2RingImage, 127
croGetConfiguration, 126
croGetL2, 128
croGetL2Data, 128
croGetL2Dir, 128
croGetL2Tag, 129
croInitExtension, 123
croIsDebugOn, 124
croJtagScanRead, 124
croJtagScanReadWrite, 125
croJtagScanWrite, 125
croPutL2, 129
croPutL2Data, 130
croPutL2Dir, 130
croPutL2Tag, 130
croRamInstruction, 127
croReset, 123
croSetConfiguration, 126
croSetDebug, 123
croDisplayVersion
croClientCapi.H, 123
croEclipzL2Fields, 13
croEclipzL2Fields
i_classbits, 13
i_endaddress, 13
i_endcongclass, 13
i_mesibits, 13
i_set, 13
i_slice, 13
i_startcongclass, 13
i_tagbits, 13
croEclipzPlusL2Fields, 14
croEclipzPlusL2Fields
undefined, 14
croFastLoadL2
croClientCapi.H, 127
croFastLoadL2RingImage
croClientCapi.H, 127
croGetConfiguration
croClientCapi.H, 126
croGetL2
croClientCapi.H, 128
croGetL2Data
croClientCapi.H, 128
croGetL2Dir
croClientCapi.H, 128
croGetL2Tag
croClientCapi.H, 129
croInitExtension
croClientCapi.H, 123
croIsDebugOn
croClientCapi.H, 124
croJtagScanMode
croStructs.H, 132
croJtagScanRead
croClientCapi.H, 124
croJtagScanReadWrite
croClientCapi.H, 125
croJtagScanWrite
croClientCapi.H, 125
croL2Fields, 15
croL2Fields
dllProduct, 15
eclipz, 15
eclipzplus, 15
croPutL2
croClientCapi.H, 129
croPutL2Data
croClientCapi.H, 130
croPutL2Dir
croClientCapi.H, 130
croPutL2Tag
croClientCapi.H, 130
croRamInstruction
croClientCapi.H, 127
croReset
croClientCapi.H, 123
croSetConfiguration
croClientCapi.H, 126
croSetDebug
croClientCapi.H, 123
croStructs.H, 132
CRO_JTAG_NOTAPSTATEMOVE, 132
CRO_JTAG_SHIFTDR, 132
croStructs.H
croJtagScanMode, 132

ECMD_CRO_CAPI_VERSION, 132
curUnitIdTarget
ecmdLooperData, 83

data
ecmdMemoryEntry, 84
deadbitsMask
ecmdSpyGroupData, 98
decToUInt32
ecmdUtils.H, 254
detailLevel
ecmdQueryData, 90
disableXstateBuffer
ecmdDataBuffer, 64
dllBuildDate
ecmdDllInfo, 74
dllBuildInfo
ecmdDllInfo, 74
dllCapiVersion
ecmdDllInfo, 74
dllEnv
ecmdDllInfo, 73
dllProduct
croL2Fields, 15
ecmdDllInfo, 73
dllProductType
ecmdDllInfo, 73
dllType
ecmdDllInfo, 73

eclipz
croL2Fields, 15
eclipzplus
croL2Fields, 15
ECMD_ALL_TARGETS_LOOP
ecmdStructs.H, 245
ECMD_APPEND_MODE
ecmdDataBuffer.H, 217
ECMD_CACHE_LEVEL1D
ecmdStructs.H, 243
ECMD_CACHE_LEVEL1I
ecmdStructs.H, 243
ECMD_CACHE_LEVEL2
ecmdStructs.H, 243
ECMD_CACHE_LEVEL3
ecmdStructs.H, 243
ECMD_CACHE_LEVEL4
ecmdStructs.H, 243
ECMD_CACHE_UNKNOWN
ecmdStructs.H, 243
ECMD_CAPI_VERSION
ecmdStructs.H, 242
ECMD_CHIPFLAG_BUSMASK
ecmdStructs.H, 242
ECMD_CHIPFLAG_FSI
ecmdStructs.H, 243
ECMD_CHIPFLAG_JTAG
ecmdStructs.H, 242
ECMD_CHIPFLAG_RSVDBUS1
ecmdStructs.H, 242
ECMD_CHIPFLAG_RSVDBUS2
ecmdStructs.H, 243
ECMD_CHIPT_IOBDG
ecmdStructs.H, 242
ECMD_CHIPT_IOHUB
ecmdStructs.H, 242
ECMD_CHIPT_MEM_BUF
ecmdStructs.H, 242
ECMD_CHIPT_MEM_CNTRL
ecmdStructs.H, 242
ECMD_CHIPT_MEM_L2CACHE
ecmdStructs.H, 242
ECMD_CHIPT_MEM_L3CACHE
ecmdStructs.H, 242
ECMD_CHIPT_MUX
ecmdStructs.H, 242
ECMD_CHIPT_PROCESSOR
ecmdStructs.H, 242
ECMD_CHIPT_SERVICE_PROCESSOR
ecmdStructs.H, 242
ECMD_CIP_CAPI_VERSION
cipStructs.H, 117
ECMD_CLOCK_CYCLETIME_SPEC
ecmdStructs.H, 244
ECMD_CLOCK_FREQUENCY_SPEC
ecmdStructs.H, 244
ECMD_CLOCK_ONE_STEP
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_DEFAULT
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_HIGH
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_HIGHEST
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_LOW
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_LOWEST
ecmdStructs.H, 244
ECMD_CLOCK_RANGE_MIDDLE
ecmdStructs.H, 244
ECMD_CLOCK_STEER
ecmdStructs.H, 244
ECMD_CLOCKS_ALREADY_OFF
ecmdReturnCodes.H, 226
ECMD_CLOCKS_ALREADY_ON
ecmdReturnCodes.H, 226
ECMD_CLOCKS_IN_INVALID_STATE
ecmdReturnCodes.H, 225

ECMD_CLOCKSTATE_NA
ecmdStructs.H, 244
ECMD_CLOCKSTATE_OFF
ecmdStructs.H, 244
ECMD_CLOCKSTATE_ON
ecmdStructs.H, 244
ECMD_CLOCKSTATE_UNKNOWN
ecmdStructs.H, 244
ECMD_CMD_CAPI_VERSION
cmdStructs.H, 120
ECMD_CONFIG_VALID_FIELD_-
ALPHA
ecmdStructs.H, 245
ECMD_CONFIG_VALID_FIELD_BOTH
ecmdStructs.H, 245
ECMD_CONFIG_VALID_FIELD_NONE
ecmdStructs.H, 245
ECMD_CONFIG_VALID_FIELD_-
NUMERIC
ecmdStructs.H, 245
ECMD_CRO_CAPI_VERSION
croStructs.H, 132
ECMD_DATA_BOUNDS_OVERFLOW
ecmdReturnCodes.H, 228
ECMD_DATA_OVERFLOW
ecmdReturnCodes.H, 225
ECMD_DATA_UNDERFLOW
ecmdReturnCodes.H, 225
ECMD_DBUF_BUFFER_OVERFLOW
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_DATANUMBER_NOT_-
FOUND
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_FILE_FORMAT_-
MISMATCH
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_FILE_OPERATION_-
FAIL
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_FOPEN_FAIL
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_INIT_FAIL
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_INVALID_ARGS
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_INVALID_DATA_-
FORMAT
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_NOT_OWNER
ecmdDataBuffer.H, 215
ecmdReturnCodes.H, 230
ECMD_DBUF_SUCCESS
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 228
ECMD_DBUF_UNDEFINED_-
FUNCTION
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_XSTATE_ERROR
ecmdDataBuffer.H, 214
ecmdReturnCodes.H, 229
ECMD_DBUF_XSTATE_NOT_-
ENABLED
ecmdDataBuffer.H, 215
ecmdReturnCodes.H, 230
ECMD_DEPTH_CAGE
ecmdSharedUtils.H, 232
ECMD_DEPTH_CHIP
ecmdSharedUtils.H, 232
ECMD_DEPTH_CORE
ecmdSharedUtils.H, 232
ECMD_DEPTH_NODE
ecmdSharedUtils.H, 232
ECMD_DEPTH_SLOT
ecmdSharedUtils.H, 232
ECMD_DEPTH_THREAD
ecmdSharedUtils.H, 232
ECMD_DIO_INPUT
ecmdStructs.H, 245
ECMD_DIO_OPEN_DRAIN
ecmdStructs.H, 245
ECMD_DIO_OPEN_SOURCE
ecmdStructs.H, 246
ECMD_DIO_PUSH_PULL
ecmdStructs.H, 246
ECMD_DISPLAY_TARGET_DEFAULT
ecmdSharedUtils.H, 232
ECMD_DLL_CRONUS
ecmdStructs.H, 246
ECMD_DLL_ENV_HW
ecmdStructs.H, 246
ECMD_DLL_ENV_SIM
ecmdStructs.H, 246
ECMD_DLL_INVALID
ecmdReturnCodes.H, 224
ECMD_DLL_IPSERIES
ecmdStructs.H, 246
ECMD_DLL_LOAD_FAILURE
ecmdReturnCodes.H, 223
ECMD_DLL_PRODUCT_ECLIPZ

ecmdStructs.H, 246
ECMD_DLL_PRODUCT_UNKNOWN
ecmdStructs.H, 246
ECMD_DLL_SCAND
ecmdStructs.H, 246
ECMD_DLL_STUB
ecmdStructs.H, 246
ECMD_DLL_UNINITIALIZED
ecmdReturnCodes.H, 224
ECMD_DLL_UNKNOWN
ecmdStructs.H, 246
ECMD_DLL_UNLOAD_FAILURE
ecmdReturnCodes.H, 223
ECMD_DLL_ZSERIES
ecmdStructs.H, 246
ECMD_ERR_CRONUS
ecmdReturnCodes.H, 223
ECMD_ERR_ECMD
ecmdReturnCodes.H, 223
ECMD_ERR_IP
ecmdReturnCodes.H, 223
ECMD_ERR_UNKNOWN
ecmdReturnCodes.H, 223
ECMD_ERR_Z
ecmdReturnCodes.H, 223
ECMD_EXPECT_FAILURE
ecmdReturnCodes.H, 228
ECMD_EXTENSION_NOT_-
SUPPORTED
ecmdReturnCodes.H, 227
ECMD_FAILURE
ecmdReturnCodes.H, 224
ECMD_FILE_ARRAYDEF
ecmdStructs.H, 246
ECMD_FILE_HELPTEXT
ecmdStructs.H, 246
ECMD_FILE_SCANDEF
ecmdStructs.H, 246
ECMD_FILE_SCANDEFHASH
ecmdStructs.H, 246
ECMD_FILE_SCOMDATA
ecmdStructs.H, 246
ECMD_FILE_SPYDEF
ecmdStructs.H, 246
ECMD_FILE_SPYDEFHASH
ecmdStructs.H, 246
ECMD_FUNCTION_NOT_SUPPORTED
ecmdReturnCodes.H, 224
ECMD_GIP_CAPI_VERSION
gipStructs.H, 263
ECMD_GLOBALVAR_DEBUG
ecmdStructs.H, 247
ECMD_GLOBALVAR_QUIETMODE
ecmdStructs.H, 247
ECMD_I2C_BUSSPEED_100KHZ
ecmdStructs.H, 247
ECMD_I2C_BUSSPEED_400KHZ
ecmdStructs.H, 247
ECMD_I2C_BUSSPEED_50KHZ
ecmdStructs.H, 247
ECMD_INSTRUCTIONS_IN_-
INVALID_STATE
ecmdReturnCodes.H, 227
ECMD_INT_UNKNOWN_COMMAND
ecmdReturnCodes.H, 228
ECMD_INTERFACE_ACCESS
ecmdStructs.H, 243
ECMD_INTERFACE_CFAM
ecmdStructs.H, 243
ECMD_INTERFACE_UNKNOWN
ecmdStructs.H, 243
ECMD_INVALID_ARGS
ecmdReturnCodes.H, 224
ECMD_INVALID_ARRAY
ecmdReturnCodes.H, 225
ECMD_INVALID_CLOCK_DOMAIN
ecmdReturnCodes.H, 226
ECMD_INVALID_CONFIG
ecmdReturnCodes.H, 225
ECMD_INVALID_CONFIG_NAME
ecmdReturnCodes.H, 226
ECMD_INVALID_DLL_FILENAME
ecmdReturnCodes.H, 223
ECMD_INVALID_DLL_VERSION
ecmdReturnCodes.H, 223
ECMD_INVALID_ENTRY_-
REQUESTED
ecmdReturnCodes.H, 227
ECMD_INVALID_FPR
ecmdReturnCodes.H, 226
ECMD_INVALID_GPR
ecmdReturnCodes.H, 226
ECMD_INVALID_LATCHNAME
ecmdReturnCodes.H, 226
ECMD_INVALID_MEMORY_ADDRESS
ecmdReturnCodes.H, 228
ECMD_INVALID_RETURN_DATA
ecmdReturnCodes.H, 228
ECMD_INVALID_RING
ecmdReturnCodes.H, 225
ECMD_INVALID_SPR
ecmdReturnCodes.H, 225
ECMD_INVALID_SPY
ecmdReturnCodes.H, 225
ECMD_INVALID_SPY_ENUM
ecmdReturnCodes.H, 224
ECMD_ISTEPS_INVALID_STEP
ecmdReturnCodes.H, 227

ECMD_LATCHMODE_FULL
ecmdStructs.H, 248
ECMD_LATCHMODE_PARTIAL
ecmdStructs.H, 248
ECMD_MEMCTRL_REFCLOCK
ecmdStructs.H, 244
ECMD_NON_FSI_CHIP
ecmdReturnCodes.H, 225
ECMD_NON_JTAG_CHIP
ecmdReturnCodes.H, 225
ECMD_POLLING_FAILURE
ecmdReturnCodes.H, 227
ECMD_PROC_REFCLOCK
ecmdStructs.H, 244
ECMD_QUERY_DETAIL_HIGH
ecmdStructs.H, 248
ECMD_QUERY_DETAIL_LOW
ecmdStructs.H, 248
ECMD_RETRY_WITH_VIRTUAL_-
ADDR
ecmdReturnCodes.H, 228
ECMD_RING_CACHE_ENABLED
ecmdReturnCodes.H, 226
ECMD_SAVE_FORMAT_ASCII
ecmdDataBuffer.H, 216
ECMD_SAVE_FORMAT_BINARY
ecmdDataBuffer.H, 216
ECMD_SAVE_FORMAT_BINARY_-
DATA
ecmdDataBuffer.H, 216
ECMD_SAVE_FORMAT_XSTATE
ecmdDataBuffer.H, 216
ECMD_SCANDEF_LOOKUP_FAILURE
ecmdReturnCodes.H, 228
ECMD_SCANDEFHASH_MULT_RINGS
ecmdReturnCodes.H, 227
ECMD_SCOMADDRESS_NOT_FOUND
ecmdReturnCodes.H, 227
ECMD_SELECTED_TARGETS_LOOP
ecmdStructs.H, 245
ECMD_SELECTED_TARGETS_-
LOOP_DEFALL
ecmdStructs.H, 245
ECMD_SELECTED_TARGETS_-
LOOP_VD
ecmdStructs.H, 245
ECMD_SELECTED_TARGETS_-
LOOP_VD_DEFALL
ecmdStructs.H, 245
ECMD_SIM_ERROR
ecmdStructs.H, 247
ECMD_SIM_INFO
ecmdStructs.H, 247
ECMD_SIM_MSG_ALWAYS
ecmdStructs.H, 247
ECMD_SIM_MSG_BROADSIDE
ecmdStructs.H, 247
ECMD_SIM_MSG_CMD_EXE
ecmdStructs.H, 247
ECMD_SIM_MSG_CMD_RS
ecmdStructs.H, 247
ECMD_SIM_MSG_DEBUG
ecmdStructs.H, 247
ECMD_SIM_MSG_END_SD_MSGS
ecmdStructs.H, 247
ECMD_SIM_MSG_EXCEPTION
ecmdStructs.H, 247
ECMD_SIM_MSG_TESTCASE
ecmdStructs.H, 247
ECMD_SIM_PLAIN
ecmdStructs.H, 247
ECMD_SIM_WARNING
ecmdStructs.H, 247
ECMD_SPY_FAILED_ECC_CHECK
ecmdReturnCodes.H, 224
ECMD_SPY_GROUP_MISMATCH
ecmdReturnCodes.H, 226
ECMD_SPY_IS_EDIAL
ecmdReturnCodes.H, 225
ECMD_SPY_NOT_ENUMERATED
ecmdReturnCodes.H, 224
ECMD_SPYTYPE_ALIAS
ecmdStructs.H, 248
ECMD_SPYTYPE_ECCGROUP
ecmdStructs.H, 248
ECMD_SPYTYPE_EDIAL
ecmdStructs.H, 248
ECMD_SPYTYPE_IDIAL
ecmdStructs.H, 248
ECMD_STATIC_DEPTH_LOOP
ecmdStructs.H, 245
ECMD_SUCCESS
ecmdReturnCodes.H, 223
ECMD_TARGET_FIELD_UNUSED
ecmdStructs.H, 243
ECMD_TARGET_FIELD_VALID
ecmdStructs.H, 243
ECMD_TARGET_FIELD_WILDCARD
ecmdStructs.H, 243
ECMD_TARGET_INVALID_TYPE
ecmdReturnCodes.H, 227
ECMD_TARGET_NOT_CONFIGURED
ecmdReturnCodes.H, 224
ECMD_TARGET_THREAD_ALIVE
ecmdStructs.H, 243
ECMD_TARGET_UNKNOWN_STATE
ecmdStructs.H, 243
ECMD_TRACE_PROCEDURE

ecmdStructs.H, 248
ECMD_TRACE_SCAN
ecmdStructs.H, 248
ECMD_UNABLE_TO_MAP_HASHID
ecmdReturnCodes.H, 228
ECMD_UNABLE_TO_OPEN_-ARRAYDEF
ecmdReturnCodes.H, 228
ECMD_UNABLE_TO_OPEN_-SCANDEF
ecmdReturnCodes.H, 226
ECMD_UNABLE_TO_OPEN_-SCANDEFHASH
ecmdReturnCodes.H, 227
ECMD_UNABLE_TO_OPEN_-SCOMDEF
ecmdReturnCodes.H, 227
ECMD_UNKNOWN_FILE
ecmdReturnCodes.H, 224
ECMD_WRITE_MODE
ecmdDataBuffer.H, 217
ECMD_ZSE_CAPI_VERSION
zseStructs.H, 266
ecmdArrayData, 16
ecmdArrayData
arrayName, 16
clockDomain, 17
clockState, 17
isCoreRelated, 17
length, 17
readAddressLength, 16
width, 17
writeAddressLength, 16
ecmdArrayEntry, 18
ecmdArrayEntry
address, 18
buffer, 18
rc, 18
ecmdBitsHeader
ecmdUtils.H, 254
ecmdBreakpointType_t
cipStructs.H, 117
ecmdCacheFlush
ecmdClientCapi.H, 184
ecmdCacheType_t
ecmdStructs.H, 243
ecmdCageData, 19
ecmdCageData
cageId, 19
nodeData, 19
unitId, 19
ecmdChipData, 20
ecmdChipData
chipCommonType, 21
chipEc, 21
chipFlags, 21
chipShortType, 21
chipType, 21
coreData, 21
interfaceType, 21
numProcCores, 21
pos, 21
simModelEc, 21
unitId, 21
ecmdChipInterfaceType_t
ecmdStructs.H, 243
ecmdChipTarget, 23
ecmdChipTarget
cage, 24
cageState, 25
chipType, 24
chipTypeState, 25
core, 24
coreState, 25
node, 24
nodeState, 25
pos, 24
posState, 25
slot, 24
slotState, 25
thread, 24
threadState, 25
unitId, 25
unitIdState, 25
ecmdChipTargetState_t
ecmdStructs.H, 243
ecmdClientCapi.H, 133
ecmdClientCapi.H
ecmdCacheFlush, 184
ecmdCommandArgs, 147
ecmdConfigureTarget, 201
ecmdDeconfigureTarget, 201
ecmdDelay, 199
ecmdDisableRingCache, 164
ecmdEnableRingCache, 163
ecmdFlushRegisteredErrorMsgs, 198
ecmdFlushRingCache, 164
ecmdGetConfiguration, 200
ecmdGetErrorMsg, 198
ecmdGetGlobalVar, 199
ecmdGpioConfigPin, 208
ecmdGpioReadLatch, 209
ecmdGpioReadPin, 208
ecmdGpioReadPins, 210
ecmdGpioWriteLatch, 209
ecmdGpioWriteLatches, 210
ecmdI2cRead, 206
ecmdI2cReadOffset, 206

ecmdI2cReset, 205
ecmdI2cWrite, 207
ecmdI2cWriteOffset, 207
ecmdIsRingCacheEnabled, 164
ecmdLoadDll, 146
ecmdOutput, 198
ecmdOutputError, 198
ecmdOutputWarning, 198
ecmdQueryArray, 151
ecmdQueryChipScandefVersion, 194
ecmdQueryChipSimModelVersion, 194
ecmdQueryClockState, 167
ecmdQueryConfig, 148
ecmdQueryDllInfo, 147
ecmdQueryFileLocation, 153
ecmdQueryLatch, 150
ecmdQueryProcRegisterInfo, 171
ecmdQueryRing, 150
ecmdQueryScom, 152
ecmdQuerySelected, 149
ecmdQuerySpy, 151
ecmdQueryTargetConfigured, 153
ecmdQueryTraceArray, 152
ecmdQueryTraceMode, 199
ecmdQueryVersionGreater, 148
ecmdRegisterErrorMsg, 198
ecmdSequenceIdToTarget, 203
ecmdSetClockMultDiv, 169
ecmdSetClockSpeed, 169
ecmdSetConfiguration, 200
ecmdSetTraceMode, 199
ecmdTargetToUnitId, 202
ecmdUnitIdStringToTarget, 202
ecmdUnitIdToString, 203
ecmdUnitIdToTarget, 202
ecmdUnloadDll, 147
getArray, 164
getArrayMultiple, 165
getCfamRegister, 159
getFpr, 176
getFprMultiple, 176
getGpr, 174
getGprMultiple, 174
getLatch, 155
getMemDma, 182
getMemMemCtrl, 183
getMemProc, 181
getModuleVpdImage, 204
getModuleVpdKeyword, 203
getRing, 153
getRingWithModifier, 156
getScom, 157
getSlb, 178
getSlbMultiple, 179
getSpr, 171
getSprMultiple, 172
getSpy, 160
getSpyEnum, 160
getSpyEpCheckers, 161
getSpyGroups, 162
getTraceArray, 180
getTraceArrayMultiple, 181
iStepsByName, 170
iStepsByNameMultiple, 170
iStepsByNameRange, 171
iStepsByNumber, 169
makeSPSystemCall, 199
putArray, 166
putArrayMultiple, 166
putCfamRegister, 159
putFpr, 177
putFprMultiple, 178
putGpr, 175
putGprMultiple, 175
putLatch, 155
putMemDma, 183
putMemMemCtrl, 184
putMemProc, 182
putModuleVpdImage, 205
putModuleVpdKeyword, 204
putRing, 154
putRingWithModifier, 157
putScom, 158
putSlb, 179
putSlbMultiple, 180
putSpr, 173
putSprMultiple, 173
putSpy, 162
putSpyEnum, 163
sendCmd, 159
simaet, 185
simCallFusionCommand, 194
simcheckpoint, 185
simclock, 185
simecho, 186
simexit, 186
simEXPECTFAC, 186
simexpecttcfac, 187
simFusionError, 146
simFusionInfo, 146
simFusionOut, 146
simFusionRand32, 195
simFusionRand64, 195
simFusionWarning, 146
simgetcurrentcycle, 187
simGetDial, 196
simGetEnvironment, 197
simGETFAC, 187

simGETFACX, 188
simGetHierarchy, 193
simGetInFile, 197
simGetModelInfo, 197
simGetOutFile, 197
simgettcfac, 188
siminit, 189
simOutputFusionMessage, 195
simPOLLFAC, 189
simpolltcfac, 189
simPutDial, 196
simPUTFAC, 190
simPUTFACX, 190
simputtcfac, 191
simrestart, 191
simSetFusionMessageFormat, 196
simSTKFAC, 191
simstktcfac, 192
simSUBCMD, 192
simtckinterval, 192
simUNSTICK, 193
simunsticktcfac, 193
startClocks, 167
stopClocks, 168
ecmdClockRange_t
ecmdStructs.H, 243
ecmdClockSetMode_t
ecmdStructs.H, 244
ecmdClockSpeedType_t
ecmdStructs.H, 244
ecmdClockState_t
ecmdStructs.H, 244
ecmdClockType_t
ecmdStructs.H, 244
ecmdCommandArgs
ecmdClientCapi.H, 147
ecmdConfigLooperInit
ecmdUtils.H, 253
ecmdConfigLooperNext
ecmdUtils.H, 253
ecmdConfigLoopMode_t
ecmdStructs.H, 244
ecmdConfigLoopType_t
ecmdStructs.H, 245
ecmdConfigureTarget
ecmdClientCapi.H, 201
ecmdConfigValid_t
ecmdStructs.H, 245
ecmdCoreData, 26
ecmdCoreData
coreId, 26
numProcThreads, 26
threadData, 26
unitId, 26
ecmdCurCage
ecmdLooperData, 82
ecmdCurChip
ecmdLooperData, 82
ecmdCurCore
ecmdLooperData, 82
ecmdCurNode
ecmdLooperData, 82
ecmdCurSlot
ecmdLooperData, 82
ecmdCurThread
ecmdLooperData, 82
ecmdDataBuffer, 27
ecmdDataBuffer, 36
ecmdDataBuffer
~ecmdDataBuffer, 36
applyInversionMask, 47
clear, 36
clearBit, 42
concat, 52
copy, 56
disableXstateBuffer, 64
ecmdDataBuffer, 36
ecmdDataBufferImplementationHelper, 70
enableXstateBuffer, 63
evenParity, 58, 59
extract, 49, 50
extractPreserve, 50
extractToRight, 51
fillDataStr, 70
flatten, 57
flattenSize, 58
flipBit, 42, 43
flushTo0, 46
flushTo1, 46
flushToX, 64
genAsciiStr, 60
genBinStr, 59, 60
genHexLeftStr, 59, 60
genHexRightStr, 59, 60
genXstateStr, 61
getBitLength, 37
getByte, 41
getByteLength, 37
getCapacity, 37
getDoubleWord, 42
getHalfWord, 41
getNumBitsSet, 44
getWord, 40
getWordLength, 36
getXstate, 65
growBitLength, 39
hasXstate, 64

insert, 47, 48
insertFromBin, 63
insertFromBinAndResize, 63
insertFromHexLeft, 61
insertFromHexLeftAndResize, 61
insertFromHexRight, 62
insertFromHexRightAndResize, 62
insertFromRight, 48, 49
invert, 46
isBitClear, 43, 44
isBitSet, 43
isBufferOptimizable, 39
isXstateEnabled, 64
iv_BufferOptimizable, 70
iv_Capacity, 70
iv_Data, 70
iv_DataStr, 71
iv_NumBits, 70
iv_NumWords, 70
iv_RealData, 70
iv_UserOwned, 70
iv_XstateEnabled, 71
memCopyIn, 56
memCopyInXstate, 66
memCopyOut, 57
memCopyOutXstate, 66
merge, 53
oddParity, 58
operator &, 70
operator!=, 69
operator=, 56
operator==, 69
operator|, 70
queryErrorState, 69
queryNumOfBuffers, 68
readFile, 67
readFileMultiple, 68
readFileStream, 69
reverse, 46
rotateLeft, 46
rotateRight, 45
setAnd, 55
setBit, 39
setBitLength, 38
setByte, 40
setByteLength, 37
setCapacity, 38
setDoubleWord, 41
setHalfWord, 41
setOr, 52, 53
setWord, 40
setWordLength, 37
setXor, 54
setXstate, 65
shareBuffer, 69
shiftLeft, 44
shiftLeftAndResize, 45
shiftRight, 44
shiftRightAndResize, 45
shrinkBitLength, 38
unflatten, 57
writeBit, 40
writeFile, 66
writeFileMultiple, 67
writeFileStream, 67
ecmdDataBuffer.H, 212
ECMD_APPEND_MODE, 217
ECMD_SAVE_FORMAT_ASCII, 216
ECMD_SAVE_FORMAT_BINARY, 216
ECMD_SAVE_FORMAT_BINARY_DATA, 216
ECMD_SAVE_FORMAT_XSTATE, 216
ECMD_WRITE_MODE, 217
ecmdDataBuffer.H
ECMD_DBUF_BUFFER_OVERFLOW, 214
ECMD_DBUF_DATANUMBER_NOT_FOUND, 214
ECMD_DBUF_FILE_FORMAT_MISMATCH, 214
ECMD_DBUF_FILE_OPERATION_FAIL, 214
ECMD_DBUF_FOPEN_FAIL, 214
ECMD_DBUF_INIT_FAIL, 214
ECMD_DBUF_INVALID_ARGS, 214
ECMD_DBUF_INVALID_DATA_FORMAT, 214
ECMD_DBUF_NOT_OWNER, 215
ECMD_DBUF_SUCCESS, 214
ECMD_DBUF_UNDEFINED_FUNCTION, 214
ECMD_DBUF_XSTATE_ERROR, 214
ECMD_DBUF_XSTATE_NOT_ENABLED, 215
ecmdFormatType_t, 216
ecmdWriteMode_t, 216
ETRAC0, 215
ETRAC1, 216
ETRAC2, 216
ETRAC3, 216
ETRAC4, 216
ETRAC5, 216
ETRAC6, 216
ETRAC7, 216
ETRAC8, 216

ETRAC9, 216
ecmdDataBufferImplementationHelper, 72
ecmdDataBuffer, 70
ecmdDataBufferImplementationHelper
applyRawBufferToXstate, 72
getDataPtr, 72
ecmdDeconfigureTarget
ecmdClientCapi.H, 201
ecmdDelay
ecmdClientCapi.H, 199
ecmdDioMode_t
ecmdStructs.H, 245
ecmdDisableRingCache
ecmdClientCapi.H, 164
ecmdDisplayDllInfo
ecmdUtils.H, 255
ecmdDisplayScomData
ecmdUtils.H, 255
ecmdDllEnv_t
ecmdStructs.H, 246
ecmdDllInfo, 73
ecmdDllInfo
dllBuildDate, 74
dllBuildInfo, 74
dllCapiVersion, 74
dllEnv, 73
dllProduct, 73
dllProductType, 73
dllType, 73
ecmdDllProduct_t
ecmdStructs.H, 246
ecmdDllType_t
ecmdStructs.H, 246
ecmdEnableRingCache
ecmdClientCapi.H, 163
ecmdFileType_t
ecmdStructs.H, 246
ecmdFlushRegisteredErrorMsgs
ecmdClientCapi.H, 198
ecmdFlushRingCache
ecmdClientCapi.H, 164
ecmdFormatType_t
ecmdDataBuffer.H, 216
ecmdFusionMessageType_t
ecmdStructs.H, 246
ecmdFusionSeverity_t
ecmdStructs.H, 247
ecmdGenB32FromHex
ecmdSharedUtils.H, 234
ecmdGenB32FromHexLeft
ecmdSharedUtils.H, 234
ecmdGenB32FromHexRight
ecmdSharedUtils.H, 234
ecmdGenEbcdic
ecmdSharedUtils.H, 233
ecmdGetChipData
ecmdUtils.H, 255
ecmdGetConfiguration
ecmdClientCapi.H, 200
ecmdGetErrorMsg
ecmdClientCapi.H, 198
ecmdGetGlobalVar
ecmdClientCapi.H, 199
ecmdGetSharedLibVersion
ecmdStructs.H, 249
ecmdGlobalVarType_t
ecmdStructs.H, 247
ecmdGpioConfigPin
ecmdClientCapi.H, 208
ecmdGpioReadLatch
ecmdClientCapi.H, 209
ecmdGpioReadPin
ecmdClientCapi.H, 208
ecmdGpioReadPins
ecmdClientCapi.H, 210
ecmdGpioWriteLatch
ecmdClientCapi.H, 209
ecmdGpioWriteLatches
ecmdClientCapi.H, 210
ecmdHashString32
ecmdSharedUtils.H, 235
ecmdHexToUInt32
ecmdSharedUtils.H, 235
ecmdI2cBusSpeed_t
ecmdStructs.H, 247
ecmdI2cRead
ecmdClientCapi.H, 206
ecmdI2cReadOffset
ecmdClientCapi.H, 206
ecmdI2cReset
ecmdClientCapi.H, 205
ecmdI2cWrite
ecmdClientCapi.H, 207
ecmdI2cWriteOffset
ecmdClientCapi.H, 207
ecmdIndexEntry, 75
ecmdIndexEntry
buffer, 75
index, 75
rc, 75
ecmdIndexVectorEntry, 76
ecmdIndexVectorEntry
buffer, 76
index, 76
rc, 76
ecmdIsRingCacheEnabled
ecmdClientCapi.H, 164
ecmdLatchData, 77

ecmdLatchData
bitLength, 77
clockDomain, 77
clockState, 78
isCoreRelated, 77
latchName, 77
ringName, 77
ecmdLatchEntry, 79
ecmdLatchEntry
buffer, 79
latchEndBit, 79
latchName, 79
latchStartBit, 79
rc, 80
ringName, 79
ecmdLatchMode_t
ecmdStructs.H, 247
ecmdLoadDll
ecmdClientCapi.H, 146
ecmdLooperData, 81
ecmdLooperData
curUnitIdTarget, 83
ecmdCurCage, 82
ecmdCurChip, 82
ecmdCurCore, 82
ecmdCurNode, 82
ecmdCurSlot, 82
ecmdCurThread, 82
ecmdLooperInitFlag, 82
ecmdLoopMode, 82
ecmdSystemConfigData, 82
ecmdUseUnitid, 82
prevTarget, 82
unitIdTargets, 82
ecmdLooperInitFlag
ecmdLooperData, 82
ecmdLoopMode
ecmdLooperData, 82
ecmdMemoryEntry, 84
ecmdMemoryEntry
address, 84
data, 84
tags, 84
ecmdNameEntry, 85
ecmdNameEntry
buffer, 85
name, 85
rc, 85
ecmdNameVectorEntry, 86
ecmdNameVectorEntry
buffer, 86
name, 86
rc, 86
ecmdNodeData, 87
ecmdNodeData
nodeId, 87
slotData, 87
unitId, 87
ecmdOptimizableDataBuffer, 88
ecmdOptimizableDataBuffer, 88
ecmdOptimizableDataBuffer
~ecmdOptimizableDataBuffer, 88
ecmdOptimizableDataBuffer, 88
ecmdOutput
ecmdClientCapi.H, 198
ecmdOutputError
ecmdClientCapi.H, 198
ecmdOutputWarning
ecmdClientCapi.H, 198
ecmdParseOption
ecmdSharedUtils.H, 233
ecmdParseOptionWithArgs
ecmdSharedUtils.H, 233
ecmdParseTokens
ecmdSharedUtils.H, 233
ecmdProcRegisterInfo, 89
ecmdProcRegisterInfo
bitLength, 89
threadReplicated, 89
totalEntries, 89
ecmdQueryArray
ecmdClientCapi.H, 151
ecmdQueryChipScandefVersion
ecmdClientCapi.H, 194
ecmdQueryChipSimModelVersion
ecmdClientCapi.H, 194
ecmdQueryClockState
ecmdClientCapi.H, 167
ecmdQueryConfig
ecmdClientCapi.H, 148
ecmdQueryData, 90
ecmdQueryData
cageData, 90
detailLevel, 90
ecmdQueryDetail_t
ecmdStructs.H, 248
ecmdQueryDllInfo
ecmdClientCapi.H, 147
ecmdQueryFileLocation
ecmdClientCapi.H, 153
ecmdQueryLatch
ecmdClientCapi.H, 150
ecmdQueryProcRegisterInfo
ecmdClientCapi.H, 171
ecmdQueryRing
ecmdClientCapi.H, 150
ecmdQueryScom
ecmdClientCapi.H, 152

ecmdQuerySelected
ecmdClientCapi.H, 149
ecmdQuerySpy
ecmdClientCapi.H, 151
ecmdQueryTargetConfigured
ecmdClientCapi.H, 153
ecmdQueryTraceArray
ecmdClientCapi.H, 152
ecmdQueryTraceMode
ecmdClientCapi.H, 199
ecmdQueryVersionGreater
ecmdClientCapi.H, 148
ecmdReadDataFormatted
ecmdUtils.H, 253
ecmdReadDcard
ecmdSharedUtils.H, 236
ecmdRegisterErrorMsg
ecmdClientCapi.H, 198
ecmdReturnCodes.H, 218
ecmdReturnCodes.H
ECMD_CLOCKS_ALREADY_OFF, 226
ECMD_CLOCKS_ALREADY_ON, 226
ECMD_CLOCKS_IN_INVALID_STATE, 225
ECMD_DATA_BOUNDS_OVERFLOW, 228
ECMD_DATA_OVERFLOW, 225
ECMD_DATA_UNDERFLOW, 225
ECMD_DBUF_BUFFER_OVERFLOW, 229
ECMD_DBUF_DATANUMBER_NOT_FOUND, 229
ECMD_DBUF_FILE_FORMAT_MISMATCH, 229
ECMD_DBUF_FILE_OPERATION_FAIL, 229
ECMD_DBUF_FOPEN_FAIL, 229
ECMD_DBUF_INIT_FAIL, 229
ECMD_DBUF_INVALID_ARGS, 229
ECMD_DBUF_INVALID_DATA_FORMAT, 229
ECMD_DBUF_NOT_OWNER, 230
ECMD_DBUF_SUCCESS, 228
ECMD_DBUF_UNDEFINED_FUNCTION, 229
ECMD_DBUF_XSTATE_ERROR, 229
ECMD_DBUF_XSTATE_NOT_ENABLED, 230
ECMD_DLL_INVALID, 224
ECMD_DLL_LOAD_FAILURE, 223
ECMD_DLL_UNINITIALIZED, 224
ECMD_DLL_UNLOAD_FAILURE, 223
ECMD_ERR_CRONUS, 223
ECMD_ERR_ECMD, 223
ECMD_ERR_IP, 223
ECMD_ERR_UNKNOWN, 223
ECMD_ERR_Z, 223
ECMD_EXPECT_FAILURE, 228
ECMD_EXTENSION_NOT_SUPPORTED, 227
ECMD_FAILURE, 224
ECMD_FUNCTION_NOT_SUPPORTED, 224
ECMD_INSTRUCTIONS_IN_INVALID_STATE, 227
ECMD_INT_UNKNOWN_COMMAND, 228
ECMD_INVALID_ARGS, 224
ECMD_INVALID_ARRAY, 225
ECMD_INVALID_CLOCK_DOMAIN, 226
ECMD_INVALID_CONFIG, 225
ECMD_INVALID_CONFIG_NAME, 226
ECMD_INVALID_DLL_FILENAME, 223
ECMD_INVALID_DLL_VERSION, 223
ECMD_INVALID_ENTRY_REQUESTED, 227
ECMD_INVALID_FPR, 226
ECMD_INVALID_GPR, 226
ECMD_INVALID_LATCHNAME, 226
ECMD_INVALID_MEMORY_ADDRESS, 228
ECMD_INVALID_RETURN_DATA, 228
ECMD_INVALID_RING, 225
ECMD_INVALID_SPR, 225
ECMD_INVALID_SPY, 225
ECMD_INVALID_SPY_ENUM, 224
ECMD_ISTEPS_INVALID_STEP, 227
ECMD_NON_FSI_CHIP, 225
ECMD_NON_JTAG_CHIP, 225
ECMD_POLLING_FAILURE, 227
ECMD_RETRY_WITH_VIRTUAL_ADDR, 228
ECMD_RING_CACHE_ENABLED, 226
ECMD_SCANDEF_LOOKUP_FAILURE, 228
ECMD_SCANDEFHASH_MULT_RINGS, 227

ECMD_SCOMADDRESS_NOT_FOUND, 227
ECMD_SPY_FAILED_ECC_CHECK, 224
ECMD_SPY_GROUP_MISMATCH, 226
ECMD_SPY_IS_EDIAL, 225
ECMD_SPY_NOT_ENUMERATED, 224
ECMD_SUCCESS, 223
ECMD_TARGET_INVALID_TYPE, 227
ECMD_TARGET_NOT_CONFIGURED, 224
ECMD_UNABLE_TO_MAP_HASHID, 228
ECMD_UNABLE_TO_OPEN_ARRAYDEF, 228
ECMD_UNABLE_TO_OPEN_SCANDEF, 226
ECMD_UNABLE_TO_OPEN_SCANDEFHASH, 227
ECMD_UNABLE_TO_OPEN_SCOMDEF, 227
ECMD_UNKNOWN_FILE, 224
ecmdRingData, 91
ecmdRingData
address, 91
bitLength, 91
clockDomain, 92
clockState, 92
hasInversionMask, 92
isCheckable, 92
isCoreRelated, 92
ringNames, 91
supportsBroadsideLoad, 92
ecmdScomData, 93
ecmdScomData
address, 93
clockDomain, 93
clockState, 93
isCoreRelated, 93
ecmdSequenceIdToTarget
ecmdClientCapi.H, 203
ecmdSetClockMultDiv
ecmdClientCapi.H, 169
ecmdSetClockSpeed
ecmdClientCapi.H, 169
ecmdSetConfiguration
ecmdClientCapi.H, 200
ecmdSetTargetDepth
ecmdSharedUtils.H, 235
ecmdSetTraceMode
ecmdClientCapi.H, 199
ecmdSharedUtils.H, 231
ECMD_DEPTH_CAGE, 232
ECMD_DEPTH_CHIP, 232
ECMD_DEPTH_CORE, 232
ECMD_DEPTH_NODE, 232
ECMD_DEPTH_SLOT, 232
ECMD_DEPTH_THREAD, 232
ECMD_DISPLAY_TARGET_DEFAULT, 232
ecmdSharedUtils.H
ecmdGenB32FromHex, 234
ecmdGenB32FromHexLeft, 234
ecmdGenB32FromHexRight, 234
ecmdGenEbcdic, 233
ecmdHashString32, 235
ecmdHexToUInt32, 235
ecmdParseOption, 233
ecmdParseOptionWithArgs, 233
ecmdParseTokens, 233
ecmdReadDcard, 236
ecmdSetTargetDepth, 235
ecmdTargetDepth_t, 232
ecmdTargetDisplayMode_t, 232
ecmdWriteTarget, 236
ecmdSimModelInfo, 94
ecmdSimModelInfo
modeldate, 94
modelname, 94
modeltime, 94
multivalue, 94
ecmdSlotData, 95
ecmdSlotData
chipData, 95
slotId, 95
unitId, 95
ecmdSpyData, 96
ecmdSpyData
bitLength, 96
clockDomain, 97
clockState, 97
enums, 97
epCheckers, 97
isCoreRelated, 97
isEccChecked, 97
isEnumerated, 97
spyName, 96
spyType, 97
ecmdSpyGroupData, 98
ecmdSpyGroupData
deadbitsMask, 98
extractBuffer, 98
ecmdSpyType_t
ecmdStructs.H, 248
ecmdStructs.H, 237

ECMD_ALL_TARGETS_LOOP, 245
ECMD_CACHE_LEVEL1D, 243
ECMD_CACHE_LEVEL1I, 243
ECMD_CACHE_LEVEL2, 243
ECMD_CACHE_LEVEL3, 243
ECMD_CACHE_LEVEL4, 243
ECMD_CACHE_UNKNOWN, 243
ECMD_CLOCK_CYCLETIME_-SPEC, 244
ECMD_CLOCK_FREQUENCY_-SPEC, 244
ECMD_CLOCK_ONE_STEP, 244
ECMD_CLOCK_RANGE_-DEFAULT, 244
ECMD_CLOCK_RANGE_HIGH, 244
ECMD_CLOCK_RANGE_-HIGHEST, 244
ECMD_CLOCK_RANGE_LOW, 244
ECMD_CLOCK_RANGE_LOWEST, 244
ECMD_CLOCK_RANGE_MIDDLE, 244
ECMD_CLOCK_STEER, 244
ECMD_CLOCKSTATE_NA, 244
ECMD_CLOCKSTATE_OFF, 244
ECMD_CLOCKSTATE_ON, 244
ECMD_CLOCKSTATE_UNKNOWN, 244
ECMD_CONFIG_VALID_FIELD_-ALPHA, 245
ECMD_CONFIG_VALID_FIELD_-BOTH, 245
ECMD_CONFIG_VALID_FIELD_-NONE, 245
ECMD_CONFIG_VALID_FIELD_-NUMERIC, 245
ECMD_DIO_INPUT, 245
ECMD_DIO_OPEN_DRAIN, 245
ECMD_DIO_OPEN_SOURCE, 246
ECMD_DIO_PUSH_PULL, 246
ECMD_DLL_CRONUS, 246
ECMD_DLL_ENV_HW, 246
ECMD_DLL_ENV_SIM, 246
ECMD_DLL_IPSERIES, 246
ECMD_DLL_PRODUCT_ECLIPZ, 246
ECMD_DLL_PRODUCT_-UNKNOWN, 246
ECMD_DLL_SCAND, 246
ECMD_DLL_STUB, 246
ECMD_DLL_UNKNOWN, 246
ECMD_DLL_ZSERIES, 246
ECMD_FILE_ARRAYDEF, 246
ECMD_FILE_HELPTEXT, 246
ECMD_FILE_SCANDEF, 246
ECMD_FILE_SCANDEFHASH, 246
ECMD_FILE_SCOMDATA, 246
ECMD_FILE_SPYDEF, 246
ECMD_FILE_SPYDEFHASH, 246
ECMD_GLOBALVAR_DEBUG, 247
ECMD_GLOBALVAR_-QUIETMODE, 247
ECMD_I2C_BUSSPEED_100KHZ, 247
ECMD_I2C_BUSSPEED_400KHZ, 247
ECMD_I2C_BUSSPEED_50KHZ, 247
ECMD_INTERFACE_ACCESS, 243
ECMD_INTERFACE_CFAM, 243
ECMD_INTERFACE_UNKNOWN, 243
ECMD_LATCHMODE_FULL, 248
ECMD_LATCHMODE_PARTIAL, 248
ECMD_MEMCTRL_REFCLOCK, 244
ECMD_PROC_REFCLOCK, 244
ECMD_QUERY_DETAIL_HIGH, 248
ECMD_QUERY_DETAIL_LOW, 248
ECMD_SELECTED_TARGETS_-LOOP, 245
ECMD_SELECTED_TARGETS_-LOOP_DEFALL, 245
ECMD_SELECTED_TARGETS_-LOOP_VD, 245
ECMD_SELECTED_TARGETS_-LOOP_VD_DEFALL, 245
ECMD_SIM_ERROR, 247
ECMD_SIM_INFO, 247
ECMD_SIM_MSG_ALWAYS, 247
ECMD_SIM_MSG_BROADSIDE, 247
ECMD_SIM_MSG_CMD_EXE, 247
ECMD_SIM_MSG_CMD_RS, 247
ECMD_SIM_MSG_DEBUG, 247
ECMD_SIM_MSG_END_SD_-MSGS, 247
ECMD_SIM_MSG_EXCEPTION, 247
ECMD_SIM_MSG_TESTCASE, 247
ECMD_SIM_PLAIN, 247
ECMD_SIM_WARNING, 247
ECMD_SPYTYPE_ALIAS, 248
ECMD_SPYTYPE_ECCGROUP, 248
ECMD_SPYTYPE_EDIAL, 248
ECMD_SPYTYPE_IDIAL, 248

ECMD_STATIC_DEPTH_LOOP, 245
ECMD_TARGET_FIELD_UNUSED, 243
ECMD_TARGET_FIELD_VALID, 243
ECMD_TARGET_FIELD_WILDCARD, 243
ECMD_TARGET_THREAD_ALIVE, 243
ECMD_TARGET_UNKNOWN_STATE, 243
ECMD_TRACE_PROCEDURE, 248
ECMD_TRACE_SCAN, 248
ecmdStructs.H
CAGE_HDR_MAGIC, 242
CHIP_HDR_MAGIC, 242
CORE_HDR_MAGIC, 242
ECMD_CAPI_VERSION, 242
ECMD_CHIPFLAG_BUSMASK, 242
ECMD_CHIPFLAG_FSI, 243
ECMD_CHIPFLAG_JTAG, 242
ECMD_CHIPFLAG_RSVDBUS1, 242
ECMD_CHIPFLAG_RSVDBUS2, 243
ECMD_CHIPT_IOBDG, 242
ECMD_CHIPT_IOHUB, 242
ECMD_CHIPT_MEM_BUF, 242
ECMD_CHIPT_MEM_CNTRL, 242
ECMD_CHIPT_MEM_L2CACHE, 242
ECMD_CHIPT_MEM_L3CACHE, 242
ECMD_CHIPT_MUX, 242
ECMD_CHIPT_PROCESSOR, 242
ECMD_CHIPT_SERVICE_PROCESSOR, 242
ecmdCacheType_t, 243
ecmdChipInterfaceType_t, 243
ecmdChipTargetState_t, 243
ecmdClockRange_t, 243
ecmdClockSetMode_t, 244
ecmdClockSpeedType_t, 244
ecmdClockState_t, 244
ecmdClockType_t, 244
ecmdConfigLoopMode_t, 244
ecmdConfigLoopType_t, 245
ecmdConfigValid_t, 245
ecmdDioMode_t, 245
ecmdDllEnv_t, 246
ecmdDllProduct_t, 246
ecmdDllType_t, 246
ecmdFileType_t, 246
ecmdFusionMessageType_t, 246
ecmdFusionSeverity_t, 247
ecmdGetSharedLibVersion, 249
ecmdGlobalVarType_t, 247
ecmdI2cBusSpeed_t, 247
ecmdLatchMode_t, 247
ecmdQueryDetail_t, 248
ecmdSpyType_t, 248
ecmdTraceType_t, 248
NODE_HDR_MAGIC, 242
operator<, 248, 249
QD_HDR_MAGIC, 242
SLOT_HDR_MAGIC, 242
THREAD_HDR_MAGIC, 242
ecmdSystemConfigData
ecmdLooperData, 82
ecmdTargetDepth_t
ecmdSharedUtils.H, 232
ecmdTargetDisplayMode_t
ecmdSharedUtils.H, 232
ecmdTargetToUnitId
ecmdClientCapi.H, 202
ecmdThreadData, 99
ecmdThreadData
threadId, 99
unitId, 99
ecmdTraceArrayData, 100
ecmdTraceArrayData
clockDomain, 100
clockState, 101
isCoreRelated, 100
length, 100
traceArrayName, 100
width, 100
ecmdTraceType_t
ecmdStructs.H, 248
ecmdUnitIdStringToTarget
ecmdClientCapi.H, 202
ecmdUnitIdToString
ecmdClientCapi.H, 203
ecmdUnitIdToTarget
ecmdClientCapi.H, 202
ecmdUnloadDll
ecmdClientCapi.H, 147
ecmdUseUnitid
ecmdLooperData, 82
ecmdUtils.H, 250
ecmdUtils.H
decToUInt32, 254
ecmdBitsHeader, 254
ecmdConfigLooperInit, 253
ecmdConfigLooperNext, 253
ecmdDisplayDllInfo, 255
ecmdDisplayScomData, 255
ecmdGetChipData, 255
ecmdReadDataFormatted, 253

ecmdWriteDataFormatted, 254
PTRAC0, 251
PTRAC1, 251
PTRAC2, 251
PTRAC3, 251
PTRAC4, 252
PTRAC5, 252
PTRAC6, 252
PTRAC7, 252
PTRAC8, 252
PTRAC9, 252
ecmdWriteDataFormatted
ecmdUtils.H, 254
ecmdWriteMode_t
ecmdDataBuffer.H, 216
ecmdWriteTarget
ecmdSharedUtils.H, 236
enableXstateBuffer
ecmdDataBuffer, 63
enums
ecmdSpyData, 97
epCheckers
ecmdSpyData, 97
errorState
cpcfSysInfo_t, 12
ETRAC0
ecmdDataBuffer.H, 215
ETRAC1
ecmdDataBuffer.H, 216
ETRAC2
ecmdDataBuffer.H, 216
ETRAC3
ecmdDataBuffer.H, 216
ETRAC4
ecmdDataBuffer.H, 216
ETRAC5
ecmdDataBuffer.H, 216
ETRAC6
ecmdDataBuffer.H, 216
ETRAC7
ecmdDataBuffer.H, 216
ETRAC8
ecmdDataBuffer.H, 216
ETRAC9
ecmdDataBuffer.H, 216
evenParity
ecmdDataBuffer, 58, 59
extract
ecmdDataBuffer, 49, 50
extractBuffer
ecmdSpyGroupData, 98
extractPreserve
ecmdDataBuffer, 50
extractToRight
ecmdDataBuffer, 51

fillDataStr
ecmdDataBuffer, 70
flatten
cpcfSysInfo_t, 12
ecmdDataBuffer, 57
flattenSize
ecmdDataBuffer, 58
flipBit
ecmdDataBuffer, 42, 43
flushTo0
ecmdDataBuffer, 46
flushTo1
ecmdDataBuffer, 46
flushToX
ecmdDataBuffer, 64

genAsciiStr
ecmdDataBuffer, 60
genBinStr
ecmdDataBuffer, 59, 60
genHexLeftStr
ecmdDataBuffer, 59, 60
genHexRightStr
ecmdDataBuffer, 59, 60
genXstateStr
ecmdDataBuffer, 61
getArray
ecmdClientCapi.H, 164
getArrayMultiple
ecmdClientCapi.H, 165
getBitLength
ecmdDataBuffer, 37
getByte
ecmdDataBuffer, 41
getByteLength
ecmdDataBuffer, 37
getCapacity
ecmdDataBuffer, 37
getCfamRegister
ecmdClientCapi.H, 159
getDataPtr
ecmdDataBufferImplementationHelper, 72
getDoubleWord
ecmdDataBuffer, 42
getFpr
ecmdClientCapi.H, 176
getFprMultiple
ecmdClientCapi.H, 176
getGpr
ecmdClientCapi.H, 174
getGprMultiple

ecmdClientCapi.H, 174
getHalfWord
ecmdDataBuffer, 41
getLatch
ecmdClientCapi.H, 155
getMemDma
ecmdClientCapi.H, 182
getMemMemCtrl
ecmdClientCapi.H, 183
getMemProc
ecmdClientCapi.H, 181
getModuleVpdImage
ecmdClientCapi.H, 204
getModuleVpdKeyword
ecmdClientCapi.H, 203
getNumBitsSet
ecmdDataBuffer, 44
getRing
ecmdClientCapi.H, 153
getRingWithModifier
ecmdClientCapi.H, 156
getScom
ecmdClientCapi.H, 157
getSlb
ecmdClientCapi.H, 178
getSlbMultiple
ecmdClientCapi.H, 179
getSpr
ecmdClientCapi.H, 171
getSprMultiple
ecmdClientCapi.H, 172
getSpy
ecmdClientCapi.H, 160
getSpyEnum
ecmdClientCapi.H, 160
getSpyEpCheckers
ecmdClientCapi.H, 161
getSpyGroups
ecmdClientCapi.H, 162
getTraceArray
ecmdClientCapi.H, 180
getTraceArrayMultiple
ecmdClientCapi.H, 181
getWord
ecmdDataBuffer, 40
getWordLength
ecmdDataBuffer, 36
getXstate
ecmdDataBuffer, 65
GIP_MAINSTORE_EFFECTIVE_ADDR
gipStructs.H, 264
GIP_MAINSTORE_REAL_ADDR
gipStructs.H, 264
GIP_MAINSTORE_RMO_ADDR
gipStructs.H, 264
GIP_MAINSTORE_VIRTUAL_ADDR
gipStructs.H, 264
GIP_MAINSTORE_VRM_ADDR
gipStructs.H, 264
gipAbort
gipClientCapi.H, 258
gipClearSoftwareBreakpoint
gipClientCapi.H, 261
gipClientCapi.H, 256
gipClientCapi.H
gipAbort, 258
gipClearSoftwareBreakpoint, 261
gipConnectionTest, 262
gipGetMemProcVariousAddrType, 259
gipGetSoftwareBreakpoint, 261
gipGetSystemInfo, 262
gipInitExtension, 257
gipPutMemProcVariousAddrType, 259
gipRelease, 258
gipReserve, 257
gipSetServAddr, 258
gipSetSoftwareBreakpoint, 260
gipConnectionTest
gipClientCapi.H, 262
gipGetMemProcVariousAddrType
gipClientCapi.H, 259
gipGetSoftwareBreakpoint
gipClientCapi.H, 261
gipGetSystemInfo
gipClientCapi.H, 262
gipInitExtension
gipClientCapi.H, 257
gipMainstoreAddrType_t
gipStructs.H, 263
gipPutMemProcVariousAddrType
gipClientCapi.H, 259
gipRelease
gipClientCapi.H, 258
gipReserve
gipClientCapi.H, 257
gipSetServAddr
gipClientCapi.H, 258
gipSetSoftwareBreakpoint
gipClientCapi.H, 260
gipStructs.H, 263
GIP_MAINSTORE_EFFECTIVE_-
ADDR, 264
GIP_MAINSTORE_REAL_ADDR,
264
GIP_MAINSTORE_RMO_ADDR,
264
GIP_MAINSTORE_VIRTUAL_-
ADDR, 264

GIP_MAINSTORE_VRM_ADDR, 264
gipStructs.H
ECMD_GIP_CAPI_VERSION, 263
gipMainstoreAddrType_t, 263
HANDLE_SYSV_PRAGMA, 263
gipXlateVariables, 102
gipXlateVariables
addrType, 103
littleEndian, 102
manualXlateFlag, 103
mode32bit, 102
partitionId, 103
tagsActive, 102
writeECC, 102
growBitLength
ecmdDataBuffer, 39

HANDLE_SYSV_PRAGMA
gipStructs.H, 263
hasInversionMask
ecmdRingData, 92
hasXstate
ecmdDataBuffer, 64

i_classbits
croEclipzL2Fields, 13
i_endaddress
croEclipzL2Fields, 13
i_endcongclass
croEclipzL2Fields, 13
i_mesibits
croEclipzL2Fields, 13
i_set
croEclipzL2Fields, 13
i_slice
croEclipzL2Fields, 13
i_startcongclass
croEclipzL2Fields, 13
i_tagbits
croEclipzL2Fields, 13
index
ecmdIndexEntry, 75
ecmdIndexVectorEntry, 76
insert
ecmdDataBuffer, 47, 48
insertFromBin
ecmdDataBuffer, 63
insertFromBinAndResize
ecmdDataBuffer, 63
insertFromHexLeft
ecmdDataBuffer, 61
insertFromHexLeftAndResize
ecmdDataBuffer, 61
insertFromHexRight
ecmdDataBuffer, 62
insertFromHexRightAndResize
ecmdDataBuffer, 62
insertFromRight
ecmdDataBuffer, 48, 49
interfaceType
ecmdChipData, 21
invert
ecmdDataBuffer, 46
iopType
cpcfSysInfo_t, 12
isBitClear
ecmdDataBuffer, 43, 44
isBitSet
ecmdDataBuffer, 43
isBufferOptimizable
ecmdDataBuffer, 39
isCheckable
ecmdRingData, 92
isCoreRelated
ecmdArrayData, 17
ecmdLatchData, 77
ecmdRingData, 92
ecmdScomData, 93
ecmdSpyData, 97
ecmdTraceArrayData, 100
isEccChecked
ecmdSpyData, 97
isEnumerated
ecmdSpyData, 97
iSeries
cpcfSysInfo_t, 12
iStepsByName
ecmdClientCapi.H, 170
iStepsByNameMultiple
ecmdClientCapi.H, 170
iStepsByNameRange
ecmdClientCapi.H, 171
iStepsByNumber
ecmdClientCapi.H, 169
isXstateEnabled
ecmdDataBuffer, 64
iv_BufferOptimizable
ecmdDataBuffer, 70
iv_Capacity
ecmdDataBuffer, 70
iv_Data
ecmdDataBuffer, 70
iv_DataStr
ecmdDataBuffer, 71
iv_NumBits
ecmdDataBuffer, 70
iv_NumWords

ecmdDataBuffer, 70
iv_RealData
ecmdDataBuffer, 70
iv_UserOwned
ecmdDataBuffer, 70
iv_XstateEnabled
ecmdDataBuffer, 71

latchEndBit
ecmdLatchEntry, 79
latchName
ecmdLatchData, 77
ecmdLatchEntry, 79
latchStartBit
ecmdLatchEntry, 79
length
ecmdArrayData, 17
ecmdTraceArrayData, 100
littleEndian
gipXlateVariables, 102

makeSPSystemCall
ecmdClientCapi.H, 199
manualXlateFlag
gipXlateVariables, 103
memCopyIn
ecmdDataBuffer, 56
memCopyInXstate
ecmdDataBuffer, 66
memCopyOut
ecmdDataBuffer, 57
memCopyOutXstate
ecmdDataBuffer, 66
merge
ecmdDataBuffer, 53
mode32bit
gipXlateVariables, 102
modeldate
ecmdSimModelInfo, 94
modelname
ecmdSimModelInfo, 94
modeltime
ecmdSimModelInfo, 94
multivalue
ecmdSimModelInfo, 94

name
ecmdNameEntry, 85
ecmdNameVectorEntry, 86
node
ecmdChipTarget, 24
NODE_HDR_MAGIC
ecmdStructs.H, 242
nodeData
ecmdCageData, 19
nodeId
ecmdNodeData, 87
nodeState
ecmdChipTarget, 25
numProcCores
ecmdChipData, 21
numProcThreads
ecmdCoreData, 26

oddParity
ecmdDataBuffer, 58
olcRcvd
cpcfSysInfo_t, 12
operator &
ecmdDataBuffer, 70
operator!=
ecmdDataBuffer, 69
operator<
ecmdStructs.H, 248, 249
operator=
ecmdDataBuffer, 56
operator==
ecmdDataBuffer, 69
operator|
ecmdDataBuffer, 70

partitionId
gipXlateVariables, 103
pos
ecmdChipData, 21
ecmdChipTarget, 24
posState
ecmdChipTarget, 25
prevTarget
ecmdLooperData, 82
primaryProc
cpcfSysInfo_t, 12
procExist
cpcfSysInfo_t, 12
procFunct
cpcfSysInfo_t, 12
procType
cpcfSysInfo_t, 12
PTRAC0
ecmdUtils.H, 251
PTRAC1
ecmdUtils.H, 251
PTRAC2
ecmdUtils.H, 251
PTRAC3
ecmdUtils.H, 251
PTRAC4
ecmdUtils.H, 252

PTRAC5
ecmdUtils.H, 252
PTRAC6
ecmdUtils.H, 252
PTRAC7
ecmdUtils.H, 252
PTRAC8
ecmdUtils.H, 252
PTRAC9
ecmdUtils.H, 252
puInstState
cpcfSysInfo_t, 12
puInstValid
cpcfSysInfo_t, 12
putArray
ecmdClientCapi.H, 166
putArrayMultiple
ecmdClientCapi.H, 166
putCfamRegister
ecmdClientCapi.H, 159
putFpr
ecmdClientCapi.H, 177
putFprMultiple
ecmdClientCapi.H, 178
putGpr
ecmdClientCapi.H, 175
putGprMultiple
ecmdClientCapi.H, 175
putLatch
ecmdClientCapi.H, 155
putMemDma
ecmdClientCapi.H, 183
putMemMemCtrl
ecmdClientCapi.H, 184
putMemProc
ecmdClientCapi.H, 182
putModuleVpdImage
ecmdClientCapi.H, 205
putModuleVpdKeyword
ecmdClientCapi.H, 204
putRing
ecmdClientCapi.H, 154
putRingWithModifier
ecmdClientCapi.H, 157
putScom
ecmdClientCapi.H, 158
putSlb
ecmdClientCapi.H, 179
putSlbMultiple
ecmdClientCapi.H, 180
putSpr
ecmdClientCapi.H, 173
putSprMultiple
ecmdClientCapi.H, 173
putSpy
ecmdClientCapi.H, 162
putSpyEnum
ecmdClientCapi.H, 163

QD_HDR_MAGIC
ecmdStructs.H, 242
queryErrorState
ecmdDataBuffer, 69
queryNumOfBuffers
ecmdDataBuffer, 68

rc
ecmdArrayEntry, 18
ecmdIndexEntry, 75
ecmdIndexVectorEntry, 76
ecmdLatchEntry, 80
ecmdNameEntry, 85
ecmdNameVectorEntry, 86
readAddressLength
ecmdArrayData, 16
readFile
ecmdDataBuffer, 67
readFileMultiple
ecmdDataBuffer, 68
readFileStream
ecmdDataBuffer, 69
reserved0
cpcfSysInfo_t, 12
reserved1
cpcfSysInfo_t, 12
reserved2
cpcfSysInfo_t, 12
reservedA
cpcfSysInfo_t, 12
reverse
ecmdDataBuffer, 46
ringName
ecmdLatchData, 77
ecmdLatchEntry, 79
ringNames
ecmdRingData, 91
rotateLeft
ecmdDataBuffer, 46
rotateRight
ecmdDataBuffer, 45

sendCmd
ecmdClientCapi.H, 159
setAnd
ecmdDataBuffer, 55
setBit
ecmdDataBuffer, 39
setBitLength

ecmdDataBuffer, 38
setByte
ecmdDataBuffer, 40
setByteLength
ecmdDataBuffer, 37
setCapacity
ecmdDataBuffer, 38
setDoubleWord
ecmdDataBuffer, 41
setHalfWord
ecmdDataBuffer, 41
setOr
ecmdDataBuffer, 52, 53
setWord
ecmdDataBuffer, 40
setWordLength
ecmdDataBuffer, 37
setXor
ecmdDataBuffer, 54
setXstate
ecmdDataBuffer, 65
shareBuffer
ecmdDataBuffer, 69
shiftLeft
ecmdDataBuffer, 44
shiftLeftAndResize
ecmdDataBuffer, 45
shiftRight
ecmdDataBuffer, 44
shiftRightAndResize
ecmdDataBuffer, 45
shrinkBitLength
ecmdDataBuffer, 38
simaet
ecmdClientCapi.H, 185
simCallFusionCommand
ecmdClientCapi.H, 194
simcheckpoint
ecmdClientCapi.H, 185
simclock
ecmdClientCapi.H, 185
simecho
ecmdClientCapi.H, 186
simexit
ecmdClientCapi.H, 186
simEXPECTFAC
ecmdClientCapi.H, 186
simexpecttcfac
ecmdClientCapi.H, 187
simFusionError
ecmdClientCapi.H, 146
simFusionInfo
ecmdClientCapi.H, 146
simFusionOut
ecmdClientCapi.H, 146
simFusionRand32
ecmdClientCapi.H, 195
simFusionRand64
ecmdClientCapi.H, 195
simFusionWarning
ecmdClientCapi.H, 146
simgetcurrentcycle
ecmdClientCapi.H, 187
simGetDial
ecmdClientCapi.H, 196
simGetEnvironment
ecmdClientCapi.H, 197
simGETFAC
ecmdClientCapi.H, 187
simGETFACX
ecmdClientCapi.H, 188
simGetHierarchy
ecmdClientCapi.H, 193
simGetInFile
ecmdClientCapi.H, 197
simGetModelInfo
ecmdClientCapi.H, 197
simGetOutFile
ecmdClientCapi.H, 197
simgettcfac
ecmdClientCapi.H, 188
siminit
ecmdClientCapi.H, 189
simModelEc
ecmdChipData, 21
simOutputFusionMessage
ecmdClientCapi.H, 195
simPOLLFAC
ecmdClientCapi.H, 189
simpolltcfac
ecmdClientCapi.H, 189
simPutDial
ecmdClientCapi.H, 196
simPUTFAC
ecmdClientCapi.H, 190
simPUTFACX
ecmdClientCapi.H, 190
simputtcfac
ecmdClientCapi.H, 191
simrestart
ecmdClientCapi.H, 191
simSetFusionMessageFormat
ecmdClientCapi.H, 196
simSTKFAC
ecmdClientCapi.H, 191
simstktcfac
ecmdClientCapi.H, 192
simSUBCMD

ecmdClientCapi.H, 192
simtckinterval
ecmdClientCapi.H, 192
simUNSTICK
ecmdClientCapi.H, 193
simunsticktcfac
ecmdClientCapi.H, 193
slot
ecmdChipTarget, 24
SLOT_HDR_MAGIC
ecmdStructs.H, 242
slotData
ecmdNodeData, 87
slotId
ecmdSlotData, 95
slotState
ecmdChipTarget, 25
spyName
ecmdSpyData, 96
spyType
ecmdSpyData, 97
startClocks
ecmdClientCapi.H, 167
stopClocks
ecmdClientCapi.H, 168
supportsBroadsideLoad
ecmdRingData, 92

tags
ecmdMemoryEntry, 84
tagsActive
gipXlateVariables, 102
thread
ecmdChipTarget, 24
THREAD_HDR_MAGIC
ecmdStructs.H, 242
threadData
ecmdCoreData, 26
threadId
ecmdThreadData, 99
threadReplicated
ecmdProcRegisterInfo, 89
threadState
ecmdChipTarget, 25
totalEntries
ecmdProcRegisterInfo, 89
traceArrayName
ecmdTraceArrayData, 100

undefined
croEclipzPlusL2Fields, 14
unflatten
cpcfSysInfo_t, 12
ecmdDataBuffer, 57
unitId
ecmdCageData, 19
ecmdChipData, 21
ecmdChipTarget, 25
ecmdCoreData, 26
ecmdNodeData, 87
ecmdSlotData, 95
ecmdThreadData, 99
unitIdState
ecmdChipTarget, 25
unitIdTargets
ecmdLooperData, 82

width
ecmdArrayData, 17
ecmdTraceArrayData, 100
writeAddressLength
ecmdArrayData, 16
writeBit
ecmdDataBuffer, 40
writeECC
gipXlateVariables, 102
writeFile
ecmdDataBuffer, 66
writeFileMultiple
ecmdDataBuffer, 67
writeFileStream
ecmdDataBuffer, 67

zseClientCapi.H, 265
zseClientCapi.H
zseInitExtension, 265
zseInitExtension
zseClientCapi.H, 265
zseStructs.H, 266
zseStructs.H
ECMD_ZSE_CAPI_VERSION, 266