eCMD C/C++ Dll Reference Manual

Generated by Doxygen 1.2.18

Tue Aug 10 10:47:47 2004

Contents

1 eCMD C/C++ Dll Main Page 1

2 eCMD C/C++ Dll Compound Index 7

2.1 eCMD C/C++ Dll Compound List 7

3 eCMD C/C++ Dll File Index 9

3.1 eCMD C/C++ Dll File List 9

4 eCMD C/C++ Dll Class Documentation 11

4.1 ecmdArrayData Struct Reference 11

4.2 ecmdArrayEntry Struct Reference 13

4.3 ecmdCageData Struct Reference 14

4.4 ecmdChipData Struct Reference 15

4.5 ecmdChipTarget Struct Reference 17

4.6 ecmdCoreData Struct Reference 20

4.7 ecmdDataBuffer Class Reference 21

4.8 ecmdDllInfo Struct Reference 45

4.9 ecmdGroupData Struct Reference 47

4.10 ecmdIndexEntry Struct Reference 48

4.11 ecmdLatchEntry Struct Reference 49

4.12 ecmdLooperData Struct Reference 51

4.13 ecmdNameEntry Struct Reference 53

4.14 ecmdNameVectorEntry Struct Reference 54

4.15 ecmdNodeData Struct Reference 55

4.16 ecmdProcRegisterInfo Struct Reference 56

4.17 ecmdQueryData Struct Reference 57

4.18 ecmdRingData Struct Reference 58

4.19 ecmdSlotData Struct Reference 60

4.20 ecmdSpyData Struct Reference 61

4.21 ecmdThreadData Struct Reference 63

5 eCMD C/C++ Dll File Documentation 65

5.1 cipClientCapi.H File Reference 65

5.2 ecmdClientCapi.H File Reference 68

5.3 ecmdDataBuffer.H File Reference 113

5.4 ecmdReturnCodes.H File Reference 114

5.5 ecmdStructs.H File Reference 122

5.6 ecmdUtils.H File Reference 130

Chapter 1

eCMD C/C++ Dll Main Page

1.0.1 Introduction

Common Hardware Access Programming Interface (eCMD)

This is the documentation of the eCMD C/C++ Programming Api

1.0.2 eCMD Core Include Files

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

- **ecmdClientCapi.H**
- **ecmdDataBuffer.H**
- **ecmdStructs.H**
- **ecmdReturnCodes.H**
- **ecmdUtils.H**
- ecmdSharedUtils.H

1.0.3 Link objects

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

- ecmdClientCapi_aix.a
- libecmd_aix.so
- xlC v5.0

To create Linux x86 binaries, the following is required:

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

1.0.4 eCMD Extensions

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

1.0.4.1 CIP (Cronus/IP) Extension

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

Include files :

- **cipClientCapi.H**

1.0.5 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.

1.0.6 The ecmdDataBuffer class

Data is passed between the client and the DLL with the **ecmdDataBuffer** (p. 21) class. The **ecmdDataBuffer** (p. 21) object is linked on both the client side and the DLL side.

The **ecmdDataBuffer** (p. 21) 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. 21) 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. 21) conversion-to-string call.

1.0.7 Makefile Example

These examples assume you linked to the required files in a subdir called dll.

For Cronus these files can be found in your location at .../cronus/ecommon/dll

1.0.7.1 Aix

```
testclient: testclient.o dll/ecmdClientCapi_aix.a
xlC -+ -g -brtl -L../capi/export -lecmd_aix testclient.o dll/ecmdClientCapi_aix.a -o testclient

testclient.o: testclient.C dll/ecmdClientCapi.H dll/ecmdDataBuffer.H dll/ecmdReturnCodes.H dll/ecmdStructs.H dl
xlC -+ -g -c -Idll/ testclient.C -o testclient.o
```

1.0.7.2 Linux x86

```
testclient.linux: testclient_linux.o dll/ecmdClientCapi_x86.a
g++ -g -ldl -L../capi/export -lecmd_x86 testclient_linux.o dll/ecmdClientCapi_x86.a -o testclient.linux

testclient_linux.o: testclient.c dll/ecmdClientCapi.H dll/ecmdDataBuffer.H dll/ecmdReturnCodes.H dll/ecmdStruct
g++ -g -c -Idll/  -ftemplate-depth-30 testclient.c -o testclient_linux.o
```

1.0.8 Example

```
&#35;include <list>
```

```
&#35;include <string>

&#35;include < ecmdClientCapi.H>
&#35;include < ecmdDataBuffer.H>


int main (int argc, char *argv[])
{

  // A buffer to store our data
   ecmdDataBuffer  (p. 21) data;
  uint32_t rc = 0;
  // This is the chip target to operate on
   ecmdChipTarget  (p. 17) target;


  // Load and initialize the eCMD Dll
  // Which DLL to load is determined by the ECMD_DLL_FILE environment variable
  rc = ecmdLoadDll("");
  if (rc) {
    printf("**** ERROR : Problems loading eCMD Dll!");
    return rc;
  }

  // Pass your arguments to the Dll so it can parse out any common args
  // Common args like -p# -c# will be removed from arg list upon return
  rc = ecmdCommandArgs(&argc, &argv);
  if (rc) return rc;

  // Let's setup our target
  target.cage = target.node = target.slot = 0;
  target.chipType = "pu";
  target.pos = target.core = 0;
  // We have to tell the Dll what type of target we are querying
  // We are not dealing with cores here so let the Dll know we want to know everything above that
  target.coreState = ECMD_TARGET_FIELD_UNUSED

  // Is this target configured ?
  if (ecmdQueryTargetConfigured(target)) {
    printf("pu 0:0 is configured");
  } else {
    printf("**** ERROR : pu 0:0 is not configured, unable to complete test");
    return 1;
  }

  // -----------------
  // Ring's
  // -----------------
  rc = getRing (target, "sgxbs", data);
  if (rc) return rc;
  printf("Scanned ring sgxbs - length = d",data.getBitLength());

  // We need to set a few bits
  // Set an entire word
  data.setWord(1, 0xFEEDBEEF);
  // Set bit 2
```

```
data.setBit(2);
// Set bits 5-9
data.setBit(5,5);
// Clear bit 12
data.clearBit(12);

// Scan the ring back in
rc = putRing (target, "sgxbs", data);
if (rc) return rc;


// -----------------
// Spy's
// -----------------
// We will enable ring caching this will reduce the scans to the hardware
 ecmdEnableRingCache() (p. 88);

// First we will try a non-enumerated spy
rc = getSpy (target, "MYSPY", data);
if (rc) return rc;
data.setWord(0,0xAAAAAAAA);
rc = putSpy (target, "MYSPY", data);
if (rc) return rc;

// Now an enumerated spy
std::string enumval;
rc = getSpyEnum (target, "MYENUMSPY", enumval);
if (rc) return rc;
printf("pu 0:0 MYENUMSPY is set to : s",enumval.c_str());
rc = putSpyEnum (target, "MYENUMSPY", "ENABLE");
if (rc) return rc;

// Now that we are done with that, flush all the rings to the hardware that were modified
rc = ecmdDisableRingCache() (p. 88);
if (rc) return rc;


// -----------------
// Scom's
// -----------------

rc = getScom (target, 0x800003, data);
if (rc) return rc;
printf("pu 0:0 800003 %.08X %.08X",data.getWord(0),data.getWord(1));
data.setWord(1,0x5555AAAA);
rc = putScom (target, 0x800003, data);
if (rc) return rc;


// --------------
// Config Looping
// --------------
// I want to loop on all the pu chips that the user selected with -p# -n#
// Looping on selected positions only works when ecmdCommandArgs has been previously called

// Setup the target we will use
```

```
  // We want to loop on all 'pu' chips so set that, everything else is wildcard
  target.chipType = "pu";
  target.chipTypeState = ECMD_TARGET_QUERY_FIELD_VALID;
  target.cageState = target.nodeState = target.slotState = target.posState = target.coreState = ECMD_TARGET_QUE
  // For the function we are doing we know that we don't care about threads
  target.threadState = ECMD_TARGET_FIELD_UNUSED;

  bool validPosFound = false;
   ecmdLooperData (p. 51) looperdata;
  // Initialize the config looper, tell it to loop on targets selected by the user -p# -c#
  // To loop on all targets in the system, not just those selected change this to : ECMD_ALL_TARGETS_LOOP
  rc = ecmdConfigLooperInit(target, ECMD_SELECTED_TARGETS_LOOP, looperdata);
  if (rc) return rc;

  // This loop will continue as long as valid targets are found
  //  each time returning with the target variable filled it
  while ( ecmdConfigLooperNext(target, looperdata) ) {

    // We will dump all the idregs
    rc = getRing(target, "idreg", data);
    printf("Idreg for s : 0x%.08X", ecmdWriteTarget(target).c_str(), data.getWord(0));

    // Signify that we looped at least once
    validPosFound = true;
  }
  if (!validPosFound) {
    // We never went into the while loop this means the positions the user selected where not in the system
    printf("**** ERROR : Position selected was not valid");
  }


  // Unload the eCMD Dll, this should always be the last thing you do
   ecmdUnloadDll() (p. 77);

  return rc;

}
```

Chapter 2

eCMD C/C++ Dll Compound Index

2.1 eCMD C/C++ Dll Compound List

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

ecmdArrayData (Used for the ecmdQueryArray function to return array info) 11
ecmdArrayEntry (Used by the getArrayMultiple function to pass data) 13
ecmdCageData (Used for the ecmdQueryConfig function to return cage data) 14
ecmdChipData (Used for the ecmdQueryConfig function to return chip data) 15
ecmdChipTarget (Structure used to designate which cec object/chip you would like the function to operate on) 17
ecmdCoreData (Used for the ecmdQueryConfig function to return core data) 20
ecmdDataBuffer (Provides a means to handle data from the eCMD C API) 21
ecmdDllInfo (This is used by ecmdQueryDllInfo to return info to the client about what Dll instance they are actually running with) 45
ecmdGroupData (Used by get/putspy function to create the return data from a group) 47
ecmdIndexEntry (Used by get/put Gpr/Fpr Multiple function to pass data) 48
ecmdLatchEntry (Used by getlatch function to return data) 49
ecmdLooperData (Used internally by ecmdConfigLooper to store looping state information) 51
ecmdNameEntry (Used by get/putSprMultiple function to pass data) 53
ecmdNameVectorEntry (Used by getTraceArrayMultiple function to pass data) . . . 54
ecmdNodeData (Used for the ecmdQueryConfig function to return node data) 55
ecmdProcRegisterInfo (Used by ecmdQueryProcRegisterInfo function to return data about a Architected register) 56
ecmdQueryData (Used by the ecmdQueryConfig function to return data) 57
ecmdRingData (Used for the ecmdQueryRing function to return ring info) 58
ecmdSlotData (Used for the ecmdQueryConfig function to return slot data) 60
ecmdSpyData (Used for the ecmdQuerySpy function to return spy info) 61
ecmdThreadData (Used for the ecmdQueryConfig function to return thread data) . . 63

Chapter 3

eCMD C/C++ Dll File Index

3.1 eCMD C/C++ Dll File List

Here is a list of all files with brief descriptions:

cipClientCapi.H (Cronus & IP eCMD Extension) 65
ecmdClientCapi.H (ECMD C/C++ Client Interface) 68
ecmdDataBuffer.H (Provides a means to handle data from the eCMD C API) 113
ecmdReturnCodes.H (All Return Codes for the eCmd Capi) 114
ecmdStructs.H (All the Structures required for the eCMD Capi) 122
ecmdUtils.H (Useful functions for use throughout the ecmd C API) 130

Chapter 4

eCMD C/C++ Dll Class Documentation

4.1 ecmdArrayData Struct Reference

Used for the ecmdQueryArray function to return array info.

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

Public Attributes

- std::string **arrayName**

  Names used to reference this array.

- int **addressLength**

  Bit length of address.

- int **length**

  Length of array (number of entries).

- int **width**

  Bit width of array entry.

- **ecmdClockState_t clockState**

  Required clock state to access this array.

4.1.1 Detailed Description

Used for the ecmdQueryArray function to return array info.

4.1.2 Member Data Documentation

4.1.2.1 std::string ecmdArrayData::arrayName

Names used to reference this array.

4.1.2.2 int ecmdArrayData::addressLength

Bit length of address.

4.1.2.3 int ecmdArrayData::length

Length of array (number of entries).

4.1.2.4 int ecmdArrayData::width

Bit width of array entry.

4.1.2.5 ecmdClockState_t ecmdArrayData::clockState

Required clock state to access this array.

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

- **ecmdStructs.H**

4.2 ecmdArrayEntry Struct Reference

Used by the getArrayMultiple function to pass data.

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdDataBuffer address**

  Array address/element to access.

- **ecmdDataBuffer buffer**

  Array data from address.

- uint32_t **rc**

  Error code in retrieving this entry.

4.2.1 Detailed Description

Used by the getArrayMultiple function to pass data.

4.2.2 Member Data Documentation

4.2.2.1 ecmdDataBuffer ecmdArrayEntry::address

Array address/element to access.

4.2.2.2 ecmdDataBuffer ecmdArrayEntry::buffer

Array data from address.

4.2.2.3 uint32_t ecmdArrayEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

4.3 ecmdCageData Struct Reference

Used for the ecmdQueryConfig function to return cage data.

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **cageId**

  (Detail: Low) Cage number of this entry

- std::list< **ecmdNodeData** > **nodeData**

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

4.3.1 Detailed Description

Used for the ecmdQueryConfig function to return cage data.

Operators Supported : <

4.3.2 Member Data Documentation

4.3.2.1 uint32_t ecmdCageData::cageId

(Detail: Low) Cage number of this entry

4.3.2.2 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**

4.4 ecmdChipData Struct Reference

Used for the ecmdQueryConfig function to return chip data.

`#include <ecmdStructs.H>`

Public Attributes

- std::string **chipType**

  (Detail: Low) actual name of chip , ie. gr, ent (should be 3chars or less)

- std::string **chipCommonType**

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

- uint32_t **pos**

  (Detail: Low) Position 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, usually 0-F (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

4.4.1 Detailed Description

Used for the ecmdQueryConfig function to return chip data.

Operators Supported : <

4.4.2 Member Data Documentation

4.4.2.1 std::string ecmdChipData::chipType

(Detail: Low) actual name of chip , ie. gr, ent (should be 3chars or less)

4.4.2.2 std::string ecmdChipData::chipCommonType

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

4.4.2.3 uint32_t ecmdChipData::pos

(Detail: Low) Position of this entry

4.4.2.4 uint8_t ecmdChipData::numProcCores

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

4.4.2.5 uint32_t ecmdChipData::chipEc

(Detail: High) EC level of this chip, usually 0-F (ec read from 'jtag' chip id or CFAM id)

4.4.2.6 uint32_t ecmdChipData::simModelEc

(Detail: High) Model EC level of this chip

4.4.2.7 ecmdChipInterfaceType_t ecmdChipData::interfaceType

(Detail: High) Interface Macro used by the chip

4.4.2.8 uint32_t ecmdChipData::chipFlags

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

4.4.2.9 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**

4.5 ecmdChipTarget Struct Reference

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

`#include <ecmdStructs.H>`

Public Attributes

- uint32_t **cage**

  cage that contains node with chip

- uint32_t **node**

  node that contains chip

- uint32_t **slot**

  Card Slot/Fru to target.

- std::string **chipType**

  name of chip to access , either actual or common name

- uint32_t **pos**

  position of chip within node

- uint8_t **core**

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

- uint8_t **thread**

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

- **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

- std::string **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 unitIdState**

  unitId field state

4.5.1 Detailed Description

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

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

4.5.2 Member Data Documentation

4.5.2.1 uint32_t ecmdChipTarget::cage

cage that contains node with chip

4.5.2.2 uint32_t ecmdChipTarget::node

node that contains chip

4.5.2.3 uint32_t ecmdChipTarget::slot

Card Slot/Fru to target.

4.5.2.4 std::string ecmdChipTarget::chipType

name of chip to access , either actual or common name

4.5.2.5 uint32_t ecmdChipTarget::pos

position of chip within node

4.5.2.6 uint8_t ecmdChipTarget::core

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

4.5.2.7 uint8_t ecmdChipTarget::thread

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

4.5.2.8 ecmdChipTargetState_t ecmdChipTarget::cageState

cage field state

4.5.2.9 ecmdChipTargetState_t ecmdChipTarget::nodeState

node field state

4.5.2.10 ecmdChipTargetState_t ecmdChipTarget::slotState

slot field state

4.5.2.11 ecmdChipTargetState_t ecmdChipTarget::chipTypeState

chipType field state

4.5.2.12 ecmdChipTargetState_t ecmdChipTarget::posState

pos field state

4.5.2.13 ecmdChipTargetState_t ecmdChipTarget::coreState

core field state

4.5.2.14 ecmdChipTargetState_t ecmdChipTarget::threadState

thread field state

4.5.2.15 std::string 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.

4.5.2.16 ecmdChipTargetState_t ecmdChipTarget::unitIdState

unitId field state

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

- **ecmdStructs.H**

4.6 ecmdCoreData Struct Reference

Used for the ecmdQueryConfig function to return core data.

`#include <ecmdStructs.H>`

Public Attributes

- uint8_t **coreId**

  (Detail: Low) core number of this entry

- uint8_t **numProcThreads**

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

- std::list< **ecmdThreadData** > **threadData**

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

4.6.1 Detailed Description

Used for the ecmdQueryConfig function to return core data.

Operators Supported : <

4.6.2 Member Data Documentation

4.6.2.1 uint8_t ecmdCoreData::coreId

(Detail: Low) core number of this entry

4.6.2.2 uint8_t ecmdCoreData::numProcThreads

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

4.6.2.3 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**

4.7 ecmdDataBuffer Class Reference

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

`#include <ecmdDataBuffer.H>`

Public Methods

ecmdDataBuffer Constructors

- **ecmdDataBuffer** ()
  Default Constructor.
- **ecmdDataBuffer** (int i_numWords)
  Constructor.
- **ecmdDataBuffer** (const ecmdDataBuffer &other)
  Copy Constructor.
- **~ecmdDataBuffer** ()
  Default Destructor.

Buffer Size Function

- int **getWordLength** () const
  Return the length of the buffer in words.
- int **getByteLength** () const
  Return the length of the buffer in bytes.
- int **getBitLength** () const
  Return the length of the buffer in bits.
- int **getCapacity** () const
  Return the actual capacity of the internal buffer in words.
- void **setWordLength** (int i_newNumWords)
  Reinitialize the Buffer to specified length.
- void **setBitLength** (int i_newNumBits)
  Reinitialize the Buffer to specified length.
- void **setCapacity** (int i_newNumWords)
  Reinitialize the internal buffer to specified length.

Bit/Word Manipulation Functions

- void **setBit** (int i_bit)
  Turn on a bit in buffer.
- void **setBit** (int i_bit, int i_len)
  Turn on a bit in buffer.

- void **writeBit** (int i_bit, int i_value)
  Write a bit to specified value in buffer.
- void **setWord** (int i_wordoffset, uint32_t i_value)
  Set a word of data in buffer.
- uint32_t **getWord** (int i_wordoffset)
  Fetch a word from ecmdDataBuffer.
- void **setByte** (int i_byteoffset, uint8_t i_value)
  Set a byte of data in buffer.
- uint8_t **getByte** (int i_byteoffset)
  Fetch a byte from ecmdDataBuffer.
- void **clearBit** (int i_bit)
  Clear a bit in buffer.
- void **clearBit** (int i_bit, int i_len)
  Clear multiple bits in buffer.
- void **flipBit** (int i_bit)
  Invert bit.
- void **flipBit** (int i_bit, int i_len)
  Invert multiple bits.
- int **isBitSet** (int i_bit)
  Test if bit is set.
- int **isBitSet** (int i_bit, int i_len)
  Test if multiple bits are set.
- int **isBitClear** (int i_bit)
  Test if bit is clear.
- int **isBitClear** (int i_bit, int i_len)
  Test if multiple bits are clear.
- int **getNumBitsSet** (int i_bit, int i_len)
  Count number of bits set in a range.

Buffer Manipulation Functions

- void **shiftRight** (int i_shiftnum)
  Shift data to right.
- void **shiftLeft** (int i_shiftnum)
  Shift data to left.
- void **shiftRightAndResize** (int i_shiftnum)
  Shift data to right - resizing buffer.

- void **shiftLeftAndResize** (int i_shiftnum)
  Shift data to left - resizing buffer.
- void **rotateRight** (int i_rotatenum)
  Rotate data to right.
- void **rotateLeft** (int i_rotatenum)
  Rotate data to left.
- void **flushTo0** ()
  Clear entire buffer to 0's.
- void **flushTo1** ()
  Set entire buffer to 1's.
- void **invert** ()
  Invert entire buffer.
- void **applyInversionMask** (uint32_t *i_invMask, int i_invByteLen)
  Apply an inversion mask to data inside buffer.
- void **insert** (ecmdDataBuffer &i_bufferIn, int i_start, int i_len)
  Insert part of another DataBuffer into this one.
- void **insert** (uint32_t *i_datain, int i_start, int i_len)
  Insert a uint32_t array into this DataBuffer.
- void **insert** (uint32_t i_datain, int i_start, int i_len)
  Insert a uint32_t into the DataBuffer.
- void **insertFromRight** (uint32_t *i_datain, int i_start, int i_len)
  Insert a right aligned (decimal) uint32_t array into this DataBuffer.
- void **insertFromRight** (uint32_t i_datain, int i_start, int i_len)
  Insert a right aligned (decimal) uint32_t into the DataBuffer.
- void **extract** (ecmdDataBuffer &o_bufferOut, int i_start, int i_len)
  Copy data from this DataBuffer into another.
- void **extract** (uint32_t *o_data, int i_start, int i_len)
  Copy data from this DataBuffer into another.
- void **setOr** (ecmdDataBuffer &i_bufferIn, int i_startbit, int i_len)
  OR data into DataBuffer.
- void **setOr** (uint32_t *i_datain, int i_startbit, int i_len)
  OR data into DataBuffer.
- void **setOr** (uint32_t i_datain, int i_startbit, int i_len)
  OR data into DataBuffer.
- void **merge** (ecmdDataBuffer &i_bufferIn)
  OR data into DataBuffer.
- void **setXor** (ecmdDataBuffer &i_bufferIn, int i_startbit, int i_len)

XOR data into DataBuffer.

- void **setXor** (uint32_t *i_datain, int i_startbit, int i_len)
  XOR data into DataBuffer.
- void **setXor** (uint32_t i_datain, int i_startbit, int i_len)
  XOR data into DataBuffer.
- void **setAnd** (ecmdDataBuffer &i_bufferIn, int i_startbit, int i_len)
  AND data into DataBuffer.
- void **setAnd** (uint32_t *i_datain, int i_startbit, int i_len)
  AND data into DataBuffer.
- void **setAnd** (uint32_t i_datain, int i_startbit, int i_len)
  AND data into DataBuffer.
- void **copy** (ecmdDataBuffer &o_copyBuffer)
  Copy entire contents of this ecmdDataBuffer into o_copyBuffer.
- ecmdDataBuffer & **operator=** (const ecmdDataBuffer &i_master)
  Copy Constructor.
- void **memCopyIn** (uint32_t *i_buf, int i_bytes)
  Copy buffer into this ecmdDataBuffer.
- void **memCopyOut** (uint32_t *o_buf, int i_bytes)
  Copy DataBuffer into supplied uint32_t buffer.

Parity Functions

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

Buffer Character Conversion Functions

- std::string **genHexLeftStr** (int i_start, int i_bitlen)
  Return Data as a hex left aligned char string.
- std::string **genHexRightStr** (int i_start, int i_bitlen)
  Return Data as a hex right aligned char string.
- std::string **genBinStr** (int i_start, int i_bitlen)
  Return Data as a binary char string.

- std::string **genHexLeftStr** ()

  Return entire buffer as a hex left aligned char string.

- std::string **genHexRightStr** ()

  Return entire buffer as a hex right aligned char string.

- std::string **genBinStr** ()

  Return entire buffer as a binary char string.

- std::string **genXstateStr** (int i_start, int i_bitlen)

  Retrieve a section of the Xstate Data.

- std::string **genXstateStr** ()

  Retrieve entire Xstate Data buffer.

String to Data conversion functions

- int **insertFromHexLeft** (const char *i_hexChars, int i_start=0, int i_length=0)

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

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

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

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

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

Simulation Buffer Functions

- int **hasXstate** ()

  Check Entire buffer for any X-state values.

- int **hasXstate** (int i_start, int i_length)

  Check section of buffer for any X-state values.

- char **getXstate** (int i_bit)

  Retrieve an Xstate value from the buffer.

- void **setXstate** (int i_bit, char i_value)

  Set an Xstate value in the buffer.

- void **setXstate** (int i_bitoffset, const char *i_datastr)

  Set a range of Xstate values in buffer.

- void **memCopyInXstate** (const char *i_buf, int i_bytes)

  Copy buffer into the Xstate data of this ecmdDataBuffer.

- void **memCopyOutXstate** (char *o_buf, int i_bytes)

  Copy DataBuffer into supplied char buffer from Xstate data.

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.

4.7.1 Detailed Description

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

4.7.2 Constructor & Destructor Documentation

4.7.2.1 ecmdDataBuffer::ecmdDataBuffer ()

Default Constructor.

Postcondition:

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

4.7.2.2 ecmdDataBuffer::ecmdDataBuffer (int *i_numWords*)

Constructor.

Parameters:

i_numWords Size of data to initialize in 32-bit words

Postcondition:

ecmdDataBuffer is initialzed with a buffer

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

Copy Constructor.

Parameters:

other Buffer to copy

4.7.2.4 ecmdDataBuffer::~ecmdDataBuffer ()

Default Destructor.

4.7.3 Member Function Documentation

4.7.3.1 int ecmdDataBuffer::getWordLength ()

Return the length of the buffer in words.

Return values:

Buffer length in words rounded up

4.7.3.2 int ecmdDataBuffer::getByteLength ()

Return the length of the buffer in bytes.

Return values:

Buffer length in bytes rounded up

4.7.3.3 int ecmdDataBuffer::getBitLength ()

Return the length of the buffer in bits.

Return values:

Buffer length in bits

4.7.3.4 int ecmdDataBuffer::getCapacity ()

Return the actual capacity of the internal buffer in words.

Return values:

Actual capacity of internal buffer

4.7.3.5 void ecmdDataBuffer::setWordLength (int *i_newNumWords*)

Reinitialize the Buffer to specified length.

Parameters:

i_newNumWords Length of new buffer in words

Postcondition:

Buffer is reinitialized

CAUTION : All data stored in buffer will be lost

4.7.3.6 void ecmdDataBuffer::setBitLength (int *i_newNumBits*)

Reinitialize the Buffer to specified length.

Parameters:

i_newNumBits Length of new buffer in bits

Postcondition:

Buffer is reinitialized

CAUTION : All data stored in buffer will be lost

4.7.3.7 void ecmdDataBuffer::setCapacity (int *i_newNumWords*)

Reinitialize the internal buffer to specified length.

Parameters:

i_newNumWords length of internal data buffer in words

Postcondition:

Internal buffer is reinitialzied

CAUTION : All data stored in buffer will be lost

4.7.3.8 void ecmdDataBuffer::setBit (int *i_bit*)

Turn on a bit in buffer.

Parameters:

i_bit Bit in buffer to turn on

4.7.3.9 void ecmdDataBuffer::setBit (int *i_bit*, int *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

4.7.3.10 void ecmdDataBuffer::writeBit (int *i_bit*, int *i_value*)

Write a bit to specified value in buffer.

Parameters:

i_bit Bit in buffer to turn on

i_value Value to write

4.7.3.11 void ecmdDataBuffer::setWord (int *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

4.7.3.12 uint32_t ecmdDataBuffer::getWord (int *i_wordoffset*)

Fetch a word from ecmdDataBuffer.

Parameters:

i_wordoffset Offset of word to fetch

Return values:

Value of word requested

4.7.3.13 void ecmdDataBuffer::setByte (int *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

4.7.3.14 uint8_t ecmdDataBuffer::getByte (int *i_byteoffset*)

Fetch a byte from ecmdDataBuffer.

Parameters:

i_byteoffset Offset of byte to fetch

Return values:

Value of byte requested

4.7.3.15 void ecmdDataBuffer::clearBit (int *i_bit*)

Clear a bit in buffer.

Parameters:

i_bit Bit in buffer to turn off

4.7.3.16 void ecmdDataBuffer::clearBit (int *i_bit*, int *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

4.7.3.17 void ecmdDataBuffer::flipBit (int *i_bit*)

Invert bit.

Parameters:

i_bit Bit in buffer to invert

4.7.3.18 void ecmdDataBuffer::flipBit (int *i_bit*, int *i_len*)

Invert multiple bits.

Parameters:

i_bit Start bit in buffer to invert

i_len Number of consecutive bits to invert

4.7.3.19 int ecmdDataBuffer::isBitSet (int *i_bit*)

Test if bit is set.

Parameters:

i_bit Bit to test

Return values:

true if bit is set - false if bit is clear

4.7.3.20 int ecmdDataBuffer::isBitSet (int *i_bit*, int *i_len*)

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

4.7.3.21 int ecmdDataBuffer::isBitClear (int *i_bit*)

Test if bit is clear.

Parameters:

i_bit Bit to test

Return values:

true if bit is clear - false if bit is set

4.7.3.22 int ecmdDataBuffer::isBitClear (int *i_bit*, int *i_len*)

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

4.7.3.23 int ecmdDataBuffer::getNumBitsSet (int *i_bit*, int *i_len*)

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

4.7.3.24 void ecmdDataBuffer::shiftRight (int *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

4.7.3.25 void ecmdDataBuffer::shiftLeft (int *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

4.7.3.26 void ecmdDataBuffer::shiftRightAndResize (int *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

4.7.3.27 void ecmdDataBuffer::shiftLeftAndResize (int *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

4.7.3.28 void ecmdDataBuffer::rotateRight (int *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

4.7.3.29 void ecmdDataBuffer::rotateLeft (int *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

4.7.3.30 void ecmdDataBuffer::flushTo0 ()

Clear entire buffer to 0's.

4.7.3.31 void ecmdDataBuffer::flushTo1 ()

Set entire buffer to 1's.

4.7.3.32 void ecmdDataBuffer::invert ()

Invert entire buffer.

4.7.3.33 void ecmdDataBuffer::applyInversionMask (uint32_t * *i_invMask*, int *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

4.7.3.34 void ecmdDataBuffer::insert (ecmdDataBuffer & *i_bufferIn*, int *i_start*, int *i_len*)

Insert part of another DataBuffer into this one.

Parameters:

i_bufferIn DataBuffer to copy data from - data is taken left aligned

i_start Start bit to insert to

i_len Length of bits to insert

Postcondition:

Data is copied from bufferIn to this DataBuffer in specified location for length or to end of this DataBuffer size whichever is less

4.7.3.35 void ecmdDataBuffer::insert (uint32_t * *i_datain*, int *i_start*, int *i_len*)

Insert a uint32_t array into this DataBuffer.

Parameters:

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

i_start Start bit to insert into

i_len Length of bits to insert

Postcondition:

Data is copied from datain into this DataBuffer at specified location for length or to end of this DataBuffer size whichever is less

4.7.3.36 void ecmdDataBuffer::insert (uint32_t *i_datain*, int *i_start*, int *i_len*)

Insert a uint32_t into the DataBuffer.

Parameters:

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

i_start Start bit to insert into

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

Postcondition:

Data is copied from datain into this DataBuffer at specified location for length or to end of this DataBuffer size whichever is less

4.7.3.37 void ecmdDataBuffer::insertFromRight (uint32_t * *i_datain*, int *i_start*, int *i_len*)

Insert 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

Postcondition:

Data is copied from datain into this DataBuffer at specified location for length or to end of this DataBuffer size whichever is less

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

4.7.3.38 void ecmdDataBuffer::insertFromRight (uint32_t *i_datain*, int *i_start*, int *i_len*)

Insert 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)

Postcondition:

Data is copied from datain into this DataBuffer at specified location for length or to end of this DataBuffer size whichever is less

4.7.3.39 void ecmdDataBuffer::extract (ecmdDataBuffer & *o_bufferOut*, int *i_start*, int *i_len*)

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 for length or to end of bufferOut size whichever is less

4.7.3.40 void ecmdDataBuffer::extract (uint32_t * *o_data*, int *i_start*, int *i_len*)

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

4.7.3.41 void ecmdDataBuffer::setOr (ecmdDataBuffer & *i_bufferIn*, int *i_startbit*, int *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

4.7.3.42 void ecmdDataBuffer::setOr (uint32_t * *i_datain*, int *i_startbit*, int *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

4.7.3.43 void ecmdDataBuffer::setOr (uint32_t *i_datain*, int *i_startbit*, int *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

4.7.3.44 void ecmdDataBuffer::merge (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

4.7.3.45 void ecmdDataBuffer::setXor (ecmdDataBuffer & *i_bufferIn*, int *i_startbit*, int *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

4.7.3.46 void ecmdDataBuffer::setXor (uint32_t * *i_datain*, int *i_startbit*, int *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

4.7.3.47 void ecmdDataBuffer::setXor (uint32_t *i_datain*, int *i_startbit*, int *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

4.7.3.48 void ecmdDataBuffer::setAnd (ecmdDataBuffer & *i_bufferIn*, int *i_startbit*, int *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

4.7.3.49 void ecmdDataBuffer::setAnd (uint32_t * *i_datain*, int *i_startbit*, int *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

4.7.3.50 void ecmdDataBuffer::setAnd (uint32_t *i_datain*, int *i_startbit*, int *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

4.7.3.51 void ecmdDataBuffer::copy (ecmdDataBuffer & *o_copyBuffer*)

Copy entire contents of this ecmdDataBuffer into o_copyBuffer.

Parameters:

o_copyBuffer DataBuffer to copy data into

Postcondition:

copyBuffer is an exact duplicate of this DataBuffer

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

Copy Constructor.

Parameters:

i_master DataBuffer to copy from

4.7.3.53 void ecmdDataBuffer::memCopyIn (uint32_t * *i_buf*, int *i_bytes*)

Copy buffer into this ecmdDataBuffer.

Parameters:

i_buf Buffer to copy from

i_bytes Byte length to copy

Postcondition:

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

4.7.3.54 void ecmdDataBuffer::memCopyOut (uint32_t * *o_buf*, int *i_bytes*)

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

4.7.3.55 int ecmdDataBuffer::oddParity (int *i_start*, int *i_stop*)

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

4.7.3.56 int ecmdDataBuffer::evenParity (int *i_start*, int *i_stop*)

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

4.7.3.57 int ecmdDataBuffer::oddParity (int *i_start*, int *i_stop*, int *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:

0 on success - nonzero on failure

4.7.3.58 int ecmdDataBuffer::evenParity (int *i_start*, int *i_stop*, int *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:

0 on success - nonzero on failure

4.7.3.59 std::string ecmdDataBuffer::genHexLeftStr (int *i_start*, int *i_bitlen*)

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

4.7.3.60 std::string ecmdDataBuffer::genHexRightStr (int *i_start*, int *i_bitlen*)

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

4.7.3.61 std::string ecmdDataBuffer::genBinStr (int *i_start*, int *i_bitlen*)

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

4.7.3.62 std::string ecmdDataBuffer::genHexLeftStr ()

Return entire buffer as a hex left aligned char string.

Return values:

String containing requested data

4.7.3.63 std::string ecmdDataBuffer::genHexRightStr ()

Return entire buffer as a hex right aligned char string.

Return values:

String containing requested data

4.7.3.64 std::string ecmdDataBuffer::genBinStr ()

Return entire buffer as a binary char string.

Return values:

String containing requested data

4.7.3.65 std::string ecmdDataBuffer::genXstateStr (int *i_start*, int *i_bitlen*)

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

4.7.3.66 std::string ecmdDataBuffer::genXstateStr ()

Retrieve entire Xstate Data buffer.

Return values:

String containing requested data

4.7.3.67 int ecmdDataBuffer::insertFromHexLeft (const char * *i_hexChars*, int *i_start* = 0, int *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

4.7.3.68 int ecmdDataBuffer::insertFromHexRight (const char * *i_hexChars*, int *i_start* = 0, int *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

4.7.3.69 int ecmdDataBuffer::insertFromBin (const char * *i_binChars*, int *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

4.7.3.70 int ecmdDataBuffer::hasXstate ()

Check Entire buffer for any X-state values.

Return values:

1 if xstate found 0 if none

4.7.3.71 int ecmdDataBuffer::hasXstate (int *i_start*, int *i_length*)

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:

1 if xstate found 0 if none

4.7.3.72 char ecmdDataBuffer::getXstate (int *i_bit*)

Retrieve an Xstate value from the buffer.

Parameters:

i_bit Bit to retrieve

NOTE - To retrieve multiple bits use genXstateStr

4.7.3.73 void ecmdDataBuffer::setXstate (int *i_bit*, char *i_value*)

Set an Xstate value in the buffer.

Parameters:

i_bit Bit to set

i_value Xstate value to set

4.7.3.74 void ecmdDataBuffer::setXstate (int *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”

4.7.3.75 void ecmdDataBuffer::memCopyInXstate (const char ∗ *i_buf*, int *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

4.7.3.76 void ecmdDataBuffer::memCopyOutXstate (char ∗ *o_buf*, int *i_bytes*)

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

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

Overload the == operator.

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

Overload the != operator.

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

Overload the & operator.

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

Overload the | operator.

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

- **ecmdDataBuffer.H**

4.8 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.

- **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.

4.8.1 Detailed Description

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

4.8.2 Member Data Documentation

4.8.2.1 ecmdDllType_t ecmdDllInfo::dllType

Dll instance type running.

4.8.2.2 ecmdDllProduct_t ecmdDllInfo::dllProduct

Dll product supported.

4.8.2.3 ecmdDllEnv_t ecmdDllInfo::dllEnv

Dll environment (Simulation vs Hardware).

4.8.2.4 std::string ecmdDllInfo::dllBuildDate

Date the Dll was built.

4.8.2.5 std::string ecmdDllInfo::dllCapiVersion

should be set to ECMD_CAPI_VERSION

4.8.2.6 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**

4.9 ecmdGroupData 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.

4.9.1 Detailed Description

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

4.9.2 Member Data Documentation

4.9.2.1 ecmdDataBuffer ecmdGroupData::extractBuffer

The data read from the ring buffer.

4.9.2.2 ecmdDataBuffer ecmdGroupData::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**

4.10 ecmdIndexEntry Struct Reference

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

`#include <ecmdStructs.H>`

Public Attributes

- int **index**

  Index of entry.

- **ecmdDataBuffer buffer**

  Data to/from entry.

- uint32_t **rc**

  Error code in retrieving this entry.

4.10.1 Detailed Description

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

4.10.2 Member Data Documentation

4.10.2.1 int ecmdIndexEntry::index

Index of entry.

4.10.2.2 ecmdDataBuffer ecmdIndexEntry::buffer

Data to/from entry.

4.10.2.3 uint32_t ecmdIndexEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

4.11 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.

4.11.1 Detailed Description

Used by getlatch function to return data.

4.11.2 Member Data Documentation

4.11.2.1 std::string ecmdLatchEntry::latchName

Latch name of entry.

4.11.2.2 std::string ecmdLatchEntry::ringName

Ring that latch came from.

4.11.2.3 ecmdDataBuffer ecmdLatchEntry::buffer

Latch data.

4.11.2.4 int ecmdLatchEntry::latchStartBit

Start bit of data inside latch.

4.11.2.5 int ecmdLatchEntry::latchEndBit

End bit of data inside latch.

4.11.2.6 uint32_t ecmdLatchEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

4.12 ecmdLooperData Struct Reference

Used internally by ecmdConfigLooper to store looping state information.

`#include <ecmdStructs.H>`

Public Attributes

- **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.
- bool **ecmdLooperInitFlag**
  Is fresh ?

4.12.1 Detailed Description

Used internally by ecmdConfigLooper to store looping state information.

4.12.2 Member Data Documentation

4.12.2.1 ecmdQueryData ecmdLooperData::ecmdSystemConfigData

Config data queried from the system.

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

Pointer to current Cage.

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

Pointer to current Node.

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

Pointer to current Slot.

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

Pointer to current Chip.

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

Pointer to current Core.

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

Pointer to current Thread.

4.12.2.8 ecmdChipTarget ecmdLooperData::prevTarget

Pointer to previous target.

4.12.2.9 bool ecmdLooperData::ecmdLooperInitFlag

Is fresh ?

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

- **ecmdStructs.H**

4.13 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.

4.13.1 Detailed Description

Used by get/putSprMultiple function to pass data.

4.13.2 Member Data Documentation

4.13.2.1 std::string ecmdNameEntry::name

Name of entry.

4.13.2.2 ecmdDataBuffer ecmdNameEntry::buffer

Data to/from entry.

4.13.2.3 uint32_t ecmdNameEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

4.14 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.

4.14.1 Detailed Description

Used by getTraceArrayMultiple function to pass data.

4.14.2 Member Data Documentation

4.14.2.1 std::string ecmdNameVectorEntry::name

Name of entry.

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

Vector of data to/from entry.

4.14.2.3 uint32_t ecmdNameVectorEntry::rc

Error code in retrieving this entry.

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

- **ecmdStructs.H**

4.15 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

- std::list< **ecmdSlotData** > **slotData**

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

4.15.1 Detailed Description

Used for the ecmdQueryConfig function to return node data.

Operators Supported : <

4.15.2 Member Data Documentation

4.15.2.1 uint32_t ecmdNodeData::nodeId

(Detail: Low) Node number of this entry

4.15.2.2 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**

4.16 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.

4.16.1 Detailed Description

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

4.16.2 Member Data Documentation

4.16.2.1 int ecmdProcRegisterInfo::bitLength

Bit length of each entry.

4.16.2.2 int ecmdProcRegisterInfo::totalEntries

Total number of entries available.

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

- **ecmdStructs.H**

4.17 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

4.17.1 Detailed Description

Used by the ecmdQueryConfig function to return data.

4.17.2 Member Data Documentation

4.17.2.1 ecmdQueryDetail_t ecmdQueryData::detailLevel

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

4.17.2.2 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**

4.18 ecmdRingData Struct Reference

Used for the ecmdQueryRing function to return ring info.

`#include <ecmdStructs.H>`

Public Attributes

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

  Names used to reference this ring.

- uint32_t **address**

  Address modifier.

- int **bitLength**

  length of ring

- bool **hasInversionMask**

  Ring has an inversion mask applied before scanning.

- bool **supportsBroadsideLoad**

  This ring supports broadside load in simulation.

- bool **isCheckable**

  This ring can be run through the check_rings command.

- **ecmdClockState_t clockState**

  Required clock state to access this ring.

4.18.1 Detailed Description

Used for the ecmdQueryRing function to return ring info.

4.18.2 Member Data Documentation

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

Names used to reference this ring.

4.18.2.2 uint32_t ecmdRingData::address

Address modifier.

4.18.2.3 int ecmdRingData::bitLength

length of ring

4.18.2.4 bool ecmdRingData::hasInversionMask

Ring has an inversion mask applied before scanning.

4.18.2.5 bool ecmdRingData::supportsBroadsideLoad

This ring supports broadside load in simulation.

4.18.2.6 bool ecmdRingData::isCheckable

This ring can be run through the check_rings command.

4.18.2.7 ecmdClockState_t ecmdRingData::clockState

Required clock state to access this ring.

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

- **ecmdStructs.H**

4.19 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

- std::list< **ecmdChipData** > **chipData**

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

4.19.1 Detailed Description

Used for the ecmdQueryConfig function to return slot data.

Operators Supported : <

4.19.2 Member Data Documentation

4.19.2.1 uint32_t ecmdSlotData::slotId

(Detail: Low) Slot number of this entry

4.19.2.2 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**

4.20 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.

- **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 > **eccGroups**

  Possible ecc groups names affected by this Spy.

4.20.1 Detailed Description

Used for the ecmdQuerySpy function to return spy info.

4.20.2 Member Data Documentation

4.20.2.1 std::string ecmdSpyData::spyName

Names used to reference this spy.

4.20.2.2 int ecmdSpyData::bitLength

length of spy

4.20.2.3 ecmdSpyType_t ecmdSpyData::spyType

Type of spy.

4.20.2.4 bool ecmdSpyData::isEccChecked

This spy affects some ECC groupings.

4.20.2.5 bool ecmdSpyData::isEnumerated

This spy has enumerated values.

4.20.2.6 ecmdClockState_t ecmdSpyData::clockState

Required clock state to access this spy.

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

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

4.20.2.8 std::list<std::string> ecmdSpyData::eccGroups

Possible ecc groups names affected by this Spy.

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

- **ecmdStructs.H**

4.21 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

4.21.1 Detailed Description

Used for the ecmdQueryConfig function to return thread data.

Operators Supported : <

4.21.2 Member Data Documentation

4.21.2.1 uint8_t ecmdThreadData::threadId

(Detail: Low) Thread number of this entry

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

- **ecmdStructs.H**

Chapter 5

eCMD C/C++ Dll File Documentation

5.1 cipClientCapi.H File Reference

Cronus & IP eCMD Extension.

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

Processor Functions

- uint32_t **cipStartInstructions** (**ecmdChipTarget** &i_target)

  Start Instructions.

- uint32_t **cipStopInstructions** (**ecmdChipTarget** &i_target)

  Stop 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 Breakpoint in Processor.

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

  Clear Breakpoint from Processor.

5.1.1 Detailed Description

Cronus & IP eCMD Extension.

Extension Owner : Chris Engel

5.1.2 Function Documentation

5.1.2.1 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

5.1.2.2 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

5.1.2.3 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

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

Set 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_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

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

Clear Breakpoint from Processor.

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

5.2 ecmdClientCapi.H File Reference

eCMD C/C++ Client Interface

`#include <ecmdReturnCodes.H>`

`#include <ecmdStructs.H>`

`#include <ecmdDataBuffer.H>`

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.

- 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)

  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)

  Query Ring information from the DLL.

- uint32_t **ecmdQueryArray** (**ecmdChipTarget** &i_target, **ecmdArrayData** &o_queryData, const char *i_arrayName)

  Query Array information from the DLL.

- uint32_t **ecmdQuerySpy** (**ecmdChipTarget** &i_target, **ecmdSpyData** &o_queryData, const char *i_spyName)

  Query Spy 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 selected number of bits from the selected position in the selected ring into the data buffer.

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

  Scans the selected number of bits from the data buffer into the selected position 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, **ecmdLatchMode_t** i_mode)

  Writes the data buffer into the all latches matching i_latchName.

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 **getSpyEccGrouping** (**ecmdChipTarget** &i_target, const char *i_spyEccGroupName, **ecmdDataBuffer** &o_groupData, **ecmdDataBuffer** &o_eccData, **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 **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.

iSteps Functions

- uint32_t **iSteps** (**ecmdDataBuffer** &i_steps)

  Run iSteps.

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).

Trace Array Functions

- uint32_t **getTraceArray** (**ecmdChipTarget** &i_target, const char *i_name, std::vector< **ecmdDataBuffer** > &o_data)

  Dump all entries of specified trace array.

- uint32_t **getTraceArrayMultiple** (**ecmdChipTarget** &i_target, 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.

- uint32_t **putMemProc** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_data)

  Writes System Mainstore through the processor chip.

- 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).

- 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).

- uint32_t **getMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &o_data)

  Reads System Mainstore through the memory controller.

- uint32_t **putMemMemCtrl** (**ecmdChipTarget** &i_target, uint64_t i_address, uint32_t i_bytes, **ecmdDataBuffer** &i_data)

  Writes System Mainstore through the memory controller.

Simulation Functions

- uint32_t **simaet** (const char *i_function)

  Enable/Disable Simulation AET Logging.

- uint32_t **simcheckpoint** (const char *i_checkpoint)

  Store a checkpoint to specified file.

- uint32_t **simclock** (uint32_t i_cycles)

  Clock the model.

- uint32_t **simecho** (const char *i_message)

  Echo message to stdout and sim log.

- uint32_t **simexit** (uint32_t i_rc=0, const char *i_message=NULL)

  Close down the simulation model.

- uint32_t **simEXPECTFAC** (const char *i_facname, uint32_t i_bitlength, **ecmdDataBuffer** &i_expect, uint32_t i_row=0, uint32_t i_offset=0)

  Perform expect on facility using name.

- uint32_t **simexpecttcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_expect, uint32_t i_row=0)

  Perform expect on TCFAC facility.

- uint32_t **simgetcurrentcycle** (uint32_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 valud.

- 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.

Error Handling Functions

- std::string **ecmdGetErrorMsg** (uint32_t i_errorCode)

  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.

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.

Configuration Functions

- uint32_t **ecmdGetConfiguration** (std::string i_name, std::string &o_value)

  Retrieve the value of a Configuration Setting.

- uint32_t **ecmdSetConfiguration** (std::string i_name, std::string i_value)

  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.

5.2.1 Detailed Description

eCMD C/C++ Client Interface

5.2.2 Function Documentation

5.2.2.1 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.

5.2.2.2 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

5.2.2.3 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.
- Position flags can be queried later with functions like ????? NOTE : This function does not affect ring caching

5.2.2.4 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

5.2.2.5 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_QUERY_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_QUERY_IGNORE 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

5.2.2.6 uint32_t ecmdQuerySelected (ecmdChipTarget & *io_target*, ecmdQueryData & *o_queryData*)

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

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

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

Query Ring 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_ringName if != NULL used to refine query to a single ring

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

5.2.2.8 uint32_t ecmdQueryArray (ecmdChipTarget & *i_target*, ecmdArrayData & *o_queryData*, const char * *i_arrayName*)

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 data from query

i_arrayName array to access data for

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

5.2.2.9 uint32_t ecmdQuerySpy (ecmdChipTarget & *i_target*, ecmdSpyData & *o_queryData*, const char * *i_spyName*)

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*** Spy to access data for

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

5.2.2.10 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

5.2.2.11 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

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

Scans the selected number of bits from the selected position in the selected ring 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. 81)

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

Scans the selected number of bits from the data buffer into the selected position 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. 81)

5.2.2.14 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

NOTE : This function is ring cache enabled

5.2.2.15 uint32_t putLatch (ecmdChipTarget & *i_target*, const char * *i_ringName*, const char * *i_latchName*, ecmdDataBuffer & *i_data*, 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
- ***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

NOTE : This function is ring cache enabled

5.2.2.16 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. 83)

5.2.2.17 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. 83)

5.2.2.18 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

5.2.2.19 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

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

5.2.2.20 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

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

5.2.2.21 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

5.2.2.22 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

5.2.2.23 uint32_t getSpyEccGrouping (ecmdChipTarget & *i_target*, const char * *i_spyEccGroupName*, ecmdDataBuffer & *o_groupData*, ecmdDataBuffer & *o_eccData*, 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_spyEccGroupName Name of spy to read from

o_groupData Return the data for the input to the eccGroup

o_eccData 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

5.2.2.24 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

5.2.2.25 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

5.2.2.26 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

5.2.2.27 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

5.2.2.28 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

5.2.2.29 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

5.2.2.30 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. 90) , **getArrayMultiple** (p. 89)

5.2.2.31 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. 90) , **getArray** (p. 88)

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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.32 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. 88)

5.2.2.33 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. 88)

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 retrieving multiple entries. The function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.34 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

5.2.2.35 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 : i_clockDomain has to be "ALL" unless i_target refers to a specific chip if i_target refers to a Node or Cage then an individual clockDomain cannot be specified

5.2.2.36 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 : i_clockDomain has to be "ALL" unless i_target refers to a specific chip if i_target refers to a Node or Cage then an individual clockDomain cannot be specified

5.2.2.37 uint32_t iSteps (ecmdDataBuffer & *i_steps*)

Run iSteps.

Return values:

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:

iSteps are complete

Parameters:

i_steps Bit mask defining which steps to run

5.2.2.38 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

o_data Data retrieved about the register

5.2.2.39 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_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

5.2.2.40 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_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 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 = <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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.41 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_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

5.2.2.42 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_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. 53) fields 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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.43 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_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

5.2.2.44 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_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 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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.45 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_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

5.2.2.46 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_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. 48) fields 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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.47 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_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

5.2.2.48 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_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 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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.49 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_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

5.2.2.50 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_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. 48) fields 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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

5.2.2.51 uint32_t getTraceArray (ecmdChipTarget & *i_target*, const char * *i_name*, 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

o_data Vector of trace array data retrieved

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

5.2.2.52 uint32_t getTraceArrayMultiple (ecmdChipTarget & *i_target*, 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)

o_data List of trace array data retrieved

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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 function will still continue through all entries requested. You must walk through the list returned to find out which entry caused the failure.

- NOTE : to fetch all Trace Arrays available add only one entry to io_entries and set ecmdNameVectorEntry.name = "ALL"

5.2.2.53 uint32_t getMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Reads System Mainstore through the processor chip.

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 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

5.2.2.54 uint32_t putMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_data*)

Writes System Mainstore through the processor chip.

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 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

5.2.2.55 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).

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

5.2.2.56 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).

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

5.2.2.57 uint32_t getMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_data*)

Reads System Mainstore through the memory controller.

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 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

5.2.2.58 uint32_t putMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_data*)

Writes System Mainstore through the memory controller.

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 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

5.2.2.59 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

5.2.2.60 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

5.2.2.61 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

5.2.2.62 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

5.2.2.63 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

5.2.2.64 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

5.2.2.65 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

5.2.2.66 uint32_t simgetcurrentcycle (uint32_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

5.2.2.67 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

5.2.2.68 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

5.2.2.69 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

5.2.2.70 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

5.2.2.71 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 valud.

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

5.2.2.72 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

5.2.2.73 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

5.2.2.74 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

5.2.2.75 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

5.2.2.76 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

5.2.2.77 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

5.2.2.78 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

5.2.2.79 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

5.2.2.80 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

5.2.2.81 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

5.2.2.82 std::string ecmdGetErrorMsg (uint32_t *i_errorCode*)

Retrieve additional error information for errorcode.

Parameters:

i_errorCode Error code to lookup up message for

Return values:

point to NULL terminated string containing error data, NULL if error occurs

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

Register an Error Message that has occured.

5.2.2.84 void ecmdOutputError (const char * *i_message*)

Output a message related to an error.

Parameters:

i_message String to output

5.2.2.85 void ecmdOutputWarning (const char * *i_message*)

Output a message related to an warning.

Parameters:

i_message String to output

5.2.2.86 void ecmdOutput (const char * *i_message*)

Output a message to the screen or logs.

Parameters:

i_message String to output

5.2.2.87 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

5.2.2.88 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

5.2.2.89 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

5.2.2.90 uint32_t ecmdGetConfiguration (std::string *i_name*, std::string & *o_value*)

Retrieve the value of a Configuration Setting.

Parameters:

i_name Name of setting as defined by eCMD Api

o_value Value of setting

Return values:

ECMD_INVALID_CONFIG_NAME Name specified is not valid

ECMD_SUCCESS if successful

5.2.2.91 uint32_t ecmdSetConfiguration (std::string *i_name*, std::string *i_value*)

Set the value of a Configuration Setting.

Parameters:

i_name Name of setting as defined by eCMD Api

i_value Value to apply to setting

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_SUCCESS if successful

nonzero on failure

5.2.2.92 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

5.2.2.93 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

ECMD_SUCCESS if successful

nonzero on failure

5.3 ecmdDataBuffer.H File Reference

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

`#include <string>`

`#include <inttypes.h>`

Compounds

- class **ecmdDataBuffer**

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

5.3.1 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

5.4 ecmdReturnCodes.H File Reference

All Return Codes for the eCmd Capi.

Defines

- #define **ECMD_SUCCESS** 0x0

  API Returned Successfully.

- #define **ECMD_INVALID_DLL_VERSION** 0x1000

  Dll Version didn't match the Client version detected.

- #define **ECMD_INVALID_DLL_FILENAME** 0x1001

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

- #define **ECMD_DLL_LOAD_FAILURE** 0x1002

  Error occured on call to dlopen.

- #define **ECMD_DLL_UNLOAD_FAILURE** 0x1003

  Error occurred on call to dlclose.

- #define **ECMD_DLL_UNINITIALIZED** 0x1004

  A function was called before ecmdLoadDll was called.

- #define **ECMD_DLL_INVALID** 0x1005

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

- #define **ECMD_FAILURE** 0x1010

  General Failure occurred in eCMD.

- #define **ECMD_TARGET_NOT_CONFIGURED** 0x1011

  Chip target provided was not configured in the system.

- #define **ECMD_FUNCTION_NOT_SUPPORTED** 0x1012

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

- #define **ECMD_UNKNOWN_FILE** 0x1013

  ecmdQueryFileLocation was unable to find the file you requested

- #define **ECMD_INVALID_ARGS** 0x1020

  Not enough arguments provided to the function.

- #define **ECMD_INVALID_SPY_ENUM** 0x1021

  getSpyEnum or putSpyEnum used an invalid enum

- #define **ECMD_SPY_FAILED_ECC_CHECK** 0x1022

  getSpy or getSpyEnum failed with invalid ECC detected in the hardware

- #define **ECMD_SPY_NOT_ENUMERATED** 0x1023 *getSpyEnum or putSpyEnum was called on a non-enumerated spy*

- #define **ECMD_SPY_IS_EDIAL** 0x1024

  getSpy or Putspy was called on an edial

- #define **ECMD_INVALID_SPY** 0x1025

  Spy functions found an invalid Spy name or type.

- #define **ECMD_DATA_OVERFLOW** 0x1026

  Too much data was provided to a write function.

- #define **ECMD_DATA_UNDERFLOW** 0x1027

  Too little data was provided to a write function.

- #define **ECMD_INVALID_RING** 0x1028

  Invalid ring name was provided.

- #define **ECMD_INVALID_ARRAY** 0x1029

  Invalid array name was provided.

- #define **ECMD_INVALID_CONFIG** 0x1030

  There was an error processing the configuration information.

- #define **ECMD_CLOCKS_IN_INVALID_STATE** 0x1031

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

- #define **ECMD_NON_JTAG_CHIP** 0x1032

  JTag function called on non-jtag attached chip.

- #define **ECMD_NON_FSI_CHIP** 0x1033

  Fsi function called on non-fsi attached chip.

- #define **ECMD_INVALID_SPR** 0x1034

  Invalid SPR was specified to get/put spr functions.

- #define **ECMD_INVALID_GPR** 0x1035

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

- #define **ECMD_INVALID_FPR** 0x1036

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

- #define **ECMD_RING_CACHE_ENABLED** 0x1037

  Ring Cache enabled during call non-cache enabled function.

- #define **ECMD_INVALID_CONFIG_NAME** 0x1038

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

- #define **ECMD_SPY_GROUP_MISMATCH** 0x1039

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

- #define **ECMD_INVALID_CLOCK_DOMAIN** 0x1040

  An invalid clock domain name was specified.

- #define **ECMD_CLOCKS_ALREADY_OFF** 0x1041

  A stopclocks was requested when clocks are already off.

- #define **ECMD_CLOCKS_ALREADY_ON** 0x1042

  A startclocks was requested when clocks are already on.

- #define **ECMD_UNABLE_TO_OPEN_SCANDEF** 0x1043

  eCMD was unable to open the scandef

- #define **ECMD_INVALID_LATCHNAME** 0x1044

  eCMD was unable to find the specified latch in the scandef

- #define **ECMD_POLLING_FAILURE** 0x1045

  eCMD failed waiting for a poll to match expected value

- #define **ECMD_INT_UNKNOWN_COMMAND** 0x1900

  Command interpreter didn't understand command.

- #define **ECMD_EXPECT_FAILURE** 0x1901

  An expect was performed and a miscompare was found.

- #define **ECMD_SCANDEF_LOOKUP_FAILURE** 0x1902

  An Error occurred trying to process the scandef file.

- #define **ECMD_DATA_BOUNDS_OVERFLOW** 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** 0x2000

  Initialization of the DataBuffer failed.

- #define **ECMD_DBUF_BUFFER_OVERFLOW** 0x2010

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

- #define **ECMD_DBUF_XSTATE_ERROR** 0x2020

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

- #define **ECMD_DBUF_UNDEFINED_FUNCTION** 0x2030

  Function not included in this version of DataBuffer.

- #define **ECMD_DBUF_INVALID_ARGS** 0x2040

  Args provided to dataBuffer were invalid.

- #define **ECMD_DBUF_INVALID_DATA_FORMAT** 0x2041

  String data didn't match expected input format.

5.4.1 Detailed Description

All Return Codes for the eCmd Capi.

5.4.2 Define Documentation

5.4.2.1 #define ECMD_SUCCESS 0x0

API Returned Successfully.

5.4.2.2 #define ECMD_INVALID_DLL_VERSION 0x1000

Dll Version didn't match the Client version detected.

5.4.2.3 #define ECMD_INVALID_DLL_FILENAME 0x1001

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

5.4.2.4 #define ECMD_DLL_LOAD_FAILURE 0x1002

Error occured on call to dlopen.

5.4.2.5 #define ECMD_DLL_UNLOAD_FAILURE 0x1003

Error occurred on call to dlclose.

5.4.2.6 #define ECMD_DLL_UNINITIALIZED 0x1004

A function was called before ecmdLoadDll was called.

5.4.2.7 #define ECMD_DLL_INVALID 0x1005

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

5.4.2.8 #define ECMD_FAILURE 0x1010

General Failure occurred in eCMD.

5.4.2.9 #define ECMD_TARGET_NOT_CONFIGURED 0x1011

Chip target provided was not configured in the system.

5.4.2.10 #define ECMD_FUNCTION_NOT_SUPPORTED 0x1012

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

5.4.2.11 #define ECMD_UNKNOWN_FILE 0x1013

ecmdQueryFileLocation was unable to find the file you requested

5.4.2.12 #define ECMD_INVALID_ARGS 0x1020

Not enough arguments provided to the function.

5.4.2.13 #define ECMD_INVALID_SPY_ENUM 0x1021

getSpyEnum or putSpyEnum used an invalid enum

5.4.2.14 #define ECMD_SPY_FAILED_ECC_CHECK 0x1022

getSpy or getSpyEnum failed with invalid ECC detected in the hardware

5.4.2.15 #define ECMD_SPY_NOT_ENUMERATED 0x1023

getSpyEnum or putSpyEnum was called on a non-enumerated spy

5.4.2.16 #define ECMD_SPY_IS_EDIAL 0x1024

getSpy or Putspy was called on an edial

5.4.2.17 #define ECMD_INVALID_SPY 0x1025

Spy functions found an invalid Spy name or type.

5.4.2.18 #define ECMD_DATA_OVERFLOW 0x1026

Too much data was provided to a write function.

5.4.2.19 #define ECMD_DATA_UNDERFLOW 0x1027

Too little data was provided to a write function.

5.4.2.20 #define ECMD_INVALID_RING 0x1028

Invalid ring name was provided.

5.4.2.21 #define ECMD_INVALID_ARRAY 0x1029

Invalid array name was provided.

5.4.2.22 #define ECMD_INVALID_CONFIG 0x1030

There was an error processing the configuration information.

5.4.2.23 #define ECMD_CLOCKS_IN_INVALID_STATE 0x1031

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

5.4.2.24 #define ECMD_NON_JTAG_CHIP 0x1032

JTag function called on non-jtag attached chip.

5.4.2.25 #define ECMD_NON_FSI_CHIP 0x1033

Fsi function called on non-fsi attached chip.

5.4.2.26 #define ECMD_INVALID_SPR 0x1034

Invalid SPR was specified to get/put spr functions.

5.4.2.27 #define ECMD_INVALID_GPR 0x1035

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

5.4.2.28 #define ECMD_INVALID_FPR 0x1036

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

5.4.2.29 #define ECMD_RING_CACHE_ENABLED 0x1037

Ring Cache enabled during call non-cache enabled function.

5.4.2.30 #define ECMD_INVALID_CONFIG_NAME 0x1038

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

5.4.2.31 #define ECMD_SPY_GROUP_MISMATCH 0x1039

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

5.4.2.32 #define ECMD_INVALID_CLOCK_DOMAIN 0x1040

An invalid clock domain name was specified.

5.4.2.33 #define ECMD_CLOCKS_ALREADY_OFF 0x1041

A stopclocks was requested when clocks are already off.

5.4.2.34 #define ECMD_CLOCKS_ALREADY_ON 0x1042

A startclocks was requested when clocks are already on.

5.4.2.35 #define ECMD_UNABLE_TO_OPEN_SCANDEF 0x1043

eCMD was unable to open the scandef

5.4.2.36 #define ECMD_INVALID_LATCHNAME 0x1044

eCMD was unable to find the specified latch in the scandef

5.4.2.37 #define ECMD_POLLING_FAILURE 0x1045

eCMD failed waiting for a poll to match expected value

5.4.2.38 #define ECMD_INT_UNKNOWN_COMMAND 0x1900

Command interpreter didn't understand command.

5.4.2.39 #define ECMD_EXPECT_FAILURE 0x1901

An expect was performed and a miscompare was found.

5.4.2.40 #define ECMD_SCANDEF_LOOKUP_FAILURE 0x1902

An Error occurred trying to process the scandef file.

5.4.2.41 #define ECMD_DATA_BOUNDS_OVERFLOW 0x1903

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

5.4.2.42 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

5.4.2.43 #define ECMD_DBUF_INIT_FAIL 0x2000

Initialization of the DataBuffer failed.

5.4.2.44 #define ECMD_DBUF_BUFFER_OVERFLOW 0x2010

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

5.4.2.45 #define ECMD_DBUF_XSTATE_ERROR 0x2020

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

5.4.2.46 #define ECMD_DBUF_UNDEFINED_FUNCTION 0x2030

Function not included in this version of DataBuffer.

5.4.2.47 #define ECMD_DBUF_INVALID_ARGS 0x2040

Args provided to dataBuffer were invalid.

5.4.2.48 #define ECMD_DBUF_INVALID_DATA_FORMAT 0x2041

String data didn't match expected input format.

5.5 ecmdStructs.H File Reference

All the Structures required for the eCMD Capi.

`#include <inttypes.h>`

`#include <list>`

`#include <vector>`

`#include <string>`

`#include <ecmdDataBuffer.H>`

Compounds

- 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 **ecmdArrayEntry**

  Used by the getArrayMultiple function to pass data.

- struct **ecmdGroupData**

  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 **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 **ecmdLooperData**

  Used internally by ecmdConfigLooper to store looping state information.

Defines

- #define **ECMD_CAPI_VERSION** "1.2d"

  eCMD API Version

- #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 **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 **ecmdDllProduct_t** { **ECMD_DLL_PRODUCT_UNKNOWN**, **ECMD_DLL_PRODUCT_ECLIPZ** }

This is used by ecmdQueryDllInfo to return what product the dll supports.

- 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 **ecmdChipTargetState_t** { **ECMD_TARGET_UNKNOWN_STATE**, **ECMD_TARGET_FIELD_VALID**, **ECMD_TARGET_FIELD_UNUSED**, **ECMD_TARGET_QUERY_FIELD_VALID**, **ECMD_TARGET_QUERY_WILDCARD**, **ECMD_TARGET_QUERY_IGNORE**, **ECMD_TARGET_THREAD_ALIVE** }

  Used by **ecmdChipTarget** (p. 17) *to describe the value in the state fields - The ECMD_TARGET_FIELD_* states are used for functions to return applicable values - The ECMD_TARGET_QUERY_* states are used by the ecmdQueryConfig and ecmdQuerySelected functions to refine the query.*

- enum **ecmdChipInterfaceType_t** { **ECMD_INTERFACE_ACCESS**, **ECMD_INTERFACE_CFAM**, **ECMD_INTERFACE_UNKNOWN** }

  Used in **ecmdChipData** (p. 15) *to describe the interface macro used by the chip.*

- enum **ecmdQueryDetail_t** { **ECMD_QUERY_DETAIL_LOW**, **ECMD_QUERY_DETAIL_HIGH** }

  Used by ecmdQueryConfig to specify detail level of query.

- 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 **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

  See also:

  ecmdSpyData (p. 61).

- enum **ecmdFileType_t** { **ECMD_FILE_SCANDEF**, **ECMD_FILE_SPYDEF**, **ECMD_FILE_ARRAYDEF**, **ECMD_FILE_HELPTEXT**, **ECMD_FILE_SCOMDATA** }

  Used for the ecmdQueryFileLocation function to specify the file type you are looking for.

- enum **ecmdConfigLoopType_t** { **ECMD_SELECTED_TARGETS_LOOP**, **ECMD_ALL_TARGETS_LOOP** }

  Used by ecmdConfigLooperInit function to specify what type of data to loop on.

- enum **ecmdGlobalVarType_t** { **ECMD_GLOBALVAR_DEBUG**, **ECMD_GLOBALVAR_QUIETMODE** }

  Used by ecmdGetGlobalVar to specify what variable you are looking for.

- enum **ecmdTraceType_t** { **ECMD_TRACE_SCAN**, **ECMD_TRACE_PROCEDURE** }

  Used by ecmdSetTraceMode to specify which trace to control.

- 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.

5.5.1 Detailed Description

All the Structures required for the eCMD Capi.

5.5.2 Define Documentation

5.5.2.1 #define ECMD_CAPI_VERSION "1.2d"

eCMD API Version

5.5.2.2 #define ECMD_CHIPFLAG_BUSMASK 0xC0000000

Defines for the **ecmdChipData** (p. 15) chipFlags field

5.5.2.3 #define ECMD_CHIPFLAG_RSVDBUS1 0x00000000

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

5.5.2.4 #define ECMD_CHIPFLAG_JTAG 0x40000000

5.5.2.5 #define ECMD_CHIPFLAG_FSI 0x80000000

5.5.2.6 #define ECMD_CHIPFLAG_RSVDBUS2 0xC0000000

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

5.5.3 Enumeration Type Documentation

5.5.3.1 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.

5.5.3.2 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.

5.5.3.3 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.

5.5.3.4 enum ecmdChipTargetState_t

Used by **ecmdChipTarget** (p. 17) to describe the value in the state fields - The ECMD_TARGET_FIELD_* states are used for functions to return applicable values - The ECMD_TARGET_QUERY_* states are used by the ecmdQueryConfig and ecmdQuerySelected functions to refine the query.

Enumeration values:

ECMD_TARGET_UNKNOWN_STATE State field has not been initialized.

ECMD_TARGET_FIELD_VALID Associated State Field is valid for this function.

ECMD_TARGET_FIELD_UNUSED Associated State Field is unused for this function.

ECMD_TARGET_QUERY_FIELD_VALID Associated State Field is valid for the query.

ECMD_TARGET_QUERY_WILDCARD Associated State Field should be itterated on and all valid results returned.

ECMD_TARGET_QUERY_IGNORE Query should be limited to data above this field, ignoring data.

ECMD_TARGET_THREAD_ALIVE Used when calling thread dependent functions tell the function to check for the thread to be alive before running.

5.5.3.5 enum ecmdChipInterfaceType_t

Used in **ecmdChipData** (p. 15) to describe the interface macro used by the chip.

Enumeration values:

ECMD_INTERFACE_ACCESS Standard Jtag Access Macro.

ECMD_INTERFACE_CFAM CommonFirmwareAccessMacro - Fsi interface.

ECMD_INTERFACE_UNKNOWN Unknown Interface.

5.5.3.6 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.

5.5.3.7 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.

5.5.3.8 enum ecmdSpyType_t

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

See also:

ecmdSpyData (p. 61).

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.

5.5.3.9 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

5.5.3.10 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.

ECMD_ALL_TARGETS_LOOP Loop on all valid targets in the system.

5.5.3.11 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.

5.5.3.12 enum ecmdTraceType_t

Used by ecmdSetTraceMode to specify which trace to control.

Enumeration values:

ECMD_TRACE_SCAN Scan Trace.

ECMD_TRACE_PROCEDURE Procedure Trace.

5.5.3.13 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.

5.5.4 Function Documentation

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

Used to sort Cage entries in an **ecmdCageData** (p. 14) list.

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

Used to sort Node entries in an **ecmdNodeData** (p. 55) list.

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

Used to sort Slot entries in an **ecmdSlotData** (p. 60) list.

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

Used to sort Chip entries (based on Pos) in an **ecmdChipData** (p. 15) list.

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

Used to sort Core entries in an **ecmdCoreData** (p. 20) list.

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

Used to sort Thread entries in an **ecmdThreadData** (p. 63) list.

5.5.4.7 std::string ecmdGetSharedLibVersion ()

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

5.6 ecmdUtils.H File Reference

Useful functions for use throughout the ecmd C API.

`#include <inttypes.h>`

`#include <string>`

`#include <vector>`

`#include <ecmdClientCapi.H>`

Functions

- uint32_t **ecmdConfigLooperInit** (**ecmdChipTarget** &io_target, **ecmdConfigLoopType_t** i_looptype, **ecmdLooperData** &io_state)

  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.

- std::string **ecmdWriteDataFormatted** (**ecmdDataBuffer** &i_data, std::string &i_format, int 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.

- std::string **ecmdWriteTarget** (**ecmdChipTarget** &i_target)

  Returns a formatted string containing the data in the given **ecmdChipTarget** (p. 17).

- 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.

5.6.1 Detailed Description

Useful functions for use throughout the ecmd C API.

5.6.2 Function Documentation

5.6.2.1 uint32_t ecmdConfigLooperInit (ecmdChipTarget & *io_target*, ecmdConfigLoopType_t *i_looptype*, ecmdLooperData & *io_state*)

Initializes data structures and code to loop over configured and selected elements of the system.

Parameters:

io_target Initial **ecmdChipTarget** (p. 17) 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

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. 131)

5.6.2.2 uint32_t ecmdConfigLooperNext (ecmdChipTarget & *io_target*, ecmdLooperData & *io_state*)

Loops over configured and selected elements of the system, updating target to point to them.

Parameters:

io_target **ecmdChipTarget** (p. 17) 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. 131)

5.6.2.3 uint32_t ecmdReadDataFormatted (ecmdDataBuffer & *o_data*, const char * *i_dataStr*, std::string & *i_format*, int *i_expectedLength* = 0)

Reads data from data string into data buffer based on a format type.

Return values:

ECMD_SUCCESS if data is well-formatted, non-zero otherwise

Parameters:

o_data **ecmdDataBuffer** (p. 21) 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

5.6.2.4 std::string ecmdWriteDataFormatted (ecmdDataBuffer & *i_data*, std::string & *i_format*, int *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. 21) 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

address A base address value that can be used in formating certain data- i.e., data from memory

5.6.2.5 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

5.6.2.6 std::string ecmdWriteTarget (ecmdChipTarget & *i_target*)

Returns a formatted string containing the data in the given **ecmdChipTarget** (p. 17).

Returns:
String with formatted target data

Parameters:
i_target **ecmdChipTarget** (p. 17) containing data to format into string

5.6.2.7 uint32_t ecmdGetChipData (ecmdChipTarget & *i_target*, ecmdChipData & *o_data*)

Fetch the detailed chip data structure for the selected target.

Return values:
ECMD_SUCCESS if chip data for target is found, non-zero otherwise

Parameters:
i_target **ecmdChipTarget** (p. 17) that information is requested for

o_data **ecmdChipData** (p. 15) struct that contains detailed info on chip ec level, etc.

5.6.2.8 uint32_t ecmdDisplayDllInfo ()

Function calls ecmdQueryDllInfo and displays the output to stdout.

Return values:

ECMD_SUCCESS if successful

nonzero on failure

Index

- ~ecmdDataBuffer
  - ecmdDataBuffer, 26

- address
  - ecmdArrayEntry, 13
  - ecmdRingData, 58
- addressLength
  - ecmdArrayData, 12
- applyInversionMask
  - ecmdDataBuffer, 33
- arrayName
  - ecmdArrayData, 12

- bitLength
  - ecmdProcRegisterInfo, 56
  - ecmdRingData, 58
  - ecmdSpyData, 61
- buffer
  - ecmdArrayEntry, 13
  - ecmdIndexEntry, 48
  - ecmdLatchEntry, 49
  - ecmdNameEntry, 53
  - ecmdNameVectorEntry, 54

- cage
  - ecmdChipTarget, 18
- cageData
  - ecmdQueryData, 57
- cageId
  - ecmdCageData, 14
- cageState
  - ecmdChipTarget, 19
- chipCommonType
  - ecmdChipData, 15
- chipData
  - ecmdSlotData, 60
- chipEc
  - ecmdChipData, 16
- chipFlags
  - ecmdChipData, 16
- chipType
  - ecmdChipData, 15
  - ecmdChipTarget, 18
- chipTypeState
  - ecmdChipTarget, 19
- cipClearBreakpoint
  - cipClientCapi.H, 67
- cipClientCapi.H, 65
- cipClientCapi.H
  - cipClearBreakpoint, 67
  - cipSetBreakpoint, 67
  - cipStartInstructions, 66
  - cipStepInstructions, 66
  - cipStopInstructions, 66
- cipSetBreakpoint
  - cipClientCapi.H, 67
- cipStartInstructions
  - cipClientCapi.H, 66
- cipStepInstructions
  - cipClientCapi.H, 66
- cipStopInstructions
  - cipClientCapi.H, 66
- clearBit
  - ecmdDataBuffer, 29
- clockState
  - ecmdArrayData, 12
  - ecmdRingData, 59
  - ecmdSpyData, 62
- copy
  - ecmdDataBuffer, 38
- core
  - ecmdChipTarget, 18
- coreData
  - ecmdChipData, 16
- coreId
  - ecmdCoreData, 20
- coreState
  - ecmdChipTarget, 19

- deadbitsMask
  - ecmdGroupData, 47
- detailLevel
  - ecmdQueryData, 57
- dllBuildDate
  - ecmdDllInfo, 45
- dllBuildInfo
  - ecmdDllInfo, 46
- dllCapiVersion
  - ecmdDllInfo, 46
- dllEnv

ecmdDllInfo, 45
dllProduct
ecmdDllInfo, 45
dllType
ecmdDllInfo, 45

eccGroups
ecmdSpyData, 62
ECMD_ALL_TARGETS_LOOP
ecmdStructs.H, 128
ECMD_CAPI_VERSION
ecmdStructs.H, 125
ECMD_CHIPFLAG_BUSMASK
ecmdStructs.H, 125
ECMD_CHIPFLAG_FSI
ecmdStructs.H, 125
ECMD_CHIPFLAG_JTAG
ecmdStructs.H, 125
ECMD_CHIPFLAG_RSVDBUS1
ecmdStructs.H, 125
ECMD_CHIPFLAG_RSVDBUS2
ecmdStructs.H, 125
ECMD_CLOCKS_ALREADY_OFF
ecmdReturnCodes.H, 119
ECMD_CLOCKS_ALREADY_ON
ecmdReturnCodes.H, 119
ECMD_CLOCKS_IN_INVALID_STATE
ecmdReturnCodes.H, 118
ECMD_CLOCKSTATE_NA
ecmdStructs.H, 127
ECMD_CLOCKSTATE_OFF
ecmdStructs.H, 127
ECMD_CLOCKSTATE_ON
ecmdStructs.H, 127
ECMD_CLOCKSTATE_UNKNOWN
ecmdStructs.H, 127
ECMD_DATA_BOUNDS_OVERFLOW
ecmdReturnCodes.H, 120
ECMD_DATA_OVERFLOW
ecmdReturnCodes.H, 118
ECMD_DATA_UNDERFLOW
ecmdReturnCodes.H, 118
ECMD_DBUF_BUFFER_OVERFLOW
ecmdReturnCodes.H, 120
ECMD_DBUF_INIT_FAIL
ecmdReturnCodes.H, 120
ECMD_DBUF_INVALID_ARGS
ecmdReturnCodes.H, 120
ECMD_DBUF_INVALID_DATA_FORMAT
ecmdReturnCodes.H, 121
ECMD_DBUF_SUCCESS
ecmdReturnCodes.H, 120
ECMD_DBUF_UNDEFINED_FUNCTION
ecmdReturnCodes.H, 120
ECMD_DBUF_XSTATE_ERROR
ecmdReturnCodes.H, 120
ECMD_DLL_CRONUS
ecmdStructs.H, 125
ECMD_DLL_ENV_HW
ecmdStructs.H, 126
ECMD_DLL_ENV_SIM
ecmdStructs.H, 126
ECMD_DLL_INVALID
ecmdReturnCodes.H, 117
ECMD_DLL_IPSERIES
ecmdStructs.H, 125
ECMD_DLL_LOAD_FAILURE
ecmdReturnCodes.H, 117
ECMD_DLL_PRODUCT_ECLIPZ
ecmdStructs.H, 126
ECMD_DLL_PRODUCT_UNKNOWN
ecmdStructs.H, 126
ECMD_DLL_SCAND
ecmdStructs.H, 125
ECMD_DLL_STUB
ecmdStructs.H, 125
ECMD_DLL_UNINITIALIZED
ecmdReturnCodes.H, 117
ECMD_DLL_UNKNOWN
ecmdStructs.H, 125
ECMD_DLL_UNLOAD_FAILURE
ecmdReturnCodes.H, 117
ECMD_DLL_ZSERIES
ecmdStructs.H, 125
ECMD_EXPECT_FAILURE
ecmdReturnCodes.H, 120
ECMD_FAILURE
ecmdReturnCodes.H, 117
ECMD_FILE_ARRAYDEF
ecmdStructs.H, 127
ECMD_FILE_HELPTEXT
ecmdStructs.H, 127
ECMD_FILE_SCANDEF
ecmdStructs.H, 127
ECMD_FILE_SCOMDATA
ecmdStructs.H, 127
ECMD_FILE_SPYDEF
ecmdStructs.H, 127
ECMD_FUNCTION_NOT_SUPPORTED
ecmdReturnCodes.H, 117
ECMD_GLOBALVAR_DEBUG
ecmdStructs.H, 128
ECMD_GLOBALVAR_QUIETMODE
ecmdStructs.H, 128
ECMD_INT_UNKNOWN_COMMAND
ecmdReturnCodes.H, 120
ECMD_INTERFACE_ACCESS
ecmdStructs.H, 126

ECMD_INTERFACE_CFAM
  ecmdStructs.H, 126
ECMD_INTERFACE_UNKNOWN
  ecmdStructs.H, 126
ECMD_INVALID_ARGS
  ecmdReturnCodes.H, 118
ECMD_INVALID_ARRAY
  ecmdReturnCodes.H, 118
ECMD_INVALID_CLOCK_DOMAIN
  ecmdReturnCodes.H, 119
ECMD_INVALID_CONFIG
  ecmdReturnCodes.H, 118
ECMD_INVALID_CONFIG_NAME
  ecmdReturnCodes.H, 119
ECMD_INVALID_DLL_FILENAME
  ecmdReturnCodes.H, 117
ECMD_INVALID_DLL_VERSION
  ecmdReturnCodes.H, 117
ECMD_INVALID_FPR
  ecmdReturnCodes.H, 119
ECMD_INVALID_GPR
  ecmdReturnCodes.H, 119
ECMD_INVALID_LATCHNAME
  ecmdReturnCodes.H, 120
ECMD_INVALID_RING
  ecmdReturnCodes.H, 118
ECMD_INVALID_SPR
  ecmdReturnCodes.H, 119
ECMD_INVALID_SPY
  ecmdReturnCodes.H, 118
ECMD_INVALID_SPY_ENUM
  ecmdReturnCodes.H, 118
ECMD_LATCHMODE_FULL
  ecmdStructs.H, 128
ECMD_LATCHMODE_PARTIAL
  ecmdStructs.H, 128
ECMD_NON_FSI_CHIP
  ecmdReturnCodes.H, 119
ECMD_NON_JTAG_CHIP
  ecmdReturnCodes.H, 119
ECMD_POLLING_FAILURE
  ecmdReturnCodes.H, 120
ECMD_QUERY_DETAIL_HIGH
  ecmdStructs.H, 127
ECMD_QUERY_DETAIL_LOW
  ecmdStructs.H, 127
ECMD_RING_CACHE_ENABLED
  ecmdReturnCodes.H, 119
ECMD_SCANDEF_LOOKUP_FAILURE
  ecmdReturnCodes.H, 120
ECMD_SELECTED_TARGETS_LOOP
  ecmdStructs.H, 128
ECMD_SPY_FAILED_ECC_CHECK
  ecmdReturnCodes.H, 118
ECMD_SPY_GROUP_MISMATCH
  ecmdReturnCodes.H, 119
ECMD_SPY_IS_EDIAL
  ecmdReturnCodes.H, 118
ECMD_SPY_NOT_ENUMERATED
  ecmdReturnCodes.H, 118
ECMD_SPYTYPE_ALIAS
  ecmdStructs.H, 127
ECMD_SPYTYPE_ECCGROUP
  ecmdStructs.H, 127
ECMD_SPYTYPE_EDIAL
  ecmdStructs.H, 127
ECMD_SPYTYPE_IDIAL
  ecmdStructs.H, 127
ECMD_SUCCESS
  ecmdReturnCodes.H, 117
ECMD_TARGET_FIELD_UNUSED
  ecmdStructs.H, 126
ECMD_TARGET_FIELD_VALID
  ecmdStructs.H, 126
ECMD_TARGET_NOT_CONFIGURED
  ecmdReturnCodes.H, 117
ECMD_TARGET_QUERY_FIELD_VALID
  ecmdStructs.H, 126
ECMD_TARGET_QUERY_IGNORE
  ecmdStructs.H, 126
ECMD_TARGET_QUERY_WILDCARD
  ecmdStructs.H, 126
ECMD_TARGET_THREAD_ALIVE
  ecmdStructs.H, 126
ECMD_TARGET_UNKNOWN_STATE
  ecmdStructs.H, 126
ECMD_TRACE_PROCEDURE
  ecmdStructs.H, 128
ECMD_TRACE_SCAN
  ecmdStructs.H, 128
ECMD_UNABLE_TO_OPEN_SCANDEF
  ecmdReturnCodes.H, 119
ECMD_UNKNOWN_FILE
  ecmdReturnCodes.H, 117
ecmdArrayData, 11
ecmdArrayData
  addressLength, 12
  arrayName, 12
  clockState, 12
  length, 12
  width, 12
ecmdArrayEntry, 13
ecmdArrayEntry
  address, 13
  buffer, 13
  rc, 13
ecmdBitsHeader
  ecmdUtils.H, 132

ecmdCageData, 14
ecmdCageData
    cageId, 14
    nodeData, 14
ecmdChipData, 15
ecmdChipData
    chipCommonType, 15
    chipEc, 16
    chipFlags, 16
    chipType, 15
    coreData, 16
    interfaceType, 16
    numProcCores, 16
    pos, 16
    simModelEc, 16
ecmdChipInterfaceType_t
    ecmdStructs.H, 126
ecmdChipTarget, 17
ecmdChipTarget
    cage, 18
    cageState, 19
    chipType, 18
    chipTypeState, 19
    core, 18
    coreState, 19
    node, 18
    nodeState, 19
    pos, 18
    posState, 19
    slot, 18
    slotState, 19
    thread, 18
    threadState, 19
    unitId, 19
    unitIdState, 19
ecmdChipTargetState_t
    ecmdStructs.H, 126
ecmdClientCapi.H, 68
ecmdClientCapi.H
    ecmdCommandArgs, 77
    ecmdConfigureTarget, 112
    ecmdDeconfigureTarget, 112
    ecmdDisableRingCache, 88
    ecmdEnableRingCache, 87
    ecmdFlushRingCache, 88
    ecmdGetConfiguration, 111
    ecmdGetErrorMsg, 110
    ecmdGetGlobalVar, 111
    ecmdIsRingCacheEnabled, 88
    ecmdLoadDll, 76
    ecmdOutput, 111
    ecmdOutputError, 110
    ecmdOutputWarning, 110
    ecmdQueryArray, 79
    ecmdQueryClockState, 91
    ecmdQueryConfig, 77
    ecmdQueryDllInfo, 77
    ecmdQueryFileLocation, 80
    ecmdQueryProcRegisterInfo, 92
    ecmdQueryRing, 79
    ecmdQuerySelected, 78
    ecmdQuerySpy, 79
    ecmdQueryTargetConfigured, 80
    ecmdQueryTraceMode, 111
    ecmdRegisterErrorMsg, 110
    ecmdSetConfiguration, 112
    ecmdSetTraceMode, 111
    ecmdUnloadDll, 76
    getArray, 88
    getArrayMultiple, 89
    getCfamRegister, 84
    getFpr, 97
    getFprMultiple, 97
    getGpr, 95
    getGprMultiple, 95
    getLatch, 81
    getMemDma, 100
    getMemMemCtrl, 101
    getMemProc, 99
    getRing, 80
    getScom, 82
    getSpr, 93
    getSprMultiple, 93
    getSpy, 85
    getSpyEccGrouping, 86
    getSpyEnum, 85
    getTraceArray, 98
    getTraceArrayMultiple, 99
    iSteps, 92
    putArray, 89
    putArrayMultiple, 90
    putCfamRegister, 84
    putFpr, 97
    putFprMultiple, 98
    putGpr, 96
    putGprMultiple, 96
    putLatch, 82
    putMemDma, 100
    putMemMemCtrl, 101
    putMemProc, 100
    putRing, 81
    putScom, 83
    putSpr, 94
    putSprMultiple, 94
    putSpy, 86
    putSpyEnum, 87
    sendCmd, 83
    simaet, 102

simcheckpoint, 102
simclock, 102
simecho, 103
simexit, 103
simEXPECTFAC, 103
simexpecttcfac, 104
simgetcurrentcycle, 104
simGETFAC, 104
simGETFACX, 105
simgettcfac, 105
siminit, 106
simPOLLFAC, 106
simPUTFAC, 106
simPUTFACX, 107
simputtcfac, 107
simrestart, 108
simSTKFAC, 108
simstktcfac, 108
simSUBCMD, 109
simtckinterval, 109
simUNSTICK, 109
simunsticktcfac, 110
startClocks, 91
stopClocks, 92

ecmdClockState_t
ecmdStructs.H, 127

ecmdCommandArgs
ecmdClientCapi.H, 77

ecmdConfigLooperInit
ecmdUtils.H, 131

ecmdConfigLooperNext
ecmdUtils.H, 131

ecmdConfigLoopType_t
ecmdStructs.H, 127

ecmdConfigureTarget
ecmdClientCapi.H, 112

ecmdCoreData, 20

ecmdCoreData
coreId, 20
numProcThreads, 20
threadData, 20

ecmdCurCage
ecmdLooperData, 51

ecmdCurChip
ecmdLooperData, 52

ecmdCurCore
ecmdLooperData, 52

ecmdCurNode
ecmdLooperData, 51

ecmdCurSlot
ecmdLooperData, 52

ecmdCurThread
ecmdLooperData, 52

ecmdDataBuffer, 21
ecmdDataBuffer, 26

ecmdDataBuffer
~ecmdDataBuffer, 26
applyInversionMask, 33
clearBit, 29
copy, 38
ecmdDataBuffer, 26
evenParity, 39
extract, 35
flipBit, 30
flushTo0, 33
flushTo1, 33
genBinStr, 40
genHexLeftStr, 39, 40
genHexRightStr, 40
genXstateStr, 41
getBitLength, 27
getByte, 29
getByteLength, 27
getCapacity, 27
getNumBitsSet, 31
getWord, 29
getWordLength, 27
getXstate, 42
hasXstate, 42
insert, 33, 34
insertFromBin, 42
insertFromHexLeft, 41
insertFromHexRight, 41
insertFromRight, 34
invert, 33
isBitClear, 30, 31
isBitSet, 30
memCopyIn, 38
memCopyInXstate, 43
memCopyOut, 38
memCopyOutXstate, 43
merge, 36
oddParity, 38, 39
operator &, 44
operator!=, 44
operator=, 38
operator==, 43
operator|, 44
rotateLeft, 32
rotateRight, 32
setAnd, 37
setBit, 28
setBitLength, 27
setByte, 29
setCapacity, 28
setOr, 35, 36
setWord, 28
setWordLength, 27

setXor, 36, 37
setXstate, 43
shiftLeft, 31
shiftLeftAndResize, 32
shiftRight, 31
shiftRightAndResize, 32
writeBit, 28
ecmdDataBuffer.H, 113
ecmdDeconfigureTarget
ecmdClientCapi.H, 112
ecmdDisableRingCache
ecmdClientCapi.H, 88
ecmdDisplayDllInfo
ecmdUtils.H, 132
ecmdDllEnv_t
ecmdStructs.H, 126
ecmdDllInfo, 45
ecmdDllInfo
dllBuildDate, 45
dllBuildInfo, 46
dllCapiVersion, 46
dllEnv, 45
dllProduct, 45
dllType, 45
ecmdDllProduct_t
ecmdStructs.H, 125
ecmdDllType_t
ecmdStructs.H, 125
ecmdEnableRingCache
ecmdClientCapi.H, 87
ecmdFileType_t
ecmdStructs.H, 127
ecmdFlushRingCache
ecmdClientCapi.H, 88
ecmdGetChipData
ecmdUtils.H, 132
ecmdGetConfiguration
ecmdClientCapi.H, 111
ecmdGetErrorMsg
ecmdClientCapi.H, 110
ecmdGetGlobalVar
ecmdClientCapi.H, 111
ecmdGetSharedLibVersion
ecmdStructs.H, 129
ecmdGlobalVarType_t
ecmdStructs.H, 128
ecmdGroupData, 47
ecmdGroupData
deadbitsMask, 47
extractBuffer, 47
ecmdIndexEntry, 48
ecmdIndexEntry
buffer, 48
index, 48
rc, 48
ecmdIsRingCacheEnabled
ecmdClientCapi.H, 88
ecmdLatchEntry, 49
ecmdLatchEntry
buffer, 49
latchEndBit, 49
latchName, 49
latchStartBit, 49
rc, 50
ringName, 49
ecmdLatchMode_t
ecmdStructs.H, 128
ecmdLoadDll
ecmdClientCapi.H, 76
ecmdLooperData, 51
ecmdLooperData
ecmdCurCage, 51
ecmdCurChip, 52
ecmdCurCore, 52
ecmdCurNode, 51
ecmdCurSlot, 52
ecmdCurThread, 52
ecmdLooperInitFlag, 52
ecmdSystemConfigData, 51
prevTarget, 52
ecmdLooperInitFlag
ecmdLooperData, 52
ecmdNameEntry, 53
ecmdNameEntry
buffer, 53
name, 53
rc, 53
ecmdNameVectorEntry, 54
ecmdNameVectorEntry
buffer, 54
name, 54
rc, 54
ecmdNodeData, 55
ecmdNodeData
nodeId, 55
slotData, 55
ecmdOutput
ecmdClientCapi.H, 111
ecmdOutputError
ecmdClientCapi.H, 110
ecmdOutputWarning
ecmdClientCapi.H, 110
ecmdProcRegisterInfo, 56
ecmdProcRegisterInfo
bitLength, 56
totalEntries, 56
ecmdQueryArray
ecmdClientCapi.H, 79

- ecmdQueryClockState
  - ecmdClientCapi.H, 91
- ecmdQueryConfig
  - ecmdClientCapi.H, 77
- ecmdQueryData, 57
- ecmdQueryData
  - cageData, 57
  - detailLevel, 57
- ecmdQueryDetail_t
  - ecmdStructs.H, 126
- ecmdQueryDllInfo
  - ecmdClientCapi.H, 77
- ecmdQueryFileLocation
  - ecmdClientCapi.H, 80
- ecmdQueryProcRegisterInfo
  - ecmdClientCapi.H, 92
- ecmdQueryRing
  - ecmdClientCapi.H, 79
- ecmdQuerySelected
  - ecmdClientCapi.H, 78
- ecmdQuerySpy
  - ecmdClientCapi.H, 79
- ecmdQueryTargetConfigured
  - ecmdClientCapi.H, 80
- ecmdQueryTraceMode
  - ecmdClientCapi.H, 111
- ecmdReadDataFormatted
  - ecmdUtils.H, 131
- ecmdRegisterErrorMsg
  - ecmdClientCapi.H, 110
- ecmdReturnCodes.H, 114
- ecmdReturnCodes.H
  - ECMD_CLOCKS_ALREADY_OFF, 119
  - ECMD_CLOCKS_ALREADY_ON, 119
  - ECMD_CLOCKS_IN_INVALID_STATE, 118
  - ECMD_DATA_BOUNDS_OVERFLOW, 120
  - ECMD_DATA_OVERFLOW, 118
  - ECMD_DATA_UNDERFLOW, 118
  - ECMD_DBUF_BUFFER_OVERFLOW, 120
  - ECMD_DBUF_INIT_FAIL, 120
  - ECMD_DBUF_INVALID_ARGS, 120
  - ECMD_DBUF_INVALID_DATA_FORMAT, 121
  - ECMD_DBUF_SUCCESS, 120
  - ECMD_DBUF_UNDEFINED_FUNCTION, 120
  - ECMD_DBUF_XSTATE_ERROR, 120
  - ECMD_DLL_INVALID, 117
  - ECMD_DLL_LOAD_FAILURE, 117
  - ECMD_DLL_UNINITIALIZED, 117
  - ECMD_DLL_UNLOAD_FAILURE, 117
  - ECMD_EXPECT_FAILURE, 120
  - ECMD_FAILURE, 117
  - ECMD_FUNCTION_NOT_SUPPORTED, 117
  - ECMD_INT_UNKNOWN_COMMAND, 120
  - ECMD_INVALID_ARGS, 118
  - ECMD_INVALID_ARRAY, 118
  - ECMD_INVALID_CLOCK_DOMAIN, 119
  - ECMD_INVALID_CONFIG, 118
  - ECMD_INVALID_CONFIG_NAME, 119
  - ECMD_INVALID_DLL_FILENAME, 117
  - ECMD_INVALID_DLL_VERSION, 117
  - ECMD_INVALID_FPR, 119
  - ECMD_INVALID_GPR, 119
  - ECMD_INVALID_LATCHNAME, 120
  - ECMD_INVALID_RING, 118
  - ECMD_INVALID_SPR, 119
  - ECMD_INVALID_SPY, 118
  - ECMD_INVALID_SPY_ENUM, 118
  - ECMD_NON_FSI_CHIP, 119
  - ECMD_NON_JTAG_CHIP, 119
  - ECMD_POLLING_FAILURE, 120
  - ECMD_RING_CACHE_ENABLED, 119
  - ECMD_SCANDEF_LOOKUP_FAILURE, 120
  - ECMD_SPY_FAILED_ECC_CHECK, 118
  - ECMD_SPY_GROUP_MISMATCH, 119
  - ECMD_SPY_IS_EDIAL, 118
  - ECMD_SPY_NOT_ENUMERATED, 118
  - ECMD_SUCCESS, 117
  - ECMD_TARGET_NOT_CONFIGURED, 117
  - ECMD_UNABLE_TO_OPEN_SCANDEF, 119
  - ECMD_UNKNOWN_FILE, 117
- ecmdRingData, 58
- ecmdRingData
  - address, 58
  - bitLength, 58
  - clockState, 59
  - hasInversionMask, 58
  - isCheckable, 59
  - ringNames, 58
  - supportsBroadsideLoad, 59
- ecmdSetConfiguration
  - ecmdClientCapi.H, 112

- ecmdSetTraceMode
  - ecmdClientCapi.H, 111
- ecmdSlotData, 60
- ecmdSlotData
  - chipData, 60
  - slotId, 60
- ecmdSpyData, 61
- ecmdSpyData
  - bitLength, 61
  - clockState, 62
  - eccGroups, 62
  - enums, 62
  - isEccChecked, 62
  - isEnumerated, 62
  - spyName, 61
  - spyType, 61
- ecmdSpyType_t
  - ecmdStructs.H, 127
- ecmdStructs.H, 122
  - ECMD_ALL_TARGETS_LOOP, 128
  - ECMD_CLOCKSTATE_NA, 127
  - ECMD_CLOCKSTATE_OFF, 127
  - ECMD_CLOCKSTATE_ON, 127
  - ECMD_CLOCKSTATE_UNKNOWN, 127
  - ECMD_DLL_CRONUS, 125
  - ECMD_DLL_ENV_HW, 126
  - ECMD_DLL_ENV_SIM, 126
  - ECMD_DLL_IPSERIES, 125
  - ECMD_DLL_PRODUCT_ECLIPZ, 126
  - ECMD_DLL_PRODUCT_UNKNOWN, 126
  - ECMD_DLL_SCAND, 125
  - ECMD_DLL_STUB, 125
  - ECMD_DLL_UNKNOWN, 125
  - ECMD_DLL_ZSERIES, 125
  - ECMD_FILE_ARRAYDEF, 127
  - ECMD_FILE_HELPTEXT, 127
  - ECMD_FILE_SCANDEF, 127
  - ECMD_FILE_SCOMDATA, 127
  - ECMD_FILE_SPYDEF, 127
  - ECMD_GLOBALVAR_DEBUG, 128
  - ECMD_GLOBALVAR_QUIETMODE, 128
  - ECMD_INTERFACE_ACCESS, 126
  - ECMD_INTERFACE_CFAM, 126
  - ECMD_INTERFACE_UNKNOWN, 126
  - ECMD_LATCHMODE_FULL, 128
  - ECMD_LATCHMODE_PARTIAL, 128
  - ECMD_QUERY_DETAIL_HIGH, 127
  - ECMD_QUERY_DETAIL_LOW, 127
  - ECMD_SELECTED_TARGETS_LOOP, 128
  - ECMD_SPYTYPE_ALIAS, 127
  - ECMD_SPYTYPE_ECCGROUP, 127
  - ECMD_SPYTYPE_EDIAL, 127
  - ECMD_SPYTYPE_IDIAL, 127
  - ECMD_TARGET_FIELD_UNUSED, 126
  - ECMD_TARGET_FIELD_VALID, 126
  - ECMD_TARGET_QUERY_FIELD_VALID, 126
  - ECMD_TARGET_QUERY_IGNORE, 126
  - ECMD_TARGET_QUERY_WILDCARD, 126
  - ECMD_TARGET_THREAD_ALIVE, 126
  - ECMD_TARGET_UNKNOWN_STATE, 126
  - ECMD_TRACE_PROCEDURE, 128
  - ECMD_TRACE_SCAN, 128
- ecmdStructs.H
  - ECMD_CAPI_VERSION, 125
  - ECMD_CHIPFLAG_BUSMASK, 125
  - ECMD_CHIPFLAG_FSI, 125
  - ECMD_CHIPFLAG_JTAG, 125
  - ECMD_CHIPFLAG_RSVDBUS1, 125
  - ECMD_CHIPFLAG_RSVDBUS2, 125
  - ecmdChipInterfaceType_t, 126
  - ecmdChipTargetState_t, 126
  - ecmdClockState_t, 127
  - ecmdConfigLoopType_t, 127
  - ecmdDllEnv_t, 126
  - ecmdDllProduct_t, 125
  - ecmdDllType_t, 125
  - ecmdFileType_t, 127
  - ecmdGetSharedLibVersion, 129
  - ecmdGlobalVarType_t, 128
  - ecmdLatchMode_t, 128
  - ecmdQueryDetail_t, 126
  - ecmdSpyType_t, 127
  - ecmdTraceType_t, 128
  - operator<, 128, 129
- ecmdSystemConfigData
  - ecmdLooperData, 51
- ecmdThreadData, 63
- ecmdThreadData
  - threadId, 63
- ecmdTraceType_t
  - ecmdStructs.H, 128
- ecmdUnloadDll
  - ecmdClientCapi.H, 76
- ecmdUtils.H, 130
- ecmdUtils.H
  - ecmdBitsHeader, 132
  - ecmdConfigLooperInit, 131
  - ecmdConfigLooperNext, 131

ecmdDisplayDllInfo, 132
ecmdGetChipData, 132
ecmdReadDataFormatted, 131
ecmdWriteDataFormatted, 131
ecmdWriteTarget, 132
ecmdWriteDataFormatted
ecmdUtils.H, 131
ecmdWriteTarget
ecmdUtils.H, 132
enums
ecmdSpyData, 62
evenParity
ecmdDataBuffer, 39
extract
ecmdDataBuffer, 35
extractBuffer
ecmdGroupData, 47

flipBit
ecmdDataBuffer, 30
flushTo0
ecmdDataBuffer, 33
flushTo1
ecmdDataBuffer, 33

genBinStr
ecmdDataBuffer, 40
genHexLeftStr
ecmdDataBuffer, 39, 40
genHexRightStr
ecmdDataBuffer, 40
genXstateStr
ecmdDataBuffer, 41
getArray
ecmdClientCapi.H, 88
getArrayMultiple
ecmdClientCapi.H, 89
getBitLength
ecmdDataBuffer, 27
getByte
ecmdDataBuffer, 29
getByteLength
ecmdDataBuffer, 27
getCapacity
ecmdDataBuffer, 27
getCfamRegister
ecmdClientCapi.H, 84
getFpr
ecmdClientCapi.H, 97
getFprMultiple
ecmdClientCapi.H, 97
getGpr
ecmdClientCapi.H, 95
getGprMultiple
ecmdClientCapi.H, 95
getLatch
ecmdClientCapi.H, 81
getMemDma
ecmdClientCapi.H, 100
getMemMemCtrl
ecmdClientCapi.H, 101
getMemProc
ecmdClientCapi.H, 99
getNumBitsSet
ecmdDataBuffer, 31
getRing
ecmdClientCapi.H, 80
getScom
ecmdClientCapi.H, 82
getSpr
ecmdClientCapi.H, 93
getSprMultiple
ecmdClientCapi.H, 93
getSpy
ecmdClientCapi.H, 85
getSpyEccGrouping
ecmdClientCapi.H, 86
getSpyEnum
ecmdClientCapi.H, 85
getTraceArray
ecmdClientCapi.H, 98
getTraceArrayMultiple
ecmdClientCapi.H, 99
getWord
ecmdDataBuffer, 29
getWordLength
ecmdDataBuffer, 27
getXstate
ecmdDataBuffer, 42

hasInversionMask
ecmdRingData, 58
hasXstate
ecmdDataBuffer, 42

index
ecmdIndexEntry, 48
insert
ecmdDataBuffer, 33, 34
insertFromBin
ecmdDataBuffer, 42
insertFromHexLeft
ecmdDataBuffer, 41
insertFromHexRight
ecmdDataBuffer, 41
insertFromRight
ecmdDataBuffer, 34
interfaceType

ecmdChipData, 16
invert
ecmdDataBuffer, 33
isBitClear
ecmdDataBuffer, 30, 31
isBitSet
ecmdDataBuffer, 30
isCheckable
ecmdRingData, 59
isEccChecked
ecmdSpyData, 62
isEnumerated
ecmdSpyData, 62
iSteps
ecmdClientCapi.H, 92

latchEndBit
ecmdLatchEntry, 49
latchName
ecmdLatchEntry, 49
latchStartBit
ecmdLatchEntry, 49
length
ecmdArrayData, 12

memCopyIn
ecmdDataBuffer, 38
memCopyInXstate
ecmdDataBuffer, 43
memCopyOut
ecmdDataBuffer, 38
memCopyOutXstate
ecmdDataBuffer, 43
merge
ecmdDataBuffer, 36

name
ecmdNameEntry, 53
ecmdNameVectorEntry, 54
node
ecmdChipTarget, 18
nodeData
ecmdCageData, 14
nodeId
ecmdNodeData, 55
nodeState
ecmdChipTarget, 19
numProcCores
ecmdChipData, 16
numProcThreads
ecmdCoreData, 20

oddParity
ecmdDataBuffer, 38, 39
operator &
ecmdDataBuffer, 44
operator!=
ecmdDataBuffer, 44
operator<
ecmdStructs.H, 128, 129
operator=
ecmdDataBuffer, 38
operator==
ecmdDataBuffer, 43
operator|
ecmdDataBuffer, 44

pos
ecmdChipData, 16
ecmdChipTarget, 18
posState
ecmdChipTarget, 19
prevTarget
ecmdLooperData, 52
putArray
ecmdClientCapi.H, 89
putArrayMultiple
ecmdClientCapi.H, 90
putCfamRegister
ecmdClientCapi.H, 84
putFpr
ecmdClientCapi.H, 97
putFprMultiple
ecmdClientCapi.H, 98
putGpr
ecmdClientCapi.H, 96
putGprMultiple
ecmdClientCapi.H, 96
putLatch
ecmdClientCapi.H, 82
putMemDma
ecmdClientCapi.H, 100
putMemMemCtrl
ecmdClientCapi.H, 101
putMemProc
ecmdClientCapi.H, 100
putRing
ecmdClientCapi.H, 81
putScom
ecmdClientCapi.H, 83
putSpr
ecmdClientCapi.H, 94
putSprMultiple
ecmdClientCapi.H, 94
putSpy
ecmdClientCapi.H, 86
putSpyEnum
ecmdClientCapi.H, 87

rc
ecmdArrayEntry, 13
ecmdIndexEntry, 48
ecmdLatchEntry, 50
ecmdNameEntry, 53
ecmdNameVectorEntry, 54
ringName
ecmdLatchEntry, 49
ringNames
ecmdRingData, 58
rotateLeft
ecmdDataBuffer, 32
rotateRight
ecmdDataBuffer, 32

sendCmd
ecmdClientCapi.H, 83
setAnd
ecmdDataBuffer, 37
setBit
ecmdDataBuffer, 28
setBitLength
ecmdDataBuffer, 27
setByte
ecmdDataBuffer, 29
setCapacity
ecmdDataBuffer, 28
setOr
ecmdDataBuffer, 35, 36
setWord
ecmdDataBuffer, 28
setWordLength
ecmdDataBuffer, 27
setXor
ecmdDataBuffer, 36, 37
setXstate
ecmdDataBuffer, 43
shiftLeft
ecmdDataBuffer, 31
shiftLeftAndResize
ecmdDataBuffer, 32
shiftRight
ecmdDataBuffer, 31
shiftRightAndResize
ecmdDataBuffer, 32
simaet
ecmdClientCapi.H, 102
simcheckpoint
ecmdClientCapi.H, 102
simclock
ecmdClientCapi.H, 102
simecho
ecmdClientCapi.H, 103
simexit
ecmdClientCapi.H, 103
simEXPECTFAC
ecmdClientCapi.H, 103
simexpecttcfac
ecmdClientCapi.H, 104
simgetcurrentcycle
ecmdClientCapi.H, 104
simGETFAC
ecmdClientCapi.H, 104
simGETFACX
ecmdClientCapi.H, 105
simgettcfac
ecmdClientCapi.H, 105
siminit
ecmdClientCapi.H, 106
simModelEc
ecmdChipData, 16
simPOLLFAC
ecmdClientCapi.H, 106
simPUTFAC
ecmdClientCapi.H, 106
simPUTFACX
ecmdClientCapi.H, 107
simputtcfac
ecmdClientCapi.H, 107
simrestart
ecmdClientCapi.H, 108
simSTKFAC
ecmdClientCapi.H, 108
simstktcfac
ecmdClientCapi.H, 108
simSUBCMD
ecmdClientCapi.H, 109
simtckinterval
ecmdClientCapi.H, 109
simUNSTICK
ecmdClientCapi.H, 109
simunsticktcfac
ecmdClientCapi.H, 110
slot
ecmdChipTarget, 18
slotData
ecmdNodeData, 55
slotId
ecmdSlotData, 60
slotState
ecmdChipTarget, 19
spyName
ecmdSpyData, 61
spyType
ecmdSpyData, 61
startClocks
ecmdClientCapi.H, 91
stopClocks

ecmdClientCapi.H, 92
supportsBroadsideLoad
ecmdRingData, 59

thread
ecmdChipTarget, 18
threadData
ecmdCoreData, 20
threadId
ecmdThreadData, 63
threadState
ecmdChipTarget, 19
totalEntries
ecmdProcRegisterInfo, 56

unitId
ecmdChipTarget, 19
unitIdState
ecmdChipTarget, 19

width
ecmdArrayData, 12
writeBit
ecmdDataBuffer, 28